

Giuliano Donzellini
Luca Oneto
Domenico Ponta
Davide Anguita



Introduzione al Progetto di Sistemi Digitali

Introduzione al Progetto di Sistemi Digitali

Giuliano Donzellini · Luca Oneto
Domenico Ponta · Davide Anguita

Introduzione al Progetto di Sistemi Digitali

Giuliano Donzellini
Dipartimento di Ingegneria Navale, Elettrica,
Elettronica e delle Telecomunicazioni
(DITEN)

Università degli Studi di Genova
Genoa
Italy

Luca Oneto
Dipartimento di Informatica, Bioingegneria,
Robotica e Ingegneria dei Sistemi
(DIBRIS)

Università degli Studi di Genova
Genoa
Italy

Domenico Ponta
Università degli Studi di Genova
Genoa
Italy

Davide Anguita
Dipartimento di Informatica, Bioingegneria,
Robotica e Ingegneria dei Sistemi
(DIBRIS)

Università degli Studi di Genova
Genoa
Italy

ISBN 978-88-470-3962-9 ISBN 978-88-470-3963-6 (eBook)
<https://doi.org/10.1007/978-88-470-3963-6>

Library of Congress Control Number: 2017956313

© Springer-Verlag Italia S.r.l. 2018

Quest'opera è protetta dalla legge sul diritto d'autore e la sua riproduzione è ammessa solo ed esclusivamente nei limiti stabiliti dalla stessa. Le fotocopie per uso personale possono essere effettuate nei limiti del 15 % di ciascun volume dietro pagamento alla SIAE del compenso previsto dall'art. 68. Le riproduzioni per uso non personale e/o oltre il limite del 15 % potranno avvenire solo a seguito di specifica autorizzazione rilasciata da AIDRO, Corso di Porta Romana n.108, Milano 20122, e-mail segreteria@aidro.org e sito web www.aidro.org. Tutti i diritti, in particolare quelli relativi alla traduzione, alla ristampa, all'utilizzo di illustrazioni e tabelle, alla citazione orale, alla trasmissione radiofonica o televisiva, alla registrazione su microfilm o in database, o alla riproduzione in qualsiasi altra forma (stampata o elettronica) rimangono riservati anche nel caso di utilizzo parziale. La violazione delle norme comporta le sanzioni previste dalla legge. L'utilizzo in questa pubblicazione di denominazioni generiche, nomi commerciali, marchi registrati ecc., anche se non specificatamente identificati, non implica che tali denominazioni o marchi non siano protetti dalle relative leggi e regolamenti.

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer-Verlag Italia S.r.l.
The registered company address is: Via Decembrio 28, 20137 Milano, Italy

*A mia moglie Melina e ai miei figli Sara e Paolo,
che mi hanno aiutato con il loro affetto in questo lavoro,
concedendomi tutto il tempo necessario,
spesso sottratto alle loro necessità.
A mio Padre e mia Madre, che non ho fatto in tempo
a ringraziare come avrei voluto.*

Giuliano Donzellini

*A mio Padre e mia Madre per il mio essere spesso lontano.
A Federica per essermi sempre vicina.*

Luca Oneto

*A mia moglie Luisa per tutto il tempo che mi ha dedicato ed il
suo costante sostegno ed incoraggiamento di tutta una vita.
Alla memoria di Sandro Chiabrera, maestro e amico.*

Domenico Ponta

*Ai miei genitori e nonni che mi hanno sopportato da bambino.
A mio marito che mi sopporta da adulto.*

Davide Anguita

Prefazione del Prof. Filippo Sorbello

Con vero piacere presento il libro “Introduzione al progetto dei sistemi digitali” degli amici e Colleghi Donzellini, Oneto, Ponta e Anguita. Il testo è rivolto agli allievi di un primo corso di reti logiche e tratta i sistemi digitali partendo dalle basi teoriche, scelte ed approfondite al giusto livello, fino ad arrivare all’analisi ed alla sintesi delle reti combinatorie e delle reti sequenziali. Numerosi sono gli esempi e gli esercizi proposti per i quali viene data la soluzione.

L’evoluzione delle tecnologie elettroniche ha portato a una sempre più ampia diffusione dei sistemi digitali in ogni ambito della vita quotidiana. La velocità, la densità e la complessità degli attuali circuiti digitali sono stati resi possibili dalle metodologie automatiche di progetto e dal progresso della tecnologia.

La conoscenza delle basi teoriche delle reti logiche è comunque indispensabile per una padronanza delle architetture dei sistemi digitali di qualunque complessità ed anche per un corretto uso degli strumenti di progettazione automatica basati sui linguaggi per la descrizione dell’hardware HDL.

Gli allievi dei corsi iniziali non possiedono adeguate capacità di programmazione e di astrazione e neanche le conoscenze di fisica ed elettronica necessarie per utilizzare in modo adeguato i linguaggi HDL. Nel testo questa difficoltà viene superata avvalendosi di uno strumento di simulazione (*Deeds*), sviluppato da uno degli autori, che utilizza una interfaccia di uso immediato. *Deeds* viene usato per simulare il comportamento sia dei circuiti proposti nel testo sia dei circuiti che in autonomia, ciascuno di loro, vuole verificare dopo averli progettati. I progetti messi a punto con *Deeds* si possono poi esportare in linguaggio HDL o provare sui circuiti FPGA.

L’uso dei linguaggi per la descrizione dell’hardware unito alla conoscenza delle basi teoriche dei circuiti logici costituiscono il bagaglio di conoscenze indispensabili per entrare nel mondo digitale.

VIII

Ritengo che il testo sia un ottimo strumento per affrontare questa sfida vista la capacità che gli Autori hanno usato nel trasferire le necessarie conoscenze teoriche e professionali in un testo chiaro nei contenuti, scorrevole nella forma, gradevole nell'aspetto.

Palermo (Italia), 2 settembre 2017

Filippo Sorbello

Prefazione del Prof. Mauro Olivieri

Il libro di testo scritto da Giuliano Donzellini, Luca Oneto, Domenico Ponta e Davide Anguita si caratterizza per due note distintive nel vasto panorama dei testi universitari introduttivi alla progettazione digitale.

La prima nota distintiva è la sua focalizzazione su un ben definito insieme di nozioni e strumenti che costituiscono il fondamento dei sistemi digitali, ovvero la sintesi logica combinatoria e sequenziale e gli argomenti strettamente correlati ad essa. Il libro non si estende ai circuiti né ai sistemi a microprocessore, e rimanendo in tali confini si permette grande chiarezza, precisione, completezza ed autoconsistenza di sicuro giovamento per chi apprende. L'ispezione visiva dei numerosi schemi grafici e diagrammi a corredo del testo rende immediatamente al lettore questa impressione. La presenza di esercizi risolti insieme al software per la simulazione *Deeds*, inoltre, è un elemento decisivo giustamente sempre richiesto dagli studenti.

La seconda nota distintiva è l'equilibrio fra l'impianto teorico e il riferimento all'implementazione pratica, che consente un reale e solido apprendimento. Sebbene nel testo non si citi mai una volta la parola "tensione", né altre grandezze elettriche, lo studente che si serve del libro ha sempre presente che sta studiando la formalizzazione di un sistema elettronico. Allo stesso tempo, il testo non si abbandona alla facile tentazione di presentare la materia come una serie di esempi pratici di progetto senza basi teoriche. Una particolarità, questa, tipica di quella che definirei "scuola genovese" di elettronica digitale, della quale mi considero un esponente esterno.

Per le suddette motivazioni, il libro di Donzellini, Oneto, Ponta e Anguita è un testo prezioso per lo studente che si accinge a comprendere a fondo i concetti alla base del grande mondo della progettazione elettronica digitale.

Roma (Italia), 2 settembre 2017

Prof. Mauro Olivieri

Prefazione degli autori

La grande e continuamente crescente complessità dei sistemi digitali di oggi pone pesanti richieste ai sistemi educativi che hanno il compito di formare le nuove generazioni di progettisti o anche solo di fornire una solida conoscenza di base sul digitale. Le istituzioni accademiche fanno fatica a tenere il passo con l'avanzamento delle tecnologie ed è compito di chi, come gli autori di questo lavoro, è incaricato della formazione a livello introduttivo ed intermedio, di affrontare il problema e compiere scelte.

È certamente ovvio che un progettista digitale deve essere addestrato all'uso di linguaggi per la descrizione dell'hardware (HDL) ed è ormai pratica corrente introdurli molto presto nei corsi, affiancando o sostituendo l'approccio tradizionale basato su componenti e schemi. La scelta di descrivere i sistemi digitali tramite HDL permette anche l'adozione di FPGA (Field Programmable Gate Arrays) per la realizzazione pratica del progetto, usando schede prototipo fornite dai produttori dei chip.

Tuttavia, a nostro parere, l'adozione di un HDL in un primo corso di reti logiche con risorse ridotte in termini di crediti (come nel nostro caso), presenta problemi. Crediamo che, sostituendo componenti logici e schemi con un HDL, non sia facile costruire, nello spazio di un corso introduttivo, una buona comprensione dei fondamenti dei circuiti digitali, in quanto sono richiesti un livello di astrazione ed una familiarità con la programmazione che gli studenti del primo anno generalmente non possiedono ancora.

Inoltre, l'impiego del software di simulazione e sintesi fornito dai produttori dei chip FPGA, anche a livello di base, pone ulteriori problemi. Tools sviluppati per il progettista di sistemi digitali non soddisfano necessariamente le esigenze della formazione: il loro uso non è immediato da parte degli studenti, che possono imparare ad usarli in modo parziale e meccanico, con il rischio che, in questo modo, tralascino importanti concetti base, nascosti sotto i tecnicismi degli HDL e degli strumenti di lavoro.

Si tratta quindi di fare in modo che gli studenti acquisiscano una solida base sulla quale costruire le loro capacità progettuali e, allo stesso tempo, adattarsi al passo veloce dell'innovazione tecnologica e quindi acquistare dimestichezza con linguaggi e strumenti di progetto.

Per queste ragioni il libro mantiene un approccio tradizionale alle reti logiche, descritte e progettate per mezzo di simboli e schemi, pur tenendo conto dello stato dell'arte di oggi nella scelta degli argomenti e, soprattutto, a livello di esercizi e progetti. Questa caratteristica ne permette un impiego ottimale nei corsi di studio che non prevedono approfondimenti nel progetto delle reti, mentre fornisce una solida base per chi andrà avanti in questa direzione.

Il libro è autosufficiente, in quanto contiene, oltre alla parte teorica, un gran numero di esempi e di esercizi, completi di soluzioni. In corsi in cui ci sia spazio per attività di laboratorio si può sfruttare, con notevoli vantaggi didattici, la simbiosi con uno strumento di simulazione, *Deeds (Digital Electronics Education and Design Suite)*, sviluppato nel corso degli anni da uno degli autori (Giuliano Donzellini), con lo scopo preciso di supportare apprendimento e attività di laboratorio per i nostri studenti dell'ingegneria dell'informazione. La stretta connessione con *Deeds* rappresenta un importante elemento di forza e di originalità del nostro lavoro, in quanto tutti gli schemi, esempi e esercizi di progetto, dal più semplice al più complesso, presentati in questo libro, sono stati creati con *Deeds* e sono disponibili online per l'immediata simulazione.

L'ambiente *Deeds* copre tutti i principali aspetti del progetto di sistemi digitali, dalle reti combinatorie e sequenziali alle macchine a stati finiti e i sistemi a microprocessore “*embedded*”, permettendo il progetto e la simulazione di reti abbastanza complesse contenenti logiche standard, macchine a stati finiti, componenti definiti dall'utente e microprocessori, inclusa la loro programmazione in *linguaggio assembly*.

Deeds è stato sviluppato con in mente una estrema facilità d'uso, insieme a caratteristiche quasi professionali. Le differenze principali tra *Deeds* e un simulatore professionale sono la semplicità ed immediatezza dell'interfaccia con l'utente e la disponibilità di una vasta collezione di materiale didattico e progetti. *Deeds* è un sistema “vivo” in continua evoluzione: aggiornamenti sono disponibili periodicamente per migliorare gli strumenti esistenti ed aggiungerne di nuovi. Lo stesso vale per il materiale didattico.

La transizione verso i dispositivi FPGA avviene grazie alla possibilità di esportare un intero progetto creato e simulato con *Deeds* verso uno strumento professionale e di provarlo realmente in hardware. *Deeds* consente di evitare la complessità dell'intero processo che normalmente è necessario sul software specifico professionale; permette ad uno studente alle prime armi di non dover scrivere codice HDL, che è generato automaticamente da *Deeds*. Il vasto repertorio del materiale di apprendimento di *Deeds* è quindi ridiretto verso la realizzazione con FPGA senza bisogno di sostanziali modifiche.

Tuttavia, a valle della generazione automatica del codice HDL da parte di *Deeds*, gli studenti avanzati possono interagire direttamente con gli strumenti specifici FPGA, avendo quindi la possibilità di osservare, modificare e riusare il codice HDL (VHDL nel nostro caso), compiendo così una vera graduale transizione verso le tecniche di progettazione moderne.

Obiettivi didattici

Secondo l'esperienza degli autori l'intero contenuto del libro, accompagnato da esercitazioni di progetto e simulazione basate su *Deeds*, può essere usato in un corso introduttivo ai sistemi digitali di almeno 9 crediti.

Qui sotto riportiamo in modo sintetico i contenuti dei capitoli, indicando in corsivo gli argomenti che possono essere tralasciati senza perdere continuità con il progetto didattico, in corsi con un numero inferiore di crediti:

1. Algebra booleana e reti combinatorie
 - Trattazione classica che non richiede conoscenze preliminari.
Si possono tralasciare i teoremi di Shannon.
2. Progetto di reti combinatorie
 - Sintesi e minimizzazione con le mappe di Karnaugh.
 - Reti combinatorie standard.
 - Ritardi di propagazione.
Le mappe con variabili riportate e le alee possono essere omesse.
3. Aritmetica binaria
 - Trattazione classica.
 - Reti aritmetiche.
La trattazione dei numeri negativi in binario può essere omessa, come pure l'aritmetica BCD.
4. Complementi sul progetto di reti combinatorie
 - Minimizzazione di espressioni con il metodo Quine-McCluskey.
L'intero capitolo può essere omissso.
5. Introduzione alle reti sequenziali
 - Transizione intuitiva dalle reti combinatorie alle sequenziali.
 - Struttura e funzionamento dei principali tipi di flip-flop.
 - Caratteristiche dinamiche dei flip-flop.
Si possono trattare soltanto i tipi logici "D" ed "E", senza entrare in dettagli circuitali.

6. Reti sincrone di flip-flop

- Introduzione alle reti sincrone di flip-flop.
- Reti sequenziali: registri e contatori.
- Tecniche di analisi temporale delle reti sincrone.

La trattazione dei contatori e dei registri può essere ridotta, come pure l'analisi temporale delle reti sequenziali.

7. Reti sequenziali come Macchine a Stati Finiti

- Progetto della MSF, realizzata tramite i diagrammi ASM.
- Esercizi risolti di diagrammi ASM.
- Sintesi della MSF con tabelle di stato e mappe.

La sintesi della MSF può essere ridotta, omettendo le mappe a variabili riportate, o tralasciata del tutto.

8. La Macchina a Stati Finiti come controllore di sistema.

- Progettazione di sistemi controllore - datapath.
- Esercizi risolti sui sistemi controllore - datapath.

Il capitolo rappresenta l'obiettivo ultimo del corso: la trattazione ed il livello degli esercizi possono essere adattati alle esigenze e ai limiti della situazione didattica.

Come usare al meglio il libro

La strettissima connessione di questo libro con l'ambiente *Deeds* consiglia di usarlo in simbiosi con gli strumenti di simulazione, sia per verificare e confermare in modo attivo i concetti e le procedure esposte nel libro, sia per avere un supporto per la soluzione degli esercizi e il progetto dei sistemi.

Questa pratica "learning by doing" permette di costruire progressivamente le capacità di analisi e di progetto che costituiscono l'obiettivo da raggiungere.

Materiale digitale di supporto al libro

Questo libro di testo alterna parti teoriche, esempi, esercizi e soluzioni. Tutti gli esempi e gli esercizi presentati nel libro sono stati realizzati con il simulatore *Deeds* reperibile al link:

<https://www.digitalelectronicsdeeds.com>

Nel sito sono descritte le caratteristiche del simulatore e le istruzioni su come scaricarlo e utilizzarlo (in ambiente PC Windows). L'uso del simulatore è *in locale* e non richiede il costante collegamento in rete.

Nello stesso sito abbiamo reso disponibili, come materiale addizionale al libro, gli schemi *Deeds* relativi alla quasi totalità delle figure ed esempi proposti. È inoltre disponibile tutto il materiale necessario allo svolgimento degli esercizi e alla verifica del comportamento delle soluzioni mediante *Deeds*.

Il materiale è stato ordinato seguendo la denominazione dei capitoli, delle sezioni e delle sottosezioni del libro stesso, in modo da facilitarne la fruizione. Nello stesso sito saranno resi disponibili eventuali futuri aggiornamenti, correzioni, e miglioramenti del libro.

Indice

Prefazione del Prof. Filippo Sorbello	VII
Prefazione del Prof. Mauro Olivieri	IX
Prefazione degli autori	XI
Materiale digitale di supporto al libro	XV
Indice	XVII
1 Algebra booleana e reti combinatorie	1
1.1 Grandezze analogiche e logiche	1
1.2 Variabili booleane	4
1.3 Funzioni booleane	5
1.4 Tabelle di verità	5
1.5 Definizione dell'algebra booleana.	6
1.6 Proprietà fondamentali dell'algebra booleana.	7
1.7 Altre operazioni	10
1.8 Insiemi di operazioni funzionalmente completi	12
1.9 Teorema di espansione di Shannon	15
1.10 Livello di un'espressione booleana	18
1.11 Letterali	18
1.12 Prodotti fondamentali	18
1.13 Somme fondamentali	19
1.14 Implicanti	19
1.15 Implicanti primi	19
1.16 Reti combinatorie	20
1.16.1 Esempio: analisi di rete logica	20
1.16.2 Esempio: analisi di rete logica a due livelli	20
1.16.3 Esempio: schema circuitale di una rete logica (1)	21
1.16.4 Esempio: schema circuitale di una rete logica (2)	21

1.16.5	Esempio: definizione del comportamento di una rete logica	22
1.16.6	Esempio: schema circuitale da tabella di verità	23
1.16.7	Esempio: controllo di un impianto di riscaldamento	23
1.16.8	Esempio: selettore (multiplexer) a due canali	25
1.17	Esercizi	28
1.18	Soluzioni	30
2	Progetto di reti combinatorie	33
2.1	Mappe di Karnaugh	33
2.2	Sintesi AND-OR con uso delle mappe	36
2.2.1	Implicanti e implicanti primi nelle mappe	38
2.2.2	Minimizzazione con l'uso delle mappe	40
2.2.3	Mappe "a scacchiera"	41
2.2.4	Esempi di sintesi AND-OR	41
2.3	Sintesi OR-AND	43
2.3.1	Sintesi della funzione negata	44
2.4	Sintesi NAND-NAND	45
2.5	Sintesi NOR-NOR	46
2.6	Reti combinatorie standard	46
2.6.1	Decodificatore (Decoder)	46
2.6.2	Selettore (multiplexer)	50
2.6.3	Deselettore (demultiplexer)	52
2.6.4	Decodificatore per display a sette segmenti	54
2.6.5	Decodificatore BCD - 7 segmenti (con l'impiego delle indifferenze)	56
2.6.6	Uso di selettori per la sintesi di reti combinatorie	57
2.7	Mappe con variabili riportate	59
2.7.1	Sintesi di una mappa con variabili riportate	59
2.7.2	Le variabili riportate e i teoremi di espansione	62
2.8	Comportamento nel tempo dei circuiti logici	63
2.8.1	Tempi Caratteristici	63
2.8.2	Alee	65
2.8.3	Eliminazione delle alee statiche	67
2.8.4	Note sulla eliminazione delle alee	69
2.9	Esercizi	70
2.9.1	Mappe	70
2.9.2	Alee	72
2.10	Soluzioni	73
2.10.1	Mappe	73
2.10.2	Alee	76

3	Aritmetica binaria	79
3.1	Informazione binaria	79
3.2	Numerazione binaria (BIN)	80
3.2.1	Conversione da numerazione binaria a decimale	80
3.2.2	Conversione da numerazione decimale a binaria	81
3.2.3	Massimo numero rappresentabile	82
3.3	Numerazione ottale (OCT)	82
3.4	Numerazione esadecimale (HEX)	83
3.5	Generalità sui codici binari	85
3.6	Aritmetica binaria	86
3.6.1	Somma	86
3.6.2	Sottrazione	88
3.6.3	Prodotto	89
3.7	Aritmetica BCD 8421	89
3.8	Numeri razionali in binario	90
3.9	Reti aritmetiche	91
3.9.1	Semi-sommatore (half adder)	91
3.9.2	Sommatore (full adder)	91
3.9.3	Sommatore con riporto in cascata	93
3.9.4	Unità aritmetico-logica (ALU, Arithmetic Logic Unit) ..	93
3.10	Numeri relativi in binario	95
3.10.1	Rappresentazione in codice “modulo e segno”	95
3.10.2	Complementazione	96
3.10.3	Rappresentazione in codice “complemento a uno”	99
3.10.4	Rappresentazione in codice “complemento a due”	100
3.10.5	Estensione del segno	101
3.11	Rappresentazione di numeri reali	102
3.12	Codici alfanumerici	103
3.13	Codici a rivelazione di errore: generatore e rivelatore di parità ..	105
3.14	Esercizi	108
3.14.1	Numerazione Binaria	108
3.14.2	Numeri binari con segno	108
3.14.3	Numerazione Ottale e Esadecimale	109
3.15	Soluzioni	110
3.15.1	Numerazione Binaria	110
3.15.2	Numeri binari con segno	111
3.15.3	Numerazione Ottale e Esadecimale	112
4	Complementi sul progetto di reti combinatorie	115
4.1	Minimizzazione di espressioni booleane con il metodo di Quine-McCluskey	115
4.1.1	Fase di espansione	116
4.1.2	Fase di copertura	119
4.1.3	Funzioni non completamente specificate	120
4.1.4	Ottimizzazione della fase di copertura	121

4.1.5	Ottimizzazione contemporanea di più funzioni	126
4.2	Esercizi	131
4.2.1	Quine-McCluskey: sintesi di singola funzione	131
4.2.2	Quine-McCluskey: sintesi congiunta di più funzioni	134
4.3	Soluzioni	137
4.3.1	Quine-McCluskey: sintesi di singola funzione	137
4.3.2	Quine-McCluskey: sintesi congiunta di più funzioni	139
5	Introduzione alle reti sequenziali	141
5.1	Dalle reti combinatorie alle reti sequenziali	141
5.1.1	Esempio introduttivo	142
5.1.2	Memorizzare un bit di informazione: il flip-flop	143
5.1.3	Tipi logici e tipi di comando dei flip-flop	145
5.2	Flip-flop a comando diretto	145
5.2.1	Flip-flop SR (a comando diretto)	146
5.2.2	Flip-flop D (a comando diretto)	150
5.2.3	Flip-flop JK (a comando diretto)	150
5.3	Inizializzazione di una rete sequenziale	152
5.3.1	Ingressi di inizializzazione dei flip-flop	153
5.3.2	Generazione del segnale di inizializzazione	155
5.4	Flip-flop a comando abilitato a livello	156
5.4.1	Flip-flop SR con abilitazione a livello (SR-Latch)	156
5.4.2	Flip-flop D con abilitazione a livello (<i>D-Latch</i>)	158
5.4.3	Flip-flop JK con abilitazione a livello (JK-Latch)	159
5.5	Sincronizzazione delle reti sequenziali	160
5.5.1	Il segnale di sincronizzazione	160
5.5.2	Comando impulsivo dei flip-flop abilitati a livello	161
5.5.3	Il “Clock” e il “comando abilitato sul fronte”	163
5.5.4	La struttura “master-slave”	164
5.6	Flip-flop a comando abilitato sul fronte	166
5.6.1	Flip-flop D-PET	166
5.6.2	Flip-flop E-PET	170
5.6.3	Flip-flop JK-PET	172
5.6.4	Flip-flop T-PET	174
5.6.5	Inizializzazione sincrona dei flip-flop	174
5.7	Tempi caratteristici dei flip-flop	175
5.7.1	Relazione tra i tempi di propagazione e di mantenimento	176
5.7.2	Massima frequenza del clock di una rete con flip-flop	176
5.8	Flip-flop: simboli grafici e tabelle	178
5.8.1	Tipi logici	178
5.8.2	Tipi di comando	179
5.8.3	Tabelle di eccitazione	181
5.9	Esercizi	183
5.10	Soluzioni	186

6	Reti sincrone di flip-flop	189
6.1	Segnali sincroni e asincroni	191
6.1.1	Sincronizzatore	192
6.1.2	Sincronizzatore a più stadi	193
6.2	Registri	194
6.2.1	Registro parallelo	194
6.2.2	Registro a scorrimento	196
6.2.3	Registro a scorrimento con caricamento parallelo	199
6.2.4	Registro a scorrimento universale	202
6.3	Contatori	205
6.3.1	Contatore binario	205
6.3.2	Contatore con abilitazione	211
6.3.3	Contatore bidirezionale	213
6.3.4	Contatori “universali”	216
6.3.5	Contatori asincroni	219
6.4	Analisi di reti	221
6.4.1	Esempio n. 1	221
6.4.2	Esempio n. 2	224
6.4.3	Esempio n. 3	226
6.4.4	Esempio n. 4	228
6.4.5	Esempio n. 5	230
6.5	Esercizi	234
6.6	Soluzioni	254
7	Reti sequenziali come Macchine a Stati Finiti	265
7.1	Modello generale di Macchina a Stati Finiti	266
7.1.1	Macchine Sincrone e Asincrone	266
7.1.2	Macchine di Moore e di Mealy	267
7.1.3	Esempio di Macchina a Stati Finiti Sincrona	268
7.1.4	Equazioni generali dello stato successivo e delle uscite	269
7.2	Diagrammi ASM	271
7.2.1	Descrizione degli stati	271
7.2.2	Ingressi	275
7.2.3	Uscite condizionate	284
7.3	Esempi di costruzione di diagrammi ASM	288
7.3.1	Esempi introduttivi	288
7.3.2	Generatore di impulsi a rapporto pieno/vuoto regolabile	298
7.3.3	Riconoscitore di sequenza	303
7.3.4	Trasmettitore Seriale Sincrono (2 bit)	305
7.3.5	Ricevitore di comando in formato seriale sincrono	307
7.3.6	Ricevitore Seriale Sincrono (2 bit)	310
7.3.7	Gestione di pulsanti	313
7.3.8	Registro a Scorrimento (3 bit)	316
7.3.9	Rete sequenziale con uscita condizionata	319
7.3.10	Registro a scorrimento con albero di EXOR	321

7.4	Sintesi della MSF sincrona	323
7.4.1	Assegnazione degli stati	324
7.4.2	Descrizione della MSF mediante tabella degli stati	325
7.4.3	Sintesi dalla tabella degli stati	327
7.4.4	Esempi di sintesi di MSF sincrone	330
7.5	Comportamento nel tempo della MSF sincrona	344
7.6	Esercizi	350
7.6.1	Analisi di reti sequenziali in termini di MSF	350
7.6.2	Progetto di MSF a partire da specifiche testuali	353
7.7	Soluzioni	358
7.7.1	Analisi di reti sequenziali in termini di MSF	358
7.7.2	Progetto di MSF a partire da specifiche testuali	362
8	La Macchina a Stati Finiti come controllore di sistema	371
8.1	I sistemi digitali	371
8.2	Sistemi a controllo aperto	372
8.2.1	Ricevitore seriale (2 bit)	373
8.3	Sistemi a controllo retroazionato	377
8.3.1	Ricevitore e trasmettitore seriale (2 bit)	377
8.3.2	Generatore di impulso	380
8.3.3	Ricevitore seriale (8 bit)	385
8.3.4	Regolatore di Luminosità per Lampada	392
8.3.5	Serratura a combinazione	396
8.3.6	Distributore automatico di bevande	401
8.3.7	Progetto di generatore di onda quadra programmabile ..	405
8.3.8	Progetto di sistema per luminarie natalizie	409
8.4	Esercizi	415
8.4.1	Progetto di controllore, con datapath assegnato	415
8.4.2	Progetto di sistema completo (controllore e datapath) ..	434
8.5	Suggerimenti	437
8.5.1	Progetto di sistema completo (controllore e datapath) ..	437
8.6	Soluzioni	444
8.6.1	Progetto di controllore, con datapath assegnato	444
8.6.2	Progetto di sistema completo (controllore e datapath) ..	462

Algebra booleana e reti combinatorie

1.1 Grandezze analogiche e logiche

In tutti i campi del sapere umano si ha a che fare con l'osservazione, la memorizzazione, l'elaborazione e la comunicazione dell'informazione. La definizione di “*informazione*” può sembrare ovvia, perché il termine è usato comunemente nel linguaggio quotidiano, ma per i nostri scopi abbiamo bisogno di una definizione che non lasci spazio ad ambiguità di interpretazione. Ricorriamo quindi a R. V. L. Hartley (1888-1970), uno dei padri della *Teoria dell'Informazione*, che ci assiste con la seguente definizione:

“L'informazione è una diminuzione di incertezza.”

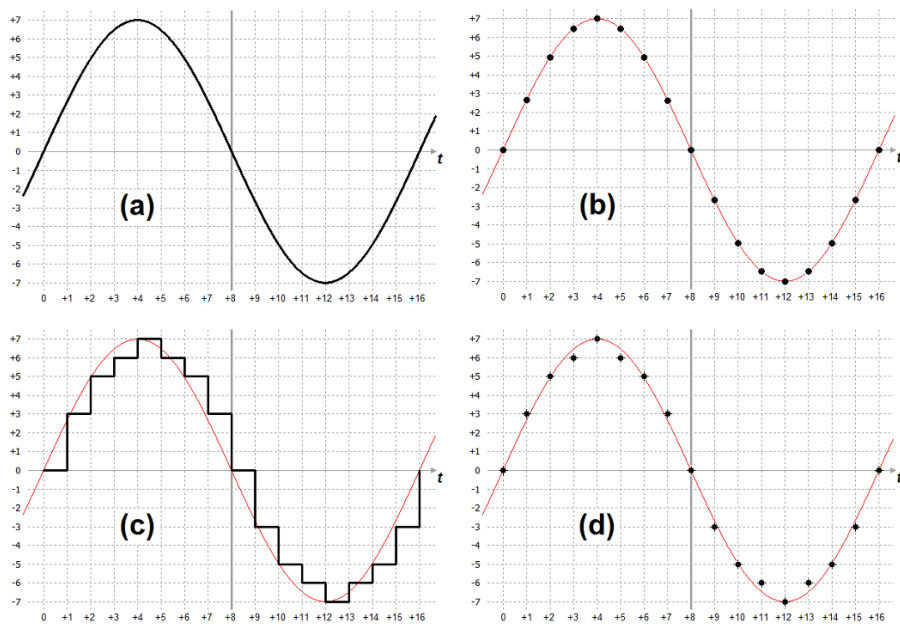
Appare subito chiaro che l'informazione è legata ad un “prima” e un “dopo”, relativamente ad un evento che ha qualche possibilità di verificarsi, e grazie alla quale un osservatore diminuisce la propria incertezza relativamente all'evento stesso. Da questa definizione discende facilmente che l'informazione può essere veicolata attraverso l'utilizzo di qualche grandezza fisica che varia nel tempo o nello spazio. Si pensi ad esempio all'informazione trasmessa da uno schermo di un computer attraverso immagini, ovvero variazioni di luminosità e di colore nel tempo e nello spazio, o a quella trasmessa dall'auricolare di un telefono, attraverso un suono, ovvero variazioni della pressione dell'aria nel tempo.

Allo scopo di studiare l'elaborazione delle informazioni non ricorremo direttamente ad una grandezza fisica ma, piuttosto, ad una sua rappresentazione numerica, che indicheremo con “ G ”, ed alla sua variazione nel tempo, che indicheremo con “ T ”.

Questo approccio ci permetterà di dividere la rappresentazione in due grandi famiglie: se G può variare con continuità tra un valore ed un altro, assumendo tutti gli infiniti valori intermedi, diciamo che utilizziamo una rappresentazione “analogica”. Invece, se G può assumere solo un numero finito di valori, diciamo che utilizziamo una rappresentazione “discreta” o “digitale”, che diventa “binaria” o “logica” se i valori possibili sono solo due.

Si noti che spesso si fa confusione tra una grandezza fisica e la sua rappresentazione, ma noi siamo interessati solo a quest'ultima. La luce, ad esempio, può essere descritta sia attraverso una rappresentazione discreta (*fotoni*) che continua (*onde elettromagnetiche*), ma non è tra i nostri scopi individuare quale sia la rappresentazione “vera”, ammesso che esista: questo è un obiettivo che lasciamo volentieri ai filosofi. Per l'ingegnere ciò che importa è usare lo strumento più appropriato per risolvere il problema in esame.

In figura sono raffigurate le quattro possibili combinazioni che si ottengono utilizzando valori discreti o continui per G e T [le curve in rosso sono rappresentate solo per riferimento]:



- (a) grandezza continua che varia nel tempo;
- (b) grandezza continua e campionata nel tempo;
- (c) grandezza quantizzata in ampiezza e continua nel tempo;
- (d) grandezza quantizzata in ampiezza e campionata nel tempo.

Nel caso analogico lo strumento principale a nostra disposizione è la matematica che studia i numeri reali e le funzioni definite su di essi, ovvero l'algebra e il calcolo infinitesimale. La rappresentazione analogica è molto efficace per trattare le grandezze fisiche naturali a livello macroscopico.

Ciò non significa che le grandezze naturali siano analogiche, ma solo che in questo caso la rappresentazione analogica è la più comoda o efficace. Infatti, se scendiamo a livello microscopico la materia rivela la sua natura discreta (atomi e particelle) e la rappresentazione analogica non è necessariamente la

più conveniente. In generale, si rende necessaria la presenza di un dispositivo di ingresso ad un sistema di elaborazione che converta le grandezze continue in grandezze digitali ed eventualmente un convertitore che svolga la funzione opposta dal sistema verso l'uscita. Questi dispositivi si chiamano rispettivamente “convertitore analogico/digitale” (*analog to digital converter* o *ADC*) e “convertitore digitale/analogico” (*digital to analog converter* o *DAC*) ma non verranno trattati in questo corso.

Il caso digitale può essere affrontato con la matematica discreta, che però è generalmente molto più complessa della matematica sui numeri reali. Se ci limitiamo al caso in cui G può assumere solo due valori, ovvero ci limitiamo al caso binario, possiamo ricorrere all'*algebra booleana*, dal suo inventore, il matematico irlandese George Boole (1815-1864). L'*algebra di Boole* sarà sufficiente per i nostri scopi, ovvero per affrontare le basi del progetto di reti logiche e sistemi digitali.

Le grandezze binarie di indicano normalmente con i simboli $\{0, 1\}$ oppure, a seconda dal particolare contesto, anche con altri simboli, come $\{-1, +1\}$, $\{L, H\}$ (abbreviazioni di *Low* e *High*) o $\{T, F\}$ (*True* e *False*).

Nel campo digitale utilizzeremo sia la rappresentazione a tempo continuo, detta “*asincrona*”, che quella a tempo discreto, detta “*sincrona*”. Una rete logica è detta *sincrona* se le sue parti operano simultaneamente, in accordo ad un segnale di sincronizzazione comune; è invece definita *asincrona* se le sue parti operano tra di loro in modo autonomo.

La rappresentazione binaria (digitale), rispetto a quella analogica, ha pregi e difetti:

- un valore analogico è una pura astrazione matematica, poiché richiede una precisione infinita per essere espresso;
- un valore discreto (binario) è facilmente memorizzabile, poiché richiede un numero finito (due) di valori della grandezza fisica utilizzata per la sua memorizzazione;
- la gestione e l'elaborazione di una grandezza binaria sono meno sensibili al rumore: è sufficiente che la quantità di rumore non sia così elevata da impedire la distinzione tra i livelli di un segnale (tale da confondere, cioè, il livello logico *alto* con il *basso*, o viceversa);
- è possibile gestire agevolmente la precisione del sistema (ovvero il numero di cifre con il quale il sistema rappresenta le informazioni che elabora);
- i dispositivi che elaborano le informazioni digitali, ovvero i sistemi e circuiti digitali o reti logiche, sono più semplici da progettare anche se la realizzazione fisica richiede un numero molto più elevato di componenti circuitali.

Le potenze di 2

Poiché trattiamo valori binari sarà utile avere ben presenti i valori delle potenze di 2, e ricordare le più significative. Qui di seguito le più piccole:

2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
1	2	4	8	16	32	64	128	256	512	1024

Si noti che $2^{10} = 1024$ è pari a circa $10^3 = 1000$; utilizzeremo il prefisso K (ovvero Kilo) per indicare il valore 1024 anche se lo standard IEC (*International Electrotechnical Commission*) imporrebbe di usare il prefisso Kibi (da Kilo binary):

2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
1K	2K	4K	8K	16K	32K	64K	128K	256K	512K	1024K

Analogamente 2^{20} (Mebi), pari a circa 10^6 , che indichiamo con M (Mega):

2^{20}	2^{21}	2^{22}	2^{23}	2^{24}	2^{25}	2^{26}	2^{27}	2^{28}	2^{29}	2^{30}
1M	2M	4M	8M	16M	32M	64M	128M	256M	512M	1024M

Indicheremo con la lettera G il Giga (10^9), che in realtà vale 2^{30} (Gibi):

2^{30}	2^{31}	2^{32}	2^{33}	2^{34}	2^{35}	2^{36}	2^{37}	2^{38}	2^{39}	2^{40}
1G	2G	4G	8G	16G	32G	64G	128G	256G	512G	1024G

Il valore 2^{40} (Tebi), all'incirca 10^{12} , viene indicato con T (Tera). Per completezza citiamo qui altri prefissi:

$$\begin{aligned}
 2^{50} \text{ (Pebi)} &\approx 10^{15} \text{ (Peta)}; & 2^{60} \text{ (Exbi)} &\approx 10^{18} \text{ (Exa)}; \\
 2^{70} \text{ (Zebi)} &\approx 10^{21} \text{ (Zetta)}; & 2^{80} \text{ (Yobi)} &\approx 10^{24} \text{ (Yotta)}.
 \end{aligned}$$

1.2 Variabili booleane

Sia X una certa variabile discreta. Chiameremo *variabile booleana* una qualunque variabile discreta che può assumere solo due valori. Questi valori si indicano con:

$$\begin{aligned}
 X = 0 & \quad \text{false (falso)} \\
 X = 1 & \quad \text{true (vero)}
 \end{aligned}$$

Nel seguito si useranno i valori 0, 1.

1.3 Funzioni booleane

Se abbiamo X_1, X_2, \dots, X_n variabili booleane, si dice funzione booleana

$$f(X_1, X_2, \dots, X_n)$$

una funzione che associa ad ogni elemento del dominio un valore booleano. Essa può assumere solo i valori 0, 1. Gli elementi che compongono il dominio di tale funzione sono numerabili: una funzione di n variabili ha il dominio formato dalle loro 2^n combinazioni. Due funzioni sono equivalenti se assumono lo stesso valore per qualunque combinazione dei valori delle variabili.

1.4 Tabelle di verità

Con il principio di induzione completa (ovvero eseguendo tutti le valutazioni) è possibile verificare il valore di f per tutti i 2^n punti del suo dominio. Rappresentiamo la funzione mediante tabelle di verità.

Supponiamo di avere una funzione di 3 variabili X_1, X_2, X_3 . E' possibile costruire una tabella con tutti i valori assunti da f :

X_1	X_2	X_3	f
0	0	0	valori
0	0	1	
0	1	0	assunti
0	1	1	
1	0	0	da
1	0	1	
1	1	0	f
1	1	1	

Osservazione: per scrivere i $2^3 = 8$ elementi del dominio, si parte dalla colonna più a destra (X_3), alternando uno 0 e un 1, iniziando dall'alto. Nella colonna accanto alterniamo due 0 e due 1; poi quattro 0 e quattro 1 nella colonna ancora a sinistra, e così via, raddoppiandone sempre il numero.

Esempi:

Ricavare le **tabelle di verità** dalle *definizioni a parole*:

		C	B	A	U
		0	0	0	0
		0	0	1	0
		0	1	0	0
		0	1	1	1
		1	0	0	1
		1	0	1	1
		1	1	0	1
		1	1	1	1

1. $\ll U$ è vera se C è vera, oppure se B e A sono entrambi veri \gg

2. $\ll Z$ è vera se il numero di uno presenti in ingresso è pari a due \gg	M	G	D	Z
	0	0	0	0
	0	0	1	0
	0	1	0	0
	0	1	1	1
	1	0	0	0
	1	0	1	1
	1	1	0	1
	1	1	1	0

1.5 Definizione dell'algebra booleana.

L'algebra booleana fornisce gli strumenti necessari per elaborare ed interpretare le informazioni presentate nel formato binario.

L'algebra di Boole è un sistema algebrico (cioè un insieme di elementi per il quale è definito un insieme di operazioni) definito da:

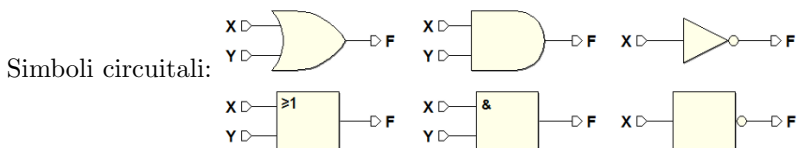
- L'insieme dei valori $\{0,1\}$;
- Le operazioni *OR*, *AND* e *NOT*;
- La relazione di equivalenza “=” con le proprietà:
 - riflessiva;
 - simmetrica;
 - transitiva.

Definiamo le tre operazioni:

Operazione:	OR	AND	NOT
	(somma logica)	(prodotto logico)	(negazione)

Simboli algebrici:	$X + Y$ $X \vee Y$ $X \cup Y$	$X \cdot Y = XY$ $X \wedge Y$ $X \cap Y$	$\text{NOT}(X) = \bar{X}$
--------------------	-------------------------------------	--	---------------------------

	$X \ Y \ \ X + Y$	$X \ Y \ \ X \cdot Y$	$X \ \ \bar{X}$																												
Tabella di verità:	<table style="width: 100%; border-collapse: collapse;"> <tr><td style="padding: 2px;">0</td><td style="padding: 2px;">0</td><td style="padding: 2px;">0</td></tr> <tr><td style="padding: 2px;">0</td><td style="padding: 2px;">1</td><td style="padding: 2px;">1</td></tr> <tr><td style="padding: 2px;">1</td><td style="padding: 2px;">0</td><td style="padding: 2px;">1</td></tr> <tr><td style="padding: 2px;">1</td><td style="padding: 2px;">1</td><td style="padding: 2px;">1</td></tr> </table>	0	0	0	0	1	1	1	0	1	1	1	1	<table style="width: 100%; border-collapse: collapse;"> <tr><td style="padding: 2px;">0</td><td style="padding: 2px;">0</td><td style="padding: 2px;">0</td></tr> <tr><td style="padding: 2px;">0</td><td style="padding: 2px;">1</td><td style="padding: 2px;">0</td></tr> <tr><td style="padding: 2px;">1</td><td style="padding: 2px;">0</td><td style="padding: 2px;">0</td></tr> <tr><td style="padding: 2px;">1</td><td style="padding: 2px;">1</td><td style="padding: 2px;">1</td></tr> </table>	0	0	0	0	1	0	1	0	0	1	1	1	<table style="width: 100%; border-collapse: collapse;"> <tr><td style="padding: 2px;">0</td><td style="padding: 2px;">1</td></tr> <tr><td style="padding: 2px;">1</td><td style="padding: 2px;">0</td></tr> </table>	0	1	1	0
0	0	0																													
0	1	1																													
1	0	1																													
1	1	1																													
0	0	0																													
0	1	0																													
1	0	0																													
1	1	1																													
0	1																														
1	0																														



1.6 Proprietà fondamentali dell'algebra booleana.

Convenzioni usate.

- Si considerano $X, Y, Z, X_1, X_2, X_3, \dots, X_n$, variabili booleane;
- le parentesi stabiliscono le priorità nei calcoli come nell'algebra ordinaria;
- l'AND prioritario rispetto all'OR (es. $X + YZ = X + (YZ)$).

Anche quest'ultima proprietà è in analogia con l'algebra ordinaria. Tutte le proprietà si possono dimostrare per mezzo dell'induzione completa, cioè verificandone la validità per ogni combinazione di valori assunti dalle variabili che compaiono nell'espressione.

Esempio: $X \cdot 0 = 0$ si verifica tramite la tabella di verità:

X	0	$X \cdot 0$
0	0	0
1	0	0

Dualità

Se una data espressione è valida, è valida anche l'espressione duale, cioè quella che si ottiene dall'espressione di partenza scambiando gli OR con gli AND e le costanti 0 con le costanti 1. Esempi:

$$\begin{array}{l} X + 1 = 1 \\ \text{(duale:)} \quad X \cdot 0 = 0 \end{array}$$

$$\begin{array}{l} X + 0 = X \\ \text{(duale:)} \quad X \cdot 1 = X \end{array}$$

Idempotenza

$$\begin{array}{l} X + X = X \\ \text{(duale:)} \quad X \cdot X = X \end{array}$$

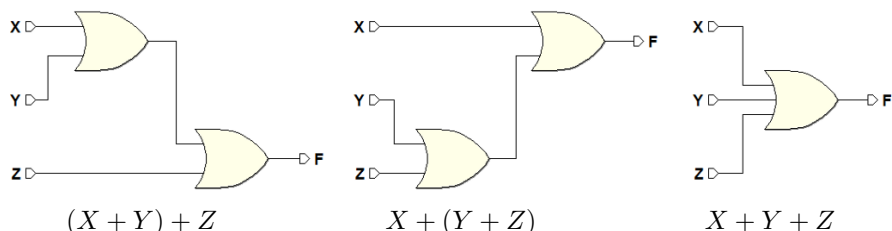
Commutatività

$$\begin{array}{l} X + Y = Y + X \\ \text{(duale:)} \quad X \cdot Y = Y \cdot X \end{array}$$

Associatività

$$\begin{array}{l} (X + Y) + Z = X + (Y + Z) = X + Y + Z \\ \text{(duale:)} \quad (X \cdot Y) \cdot Z = X \cdot (Y \cdot Z) = X \cdot Y \cdot Z \end{array}$$

L'associatività permette di estendere le operazioni fondamentali a più di due variabili. Le simbologie circuitali per la prima espressione sono:



Poiché non c'è distinzione fra i primi due circuiti, ha senso definire, in generale, un OR (e lo stesso vale per l'AND) a tre o più ingressi. Quindi è proprio la proprietà dell'associatività che ci permette di dare un senso a porte OR e AND con più di due ingressi. A questo punto le operazioni OR e AND andrebbero ridefinite, in modo più generale, per n ingressi:

- un OR a n ingressi dà uscita 0 se e solo se tutti gli n ingressi sono 0, altrimenti dà uscita 1;
- un AND a n ingressi dà uscita 1 se e solo se tutti gli n ingressi sono 1, altrimenti dà uscita 0.

Distributività

$$\begin{aligned} \text{Factoring law} & \quad (X + Y) \cdot (X + Z) = X + (Y \cdot Z) \\ \text{Distributive law (duale)} & \quad (X \cdot Y) + (X \cdot Z) = X \cdot (Y + Z) \end{aligned}$$

Dimostrazione della factoring law:

$$\begin{aligned} (X + Y) \cdot (X + Z) &= X \cdot X + X \cdot Z + X \cdot Y + Y \cdot Z = \\ &= X + X \cdot Z + X \cdot Y + Y \cdot Z \\ &= X \cdot (1 + Y) + X \cdot Z + Y \cdot Z \\ &= X + X \cdot Z + Y \cdot Z \\ &= X \cdot (1 + Z) + Y \cdot Z \\ &= X + (Y \cdot Z) \end{aligned}$$

Si poteva anche dimostrare con il principio di induzione completa (ovvero verificando tutte le possibili combinazioni per X, Y, Z):

X	Y	Z	$Y \cdot Z$	$X + YZ$	$X + Y$	$X + Z$	$(X + Y)(X + Z)$
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1

Risulta evidente che le colonne $X + Y \cdot Z$ e $(X + Y)(X + Z)$ sono uguali.

Complementazione

$$X + \bar{X} = 1$$

(duale:) $X \cdot \bar{X} = 0$

Assorbimento

$$X + X \cdot Y = X$$

Prima forma: (duale:) $X \cdot (X + Y) = X$

$$X + (\bar{X} \cdot Y) = X + Y$$

Seconda forma: (duale:) $X \cdot (\bar{X} + Y) = X \cdot Y$

Dimostrazioni:

$$\begin{aligned} X + X \cdot Y &= X \cdot (1 + Y) = X \cdot 1 = X \\ X \cdot (X + Y) &= X \cdot X + X \cdot Y = X + X \cdot Y = X \\ X + \bar{X} \cdot Y &= X + X \cdot Y + \bar{X} \cdot Y = X + Y(X + \bar{X}) = X + Y \\ X \cdot (\bar{X} + Y) &= X \cdot \bar{X} + X \cdot Y = X \cdot Y \end{aligned}$$

Consenso

$$X \cdot Y + Y \cdot Z + Z \cdot \bar{X} = X \cdot Y + Z \cdot \bar{X}$$

(duale:) $(X + Y)(Y + Z)(Z + \bar{X}) = (X + Y)(Z + \bar{X})$

Dimostrazioni:

$$\begin{aligned} X \cdot Y + Y \cdot Z + Z \cdot \bar{X} &= \\ &= X \cdot Y + Y \cdot (X + \bar{X}) \cdot Z + Z \cdot \bar{X} = \\ &= (X \cdot Y + X \cdot Y \cdot Z) + (Z \cdot \bar{X} \cdot Y + Z \cdot \bar{X}) = \\ &= X \cdot Y + Z \cdot \bar{X} \\ (X + Y)(Y + Z)(Z + \bar{X}) &= \\ &= (X + Y)[(X + Y + Z)(\bar{X} + Y + Z)](Z + \bar{X}) = \\ &= [(X + Y)(X + Y + Z)][(Z + \bar{X} + Y)(Z + \bar{X})] = \\ &= (X + Y)(Z + \bar{X}) \end{aligned}$$

InvoluzioneNota anche come *legge del NOT*:

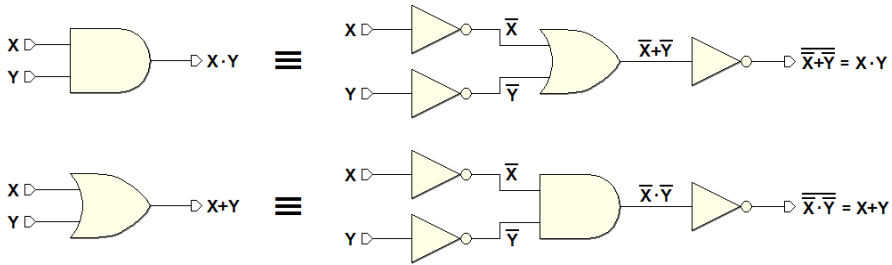
$$\overline{\bar{X}} = X$$

Dualizzazione o Teorema di De Morgan

Un prodotto logico tra due variabili può essere sostituito dalla negazione della somma logica delle stesse negate. Duale: una somma logica tra due variabili può essere sostituita dalla negazione del prodotto logico delle stesse negate:

$$X \cdot Y = \overline{\overline{X} + \overline{Y}}$$

(duale:) $X + Y = \overline{\overline{X} \cdot \overline{Y}}$



Questo teorema è importante: ci permette di ottenere la funzione di una AND per mezzo di una porta OR, e viceversa. Il teorema ci dice che una delle due funzioni AND e OR è superflua, nella definizione dell'algebra booleana.

Teorema di De Morgan generalizzato

Il teorema può essere generalizzato ad un numero qualunque di variabili:

$$X_1 \cdot X_2 \cdot \dots \cdot X_n = \overline{\overline{X_1} + \overline{X_2} + \dots + \overline{X_n}}$$

(duale:) $X_1 + X_2 + \dots + X_n = \overline{\overline{X_1} \cdot \overline{X_2} \cdot \dots \cdot \overline{X_n}}$

1.7 Altre operazioni

Definiamo in questo paragrafo altre operazioni dell'algebra booleana.

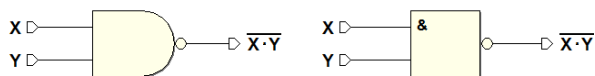
NAND

L'operazione NAND equivale ad una AND negata:

$$X \text{ nand } Y = \overline{X \cdot Y}$$

X	Y	(X nand Y)
0	0	1
0	1	1
1	0	1
1	1	0

Simboli circuitali:



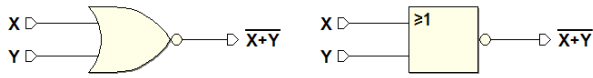
NOR

L'operazione NOR equivale ad una OR negata:

$$X \text{ nor } Y = \overline{(X + Y)}$$

X	Y	(X nor Y)
0	0	1
0	1	0
1	0	0
1	1	0

Simboli circuitali:



NAND e NOR godono della proprietà commutativa, ma **non** di quella associativa.

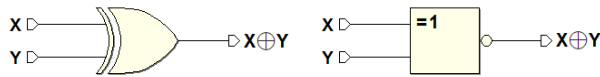
EXOR (OR esclusivo)

La funzione EXOR è detta di “anticoincidenza” (non coincidenza degli ingressi, fornisce 1 quando gli ingressi sono diversi):

$$X \oplus Y = X \text{ xor } Y = X \overline{Y} + \overline{X} Y$$

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

Simboli circuitali:



L'operazione di EXOR è commutativa e associativa. Se neghiamo la EXOR otteniamo la funzione di “coincidenza” (uguaglianza degli ingressi):

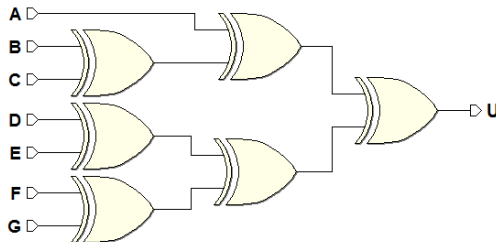
$$\overline{X \oplus Y} = X Y + \overline{X} \overline{Y}$$

EXOR generalizzato

È un EXOR a più ingressi, che scriviamo così:

$$X_1 \oplus X_2 \oplus \dots \oplus X_n = \begin{cases} 1 & \text{se gli 1 di ingresso sono in numero dispari} \\ 0 & \text{se gli 1 di ingresso sono in numero pari} \end{cases}$$

Sono realizzati con la struttura tipica detta **albero** di EXOR:



1.8 Insiemi di operazioni funzionalmente completi

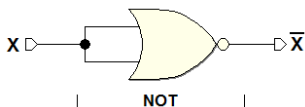
Abbiamo visto che l'algebra booleana si basa su un insieme di due elementi $\{0, 1\}$ e su un insieme di operazioni: OR, AND, NOT. D'altra parte il teorema di De Morgan mostra che una operazione tra quelle di AND e OR, a scelta, può essere considerata superflua, e che gli insiemi di operazioni OR/NOT oppure AND/NOT sono già sufficienti per costruire tutta l'algebra booleana. Ampliamo il discorso discutendo gli insiemi di operazioni a partire dai quali si può costruire equivalentemente l'algebra booleana, motivo per cui sono detti funzionalmente completi:

1. AND, OR, NOT
2. NOR
3. NAND
4. OR, NOT
5. AND, NOT
6. EXOR, AND
7. EXOR, OR

Nota: nella pratica risultano impiegati solo gli insiemi NOR e NAND.

Insieme {NOR}

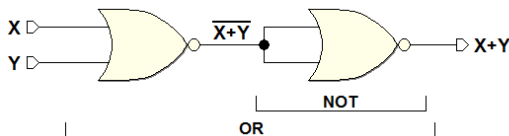
Possiamo ottenere OR e NOT dalle porte NOR. Se colleghiamo un NOR come nella figura che segue, otteniamo un NOT. Dalla tabella del NOR si ottiene, dato che i due ingressi X e Y sono collegati insieme:



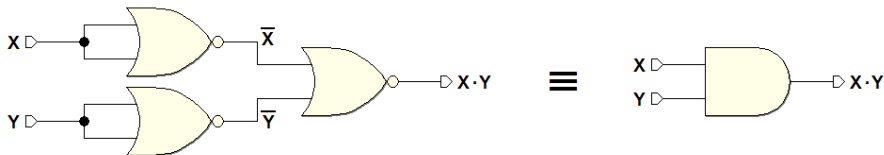
Infatti:

X	Y	$X \text{ nor } Y$
0	0	1
1	1	0

La porta OR si ottiene, invece, negando l'uscita della NOR, con il NOT:

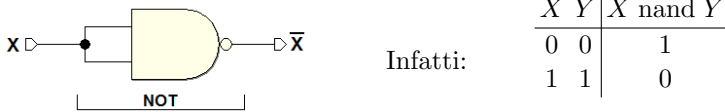


Per ottenere la AND applichiamo De Morgan: $X \cdot Y = \overline{\overline{X} + \overline{Y}}$. Si ha:

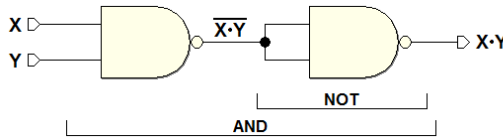


Insieme {NAND}

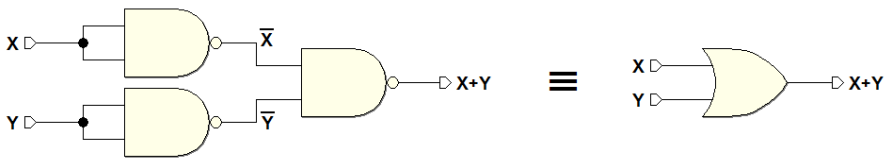
Analogamente a quanto visto prima, il NOT si ottiene come segue, tenendo conto delle due della tabella del NAND in cui i due ingressi X e Y sono uguali:



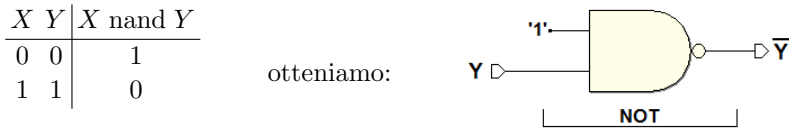
Per ottenere l'AND sarà dunque sufficiente far seguire il NAND da un NOT realizzato con una NAND.



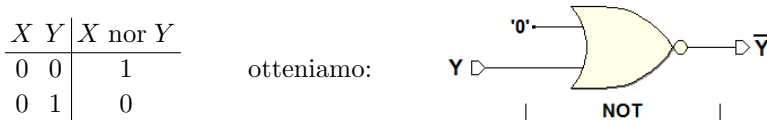
Dal teorema di De Morgan, infine, otteniamo l'OR:



Osservazione: c'è un altro modo di ottenere il NOT con il NAND. Se infatti si pone $X = 1$:



Analogamente, ponendo $X = 0$, per il NOR si ottiene:



Insieme {OR, NOT}

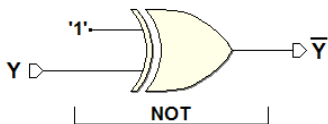
L'AND si ottiene utilizzando il teorema di De Morgan.

Insieme {AND, NOT}

L'OR si ottiene utilizzando il teorema di De Morgan.

Insieme {EXOR, AND}

Il NOT si ottiene dall'EXOR nel modo seguente:



Infatti, dalla tavola di verità dell'EXOR:

X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

ponendo $X = 1$:

X	Y	$X \oplus Y$
1	0	1
1	1	0

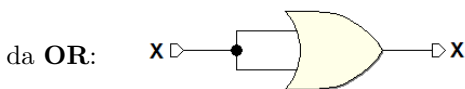
Nota: Se la costante 1 è cambiata in 0, si ottiene l'identità: è quindi possibile una funzione invertitore/identità "programmabile" al variare di quell'ingresso.

Insieme {EXOR, OR}

Il NOT si ottiene dall'EXOR e l'AND utilizzando il teorema di De Morgan.

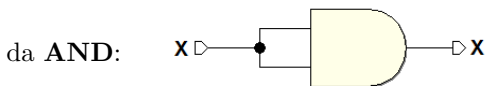
Identità

L'identità si può ottenere nei modi seguenti:



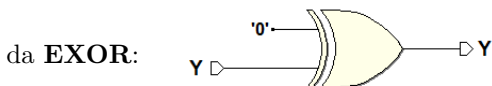
Infatti:

X	Y	$X + Y$
0	0	0
1	1	1



Infatti:

X	Y	$X \cdot Y$
0	0	0
1	1	1



Infatti:

X	Y	$X \oplus Y$
0	0	0
0	1	1

1.9 Teorema di espansione di Shannon

Prima forma

Una generica funzione booleana f può essere scomposta in questo modo:

$$f(X_1, X_2, X_3, \dots, X_n) = \overline{X_1} \cdot f(0, X_2, X_3, \dots, X_n) + X_1 \cdot f(1, X_2, X_3, \dots, X_n)$$

Il primo dei due termini ottenuti equivale alla funzione di partenza, ma limitatamente al caso $X_1 = 0$, e quindi è condizionata da $\overline{X_1}$; analogamente il secondo termine, che vale per $X_1 = 1$, ed è quindi condizionato da X_1 .

Il procedimento ora visto può essere iterato fino ad estrarre tutte le variabili dalla funzione:

$$\begin{aligned} f(X_1, X_2, X_3, \dots, X_n) &= \overline{X_1} \cdot \overline{X_2} \cdot f(0, 0, X_3, \dots, X_n) + \\ &\quad \overline{X_1} \cdot X_2 \cdot f(0, 1, X_3, \dots, X_n) + \\ &\quad X_1 \cdot \overline{X_2} \cdot f(1, 0, X_3, \dots, X_n) + \\ &\quad X_1 \cdot X_2 \cdot f(1, 1, X_3, \dots, X_n) = \\ &= \dots \end{aligned}$$

Alla fine, ogni voce del tipo $f(0, 1, \dots)$ risulterà essere una costante (0 o 1).

Da una funzione in n variabili si ottengono 2^n termini di prodotto in OR tra di loro, dove ogni termine comprende tutte le variabili (dirette o negate).

A titolo di esempio, scomponiamo una generica $f(C, B, A)$ in tre variabili:

$$\begin{aligned} f(C, B, A) &= \overline{C} \overline{B} \overline{A} \cdot f(0, 0, 0) + \\ &\quad \overline{C} \overline{B} A \cdot f(0, 0, 1) + \\ &\quad \overline{C} B \overline{A} \cdot f(0, 1, 0) + \\ &\quad \overline{C} B A \cdot f(0, 1, 1) + \\ &\quad C \overline{B} \overline{A} \cdot f(1, 0, 0) + \\ &\quad C \overline{B} A \cdot f(1, 0, 1) + \\ &\quad C B \overline{A} \cdot f(1, 1, 0) + \\ &\quad C B A \cdot f(1, 1, 1) \end{aligned}$$

La forma espansa che assume la funzione è detta *somma di prodotti* o *prima forma canonica*, o anche forma AND-OR.

Esempio

Vogliamo ricavare l'espressione analitica di una funzione booleana $f(C, B, A)$, definita tramite tabella di verità, utilizzando la prima forma del teorema.

C	B	A	f	$f =$	$\bar{C} \bar{B} \bar{A} \cdot f(0, 0, 0) +$	$\bar{C} \bar{B} \bar{A} \cdot 0 +$	
0	0	0	0	\Rightarrow	$\bar{C} \bar{B} A \cdot f(0, 0, 1) +$	$\bar{C} \bar{B} A \cdot 1 +$	
0	0	1	1		$\bar{C} B \bar{A} \cdot f(0, 1, 0) +$	$\bar{C} B \bar{A} \cdot 0 +$	$\bar{C} \bar{B} A +$
0	1	0	0		$\bar{C} B A \cdot f(0, 1, 1) +$	$\bar{C} B A \cdot 0 +$	$C \bar{B} A +$
0	1	1	0		$C \bar{B} \bar{A} \cdot f(1, 0, 0) +$	$C \bar{B} \bar{A} \cdot 0 +$	$C B \bar{A}$
1	0	0	0		$C \bar{B} A \cdot f(1, 0, 1) +$	$C \bar{B} A \cdot 1 +$	
1	0	1	1		$C B \bar{A} \cdot f(1, 1, 0) +$	$C B \bar{A} \cdot 1 +$	
1	1	0	1		$C B A \cdot f(1, 1, 1)$	$C B A \cdot 0$	

Partiamo dalla definizione. Sostituiamo quindi tutti i termini costanti del tipo $f(\dots)$ con l'effettivo valore della funzione, preso direttamente dalla tabella di verità. Osserviamo che nella somma logica i termini in AND con 0 si possono omettere, in quanto sempre a 0; semplifichiamo poi i termini che restano, corrispondenti alle righe con 1 in uscita, eliminando il prodotto per la costante. L'espressione rimanente è quella cercata, ed esprime analiticamente, in *prima forma canonica*, il comportamento della funzione.

Seconda forma

La seconda forma consente la scomposizione della funzione f in un prodotto di somme:

$$f(X_1, X_2, X_3, \dots, X_n) = (X_1 + f(0, X_2, X_3, \dots, X_n)) \cdot (\bar{X}_1 + f(1, X_2, X_3, \dots, X_n))$$

Il primo termine di somma equivale alla funzione di partenza, dopo avere sostituito $X_1 = 0$, e vale nel caso in $X_1 = 0$ (altrimenti tutto il termine è a 1). Il secondo termine di somma equivale alla funzione di partenza, dopo avere sostituito $X_1 = 1$, e vale nel caso in $X_1 = 1$ (altrimenti tutto il termine è a 1). In altre parole, per un certo valore di X_1 , uno dei due termini vale sempre 1, mentre l'altro assume il valore della funzione.

Se iteriamo il procedimento di scomposizione, fino ad esaurire tutte le variabili argomento di f , otteniamo 2^n termini in AND tra di loro. Ogni termine risulta composto da una somma logica di tutte le variabili, dirette o negate, e del valore della funzione per quella particolare combinazione:

$$\begin{aligned} f(X_1, X_2, X_3, \dots, X_n) &= (X_1 + X_2 + f(0, 0, X_3, \dots, X_n)) \cdot \\ &\quad (X_1 + \bar{X}_2 + f(0, 1, X_3, \dots, X_n)) \cdot \\ &\quad (\bar{X}_1 + X_2 + f(1, 0, X_3, \dots, X_n)) \cdot \\ &\quad (\bar{X}_1 + \bar{X}_2 + f(1, 1, X_3, \dots, X_n)) = \\ &= \dots \end{aligned}$$

Consideriamo una generica funzione in tre variabili $f(C, B, A)$; si ottengono 2^3 termini OR, in AND tra loro:

$$\begin{aligned}
 f(C, B, A) = & (C + B + A + f(0, 0, 0)) \cdot \\
 & (C + B + \bar{A} + f(0, 0, 1)) \cdot \\
 & (C + \bar{B} + A + f(0, 1, 0)) \cdot \\
 & (C + \bar{B} + \bar{A} + f(0, 1, 1)) \cdot \\
 & (\bar{C} + B + A + f(1, 0, 0)) \cdot \\
 & (\bar{C} + B + \bar{A} + f(1, 0, 1)) \cdot \\
 & (\bar{C} + \bar{B} + A + f(1, 1, 0)) \cdot \\
 & (\bar{C} + \bar{B} + \bar{A} + f(1, 1, 1))
 \end{aligned}$$

Una funzione così espansa si dice che ha assunto la *seconda forma canonica*, o *prodotto di somme*, o anche forma OR-AND. Qualsiasi funzione booleana può essere espressa sotto tale forma; per una funzione di n variabili si ottengono 2^n fattori da moltiplicare.

Esempio

Tramite la seconda forma del teorema di espansione di Shannon, ricaviamo l'espressione analitica di una funzione booleana $f(C, B, A)$, di cui abbiamo la descrizione tramite tabella di verità:

C	B	A	f	$f = (C + B + A + f(0, 0, 0)) \cdot$	$f = (C + B + A + 1) \cdot$
0	0	0	1	$(C + B + \bar{A} + f(0, 0, 1)) \cdot$	$(C + B + \bar{A} + 1) \cdot$
0	0	1	1	$(C + \bar{B} + A + f(0, 1, 0)) \cdot$	$(C + \bar{B} + A + 0) \cdot$
0	1	0	0	$(C + \bar{B} + \bar{A} + f(0, 1, 1)) \cdot$	$(C + \bar{B} + \bar{A} + 0) \cdot$
0	1	1	0	$(\bar{C} + B + A + f(1, 0, 0)) \cdot$	$(\bar{C} + B + A + 0) \cdot$
1	0	0	0	$(\bar{C} + B + \bar{A} + f(1, 0, 1)) \cdot$	$(\bar{C} + B + \bar{A} + 1) \cdot$
1	0	1	1	$(\bar{C} + \bar{B} + A + f(1, 1, 0)) \cdot$	$(\bar{C} + \bar{B} + A + 1) \cdot$
1	1	0	1	$(\bar{C} + \bar{B} + \bar{A} + f(1, 1, 1)) \cdot$	$(\bar{C} + \bar{B} + \bar{A} + 1) \cdot$
1	1	1	1	$(\bar{C} + \bar{B} + \bar{A} + f(1, 1, 1)) \cdot$	$(\bar{C} + \bar{B} + \bar{A} + 1) \cdot$

Abbiamo applicato la definizione del teorema nella seconda forma, sostituendo tutti i termini costanti del tipo $f(\dots)$ con i valori contenuti nella tabella di verità data. Tenendo conto che in una somma logica il termine costante 1 assorbe le altre variabili, e che sommare 0 è ridondante, l'espressione finale della funzione è:

$$f = (C + \bar{B} + A) \cdot (C + \bar{B} + \bar{A}) \cdot (\bar{C} + B + A)$$

L'espressione esprime il comportamento della funzione nella *seconda forma canonica*.

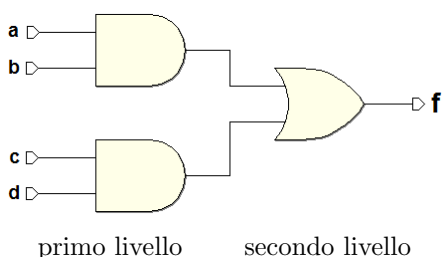
1.10 Livello di un'espressione booleana

Il livello è il massimo numero di operazioni eseguite in cascata sulle variabili d'ingresso. Ad esempio:

$f = a + b$ è una espressione ad un livello

$f = ab + c$ è una espressione a due livelli

$f = ab + cd$ è una espressione a due livelli



Osservazione: i livelli hanno un'importanza di tipo tecnico. Più livelli ci sono, maggiori sono i ritardi: noi ci occuperemo generalmente di sintesi di reti combinatorie a 2 livelli. Nel computo dei livelli, si suppongono disponibili le variabili d'ingresso negate: per questo motivo, l'espressione $f = \bar{a}b + \bar{c}d$ è considerata a due livelli.

1.11 Letterali

Sono il numero delle variabili d'ingresso che compaiono in un'espressione booleana (da non confondersi con il numero delle variabili).

Ad esempio: Sia $f(a, b)$ funzione logica delle sole variabili binarie a, b :

$f = a + b$ ha 2 letterali

$f = ab + \bar{a}b$ ha 4 letterali

1.12 Prodotti fondamentali

Se un termine AND di un'espressione booleana contiene, in forma diretta o negata, tutte le variabili presenti nell'espressione presa globalmente, questo termine si chiama *prodotto fondamentale* o *MINTERM*. Ad esempio:

$f(X_1, X_2, X_3) = X_1 \cdot X_2 \cdot \bar{X}_3$ è un prodotto fondamentale.

Una funzione ad n variabili ha 2^n prodotti fondamentali, in quanto ogni variabile della funzione può far parte di un prodotto fondamentale in forma diretta o negata. Si osservi che, tra tutte le possibili combinazioni delle variabili ne esiste una sola per la quale un certo prodotto fondamentale vale 1 (nell'esempio $X_1 \cdot X_2 \cdot \bar{X}_3 = 1$ se e solo se $X_1 = 1, X_2 = 1, X_3 = 0$).

1.13 Somme fondamentali

Se un termine OR di un'espressione booleana contiene, dirette o negate, tutte le variabili presenti globalmente nell'espressione, questo termine si chiama *somma fondamentale* o *MAXTERM*. Anche in questo caso, se ho n variabili, ho 2^n somme fondamentali. Ad esempio:

$$f(X_1, X_2, X_3) = X_1 + \overline{X_2} + X_3 \quad \text{è una somma fondamentale.}$$

Si tenga presente che esiste un'unica combinazione delle variabili per la quale una certa somma fondamentale vale 0 (nell'esempio $X_1 + \overline{X_2} + X_3 = 0$ se e solo se $X_1 = 0, X_2 = 1, X_3 = 0$).

1.14 Implicanti

Date f e g espressioni booleane si dice che g è un implicante di f o che g implica f ($g \Rightarrow f$) o che f copre g ($f \supset g$) se è sempre $f=1$ quando $g=1$.

$$\text{Nell'esempio: } f(X, Y, Z) = XY + Z \quad \text{si ha che} \quad \begin{array}{l} XY \Rightarrow f \\ Z \Rightarrow f \end{array}$$

Infatti ogni volta che Z e/o XY valgono 1 anche f vale 1. X non è implicante di f : infatti se X vale 1 non necessariamente anche f vale 1.

1.15 Implicanti primi

Si dice che g è implicante primo di f se:

- $g \Rightarrow f$ ($f \supset g$);
- g non è a sua volta coperto da un altro implicante con meno letterali

In altre parole, un implicante è primo se, quando vale 1, nessun altro implicante vale 1. Un implicante NON primo può essere eliminato dall'espressione in quanto è superfluo. Nell'esempio:

$$f = XY + X + Z$$

X e Z sono implicanti primi, mentre XY è implicante, ma non primo, di f . Infatti, condizione necessaria perché $XY = 1$ è che sia $X = 1$, ma se $X = 1$ allora $f = 1$ in quanto X è già implicante di f ($X \supset XY$).

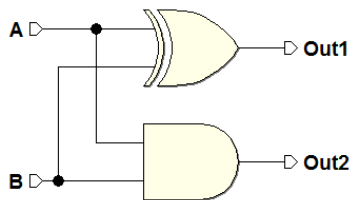
1.16 Reti combinatorie

Si definisce “rete combinatoria” quel circuito logico la cui uscita dipende soltanto dalla combinazione dei suoi ingressi. In seguito incontreremo le “reti sequenziali”, la cui uscita non dipende soltanto dai valori attuali degli ingressi, ma anche dalla loro “storia precedente”: vedremo cioè che tali reti possiedono capacità di memoria.

Una rete combinatoria è descrivibile in termini di funzioni booleane. Vediamo ora alcuni esempi di reti combinatorie.

1.16.1 Esempio: analisi di rete logica

Si vuole analizzare il circuito tramite ricavandone la tabella di verità:



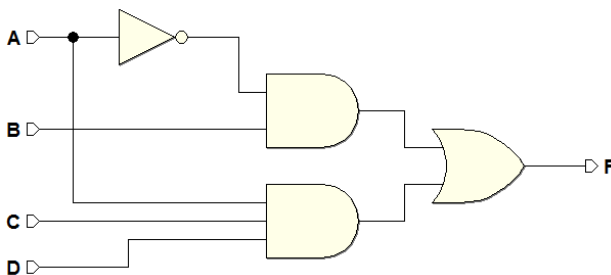
Si tratta di due porte logiche conosciute, possiamo compilare direttamente la tavola di verità:

A	B	Out1	Out2
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Ritroveremo questo circuito nel prossimo capitolo (è un circuito aritmetico).

1.16.2 Esempio: analisi di rete logica a due livelli

Anche in questo caso si vuole analizzare il circuito, ricavandone la tabella di verità:



Dobbiamo determinare, analizzando i percorsi del circuito, il valore dell'uscita F per tutte le combinazioni di A, B, C e D. Risulta utile includere nella tabella di verità anche le uscite intermedie. Alla fine dell'analisi risulta:

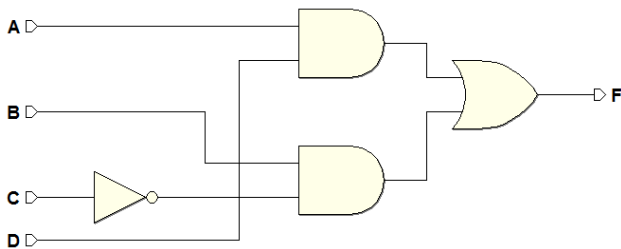
A	B	C	D	\bar{A}	$\bar{A}B$	ACD	F
0	0	0	0	1	0	0	0
0	0	0	1	1	0	0	0
0	0	1	0	1	0	0	0
0	0	1	1	1	0	0	0
0	1	0	0	1	1	0	1
0	1	0	1	1	1	0	1
0	1	1	0	1	1	0	1
0	1	1	1	1	1	0	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0
1	0	1	0	0	0	0	0
1	0	1	1	0	0	1	1
1	1	0	0	0	0	0	0
1	1	0	1	0	0	0	0
1	1	1	0	0	0	0	0
1	1	1	1	0	0	1	1

1.16.3 Esempio: schema circuitale di una rete logica (1)

Disegnare lo schema logico del circuito, data l'espressione booleana:

$$F = AD + \bar{C}B$$

Risulta:

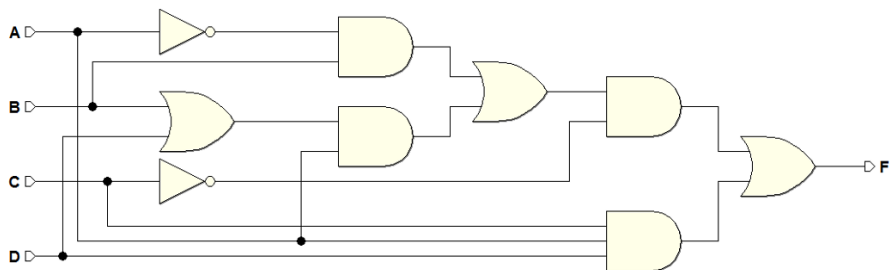


1.16.4 Esempio: schema circuitale di una rete logica (2)

Disegnare lo schema logico del circuito, data la sua espressione booleana:

$$F = ADC + \bar{C}(\bar{A}B + A(B + D))$$

Ne risulta una rete a cinque livelli:



Se, per esercizio, ricaviamo la tabella di verità di questo circuito e di quello del precedente esempio, ci si accorge che le due tavole di verità sono identiche! Le due reti risultano quindi tra di loro equivalenti (per induzione completa). Tuttavia, possiamo dimostrarne l'equivalenza anche tramite le proprietà dell'algebra booleana, nel modo seguente:

$$\begin{aligned}
 F &= ADC + \overline{C}(\overline{A}B + A(B + D)) = \\
 &= ADC + \overline{C}(\overline{A}B + AB + AD) = \\
 &= ADC + \overline{C}(B(\overline{A} + A) + AD) = \\
 &= ADC + \overline{C}(B + AD) = ADC + AD\overline{C} + \overline{C}B = \\
 &= AD(C + \overline{C}) + \overline{C}B = AD + \overline{C}B
 \end{aligned}$$

1.16.5 Esempio: definizione del comportamento di una rete logica

Ricaviamo la tabella di verità di una rete combinatoria avente tre ingressi A , B e C : l'uscita F deve assumere il valore 1 quando il numero di 1 in ingresso è dispari.

Dapprima prepariamo la tabella di verità (qui sotto a sinistra). Poi, riga per riga, contiamo gli 1 e scriviamo il valore di F sulla base della definizione data. Per esempio, nell'ultima riga ne contiamo tre, che è dispari, per cui inseriamo un 1 nella colonna dell'uscita. Infine, ne risulta la tabella a destra:

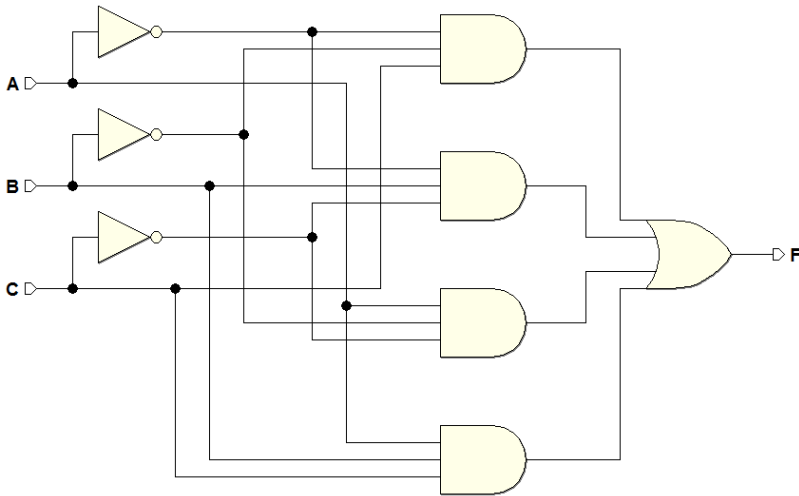
A	B	C	F		A	B	C	F
0	0	0	.		0	0	0	0
0	0	1	.		0	0	1	1
0	1	0	.		0	1	0	1
0	1	1	.	→	0	1	1	0
1	0	0	.		1	0	0	1
1	0	1	.		1	0	1	0
1	1	0	.		1	1	0	0
1	1	1	.		1	1	1	1

1.16.6 Esempio: schema circuitale da tabella di verità

Prendiamo in esame la tabella ricavata nell'esempio precedente e scriviamo dapprima l'espressione booleana della funzione, utilizzando la prima forma (AND-OR) del teorema di Shannon. In ultimo, ne disegniamo lo schema circuitale. Dalla tabella otteniamo:

$$F = \bar{A} \bar{B} C + \bar{A} B \bar{C} + A \bar{B} \bar{C} + A B C$$

L'espressione definisce quattro AND a 3 ingressi ciascuno, che confluiscono in un unico OR a 4 ingressi. Dopo avere disegnato e interconnesso questi componenti nello schema, andremo a collegare agli AND gli ingressi A , B e C , tenendo conto delle negazioni, quando presenti. Risulta infine:



1.16.7 Esempio: controllo di un impianto di riscaldamento

Progettare il circuito di controllo per un impianto di riscaldamento.



L'impianto è costituito da un termostato, da una caldaia e da un interruttore. T , I e C sono variabili booleane, le prime due di ingresso, l'ultima di uscita.

Definizione delle variabili

Detta t_0 una certa temperatura di soglia, il termostato indica se ci troviamo al di sopra o al di sotto di t_0 . Supponiamo quindi che:

$$T = 0 \text{ (se } t \geq t_0) \quad T = 1 \text{ (se } t < t_0)$$

La caldaia può essere accesa o spenta:

$$C = 0 \text{ (caldaia spenta)} \quad C = 1 \text{ (caldaia accesa)}$$

Lo stesso vale per l'interruttore che può essere ON/OFF:

$$I = 0 \text{ (OFF = escluso)} \quad I = 1 \text{ (ON = inserito)}$$

Definizione del funzionamento della rete

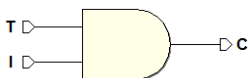
Perché la caldaia sia accesa ($C = 1$) è necessario che sia $t < t_0$ ($T = 1$) e che l'interruttore sia ON ($I = 1$).

Traduciamo questa "descrizione a parole" in una tabella di verità:

• se I è escluso la caldaia è spenta $\Rightarrow C = 0$;		I	T	C
• se I è inserito ma $t \geq t_0 \Rightarrow C = 0$;	\Rightarrow	0	0	0
• se I è inserito e $t < t_0 \Rightarrow C = 1$;		0	1	0
		1	0	0
		1	1	1

Sintesi

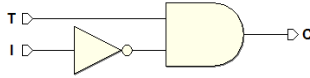
Si definisce **sintesi** il processo che permette di ottenere l'espressione logica (e quindi anche lo schema della rete) da una tavola di verità. L'espressione logica risultante è $C = I \cdot T$, e la rappresentazione circuitale è:



Nota: i valori assegnati alle tre variabili per rappresentare le diverse condizioni fisiche sono arbitrari. Ad esempio, avremmo potuto definire I e C come prima e T nel modo seguente:

$T = 0$ se $t < t_0$		I	T	C
$T = 1$ se $t \geq t_0$	da cui:	0	0	0
		0	1	0
		1	0	1
		1	1	0

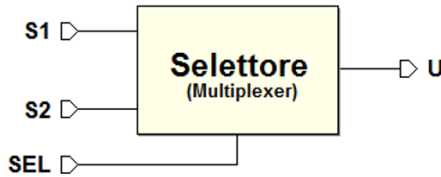
Si osservi che $C = 1$ se e solo se $I = 1$ e $T = 0$. In questo caso l'espressione logica è $C = I \cdot \bar{T}$. La nuova rappresentazione circuitale è:



Nelle due differenti casi, sia $C = I \cdot T$ che $C = I \cdot \bar{T}$ sono prodotti fondamentali. Esiste un'unica combinazione di I e T il cui prodotto dà $C = 1$. In generale, per sintetizzare una rete che ha un solo 1 in uscita, si prende il prodotto fondamentale dell'unica combinazione delle variabili che dà 1 in uscita, inserendo "dirette" le variabili che valgono 1 in quella particolare combinazione e "negate" le variabili che valgono 0.

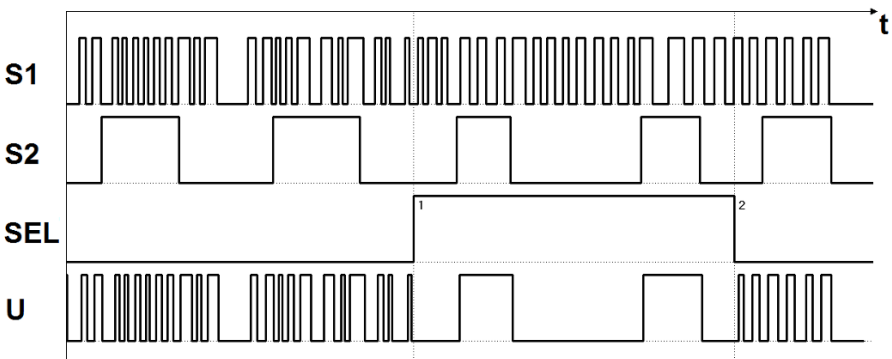
1.16.8 Esempio: selettore (multiplexer) a due canali

E' un sistema che fornisce in uscita una variabile booleana che ricopia uno dei due possibili ingressi a seconda del valore di una variabile di controllo (SEL).



Prima di definire il problema in termini booleani tracciamo i grafici di possibili $S1$, $S2$, SEL al variare del tempo e vediamo come deve varia U .

Rappresentiamo $S1$ e $S2$ come "segnali" digitali, cioè delle successioni di 0 e 1 che variano nel tempo, e codificano, secondo un certo standard, dell'informazione (per esempio una comunicazione telefonica, o una trasmissione televisiva). Senza curarci del tipo particolare di codifica, vogliamo soltanto selezionare, tramite SEL , quale dei segnali sarà instradato all'uscita U .



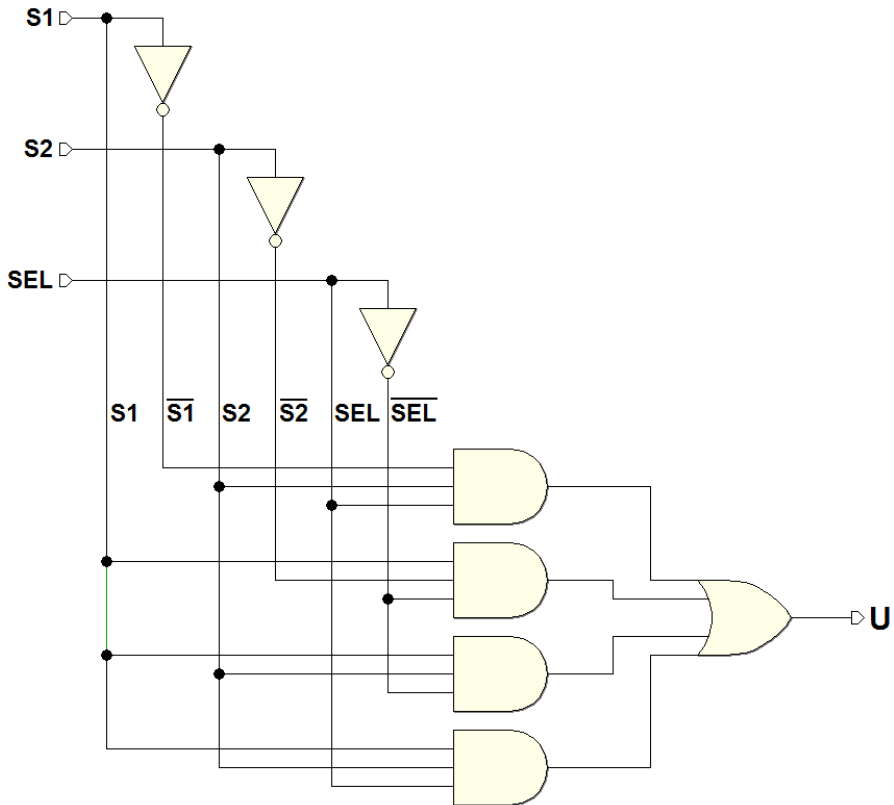
Come visibile nel tracciato, si vuole che U assuma i valori di $S1$ se $SEL = 0$, oppure di $S2$ se $SEL = 1$. È quindi possibile definire la tabella di verità. Si faccia attenzione che la tabella è valida "istante per istante", e che non descrive in ordine di tempo il comportamento della rete:

<i>SEL</i>	<i>S1</i>	<i>S2</i>	<i>U</i>
0	0	0	0
0	0	1	0
0	1	0	1 (a)
0	1	1	1 (b)
1	0	0	0
1	0	1	1 (c)
1	1	0	0
1	1	1	1 (d)

Applichiamo la prima forma del teorema di Shannon: eseguiamo, cioè, una sintesi canonica AND-OR. Scriviamo, per tutti gli 1 presenti nella colonna dell'uscita, i prodotti fondamentali corrispondenti, in OR tra loro:

$$\begin{aligned}
 U &= \overline{SEL} \cdot S1 \cdot \overline{S2} + & (a) \\
 &\overline{SEL} \cdot S1 \cdot S2 + & (b) \\
 &SEL \cdot \overline{S1} \cdot S2 + & (c) \\
 &SEL \cdot S1 \cdot S2 & (d)
 \end{aligned}$$

Questo lo schema circuitale, comprensivo dei NOT (ne bastano tre):



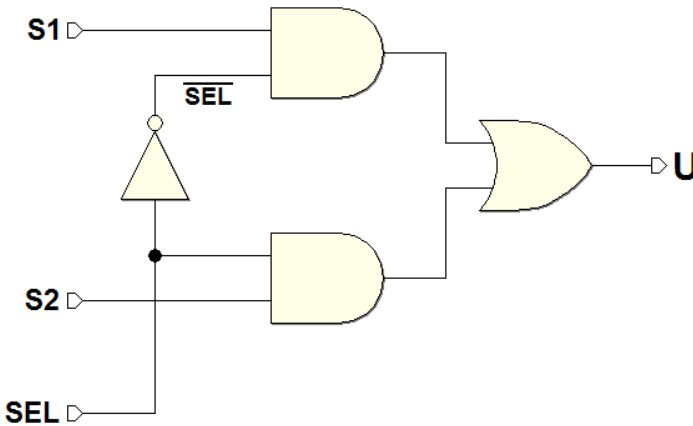
Osserviamo che ogni volta che si presenta una delle quattro combinazioni per cui in tabella abbiamo un 1, il corrispondente prodotto fondamentale (e solo quello) genera un 1. La OR in uscita fa sì che l'uscita U vada a 1 per tutte e quattro le combinazioni di cui sopra.

Minimizzazione

La forma canonica AND-OR che abbiamo ricavato può essere semplificata (minimizzata), utilizzando le proprietà dell'algebra booleana. Minimizzare significa ridurre al minimo il numero di letterali della espressione booleana che descrive la rete. Eseguiamo alcuni passaggi, raccogliendo opportunamente:

$$\begin{aligned} U &= \overline{SEL} \cdot S1 \cdot \overline{S2} + \overline{SEL} \cdot S1 \cdot S2 + SEL \cdot \overline{S1} \cdot S2 + SEL \cdot S1 \cdot S2 = \\ &= (S1 \cdot \overline{S2} + S1 \cdot S2) \cdot \overline{SEL} + (\overline{S1} \cdot S2 + S1 \cdot S2) \cdot SEL = \\ &= ((\overline{S2} + S2) \cdot S1) \cdot \overline{SEL} + ((\overline{S1} + S1) \cdot S2) \cdot SEL = \\ &= S1 \cdot \overline{SEL} + S2 \cdot SEL \end{aligned}$$

Si è tenuto conto che $(S1 + \overline{S1}) = 1$ e $(\overline{S2} + S2) = 1$. L'espressione aveva in origine 12 letterali: ora ne ha solo 4. Qui di seguito lo schema logico della rete risultante del selettore, la cui complessità circuitale si è notevolmente ridotta:



L'espressione finale:

$$U = S1 \cdot \overline{SEL} + S2 \cdot SEL$$

può essere anche re-interpretata in modo intuitivo: dice semplicemente che l'uscita U ricopia $S1$ nel caso in cui SEL sia 0 (e quindi $\overline{SEL} = 1$), mentre ricopia $S2$ nel caso in cui SEL sia 1.

Nota: il procedimento analitico di minimizzazione è stato svolto per esercizio: per ridurre la complessità di una funzione booleana possiamo usare metodi più sistematici, quali le “mappe”, che vedremo nel prossimo capitolo.

1.17 Esercizi

1. Negare il seguente termine, e quindi trasformarlo in una somma logica di quattro termini:

$$\overline{A} B \overline{C} D$$

2. Negare il seguente termine, e poi trasformarlo in un solo termine di prodotto:

$$\overline{A} + \overline{B} + C$$

3. Facendo uso dei teoremi dell'algebra booleana, minimizzare la seguente espressione logica:

$$A + AB + CB + C\overline{B}$$

4. Facendo uso dei teoremi dell'algebra booleana, dimostrare che l'espressione seguente vale 1 solo quando A e B sono contemporaneamente a 1 oppure quando D è a 1 e contemporaneamente C è a 0:

$$F = \overline{A} C B + \overline{A} C \overline{B} + \overline{A} \overline{D} + \overline{B} C D + \overline{B} C + \overline{B} \overline{D}$$

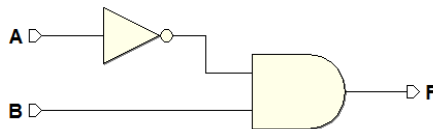
5. Minimizzare con i criteri dell'algebra booleana la seguente funzione logica:

$$Y = \overline{A}(A + B) + \overline{C} + BC$$

6. Minimizzare con i teoremi di De Morgan la seguente funzione logica:

$$Y = \overline{A + A\overline{B} + CD}$$

7. Disegnare il circuito che implementa l'espressione $\overline{A}B + A\overline{B}\overline{C}(B + C)$, e quindi verificare che è equivalente alla rete seguente:



8. Disegnare i circuiti corrispondenti alle seguenti espressioni logiche:

a) $C = (A + B) + \overline{A}B$

b) $D = A(B + C)\overline{C}$

c) $E = \overline{A}D\overline{(C + \overline{D})}$

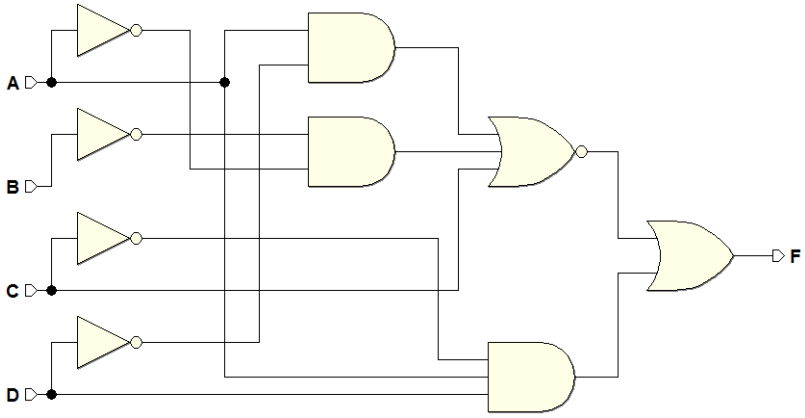
d) $G = \overline{A}B\overline{C} + D(C + A\overline{B}C)$

9. Eseguire le seguenti conversioni tra le forme canoniche:

a) $F = \overline{A}B\overline{C} + \overline{A}BC + A\overline{B}C + AB\overline{C}$ alla forma canonica OR-AND.

b) $G = (\overline{A} + B + C)(\overline{A} + \overline{B} + \overline{C})(A + B + C)$ alla forma canonica AND-OR.

10. Ricavare in forma analitica la funzione logica del seguente circuito:



1.18 Soluzioni

- $\overline{ABC\bar{D}} = \overline{A} + \overline{B} + \overline{C} + \overline{D} = A + \overline{B} + C + \overline{D}$
- $\overline{A + \overline{B} + C} = \overline{A}B\overline{C} = AB\overline{C}$
- $A + AB + CB + C\overline{B} = A(1 + B) + C(B + \overline{B}) = A + C$
- Occorre dimostrare che l'espressione data è equivalente a questa:

$$F = AB + \overline{C}D$$

Minimizzando l'espressione data si ottiene:

$$\begin{aligned} F &= \overline{\overline{ACB} + \overline{AC\overline{B}} + \overline{A\overline{D}} + \overline{BCD} + \overline{BC} + \overline{B\overline{D}}} = \\ &= \overline{\overline{AC}(B + \overline{B}) + \overline{A\overline{D}} + \overline{BC}(D + 1) + \overline{B\overline{D}}} = \\ &= \overline{\overline{AC} + \overline{A\overline{D}} + \overline{BC} + \overline{B\overline{D}}} = \\ &= \overline{\overline{A}(C + \overline{D}) + \overline{B}(C + \overline{D})} = \\ &= \overline{(\overline{A} + \overline{B}) \cdot (C + \overline{D})} = \\ &= \overline{(\overline{A} + \overline{B})} + \overline{(C + \overline{D})} = AB + \overline{C}D \end{aligned}$$

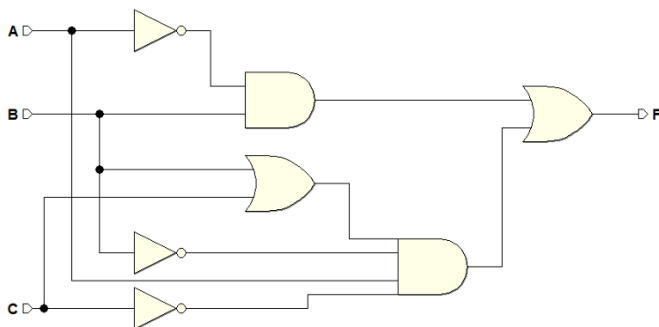
- Si ottiene:

$$\begin{aligned} Y &= \overline{A}(A + B) + \overline{C} + BC = \\ &= \overline{A}A + \overline{A}B + \overline{C} + BC = \overline{A}B + \overline{C} + B\overline{C} + BC = \\ &= \overline{A}B + \overline{C} + B(\overline{C} + C) = \overline{A}B + \overline{C} + B = \\ &= B + \overline{C} \end{aligned}$$

- Otteniamo:

$$Y = \overline{A + \overline{A\overline{B}} + \overline{CD}} = \overline{A(1 + \overline{B}) + \overline{CD}} = \overline{A + \overline{CD}} = \overline{A}(\overline{C} + \overline{D})$$

- La rete è la seguente:

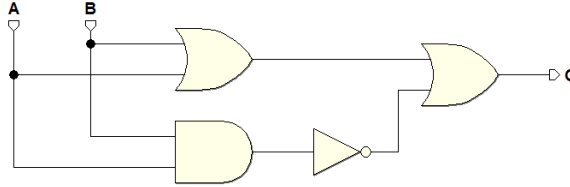


Minimizzando l'espressione di F :

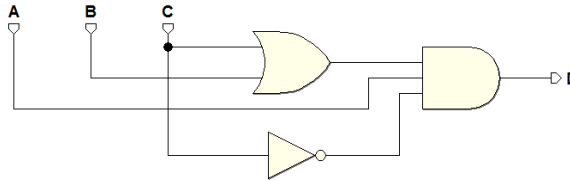
$$\begin{aligned}
 F &= \bar{A}B + A\bar{B}\bar{C}(B+C) = \bar{A}B + A\bar{B}\bar{C}B + A\bar{B}\bar{C}C = \\
 &= \bar{A}B + A\bar{C}(\bar{B}B) + A\bar{B}(\bar{C}C) = \bar{A}B + A\bar{C}(0) + A\bar{B}(0) = \bar{A}B
 \end{aligned}$$

8. Questi i circuiti corrispondente alle espressioni:

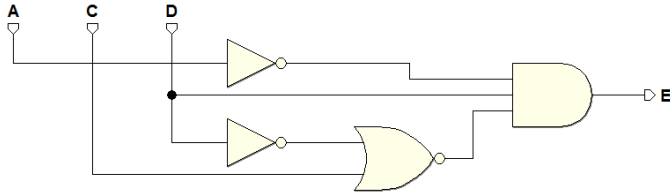
a) $C = (A + B) + \bar{A}\bar{B}$:



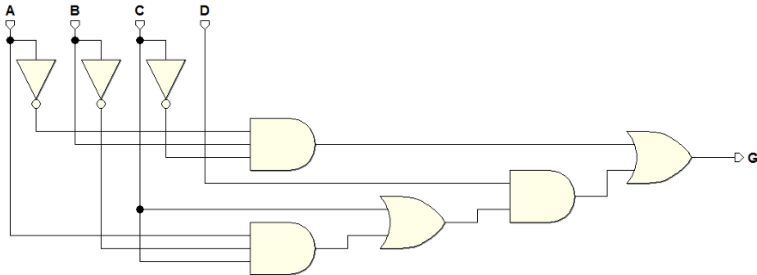
b) $D = A(B + C)\bar{C}$:



c) $E = \bar{A}D\overline{(C + \bar{D})}$:



d) $G = \bar{A}\bar{B}\bar{C} + D(C + A\bar{B}C)$:



9. a) Ricaviamo la tabella di verità, rappresentando anche l'uscita negata:

A	B	C	F	\overline{F}
0	0	0	0	1
0	0	1	0	1
0	1	0	1	0
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

Da questa ricaviamo l'espressione AND-OR della funzione negata, e applichiamo il teorema di De Morgan:

$$\begin{aligned} \overline{F} &= \overline{A} \overline{B} \overline{C} + \overline{A} \overline{B} C + A \overline{B} C + A \overline{B} \overline{C} \\ &= \overline{(A + B + C)(A + B + \overline{C})(\overline{A} + \overline{B} + \overline{C})(\overline{A} + B + C)} = \end{aligned}$$

Infine, neghiamo nuovamente tutta l'espressione:

$$F = (A + B + C)(A + B + \overline{C})(\overline{A} + \overline{B} + \overline{C})(\overline{A} + B + C)$$

- b) Compiliamo la tabella di verità, dalla quale poi ricaviamo direttamente la sintesi canonica AND-OR:

A	B	C	G
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

$$G = \overline{A} \overline{B} C + \overline{A} B \overline{C} + \overline{A} B C + A \overline{B} C + A B \overline{C}$$

10. $F = A \overline{C} D + \overline{(A \overline{D} + \overline{A} \overline{B} + C)}$

Progetto di reti combinatorie

2.1 Mappe di Karnaugh

Nel capitolo precedente, per descrivere una rete logica abbiamo utilizzato due “linguaggi”: le **espressioni booleane** e le **tabelle di verità**.

In questo paragrafo vedremo un terzo linguaggio, le **mappe**. La descrizione di una funzione mediante la sua espressione booleana o la sua tabella di verità hanno il difetto di rendere piuttosto laboriosa la loro semplificazione logica.

Le mappe sono un modo particolare di scrivere le tabelle di verità, in un formato che rende più agevole la semplificazione; come vedremo, le mappe ordinano e mettono in evidenza i prodotti fondamentali grazie ad una struttura geometrica che rende ovvia l'applicazione del teorema dell'assorbimento e dell'adiacenza logica.

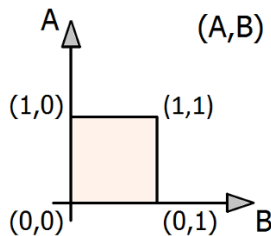
Assorbimento

$$\begin{array}{ll} A + AB = A & A(A + B) = A \\ A + \bar{A}B = A + B & A(\bar{A} + B) = AB \end{array}$$

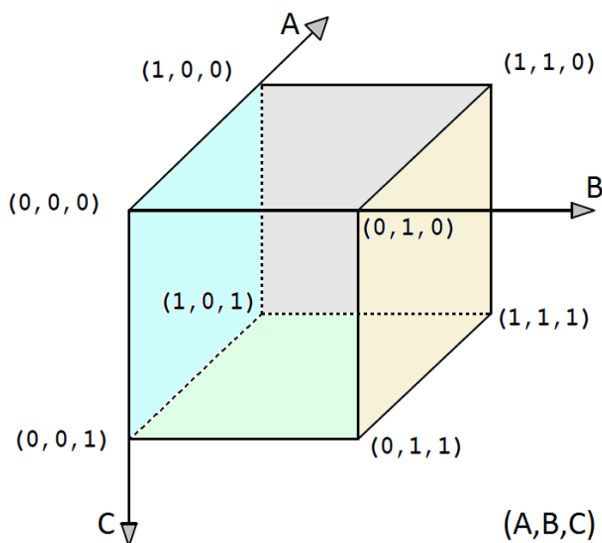
Adiacenza logica

$$AB + A\bar{B} = A \quad (A + B)(A + \bar{B}) = A$$

Data una funzione di due variabili $f(A, B)$, rappresentiamo in un piano il suo insieme di definizione, costituito dalle quattro combinazioni (A, B) :



Per una funzione a tre variabili $f(A, B, C)$, l'insieme di definizione può essere rappresentato in uno spazio a tre dimensioni, ponendo ciascuna delle otto possibili combinazioni di A, B, C ai vertici di un cubo:



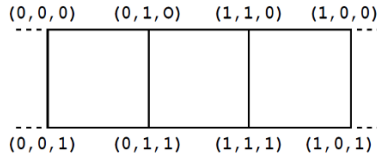
Osserviamo che, passando da un vertice ad un altro geometricamente adiacente (percorrendo un lato), cambia il valore di una sola variabile.

Si parla di “adiacenza logica” tra due combinazioni di variabili quando sono a *distanza 1*, ossia se differiscono solo per il valore una variabile.

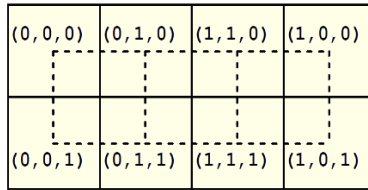
Il cubo che consideriamo è di ordine 3. All'interno di esso si individuano svariati cubi di ordine inferiore (“sottocubi”). L'ordine equivale alla dimensione geometrica: un quadrato è un cubo “di ordine 2” (a due dimensioni); un cubo di ordine n avrà 2^n vertici. Nel nostro caso osserviamo:

- 8 vertici (*cubi di ordine 0*): ogni vertice è caratterizzato da una unica combinazione di tutte le variabili, per esempio il vertice $(1,1,1)$;
- 12 lati (*cubi di ordine 1*): ogni lato raggruppa due combinazioni di variabili a distanza 1, ovvero, ogni lato è univocamente individuato da una unica coppia di valori di due delle tre variabili; ad esempio, il lato in alto a sinistra nella figura, che è definito da $B=0$ e $C=1$;
- 6 facce (*cubi di ordine 2*): ogni faccia raggruppa quattro differenti combinazioni di variabili (a due a due adiacenti) e, per questo, è univocamente individuata dal valore di un'unica variabile; per esempio la faccia più a destra in figura, definito da $B=1$.

Terminate queste considerazioni preliminari tracciamo la **mappa** corrispondente ad un cubo a tre dimensioni. Per fare ciò “ritagliamo” il nostro cubo e disponiamo su di un piano gli otto vertici del cubo, mantenendo (per quanto possibile) la stessa disposizione che avevano nello spazio:



La mappa è un reticolo di caselle, e ogni casella racchiude uno dei vertici:

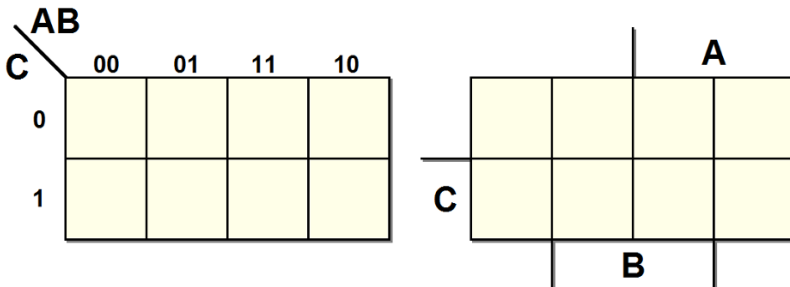


Osserviamo quanto segue:

- le 4 caselle inferiori hanno $C = 1$
- le 4 caselle superiori hanno $C = 0$
- le 4 caselle di destra hanno $A = 1$
- le 4 caselle di sinistra hanno $A = 0$
- le 4 caselle centrali hanno $B = 1$
- le 4 caselle laterali hanno $B = 0$

Possiamo rappresentare quindi le aree della mappa usando le variabili come “coordinate” delle caselle. Un primo metodo è mostrato nella figura qui sotto, a sinistra, dove i valori delle variabili identificano le righe e le colonne.

Il metodo mostrato sulla destra, invece, divide la mappa in aree geometriche corrispondenti al valore delle variabili: nel libro utilizzeremo questo metodo. Il nome di ogni variabile è disegnato in corrispondenza dell’area in cui vale 1:



Dato che una casella corrisponde univocamente ad una combinazione di valori delle variabili, in essa è possibile trascrivere il corrispondente valore assunto da una funzione $f(C, B, A)$ per quella determinata combinazione. Mappe di questo tipo sono dette *Mappe di Karnaugh* (o *K-map*).

2.2 Sintesi AND-OR con uso delle mappe

Nel capitolo precedente abbiamo esaminato un selettore, di cui rivediamo qui la tabella di verità:

<i>SEL</i>	<i>S1</i>	<i>S2</i>	<i>U</i>
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Trascriviamo i valori della funzione in una mappa di Karnaugh:

		<i>S1</i>	
		0	1
<i>SEL</i>	0	0	1
	1	1	0

L'espressione della forma canonica AND-OR che avevamo trovato era:

$$U = \overline{S1} \cdot S2 \cdot SEL + S1 \cdot \overline{S2} \cdot \overline{SEL} + S1 \cdot S2 \cdot \overline{SEL} + S1 \cdot S2 \cdot SEL$$

Nella figura seguente sono messi in evidenza i quattro prodotti fondamentali, e la loro posizione nella mappa:

		<i>S1</i>	
		0	1
<i>SEL</i>	0	0	1
	1	1	0

$\overline{S1} \cdot S2 \cdot SEL$ (casella SEL=1, S1=0, S2=1)
 $S1 \cdot \overline{S2} \cdot \overline{SEL}$ (casella SEL=0, S1=1, S2=0)
 $S1 \cdot S2 \cdot \overline{SEL}$ (casella SEL=0, S1=1, S2=1)
 $S1 \cdot S2 \cdot SEL$ (casella SEL=1, S1=1, S2=1)

Si ricorda che le caselle sono adiacenti se distano 1, cioè se hanno una sola variabile diversa. Nella mappa posso raccogliere gli 1 che si trovano a distanza unitaria in due sottocubi (o “raggruppamenti”) di dimensione 1:

		S1	
	0	0	1
SEL	0	1	1
		1	0
		S2	

In ciascun raggruppamento abbiamo una variabile il cui valore cambia al suo interno, mentre i valori delle altre no. Scriviamo quindi, per ogni raggruppamento, un termine contenente solo le variabili che in esso non variano, ignorando le altre.

Dal raggruppamento in alto a destra otteniamo $S1 \cdot \overline{SEL}$: siamo infatti nell'area in cui $S1 = 1$ e $SEL = 0$, mentre $S2$ assume valori differenti.

Analogamente dal raggruppamento centrale si ha $S2 \cdot SEL$: siamo nell'area in cui $S2 = 1$ e $SEL = 1$, mentre $S1$ varia.

In definitiva, sommando i termini ottenuti, ricaviamo rapidamente e direttamente dalla mappa l'espressione minimizzata:

$$U = S1 \cdot \overline{SEL} + S2 \cdot SEL$$

Nel capitolo precedente avevamo già ottenuto la stessa espressione, mediante le proprietà dell'algebra booleana, minimizzando quella in forma canonica.

Dimostrazione

Nel raggruppamento in alto a destra i due prodotti fondamentali sono:

$$S1 \cdot \overline{S2} \cdot \overline{SEL} \quad \text{e} \quad S1 \cdot S2 \cdot \overline{SEL}$$

In OR tra di loro, questi due termini si possono semplificare:

$$\begin{aligned} & S1 \cdot \overline{S2} \cdot \overline{SEL} + S1 \cdot S2 \cdot \overline{SEL} = \\ & S1 \cdot \overline{SEL} \cdot (S2 + \overline{S2}) = \\ & S1 \cdot \overline{SEL} \end{aligned}$$

La variabile eliminata, $S2$, è proprio quella che assume valori diversi all'interno del raggruppamento. Analoghe considerazioni vanno eseguite per l'altro: qui la variabile che cambia è $S1$, per cui si dimostra:

$$\begin{aligned} & S1 \cdot S2 \cdot SEL + \overline{S1} \cdot S2 \cdot SEL = \\ & S2 \cdot SEL \cdot (S1 + \overline{S1}) = \\ & S2 \cdot SEL \end{aligned}$$

Da cui la funzione $U = S1 \cdot \overline{SEL} + S2 \cdot SEL$, come definita sopra.

2.2.1 Implicanti e implicanti primi nelle mappe

Consideriamo la seguente mappa:

		A		
	0	0	1	1
C	0	0	1	1
		B		

Se facciamo riferimento ai due sottocubi di dimensione 1, ricaviamo:

$$U = AB + A\bar{B}$$

Ciascuno di essi è un implicante poichè se $A\bar{B}$ o AB valgono 1, anche la funzione U vale 1. Proviamo ora a considerare un unico sottocubo di dimensione 2, che comprenda tutti gli 1, come nella figura seguente:

		A		
	0	0	1	1
C	0	0	1	1
		B		

Nel sottocubo l'unica variabile che non varia è A : la funzione si riduce così alla espressione $U = A$. Potevamo arrivare a questa conclusione anche così:

$$U = A\bar{B} + AB = A(\bar{B} + B) = A$$

La semplificazione è possibile perchè $\bar{A}B$ e AB non sono implicanti primi, mentre A lo è. Per ottenere il miglior raggruppamento, occorre individuare i sottocubi più grandi possibili. Se *non esiste* un sottocubo più grande che includa interamente quello che stiamo considerando, allora da questo ricaviamo un implicante primo.

Vediamo ora alcuni esempi su come ricavare l'espressione della funzione dalle mappe. Dalla mappa qui a sinistra ricaviamo $U = A\bar{B}\bar{C}$, mentre da quella a destra si ottiene $U = \bar{B}\bar{C}$:

		A		
	0	0	0	1
C	0	0	0	0
		B		

		A		
	1	0	0	1
C	0	0	0	0
		B		

Nella mappa sulla destra, nonostante le apparenze, le due caselle sono tra loro logicamente adiacenti. Infatti, come si è visto, la mappa è una rappresentazione distesa sul piano di un cubo a più dimensioni. Dobbiamo quindi immaginare la mappa come un foglio di carta che si può ripiegare su se stesso, sia in senso orizzontale, sia in senso verticale: il bordo superiore e quello inferiore sono adiacenti, e così il bordo destro e sinistro. A riprova di questo, si noti che tra le due caselle considerate cambia soltanto la variabile A .

Si consideri ora la mappa seguente:

		A		
	1	0	0	1
C	1	0	0	1
	B			

È evidenziato il sottocubo più grande che contiene gli uno: risulta $U = \bar{B}$.

Vediamo ora due casi limite. La mappa sulla sinistra contiene un 1 in ogni cella, e rappresenta quindi la funzione costante $U = 1$; analogamente, la mappa sulla destra, contenente solo degli 0, si traduce nella costante $U = 0$:

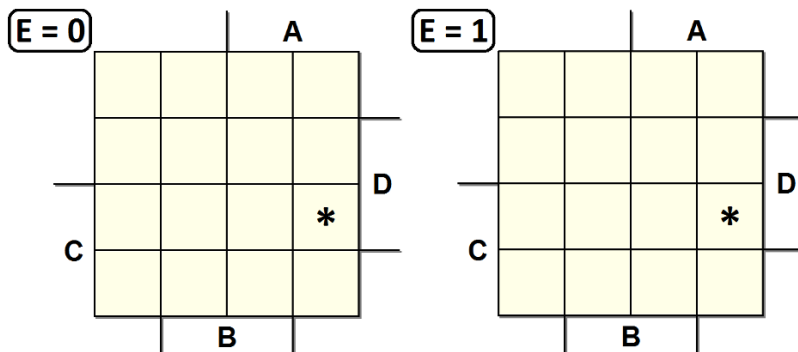
	A			
	1	1	1	1
C	1	1	1	1
	B			

	A			
	0	0	0	0
C	0	0	0	0
	B			

Abbiamo visto che in una mappa a 3 variabili tutti i prodotti fondamentali che differiscono per una sola variabile sono rappresentati sulle mappe da caselle adiacenti (con il caso particolare dei bordi). Questo vale anche per mappe a 4 variabili, come nell'esempio seguente, da cui ricaviamo: $U = \bar{A} B \bar{D} + \bar{B} C D$.

		A		
	0	1	0	0
	0	0	0	0
	1	0	0	1
C	0	1	0	0
		B		
				D

Le mappe a 5 variabili (di ordine 5) si possono rappresentare con due mappe di ordine 4, da immaginare una sopra all'altra, poste a distanza unitaria:

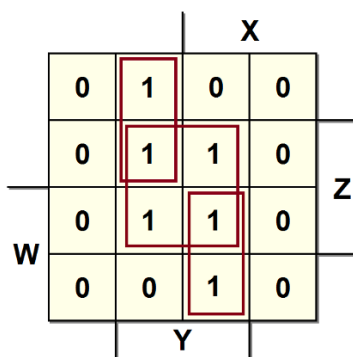


Le celle indicate dall'asterisco si trovano a distanza unitaria, e pertanto è possibile procedere al raccoglimento.

Le mappe a 6 variabili si realizzano utilizzando 4 mappe di ordine 4. Mappe di ordine maggiore non si possono rappresentare in modo semplice.

2.2.2 Minimizzazione con l'uso delle mappe

Definiamo *implicante primo essenziale* un implicante primo che è l'unico a contenere un certo 1 sulla mappa. Ad esempio nella mappa seguente sono indicati tre implicanti primi essenziali, due di ordine 1 e uno di ordine 2:



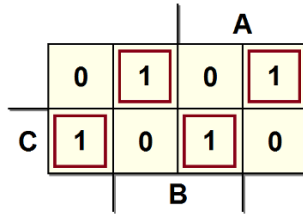
Le caselle con gli 1 si possono considerare più volte. Questo è utile per trovare sottocubi di ordine più grande. Dalla mappa precedente si ottiene la sintesi:

$$U = YZ + \bar{X}Y\bar{W} + XYW$$

Invece, un implicante già completamente coperto da implicanti primi essenziali si dice *implicante ridondante*. Gli implicanti ridondanti, come tali, devono essere eliminati dalla sintesi. In definitiva, usando le mappe, la sintesi minima si ottiene prendendo i soli implicanti primi essenziali.

2.2.3 Mappe “a scacchiera”

Una mappa “a scacchiera”, come nella figura seguente, non si può minimizzare in termini di rete AND-OR o OR-AND a due livelli:



Tuttavia si può verificare che corrisponde ad un albero di EXOR:

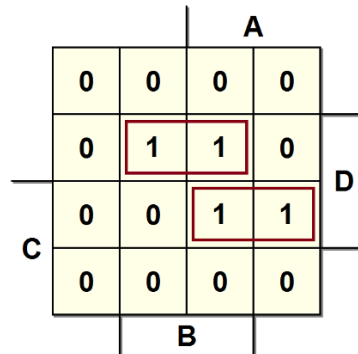
$$\begin{aligned}
 U &= \bar{A} B \bar{C} + A \bar{B} \bar{C} + \bar{A} \bar{B} C + A B C = \\
 &= (\bar{A} B + A \bar{B}) \cdot \bar{C} + (\bar{A} \bar{B} + A B) \cdot C = \\
 &= (A \oplus B) \cdot \bar{C} + \overline{(A \oplus B)} \cdot C = \\
 &= A \oplus B \oplus C
 \end{aligned}$$

2.2.4 Esempi di sintesi AND-OR

1. Consideriamo la tabella di verità qui sotto; compiliamo la mappa sulla destra:

A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

⇒



Il raggruppamento nella terza riga fornisce il termine ACD , quello nella seconda $B\bar{C}D$, da cui: $F = ACD + B\bar{C}D$. Facciamo una verifica:

$$\begin{aligned}
 ABCD + A\bar{B}CD &= ACD(B + \bar{B}) = ACD \\
 \bar{A}B\bar{C}D + AB\bar{C}D &= \bar{B}C\bar{D}(A + \bar{A}) = \bar{B}C\bar{D}
 \end{aligned}$$

2. Semplificare $F = \bar{A}\bar{B}\bar{C} + AB\bar{C} + \bar{A}\bar{B}C$. Ricaviamo la mappa:

		A			
		1	0	1	0
C		1	0	0	0
		B			

Dal gruppo dei due 1 a sinistra si ha: $\bar{A}\bar{B}$. La funzione complessiva è:

$$F = \bar{A}\bar{B} + AB\bar{C}.$$

3. Sintetizziamo la seguente mappa:

		A			
		0	0	0	0
	1	0	0	1	
C	1	0	0	1	D
		0	0	0	0
		B			

I quattro 1 (adiacenti, anche se su bordi opposti) possono essere raggruppati in un solo prodotto. Ne risulta la funzione:

$$F = \bar{B}D.$$

4. Analogamente, sintetizziamo un'altra mappa con degli 1 posti sugli angoli:

		A			
	1	0	0	1	
		0	0	0	0
C	1	0	0	1	D
		0	0	0	0
		B			

Anche qui raccogliamo un solo termine. La funzione è:

$$F = \bar{B}\bar{D}.$$

5. Ricaviamo la funzione dalla mappa qui sotto:

		A		
	1	0	0	1
	1	1	1	1
	0	0	1	1
C	0	0	1	1
		B		

I raggruppamenti sono tutti da 4 bit. Il gruppo in alto fornisce il termine $\overline{B}\overline{C}$; in seconda riga abbiamo $\overline{C}D$; in basso a destra ci ottiene AC . Quindi, la funzione cercata è:

$$F = \overline{B}\overline{C} + \overline{C}D + AC.$$

6. Facciamo ancora un esempio (*mappa ciclica*):

		X		
	0	0	1	1
	0	1	1	0
	1	1	0	0
W	1	0	0	1
		Y		

Si hanno quattro implicantti essenziali di ordine 1. Si avrà quindi:

$$U = X\overline{Z}\overline{W} + Y\overline{W}Z + \overline{X}ZW + \overline{Y}\overline{Z}W$$

2.3 Sintesi OR-AND

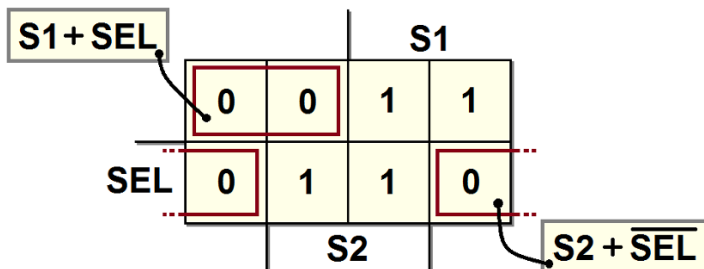
Abbiamo visto che in una sintesi AND-OR si sintetizzano gli 1 della funzione mediante l'AND delle variabili di ingresso, dirette se la variabile ha valore 1, negate se ha valore 0; si effettua poi l'OR di tutte le porte AND.

Invece, nella sintesi OR-AND, in accordo con la seconda forma del teorema di Shannon, si sintetizzano gli 0 della funzione mediante l'OR delle variabili di ingresso (dirette se 0, negate se 1) e tutte le OR finiscono in una AND.

Per minimizzare una sintesi OR-AND usando le mappe di Karnaugh si osservi che valgono sempre le proprietà dell'adiacenza logica. A titolo di esempio, ripetiamo qui la sintesi del selettore a due ingressi, ma con la tecnica OR-AND. Ricordiamo la tabella di verità:

<i>SEL</i>	<i>S1</i>	<i>S2</i>	<i>U</i>
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Dopo avere compilato la mappa con i valori corrispondenti alla tabella, individuiamo i sottocubi che raggruppano gli 0 (le regole sono le stesse già viste per i sottocubi con gli 1). Nella mappa seguente vediamo il miglior raggruppamento possibile per il nostro esempio:



Da cui ricaviamo direttamente l'espressione OR-AND minimizzata:

$$U = (S1 + SEL)(S2 + \overline{SEL}).$$

2.3.1 Sintesi della funzione negata

Per effettuare una sintesi OR-AND si può procedere anche in un altro modo, eseguendo dapprima la sintesi AND-OR degli 0 come se fossero 1 (cioè si considera la funzione negata). Applicando questo metodo all'esempio ora visto, si ottiene la seguente sintesi AND-OR di \overline{U} :

$$\overline{U} = \overline{S1} \cdot \overline{SEL} + \overline{S2} \cdot SEL$$

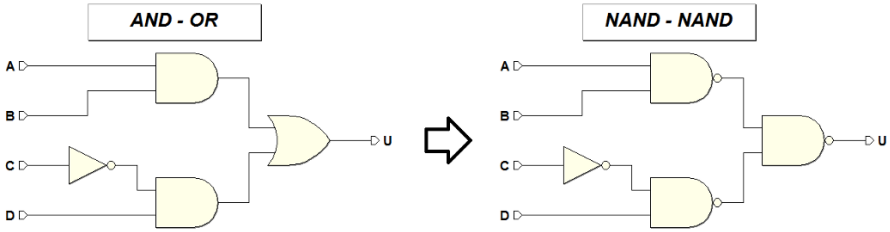
Applicando il teorema di De Morgan ottengo la cercata espressione OR-AND:

$$U = (S1 + SEL)(S2 + \overline{SEL})$$

Si noti che non è possibile ottenere la sintesi OR-AND applicando il principio di dualità direttamente all'espressione AND-OR.

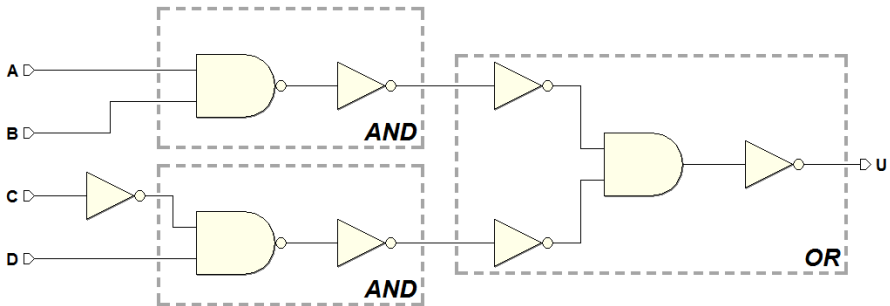
2.4 Sintesi NAND-NAND

Ottenuta una sintesi AND-OR, possiamo modificarla in una rete composta di soli NAND (sintesi NAND-NAND) sostituendo un NAND ad ogni OR o AND, come nell'esempio in figura:

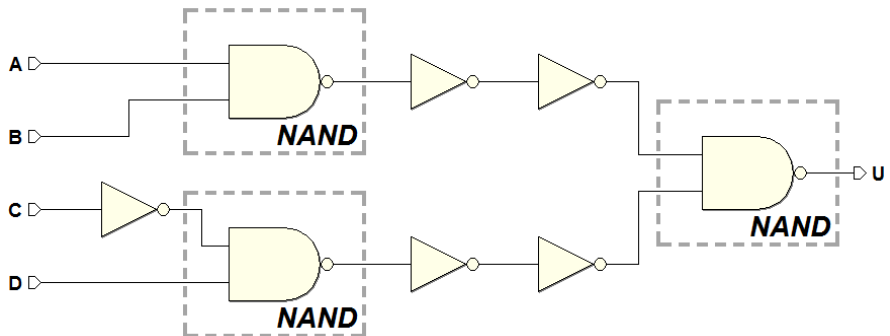


Dimostrazione:

Sostituiamo gli AND sulla sinistra con dei NAND seguiti da NOT. Appliciamo poi il Teorema di De Morgan alla porta OR, trasformandola in AND, preceduta e seguita da NOT:



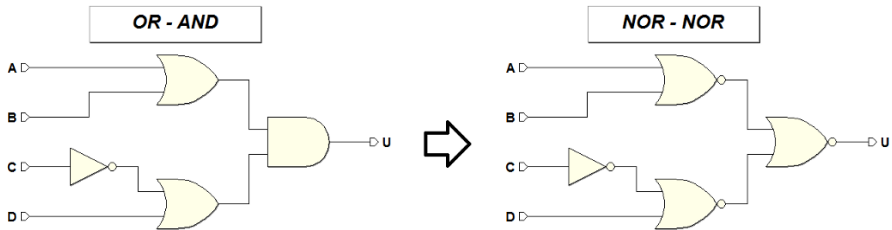
Accorpriamo la porta AND con il NOT che lo segue, ottenendo un NAND, e separiamo dai NOT i due NAND sulla sinistra:



Infine, eliminando la doppia negazione, si ottiene la rete NAND-NAND.

2.5 Sintesi NOR-NOR

Partendo da una rete OR-AND, sostituisco ogni OR o AND con una porta NOR, ottenendo una sintesi NOR-NOR, come nell'esempio seguente (la dimostrazione è analoga a quella della rete NAND-NAND):

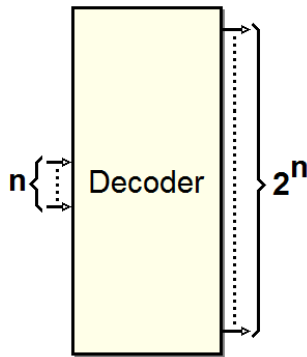


2.6 Reti combinatorie standard

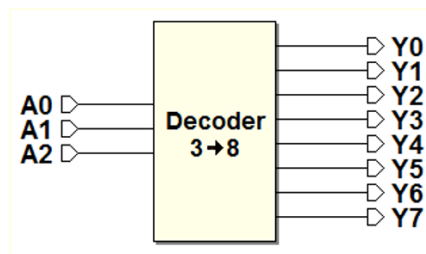
Si tratta di reti combinatorie di impiego generale, caratterizzate da una struttura circuitale regolare, disponibili al progettista come blocchi funzionali.

2.6.1 Decodificatore (Decoder)

Il decodificatore (“decoder”) dispone di n ingressi e di 2^n uscite: ogni combinazione degli ingressi attiva una ed una sola uscita.



Un decodificatore $3 \rightarrow 8$ è dotato di 3 ingressi e di 8 uscite:

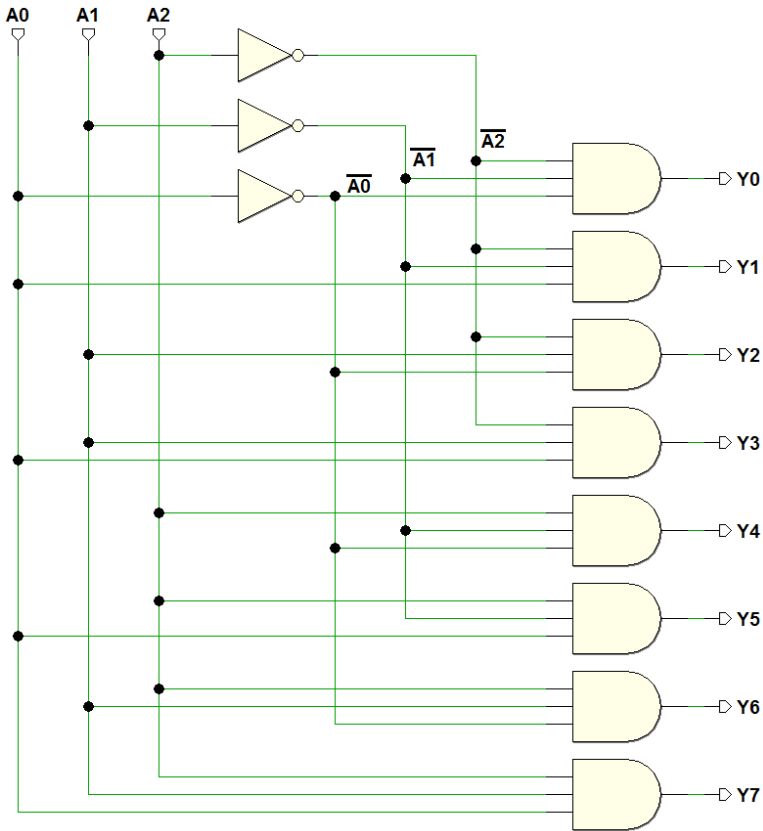


L'attivazione delle uscite avviene, come si vede dalla tabella di verità qui sotto, nell'ordine. La combinazione $A_2A_1A_0 = "000"$ abilita l'uscita Y_0 e così via fino alla $A_2A_1A_0 = "111"$ che attiva l'uscita Y_7 :

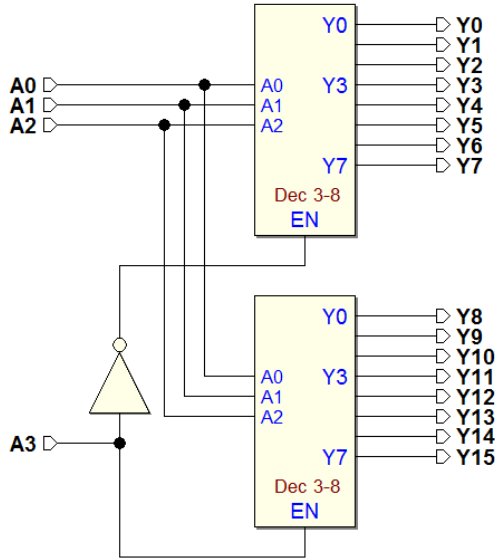
A_2	A_1	A_0	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

La sintesi di una tavola di verità con 8 uscite richiede l'impiego di una rete combinatoria per ogni uscita. In questo caso, ogni uscita si attiva per una sola combinazione di ingressi: non è quindi utile ricorrere alle mappe perché ogni uscita richiede un solo prodotto fondamentale.

Nella figura seguente, la sintesi del decodificatore $3 \rightarrow 8$:



Più decodificatori possono essere connessi in modo da formarne uno più grande: ad esempio, con due “Dec 3-8” otteniamo un decodificatore 4 → 16:



Osservando la tavola di verità (qui sotto), si nota che può essere divisa in due parti: la superiore in cui l’ingresso $A3 = 0$ e l’inferiore in cui $A3 = 1$. Il “Dec 3-8” che genera le prime 8 uscite deve essere abilitato ($EN = 1$) quando $A3 = 0$, l’altro quando $A3 = 1$. Ciò è ottenuto tramite una semplice NOT, come si vede nello schema.

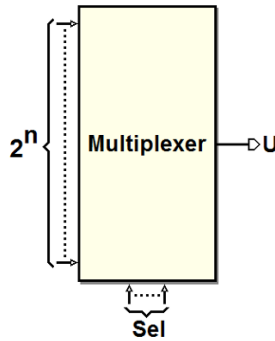
A3	A2	A1	A0	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7	Y8	Y9	Y10	Y11	Y12	Y13	Y14	Y15
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Questa tecnica può essere estesa a piacere, per ottenere decodificatori di dimensione maggiore di quelli disponibili.

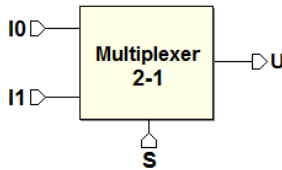
Il decodificatore è molto importante poiché molto usato. Ad esempio, rappresenta il componente fondamentale per ottenere “l’indirizzamento” di dispositivi all’interno di un sistema digitale, cioè la funzione che permette di identificare e/o selezionare, tramite un numero identificatore, gli elementi del sistema e di operare su di essi, come ad esempio avviene nei microcalcolatori.

2.6.2 Selettore (multiplexer)

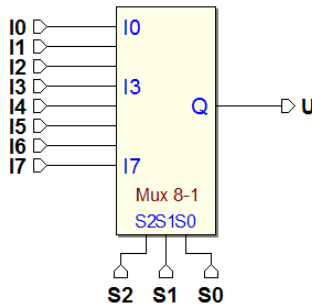
Il selettore (“multiplexer”, o “mux”) è una rete combinatoria dotata di 2^n ingressi di dato, una sola uscita e n ingressi di selezione. L’uscita assume il valore dell’ingresso identificato dagli ingressi di selezione.



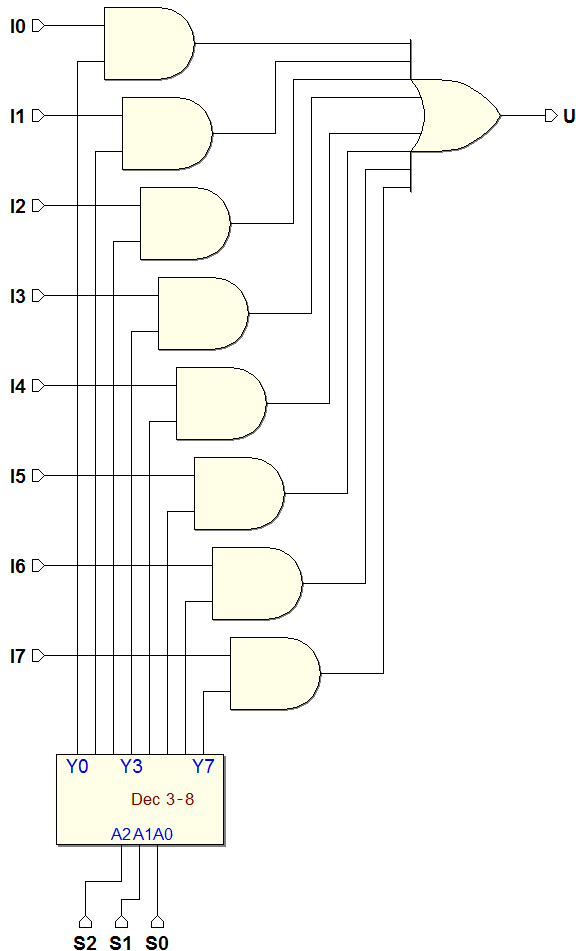
Abbiamo già incontrato un selettore a due ingressi, e la sua realizzazione mediante porte logiche: in questo caso, l’ingresso S seleziona quale dei due ingressi I0 e I1 sarà ricopiato in uscita:



E’ facile estendere il concetto a selettori con un numero maggiore di ingressi, come nella figura seguente, dove i tre ingressi S2, S1 e S0 controllano quale degli otto ingressi (I0..I7) sarà portato in uscita:

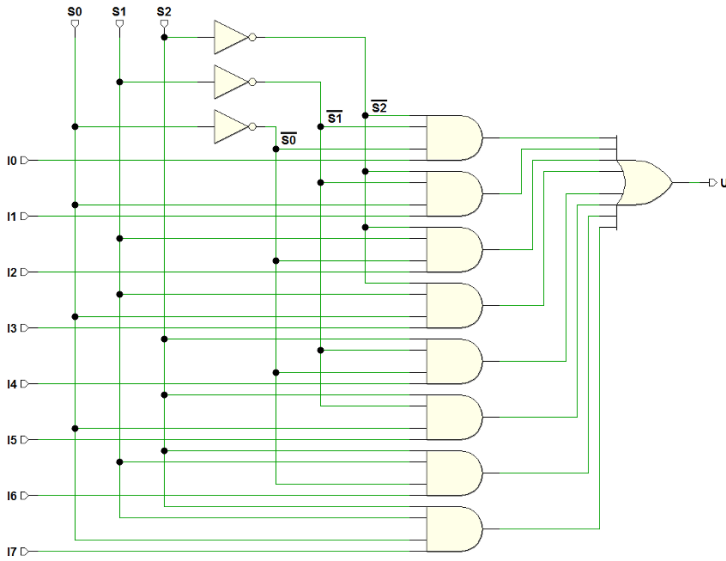


La rete ha 8 ingressi di dato e 3 di selezione. Non risulta pratico effettuare la sintesi a partire dalla tabella di verità di una rete con 11 ingressi (e quindi 2048 righe nella tabella). Viene in aiuto il concetto di decodifica (vedi figura):



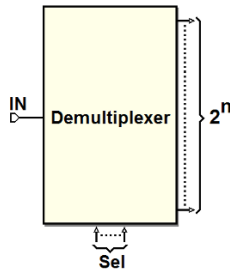
Inviando gli ingressi di selezione ad un decodificatore si può abilitare una delle 8 porte AND che trasmettono in uscita, tramite la porta OR, ognuno degli otto ingressi I_0 .. I_7 .

Se consideriamo il circuito interno al decodificatore, già esaminato in precedenza, si osserva che possiamo eliminare gli AND esterni, aggiungendo un ingresso a ciascuna delle porte AND che realizzano le uscite del decodificatore: in questo modo, la rete si riduce a due soli livelli, come risulta in figura:

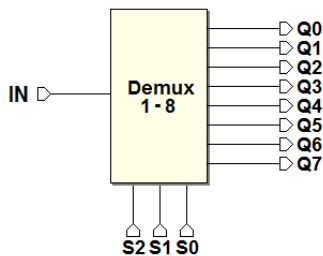


2.6.3 Deselettore (demultiplexer)

Il deselettore (“demultiplexer”, o “demux”) è una rete combinatoria dotata di un solo ingresso di dato IN , n ingressi di selezione, e 2^n uscite. Si tratta di una rete combinatoria che funziona in modo speculare rispetto al selettore (multiplexer). Il valore dell’unico ingresso è trasferito ad una delle tante uscite, scelta dalla combinazione presente sugli ingressi di selezione:



Nella figura seguente, un esempio di deselettore a 8 uscite:

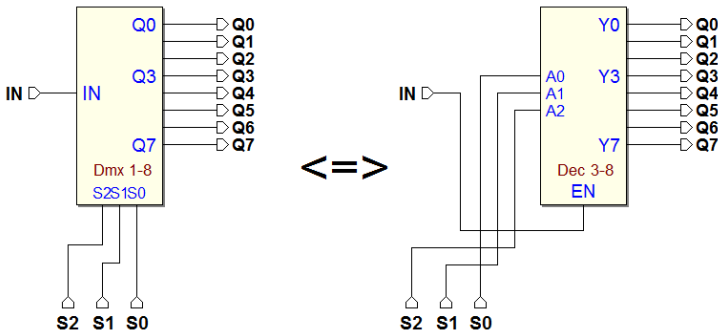


La tavola di verità è rappresentata qui sotto. Le uscite non selezionate generano sempre 0, mentre l'unica selezionata (da S_2 , S_1 e S_0) ricopia il valore dell'ingresso IN . Quindi, nella prima riga della tabella (per qualunque S_2 , S_1 e S_0 , ma con $IN = 0$), le uscite valgono tutte 0, sia quella selezionata che le altre che non lo sono. Nell'altra metà della tabella (per $IN = 1$), invece, l'uscita selezionata vale 1, dal momento che ricopia l'ingresso IN :

IN	S_2	S_1	S_0	Q_0	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7
0	-	-	-	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	0
1	0	1	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	1

Questa tavola di verità può essere riesaminata da un altro punto di vista: tutte le uscite sono a 0 quando l'ingresso IN vale 0, mentre l'unica uscita a 1 corrisponde quella selezionata, se IN vale 1. Questo comportamento, quindi, coincide con quello di un decodificatore fornito di ingresso di abilitazione. Ne consegue che un decodificatore con abilitazione può essere usato come deselettore, usando l'ingresso di abilitazione come ingresso di dato.

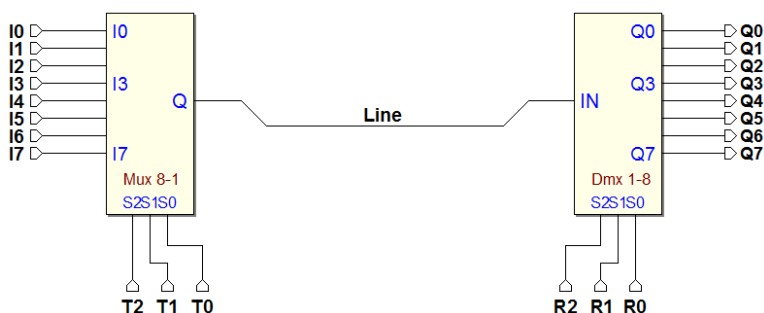
Nella figura che segue è mostrata l'equivalenza dei due componenti "Demux 1-8" e "Dec 3-8", presenti nella libreria di *Deeds*:



Esempio

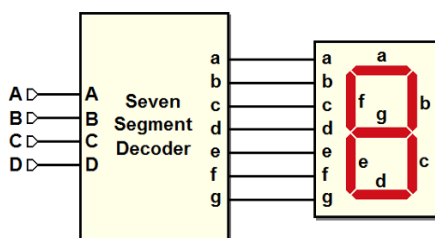
Un selettore ed un deselettore, collegati come nella figura che segue, permettono di trasferire dati, provenienti da più sorgenti, attraverso una linea monofilare, verso più destinazioni. Questo è particolarmente utile quando tra il sistema che trasmette i dati e quello che li riceve vi è grande distanza, tale da rendere pratico, per esempio, una trasmissione via radio o via cavo.

Attraverso gli ingressi di selezione $T2$, $T1$ e $T0$ del selettore si sceglie la sorgente da trasmettere, mentre dal lato del ricevitore, $R2$, $R1$ e $R0$ permettono di scegliere la destinazione dei dati:



2.6.4 Decodificatore per display a sette segmenti

Un display a sette segmenti è un dispositivo adatto a visualizzare cifre e lettere. E' dotato di 7 ingressi, uno per ogni segmento luminoso (ipotizziamo che un 1 faccia accendere il segmento corrispondente, mentre uno 0 lo mantenga spento). I segmenti sono identificati, in modo standard, dalle lettere minuscole “ a, b, c, d, e, f, g ”, come visibile in figura:



Il codice “esadecimale” (Hex) sarà esaminato nel prossimo capitolo; per ora ci basti sapere che fa corrispondere un codice di 4 bit alle cifre numeriche da 0 a 9, seguite dalle prime 6 lettere dell’alfabeto (per un totale di 16 simboli).

Il decodificatore, partendo dal numero binario a quattro bit $DCBA$ in ingresso, deve attivare sul display i segmenti che compongono il simbolo esadecimale corrispondente, come si vede in figura:

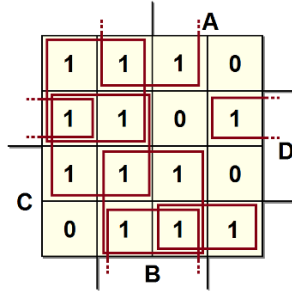


Il decodificatore avrà sette uscite a, b, c, d, e, f, g : una per ogni segmento. Ciascuna delle uscite è controllata da una rete combinatoria indipendente, che regola l’accensione o lo spegnimento di quel particolare segmento.

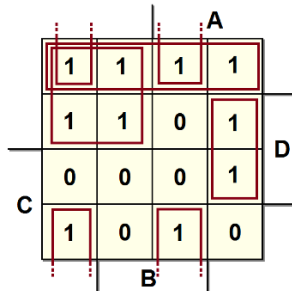
La tabella che descrive le reti per generare a, b, c, d, e, f, g è quindi la seguente (la colonna Hex è stata inserita per facilitare la comprensione della tabella):

\bar{D}	C	\bar{B}	A	a	b	c	d	e	f	g	Hex
0	0	0	0	1	1	1	1	1	1	0	0
0	0	0	1	0	1	1	0	0	0	0	1
0	0	1	0	1	1	0	1	1	0	1	2
0	0	1	1	1	1	1	1	0	0	1	3
0	1	0	0	0	1	1	0	0	1	1	4
0	1	0	1	1	0	1	1	0	1	1	5
0	1	1	0	1	0	1	1	1	1	1	6
0	1	1	1	1	1	1	0	0	0	0	7
1	0	0	0	1	1	1	1	1	1	1	8
1	0	0	1	1	1	1	1	0	1	1	9
1	0	1	0	1	1	1	0	1	1	1	A
1	0	1	1	0	0	1	1	1	1	1	B
1	1	0	0	1	0	0	1	1	1	0	C
1	1	0	1	0	1	1	1	1	0	1	D
1	1	1	0	1	0	0	1	1	1	1	E
1	1	1	1	1	0	0	0	1	1	1	F

Qui di seguito le sintesi relative alle reti che pilotano i segmenti “a” e “b”:



$$a = \bar{A}D + B\bar{D} + BC + \bar{B}\bar{C}D + AC\bar{D} + \bar{A}\bar{C}$$



$$b = \bar{C}\bar{D} + \bar{A}\bar{C} + \bar{A}\bar{B}\bar{D} + A\bar{B}D + AB\bar{D}$$

Analogamente si trovano le altre uscite.

2.6.5 Decodificatore BCD - 7 segmenti (con l'impiego delle indifferenze)

Si progetti un dispositivo analogo al precedente che però visualizzi soltanto i simboli decimali, dallo 0 al 9. Poiché si hanno solo 10 simboli da rappresentare, servono soltanto 10 combinazioni di ingressi. D'altra parte 10 combinazioni richiedono ancora 4 ingressi (infatti se ci fossero 3 ingressi si avrebbero solo $2^3 = 8$ combinazioni) e dunque restano 6 combinazioni inutilizzate.

Il problema di cosa visualizzare sul display quando in ingresso si presenta una combinazione non corrispondente ad alcuna cifra decimale può essere risolto in due approcci differenti:

- si lasciano spenti tutti i segmenti del display;
- non ci interessa quali segmenti saranno accesi o spenti.

In effetti lo scopo del dispositivo è di decodificare una cifra decimale; non rientra nei suoi scopi operare su combinazioni che non rientrano tra quelle previste. In pratica stiamo ipotizzando che gli ingressi *DCBA* siano sempre solo quelli permessi e cioè, nel nostro caso, le combinazioni da "0000" a "1001" (da 0 a 9 in decimale).

Se non ci interessa cosa sarà visualizzato nel caso di combinazioni non previste (il secondo approccio): porremo un simbolo "-", detto indifferenza ("don't care"), sulla tabella e sulle mappe, in corrispondenza delle uscite relative agli ingressi non significativi; tale simbolo andrà poi considerato come 0 o 1, indifferentemente, nella sintesi. Di conseguenza, questa è la tabella di verità del decodificatore:

<i>D</i>	<i>C</i>	<i>B</i>	<i>A</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	Dec
0	0	0	0	1	1	1	1	1	1	0	0
0	0	0	1	0	1	1	0	0	0	0	1
0	0	1	0	1	1	0	1	1	0	1	2
0	0	1	1	1	1	1	1	0	0	1	3
0	1	0	0	0	1	1	0	0	1	1	4
0	1	0	1	1	0	1	1	0	1	1	5
0	1	1	0	1	0	1	1	1	1	1	6
0	1	1	1	1	1	1	0	0	0	0	7
1	0	0	0	1	1	1	1	1	1	1	8
1	0	0	1	1	1	1	1	0	1	1	9
1	0	1	0	-	-	-	-	-	-	-	-
1	0	1	1	-	-	-	-	-	-	-	-
1	1	0	0	-	-	-	-	-	-	-	-
1	1	0	1	-	-	-	-	-	-	-	-
1	1	1	0	-	-	-	-	-	-	-	-
1	1	1	1	-	-	-	-	-	-	-	-

Le indifferenze permettono di effettuare una minimizzazione della rete in modo efficace; infatti, si possono considerare le indifferenze o come 0 o come 1, in modo da scegliere sottocubi di maggiori dimensioni.

Ad esempio, per l'uscita "a" posso considerare 4 sottocubi nel modo seguente:

		A				
	1	1	1	0		
	1	-	-	1	D	
C	-	-	-	-		
	0	1	1	1		
		B				

Si ottiene: $a = B + D + \overline{A}\overline{C} + AC$.

2.6.6 Uso di selettori per la sintesi di reti combinatorie

Nei moderni sistemi si ricorre ad un metodo per realizzare reti combinatorie programmabili che, di fatto, non richiede operazioni di sintesi. Non si tratta di reti minimizzate, ma questo metodo ha il vantaggio di consentire la ri-configurabilità della rete, anche durante il suo funzionamento, quando abbinato a sistemi di memorizzazione (che non abbiamo ancora visto).

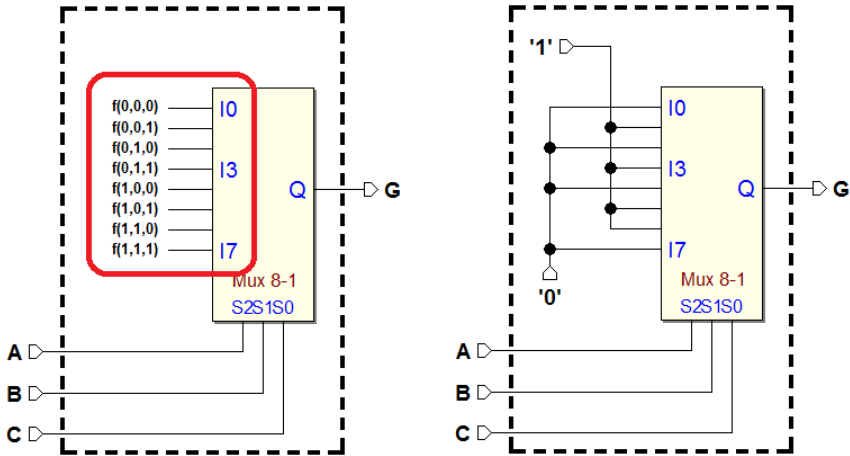
Si fa ricorso a selettori (multiplexer), come quelli esaminati in precedenza, che permettono di generare la funzione voluta selezionando i valori della funzione stessa, come se li stessi "leggendo" direttamente dalla tabella di verità. È sufficiente disporre, quindi, della tabella di verità della funzione; nel caso in cui, invece, si disponga della sua espressione, da questa dobbiamo ricavare la tabella.

- Esempio 1:** realizzare una funzione $f(A, B, C)$, di cui è data l'espressione: $G = \overline{B}C + AB\overline{C} + \overline{A}BC$. Ricaviamo la tabella di verità:

A	B	C	G
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Come mostrato nella figura seguente (a sinistra), predisponiamo una rete basata su di un selettore $8 \rightarrow 1$, collegando le tre variabili di ingresso A , B e C agli ingressi di selezione.

Quindi (lato destro della figura), colleghiamo gli 8 ingressi del selettore, nell'ordine, a delle costanti 0 o 1, pari ai valori della funzione, così come appaiono nella tabella di verità:



La rete è pronta per funzionare: per ognuna delle combinazioni degli ingressi A , B e C , il selettore trasferirà in uscita il valore che abbiamo predisposto ai suoi ingressi.

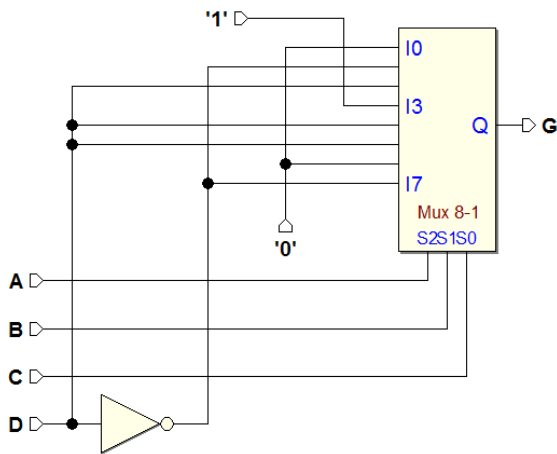
2. **Esempio 2:** la funzione è: $H = A\bar{B}D + \bar{A}BD + BC\bar{D} + \bar{A}C\bar{D}$.

Gli ingressi sono 4 (A , B , C e D); vogliamo però utilizzare un selettore con solo 3 linee di selezione, usando un piccolo trucco: condizioniamo gli 8 ingressi del selettore al valore dell'ingresso D .

Ricaviamo la tabella dalla espressione: la sua colonna più a destra riporta il valore di H in dipendenza da D , ricavato dalla tabella stessa. Sulla destra, la rete che ne risulta, collegando le costanti, l'ingresso D e \bar{D} :

A	B	C	D	H	H
0	0	0	0	0	
0	0	0	1	0	0
0	0	1	0	1	
0	0	1	1	0	\bar{D}
0	1	0	0	0	
0	1	0	1	1	D
0	1	1	0	1	
0	1	1	1	1	1
1	0	0	0	0	
1	0	0	1	1	D
1	0	1	0	0	
1	0	1	1	1	D
1	1	0	0	0	
1	1	0	1	0	0
1	1	1	0	1	
1	1	1	1	0	\bar{D}

⇒



2.7 Mappe con variabili riportate

Esponiamo ora un metodo di sintesi che impiega mappe di Karnaugh contenenti variabili al loro interno. Tale metodo verrà usato nel seguito per la sintesi delle Macchine a Stati Finiti (MSF) col metodo ASM: l'applicazione delle regole di sintesi produrrà direttamente mappe di questo tipo. Nella figura seguente vediamo un esempio di mappa con variabili riportate:

		A		
	E	0	0	G
C	0	1	0	0
		B		

Oltre alle variabili A , B e C , rappresentate intorno alla mappa, osserviamo le variabili E e G riportate all'interno di due caselle: questa mappa rappresenta una funzione in 5 variabili $f(A, B, C, E, G)$. Scrivere una variabile VAR (o anche una intera espressione ESP) all'interno di una casella vuol dire condizionare il risultato della funzione alla variabile VAR (o alla espressione ESP).

Nell'esempio considerato, la casella in alto a sinistra corrisponde al prodotto fondamentale $\bar{A} \bar{B} \bar{C}$; tuttavia la presenza della variabile nella casella obbliga a condizionare il termine ad E , ottenendo: $\bar{A} \bar{B} \bar{C} E$. Per la casella in alto a destra, condizioniamo il prodotto fondamentale corrispondente alla variabile G , ottenendo: $\bar{A} \bar{B} \bar{C} G$. Ne risulta la funzione:

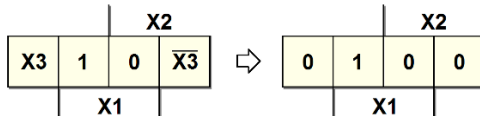
$$f = \bar{A} \bar{B} \bar{C} E + \bar{A} \bar{B} \bar{C} G + A B C$$

In questo esempio l'espressione della funzione non è minimizzabile: in generale, però, il problema della minimizzazione esiste. Vediamo ora uno dei metodi esistenti per ottenere una sintesi minima (o almeno il più possibile ridotta).

2.7.1 Sintesi di una mappa con variabili riportate

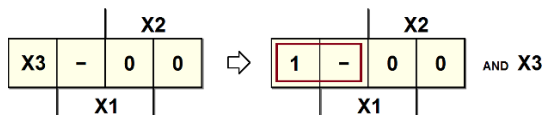
Esaminiamo qui un metodo per la sintesi di mappe con variabili riportate. Nell'esempio troviamo la variabile $X3$ in una casella, e la sua negata in un'altra. Si eseguono alcuni passi successivi:

1. Si azzerano tutte le variabili riportate nella mappa, e si sintetizzano gli 1:



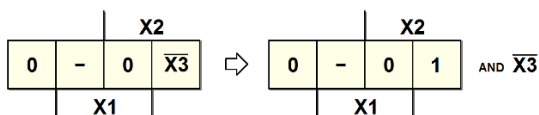
Otteniamo un primo risultato parziale: $f' = X_1 \bar{X}_2$.

2. Si pongono uguali ad indifferenza gli 1 sintetizzati al punto precedente. Si scelgono poi le caselle contenenti una stessa variabile, ponendole uguali a 1, e si lasciano le rimanenti a 0 (si considerano la variabile e la sua negata come due variabili indipendenti da considerare in due passi distinti, una alla volta). Si sintetizza la mappa così trovata, ponendo il risultato in AND con la variabile scelta:



Scriviamo il secondo risultato parziale: $f'' = \overline{X_2} X_3$.

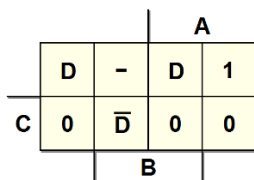
Si ripete il passo 2 per tutte le altre variabili riportate sulla mappa. Terminato questo ciclo di passi, l'espressione equivalente è data dall'OR di tutte le espressioni AND trovate. Nell'esempio, il passo 2 si ripete ancora una volta:



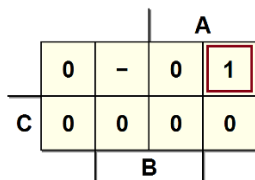
Da cui ricaviamo l'ultimo risultato parziale: $f''' = \overline{X_1} X_2 \overline{X_3}$, e quindi l'espressione finale:

$$f = f' + f'' + f''' = X_1 \overline{X_2} + \overline{X_2} X_3 + \overline{X_1} X_2 \overline{X_3}$$

Esempio 1: data la mappa seguente, ricavarne la funzione corrispondente:



1. Azzerò le caselle contenenti D e \overline{D} e sintetizzo gli 1 così rimasti; notare che l'indifferenza resta immutata e potrebbe essere usata per minimizzare la mappa (non serve in questo esempio):



Otengo l'espressione: $f' = A \overline{B} \overline{C}$.

2. Sostituisco quindi un'indifferenza all'unico 1 della mappa, e pongo ad 1 le caselle contenenti D e metto a 0 quelle contenenti \bar{D} :

		A		
	1	-	1	-
C	0	0	0	0
		B		

AND D

Così facendo ottengo dalla sintesi \bar{C} , che va messo in AND con D , per cui il secondo termine dell'espressione cercata è: $f'' = \bar{C} D$.

3. Ripeto il passo 2 per le caselle con \bar{D} , come se fosse una variabile indipendente da D , ponendo a 1 le caselle con \bar{D} e a 0 quelle con D :

		A		
	0	-	0	-
C	0	1	0	0
		B		

AND \bar{D}

Dalla mappa così ottenuta ricavo il termine $\bar{A} B$, che messo in AND con \bar{D} mi fornisce l'ultimo termine dell'espressione cercata: $f''' = \bar{A} B \bar{D}$.

4. L'espressione finale è: $f = f' + f'' + f''' = A \bar{B} \bar{C} + \bar{C} D + \bar{A} B \bar{D}$.

Esempio 2: ricaviamo la funzione da una mappa con due variabili riportate:

		A					A			
	-	D	-	1	→		-	0	-	1
C	E	0	0	\bar{E}		C	0	0	0	0
		B					B			

Eseguendo il primo passo, otteniamo $A \bar{C}$; notare come una sintesi ugualmente valida dal passo 1 poteva essere $\bar{B} \bar{C}$. Eseguo poi il passo 2 una prima volta inserendo 1 nelle caselle con D , e 0 in quelle contenenti E e \bar{E} :

		A		
	-	1	-	-
C	0	0	0	0
		B		

AND D

La sintesi della mappa mi dà \bar{C} , quindi il termine cercato è $\bar{C} D$. Si noti come nel termine entri solo la variabile che si è considerata, mentre le altre non compaiono né dirette, né negate.

Eseguo ancora il passo 2, questa volta per E : azzerò le caselle con D ed \bar{E} :

		A		
	-	0	-	-
C	1	0	0	0
		B		

AND E

La sintesi della mappa fornisce il termine $\overline{A}\overline{B}$, da cui $\overline{A}\overline{B}E$. Infine eseguo ancora il passo 2 per la casella contenente \overline{E} , azzerando quelle con D ed E :

		A		
	-	0	-	-
C	0	0	0	1
		B		

AND \overline{E}

Otteniamo il termine $A\overline{B}$, da cui ottengo $A\overline{B}\overline{E}$; l'espressione finale è:

$$f = A\overline{C} + \overline{C}D + \overline{A}\overline{B}E + A\overline{B}\overline{E}$$

2.7.2 Le variabili riportate e i teoremi di espansione

Sotto un certo punto di vista, le mappe a variabili riportate possono essere considerate come un ibrido, una via di mezzo tra la rappresentazione di reti logiche mediante la loro espressione booleana e quella mediante la loro mappa di Karnaugh.

La “classica” mappa che contiene solo 0 e 1 non è altro che una mappa a variabili riportate priva di variabili riportate, mentre, se riportiamo tutte le variabili di una mappa, quello che otteniamo nell'unica casella che resta è proprio l'espressione booleana della rete. Sarebbe possibile dimostrare come le mappe con variabili riportate si possano ottenere dalla “compressione” di mappe di Karnaugh ordinarie, mettendole in relazione con i teoremi di espansione di Shannon, ma tale dimostrazione esula dai nostri scopi.

Proviamo solo a riprendere, a titolo di esempio, la mappa del selettore a due canali, confrontandola con la equivalente mappa con variabili riportate:

		S1		
	0	0	1	1
SEL	0	1	1	0
		S2		

≡

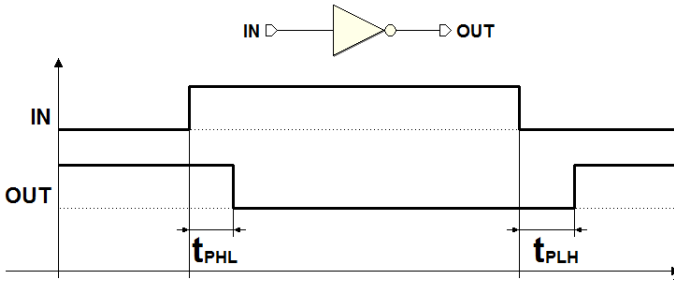
	S1
SEL	S2

Sarebbe stato decisamente più semplice scrivere la mappa con variabili riportate, riportando direttamente in essa la descrizione a parole del selettore (“se SEL vale 1, la funzione ricopia S2, altrimenti ricopia S1”).

2.8 Comportamento nel tempo dei circuiti logici

2.8.1 Tempi Caratteristici

Una porta logica presenta un ritardo tra il cambiamento del valore d'ingresso e quello di uscita. Nell'esempio in figura, i segnali di ingresso e di uscita di un NOT sono rappresentati in forma idealizzata, indicando come istantanee le transizioni tra i valori logici:



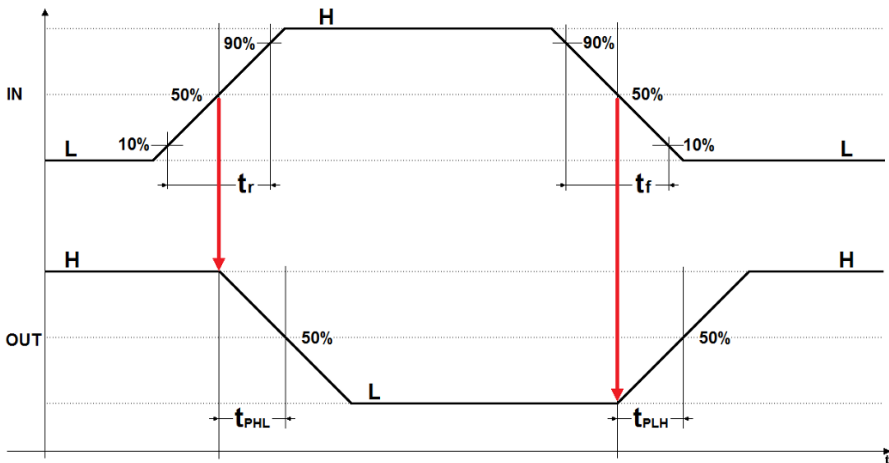
t_{PHL} : tempo di propagazione (transizione alto - basso)

t_{PLH} : tempo di propagazione (transizione basso - alto)

I ritardi definiti in questa forma sono denominati *ritardi di trasporto*: questa rappresentazione idealizzata presenta i vantaggi della immediatezza e della facilità di lettura. Inoltre, per semplicità, possiamo indicare genericamente il tempo di propagazione come:

$$t_p = \max(t_{PLH}, t_{PHL})$$

Quando necessario, è utile adottare un modello più realistico dei segnali, nella quale le transizioni di livello (fronti) avvengono non in un tempo nullo ma in tempi finiti, e con andamento lineare. Si parla di *ritardi inerziali*:



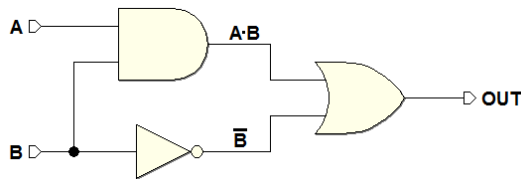
Con questa rappresentazione si assume che il cambiamento del livello logico avvenga quando il segnale ha raggiunto (in salita o in discesa) il 50% della sua escursione.

Il tempo di salita (rise time) t_r è quello che intercorre tra il 10% e il 90% dell'escursione del segnale dal livello basso a quello alto.

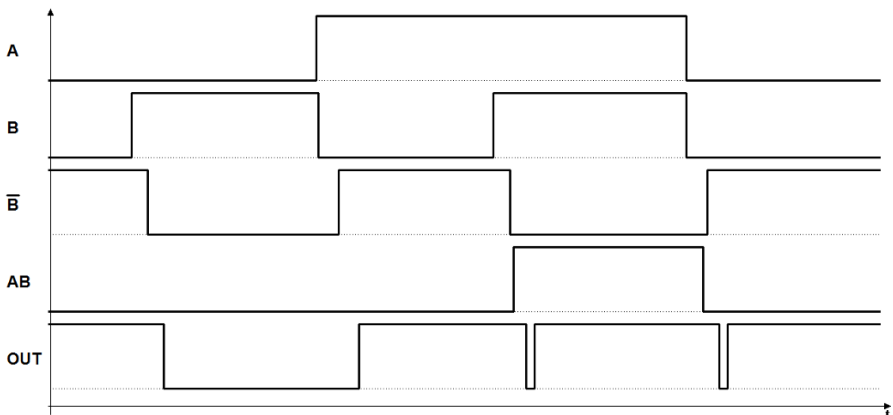
Il tempo di discesa (fall time) t_f è quello valutato tra il 10% e il 90% dell'escursione del segnale dal livello alto a quello basso.

I tempi di propagazione t_{PHL} e t_{PLH} sono invece misurati tra il cambiamento di livello del segnale in ingresso e quello corrispondente del segnale in uscita (si faccia attenzione a non confondere i tempi di salita e discesa con i tempi di propagazione).

La rappresentazione mediante i *ritardi di trasporto* prevede solo la traslazione nel tempo del segnale d'uscita rispetto all'ingresso, e non spiega il fatto che nella realtà, segnali la cui durata è inferiore ad un certo tempo non sono propagati. Utilizzando il modello inerziale, invece, riusciamo a spiegare questo comportamento. Prendiamo in considerazione il seguente circuito logico:

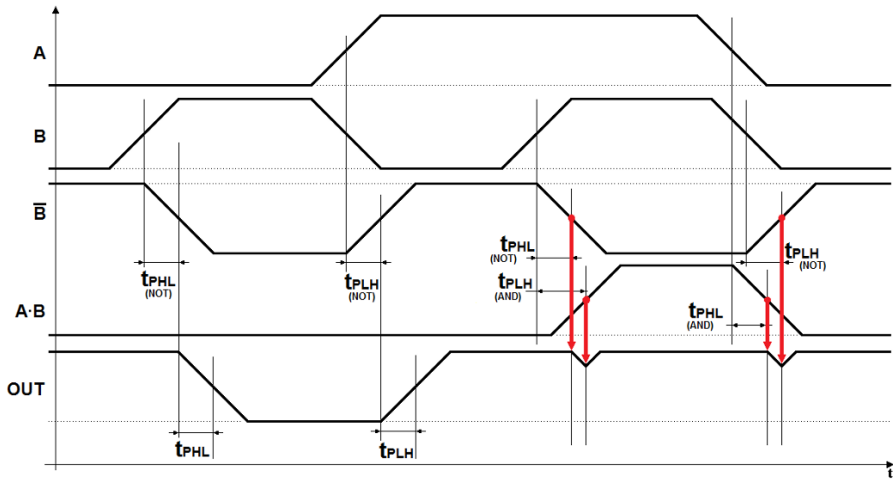


Se analizziamo il circuito con un simulatore di circuiti digitali impostato in modo da tenere conto solo dei *ritardi di trasporto*, otteniamo:



Sulla base di questo modello, quindi, sull'uscita OUT dovrebbero essere presenti due impulsi di breve durata, dovuti alle differenze di ritardo lungo i percorsi dei segnali, e alla differenza tra i tempi t_{PHL} e t_{PLH} .

Tuttavia, se ripetiamo la simulazione utilizzando il modello dei *ritardi inerziali*, otteniamo:

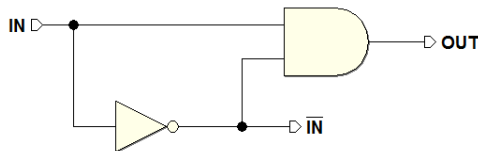


In questa nuova simulazione, maggiormente realistica, si vede che nel nostro esempio i due impulsi sono in effetti presenti, ma non saranno propagati, dal momento che la loro ampiezza, a causa delle pendenze finite delle transizioni, non riesce a raggiungere il livello di soglia.

2.8.2 Alee

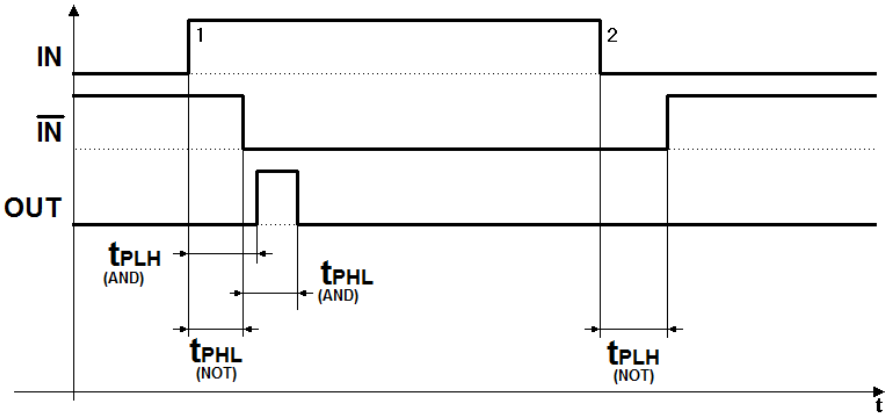
Come si è visto, le reti combinatorie possono dar luogo a comportamenti impulsivi, dovuti alle differenze di ritardo. Questi comportamenti sono denominati *alee* (hazards). A seconda della tecnologia di fabbricazione dei componenti utilizzati, tali comportamenti possono essere (o non essere) rimosse dal comportamento inerziale dei segnali. Tali fenomeni sono in ogni caso potenzialmente dannosi per i loro effetti sugli stadi successivi e devono essere evitati quando possono creare errori.

Le alee possono essere eliminate (o *mascherate*) mediante procedimenti algebrici, come quello esposto nel seguito. Le alee che nascono in seguito alla presenza di percorsi di ritardo asimmetrici, dovuti alla presenza di invertitori o altre porte, sono chiamate *alee statiche*.



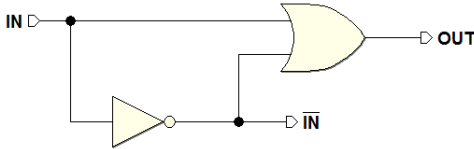
Nella figura qui sopra, una semplice rete composta da una porta AND e un NOT ci consente di osservare l'effetto dei percorsi differenti.

Questa rete dovrebbe generare una costante (infatti $OUT = IN \cdot \overline{IN} = 0$). Ai fini dell'analisi nel tempo, consideriamo i soli ritardi di trasporto, in modo che l'alea non risulti mascherata dall'andamento inerziale dei segnali:

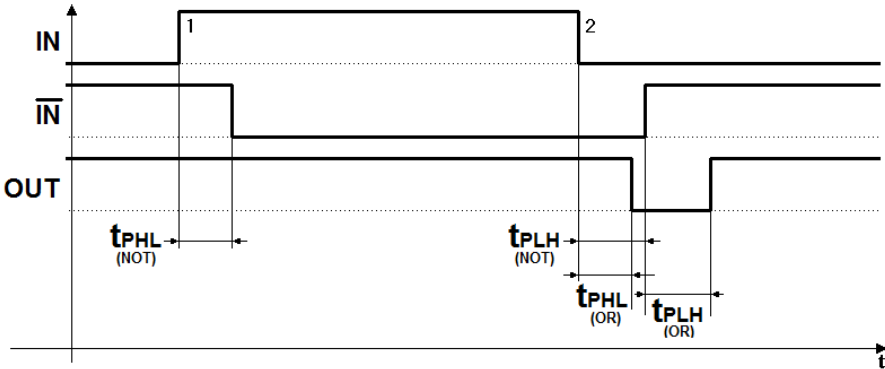


Come si vede in figura, l'uscita non risulta sempre a 0, ma manifesta un impulso a 1 (l'alea). Infatti, per un certo tempo, a causa del ritardo del NOT, gli ingressi della AND sono contemporaneamente a 1.

Per completezza, esaminiamo un altro semplice circuito, basato questa volta su di una porta OR. Anche in questo caso l'uscita dovrebbe essere costante (dato che $OUT = IN + \overline{IN} = 1$):



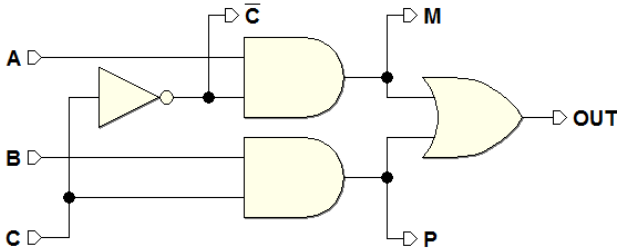
L'analisi nel tempo, eseguita con gli stessi criteri di prima, mostra anche in questo caso la presenza di un'alea (sulla transizione opposta):



In generale, nelle reti AND-OR le alee assumono la forma di una breve transizione a 0 di un segnale che dovrebbe essere stabile a 1. Nelle reti OR-AND si ha la transizione a 1 di un segnale che dovrebbe essere a 0.

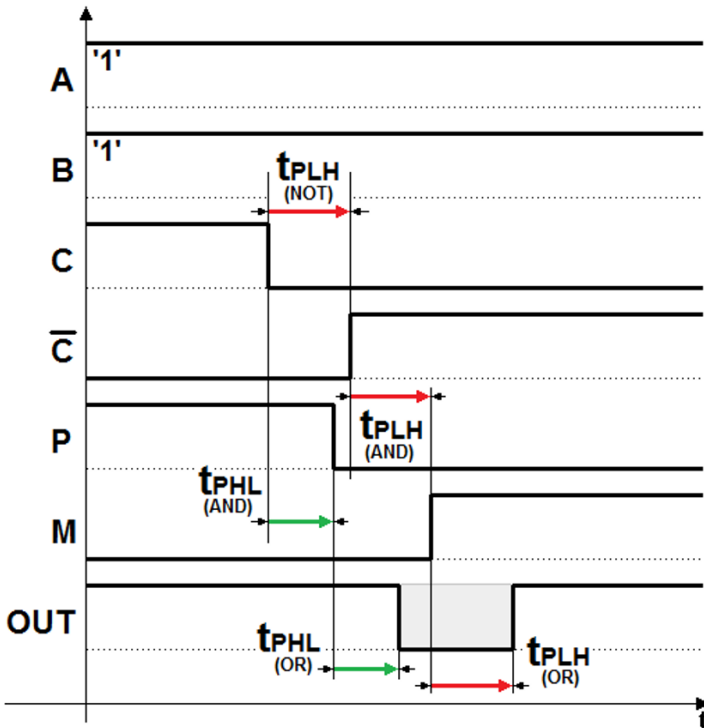
2.8.3 Eliminazione delle alee statiche

Nel seguente circuito AND-OR a due livelli (il selettore $2 \rightarrow 1$) è presente un'alea nella transizione $1 \rightarrow 0$ di C , quando gli ingressi A e B valgono 1:

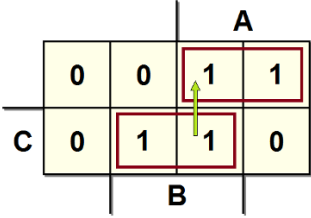


$$OUT = A\bar{C} + BC$$

La simulazione temporale mostra che l'alea si genera in quanto, per un breve intervallo di tempo, i due prodotti logici $A\bar{C}$ e BC si azzerano entrambi, a causa del ritardo introdotto dal NOT.



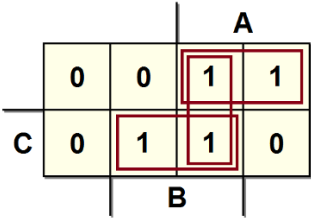
Se si osserva la mappa di Karnaugh della rete, si può notare che l'alea si sviluppa nella transizione indicata dalla freccia (cioè, come detto sopra, quando C passa da 1 a 0, mentre $A = 1$ e $B = 1$):



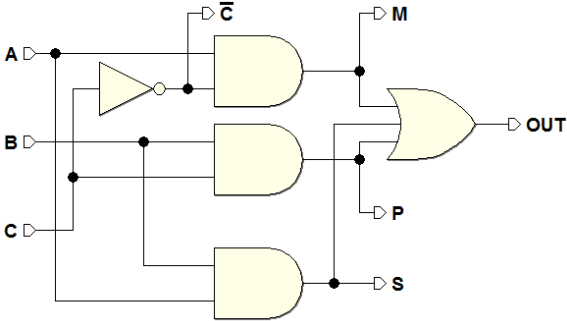
Esiste un teorema (che qui non dimostriamo) che afferma: una rete combinatoria a due livelli, realizzata come somma di prodotti, è libera da alee se è libera da alee nelle transizioni $1 \rightarrow 1$ (da un insieme di ingressi che producono 1 ad un altro insieme di ingressi che producono 1).

In una rete combinatoria AND-OR a due livelli si può avere un'alea quando sulla mappa della rete c'è una coppia di 1 vicini, non contenuti nel medesimo implicante, come nella mappa ora esaminata.

Verifichiamo la presenza di un'alea in questa rete in quanto vi sono due 1 adiacenti in implicanti differenti. Per eliminare l'alea è necessario aggiungere l'implicante che contiene questi due 1 (cioè il termine AB):

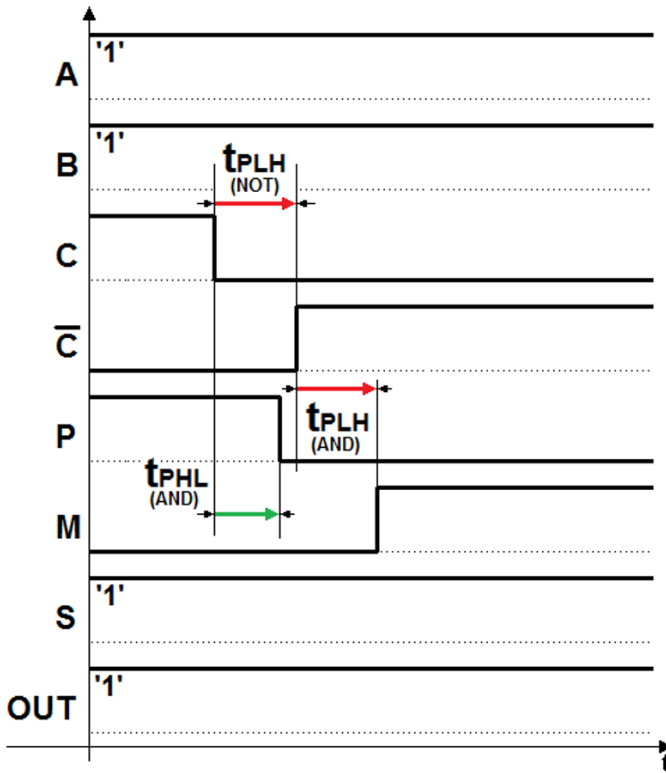


Da cui risulta: $OUT = A\bar{C} + BC + AB$; qui sotto lo schema della nuova rete:



Non è più una sintesi minima, ma il termine di prodotto aggiunto maschera (o ricopre) l'alea, che non compare più nella simulazione temporale.

Infatti, nell'intervallo di tempo in cui i due prodotti logici $A\bar{C}$ e BC si azzerano entrambi, il termine AB mantiene un 1 all'ingresso della OR, come si vede nella simulazione rappresentata in figura:



Nelle reti OR-AND la situazione è simmetrica: l'alea si verifica nelle transizioni $0 \rightarrow 0$ e si eliminano aggiungendo l'implicante che contiene due zeri vicini non contenuti nello stesso sottocubo.

2.8.4 Note sulla eliminazione delle alee

In generale, per eliminare le alee statiche, si procederà in modo sistematico aggiungendo implicanti non essenziali in tutte le situazioni che possono produrre alee. Si osservi però che l'eliminazione delle alee è necessaria solo quando sono dannose, tipicamente nei *circuiti sequenziali asincroni*, come si vedrà nel seguito.

Le alee invece possono essere generalmente tollerate nei *circuiti sequenziali sincroni*, nei quali la lettura dei valori logici avviene in istanti di tempo ben definiti e separati dagli intervalli di tempo in cui si manifestano le alee, come sarà esaminato più avanti.

2.9 Esercizi

2.9.1 Mappe

1. Tracciare le tavole di verità e le mappe delle seguenti funzioni:

a) $G = ABC + B\bar{C}$

b) $H = (A + \bar{B})(B + \bar{C})$

2. Costruire le mappe delle seguenti funzioni

a) $F = \bar{A}\bar{B}D + \bar{A}BC + ABD + A\bar{B}\bar{C}\bar{D} + \bar{A}B\bar{C}D$

b) $M = (\bar{A} + C)(\bar{A} + \bar{C} + \bar{D})(\bar{A} + B)(A + B + \bar{C} + D)$

3. Minimizzare come somma di prodotti le funzioni logiche rappresentate dalle mappe che seguono:

a) F1:

		A				
	1	1	1	0		
	1	1	1	0	D	
	0	0	1	0		
C	0	1	1	0		
		B				

b) F2:

		A				
	1	0	0	1		
	1	0	1	1	D	
	0	0	1	0		
C	0	1	1	0		
		B				

c) F3:

		A				
	0	0	1	1		
	0	0	0	0	D	
	1	1	0	1		
C	0	0	1	1		
		B				

d) F4:

		A		
	1	0	1	1
	0	0	0	0
C	0	1	1	0
	0	0	1	0
		B		

- Minimizzare la funzione $F = \overline{B}C\overline{D} + \overline{A}B\overline{C}\overline{D} + A\overline{B}\overline{D}$ come somma di prodotti, tenendo presente che le combinazioni di ingressi $ABCD = '11 - -'$ e $ABCD = '- - 11'$ non sono mai presenti (non possono mai verificarsi le combinazioni per cui $A = B = 1$ oppure $C = D = 1$).
- Sintetizzare come somma di prodotti la funzione logica rappresentata dalla mappa seguente:

		A		
	1	1	1	1
C	1	0	1	1
		B		

- Sintetizzare come prodotto di somme la funzione logica rappresentata dalla mappa dell'esercizio precedente.
- Effettuare la sintesi della mappa seguente, contenente indifferenze:

		A		
	0	1	-	1
	1	0	-	0
C	0	1	-	-
	1	0	-	-
		B		

- Disegnare, usando solo porte NAND, il piú semplice circuito che realizza la funzione:

$$F = (AB + \overline{A}\overline{B})CD + (A\overline{C} + \overline{A}C)BD + \overline{A}CD + A\overline{B}CD.$$

- Eseguire la sintesi della seguente mappa con variabili riportate (raggruppare solo le indifferenze che ci permettono di effettuare raggruppamenti

più grossi; ignorare invece quelle che aggiungerebbero gruppi):

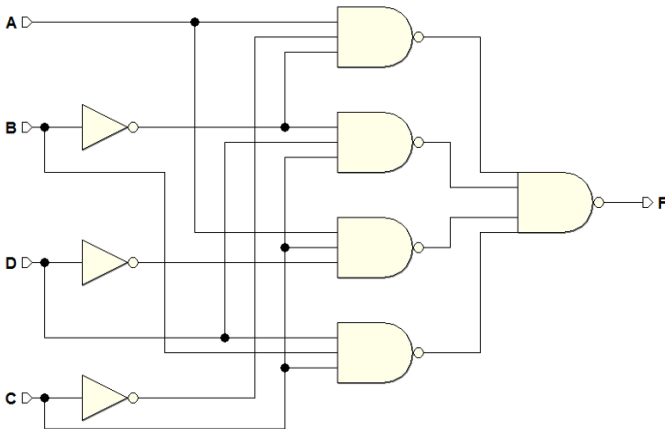
		A		
	0	X	0	0
	0	1	\bar{X}	\bar{X}
C	0	1	\bar{X}	\bar{X}
	0	X	0	0
		B		

2.9.2 Alee

- Ricavare la sintesi minima dalla mappa seguente.
 - Senza considerare le alee;
 - Eliminando le alee.

		A		
	1	1	1	1
	0	0	1	0
C	1	1	1	0
	1	1	0	0
		B		

- Identificare le alee nel circuito e modificarlo per eliminarle:



2.10 Soluzioni

2.10.1 Mappe

1. a) $G = ABC + B\bar{C}$

A	B	C	G
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

		A	
		0	1
C		0	0
		B	
		1	0
		0	0

b) $H = (A + \bar{B})(B + \bar{C})$

A	B	C	H
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

		A	
		1	0
C		0	0
		B	
		1	1
		0	0

2. a) $F = \bar{A}\bar{B}D + \bar{A}BC + ABD + A\bar{B}\bar{C}\bar{D} + \bar{A}B\bar{C}D$

		A	
		0	1
		1	0
D		1	1
		C	
C		1	0
		0	0
		B	

b) $M = (\bar{A} + C)(\bar{A} + \bar{C} + \bar{D})(\bar{A} + B)(A + B + \bar{C} + D)$

		A	
		1	1
		1	0
D		1	0
		C	
C		1	1
		0	0
		B	
		0	0
		1	1

3. a) $F1 = \overline{A}\overline{C} + AB + B\overline{D}$:

		A		
		1	1	0
		1	1	0
		0	0	1
C		0	1	1
		B		

b) $F2 = \overline{B}\overline{C} + ABD + BC\overline{D}$:

		A		
		1	0	0
		1	0	1
		0	0	1
C		0	1	1
		B		

c) $F3 = A\overline{D} + \overline{B}CD + \overline{A}CD$:

		A		
		0	0	1
		0	0	1
		0	0	0
C		1	1	0
		B		

d) $F4 = \overline{B}\overline{C}\overline{D} + AB\overline{D} + BCD$:

		A		
		1	0	1
		1	0	1
		0	0	0
C		0	1	1
		B		

4. $F = A\overline{D} + \overline{B}C + B\overline{C}\overline{D}$:

		A		
		0	1	-
		0	0	-
		-	-	-
C		1	0	-
		B		

5. Questi i tre cottocubi (di ordine 2) per la sintesi AND-OR:

		A			
		1	1	1	1
C		1	0	1	1
		B			

Otteniamo l'espressione: $F = \bar{A} + \bar{B} + \bar{C}$.

6. Per la sintesi OR-AND è sufficiente considerare l'unico 0 presente:

		A			
		1	1	1	1
C		1	0	1	1
		B			

Si ricava la stessa espressione dell'esercizio precedente (ma usando un solo raggruppamento).

7. $F = \bar{A}\bar{B}\bar{C}D + \bar{B}C\bar{D} + B\bar{C}\bar{D} + BCD + A\bar{D}$

8. Espandiamo l'espressione data in termini di prodotto fondamentali:

$$\begin{aligned}
 F &= (AB + \bar{A}\bar{B})CD + (A\bar{C} + \bar{A}C)BD + \bar{A}CD + A\bar{B}CD = \\
 &= ABCD + \bar{A}\bar{B}CD + AB\bar{C}D + \bar{A}BCD + \bar{A}(B + \bar{B})CD + A\bar{B}CD = \\
 &= ABCD + \bar{A}\bar{B}CD + AB\bar{C}D + \bar{A}BCD + A\bar{B}CD
 \end{aligned}$$

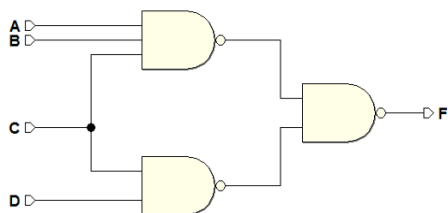
Da cui la mappa:

		A			
		0	0	0	0
		0	0	1	0
C		1	1	1	1
		0	0	0	0
		B			

Che produce l'espressione minimizzata: $F = CD + ABC$. Con il teorema di De Morgan, la trasformiamo in NAND-NAND:

$$F = \overline{\bar{C}\bar{D}} \cdot \overline{\bar{A}\bar{B}\bar{C}}$$

Qui il disegno della rete:



9. Con il metodo studiato per le mappe con variabili riportate, abbiamo:

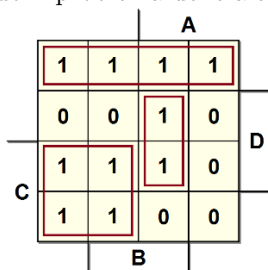
$$F = \bar{A}BD + \bar{A}BX + AD\bar{X}$$

2.10.2 Alee

1. Sintesi di una funzione considerando il problema delle alee:

a) Con le alee non eliminate:

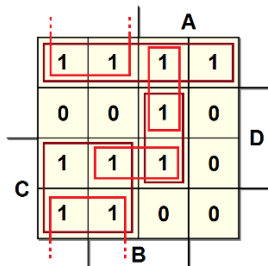
$$\bar{C}\bar{D} + \bar{A}C + ABD :$$



b) Con le alee "coperte":

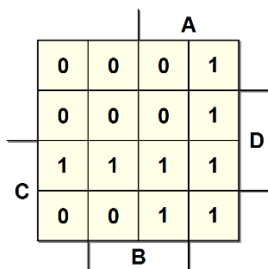
$$\bar{C}\bar{D} + \bar{A}C + ABD +$$

$$\bar{A}\bar{D} + BCD + ABC\bar{C} :$$

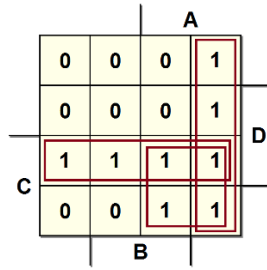


2. Dalla rete ricaviamo, tenendo conto della equivalenza con le reti NAND-NAND, l'espressione della rete (AND-OR) e, da questa, la mappa:

$$F = A\bar{B}\bar{C} + \bar{B}CD + AC\bar{D} + BCD$$

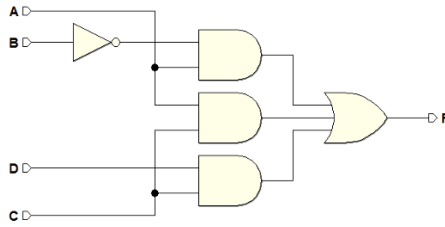


Raggruppiamo la mappa per la sintesi minima:



Nella mappa, tutti gli 1 adiacenti risultano coperti, per cui la sintesi minima è già priva di alee. Qui la sua espressione, e la rete logica:

$$F = A \bar{B} + C D + A C$$



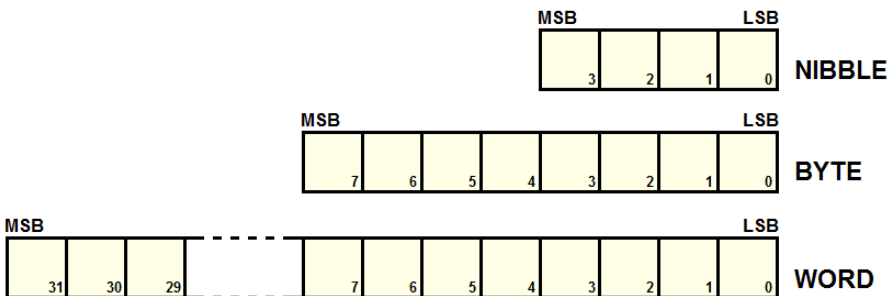
Aritmetica binaria

3.1 Informazione binaria

L'elemento fondamentale dell'informazione binaria prende il nome di *bit* (binary digit). Il bit può assumere solo il valore 0 oppure il valore 1:



Normalmente i bit sono raggruppati in insiemi significativi, denominati: *nibble* (4 bit), *byte* (8 bit) o genericamente *word* se in numero maggiore (per esempio 16, 32 o 64 bit):



Nel capitolo vedremo come utilizzare questi insiemi di bit per rappresentare numeri, o altro tipo di informazione, mediante codici standard.

Nella figura qui sopra, le abbreviazioni MSB e LSB indicano rispettivamente il bit più significativo (*Most Significant Bit*), è il bit meno significativo (*Least Significant Bit*), nel caso in cui questi insiemi di bit siano utilizzati per codificare numeri.

3.2 Numerazione binaria (BIN)

La numerazione binaria è, come quella decimale, una numerazione di tipo *posizionale*. Una notazione posizionale permette di esprimere un numero N nel seguente modo:

$$N = n_{m-1} \cdot R^{m-1} + \dots + n_1 \cdot R^1 + n_0 \cdot R^0$$

dove R è la *base* del sistema di numerazione; m è il numero di cifre di cui è composta la rappresentazione; gli esponenti indicano la *posizione* k di ogni cifra a partire da destra, compresa tra 0 e $m-1$; l'insieme $A = \{0, \dots, R-1\}$ è l'*alfabeto* del sistema di numerazione ed è costituito dai *simboli* con cui rappresentiamo i numeri; i coefficienti n sono dei numeri che corrispondono al valore dei simboli dell'insieme A .

In numerazione decimale l'alfabeto dei simboli è $A = \{0, 1, \dots, 9\}$ e, ovviamente, $R = 10$. Quando scriviamo un numero N , come ad esempio 2017, intendiamo:

$$2017_{10} = 2 \cdot 10^3 + 0 \cdot 10^2 + 1 \cdot 10^1 + 7 \cdot 10^0$$

Chiameremo *numerazione binaria* quella numerazione che ha base $R = 2$ e alfabeto $A = \{0, 1\}$. Ad esempio, il numero 1110 in base 2 corrisponde al numero 14 in base 10:

$$1110_2 = 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 8 + 4 + 2 + 0 = 14_{10}$$

Definiamo infine il *peso* come:

$$p = R^k$$

In decimale i pesi corrispondono alle unità, decine, centinaia, migliaia... (potenze della base dieci); nella numerazione binaria i pesi hanno il valore delle potenze della base due: 1, 2, 4, 8, 16, 32, 64, 128...

3.2.1 Conversione da numerazione binaria a decimale

Si ottiene applicando direttamente la definizione vista sopra. Ecco qui degli esempi con 4 e 8 cifre:

$$0000_2 = 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 0_{10}$$

$$0011_2 = 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 3_{10}$$

$$1111_2 = 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 15_{10}$$

$$00000010_2 = 0 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 2_{10}$$

$$10001110_2 = 1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 142_{10}$$

$$11110011_2 = 1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 243_{10}$$

$$11111111_2 = 1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 255_{10}$$

3.2.2 Conversione da numerazione decimale a binaria

Un numero intero N (con $N \geq 2$) può essere espresso come:

$$N = 2 \cdot q_0 + r_0 \quad \text{con} \quad r_0 = (0, 1)$$

dove q_0 ed r_0 sono rispettivamente il quoziente ed il resto della divisione di N per 2. Possiamo reiterare il processo, se $q_0 \geq 2$, scrivendo che:

$$q_0 = 2 \cdot q_1 + r_1 \quad \text{con} \quad r_1 = (0, 1)$$

Sostituendo quest'ultima nella prima, otteniamo:

$$N = 2 \cdot (2 \cdot q_1 + r_1) + r_0$$

Proseguendo nel processo, se $q_1 \geq 2$, si ricava:

$$N = 2 \cdot (2 \cdot (2 \cdot q_2 + r_2) + r_1) + r_0 = 2^3 \cdot q_2 + 2^2 \cdot r_2 + 2^1 \cdot r_1 + 2^0 \cdot r_0$$

e così via iterando, fintanto che $q_k \geq 2$, ottenendo:

$$N = 2^k \cdot q_{k-1} + 2^{k-1} \cdot r_{k-1} + \dots + 2^1 \cdot r_1 + 2^0 \cdot r_0.$$

In altre parole, i resti r_i delle divisioni per 2 rappresentano le cifre binarie di N . Consideriamo per esempio $N_{10} = 46$. Dividiamo il numero per 2 in colonna, ricorsivamente, scrivendo a destra della riga il resto e sotto al numero il risultato; otteniamo:

	46	0	(r_0)
(q_0)	23	1	(r_1)
(q_1)	11	1	(r_2)
(q_2)	5	1	(r_3)
(q_3)	2	0	(r_4)
(q_4)	1	1	(q_4)

Il risultato della conversione si ottiene scrivendo, nell'ordine, da sinistra, q_4 e poi i resti r_4, r_3, \dots, r_0 :

$$N = 46_{10} = 101110_2$$

Controprova:

$$\begin{aligned} N &= q_4 \cdot 2^5 + r_4 \cdot 2^4 + r_3 \cdot 2^3 + r_2 \cdot 2^2 + r_1 \cdot 2^1 + r_0 \cdot 2^0 = \\ &= 1 \cdot 32 + 0 \cdot 16 + 1 \cdot 8 + 1 \cdot 4 + 1 \cdot 2 + 0 \cdot 1 = 46_{10} \end{aligned}$$

A titolo di esempio, convertiamo in binario alcuni numeri decimali.

$258_{10}:$ <table style="border-collapse: collapse; margin-left: 20px;"> <tr><td style="padding-right: 5px;">258</td><td style="border-left: 1px solid black; padding-left: 5px;">0</td></tr> <tr><td style="padding-right: 5px;">129</td><td style="border-left: 1px solid black; padding-left: 5px;">1</td></tr> <tr><td style="padding-right: 5px;">64</td><td style="border-left: 1px solid black; padding-left: 5px;">0</td></tr> <tr><td style="padding-right: 5px;">32</td><td style="border-left: 1px solid black; padding-left: 5px;">0</td></tr> <tr><td style="padding-right: 5px;">16</td><td style="border-left: 1px solid black; padding-left: 5px;">0</td></tr> <tr><td style="padding-right: 5px;">8</td><td style="border-left: 1px solid black; padding-left: 5px;">0</td></tr> <tr><td style="padding-right: 5px;">4</td><td style="border-left: 1px solid black; padding-left: 5px;">0</td></tr> <tr><td style="padding-right: 5px;">2</td><td style="border-left: 1px solid black; padding-left: 5px;">0</td></tr> <tr><td style="padding-right: 5px;">1</td><td style="border-left: 1px solid black; padding-left: 5px;">1</td></tr> </table>	258	0	129	1	64	0	32	0	16	0	8	0	4	0	2	0	1	1	$205_{10}:$ <table style="border-collapse: collapse; margin-left: 20px;"> <tr><td style="padding-right: 5px;">205</td><td style="border-left: 1px solid black; padding-left: 5px;">1</td></tr> <tr><td style="padding-right: 5px;">104</td><td style="border-left: 1px solid black; padding-left: 5px;">0</td></tr> <tr><td style="padding-right: 5px;">57</td><td style="border-left: 1px solid black; padding-left: 5px;">1</td></tr> <tr><td style="padding-right: 5px;">28</td><td style="border-left: 1px solid black; padding-left: 5px;">1</td></tr> <tr><td style="padding-right: 5px;">14</td><td style="border-left: 1px solid black; padding-left: 5px;">0</td></tr> <tr><td style="padding-right: 5px;">7</td><td style="border-left: 1px solid black; padding-left: 5px;">1</td></tr> <tr><td style="padding-right: 5px;">3</td><td style="border-left: 1px solid black; padding-left: 5px;">1</td></tr> <tr><td style="padding-right: 5px;">1</td><td style="border-left: 1px solid black; padding-left: 5px;">0</td></tr> </table>	205	1	104	0	57	1	28	1	14	0	7	1	3	1	1	0
258	0																																		
129	1																																		
64	0																																		
32	0																																		
16	0																																		
8	0																																		
4	0																																		
2	0																																		
1	1																																		
205	1																																		
104	0																																		
57	1																																		
28	1																																		
14	0																																		
7	1																																		
3	1																																		
1	0																																		
$\rightarrow 100000010_2$	$\rightarrow 01101101_2$																																		

3.2.3 Massimo numero rappresentabile

Dato un insieme di m bit, il più grande numero naturale che si può rappresentare è:

$$N_{max} = 2^m - 1$$

Se, ad esempio, consideriamo un byte ($m = 8$), in cui ogni bit vale 1, sommeremo tra di loro tutti i pesi presenti:

$$\begin{aligned} 11111111_2 &= 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = \\ &= 255 = 256 - 1 = \\ &= 2^8 - 1 \end{aligned}$$

3.3 Numerazione ottale (OCT)

È una numerazione di tipo posizionale con base $R = 8$ e alfabeto $A = \{0, 1, 2, 3, 4, 5, 6, 7\}$, usata per la rappresentazione di numeri binari. La si trova usata in vecchie documentazioni, in quanto oggi è poco usata; tuttavia è utile conoscerla. Facciamo un esempio:

$$123_8 = 1 \cdot 8^2 + 2 \cdot 8^1 + 3 \cdot 8^0 = 64 + 16 + 3 = 83_{10}$$

La tabella di conversione binario-ottale è, per una sola cifra ottale, la seguente:

BIN	OCT
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Poichè la numerazione ottale ha come base una potenza del due, risulta immediata la conversione da binario ad ottale.

Infatti, dato un qualsiasi numero binario, per convertirlo in ottale basta dividerlo in gruppi di 3 cifre (partendo da destra) e sostituire ad ogni gruppo la cifra ottale corrispondente.

Qui di seguito alcuni esempi:

$$100100_2 = |100_2 | 100_2 | = 44_8$$

$$111111_2 = |111_2 | 111_2 | = 77_8$$

$$11001_2 = |011_2 | 001_2 | = 31_8$$

$$1101_2 = |001_2 | 101_2 | = 15_8$$

Un numero intero decimale può essere convertito in ottale anche utilizzando il metodo della divisione ripetuta già visto per la conversione decimale-binaria, eseguendo divisioni successive per 8.

Qui un esempio (convertiamo in ottale il numero 267_{10}):

$$\begin{array}{r|l} 267 & 3 & (267 : 8 = 33 + 3) \\ 33 & 1 & (33 : 8 = 4 + 1) \\ 4 & 4 & (4 : 8 = 0 + 4) \end{array}$$

Da cui risulta: 413_8 .

3.4 Numerazione esadecimale (HEX)

È una numerazione di tipo posizionale. La base è $R = 16$, mentre l'alfabeto è:

$$A = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}.$$

Per rappresentare le cifre di valore superiore al 9 si usano le prime sei lettere dell'alfabeto, in quanto disponibili su di una normale tastiera.

Qui di seguito una tabella di conversione DEC – BIN – HEX.

DEC	BIN	HEX
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Poiché anche la numerazione esadecimale ha come base una potenza del due, è possibile una conversione immediata BIN-HEX: si sostituisce ad ogni gruppo di 4 cifre binarie le corrispondenti cifre HEX. Alcuni esempi:

$$10001111_2 = |1000_2|1111_2| = 8F_{16} = 8F_H$$

$$11100011_2 = |1110_2|0011_2| = E3_{16} = E3_H$$

$$100101_2 = |0010_2|0101_2| = 25_{16} = 25_H$$

$$11100_2 = |0001_2|1100_2| = 1C_{16} = 1C_H$$

La conversione può essere eseguita anche in questo caso con la divisione ripetuta, dividendo per 16. Nell'esempio che segue, il numero 655_{10} è convertito in esadecimale:

$$\begin{array}{r|l}
 655 & 15 \\
 40 & 8 \\
 2 & 2
 \end{array}
 \quad
 \begin{array}{l}
 (655 : 16 = 40 + 15) \\
 (40 : 16 = 2 + 8) \\
 (2 : 16 = 0 + 2)
 \end{array}$$

Il numero risultante è: $28F_H$.

Per familiarizzare con i numeri esadecimali, può essere utile esaminare alcuni esempi di somme in colonna tra due numeri. Si faccia attenzione al fatto che si ha riporto da una colonna quando il valore della cifra supera il 15 (e non il 9, come nel decimale):

$$\begin{array}{r}
 28_H + \quad 22_H + \quad 40_H + \quad 0F_H + \quad 3A_H + \\
 33_H = \quad AA_H = \quad 51_H = \quad 0F_H = \quad 9B_H = \\
 \hline
 5B_H \quad CC_H \quad 91_H \quad 1E_H \quad D5_H
 \end{array}$$

3.5 Generalità sui codici binari

Quando si vuole rappresentare un numero in formato binario, la scelta più ovvia è utilizzare la numerazione binaria pura. Tale scelta comporta però lo svantaggio di una conversione decimale-binaria e viceversa, tanto più onerosa quanto più il numero è grande.

Per ovviare ai problemi di conversione è possibile rappresentare le cifre di un numero decimale ad una ad una, utilizzando opportuni codici, detti codici *BCD* (Binary Coded Decimal, decimale codificato in binario). Essi codificano le cifre decimali una alla volta e sono formati da 10 possibili combinazioni di 0 e 1 ad ognuna delle quali corrisponde una cifra decimale. Ovviamente le loro proprietà aritmetiche sono inferiori a quelle del binario puro.

In generale, i codici binari si dicono *pesati* quelli in cui ogni cifra ha un proprio peso a seconda della posizione che occupa e *non pesati* quelli in cui ogni combinazione di cifre associa un certo numero in modo arbitrario.

Si chiamano codici *riflessi* o *autocomplementanti* quelli in cui due numeri che sommati danno 9 sono uno il complemento a 1 dell'altro (cioè gli 1 sono scambiati con gli 0 e viceversa).

Vediamo ora alcuni tra i codici più usati:

- BCD 8421: è un codice pesato in cui le cifre hanno pesi rispettivamente 8, 4, 2, 1 (da sinistra a destra); è uguale al binario puro troncato a 1001_2 ;
- BCD 5421: è analogo al precedente, ma i pesi valgono 5, 4, 2, 1;
- AIKEN 2421: è un codice BCD pesato autocomplementante i cui pesi sono 2, 4, 2, 1.

N	BCD 8421	BCD 5421	AIKEN 2421
0	0000	0000	0000
1	0001	0001	0001
2	0010	0010	0010
3	0011	0011	0011
4	0100	0100	0100
5	0101	1000	1011
6	0110	1001	1100
7	0111	1010	1101
8	1000	1011	1110
9	1001	1100	1111

- BCD XS3: è un codice non pesato, autocomplementante che si ottiene sommando 3 (11 in binario) al BCD 8421; XS3 significa appunto excess 3 (eccesso 3).

N	BCD XS3
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

- GRAY: è un codice non pesato. E' caratterizzato dal fatto che ogni numero differisce dal precedente o dal successivo per una sola cifra. Il codice GRAY si può realizzare con qualunque numero di bit. Ci sono vari tipi di codice GRAY. Quello presentato è il codice di tipo "riflessivo".

N	GRAY
0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101
7	0100
8	1100
9	1101
10	1111
11	1110
12	1010
13	1011
14	1001
15	1000

3.6 Aritmetica binaria

3.6.1 Somma

Consideriamo la somma aritmetica tra due bit A e B . Valutiamo i 4 casi possibili:

$A +$	$0 +$	$0 +$	$1 +$	$1 +$
$B =$	$0 =$	$1 =$	$0 =$	$1 =$
<hr/>	<hr/>	<hr/>	<hr/>	<hr/>
<i>somma</i>	0	1	1	10_2

Nella colonna più a destra il risultato è ovviamente 2, ma conviene interpretarlo come “risultato 0, con il riporto di 1”. Denominato quindi S il bit della somma, introduciamo il bit C_o (*Carry Out*) che rappresenta il riporto generato. Nei casi in cui non si ha riporto, scriveremo che questo vale 0:

$$\begin{array}{rccccc}
 A + & 0 + & 0 + & 1 + & 1 + \\
 B = & 0 = & 1 = & 0 = & 1 = \\
 \hline
 C_o S & 00 & 01 & 01 & 10
 \end{array}$$

Quando sommiamo tra loro numeri binari codificati su più bit, l'eventuale riporto generato da una colonna deve essere sommato al risultato della colonna immediatamente a sinistra, come nei seguenti esempi:

$$\begin{array}{rcc}
 0010_2 + & 0001_2 + & 0011_2 + \\
 0001_2 = & 0001_2 = & 0001_2 = \\
 \hline
 0011_2 & 0010_2 & 0100_2
 \end{array}$$

Nel primo esempio non ci sono stati riporti; nel secondo c'è stato riporto dalla colonna più a destra; nell'ultimo esempio c'è stato riporto da entrambe le due colonne più a destra.

Sulla base di queste osservazioni, quindi, definiamo le **regole della somma**, per una data colonna, introducendo il riporto C_i (*Carry In*) proveniente dalla colonna affianco:

$$\begin{array}{rccccc}
 C_i + & 0 + & 0 + & 0 + & 0 + \\
 A + & 0 + & 0 + & 1 + & 1 + \\
 B = & 0 = & 1 = & 0 = & 1 = \\
 \hline
 C_o S & 00 & 01 & 01 & 10
 \end{array}$$

$$\begin{array}{rccccc}
 C_i + & 1 + & 1 + & 1 + & 1 + \\
 A + & 0 + & 0 + & 1 + & 1 + \\
 B = & 0 = & 1 = & 0 = & 1 = \\
 \hline
 C_o S & 01 & 10 & 10 & 11
 \end{array}$$

Qui di seguito alcuni esempi di somme tra numeri binari:

$$\begin{array}{rccccc}
 0001_2 + & 0110_2 + & 0111_2 + & 01011111_2 + & 01011110_2 + \\
 0111_2 = & 0110_2 = & 0111_2 = & 00100101_2 = & 00101111_2 = \\
 \hline
 1000_2 & 1100_2 & 1110_2 & 10000100_2 & 10001101_2
 \end{array}$$

Nell'effettuare la somma tra due numeri, può succedere che il risultato ecceda il massimo numero rappresentabile. Se, ad esempio, nella nostra rete logica abbiamo a disposizione solo 4 bit per rappresentare i dati, le seguenti operazioni generano un risultato troppo grande:

$$\begin{array}{r}
 1111_2 + \\
 0001_2 = \\
 \hline
 10000_2
 \end{array}
 \quad
 \begin{array}{r}
 0111_2 + \\
 1011_2 = \\
 \hline
 10010_2
 \end{array}
 \quad
 \begin{array}{r}
 0101_2 + \\
 1100_2 = \\
 \hline
 10001_2
 \end{array}
 \quad
 \begin{array}{r}
 1111_2 + \\
 1111_2 = \\
 \hline
 11110_2
 \end{array}$$

Per contenere il risultato occorrerebbero 5 bit: si è verificato un errore di *overflow* (traboccamento). Dopo avere effettuato una somma, occorre sempre controllare se si è verificato overflow, ossia se c'è stato riporto dalla colonna del bit più significativo. Questa regola vale per i numeri senza segno: vedremo in seguito come valutare la presenza di overflow con numeri che possono essere sia positivi che negativi.

3.6.2 Sottrazione

Con criteri simili a quelli ora visti per la somma, definiamo le *regole della sottrazione*. In questo caso, se necessario una colonna può chiedere un *prestito* alla colonna alla sua sinistra. Per ragioni di similitudine con i circuiti reali, lo denominiamo con la stessa sigla utilizzata per il riporto: C_i è il prestito che viene *richiesto* dalla colonna a destra, mentre C_o è il prestito la presente colonna *chiede* a quella sulla sinistra.

$$\begin{array}{r}
 C_i - \\
 A - \\
 B = \\
 \hline
 C_o S
 \end{array}
 \quad
 \begin{array}{r}
 0 - \\
 0 - \\
 0 = \\
 \hline
 00
 \end{array}
 \quad
 \begin{array}{r}
 0 - \\
 0 - \\
 1 = \\
 \hline
 11
 \end{array}
 \quad
 \begin{array}{r}
 0 - \\
 1 - \\
 0 = \\
 \hline
 11
 \end{array}
 \quad
 \begin{array}{r}
 0 - \\
 1 - \\
 1 = \\
 \hline
 10
 \end{array}$$

$$\begin{array}{r}
 C_i - \\
 A - \\
 B = \\
 \hline
 C_o S
 \end{array}
 \quad
 \begin{array}{r}
 1 - \\
 0 - \\
 0 = \\
 \hline
 01
 \end{array}
 \quad
 \begin{array}{r}
 1 - \\
 0 - \\
 1 = \\
 \hline
 00
 \end{array}
 \quad
 \begin{array}{r}
 1 - \\
 1 - \\
 0 = \\
 \hline
 00
 \end{array}
 \quad
 \begin{array}{r}
 1 - \\
 1 - \\
 1 = \\
 \hline
 11
 \end{array}$$

Ed ecco alcuni esempi di sottrazione:

$$\begin{array}{r}
 1001_2 - \\
 0011_2 = \\
 \hline
 0110_2
 \end{array}
 \quad
 \begin{array}{r}
 0111_2 - \\
 0101_2 = \\
 \hline
 0010_2
 \end{array}
 \quad
 \begin{array}{r}
 0110_2 - \\
 0001_2 = \\
 \hline
 0101_2
 \end{array}
 \quad
 \begin{array}{r}
 1111_2 - \\
 0111_2 = \\
 \hline
 1000_2
 \end{array}
 \quad
 \begin{array}{r}
 01011011_2 - \\
 00100101_2 = \\
 \hline
 00110110_2
 \end{array}$$

3.6.3 Prodotto

Le *regole del prodotto* sono le seguenti (non tratteremo della divisione):

$$\begin{aligned} 0 \times 0 &= 0 \\ 0 \times 1 &= 0 \\ 1 \times 0 &= 0 \\ 1 \times 1 &= 1 \end{aligned}$$

Qui alcuni esempi di prodotti (si eseguono nel modo in cui siamo abituati, ma applicando le regole ora viste). Si noti che nei risultati parziali dell'operazione o ricopiamo il moltiplicando (quando è moltiplicato per 1), o scriviamo tutti 0 (quando è moltiplicato per 0):

$$\begin{array}{r} 11 \times \\ \underline{11 =} \\ 11 \\ 11 - \\ \hline 1001 \end{array} \qquad \begin{array}{r} 11 \times \\ \underline{101 =} \\ 11 \\ 00 - \\ \underline{11 - -} \\ 1111 \end{array} \qquad \begin{array}{r} 110 \times \\ \underline{100 =} \\ 000 \\ 000 - \\ \underline{110 - -} \\ 11000 \end{array}$$

3.7 Aritmetica BCD 8421

In aritmetica BCD (8421) si eseguono le somme tenendo conto che ogni gruppo di 4 bit codifica una cifra decimale, per cui devono valere le regole dei riporti come nella rappresentazione decimale. Nel seguente esempio, nel calcolo in decimale sulla sinistra, non si verificano riporti dalle unità alle decine. Nel calcolo in binario, sulla destra, non si ha riporto dalla colonna delle unità, ed inoltre il risultato è composto ancora da cifre BCD (poiché ≤ 9), per cui il risultato è corretto:

$$\begin{array}{r} 24_{10} + \\ 42_{10} = \\ \hline 66_{10} \end{array} \qquad \begin{array}{r} 0010 \ 0100 + \\ 0100 \ 0010 = \\ \hline 0110 \ 0110 \end{array}$$

Invece, nel secondo esempio, sommando le cifre BCD non si ottengono altre cifre BCD, e il risultato non è quello atteso (vedi la somma in decimale sulla sinistra). Occorre quindi apportare degli aggiustamenti:

$$\begin{array}{r} 27_{10} + \\ 35_{10} = \\ \hline 62_{10} \end{array} \qquad \begin{array}{r} 0010 \ 0111 + \\ 0011 \ 0101 = \\ \hline 0101 \ 1100 \end{array}$$

Nel risultato, il numero 1100 che non è in BCD 8421 (supera il $9_{10} = 1001_2$). L'operazione di correzione da effettuare consiste nel sommare 0110 (cioè 6) al numero non BCD:

$$\begin{array}{r}
 0\ 1100 + \\
 0\ 0110 = \\
 \hline
 1\ 0010
 \end{array}$$

Le quattro cifre più a destra ora sono BCD. Va considerato il riporto e sommato alla cifra BCD immediatamente a sinistra. Nell'esempio:

$$\begin{array}{r}
 0101 + \\
 0001 = \\
 \hline
 0110
 \end{array}$$

Pertanto il risultato esatto della somma BCD, con le correzioni effettuate, è:

$$62_{10} = 01100010_{bcd}$$

3.8 Numeri razionali in binario.

Il modo di trattare i razionali binari è simile a quello usato per gli interi. Vediamo in particolare due esempi di conversione:

$$\begin{aligned}
 0.1011_2 &= 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} = \\
 &= 0.5 + 0.125 + 0.0625 = \\
 &= 0.6875_{10}.
 \end{aligned}$$

Per convertire DEC in BIN si opera nel modo seguente:

$$0.6875 \cdot 2 = 1.3750$$

La parte intera costituisce il numero binario, la parte decimale viene nuovamente moltiplicata per 2. Proseguendo:

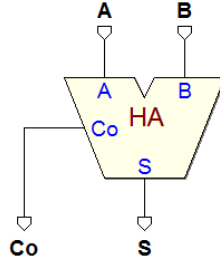
$$\begin{aligned}
 0.375 \cdot 2 &= 0.750 \\
 0.750 \cdot 2 &= 1.500 \\
 0.500 \cdot 2 &= 1.000
 \end{aligned}$$

Abbiamo ottenuto nuovamente il numero di partenza: 0.1011_2 .

3.9 Reti aritmetiche

3.9.1 Semi-sommatore (half adder)

Il semi-sommatore esegue la somma di due numeri binari di una cifra, generando in uscita risultato e riporto (dalla libreria di componenti di *Deeds*):



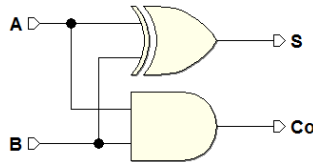
Utilizzando le regole della somma viste in precedenza, ne compiliamo la tavola di verità. Gli addendi sono A e B , la somma S ed il riporto C_o :

A	B	C_o	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Senza la necessità di utilizzare le mappe, si riconoscono in tabella le funzioni:

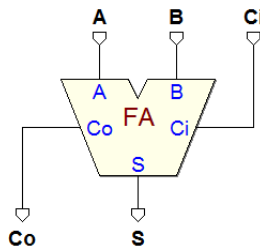
$$C_o = A \cdot B \quad \text{e} \quad S = A \oplus B$$

La rete logica del semi-sommatore avrà quindi il seguente aspetto:



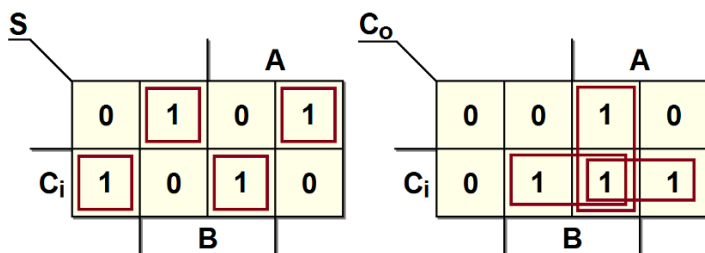
3.9.2 Sommatore (full adder)

Il sommatore estende le possibilità del semi-sommatore, aggiungendo in ingresso il riporto generato da uno stadio di somma precedente:



Ricaviamo la tavola di verità dalle regole della somma. A e B sono gli addendi, C_i è il riporto dallo stadio precedente, S è la somma e C_o il riporto generato:

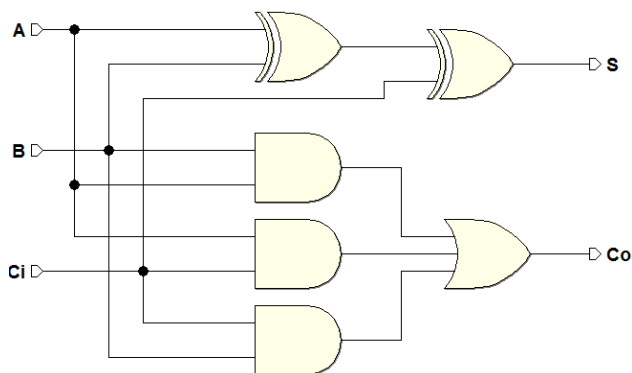
C_i	A	B	C_o	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Ne ricaviamo la sintesi:

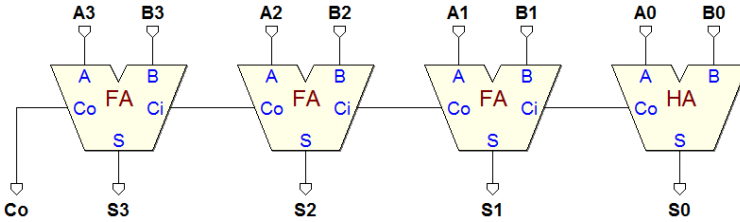
$$\begin{aligned}
 S &= \bar{A}\bar{B}C_i + \bar{A}B\bar{C}_i + A\bar{B}\bar{C}_i + ABC_i = \\
 &= \bar{A}(\bar{B}C_i + B\bar{C}_i) + A(\bar{B}\bar{C}_i + BC_i) = \\
 &= \bar{A}(B \oplus C_i) + A(\overline{B \oplus C_i}) = \\
 &= A \oplus (B \oplus C_i) \\
 C_o &= AB + AC_i + C_iB
 \end{aligned}$$

E quindi lo schema logico:



3.9.3 Sommatore con riporto in cascata

Il collegamento in figura permette di eseguire la somma di due numeri di 4 bit ed è estensibile a qualunque numero a m bit:



A causa della propagazione dei segnali, il tempo di esecuzione di una somma è proporzionale a m :

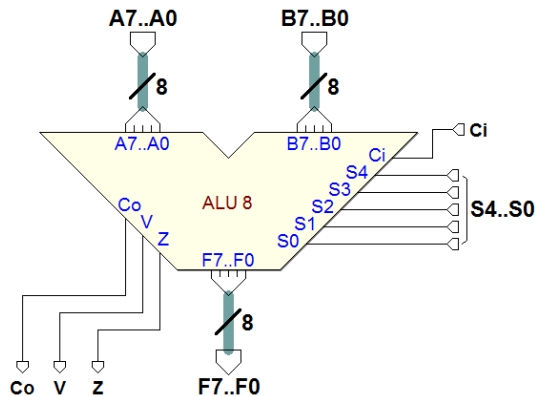
$$T_s = (m - 1) \cdot t_{fa} + t_{ha}$$

dove t_{fa} è il ritardo di propagazione dei sommatore (FA), e t_{ha} quello del semi-sommatore (HA). Infatti il C_o risulta valido solo dopo che tutti i riporti intermedi si sono propagati lungo la rete.

A causa di questo, quando è richiesta una grande velocità di calcolo su di un numero elevato di bit, è preferibile fare ricorso ad architetture più efficienti, dove i riporti non sono calcolati in cascata, ma in parallelo, come ad esempio nelle reti *Carry Look Ahead* (non esaminate in questo libro).

3.9.4 Unità aritmetico-logica (ALU, Arithmetic Logic Unit)

La ALU è una rete combinatoria che permette di eseguire diverse operazioni su due numeri binari (A e B). In figura, un esempio di ALU ad 8 bit (dalla libreria del simulatore *Deeds*):



Gli operandi sono $A_{7..0}$ e $B_{7..0}$, mentre il risultato è prelevato sulle uscite $F_{7..0}$. Per le operazioni di somma e sottrazione, sono presenti un ingresso e una uscita di riporto, rispettivamente C_i e C_o .

Sono disponibili anche le uscite V e Z : $V = 1$ indica che l'operazione aritmetica ha dato origine ad un *overflow*, mentre è riportato $Z = 1$ se il risultato è zero. Le operazioni sono impostate tramite il gruppo di ingressi $S4..0$, secondo la tabella riportata qui sotto:

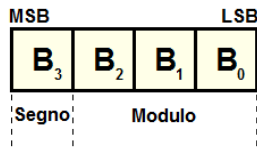
$S4..S0$	<i>Funzione</i>	<i>Note</i>
00000	$F = 0$	
00001	$F = +1$	
00010	$F = -1$	
00011	$F = -128$	Minimo negativo
00100	$F = A$	
00101	$F = B$	
00110	$F = \bar{A}$	Complemento a uno di A
00111	$F = \bar{B}$	Complemento a uno di B
01000	$F = A \text{ and } B$	(bit a bit)
01001	$F = A \text{ and } \bar{B}$	(bit a bit)
01010	$F = \bar{A} \text{ and } B$	(bit a bit)
01011	$F = \bar{A} \text{ or } \bar{B}$	(bit a bit)
01100	$F = A \text{ or } B$	(bit a bit)
01101	$F = A \text{ or } \bar{B}$	(bit a bit)
01110	$F = \bar{A} \text{ or } B$	(bit a bit)
01111	$F = \overline{A \text{ and } B}$	(bit a bit)
10000	$F = A \oplus B$	(bit a bit)
10001	$F = \overline{A \oplus B}$	(bit a bit)
10010	$F = \bar{A} + 1$	Complemento a due di A
10011	$F = \bar{B} + 1$	Complemento a due di B
10100	$F = A + 1$	Incremento di A
10101	$F = B + 1$	Incremento di B
10110	$F = A - 1$	Decremento di A
10111	$F = B - 1$	Decremento di B
11000	$F = A + B$	Addizione
11001	$F = A + B + C_i$	Addizione, con riporto in entrata
11010	$F = \text{sat}(A + B)$	Addizione, con saturazione
11011	$F = A - B$	Sottrazione
11100	$F = A - B - C_i$	Sottrazione, con prestito in entrata
11101	$F = \text{sat}(A - B)$	Sottrazione, con saturazione
11110	$F = B - A$	Sottrazione rovesciata
11111	$F = B - A - C_i$	Sottr. rovesciata, con prestito in entrata

3.10 Numeri relativi in binario

Esistono diversi metodi per rappresentare i numeri relativi (numeri con segno) in binario. In generale, i numeri positivi sono rappresentati come se fossero senza segno; i vari metodi si differenziano invece per come sono codificati quelli negativi. Al variare della codifica utilizzata, cambiano anche le regole per gestire le operazioni che coinvolgono i numeri negativi.

3.10.1 Rappresentazione in codice “modulo e segno”

Consideriamo ad esempio un pacchetto di 4 bit: si usa il bit più significativo (MSB) per il segno, il resto per il modulo:



Codifichiamo il segno $-$ con 1, e il segno $+$ con 0. Otteniamo la seguente tabella:

B_3	B_2	B_1	B_0	Dec
0	1	1	1	+7
0	1	1	0	+6
0	1	0	1	+5
0	1	0	0	+4
0	0	1	1	+3
0	0	1	0	+2
0	0	0	1	+1
0	0	0	0	+0
1	0	0	0	-0 (!)
1	0	0	1	-1
1	0	1	0	-2
1	0	1	1	-3
1	1	0	0	-4
1	1	0	1	-5
1	1	1	0	-6
1	1	1	1	-7

Questa rappresentazione presenta alcuni inconvenienti:

1. lo zero ha due diversi codici: 0000 e 1000
2. possiede cattive proprietà aritmetiche.

Come dimostrano gli esempi seguenti, non si può usare un normale sommatore:

$$\begin{array}{r} 3_{10} + \\ 4_{10} = \\ \hline 7_{10} \end{array} \quad \begin{array}{r} 0011_2 + \\ 0100_2 = \\ \hline 0111_2 \quad (\text{corretto}) \end{array} \quad \begin{array}{r} 4_{10} + \\ 4_{10} = \\ \hline -0_{10} \end{array} \quad \begin{array}{r} 0100_2 + \\ 0100_2 = \\ \hline 1000_2 \quad (\text{overflow}) \end{array}$$

$$\begin{array}{r} 7_{10} + \\ -2_{10} = \\ \hline 5_{10} \end{array} \quad \begin{array}{r} 0111_2 + \\ 1010_2 = \\ \hline 10001_2 \quad (\text{riporto}) \end{array}$$

In rappresentazione modulo più segno, quindi, occorrerebbe usare un sommatore più complesso, progettato in funzione del codice.

3.10.2 Complementazione

Complemento alla “base”

Sia il numero N in base R rappresentato in notazione posizionale con m cifre:

$$N = n_{m-1} \cdot R^{m-1} + n_{m-2} \cdot R^{m-2} + \dots + n_1 \cdot R^1 + n_0 \cdot R^0$$

Si definisce complemento alla base R del numero N :

$$C_R(N) = R^m - N$$

Consideriamo ora un altro numero Q , rappresentato con la stessa base e lo stesso numero di cifre. Sommando il numero Q a $C_R(N)$ si ottiene la differenza tra i due, più il termine R^m , che rappresenta un riporto al di fuori del formato del numero:

$$Q + C_R(N) = Q + (R^m - N) = (Q - N) + R^m$$

Portando a sinistra la differenza $Q - N$:

$$Q - N = Q + C_R(N) - R^m$$

Vediamo ora un esempio con i numeri decimali. $R = 10$, per cui il complemento è “a dieci”; utilizziamo due sole cifre ($m = 2$), e ipotizziamo $Q = 48_{10}$ e $N = 12_{10}$.

$$48_{10} - 12_{10} = 48_{10} + C_{10}(12_{10}) - 10^2$$

Applicando la definizione di complemento alla base al nostro esempio:

$$C_{10}(12_{10}) = 10^2 - 12 = 88$$

Otteniamo:

$$48_{10} - 12_{10} = 48_{10} + 88_{10} - 10^2$$

Nonostante l'apparente complicazione, abbiamo ottenuto un grande vantaggio: il numero negativo è stato sostituito dal suo complemento a dieci, che è positivo. Togliere poi 10^2 dal risultato è, come vedremo tra poco, molto semplice e non richiede di effettuare una sottrazione.

In una rete aritmetica potremo fare a meno dei sottrattori e utilizzare soltanto dei sommatore, a patto di saper calcolare facilmente il complemento alla base di un numero. Si noti che l'operazione di complemento equivale, dal punto di vista del calcolo, al cambiamento del segno del numero.

Torniamo al nostro esempio ed eseguiamo in colonna la somma:

$$\begin{array}{r|l} & 48 + \\ & 88 = \\ \hline \text{riporto} & 1 \mid 36 \end{array}$$

Togliere 10^2 è, a questo punto, banale: è sufficiente non considerare il riporto. Il risultato è quello atteso: $36_{10} = 48_{10} - 12_{10}$.

Si faccia attenzione che è necessario rappresentare tutti i numeri in gioco con lo stesso m . Per esempio, utilizziamo le stesse quantità di prima, ma rappresentate su 8 cifre ($m = 8$):

$$C_{10}(12_{10}) = 10^8 - 12_{10} = 99999988_{10}$$

$$\begin{array}{r|l} & 00000048 + \\ & 99999988 = \\ \hline \text{riporto} & 1 \mid 00000036 \end{array}$$

Il riporto è al di fuori delle m cifre: togliere R^m dal risultato significa semplicemente ignorarlo.

Consideriamo adesso i numeri binari, con $R = 2$. Prendiamo in considerazione numeri su 4 bit ($M = 4$) e valutiamo il complemento "a due" di $N = 0101_2 = 5_{10}$, dalla definizione:

$$C_2(0101_2) = 2^m - N = 10000_2 - 0101_2 = 1011_2$$

Come vedremo qui sotto, il $C_2(N)$ si può calcolare in modo più veloce utilizzando il complemento alla "base meno uno".

Complemento alla "base meno uno"

Si definisce *complemento alla base meno uno* del numero N :

$$C_{R-1}(N) = (R^m - 1) - N$$

Confrontiamo la definizione con quella del complemento alla base. Risulta:

$$C_R(N) = C_{R-1}(N) + 1$$

Calcolare il complemento alla base meno uno è più semplice che calcolare il complemento alla base: per ottenere quest'ultimo si preferisce calcolare dapprima il complemento alla base meno uno e poi sommare uno.

Facciamo un esempio in decimale, con $N = 12_{10}$, rappresentando il numero con 8 cifre. Calcoliamo quindi il complemento a nove (= "base dieci meno uno"), valutando prima il termine:

$$(R^m - 1) = 10^8 - 1 = 99999999$$

Questo numero è semplicemente composto dalla cifra "9" ripetuta m volte. Il complemento a nove di 12 sarà:

$$C_9(12) = 99999999 - 00000012 = 99999987$$

Il calcolo è più semplice del complemento a dieci, perché nel sottrarre il numero non si verifica mai la necessità di chiedere prestiti. Sommando poi 1 al risultato, si ottiene il complemento a dieci.

Riprendiamo l'esempio del numero binario $N = 0101_2 = 5_{10}$ visto in precedenza ($M = 4$). Per valutarne il complemento a uno (= "base due meno uno"), calcoliamo dapprima il termine:

$$(R^m - 1) = 2^m - 1 = 1000_2 - 1 = 1111_2$$

La considerazione è simile a quella precedente: questo numero è composto dalla cifra "1" ripetuta m volte. Il complemento a uno di 0101_2 sarà quindi:

$$C_1(0101_2) = 1111_2 - 0101_2$$

Per una maggiore evidenza, eseguiamo la sottrazione in colonna:

$$\begin{array}{r} 1111_2 - \\ 0101_2 = \\ \hline C_1(0101_2) \quad 1010_2 \end{array}$$

Osserviamo che nel sottrarre il numero da un numero composto di soli "1" non vi è la necessità di chiedere prestiti. Anzi, per calcolare il risultato è sufficiente sostituire tutti gli "1" con degli "0", e viceversa. Da un punto di vista circuitale, questo significa semplicemente negare tutti i bit di cui il numero N è composto.

$$C_1(N) = \overline{N}$$

Per quanto visto prima, sommando poi +1 al numero ottenuto, si ottiene il complemento a due:

$$C_2(N) = C_1(N) + 1 = \overline{N} + 1$$

3.10.3 Rappresentazione in codice “complemento a uno”

Si può scegliere di rappresentare i numeri binari negativi mediante il complemento a uno (C_1). Nella tabella seguente ne vediamo un esempio, per numeri di 4 bit. I positivi sono codificati in binario puro, lasciando a 0 la cifra più significativa; i negativi sono il C_1 del corrispondente numero positivo, calcolati con il criterio appena visto:

B_3	B_2	B_1	B_0	Dec
0	1	1	1	+7
0	1	1	0	+6
0	1	0	1	+5
0	1	0	0	+4
0	0	1	1	+3
0	0	1	0	+2
0	0	0	1	+1
0	0	0	0	+0
1	1	1	1	-0 (!)
1	1	1	0	-1
1	1	0	1	-2
1	1	0	0	-3
1	0	1	1	-4
1	0	1	0	-5
1	0	0	1	-6
1	0	0	0	-7

Per valutare il codice di un numero negativo, come visto sopra, basta invertire tutti i bit del numero positivo corrispondente; un esempio:

$$-3_{10} = C_1(3_{10}) = C_1(0011_2) = 1100_2$$

Proprietà aritmetiche

- Somma di due positivi:

$$\begin{array}{r} +2 + \quad 0010 + \\ +5 = \quad 0101 = \\ \hline +7 \quad \quad 0111 \end{array}$$

- Somma di positivo e negativo con risultato positivo:

$$\begin{array}{r} +5 + \quad 0101 + \\ -2 = \quad 1101 = \\ \hline +3 \quad \quad 10010 \quad (\text{“End Around Carry”}) \rightarrow 0010 + 1 = 0011 \end{array}$$

- Somma di positivo e negativo con risultato negativo:

$$\begin{array}{r} -5 + \quad 1010 + \\ +2 = \quad 0010 = \\ \hline -3 \quad \quad 1100 \end{array}$$

- Somma di due negativi:

$$\begin{array}{r}
 -3 + \quad 1100 + \\
 -3 = \quad 1100 = \\
 \hline
 -6 \quad \quad 11000 \quad (\text{"End Around Carry"}) \rightarrow 1000 + 1 = 1001
 \end{array}$$

Il riporto “fuori dal numero” deve essere sommato al bit meno significativo del risultato. Chiamiamo C_P il riporto nel bit più significativo; C_S il riporto al di fuori dal numero.

Consideriamo la somma tra due numeri $Q + N$; la seguente tabella ci mostra i valori di C_P e C_S al variare del segno degli addendi e del risultato, nel caso in cui quest’ultimo risulti corretto (cioè quando il formato del numero riesce a contenere il risultato):

Q	N	Somma	C_P	C_S
+	+	+	0	0
+	-	+	1	1
+	-	-	0	0
-	-	-	1	1

Se C_P e C_S risultano diversi, si ha errore di *overflow*, e possiamo valutarlo come $OVF = C_P \oplus C_S$. Questo un esempio di errore di overflow:

$$\begin{array}{r}
 +7 + \quad 0111 + \\
 +1 = \quad 0001 = \\
 \hline
 +8 \quad \quad 1000 \quad C_P = 1, C_S = 0, \rightarrow OVF
 \end{array}$$

3.10.4 Rappresentazione in codice “complemento a due”

Il metodo più usato per rappresentare i numeri binari negativi fa ricorso al complemento a due (C_2). Nella tabella sulla destra troviamo un esempio di codifica per numeri di 4 bit.

Anche qui i positivi sono codificati in binario puro, lasciando a 0 la cifra più significativa, mentre i negativi sono calcolati come complemento a due del corrispondente positivo.

B_3	B_2	B_1	B_0	Dec
0	1	1	1	+7
0	1	1	0	+6
0	1	0	1	+5
0	1	0	0	+4
0	0	1	1	+3
0	0	1	0	+2
0	0	0	1	+1
0	0	0	0	+0
1	1	1	1	-1
1	1	1	0	-2
1	1	0	1	-3
1	1	0	0	-4
1	0	1	1	-5
1	0	1	0	-6
1	0	0	1	-7
1	0	0	0	-8

Per esercizio proviamo a calcolare il numero -1_{10} partendo dal $+1_{10}$:

$$C_2(0001) = C_1(0001) + 1 = 1110 + 1 = 1111$$

Se calcoliamo il C_2 dello zero, otteniamo nuovamente zero (lo zero in C_2 ha una *rappresentazione univoca*). Notiamo che l'operazione di C_2 cambia il segno del numero: se N è positivo, il $C_2(N)$ è negativo, e viceversa.

Inoltre, se effettuiamo il $C_2(C_2(N))$, otteniamo nuovamente il numero N :

$$C_2(C_2(N)) = 2^m - (C_2(N)) = 2^m - (2^m - N) = N$$

Proprietà aritmetiche

Consideriamo anche qui il riporto nel bit più significativo (C_P) e il riporto al di fuori dal numero (C_S). Valgono anche per la rappresentazione in C_2 le stesse considerazioni fatte per quella in C_1 . Per la somma tra i due numeri Q e N , vale la tabella già vista nel caso di assenza di errore di *overflow*:

Q	N	Somma	C_P	C_S
+	+	+	0	0
+	-	+	1	1
-	+	-	0	0
-	-	-	1	1

Anche qui si overflow se C_P e C_S risultano diversi: $OVF = C_P \oplus C_S$.

Il codice C_2 permette di eseguire la somma indipendentemente dal segno degli addendi, utilizzando un normale sommatore binario, senza la necessità di effettuare correzioni (come invece sono richieste nel C_1).

3.10.5 Estensione del segno

Un numero binario con segno, rappresentato in codice C_2 (o in C_1), su m bit, può essere esteso ad un numero maggiore di bit $v > m$, a patto di conservarne il segno e il valore. Consideriamo un numero positivo, ad esempio 6_{10} , rappresentato su 4 bit, e il suo corrispondente negativo in C_2 :

$$6_{10} = 0110_2 \qquad -6_{10} = 1010_2$$

Prendiamo in esame quello positivo e consideriamo il segno come facente parte integrante del numero. In notazione posizionale:

$$0110_2 = 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$$

Se lo rappresentiamo su 8 bit, in notazione posizionale scriviamo:

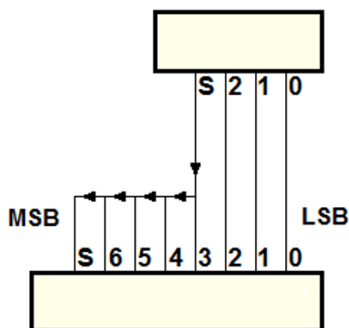
$$0 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 00000110_2$$

Abbiamo di fatto aggiunto degli zeri non significativi alla sinistra del numero, come era prevedibile. Tuttavia, questo metodo non può funzionare per i numeri negativi: in prima battuta è evidente che ne cambieremmo il segno in positivo, ma non solo, ne cambieremmo anche il valore!

Valutiamo quindi il $C_2(6_{10})$, ma rappresentato con 8 bit:

$$C_2(00000110_2) = C_1(00000110_2) + 1 = 11111001_2 + 1 = 11111010_2$$

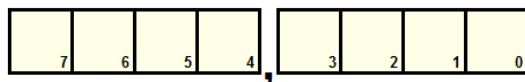
Come si vede, per estendere su più bit un numero negativo, è sufficiente aggiungere sulla sinistra degli 1 (invece che degli 0). In altre parole, occorre aggiungere sulla sinistra delle cifre di valore uguale al segno. Dal punto di vista circuitale, l'estensione del segno si traduce in una rete molto semplice:



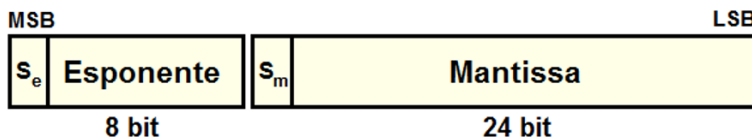
3.11 Rappresentazione di numeri reali

Esistono sostanzialmente due metodi per trattare i numeri reali nell'aritmetica binaria.

- *Virgola fissa (fixed point)*: si riserva un certo numero di bit (nell'esempio: 4), per la parte intera del numero e altri 4 per la sua parte frazionaria.



- *Virgola mobile (floating point)*: questo metodo utilizza un certo numero di bit, ad esempio 32, così suddivisi:



In figura, S_e è il segno dell'esponente, S_m quello della mantissa. Esiste una rappresentazione "normalizzata" in cui la mantissa è di tipo $0, \dots$ e l'esponente del tipo $2^{E_{sp}}$; ad esempio: $-0,10110001011011_2 * 2^{00101111_2}$.

3.12 Codici alfanumerici

Un codice alfanumerico consente di rappresentare le lettere maiuscole e minuscole, le dieci cifre decimali, i simboli di interpunzione e i cosiddetti “simboli speciali”. Esistono codici che consentono di rappresentare tutte le scritture del mondo, comprese, ad esempio, i caratteri cinesi e giapponesi. Il codice di questo tipo più diffuso è l’Unicode¹. Questo tipo di codice è molto complesso e non è organizzato soltanto su semplici tabelle di corrispondenza, ma su librerie di software supportate dai moderni ambienti per lo sviluppo di applicazioni su computer, tablet e simili, e non sarà trattato in queste pagine.

Un codice ancora piuttosto usato, e relativamente semplice, è il codice *ASCII*², che nella versione standard ha 7 bit significativi. Codifica le 26 lettere maiuscole e minuscole della lingua inglese, le 10 cifre decimali, le interpunzioni e i simboli usati in tale lingua.

Codifica inoltre anche un certo numero di codici di comunicazione, i cosiddetti caratteri “non stampabili”, utilizzati ormai solo in parte nei sistemi moderni. Riportiamo qui di seguito le tabelle del codice ASCII.

ASCII: caratteri non stampabili					
Dec	Code	Descrizione	Dec	Code	Descrizione
0	NULL	(Null character)	16	DLE	(Data link escape)
1	SOH	(Start of Header)	17	DC1	(Device control 1)
2	STX	(Start of Text)	18	DC2	(Device control 2)
3	ETX	(End of Text)	19	DC3	(Device control 3)
4	EOT	(End of Transmiss	20	DC4	(Device control 4)ion)
5	ENQ	(Enquiry)	21	NAK	(Negative acknowledgement)
6	ACK	(Acknowledgemen	22	SYN	(Synchronous idle)t)
7	BEL	(Bell)	23	ETB	(End of transmission block)
8	BS	(Backspace)	24	CAN	(Cancel)
9	HT	(Horizontal Tab)	25	EM	(End of medium)
10	LF	(Line feed)	26	SUB	(Substitute)
11	VT	(Vertical Tab)	27	ESC	(Escape)
12	FF	(Form feed)	28	FS	(File separator)
13	CR	(Carriage return)	29	GS	(Group separator)
14	SO	(Shift Out)	30	RS	(Record separator)
15	SI	(Shift In)	31	US	(Unit separator)

¹ <http://www.unicode.org>

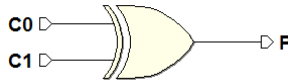
² American standard Code for Information Interchange

ASCII: caratteri stampabili					
Dec	Code	Descrizione	Dec	Code	Descrizione
32		Space	80	P	Capital P
33	!	Exclamation mark	81	Q	Capital Q
34	"	Quotation mark	82	R	Capital R
35	#	Number sign	83	S	Capital S
36	\$	Dollar sign	84	T	Capital T
37	%	Percent sign	85	U	Capital U
38	&	Ampersand	86	V	Capital V
39	'	Apostrophe	87	W	Capital W
40	(round brackets	88	X	Capital X
41)	round brackets	89	Y	Capital Y
42	*	Asterisk	90	Z	Capital Z
43	+	Plus sign	91	[square brackets
44	,	Comma	92	\	Backslash
45	-	Hyphen	93]	square brackets
46	.	Dot	94	^	Circumflex accent
47	/	Slash	95	_	underscore
48	0	number zero	96	`	Grave accent
49	1	number one	97	a	Lowercase a
50	2	number two	98	b	Lowercase b
51	3	number three	99	c	Lowercase c
52	4	number four	100	d	Lowercase d
53	5	number five	101	e	Lowercase e
54	6	number six	102	f	Lowercase f
55	7	number seven	103	g	Lowercase g
56	8	number eight	104	h	Lowercase h
57	9	number nine	105	i	Lowercase i
58	:	Colon	106	j	Lowercase j
59	;	Semicolon	107	k	Lowercase k
60	<	Less-than sign	108	l	Lowercase l
61	=	Equals sign	109	m	Lowercase m
62	>	Greater-than sign	110	n	Lowercase n
63	?	Question mark	111	o	Lowercase o
64	@	At sign	112	p	Lowercase p
65	A	Capital A	113	q	Lowercase q
66	B	Capital B	114	r	Lowercase r
67	C	Capital C	115	s	Lowercase s
68	D	Capital D	116	t	Lowercase t
69	E	Capital E	117	u	Lowercase u
70	F	Capital F	118	v	Lowercase v
71	G	Capital G	119	w	Lowercase w
72	H	Capital H	120	x	Lowercase x
73	I	Capital I	121	y	Lowercase y
74	J	Capital J	122	z	Lowercase z
75	K	Capital K	123	{	curly brackets
76	L	Capital L	124		vertical-bar
77	M	Capital M	125	}	curly brackets
78	N	Capital N	126	~	Tilde ; swung dash
79	O	Capital O	127	DEL	Delete

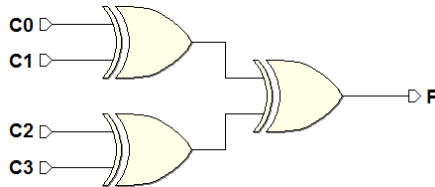
3.13 Codici a rivelazione di errore: generatore e rivelatore di parità

Supponiamo di utilizzare in un sistema ad **8 bit** il codice **ASCII standard** (che ha solo **7 bit** significativi, che indichiamo qui come $C_6 \dots C_0$). È possibile allora sfruttare l'ottavo bit per creare un codice “a rivelazione di errore”, come quello, molto usato, che fa ricorso al cosiddetto “bit di parità”: una informazione aggiuntiva, ridondante ai fini del codice **ASCII**, ma utile per controllare l'integrità dei dati.

Una porta **EXOR** segnala in uscita, con $P = 1$, la presenza di un numero dispari di **1** in ingresso. Si può quindi considerare un “rivelatore di parità”:



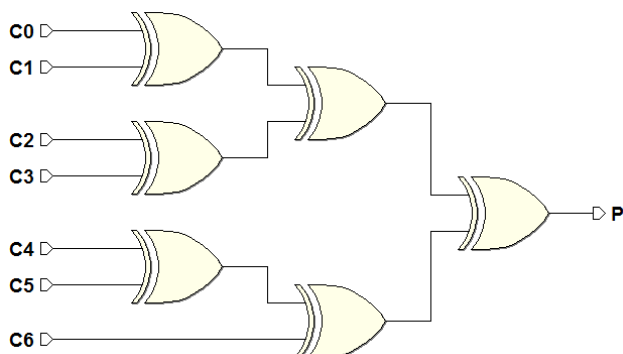
Realizzando una struttura ad albero si può estendere il rivelatore di parità ad un numero qualunque di ingressi:



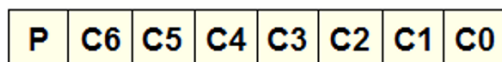
La tavola di verità dell'albero di **EXOR** a **4** ingressi presenterà un **1** per ogni combinazione con un numero dispari di **1**:

C3	C2	C1	C0	P
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Nella figura seguente, un rivelatore di parità a 7 bit processa i bit $C_6 \dots C_0$ del dato, producendo sulla sua uscita P un **1** se l'insieme ha un numero dispari di **1** (odd parity). La struttura è denominata “**generatore di parità**” (parity generator):

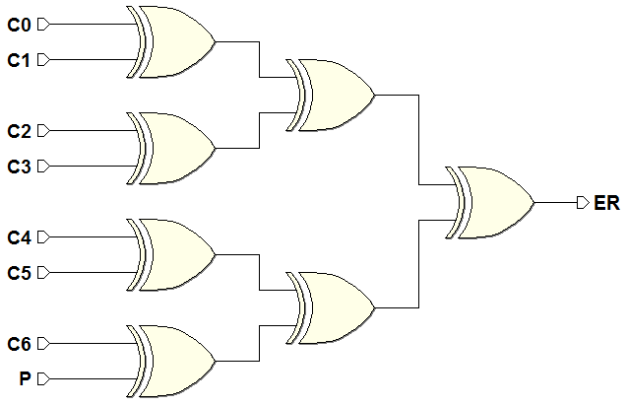


Il dato a 8 bit composto da P e da $C_6 \dots C_0$ ha quindi un numero pari di **1** (even parity). Nell'insieme dei bit del dato, P è chiamato “**bit di parità**” (parity bit):



L'associazione al dato originale di una “ridondanza” (nel nostro caso il bit di parità), consente di trasmetterlo da un sistema ad un altro fornendo al ricevitore uno strumento per effettuare un controllo sulla qualità del dato ricevuto. Infatti la distanza tra i sistemi, le caratteristiche del canale di comunicazione e la presenza di disturbi (rumore) degradano la qualità del segnale lungo il suo percorso, alterandone il contenuto: alcuni bit potranno essere ricevuti sbagliati. In un sistema reale, l'errore sarà sempre presente; non si può annullare ma, adottando le corrette tecniche, si può fare in modo che la probabilità che si verifichi sia molto ridotta. L'entità del tasso di errore può essere studiata in termini statistici.

Dal lato del ricevitore introduciamo una struttura analoga, allo scopo di verificare che la parità del dato sia conservata. Ricalcoliamo la parità dei bit ricevuti $C_0 \dots C_7$, ma includendo anche il bit P (vedi figura seguente).



Nel caso in cui non si sia verificato nessun errore, l'uscita **ER (Errore)** sarà a zero. Invece, se **ER** risulta a **1**, significa che la parità non si è conservata, a causa di un errore. Si noti che il risultato della verifica è corretto solo se l'errore riguarda **un solo bit**: infatti, se i bit errati sono **due**, ad esempio, **ER** non lo segnala.

Può essere vantaggiosamente utilizzato quindi solo quando la probabilità che si verifichi un errore è bassa. Infatti, ipotizzando che questa sia di un bit sbagliato ogni 10^4 trasmessi, la probabilità del doppio errore contemporaneo risulterà pari a uno su 10^8 bit (il prodotto delle due probabilità).

3.14 Esercizi

3.14.1 Numerazione Binaria

1. Scrivere 168_{10} e 143_{10} come numeri binari.
2. Calcolare il valore decimale dei seguenti numeri binari:
 - a) 1110111_2
 - b) 101011101001011_2
3. Eseguire i seguenti calcoli:
 - a) $11001101_2 + 1010101_2$
 - b) $1011011_2 - 101110_2$
 - c) $1001001_2 - 10101_2$
4. Moltiplicare i numeri binari:
 - a) 1110_2 e 1011_2
 - b) 1011_2 e 101_2
 - c) 11100_2 e 011_2
 - d) 110_2 e 1111_2

3.14.2 Numeri binari con segno

1. Eseguire i calcoli in complemento a 2 con 7 bit:
 - a) $15 + 11$
 - b) $15 + (-11)$
 - c) $15 - 11$
 - d) $4 - 11$
 - e) $29 + 17$
 - f) $29 - 17$
 - g) $29 + (-17)$
 - h) $(-11) - 29$
 - i) $8 + (-18)$
 - j) $7 - 17$
2. Eseguire i calcoli in complemento a 2, usando in ciascun problema il minimo numero di bit per evitare overflow:
 - a) $3 + (-7)$
 - b) $-3 + 7$
 - c) $11 + (-11)$
 - d) $18 - (-3)$

3.14.3 Numerazione Ottale e Esadecimale

1. Convertire i seguenti numeri dal codice ottale all'esadecimale:
 - a) 276534_8
 - b) 22017555724_8
2. Convertire i decimali in binario ed esadecimale:
 - a) 7722_{10}
 - b) 1435_{10}
3. Convertire i seguenti decimali nei sistemi ottale ed esadecimale:
 - a) 36625_{10}
 - b) 124_{10}
4. Calcolare il valore decimale dei seguenti numeri esadecimali:
 - a) $1A2B07_{16}$
 - b) 11047_{16}
5. Calcolare i valori decimali dei seguenti numeri ottali.
 - a) 3111_8
 - b) 276534_8
6. Eseguire i calcoli sui seguenti numeri esadecimali.
 - a) $41AB7_{16} + C2D6F_{16}$
 - b) $A23CE_{16} + 363E6_{16}$

3.15 Soluzioni

3.15.1 Numerazione Binaria

$$\begin{array}{r|l}
 168 & 0 \\
 84 & 0 \\
 42 & 0 \\
 21 & 1 \\
 10 & 0 \\
 5 & 1 \\
 2 & 0 \\
 1 & 1 \rightarrow 10101000_2
 \end{array}
 \qquad
 \begin{array}{r|l}
 143 & 1 \\
 71 & 1 \\
 35 & 1 \\
 17 & 1 \\
 8 & 0 \\
 4 & 0 \\
 2 & 0 \\
 1 & 1 \rightarrow 10001111_2
 \end{array}$$

2. a) $1110111_2 = 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + = 119_2$
 b) $101011101001011_2 = 22347_{10}$

3. a)
$$\begin{array}{r}
 11001101 \quad + \\
 1010101 \quad = \\
 \hline
 100100010
 \end{array}$$

b)
$$\begin{array}{r}
 1011011 \quad - \\
 101110 \quad = \\
 \hline
 0101101
 \end{array}$$

c)
$$\begin{array}{r}
 1001001 \quad - \\
 10101 \quad = \\
 \hline
 110100
 \end{array}$$

4. a)
$$\begin{array}{r}
 1110 \quad \times \\
 1011 \quad = \\
 \hline
 1110 \\
 11100 \\
 000000 \\
 1110000 \\
 \hline
 10011010
 \end{array}$$

b)
$$\begin{array}{r}
 1011 \quad \times \\
 101 \quad = \\
 \hline
 1011 \\
 00000 \\
 101100 \\
 \hline
 110111
 \end{array}$$

$$\begin{array}{r}
 11100 \times \\
 00011 = \\
 \hline
 11100 \\
 c) \quad 111000 \\
 0000000 \\
 00000000 \\
 000000000 \\
 \hline
 1010100
 \end{array}$$

$$\begin{array}{r}
 110 \times \\
 1111 = \\
 \hline
 110 \\
 d) \quad 1100 \\
 11000 \\
 110000 \\
 \hline
 1011010
 \end{array}$$

3.15.2 Numeri binari con segno

$$1. \ a) \quad \begin{array}{r}
 15 + \quad 0001111 + \\
 11 = \quad 0001011 = \\
 \hline
 26 \quad 0011010
 \end{array}$$

$$b) \quad \begin{array}{r}
 15 + \quad 0001111 + \\
 -11 = \quad 1110101 = \\
 \hline
 4 \quad 0000100
 \end{array}$$

$$c) \quad \begin{array}{r}
 15 - \quad 0001111 - \\
 11 = \quad 0001011 = \\
 \hline
 4 \quad 0000100
 \end{array}$$

$$d) \quad \begin{array}{r}
 4 - \quad 0000100 - \\
 11 = \quad 0001011 = \\
 \hline
 -7 \quad 1111001
 \end{array}$$

$$e) \quad \begin{array}{r}
 29 + \quad 0011101 + \\
 17 = \quad 0010001 = \\
 \hline
 46 \quad 0101110
 \end{array}$$

$$f) \quad \begin{array}{r}
 29 - \quad 0011101 - \\
 17 = \quad 0010001 = \\
 \hline
 12 \quad 0001100
 \end{array}$$

$$g) \quad \begin{array}{r}
 29 + \quad 0011101 + \\
 -17 = \quad 1101111 = \\
 \hline
 12 \quad 0001100
 \end{array}$$

$$\text{h) } \begin{array}{r} -11 - \\ 29 = \\ \hline -40 \end{array} \quad \begin{array}{r} 1\ 1\ 1\ 0\ 1\ 0\ 1 - \\ 0\ 0\ 1\ 1\ 1\ 0\ 1 = \\ \hline 1\ 0\ 1\ 1\ 0\ 0\ 0 \end{array}$$

$$\text{i) } \begin{array}{r} 8 + \\ -18 = \\ \hline -10 \end{array} \quad \begin{array}{r} 0\ 0\ 0\ 1\ 0\ 0\ 0 + \\ 1\ 1\ 0\ 1\ 1\ 1\ 0 = \\ \hline 1\ 1\ 1\ 0\ 1\ 1\ 0 \end{array}$$

$$\text{j) } \begin{array}{r} 7 - \\ 17 = \\ \hline -10 \end{array} \quad \begin{array}{r} 0\ 0\ 0\ 0\ 1\ 1\ 1 - \\ 0\ 0\ 1\ 0\ 0\ 0\ 1 = \\ \hline 1\ 1\ 1\ 0\ 1\ 1\ 0 \end{array}$$

2. a) Servono 4 bit:

$$\begin{array}{r} 3 + \\ -7 = \\ \hline -4 \end{array} \quad \begin{array}{r} 0\ 0\ 1\ 1 + \\ 1\ 0\ 0\ 1 = \\ \hline 1\ 1\ 0\ 0 \end{array}$$

b) Servono 4 bit:

$$\begin{array}{r} -3 + \\ 7 = \\ \hline 4 \end{array} \quad \begin{array}{r} 1\ 1\ 0\ 1 + \\ 0\ 1\ 1\ 1 = \\ \hline 0\ 1\ 0\ 0 \end{array}$$

c) Servono 5 bit:

$$\begin{array}{r} 11 + \\ -11 = \\ \hline 0 \end{array} \quad \begin{array}{r} 0\ 1\ 0\ 1\ 1 + \\ 1\ 0\ 1\ 0\ 1 = \\ \hline 0\ 0\ 0\ 0\ 0 \end{array}$$

d) Servono 6 bit:

$$\begin{array}{r} 18 - \\ -3 = \\ \hline 21 \end{array} \quad \begin{array}{r} 0\ 1\ 0\ 0\ 1\ 0 - \\ 1\ 1\ 1\ 1\ 0\ 1 = \\ \hline 0\ 1\ 0\ 1\ 0\ 1 \end{array}$$

3.15.3 Numerazione Ottale e Esadecimale

1. Il modo più rapido è di scrivere prima il numero in binario:

a) 276534_8 in binario è 010111110101011110_2 . La rappresentazione esadecimale è: $17D5C_H$.

b) 22017555724_8 è $0100100000011111011011011111010100_2$ in binario. Esadecimale: $903EDBD4_H$.

2. a) Conversione di 7722_{10} :

7722	0		
3861	1		
1930	0		
965	1		
482	0		
241	1	$7722 \mid 10$	(A) (es: $7722 : 16 = 482 + 10$)
120	0	$482 \mid 2$	
60	0	$30 \mid 14$	(E)
30	0	$1 \mid 1$	
15	1		
7	1		
3	1		
1	1		

In binario: 1111000101010_2 , in esadecimale: $1E2A_H$.

b) Conversione di 1435_{10} :

1435	1		
717	1		
358	0		
179	1		
89	1	$1435 \mid 11$	(B) (es: $1435 : 16 = 89 + 11$)
44	0	$89 \mid 9$	
22	0	$5 \mid 5$	
11	1		
5	1		
2	0		
1	1		

In binario: 10110011011_2 , in esadecimale: $59B_H$.

3. a) Conversione di 36625_{10} :

36625	1	$(36625 : 8 = 4578 + 1)$		
4578	2	$(4578 : 8 = 572 + 2)$	$36625 \mid 1$	$(36625 : 16 = 2289 + 1)$
572	4	$(572 : 8 = 71 + 4)$	$2289 \mid 1$	$(2289 : 16 = 143 + 1)$
71	7	$(71 : 8 = 8 + 7)$	$143 \mid F$	$(143 : 16 = 8 + 15)$
8	0	$(8 : 8 = 1 + 0)$	$8 \mid 8$	
1	1			

In ottale: 107421_8 , in esadecimale: $8F11_H$.

b) Conversione di 124_{10} :

124	4	$(124 : 8 = 15 + 4)$	$124 \mid 12$	(C) $(124 : 16 = 7 + 12)$
15	7	$(15 : 8 = 1 + 7)$	$7 \mid 7$	
1	1			

In ottale: 174_8 , in esadecimale: $7C_H$.

4. Convertiamo prima il numero in binario, poi in decimale:

a) $110100010101100000111_2 = 1714951_{10}$

b) $10001000001000111_2 = 69703_{10}$

5. Da ottale a binario, poi in decimale:

a) $11001001001_2 = 1609_{10}$

b) $10111110101011100_2 = 97628_{10}$

6. a)
$$\begin{array}{r} 4\ 1\ A\ B\ 7\ + \\ C\ 2\ D\ 6\ F\ = \\ \hline 1\ 0\ 4\ 8\ 2\ 6 \end{array}$$

b)
$$\begin{array}{r} A\ 2\ 3\ C\ E\ + \\ 3\ 6\ 3\ E\ 6\ = \\ \hline D\ 8\ 7\ B\ 4 \end{array}$$

Complementi sul progetto di reti combinatorie

Nei capitoli precedenti abbiamo affrontato la minimizzazione di espressioni booleane, ovvero la ricerca di un'espressione equivalente a quella originale ma con il minor numero di implicanti primi, utilizzando le mappe di Karnaugh. Purtroppo il metodo delle mappe può essere applicato ad espressioni con un numero massimo di quattro variabili, anche se è possibile estenderlo a cinque variabili utilizzando mappe a tre dimensioni o, in casi limitati, a più variabili attraverso l'uso di variabili riportate. Per un numero di variabili superiore a quattro è necessario abbandonare i metodi "manuali" e ricorrere a metodi algoritmici, realizzabili su calcolatore.

4.1 Minimizzazione di espressioni booleane con il metodo di Quine-McCluskey

Uno dei primi algoritmi per la minimizzazione di espressioni booleane è stato proposto da Willard V. Quine (1908 - 2000), poi migliorato da Edward J. McCluskey (1929 - 2016)¹ ed è noto come "Metodo di Quine-McCluskey" (nel seguito lo indicheremo con l'acronimo "MdQ-M").

Il MdQ-M, in effetti, è la traduzione, sotto forma di algoritmo, della procedura che viene eseguita manualmente con le mappe di Karnaugh e si compone di due fasi. Nella prima fase, detta di "espansione" vengono generati tutti gli implicanti della funzione che si vuole minimizzare (i "cubi" per Karnaugh) e si identificano tra questi gli implicanti primi, eliminando gli altri. Nella seconda fase, detta di "copertura", viene scelto il numero minimo di implicanti primi necessario ad ottenere una funzione equivalente a quella di partenza, ovvero a "coprire" tutti i *minterm* della funzione. Le due fasi del metodo utilizzano alcune tabelle di appoggio, che tengono traccia dei passi svolti dall'algoritmo, e sono facilmente implementabili al calcolatore.

¹ E.J.McCluskey, Minimization of Boolean Functions, The Bell System Technical Journal, November 1956

I *minterm* m_0 e m_2 , ad esempio, possono essere combinati in questo modo:

$$\overline{X} \overline{Y} \overline{Z} + \overline{X} Y \overline{Z} = \overline{X} \overline{Z} (\overline{Y} + Y) = \overline{X} \overline{Z}$$

ovvero, utilizzando la notazione del MdQ-M:

$$\begin{aligned} 000 + 010 &\rightarrow 0-0 \\ 0 + 2 &\rightarrow (0, 2) \end{aligned}$$

Il segno “-” è utilizzato per indicare che la corrispondente variabile è stata eliminata nella semplificazione.

Combinando tutte le possibili coppie di *minterm* si ottiene una nuova tabella in cui sono riportati tutti gli implicanti composti da due variabili, ovvero una variabile in meno rispetto a quelli di partenza:

3 Variabili			2 Variabili			1 Variabile		
Termini	$X Y Z$	P	Termini	$X Y Z$	P	Termini	$X Y Z$	P
0	000	√	0, 2	0-0				
2	010	√	0, 4	-00				
4	100	√	2, 3	01-				
3	011	√	2, 6	-10				
6	011	√	4, 6	1-0				
7	111	√	3, 7	-11				
			6, 7	11-				

È importante a questo punto analizzare due accorgimenti che sono stati utilizzati in questo primo passo dell’algoritmo.

Il primo riguarda l’ordinamento iniziale dei *minterm* che sono raggruppati per numero di variabili negate, ovvero per numero di 0 nel corrispondente numero binario². Al primo gruppo appartiene il *minterm* con tutte le variabili negate (000), al secondo gruppo quelli con due variabili negate (010, 100) e così via. Infatti, non è possibile combinare tra di loro *minterm* che differiscono per due o più variabili, come ad esempio $\overline{X} \overline{Y} \overline{Z}$ (000) e $\overline{X} Y Z$ (011), perché non si otterrebbe alcuna semplificazione: è possibile combinare solamente coppie di *minterm* che differiscono per una variabile, che compare in forma negata in uno dei due *minterm* e in forma diretta nell’altro, come ad esempio $X Y \overline{Z}$ (110) e $X Y Z$ (111). In altre parole, si possono combinare due *minterm* solo se i corrispondenti numeri binari hanno distanza di Hamming uguale a 1 tra di loro. Raggruppando i numeri che contengono la stessa quantità di 0 si creano gruppi omogenei ed è possibile ridurre il numero di confronti: invece di confrontare tutte le coppie possibili di *minterm* è possibile controllare solo quelle che appartengono a gruppi adiacenti, che sono, per costruzione, a distanza

² Nella tabella i gruppi sono separati da una linea continua

di Hamming più ravvicinata. Infatti, il *minterm* del primo gruppo (000) si può combinare solo con quelli del secondo (010, 100), i *minterm* del secondo gruppo (010, 100) si possono combinare solo con quelli del terzo (011, 110) e così via³. Si noti che non tutte le combinazioni sono possibili: il termine (100) che appartiene al secondo gruppo può essere combinato con (110) ma non con (011) del terzo gruppo perché la loro distanza di Hamming in questo caso è uguale a due.

Il secondo accorgimento ha l'obiettivo di tenere traccia dei termini (implicanti) che sono stati combinati. Se, infatti, due implicanti vengono combinati tra di loro per ottenere un implicante con una variabile in meno, significa che gli implicanti di partenza non erano implicanti primi e che quindi non devono essere considerati nella successiva fase di copertura. Sappiamo infatti che un implicante non primo può essere sempre sostituito da un implicante primo, se è coperto da quest'ultimo, con un minor numero di variabili. La colonna P della tabella ha proprio questo scopo: una spunta indica che il termine è stato combinato quindi non è un implicante primo e può essere trascurato nella successiva fase di copertura.

Applicando la stessa procedura agli implicanti a due variabili si ottengono implicanti in cui un'ulteriore variabile è stata eliminata.

3 Variabili			2 Variabili			1 Variabile		
Termini	X Y Z	P	Termini	X Y Z	P	Termini	X Y Z	P
0	000	✓	0, 2	0-0	✓	0, 2, 4, 6	--0	P ₀
2	010	✓	0, 4	-00	✓	2, 3, 6, 7	-1-	P ₁
4	100	✓	2, 3	01-	✓			
3	011	✓	2, 6	-10	✓			
6	011	✓	4, 6	1-0	✓			
7	111	✓	3, 7	-11	✓			
			6, 7	11-	✓			

Si noti che nel combinare gli implicanti a due variabili si può ottenere lo stesso implicante ad una variabile in diversi modi. Ad esempio:

$$(0, 2) + (4, 6) \rightarrow (0, 2, 4, 6)$$

$$0-0 + 1-0 \rightarrow --0$$

oppure

$$(0, 4) + (2, 6) \rightarrow (0, 4, 2, 6)$$

$$-00 + -10 \rightarrow --0$$

³ Se dovessimo eseguire tutti i confronti a coppie possibili tra n termini, per verificare se possono essere combinati insieme, dovremmo eseguire al più $n(n-1)/2$ confronti, ovvero ogni termine con tutti gli altri e tenendo in considerazione il fatto che il confronto è bidirezionale.

Ovviamente l'implicante (0, 2, 4, 6) e l'implicante (0, 4, 2, 6) identificano lo stesso termine perché corrispondono a $(- - 0)$ ovvero al termine \bar{Z} . Per questo motivo solo uno di questi è riportato nella terza tabella.

Quando non è possibile combinare ulteriormente gli implicanti, come in questo caso in cui sono rimasti solo due termini non combinabili tra di loro, la fase di espansione è conclusa e tutti i termini che non presentano un segno di spunta sono implicanti primi.

Possiamo quindi numerare gli implicanti primi della funzione di partenza che sono:

$$\begin{array}{llll} P_0 & \text{ovvero} & (0, 2, 4, 6) & \text{ovvero} & (- - 0) & \text{ovvero} & \bar{Z} \\ P_1 & \text{ovvero} & (2, 3, 6, 7) & \text{ovvero} & (- 1 -) & \text{ovvero} & Y \end{array}$$

4.1.2 Fase di copertura

Se nella fase di espansione lo scopo è trovare tutti gli implicanti primi, durante la fase di copertura l'obiettivo è, invece, l'individuazione del minor numero di implicanti primi sufficiente a coprire la funzione di partenza. Dobbiamo quindi essere sicuri che tutti i *minterm* con cui era definita la funzione da minimizzare siano coperti da almeno un implicante primo tra quelli individuati. Allo stesso tempo dobbiamo garantire di utilizzare, per ottenere questa copertura, il minor numero di implicanti.

Per raggiungere questo obiettivo il MdQ-M utilizza la cosiddetta “tabella di copertura” in cui, nelle colonne, sono indicati tutti i *minterm* e, nelle righe, tutti gli implicanti primi individuati nella fase di espansione. Il segno “X” in corrispondenza della colonna relativa al *minterm* m_i e della riga relativa all'implicante primo P_j indica che m_i è coperto da P_j :

	m_0	m_2	m_3	m_4	m_6	m_7
P_0	X	X		X	X	
P_1			X		X	X

Nel nostro caso è facile dedurre che entrambi gli implicanti primi sono necessari per una completa copertura della funzione, quindi la funzione minimizzata risulta:

$$F(X, Y, Z) = P_0 + P_1 = \bar{Z} + Y$$

Si noti che se una colonna contiene un solo segno “X”, ciò significa che esiste un solo implicante primo in grado di coprire il *minterm* corrispondente. In questo caso l'implicante è un implicante primo essenziale perché, senza di questo, la funzione non potrebbe essere completamente coperta. Nel caso appena visto entrambi gli implicanti primi sono essenziali perché i *minterm* m_0 e m_4 sono

coperti solo da P_0 mentre i *minterm* m_3 e m_7 sono coperti solo da P_1 . Vedremo in seguito che le tabelle di copertura possono essere molto più complesse di questa e quindi, dopo aver selezionato gli implicanti primi essenziali, sarà necessario ricorrere ad un algoritmo per scegliere gli implicanti rimanenti in modo da ottenere una copertura minima.

4.1.3 Funzioni non completamente specificate

Spesso abbiamo a che fare con funzioni non completamente specificate, come ad esempio quella rappresentata dalla seguente mappa:

			Z	
	1	1	-	0
X	1	1	-	0
			Y	

che nella notazione del MdQ-M si scrive:

$$F(X, Y, Z) = \Sigma(0, 2, 4, 6) + d(3, 7)$$

in cui $d()$ raccoglie tutti i *minterm* corrispondenti alle indifferenze.

In questo caso il MdQ-M può essere applicato semplicemente considerando le indifferenze come 1 nella fase di espansione e come 0 nella fase di copertura. l'idea di fondo è la seguente: nella fase di espansione, ovvero quando si costruiscono cubi sempre più grandi o, in altre parole, implicanti con sempre meno variabili, è conveniente cercare di utilizzare il maggior numero possibile di *minterm* per aumentare le possibilità di semplificazione. Nella fase di copertura, invece, si vuole evitare di coprire un *minterm* se ciò non è strettamente necessario e quindi eliminandolo dalla copertura si cerca di evitare che questo *minterm* costringa a considerare qualche implicante primo superfluo.

Per la funzione indicata sopra la fase di espansione è quindi la stessa del caso precedente, mentre la fase di copertura utilizza una tabella senza i *minterm* m_3 e m_7 , che corrispondono alle indifferenze:

	m_0	m_2	m_4	m_6
P_0	X	X	X	X
\bar{P}_1	-	X	-	X

Da questa tabella si ricava immediatamente che P_0 è un implicante primo essenziale (per m_0 e m_4) ed è anche sufficiente a coprire tutta la funzione, ottenendo, come previsto:

$$F(X, Y, Z) = P_0 = \bar{Z}$$

4.1.4 Ottimizzazione della fase di copertura

La fase di copertura può essere particolarmente complessa in presenza di un numero elevato di implicanti primi.

Supponiamo infatti che, dopo la fase di espansione e dopo l'individuazione degli implicanti primi essenziali, siano rimasti k implicanti primi $\{P_0, \dots, P_{k-1}\}$ tra i quali scegliere la copertura minima.

Allo scopo di ottenere questa, dovremmo controllare $2^k - 1$ diversi casi, ovvero tutte le combinazioni possibili con 1, 2, 3, 4 implicanti primi, e così via, fino alla (sfortunata) possibilità di doverli utilizzare tutti: $\{P_0\}$, $\{P_1\}$, \dots , $\{P_k\}$, $\{P_0, P_1\}$, $\{P_0, P_2\}$, $\{P_0, P_3\}$, e così via fino a $\{P_0, \dots, P_{k-1}\}$.

Il MdQ-M propone, invece, una procedura più efficace che, nella maggior parte dei casi, richiede un numero di confronti decisamente minore. Supponiamo di voler minimizzare la seguente funzione:

		W		
	0	0	1	1
	0	1	-	1
Z	1	-	0	1
1	1	-	0	1
		Y		

$$F(X, Y, W, Z) = \Sigma(1, 2, 3, 6, 9, 10, 11, 12) + d(5, 13, 14)$$

La fase di espansione inizia ordinando i termini a gruppi, come illustrato nelle sezioni precedenti. Il primo gruppo contiene *minterm* con tre variabili negate, il secondo con due e il terzo con una, come visibile nelle tabelle seguenti.

4 Variabili			3 Variabili			2 Variabili		
Termini	$XYZW$	P	Termini	$XYZW$	P	Termini	$XYZW$	P
1	0001							
2	0010							
3	0011							
5	0101							
6	0110							
9	1001							
10	1010							
12	1100							
11	1011							
13	1101							
14	1110							

A questo punto può iniziare il confronto tra tutti i termini del primo gruppo e tutti i termini del secondo gruppo: quando è possibile combinare due termini di quattro variabili, questi danno origine ad implicanti con tre variabili:

4 Variabili			3 Variabili			2 Variabili		
Termini	$XYZW$	P	Termini	$XYZW$	P	Termini	$XYZW$	P
1	0001	√	1, 3	00-1				
2	0010	√	1, 5	0-01				
3	0011	√	1, 9	-001				
5	0101	√	2, 3	001-				
6	0110	√	2, 6	0-10				
9	1001	√	2, 10	-010				
10	1010	√						
12	1100							
11	1011							
13	1101							
14	1110							

Quindi si procede confrontando i termini del secondo e terzo gruppo (vedi le tabelle che seguono).

4 Variabili			3 Variabili			2 Variabili		
Termini	$XYZW$	P	Termini	$XYZW$	P	Termini	$XYZW$	P
1	0001	✓	1, 3	00-1				
2	0010	✓	1, 5	0-01				
3	0011	✓	1, 9	-001				
5	0101	✓	2, 3	001-				
6	0110	✓	2, 6	0-10				
9	1001	✓	2, 10	-010				
10	1010	✓	3, 11	-011				
12	1100	✓	5, 13	-101				
11	1011	✓	6, 14	-110				
13	1101	✓	9, 11	10-1				
14	1110	✓	9, 13	1-01				
			10, 11	101-				
			10, 14	1-10				
			12, 13	110-				
			12, 14	11-0				

Una spunta sulla colonna P in corrispondenza di tutti gli implicanti a quattro variabili (ovvero i *minterm*) indica che nessuno di loro è *primo*.

L'algoritmo quindi prosegue confrontando i due gruppi di implicanti a tre variabili, per ottenere implicanti a due variabili:

4 Variabili			3 Variabili			2 Variabili		
Termini	$XYZW$	P	Termini	$XYZW$	P	Termini	$XYZW$	P
1	0001	✓	1, 3	00-1	✓	1, 3, 9, 11	-0-1	P_2
2	0010	✓	1, 5	0-01	✓	1, 5, 9, 13	--01	P_3
3	0011	✓	1, 9	-001	✓	2, 3, 10, 11	-01-	P_4
5	0101	✓	2, 3	001-	✓	2, 6, 10, 14	--10	P_5
6	0110	✓	2, 6	0-10	✓			
9	1001	✓	2, 10	-010	✓			
10	1010	✓	3, 11	-011	✓			
12	1100	✓	5, 13	-101	✓			
11	1011	✓	6, 14	-110	✓			
13	1101	✓	9, 11	10-1	✓			
14	1110	✓	9, 13	1-01	✓			
			10, 11	101-	✓			
			10, 14	1-10	✓			
			12, 13	110-	P_0			
			12, 14	11-0	P_1			

Si noti che è stato possibile combinare, ad esempio, sia $(1, 5) + (9, 13) \rightarrow (1, 5, 9, 13)$ sia $(1, 9) + (5, 13) \rightarrow (1, 9, 5, 13)$ ma questi producono lo stesso termine (-01) ovvero $\overline{W}Z$ che ovviamente è riportato una sola volta nella tabella a due variabili. Entrambe le coppie $(1, 5)$, $(9, 13)$ e $(1, 9)$, $(5, 13)$, ricevono il segno di spunta perché nessuna di loro è un implicante primo.

Il numero di implicanti primi risulta quindi essere pari a sei: P_0 e P_1 a tre variabili (corrispondenti sulla mappa di Karnaugh a due cubi da due caselle) e P_2, P_3, P_4, P_5 a due variabili (corrispondenti sulla mappa a quattro cubi da quattro caselle).

Il MdQ-M procede ora con la tabella di copertura per trovare il numero minimo di implicanti che copre la funzione:

	m_1	m_2	m_3	m_6	m_9	m_{10}	m_{11}	m_{12}
P_0	--	--	--	--	--	--	--	X
\overline{P}_1	--	--	--	--	--	--	--	\overline{X}
P_2	X	--	X	--	X	--	X	--
\overline{P}_3	\overline{X}	--	--	--	\overline{X}	--	--	--
\overline{P}_4	--	\overline{X}	\overline{X}	--	--	\overline{X}	\overline{X}	--
\overline{P}_5	--	\overline{X}	--	\overline{X}	--	\overline{X}	--	--

La tabella indica chiaramente che P_5 è un implicante primo essenziale, perché è l'unico che copre m_6 , quindi l'espressione minima conterrà sicuramente il termine:

$$F(X, Y, W, Z) = P_5 + \dots = W\overline{Z} + \dots$$

Possiamo quindi scrivere una nuova tabella eliminando la riga corrispondente al termine P_5 , che è già stato selezionato, e le colonne corrispondenti a m_2 , m_6 , m_{10} , che sono coperte da P_5 :

	m_1	m_3	m_9	m_{11}	m_{12}
P_0	--	--	--	--	X
\overline{P}_1	--	--	--	--	\overline{X}
P_2	X	X	X	X	--
\overline{P}_3	\overline{X}	--	\overline{X}	--	--
\overline{P}_4	--	\overline{X}	--	X	--

La tabella che ne risulta può essere ulteriormente semplificata analizzando le coperture per righe e per colonne ed eliminando, corrispondentemente, alcuni implicanti primi (righe) o alcuni *minterm* (colonne).

In particolare: una colonna m_i può essere eliminata se copre un'altra colonna m_j , ovvero se per ogni X nella colonna m_j c'è un X nella corrispondente riga

della colonna m_i . Questa eliminazione è possibile perché nella configurazione appena descritta il *minterm* m_i sarebbe comunque coperto da uno degli implicanti che coprono m_j , quindi è inutile considerarlo.

Nel caso della tabella indicata sopra, le colonne corrispondenti a m_9 e m_{11} possono quindi essere eliminate perché coprono, rispettivamente, m_1 e m_3 (anzi, in questo caso sono addirittura uguali):

	m_1	m_3	m_{12}
P_0	-	-	X
\bar{P}_1	-	-	\bar{X}
P_2	X	X	-
\bar{P}_3	\bar{X}	-	-
\bar{P}_4	-	\bar{X}	-

Analogamente, una riga corrispondente a P_i può essere eliminata se è coperta da un altro implicante P_j , ovvero se per ogni X nella colonna P_i c'è un X nella corrispondente colonna della riga P_j . Questa eliminazione è possibile perché l'implicante primo P_j copre tutti i *minterm* coperti da P_j (ed eventualmente anche altri in più).

Nella tabella indicata sopra è immediato verificare che P_1 copre P_0 e viceversa, mentre P_2 copre P_3 e P_4 . Quindi la tabella finale risulta essere:

	m_1	m_3	m_{12}
P_0			X
P_2	X	X	

A questo punto è estremamente semplice identificare la copertura ottima che risulta essere composta da P_0 e P_2 . Aggiungendo questi due implicanti primi a quello già identificato in precedenza (P_5) si ottiene il risultato finale:

$$F(X, Y, W, Z) = P_0 + P_2 + P_5 = X Y \bar{W} + \bar{Y} Z + W \bar{Z}$$

In rari casi può succedere che in una tabella di copertura non sia possibile eliminare né colonne né righe: in questo caso la tabella di copertura si dice "ciclica" ed è necessario verificare tutte le possibili combinazioni di implicanti primi per ottenere la copertura ottima. Esistono diversi metodi, sia ottimi sia sub-ottimi, per affrontare anche questo problema ma non li tratteremo in questo libro.

4.1.5 Ottimizzazione contemporanea di più funzioni

Nei sistemi digitali reali è molto comune dover realizzare, nello stesso progetto, diverse reti combinatorie che corrispondono ad altrettante funzioni booleane. In questo caso si può trarre vantaggio, in termini di complessità del circuito, da un'ottimizzazione congiunta delle funzioni in modo da riutilizzare più volte alcune parti dei circuiti che sono comuni a più reti. Per questo motivo il MdQ-M è stato esteso in modo da ottimizzare contemporaneamente più funzioni booleane, modificando sia la fase di espansione sia la fase di copertura, allo scopo di identificare quali implicanti primi possono essere utilizzati per coprire i *minterm* di più funzioni.

Per introdurre l'estensione del metodo a più funzioni utilizziamo un esempio, supponendo di dover ottimizzare le seguenti tre funzioni:

		W		
	0	0	1	1
	0	1	-	1
Z	0	0	0	1
	0	0	0	1
		Y		

$$F_1(X, Y, W, Z) = \Sigma(2, 3, 6, 10, 11, 12) + d(14)$$

		W		
	0	0	0	0
	0	0	0	0
Z	1	-	0	1
	1	-	0	1
		Y		

$$F_2(X, Y, W, Z) = \Sigma(1, 3, 9, 11) + d(5, 13)$$

		W		
	0	0	0	1
	0	0	0	1
Z	1	1	0	1
	1	-	0	1
		Y		

$$F_3(X, Y, W, Z) = \Sigma(1, 2, 3, 9, 10, 11) + d(5)$$

Nella fase di espansione tutti i *minterm* delle tre funzioni vengono riportati nella tabella iniziale. La differenza rispetto al caso con una sola funzione è la presenza di un'ulteriore colonna che indica, attraverso una "maschera" di tanti bit quante sono le funzioni da ottimizzare, quali di queste funzioni contengono il corrispondente *minterm*.

4 Variabili			
Term.	XYZW	F ₁ F ₂ F ₃	P
1	0001	011	
2	0010	101	
3	0011	111	
5	0101	011	
6	0110	100	
9	1001	011	
10	1010	101	
12	1100	100	
11	1011	111	
13	1101	010	
14	1110	100	

3 Variabili			
Term.	XYZW	F ₁ F ₂ F ₃	P

2 Variabili			
Term.	XYZW	F ₁ F ₂ F ₃	P

Il *minterm* m_1 , ad esempio, compare nelle funzioni F_2 e F_3 ma non nella funzione F_1 e ciò viene indicato con la maschera "011". Analogamente, il *minterm* m_{14} compare nella funzione F_1 ma non in F_2 e F_3 e quindi la corrispondente maschera è "100".

A questo punto si procede con la fase di espansione e i termini a quattro variabili sono combinati tra di loro a formare termini a tre variabili. Si deve però tenere conto del fatto che due termini possono essere combinati solo se entrambi compaiono nella stessa funzione. Si consideri, ad esempio, la combinazione di m_1 con m_3 , ovvero (0001) con (0011) per ottenere (00-1): questa è possibile solo per le funzioni F_2 e F_3 che contengono entrambi i termini. La funzione F_1 contiene il termine m_3 ma non il termine m_1 quindi, solo per questa funzione, la semplificazione considerata è impossibile da realizzare. In corrispondenza di (1, 3) quindi si utilizzerà la maschera "011" per indicare che questo termine è presente solo in due delle tre funzioni: F_2 e F_3 .

Anche la spunta che indica gli implicanti primi deve essere applicata tenendo conto di quali termini sono stati effettivamente combinati e relativamente a quali funzioni. Infatti, il *minterm* m_1 è stato semplificato in tutti i casi, quindi non è un implicante primo e il segno di spunta nella colonna P può essere inserito. Il *minterm* m_3 , invece, non è stato semplificato con alcun altro termine nella funzione F_1 quindi il segno di spunta non può essere inserito perché per questa funzione, almeno fino a questo passo dell'espansione, m_3 risulta essere ancora un potenziale implicante primo.

nella funzione F_1 non è stato possibile semplificarlo con un altro termine: infatti le due maschere sono, rispettivamente, “011” e “111”.

Completata la fase di espansione si può procedere con la fase di copertura considerando ogni funzione separatamente:

	F_1						F_2				F_3					
	m_2	m_3	m_6	m_{10}	m_{11}	m_{12}	m_1	m_3	m_9	m_{11}	m_1	m_2	m_3	m_9	m_{10}	m_{11}
P_0							X				X					
P_1		X			X			X		X			X			X
P_2						X										
P_3							X	X	X	X	X		X	X		X
P_4							X		X							
P_5	X	X		X	X							X	X		X	X
P_6	X		X	X											X	X

Anche in questo caso le maschere sono utili perché indicano a quali funzioni il corrispondente implicante primo deve essere associato. L'implicante primo P_4 , ad esempio, copre i *minterm* m_1 , m_5 , m_9 e m_{13} solo per la funzione F_2 , perché la maschera corrispondente è “010”, quindi non vengono inseriti simboli di copertura X per le altre funzioni (F_1 e F_3).

Per identificare la copertura minima si procede, come al solito, identificando gli implicanti primi essenziali che, in questo caso, risultano essere P_2 (perché è l'unico a coprire il *minterm* m_{12} in F_1), P_3 (perché è l'unico a coprire il *minterm* m_9 di F_3), P_5 (perché è l'unico a coprire i *minterm* m_2 e m_{10} in F_3) e P_6 (perché è l'unico a coprire il *minterm* m_6 in F_1). Si noti che selezionando un implicante primo, questo viene selezionato per tutte le funzioni: l'implicante P_5 , ad esempio, viene selezionato perché è essenziale per F_3 ma a quel punto è utilizzato per coprire anche i *minterm* di F_1 . In altre parole P_5 , che corrisponde al termine $(2, 3, 10, 11)$ ovvero $\bar{Y}W$, compare sicuramente nell'espressione ottima di F_3 ma può essere utilizzato anche per F_1 . In questo modo la stessa rete logica è sfruttata due volte e si ottiene un risparmio a livello circuitale.

Selezionando P_2 , P_3 , P_5 e P_6 tutti i *minterm* di tutte le funzioni risultano coperti e quindi si può procedere a scrivere le funzioni ottenute:

$$F_1(X, Y, W, Z) = P_2 + P_5 + P_6 = XY\bar{Z} + \bar{Y}W + W\bar{Z}$$

$$F_2(X, Y, W, Z) = P_3 = \bar{Y}\bar{Z}$$

$$F_3(X, Y, W, Z) = P_3 + P_5 = \bar{Y}\bar{Z} + \bar{Y}$$

Si noti che le reti combinatorie corrispondenti a P_3 e P_5 sono utilizzate due volte in funzioni diverse ottenendo un risparmio nella complessità globale del circuito.

4.3 Soluzioni

4.3.1 Quine-McCluskey: sintesi di singola funzione

1. $F(A, B, C, D) = B D + A \bar{C}$

4 Variabili		
Termini	ABCD	P
4	0100	√
8	1000	√
5	0101	√
9	1001	√
12	1100	√
7	0111	√
13	1101	√
15	1111	√

3 Variabili		
Termini	ABCD	P
5, 4	010-	√
9, 8	100-	√
12, 4	-100	√
12, 8	1-00	√
7, 5	01-1	√
13, 5	-101	√
13, 9	1-01	√
13, 12	110-	√
15, 7	-111	√
15, 13	11-1	√

2 Variabili		
Termini	ABCD	P
13, 12, 9, 8	1-0-	P_0
3, 12, 5, 4	-10-	P_1
15, 13, 7, 5	-1-1	P_2

	m_5	m_7	m_8	m_9	m_{12}	m_{13}	m_{15}
P_0			X	X	X	X	
P_1	X				X	X	
P_2	X	X				X	X

2. $F(A, B, C, D) = A B C + \bar{A} \bar{C} + \bar{A} \bar{B}$

4 Variabili		
Termini	ABCD	P
0	0000	√
1	0001	√
2	0010	√
4	0100	√
3	0011	√
5	0101	√
10	1010	√
14	1110	√
15	1111	√

3 Variabili		
Termini	ABCD	P
1, 0	000-	√
2, 0	00-0	√
4, 0	0-00	√
3, 1	00-1	√
3, 2	001-	√
5, 1	0-01	√
5, 4	010-	√
10, 2	-010	P_0
14, 10	1-10	P_1
15, 14	111-	P_2

2 Variabili		
Termini	ABCD	P
3, 2, 1, 0	00--	P_3
5, 4, 1, 0	0-0-	P_4

	m_0	m_1	m_2	m_3	m_4	m_5	m_{15}
P_0			X				
P_1							
P_2							X
P_3	X	X	X	X			
P_4	X	X			X	X	

3. $F(A, B, C, D) = A D + \overline{C} D$

4 Variabili		
Termini	$ABCD$	P
0	0000	∨
1	0001	∨
4	0100	∨
9	1001	∨
5	0101	∨
11	1011	∨
13	1101	∨
15	1111	∨
---	---	---

3 Variabili		
Termini	$ABCD$	P
1, 0	000-	∨
4, 0	0-00	∨
5, 1	0-01	∨
5, 4	010-	∨
9, 1	-001	∨
11, 9	10-1	∨
13, 5	-101	∨
13, 9	1-01	∨
15, 11	1-11	∨
15, 13	11-1	∨

2 Variabili		
Termini	$ABCD$	P
5, 4, 1, 0	0-0-	P_0
13, 5, 9, 1	--01	P_1
15, 13, 11, 9	1--1	P_2
---	---	---
---	---	---
---	---	---
---	---	---
---	---	---

	m_1	m_9	m_{11}	m_{13}	m_{15}
P_0	X				
P_1	X	X		X	
P_2		X	X	X	X

4.3.2 Quine-McCluskey: sintesi congiunta di più funzioni

1. Di seguito la soluzione

$$F_1(A, B, C, D) = B \bar{C}$$

$$F_2(A, B, C, D) = A \bar{C}$$

$$F_3(A, B, C, D) = B D$$

4 Variabili			
Term.	ABCD	F ₁ F ₂ F ₃	P
4	0100	111	√
8	1000	010	√
5	0101	111	√
9	1001	010	√
12	1100	111	√
7	0111	100	√
13	1101	111	√
15	1111	100	√
---	---	---	---

3 Variabili			
Term.	ABCD	F ₁ F ₂ F ₃	P
5, 4	010-	111	√
4, 12	-100	111	√
8, 9	100-	010	√
8, 12	1-00	010	√
5, 7	01-1	100	√
5, 13	-101	111	√
9, 13	1-01	010	√
12, 13	110-	111	√
7, 15	-111	100	√
13, 15	11-1	100	√

2 Variabili			
Term.	ABCD	F ₁ F ₂ F ₃	P
4, 5, 12, 13	-10-	111	P ₀
8, 9, 12, 13	1-0-	010	P ₁
5, 7, 13, 15	-1-1	100	P ₂
---	---	---	---

	F ₁				F ₂				F ₃			
	m ₅	m ₇	m ₁₃	m ₁₅	m ₈	m ₉	m ₁₂	m ₁₃	m ₄	m ₅	m ₁₂	m ₁₃
P ₀	X	-	X	-	-	X	X	X	X	X	X	X
P ₁	-	-	-	-	X	X	X	X	-	-	-	-
P ₂	X	X	X	X	-	-	-	-	-	-	-	-

2. Di seguito la soluzione

$$F_1(A, B, C, D) = A B C D + \bar{A} \bar{B}$$

$$F_2(A, B, C, D) = \bar{A} \bar{B} + \bar{A} \bar{C}$$

$$F_3(A, B, C, D) = \bar{A} \bar{B} \bar{C} + \bar{A} \bar{C}$$

4 Variabili			
Term.	ABCD	F ₁ F ₂ F ₃	P
0	0000	111	√
1	0001	111	√
2	0010	110	√
4	0100	011	√
3	0011	110	√
5	0101	011	√
15	1111	100	P ₀

3 Variabili			
Term.	ABCD	F ₁ F ₂ F ₃	P
0, 1	000-	111	P ₁
0, 2	00-0	110	√
0, 4	0-00	011	√
1, 3	00-1	110	√
1, 5	0-01	011	√
2, 3	001-	110	√
4, 5	010-	011	√

2 Variabili			
Term.	ABCD	F ₁ F ₂ F ₃	P
0, 1, 2, 3	00--	110	P ₂
0, 1, 4, 5	0-0-	011	P ₃
---	---	---	---

	F ₁				F ₂				F ₃			
	m ₀	m ₁	m ₂	m ₁₅	m ₂	m ₃	m ₄	m ₅	m ₀	m ₁	m ₄	m ₅
P ₀	-	-	-	X	-	-	-	-	-	-	-	-
P ₁	X	X	-	-	-	-	-	-	X	X	-	-
P ₂	X	X	X	-	X	X	-	-	-	-	-	-
P ₃	-	-	-	-	-	-	X	X	-	-	X	X

3. Di seguito la soluzione

$$F_1(A, B, C, D) = \overline{A} \overline{C} D + \overline{A} B \overline{C}$$

$$F_2(A, B, C, D) = \overline{C} D$$

$$F_3(A, B, C, D) = A D$$

4 Variabili			
Term.	ABCD	F ₁ F ₂ F ₃	P
0	0000	010	√
1	0001	111	√
4	0100	100	√
5	0101	111	√
9	1001	011	√
11	1011	001	√
13	1101	111	√
15	1111	001	√

3 Variabili			
Term.	ABCD	F ₁ F ₂ F ₃	P
0, 1	000-	010	P ₀
1, 5	0-01	111	P ₁
1, 9	-001	011	√
4, 5	010-	100	P ₂
5, 13	-101	111	P ₃
9, 11	10-1	001	√
9, 13	1-01	011	√
11, 15	1-11	001	√
13, 15	11-1	001	√

2 Variabili			
Term.	ABCD	F ₁ F ₂ F ₃	P
1, 5, 9, 13	--01	011	P ₄
9, 11, 13, 15	1--1	001	P ₅

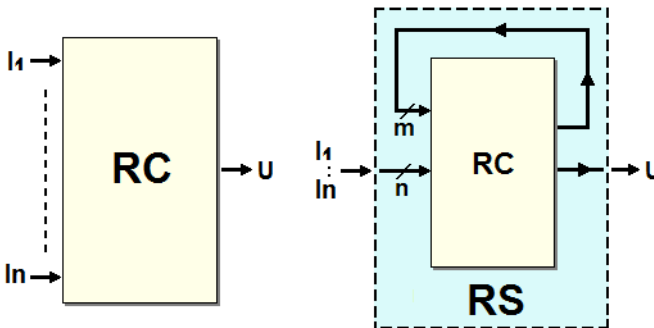
	F ₁			F ₂			F ₃			
	m ₁	m ₄	m ₅	m ₁	m ₉	m ₁₃	m ₉	m ₁₁	m ₁₃	m ₁₅
P ₀				X						
P ₁	X		X	X						
P ₂		X	X							
P ₃			X			X			X	
P ₄				X	X	X	X		X	
P ₅							X	X	X	X

Introduzione alle reti sequenziali

Un dispositivo digitale ben difficilmente sarà basato soltanto su reti combinatorie. Infatti, in una situazione reale diventa importante disporre di dispositivi in grado di memorizzare dati, generare sequenze e “rispondere” alle diverse condizioni che si possono verificare al variare del tempo.

5.1 Dalle reti combinatorie alle reti sequenziali

Abbiamo visto le reti combinatorie (RC, vedi figura, a sinistra) dove l'uscita U è, istante per istante, funzione dei soli ingressi I . Le reti combinatorie si identificano proprio per il fatto che ogni combinazione di ingressi produce *sempre* la stessa uscita $U = f(I_1, I_2, \dots, I_n)$.



Una *rete sequenziale* (RS, a destra in figura) non rispetta la regola appena esposta: la stessa combinazione di ingressi, applicata in istanti differenti, può generare uscite differenti.

Una rete sequenziale si può ottenere riportando in ingresso ad una rete combinatoria una o più delle sue uscite, come è mostrato in figura. Un collegamento di questo tipo si chiama “retroazione” (*feedback*), ossia *azione all'indietro*; si dice che la rete è *retroazionata*.

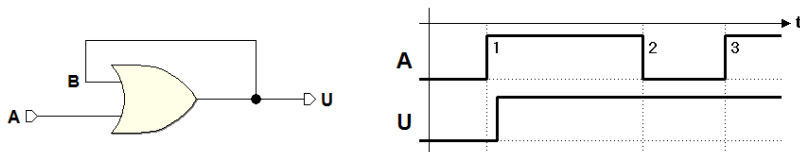
La connessione di m uscite della rete combinatoria ad altrettanti ingressi fa sì che il comportamento delle uscite rispetto agli ingressi dipenda, in generale, *anche dalle uscite stesse*. Se consideriamo ricorsivamente che le uscite attuali sono state prodotte dagli ingressi e uscite precedenti, e così via, andando indietro nel tempo, si arriva quindi ad affermare che le uscite dipendono dalla *storia precedente* degli ingressi, oltre che da quelli *attuali*.

Esprimendo lo stesso concetto con parole differenti, si può dire che il funzionamento della rete dipende dalla *sequenza degli ingressi*, che a loro volta producono una *sequenza sulle uscite*: non a caso una rete logica che presenta questo comportamento è denominata *rete sequenziale*.

Nel seguito si vedrà che, nelle reti sequenziali, l'uscita si può esprimere analiticamente come funzione, oltre che degli ingressi, anche della particolare condizione della rete in quel momento, chiamata *stato*. Come vedremo, il concetto di *stato* permetterà di esprimere in modo sintetico la *storia precedente* della rete.

5.1.1 Esempio introduttivo

Vediamo ora un semplice esempio di rete sequenziale, utilizzando la funzione OR che, come sappiamo, è una rete combinatoria. Costruiamo una *retroazione*, riportando l'uscita U ad uno dei due ingressi (B), come nella figura seguente:



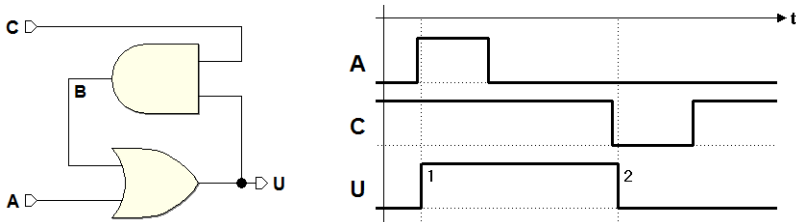
Cerchiamo di capire come agisce questa rete, operando, ovviamente, sull'unico ingresso disponibile A , essendo B già pilotato da U , e quindi non disponibile. Supponiamo che inizialmente A e U (e di conseguenza B) siano uguali a 0. Questa situazione è descritta nel diagramma temporale riportato accanto.

Se portiamo, ad un certo istante, l'ingresso A ad 1, l'uscita andrà a 1. Se la rete fosse combinatoria, con questo cambiamento avremmo esaurito il numero di casi possibili (due), tenendo conto del numero degli ingressi (uno). Azzerando nuovamente l'ingresso A , dovremmo riportare a 0 l'uscita U , ma in realtà questo non avviene. A causa della presenza del collegamento in retroazione tra U e B , l'uscita U rimane *forzata* ad 1 per qualunque valore sull'ingresso A e non sarà possibile ri-azzerarla.

Si dice che la rete ha “memorizzato” il valore 1. Si noti che nel diagramma è rappresentato, a livello indicativo, il tempo di propagazione della porta OR. Si tratta evidentemente di una rete non più combinatoria, bensì *sequenziale*, nella quale il valore dell'uscita dipende dalla *storia precedente* degli ingressi. Non è più possibile descrivere il suo funzionamento tramite una semplice tabella di verità, come abbiamo fatto con le reti combinatorie.

5.1.2 Memorizzare un bit di informazione: il flip-flop

La semplice rete sequenziale ora vista, opportunamente modificata, potrebbe essere utilizzata per immagazzinare un *bit di informazione*. Allo scopo, dovendo memorizzare sia il valore 0 che il valore 1, occorre trovare il modo di azzerare l'uscita dopo averla portata a 1. Una possibile modifica è rappresentata nella figura qui sotto, dove una porta AND è stata inserita nel percorso di retroazione ed è stato aggiunto un ingresso C :



La porta AND e l'ingresso C sono stati introdotti allo scopo di stabilire o rimuovere, di fatto, il collegamento tra U e B . Infatti, se $C = 1$ la rete si comporta come prima, poiché ogni variazione di U viene trasferita, tramite la porta AND, su B ($B = U \text{ and } 1 = U$); se $C = 0$, invece, B è a 0, qualunque sia il valore di U ($B = U \text{ and } 0 = 0$).

Nel diagramma temporale in figura si ipotizza di partire con l'ingresso A e l'uscita U a 0, e con l'ingresso C a 1. In queste condizioni, l'attivazione dell'ingresso A porta a 1 l'uscita. Il nuovo valore $U = 1$, riportato dalla porta AND sull'ingresso B della OR, fa sì che ulteriori variazioni di A non si possano più ripercuotere sull'uscita stessa. Abbiamo memorizzato un bit a 1.

Per riportare a 0 l'uscita U sarà necessario *aprire* il percorso di retroazione, applicando sull'ingresso C il valore 0, come si vede nel diagramma temporale: l'azione di azzeramento dell'uscita rappresenta la memorizzazione di un bit a 0. Ulteriori variazioni di C non alterano la situazione. Anche qui sono stati rappresentati, indicativamente, i tempi di propagazione.

Si noti che ciascun ingresso può essere considerato sotto due punti di vista:

- lo scopo che tale comando ha nella rete e
- la modalità logica con cui l'effetto è prodotto.

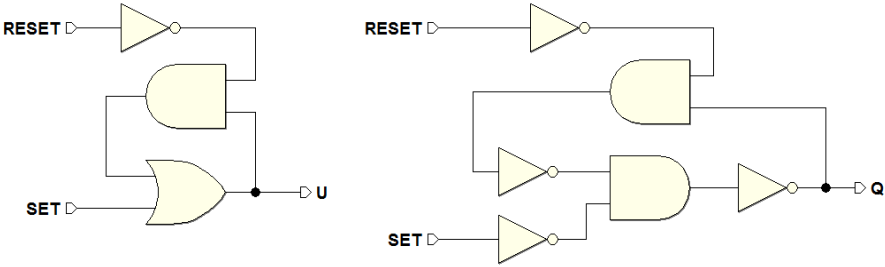
In questo caso, l'ingresso A produce l'effetto di memorizzare un 1 sull'uscita, e questo si verifica quando viene portato ad 1. Si dice allora che l'ingresso è “attivo alto”, ossia svolge il suo compito quando è *portato ad 1*; la sua condizione di “inattività” è rimanere a 0. L'ingresso C , invece, ha la funzione di memorizzazione uno 0, e questo viene ottenuto quando viene *portato a 0*: si dice che tale ingresso è “attivo basso”, mentre la sua condizione di “inattività” è rimanere a 1.

Con questa rete sequenziale siamo in grado, quindi, di *memorizzare un bit*, il cui valore viene mantenuto fino alla memorizzazione di un nuovo valore. Si tratta di una delle tante possibili realizzazioni di una *cella elementare di memoria*, chiamata anche *bistabile* o, in modo più gergale ma molto usato, “flip-flop”. I flip-flop sono gli elementi logici base con i quali si costruiscono, in generale, i sistemi digitali sequenziali.

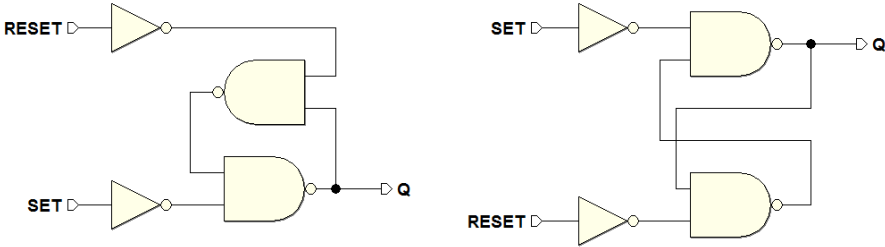
Proviamo ora a modificarne leggermente il comportamento, rendendo simmetrico e più intuitivo l’azionamento. Si vuole arrivare, per gradi, alla rete analizzata nel prossimo paragrafo: la classica cella elementare di memoria denominata flip-flop *Set-Reset*. Allo scopo, eseguiamo queste modifiche:

- aggiungiamo un NOT davanti all’ingresso *C*, per rendere *attivo alto* il comando di memorizzazione dello zero;
- denominiamo *RESET* il nuovo ingresso ottenuto (perché azzerava l’uscita);
- cambiamo nome all’ingresso *A* in *SET* (perché porta a uno l’uscita);
- cambiamo nome all’uscita *U* in *Q* (per semplice consuetudine).

La rete risultante da questa modifica è rappresentata nella figura seguente (a sinistra). Proseguendo con le trasformazioni, applichiamo il teorema di De Morgan e sostituiamo la OR con una AND (a destra in figura):



Le NOT in serie alle AND ci suggeriscono di usare direttamente delle NAND, per cui otteniamo la rete qui sotto (a sinistra). La trasformazione è ora completata; conviene soltanto ridisegnare la rete in modo da evidenziarne la simmetria, senza alterarne i collegamenti (a destra in figura):



Quest’ultima è la versione del flip-flop *Set-Reset* esaminata nel prossimo paragrafo.

5.1.3 Tipi logici e tipi di comando dei flip-flop

Nel paragrafo precedente abbiamo visto la derivazione della struttura sequenziale denominata flip-flop *Set-Reset*. Questa è soltanto uno dei diversi tipi di flip-flop impiegati per la memorizzazione di un bit di informazione. I flip-flop sono i blocchi fondamentali con cui sono costruite le reti sequenziali più complesse: li distinguiamo tra loro per “*tipo logico*” e per “*tipo di comando*”.

Tipo logico

Il flip-flop esaminato, la cui funzione consiste nell’attivare l’uscita (azione di *Set*) o disattivarla (azione di *Reset*), è dunque di tipo logico *Set-Reset*. Nel seguito prenderemo in esame altri tipi logici, quali il *D* (*Delay*), che memorizza sull’uscita il valore dell’unico ingresso *D*, e la variante *JK* del *Set-Reset*, che sostituisce gli ingressi *J* e *K* a *S* e *R*, e che prevede la funzione di inversione dello stato dell’uscita. Vedremo anche i tipi logici *T* e *E*, che derivano, rispettivamente, dal *JK* e dal *D*.

Il tipo logico è descritto, come vedremo, dalla relativa “*tabella di funzionamento*” che indica i valori delle uscite in funzione degli *ingressi logici*, cioè quelli che caratterizzano il tipo logico del flip-flop, e che gli danno il nome.

Tipo di comando

Il tipo di comando descrive invece le modalità con cui gli ingressi agiscono, e può essere di tre tipi: *diretto*, *abilitato a livello*, o *abilitato sul fronte*.

Nel primo caso l’azione degli ingressi logici non è subordinata a nessun altro ingresso di abilitazione o sincronizzazione, controllando direttamente il comportamento del flip-flop, che in questo caso è chiamato “*asincrono*”.

Negli altri due casi è presente un ingresso aggiuntivo avente la funzione di abilitare/disabilitare gli ingressi logici. L’abilitazione può avvenire in funzione del semplice livello logico dell’ingresso (comando *abilitato a livello*), oppure in presenza di una *transizione di livello logico* (“*fronte*” di *salita* o di *discesa*). In questo ultimo caso, il flip-flop, è chiamato “*sincrono*”.

Iniziamo a considerare i tipi a comando diretto. Sarà poi necessario, prima di proseguire con gli altri tipi di comando, esaminare alcuni concetti importanti, come l’inizializzazione e la sincronizzazione delle reti sequenziali.

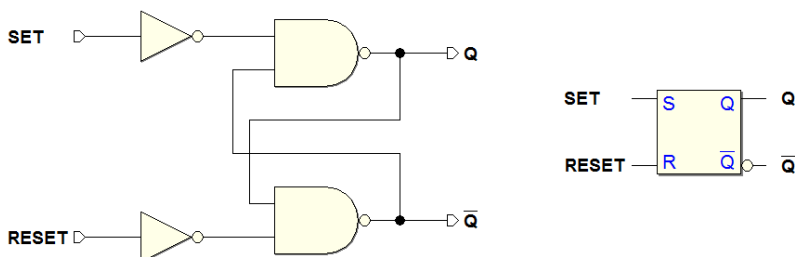
5.2 Flip-flop a comando diretto

Esaminiamo ora, nella versione a comando diretto, i tre tipi logici: *SR*, *D* e *JK*. Il tipo *SR*, che è già stato introdotto nelle sezioni precedenti, sarà qui riesaminato in alcune delle possibili varianti.

5.2.1 Flip-flop SR (a comando diretto)

SR, versione NAND, comandi attivi alti

Lo schema nella figura seguente rappresenta un flip-flop di tipo *Set-Reset* (SR), realizzato con porte NAND (e NOT), con gli ingressi *attivi alti*. Rispetto alla versione esaminata prima, è stata aggiunta l'uscita \bar{Q} che, nelle condizioni di normale funzionamento, come vedremo nel seguito, ha valore logico opposto a quello di Q . Questa è la struttura che realizza il tipo logico *Set-Reset a comando diretto*. Nella parte destra della figura è disegnato il simbolo logico che lo rappresenta negli schemi logici:



Come abbiamo visto in precedenza, è una rete sequenziale in grado di *memorizzare un bit*. L'attivazione dell'ingresso *SET* memorizza un 1 nella cella, mentre l'ingresso *RESET* impone uno 0. Naturalmente, stante questa definizione, ha poco senso attivare insieme *SET* e *RESET*. Qui sotto è riportata la sua *tabella di funzionamento*, che sintetizza quanto ora descritto a parole. La tabella riporta entrambe le uscite Q e \bar{Q} :

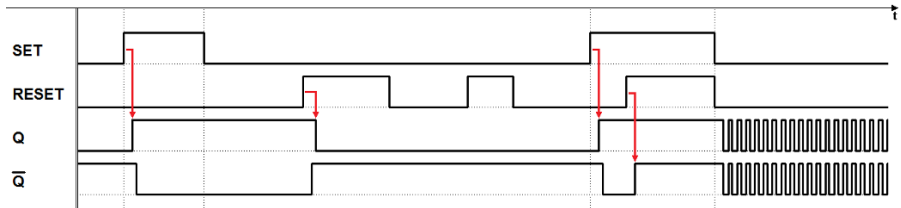
Flip-flop <i>Set-Reset</i> (comandi attivi alti)				
<i>SET</i>	<i>RESET</i>	Q	\bar{Q}	
0	0	Q_p	\bar{Q}_p	Stato precedente
1	0	1	0	Comando di <i>SET</i>
0	1	0	1	Comando di <i>RESET</i>
1	1	1	1	Non ammesso

La tabella ci dice che, se gli ingressi sono mantenuti a *riposo* (a 0), il flip-flop mantiene lo stato precedente (Q_p , \bar{Q}_p), cioè mantiene in uscita il bit precedentemente memorizzato. Le due righe successive descrivono l'azione dei comandi di *SET* e di *RESET*, visti sopra.

L'ultima riga della tabella considera, per completezza, anche il caso *non ammesso* di attivazione contemporanea degli ingressi di comando: nel caso della rete in esame, questa combinazione produce l'attivazione contemporanea delle due uscite e l'uscita \bar{Q} non è più il negato di Q . La configurazione non

ammessa meriterà un discorso a parte, poco più avanti, perché questa condizione limite presenta degli aspetti tecnicamente interessanti che, tra l'altro, dipendono dalla particolare realizzazione circuitale.

Il diagramma temporale seguente, ottenuto con il simulatore *Deeds*, mostra il comportamento del flip-flop in varie condizioni di pilotaggio. La durata dei segnali e la scala di visualizzazione del diagramma sono state scelte in modo da mettere in evidenza i *tempi di ritardo*, rilevati dalla simulazione, tra l'attivazione di *SET*, *RESET* e le uscite *Q* e \bar{Q} :



Analizziamo ora, punto per punto, il comportamento della rete.

- Nell'intervallo simulato, si fa l'ipotesi che $Q = 0$ e $\bar{Q} = 1$, prima della attivazione di *SET* a 1;
- L'attivazione di *SET* porta l'uscita *Q* a 1 e, corrispondentemente, \bar{Q} a 0;
- Dopo la disattivazione di *SET*, nel flip-flop si mantiene memorizzato $Q = 1$, fino alla successiva attivazione di *RESET*;
- L'attivazione di *RESET* azzerà l'uscita *Q*;
- Dopo la disattivazione di *RESET*, viene mantenuto memorizzato $Q = 0$;
- Ulteriori attivazioni e disattivazioni dell'ingresso *RESET* non producono cambiamenti, perché l'uscita *Q* è già azzerata;
- L'uscita *Q* cambia soltanto alla successiva attivazione di *SET*, che la riporta a 1.

Infine, il diagramma illustra gli effetti dell'applicazione della configurazione di ingresso *non ammessa*. Se *SET* e *RESET* sono attivate entrambe, le uscite *Q* e \bar{Q} vanno entrambe a 1. E' facilmente verificabile, esaminando la rete logica del flip-flop, che in questa condizione la retroazione non ha effetto all'ingresso dei NAND, poiché sull'altro ingresso hanno uno 0. La rete ha perso, insieme alla retroazione, anche la memorizzazione del dato! Da questa condizione anomala si uscirebbe senza problemi se venisse disattivato prima uno dei due ingressi, e poi l'altro: in questo caso ci troveremmo a passare da una configurazione degli ingressi che forzerebbe la memorizzazione di un valore noto, coerente con il valore di *SET* e di *RESET* in quel momento.

Nel diagramma temporale, invece, è rappresentata la *condizione limite* in cui gli ingressi *SET* e *RESET* sono disattivati contemporaneamente. *SET* e *RESET* inattivi corrispondono, nel funzionamento normale, alla memorizzazione di un valore; nel caso della sequenza esaminata, però, tale informazione

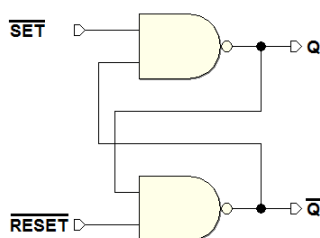
è andata perduta. Il simulatore ci mostra un comportamento oscillante delle uscite, che commutano insieme, periodicamente, tra i due livelli.

Questo risultato della simulazione è dovuto al modello semplificato e idealizzato dei componenti, dove le porte logiche hanno tutte lo stesso identico tempo di ritardo (di trasporto). La commutazione degli ingressi provoca un cambiamento contemporaneo delle uscite, che sono riportate contemporaneamente agli ingressi dalla retroazione, causando nuovamente un ulteriore cambiamento del livello di entrambe le uscite, e così via.

In una rete reale questo comportamento è estremamente improbabile. I componenti logici reali sono in realtà degli amplificatori non lineari, e uno studio approfondito del loro comportamento nella situazione limite ora considerata potrebbe risultare anche molto complesso, coinvolgendo concetti quali la *metastabilità*, di cui si parlerà più avanti. Ai nostri fini, però, può bastare osservare che i tempi di propagazione di due porte reali sono tra loro simili, ma mai identici. Una delle porte risulta quindi più veloce dell'altra, e la retroazione finisce per stabilizzare le uscite entro poco tempo, portando il flip-flop nella condizione di memorizzazione (anche se su di un valore non noto a priori).

SR, versione NAND, comandi attivi bassi

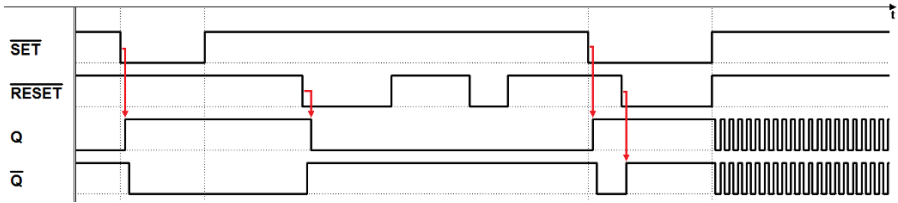
Se si eliminano i due invertitori, si ottiene lo stesso flip-flop, ma con i comandi *attivi bassi*. Per la sua semplicità, il flip-flop SR con questo tipo di comandi è la *struttura base* sulla quale sono realizzati gli altri tipi logici di flip-flop (nel seguito questa rete sarà ricordata come “*cella base*”):



Non ci sono differenze sostanziali rispetto al tipo con i comandi attivi alti, ovviamente: cambia solo il livello logico del comando. Qui di seguito la sua *tabella di funzionamento*; come si vede, è analoga alla precedente, salvo che gli ingressi sono attivi bassi:

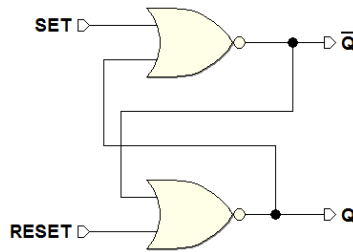
Flip-flop <i>Set-Reset</i> (comandi attivi bassi)				
\overline{SET}	\overline{RESET}	Q	\overline{Q}	
1	1	Q_p	\overline{Q}_p	Stato precedente
0	1	1	0	Comando di <i>SET</i>
1	0	0	1	Comando di <i>RESET</i>
0	0	1	1	Non ammesso

Il diagramma temporale seguente propone una sequenza di simulazione analoga a quella del caso precedente, ma con i segnali di ingresso complementati:



SR, versione NOR

Per completezza, accenniamo ora alla versione del flip-flop *Set-Reset* realizzato con porte NOR. Per ottenerla, ritorniamo alla rete composta solo da un AND e da un OR, vista in precedenza, ma questa volta la trasformiamo in una rete con sole porte NOR. Trascurando i passaggi intermedi, nei quali vengono eliminate anche le porte NOT, si ottiene la seguente rete:



Questa cella ha gli ingressi *attivi alti*. Si noti che, differentemente dalla versione NAND, l'uscita Q è generata dalla porta che riceve il segnale di *RESET*, e l'uscita \bar{Q} da quella che riceve *SET*. Nonostante la sua semplicità, questa versione NOR è tuttavia poco usata, poiché in molte tecnologie è più conveniente usare porte NAND. Questa la sua tabella di funzionamento:

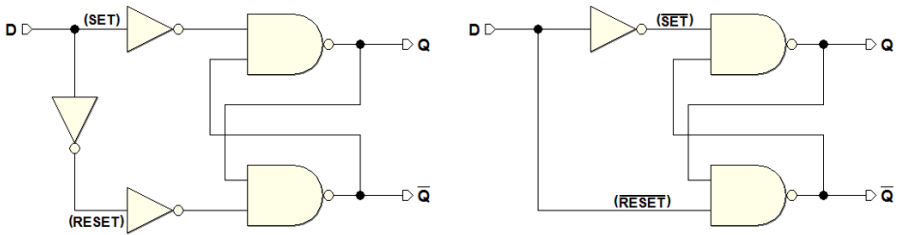
Flip-flop <i>Set-Reset</i> (con porte NOR)				
SET	$RESET$	Q	\bar{Q}	
0	0	Q_p	\bar{Q}_p	Stato precedente
1	0	1	0	Comando di <i>SET</i>
0	1	0	1	Comando di <i>RESET</i>
1	1	0	0	Non ammesso

È simile a quella del flip-flop realizzato con porte NAND, salvo il diverso comportamento nel *caso non ammesso* in cui, all'attivazione di entrambi gli ingressi, la rete risponde con tutte e due le uscite a 0. Resta valida l'analisi fatta in precedenza circa la disattivazione contemporanea degli ingressi.

5.2.2 Flip-flop D (a comando diretto)

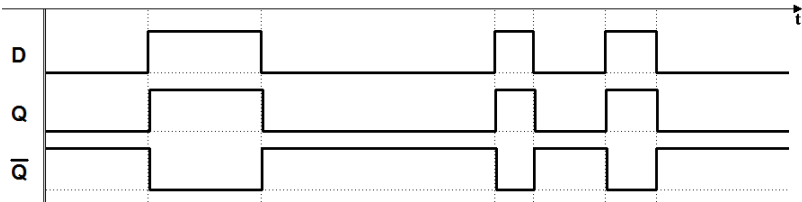
Introduciamo qui il concetto di flip-flop di tipo logico D. Si vuole disporre di un unico ingresso D di dato, invece che di due separati controlli SET e $RESET$: l'idea è di memorizzare il bit semplicemente presentandolo all'ingresso, ma anche di ovviare ai problemi legati alla configurazione non ammessa.

Nello schema nella figura qui sotto, a sinistra, proviamo a rendere unico l'ingresso, aggiungendo un NOT alla struttura del flip-flop SR con ingressi attivi alti: il dato in ingresso D è applicato al SET , mentre al $RESET$ colleghiamo \overline{D} . Eliminati i due NOT in cascata, otteniamo la rete mostrata sulla destra:



Il NOT assicura anche che gli ingressi \overline{SET} e \overline{RESET} della cella base non avranno mai lo stesso livello logico, evitando quindi la condizione critica discussa nei paragrafi precedenti.

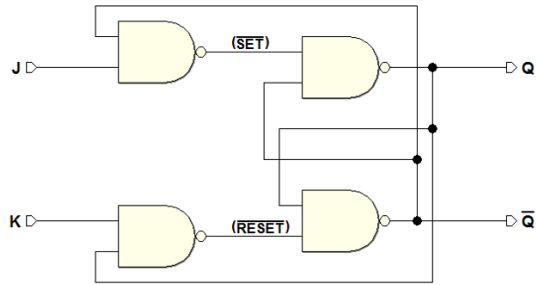
Tuttavia, come si nota nel diagramma temporale, l'uscita Q riproduce sempre l'ingresso D . Non esiste una configurazione di comando che memorizzi il dato, per cui questo circuito, nella versione a comando diretto, è inutile:



Il flip-flop D diventa invece un elemento funzionale e importante nelle realizzazioni a *comando abilitato*, come vedremo poco più avanti.

5.2.3 Flip-flop JK (a comando diretto)

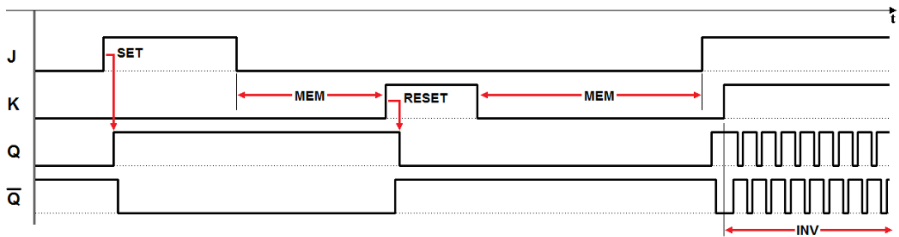
Un approccio efficace per eliminare la configurazione non ammessa è rappresentato dal flip-flop di tipo logico JK a comando diretto. Questo circuito deriva dal flip-flop *Set-Reset* con gli ingressi attivi bassi, con l'aggiunta di due porte NAND, come rappresentato nella figura che segue. Ai nuovi ingressi è assegnato il nome di J e K (da cui il nome JK, in onore di *Jack Kilby* per il suo contributo alla nascita dell'elettronica integrata).



Analizzando lo schema, si vede che l'ingresso J , attraverso una delle due nuove NAND, pilota il \overline{SET} , condizionato da \overline{Q} ; in modo simmetrico, l'ingresso K pilota il \overline{RESET} , condizionato da Q . Quindi, il \overline{SET} e il \overline{RESET} non potranno mai essere attivati nello stesso momento. Qui la tabella di funzionamento che descrive come gli ingressi J e K agiscono sulle uscite Q e \overline{Q} :

Flip-flop JK				
J	K	Q	\overline{Q}	
0	0	Q_p	\overline{Q}_p	Stato precedente
1	0	1	0	Comando di SET
0	1	0	1	Comando di $RESET$
1	1	\overline{Q}_p	Q_p	Inverte le uscite

Nella prima riga della tabella, J e K valgono entrambi 0, per cui il \overline{SET} e il \overline{RESET} sono ad 1, e le uscite mantengono il precedente valore. Nella simulazione temporale qui sotto, ritroviamo questa configurazione degli ingressi negli intervalli di tempo identificati come “MEM”:



Nella seconda riga della tabella, $J = 1$ e $K = 0$, per cui è attivato l'ingresso di SET , e l'uscita Q è portata a 1 (operazione identificata come “SET” nel tracciato temporale). Nella terza riga abbiamo il caso opposto, e l'uscita Q è azzerata (“RESET” nella figura).

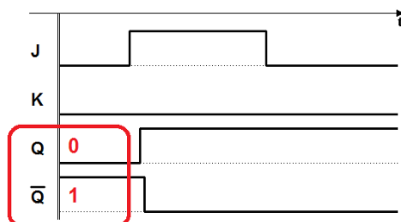
Il flip-flop JK si differenzia dal tipo SR per il fatto che la configurazione non ammessa di quest'ultimo è utilizzata per compiere una funzione utile, l'inversione dei valori delle uscite (come indicato nell'ultima riga della tabella). Il termine inglese usato per indicare il comportamento di inversione è “Toggle”.

Quando entrambi gli ingressi J e K sono a 1, viene attivato \overline{SET} se $Q = 0$, oppure il \overline{RESET} se $Q = 1$, provocando quindi il cambiamento delle uscite (intervallo di tempo indicato come “INV” nel diagramma temporale).

Si vedrà nel seguito che la funzionalità di *Toggle* è pienamente sfruttabile solo nei flip-flop *JK abilitati sul fronte*. Nelle condizioni proposte dal diagramma temporale ora visto, ossia con J e K mantenuti ad 1 per un tempo sufficientemente lungo, le uscite Q e \overline{Q} invertono *continuamente* il loro valore. Infatti, ad ogni cambiamento delle uscite Q e \overline{Q} , la retroazione inverte a sua volta i valori presenti su \overline{SET} e sul \overline{RESET} , imponendo a sua volta un'altra inversione, e così via, dando origine ad un comportamento ciclico.

5.3 Inizializzazione di una rete sequenziale

A questo punto, prima di proseguire, è necessario fare una piccola digressione. Negli diagrammi temporali che abbiamo esaminato, le sequenze descritte iniziano volutamente sempre con $Q = 0$ e $\overline{Q} = 1$, come nell'esempio seguente:



Si è cioè scelto un determinato istante, durante il normale funzionamento della rete, e si è deciso di *cominciare ad osservarla* da quel momento. Nei precedenti paragrafi, cioè, per non complicare l'esposizione, si è volutamente nascosto un aspetto importante che coinvolge tutte le reti sequenziali: la necessità di avviare la rete da una *configurazione nota*. Questo problema coinvolge tutte le reti sequenziali e sarà ripreso e approfondito anche più avanti, quando esamineremo gli aspetti della loro *progettazione*.

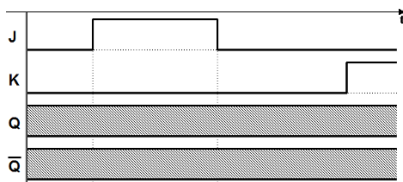
In termini tecnici si dice che, all'accensione, una rete sequenziale deve essere “*inizializzata*”, in modo che nel normale funzionamento possa lavorare coerentemente. In una rete reale saranno presenti molti flip-flop: all'attivazione del sistema, in assenza di accorgimenti particolari, ognuno di questi tenderà ad assumere un *valore casuale*.

E' comprensibile che, per una rete complessa, questo è inaccettabile, per cui è necessario ricorrere a tecniche circuitali che consentano di predisporre ogni flip-flop ad un valore noto, *prima* di avviare il normale funzionamento della rete. Nelle reti più semplici, l'inizializzazione potrebbe essere irrilevante. Per esempio, può esserlo per un circuito che faccia lampeggiare un LED su di un pannello: poco importa se, all'accensione del circuito, il LED inizi subito ad

illuminarsi, o lo faccia mezzo ciclo di lampeggio dopo. Nella quasi totalità dei casi reali, tuttavia, non è accettabile che la rete inizi da valori *casuali*.

Pensiamo ad una rete digitale che controlla qualcosa di *intrinsecamente pericoloso*, come ad esempio l'apertura delle paratie della diga di un bacino idroelettrico. Non possiamo permetterci che all'accensione del sistema i flop-flop si posizionino casualmente nella configurazione che apre le paratie: dovremo invece adottare tutti gli accorgimenti del caso affinché queste restino *rigorosamente chiuse* e si aprano solo a seguito di un esplicito comando.

Torniamo ora al flip-flop JK a comando diretto. Nel caso di un componente reale, all'accensione, l'uscita Q della cella elementare di memoria si porterà casualmente ad un valore *non noto a priori*. Da quel momento, la rete inizierà a funzionare (anche se partendo da un valore casuale, 0 o 1), e seguirà le sorti imposte dall'evoluzione degli ingressi. Tuttavia, se proviamo a *simulare* la sua rete, ci accorgiamo che il simulatore non riesce a *risolvere* il comportamento della rete e ci restituisce il risultato visibile nella figura seguente, dove le bande tratteggiate rappresentano un *valore sconosciuto (unknown)*:



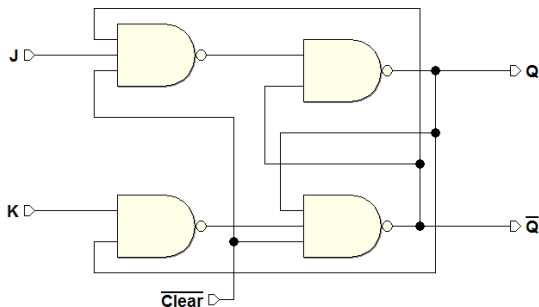
Questo comportamento del simulatore è *formalmente corretto*: non conoscendo il valore iniziale delle uscite Q e \bar{Q} , il simulatore non è in grado di calcolarne i valori successivi. Infatti, come visto, questi dipendono dal valore degli ingressi, ma anche dal valore precedente delle uscite stesse, non noto.

Il simulatore è uno *strumento di sviluppo e verifica* e, affinché sia efficace, deve consentire al progettista di accorgersi degli errori e delle dimenticanze. In questo caso, il simulatore ci dice che la rete *non è formalmente in grado* di iniziare a lavorare da una *configurazione nota*.

Se il simulatore risolvesse queste situazioni 'ipotizzando' dei valori di partenza, non ci renderebbe un gran servizio, perché potrebbe mascherare, con questo criterio, potenziali errori di progettazione.

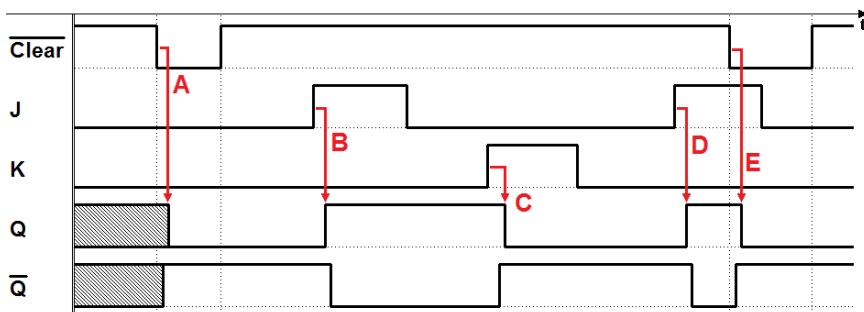
5.3.1 Ingressi di inizializzazione dei flip-flop

Come è stato possibile ottenere la simulazione della rete del flip-flop JK, dunque? In realtà, è stata simulata un'altra rete, quella riportata nella figura seguente, ove è presente un ingresso *ausiliario* di inizializzazione, il *Clear (cancellazione)*, che agisce su due dei NAND della rete (l'ingresso è indicato con la *negazione* per mettere in evidenza che è *attivo basso*):



Come si può dedurre dallo schema logico, quando $\overline{Clear} = 1$, la rete si comporta come se il nuovo ingresso non ci fosse. Invece, quando è portato a 0, sulla cella elementare viene *forzata* la configurazione che azzerà l'uscita Q . È importante notare che l'ingresso \overline{Clear} è *prioritario* rispetto a J e K : infatti, per tutto il tempo in cui \overline{Clear} è attivo, non solo *mantiene azzerata l'uscita*, ma impedisce a J e K di influire sulla cella elementare.

Nella figura seguente osserviamo la simulazione della rete, con in evidenza l'attivazione dell'ingresso \overline{Clear} :



Come si vede nel tracciato, le uscite Q e \overline{Q} sono inizialmente indeterminate, ma l'attivazione (A) dell'ingresso \overline{Clear} ne provoca la definizione (a parte i ritardi, visibili in figura, dovuti ai tempi di propagazione). Poi, la disattivazione del \overline{Clear} permette al flip-flop di lavorare liberamente (B, C e D), obbedendo al controllo da parte degli ingressi J e K . Infine, sulla destra del tracciato, a titolo di esempio, è messa in evidenza una ulteriore attivazione (E) del \overline{Clear} , per cui il flip-flop è nuovamente azzerato.

In generale i flip-flop, per ragioni di completezza e versatilità, possiedono due ingressi di inizializzazione: oltre al \overline{Clear} , anche il \overline{Preset} . Il compito del \overline{Preset} è perfettamente simmetrico rispetto al \overline{Clear} : la sua azione è di portare a 1 l'uscita.

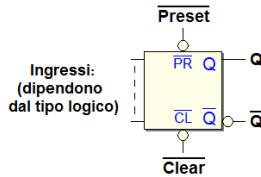
Osserviamo che la funzione dei due ingressi \overline{Clear} e \overline{Preset} è funzionalmente equivalente a quella del \overline{RESET} e del \overline{SET} di un flip-flop RS a comandi attivi

bassi. Tuttavia, lo scopo degli ingressi di inizializzazione resta quello per cui nascono, e non è buona pratica utilizzarli diversamente.

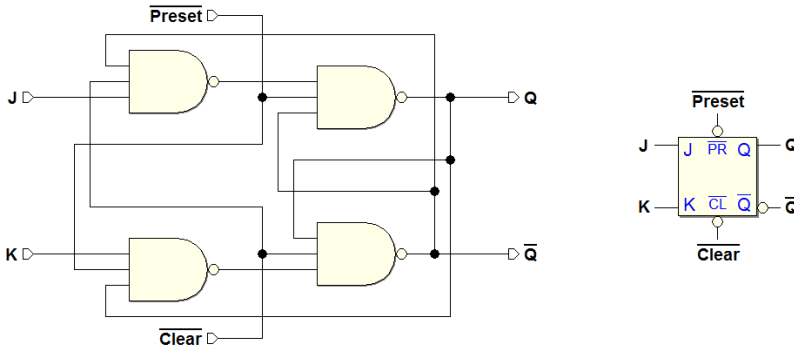
Per questo motivo, il \overline{Clear} e il \overline{Preset} sono anche da considerarsi *mutuamente esclusivi*: ai fini della inizializzazione del flip-flop se ne utilizza uno solo dei due, scelto sulla base delle esigenze di progetto, mentre l'altro sarà collegato a una costante, in modo da essere sempre inattivo.

Indipendentemente dal tipo logico, gli ingressi \overline{Clear} e \overline{Preset} sono presenti in tutti i flip-flop: sia quelli commerciali, sia quelli presenti nelle librerie di componenti dei sistemi CAD.

Nella figura seguente vediamo le terminazioni comuni a tutti i tipi logici: gli ingressi \overline{Clear} e \overline{Preset} e le uscite Q e \overline{Q} . Gli altri ingressi sono qui rappresentati in modo generico, dato che variano in dipendenza dal tipo logico.



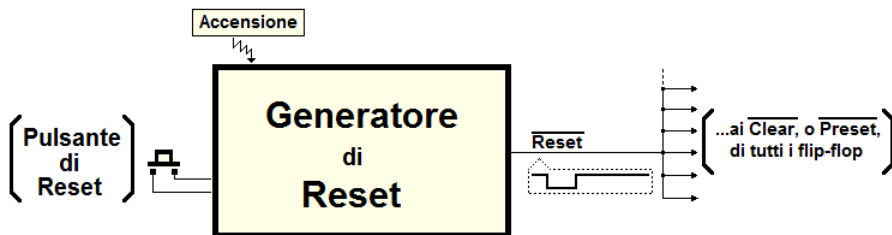
Per completezza, qui sotto, a sinistra, è rappresentato lo schema logico completo del flip-flop JK a comando diretto, con gli ingressi J e K , le uscite Q e \overline{Q} , e gli ingressi \overline{Clear} e di \overline{Preset} ; sulla destra è riportato, invece, il corrispondente simbolo circuitale:



5.3.2 Generazione del segnale di inizializzazione

Dopo avere discusso della necessità di inizializzare le reti sequenziali, dobbiamo ora esaminare come viene prodotto, all'accensione del sistema, il segnale che, portato all'ingresso \overline{Clear} , o \overline{Preset} , dei singoli flip-flop, ne consente l'inizializzazione.

Nella prossima figura è rappresentato quello che viene chiamato *Generatore di Reset (Reset Generator)*: un circuito in parte analogico, in parte digitale, che in questa sede esamineremo solo dal punto di vista funzionale.



Il circuito genera *in automatico*, all'accensione del sistema, un impulso di durata convenuta sulla linea di \overline{Reset} , collegata a tutti i flip-flop presenti nella rete (o al \overline{Clear} , o al \overline{Preset} , a seconda delle necessità). Spesso è presente anche un pulsante, a disposizione dell'utente, per re-inizializzare il sistema manualmente (come, per esempio, nel caso dei personal computer).

L'attivazione della linea di \overline{Reset} , all'accensione, mantiene bloccati tutti i flip-flop del sistema durante l'iniziale *transitorio di alimentazione*. Infatti, i dispositivi che alimentano il sistema impiegano un certo tempo (qualche decina di mS) per portare le tensioni che alimentano il circuito da zero ai valori nominali; il generatore mantiene il \overline{Reset} attivo per il tempo necessario, per poi disattivarlo e permettere quindi al sistema di iniziare a lavorare.

5.4 Flip-flop a comando abilitato a livello

I tre tipi logici SR, D e JK, presentati nelle precedenti sezioni, avevano in comune il fatto che l'azione degli ingressi logici non è subordinata a nessun altro ingresso. Nella progettazione di un sistema digitale, tuttavia, è spesso necessario che le uscite dei flip-flop che lo compongono cambino tutte nel *medesimo istante*, ossia che siano *sincronizzate* tra di loro.

In questa sezione ci occupiamo dei flip-flop *abilitati a livello*, dove un ingresso apposito condiziona l'azione degli ingressi. Questi tipi di flip-flop sono chiamati anche con il termine inglese *Latch*¹.

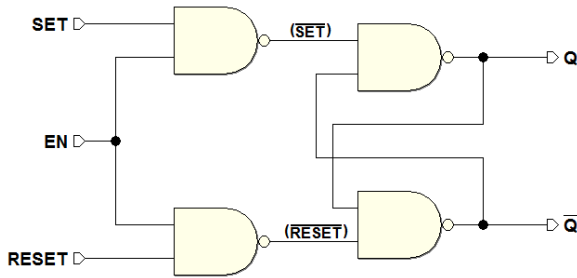
Per chiarezza espositiva, le reti logiche dei flip-flop qui presentati non includeranno i circuiti relativi alla *inizializzazione* (non saranno presenti, nelle descrizioni, gli ingressi di \overline{Clear} e di \overline{Preset}). Si tenga ovviamente presente che in realtà tutti i flip-flop considerati possiedono tali ingressi, come descritto in precedenza.

5.4.1 Flip-flop SR con abilitazione a livello (SR-Latch)

Nello schema seguente, la struttura base del flip-flop di tipo SR è stata integrata con due porte NAND. Le due nuove porte fanno in modo che gli ingressi

¹ Il verbo onomatopico "to latch" indica l'azione di chiudere il chiavistello di una porta e, per metafora, qui il termine indica la presenza di un passaggio, attraverso il quale passano i dati, che può essere chiuso o aperto, a comando.

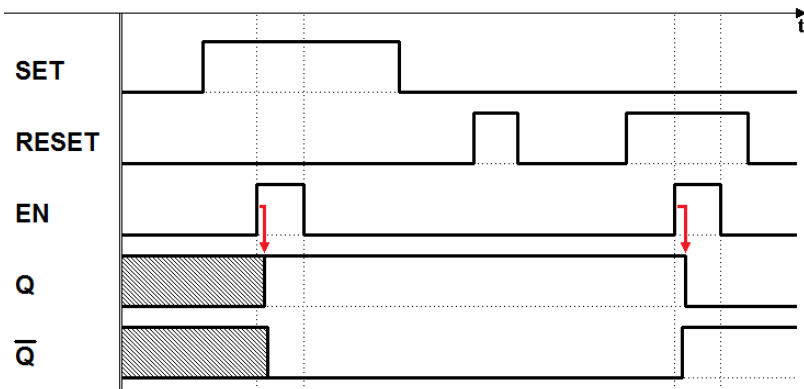
SET e *RESET* (attivi alti), agiscono sulla cella base soltanto se l'ingresso *EN* (*Enable*) è a 1. Se $EN = 1$, il comportamento complessivo della rete è identico a quello del flip-flop SR a comando diretto con ingressi attivi alti.



Se $EN = 0$, i due NAND che condizionano gli ingressi generano un 1, indipendentemente dal valore degli ingressi *SET* e *RESET*, per cui la cella base si trova nella situazione che mantiene invariato il valore delle uscite. Qui sotto la tabella di funzionamento, che riassume quanto descritto a parole:

Flip-flop <i>Set-Reset</i> con abilitazione a livello					
<i>EN</i>	<i>SET</i>	<i>RESET</i>	<i>Q</i>	\overline{Q}	
0	—	—	Q_p	$\overline{Q_p}$	Stato precedente
1	0	0	Q_p	$\overline{Q_p}$	Stato precedente
1	1	0	1	0	Comando di <i>SET</i>
1	0	1	0	1	Comando di <i>RESET</i>
1	1	1	1	1	Non ammesso

Nella simulazione temporale riportata qui sotto osserviamo il comportamento del flip-flop SR al variare degli ingressi (la configurazione non ammessa non è stata inclusa, per semplicità):



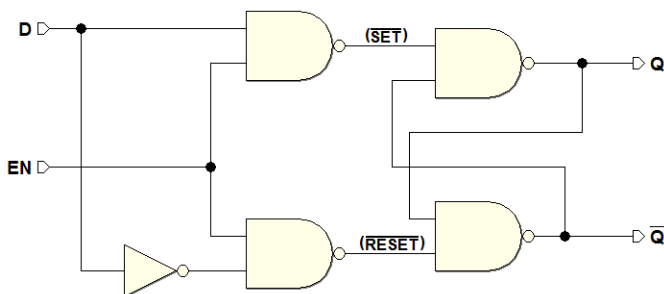
Nella prima parte del tracciato, non potendo ipotizzare un valore iniziale, il simulatore indica le uscite come *indefinite*. La prima attivazione dell'ingresso

EN consente all'ingresso SET di portare a 1 l'uscita Q . Successivamente, il flip-flop mantiene il valore delle sue uscite, dato che l'ingresso EN non è attivo. Poi EN è attivato nuovamente, mentre l'ingresso $RESET$ è attivo, producendo l'azzeramento dell'uscita Q .

L'ingresso EN è usato, come si è visto, in modo da confinare i cambiamenti delle uscite del flip-flop negli intervalli di tempo in cui è attivo. Riducendo la durata di tali intervalli, si ottiene una prima forma di *sincronizzazione*, come si vedrà nel seguito.

5.4.2 Flip-flop D con abilitazione a livello (*D-Latch*)

Nello schema qui sotto, alla struttura del flip-flop di tipo SR con abilitazione a livello è stato aggiunto un NOT. L'ingresso SET del flip-flop SR è stato rinominato in D ; questo, negato, pilota l'ingresso $RESET$:



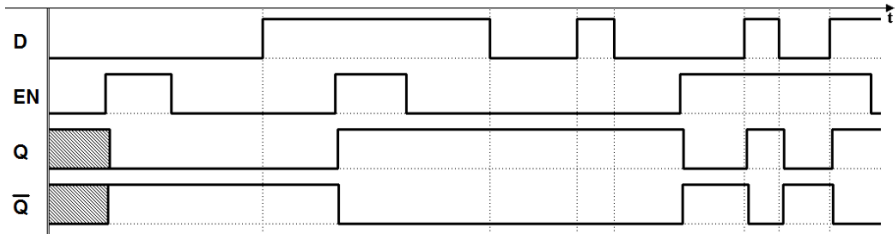
Abbiamo così ottenuto un flip-flop di tipo D, ma questa volta è presente l'ingresso di abilitazione EN (questo flip-flop è usualmente denominato, per brevità, *D-Latch*). La configurazione non ammessa degli ingressi SET e $RESET$ è automaticamente eliminata dal NOT, come avevamo già visto nel caso del flip-flop D a comando diretto. Tuttavia, grazie alla presenza dell'ingresso di abilitazione, in questo caso riusciamo a mantenere la possibilità di *memorizzare un bit*. Ecco la *tabella di funzionamento* di questo flip-flop:

Flip-flop D con abilitazione a livello (<i>D-Latch</i>)				
EN	D	Q	\bar{Q}	
0	—	Q_p	\bar{Q}_p	Stato precedente
1	1	1	0	Comando di SET
1	0	0	1	Comando di $RESET$

Quando EN è a 1 (attivo), l'uscita Q ripete il valore presente all'ingresso D ; mentre, con $EN = 0$, non c'è trasmissione tra l'ingresso D e la cella base di memoria, che mantiene il suo valore.

Grazie alla sua semplicità ed economicità, il flip-flop D-Latch è molto usato, e costituisce l'elemento su cui sono realizzati molti tipi di *registri* e di *memorie*

a *semiconduttore* che trovano applicazione nelle reti sequenziali in genere e nei computer. Per immagazzinare l'informazione, infatti, è sufficiente presentare il *bit da memorizzare* all'ingresso D , e quindi attivare e poi rilasciare l'ingresso EN . Nel diagramma temporale qui di seguito è riportata una tipica sequenza di utilizzo:



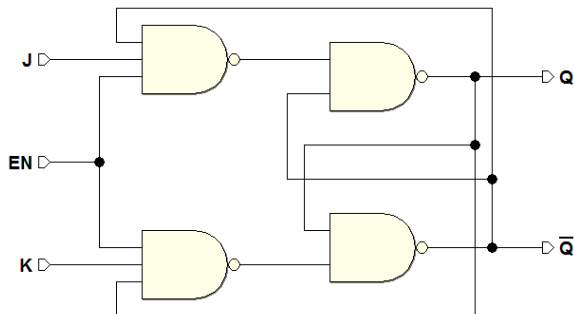
Come negli altri casi, all'inizio della simulazione non conosciamo il valore delle uscite. Dapprima l'ingresso D del dato è impostato a 0 e, durante questo intervallo, viene attivato per un certo tempo l'ingresso EN : il dato presente all'ingresso D è *trasferito* all'uscita Q . Alla disattivazione di EN , il flip-flop *cattura il valore* in quel momento presente all'uscita (si dice che *memorizza l'ultimo valore transitato*).

La stessa sequenza di attivazione è ripetuta subito dopo, ma con l'ingresso del dato impostato a 1: il flip-flop memorizza 1 sulla sua uscita. Nel diagramma è poi dimostrato che, durante l'intervallo di tempo in cui $EN = 0$, l'ingresso D , pur cambiando più volte, non produce cambiamenti alle uscite.

In ultimo, nel diagramma, EN è impostato a 1 per un certo intervallo di tempo: si osserva che l'uscita ripete quando presentato sull'ingresso D (si dice che, quando è abilitato, il flip-flop è *trasparente*).

5.4.3 Flip-flop JK con abilitazione a livello (JK-Latch)

Lo schema in figura mostra una variante del flip-flop JK a comando diretto, al quale è stato aggiunto l'ingresso di abilitazione EN :



Come già discusso nel caso del JK a comando diretto, l'ingresso J è condizionato dall'uscita \overline{Q} , e l'ingresso K dall'uscita Q , rendendo impossibile l'attivazione contemporanea degli ingressi della cella base.

L'ingresso EN condiziona entrambi gli ingressi J e K . Quando EN è attivo, la rete si comporta esattamente come il JK a comando diretto. Altrimenti, mantiene il valore precedentemente memorizzato (la condizione $EN = 0$ equivale ad avere entrambi J e K a 0). La tabella di funzionamento riassume la relazione tra gli ingressi J , K , EN e le uscite Q e \overline{Q} :

Flip-flop JK con abilitazione a livello (JK-Latch)					
EN	J	K	Q	\overline{Q}	
0	—	—	Q_p	\overline{Q}_p	Stato precedente
1	0	0	Q_p	\overline{Q}_p	Stato precedente
1	1	0	1	0	Comando di <i>SET</i>
1	0	1	0	1	Comando di <i>RESET</i>
1	1	1	\overline{Q}_p	Q_p	Inverte le uscite

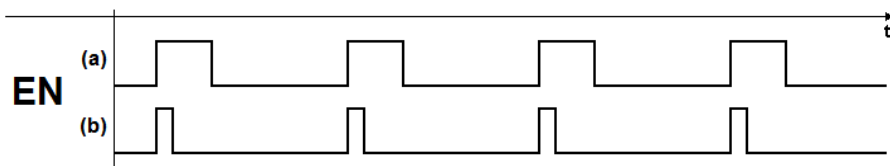
Questo tipo di flip-flop JK *con abilitazione a livello* è poco usato, perchè la configurazione di *Toggle* ($J = 1$ e $K = 1$) può essere concretamente usata soltanto se EN ha una durata inferiore al tempo di propagazione della rete, altrimenti si ricade nel funzionamento descritto per il flip-flop JK a comando diretto esaminato in precedenza. Come si è detto, la funzionalità di *Toggle* è invece utilizzata nei JK abilitati sul fronte.

5.5 Sincronizzazione delle reti sequenziali

Le strutture *abilitate a livello* soddisfano solo parzialmente le esigenze dei moderni sistemi digitali, nei quali solitamente si preferisce che le uscite dei flip-flop che li compongono cambino *periodicamente* e *contemporaneamente*. Un sistema di questo tipo viene chiamato *sincrono*. Per realizzare un sistema sincrono è necessario usare componenti sequenziali più elaborati di quelli finora incontrati ma, soprattutto, è importante comprendere a fondo cosa si vuole veramente intendere per *sincronizzazione* delle reti sequenziali, e quali sono i problemi coinvolti. Prima di introdurre, nei prossimi paragrafi, i flip-flop utilizzati nei sistemi sincroni, in questa sezione si vuole introdurre il concetto di *sincronicità* utilizzando i componenti finora presentati.

5.5.1 Il segnale di sincronizzazione

Consideriamo una rete che utilizza flip-flop con comando a livello. Per soddisfare la specifica della *periodicità*, il comando di abilitazione EN deve assumere una forma ciclica, come visibile nella figura seguente, traccia (a):



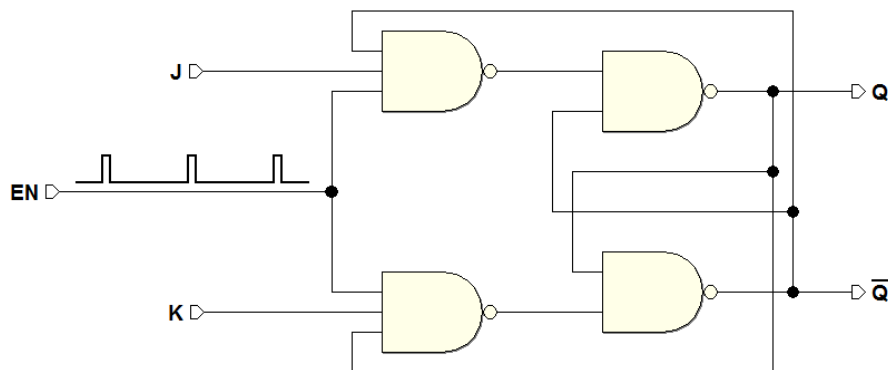
I flip-flop cambieranno le loro uscite soltanto in corrispondenza della *periodica attivazione* della abilitazione EN , che diventa quindi il *segnale di sincronizzazione*, inteso come *riferimento temporale* per l'evoluzione nel tempo della rete. Tuttavia, i flip-flop abilitati a livello garantiscono solo in parte la contemporaneità dei cambiamenti.

Infatti, le loro uscite possono cambiare soltanto quando EN è attivo ma, a causa della durata finita dell'intervallo di attivazione, le transizioni possono ancora avvenire in istanti diversi, dato che gli ingressi possono cambiare per tutto il tempo in cui EN è attivo.

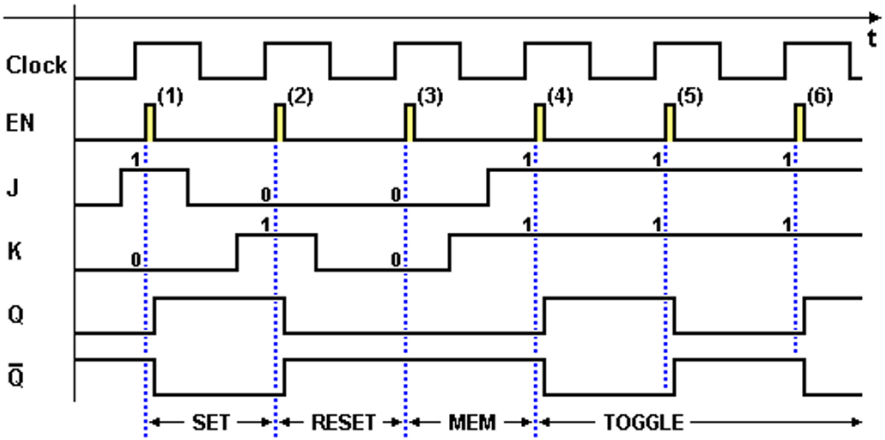
Se si restringe la durata dell'impulso di EN (traccia (b) della figura precedente), si riduce l'intervallo in cui le uscite possono cambiare, avvicinandosi quindi alla specifica della *contemporaneità* dei cambiamenti delle uscite.

5.5.2 Comando impulsivo dei flip-flop abilitati a livello

Consideriamo un flip-flop JK abilitato a livello, identico a quello esaminato nei precedenti paragrafi, e pilotiamo il suo ingresso di abilitazione EN con una sequenza periodica di impulsi:



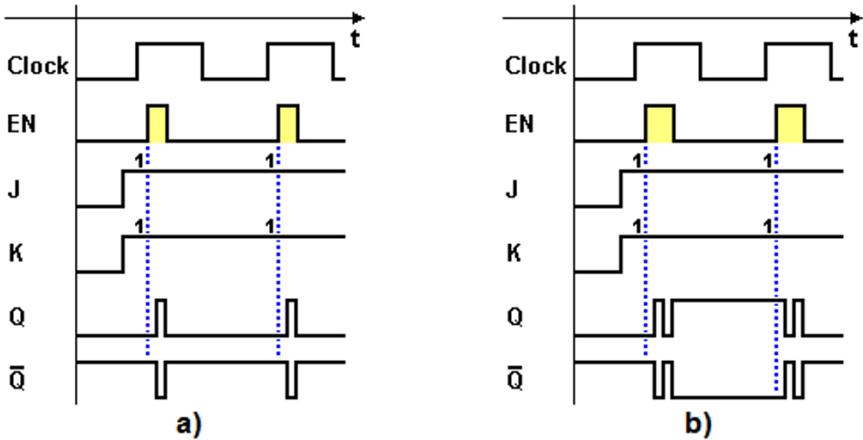
Il flip-flop cambierà le uscite Q e \bar{Q} in corrispondenza degli impulsi di EN , come visibile nel diagramma temporale qui di seguito, che ne descrive un impiego tipico. Si noti che il flip-flop risponde agli ingressi e cambia le sue uscite (dopo il tempo di propagazione della rete), soltanto quando EN è attivato (le linee tratteggiate verticali sono un utile riferimento):







In figura, l'impulso (1) di EN , con $J = 1$ e $K = 0$, porta a 1 l'uscita Q . L'impulso (2), con $J = 0$ e $K = 1$, azzerava Q . Il (3), con $J = 0$ e $K = 0$, mantiene il valore precedente dell'uscita. Dall'impulso (4) in avanti, con $J = 1$ e $K = 1$, si inverte ad ogni ciclo il valore dell'uscita (modalità di *Toggle*).

Si noti che, quando EN è attivo con la combinazione $J = 1$ e $K = 1$, la breve durata dell'abilitazione non permette il verificarsi del fenomeno della continua inversione delle uscite, che caratterizza la realizzazione a comando diretto dello stesso tipo logico. La durata dell'impulso deve essere quindi attentamente valutata in sede di progetto, in relazione ai tempi propri della rete.

Nelle due figure qui sotto si descrivono due situazioni anomale, in cui la durata eccessiva dell'impulso EN è tale da consentire, in questo esempio, a) due o b) tre inversioni delle uscite:



Possiamo riassumere il comportamento logico del flip-flop JK con *abilitazione a livello pilotata impulsivamente*, nella tabella di funzionamento riportata qui di seguito:

Flip-flop JK con abilitazione a livello <i>pilotata impulsivamente</i>					
J	K	EN	Q	\overline{Q}	
0	0		Q_p	$\overline{Q_p}$	Stato precedente
1	0		1	0	Comando di <i>SET</i>
0	1		0	1	Comando di <i>RESET</i>
1	1		$\overline{Q_p}$	Q_p	Inverte le uscite

Il simbolo grafico presente nella colonna relativa ad EN rappresenta l'*attivazione impulsiva*. La tabella si legge tenendo conto che, ad una data combinazione degli ingressi J e K , corrisponderanno le uscite indicate, ma solo dopo avere attivato *impulsivamente* l'ingresso EN .

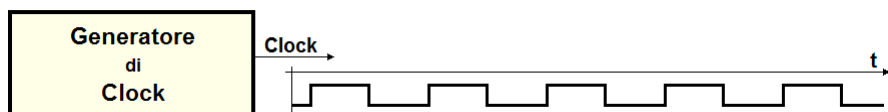
5.5.3 Il “Clock” e il “comando abilitato sul fronte”

Con questo esempio abbiamo visto che i flip-flop abilitati a livello consentono di realizzare un sistema sincrono. Tuttavia, come abbiamo visto, definire la durata dell'impulso presenta delle criticità. Inoltre, nell'esempio precedente si è supposto di mantenere stabili gli ingressi del flip-flop durante l'attivazione di EN , ma non sempre è possibile rispettare questa condizione, indispensabile per garantire l'effettiva *contemporaneità* dei cambiamenti delle uscite.

Per risolvere questi problemi, si è già detto che con i flip-flop abilitati a livello occorrerebbe abbreviare il più possibile la durata dell'impulso di attivazione: tecnicamente, però, non si può scendere al di sotto di un valore minimo. In effetti, per la loro semplicità, i flip-flop con abilitazione a livello trovarono largo impiego nei primi circuiti logici a *componenti discreti*.

Con i circuiti integrati che, a partire dagli anni sessanta del secolo scorso, hanno sostituito i componenti discreti, la situazione si è evoluta. Le tecniche di microelettronica, abbassando di molto il costo della singola funzione logica, hanno consentito lo sviluppo di strutture più affidabili ed economiche, anche se più complesse dal punto di vista circuitale. Nei sistemi digitali attuali si utilizzano prevalentemente flip-flop non più *abilitati a livello* ma *abilitati sul fronte*, ossia da una *transizione di livello* del segnale di sincronizzazione.

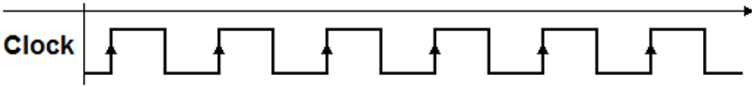
Il segnale di sincronizzazione, nei sistemi digitali, è tradizionalmente denominato *Clock* (“Orologio”). Il Clock è un segnale periodico a due livelli, generato da un apposito circuito, il *Generatore di Clock*:



L'andamento nel tempo di un segnale periodico è chiamato anche *forma d'onda*: nel caso in esempio il segnale è una *onda quadra simmetrica*, con un

rapporto pieno-vuoto (“duty-cycle”) pari al 50%. La frequenza di oscillazione del Clock è scelta dal progettista sulla base delle specifiche del sistema.

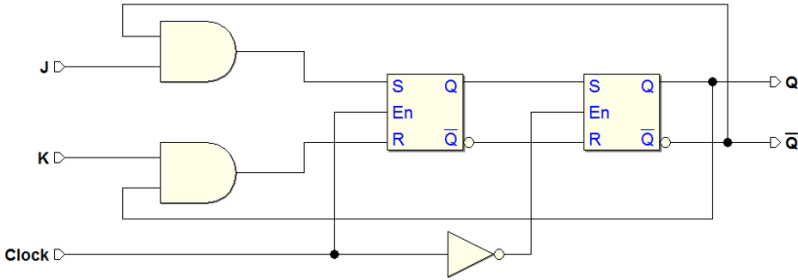
Nella figura qui sotto, le transizioni da 0 a 1 del segnale di Clock sono messe in evidenza con delle frecce verso l’alto. Una transizione di questo tipo è chiamata anche “fronte di salita” (positive edge); quella opposta, invece, “fronte di discesa” (negative edge):



Come vedremo poco più avanti, con questo tipo di dispositivi l’evoluzione temporale del sistema digitale risulterà rigorosamente *sincronizzata* dal fronte prescelto del Clock (esistono componenti attivati dai fronti di salita, e altri attivati da quelli di discesa).

5.5.4 La struttura “master-slave”

La prima struttura di impiego generale, nella quale il cambiamento delle uscite è sincronizzato da un fronte, è stata la struttura *master-slave*; non è più molto usata ma è utile esaminarla prima di passare alle strutture più moderne. La configurazione master-slave impiega due flip-flop RS con abilitazione a livello. Il segnale di Clock controlla l’abilitazione dei due flip-flop e quindi il trasferimento del dato dall’ingresso all’uscita. Qui facciamo riferimento ad una struttura master-slave che realizza un flip-flop di tipo logico JK (quello per il quale è stata sviluppata):

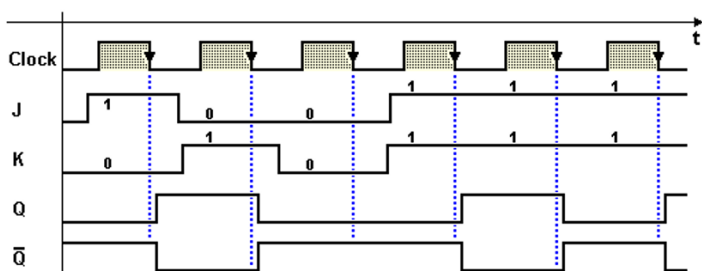


Il flip-flop RS a cui sono applicati i dati in ingresso (*master*) è pilotato direttamente dal *Clock*, mentre l’altro (*slave*) è controllato dal *Clock*. Quando *Clock* = 1, i dati presenti su *J* e *K* pilotano il flip-flop master, ma questo non ha effetto sullo slave, poiché gli ingressi di questo sono disabilitati (*Clock* = 0). Quando il *Clock* ritorna a 0, l’ingresso al master è disabilitato ed i dati sono trasferiti allo slave.

La tabella di funzionamento che descrive il JK master-slave è simile a quella della struttura con l’abilitazione a livello pilotata impulsivamente, ma si noti che i cambiamenti delle uscite avvengono sul fronte di discesa del *Clock*:

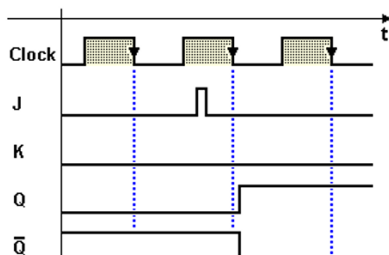
Flip-flop JK con struttura <i>master-slave</i>					
J	K	$Clock$	Q	\overline{Q}	
0	0		Q_p	\overline{Q}_p	Stato precedente
1	0		1	0	Comando di <i>SET</i>
0	1		0	1	Comando di <i>RESET</i>
1	1		\overline{Q}_p	Q_p	Inverte le uscite

Il diagramma temporale qui sotto descrive il comportamento del flip-flop JK master-slave. I *semi-periodi* del *Clock* in cui questo è a 1 sono stati rappresentati tratteggiati, per mettere in evidenza il fatto che un flip-flop master-slave acquisisce gli ingressi per tutta la permanenza a 1 del *Clock*, mentre cambia le sue uscite in corrispondenza dei suoi *fronti di discesa*:



Nella figura, la sequenza di attivazione è uguale a quella proposta per il flip-flop JK con abilitazione impulsiva: vi sono rappresentate tutte le combinazioni possibili di J e K , compresa la modalità di *Toggle* (è stato messo in evidenza, in modo indicativo, il tempo di propagazione della rete).

La struttura master-slave elimina la limitazione sulla durata della abilitazione tipica delle strutture a livello utilizzate impulsivamente. Si noti tuttavia che, per il corretto funzionamento del flip-flop JK master-slave è necessario che gli ingressi J e K siano *stabili* quando il *Clock* è a 1. Infatti, inconveniente di questa struttura è la sensibilità ai cambiamenti di J e K in tale intervallo.



Nel caso illustrato nella figura qui sopra, ad esempio, la presenza indesiderata di un breve impulso su J (se $Q = 0$) è memorizzata nel flip-flop master e provoca poi il cambiamento dell'uscita al fronte di discesa.

Questa struttura si può quindi considerare attivata sul fronte, purché sia garantita la stabilità degli ingressi per il tempo in cui il *Clock* è a 1. Il problema fu risolto con una sua variante, denominata “data lock-out” (*DLO*), che acquisisce i valori di *J* e *K* sul *fronte di salita* del *Clock* e trasferisce i valori in uscita sul *fronte di discesa*, rimanendo insensibile ai cambiamenti che dovessero verificarsi quando il *Clock* è a 1.

I flip-flop con attivazione master-slave e data lock-out sono ormai obsoleti, e sono stati sostituiti da tempo con quelli che lavorano *su di un solo fronte*, esaminati qui di seguito.

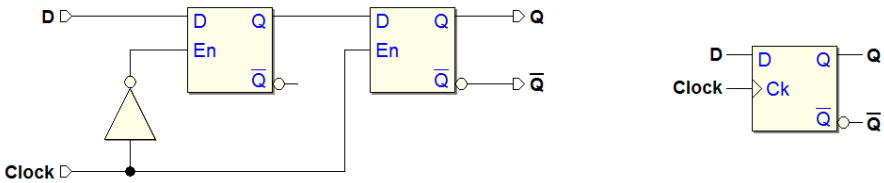
5.6 Flip-flop a comando abilitato sul fronte

Le strutture a *comando abilitato sul fronte* sono di larghissimo impiego nelle realizzazioni attuali. In tale struttura, sia l’acquisizione del dato che il cambiamento delle uscite avvengono in corrispondenza del *fronte* del *Clock*.

Se il fronte attivo è quello di salita, si parla di attivazione PET (*Positive Edge Triggered*); se è quello di discesa, NET (*Negative Edge Triggered*). Il flip-flop rimane insensibile agli ingressi per tutto il resto del tempo: gli ingressi possono quindi cambiare in qualunque momento, escluso un breve intervallo intorno al fronte attivo del clock (esamineremo questo aspetto nel seguito).

5.6.1 Flip-flop D-PET



Nella figura qui sotto, a sinistra, si osserva una rete logica che realizza il flip-flop di tipo D-PET a comando abilitato *sul fronte di salita*. La struttura ricorda quella vista per il JK master-slave, ma usa due flip-flop D a comando abilitato a livello:



Per tutto il tempo in cui il *Clock* vale 0, il primo flip-flop, avendo *En* attivo, segue le variazioni dell’ingresso *D*. Il secondo flip-flop, nello stesso intervallo di tempo, con *En* inattivo, mantiene il valore precedente. Alla transizione a 1 del *Clock*, il primo flip-flop memorizza il valore acquisito, mentre il dato presente sulla sua uscita è trasferito all’uscita del secondo.

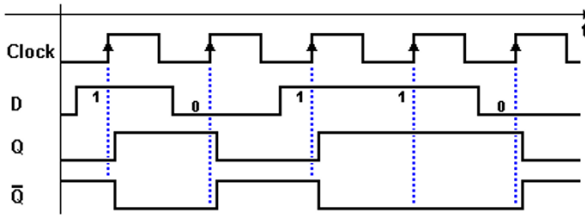
Sulla destra della figura è rappresentato anche il simbolo logico del flip-flop D-PET: il piccolo triangolo sull’ingresso del *Clock* sta ad indicare la sua sensibilità alla transizione.

Qui di seguito la tabella di funzionamento, dove il simbolo riportato nella colonna del *Clock* indica l'evento del *fronte di salita*, al quale sono associati i cambiamenti delle uscite:

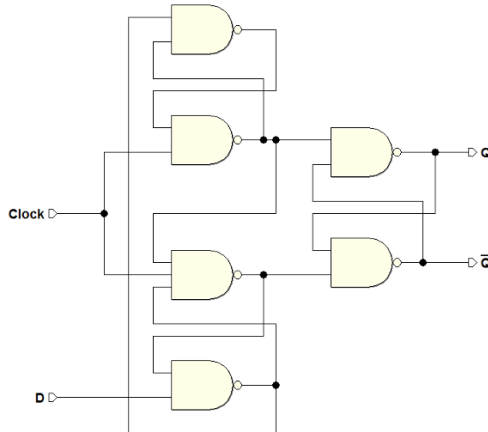
Flip-flop D - PET				
D	$Clock$	Q	\overline{Q}	
0		0	1	Memorizza 0
1		1	0	Memorizza 1
–	0	Q	\overline{Q}_p	Uscite precedenti
–	1	Q	\overline{Q}_p	Uscite precedenti

Le ultime due righe della tabella sono superflue; tuttavia mettono in evidenza il fatto che, in assenza di fronti, il flip-flop mantiene le uscite precedenti (nel seguito queste due righe saranno sempre sottintese).

Nel diagramma temporale qui sotto si vede che l'uscita Q del flip-flop assume, per la durata di un periodo di clock, il valore dell'ingresso D letto ad ogni fronte di salita del *Clock*. Si noti inoltre che Q varia soltanto in corrispondenza dei fronti di salita del *Clock*. Un segnale che mantiene una *relazione temporale* fissa con il *Clock*, è definito *sincrono*.

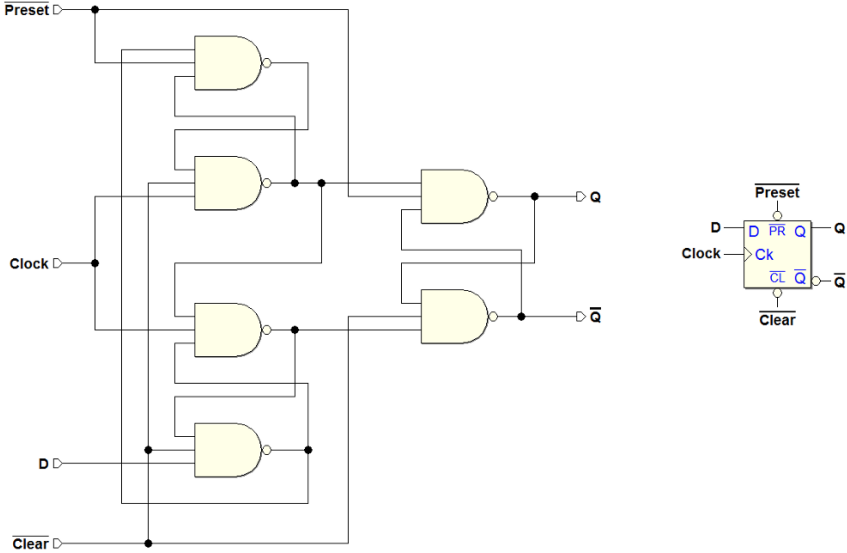


Il circuito del flip-flop D-PET comunemente impiegato in molte realizzazioni commerciali è quello riportato nella figura seguente, più veloce ed economico (usa meno porte logiche). Impiega tre *celle base*:



La tabella di funzionamento ed il diagramma temporale coincidono, ovviamente, con quelli della struttura appena vista.

Nella figura qui sotto è riportato lo schema di un flip-flop D-PET come il precedente, ma dove sono presenti anche le reti di \overline{Clear} e di \overline{Preset} . Sulla destra vediamo il simbolo circuitale corrispondente:



Le prime righe della tabella di funzionamento del flip-flop D-PET con \overline{Clear} e \overline{Preset} , riportata qui sotto, descrivono la funzionalità degli ingressi di inizializzazione. Si noti che, quando uno dei due ingressi \overline{Clear} o \overline{Preset} è attivo, D e $Clock$ non hanno effetto sul componente:

Flip-flop D-PET (con \overline{Clear} e \overline{Preset})						
\overline{Clear}	\overline{Preset}	D	$Clock$	Q	\overline{Q}	
0	1	—	—	0	1	Azione di Clear
1	0	—	—	1	0	Azione di Preset
0	0	—	—	1	1	(non ammessa)
1	1	0	\downarrow	0	1	Memorizza 0
1	1	1	\downarrow	1	0	Memorizza 1

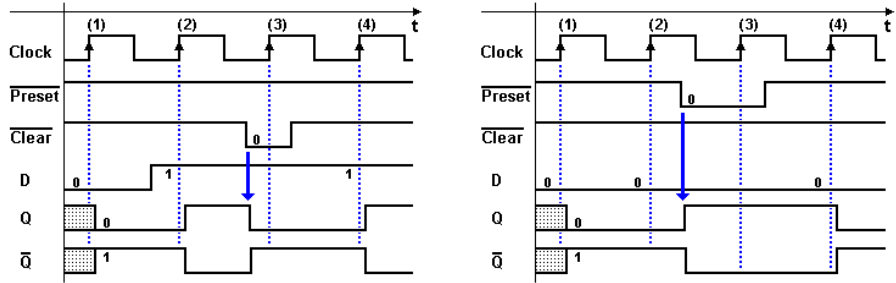
Il comportamento del flip-flop D-PET, con \overline{Clear} e \overline{Preset} entrambi inattivi (le ultime due righe della tabella), corrisponde a quello descritto dalla tabella della versione senza ingressi di inizializzazione.

L'attivazione contemporanea del \overline{Clear} e del \overline{Preset} , per strutture organizzate intorno alla cella base NAND, porta a 1 entrambe le uscite Q e \overline{Q} , creando una situazione analoga a quella descritta a proposito della cella base del

flip-flop SR, con gli stessi problemi. È tuttavia una combinazione di valori facilmente evitabile, perché normalmente sceglieremo di inizializzare il flip-flop utilizzando o il \overline{Clear} , o il \overline{Preset} , collegando l'altro alla costante 1.

Analizziamo ora nel tempo il comportamento nel tempo del flip-flop D-PET, includendo la risposta alla attivazione degli ingressi \overline{Clear} e \overline{Preset} . Per l'analisi, occorre prestare attenzione non solo al livello del segnale di ingresso D all'occorrenza del fronte attivo del $Clock$, ma anche al livello degli ingressi asincroni \overline{Clear} e \overline{Preset} .

Come abbiamo visto, questi agiscono sulla rete in modo *prioritario e indipendente* dal $Clock$. Nella figura seguente sono mostrati due esempi, il primo di attivazione del \overline{Clear} e il secondo del \overline{Preset} :



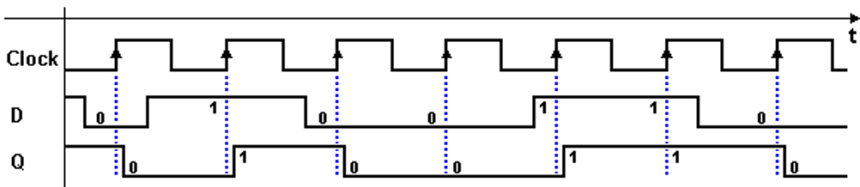
Nell'esempio a sinistra, inizialmente non conosciamo il valore di Q ma, con l'arrivo del fronte (1) del $Clock$, il valore 0 presente su D è memorizzato nel flip-flop, comparso sull'uscita Q . Sul fronte (2) il flip-flop acquisisce regolarmente il valore di D , portando Q a 1.

L'attivazione del \overline{Clear} comporta poi l'azzeramento asincrono dell'uscita e l'insensibilità al fronte (3) del $Clock$. Si noti che la disattivazione del \overline{Clear} non produce cambiamenti sull'uscita.

Nell'esempio sulla destra, invece, è attivato l'ingresso di \overline{Preset} , che forza l'uscita a 1. Si noti, anche qui, l'insensibilità del flip-flop ai fronti di $Clock$ durante l'intervallo di attivazione del \overline{Preset} .

Esempio (D-PET come “sincronizzatore”)

In ultimo, osserviamo l'andamento dell'uscita Q nell'esempio della figura seguente, in relazione all'ingresso D :



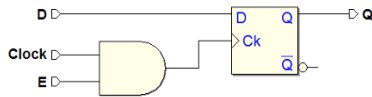
Come si vede, l'ingresso D ha un andamento nel tempo poco regolare rispetto ai fronti del clock. L'operazione di memorizzazione del segnale presente all'ingresso D , ad ogni occorrenza del fronte attivo del clock, produce anche l'effetto di ottenerne in uscita una copia *sincronizzata*, cioè mantenuta in *relazione temporale* fissa rispetto al clock.

Un'applicazione tipica del flip-flop D-PET è proprio la *sincronizzazione dei segnali*: si vedrà in seguito la grande importanza della funzione di sincronizzazione di un segnale, e le problematiche coinvolte.

5.6.2 Flip-flop E-PET

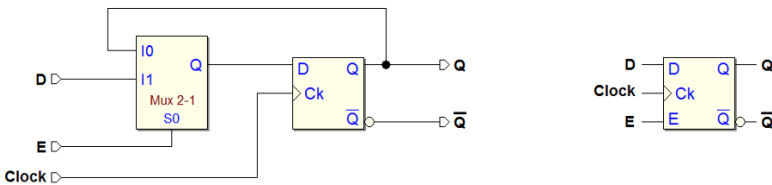
Nelle reti digitali, specialmente se realizzate su dispositivi a larga scala di integrazione, trova impiego un altro tipo di flip-flop, denominato "E-PET": una estensione del tipo D. Il flip-flop E-PET ha la capacità di acquisire il valore dell'ingresso solo quando richiesto, diversamente dal tipo D che acquisisce un nuovo dato ad ogni fronte attivo del clock.

Per ottenere questa funzionalità, saremmo erroneamente tentati di utilizzare un flip-flop D e di condizionarne il clock ad un segnale di abilitazione E , come nella rete seguente:



La porta logica consente di sospendere l'invio del *Clock*, quando l'ingresso E è a 0. Tuttavia, questa tecnica non è utilizzata, perché, a causa del ritardo di propagazione della porta, l'effettivo istante di memorizzazione del flip-flop si sposta nel tempo e vanifica la specificità della contemporaneità delle uscite rispetto agli altri flip-flop presenti nel sistema².

Si ricorre allora ad una struttura diversa, in cui l'ingresso del flip-flop D-PET è pilotato da un multiplexer, come nella figura seguente, a sinistra (sulla destra è riportato il simbolo circuitale del flip-flop E-PET):

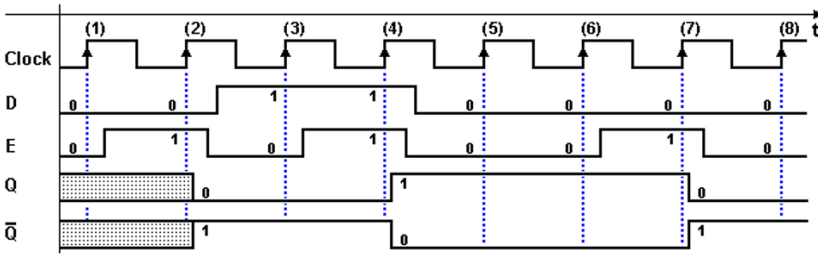


² Nei dispositivi a larga scala di integrazione, che impiegano un elevato numero di flip-flop, i collegamenti fisici del clock sono predisposti dal costruttore con grande attenzione, per non avere disallineamenti di tempo tra i vari elementi della rete, e la presenza di porte logiche lungo il percorso non è ammessa.

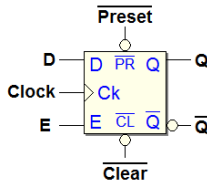
L'ingresso E ("Enable") del multiplexer consente di portare all'ingresso del flip-flop D il valore dell'uscita Q , se $E = 0$, oppure il valore dell'ingresso esterno D , se $E = 1$. All'arrivo del fronte di salita del $Clock$, nel primo caso, il dato memorizzato rimane invariato; nel secondo, l'uscita Q assume il valore dell'ingresso esterno D . Si noti che il $Clock$ non è condizionato. Qui di seguito la tabella di funzionamento del flip-flop E-PET (senza ingressi di inizializzazione):

Flip-flop E-PET					
E	D	$Clock$	Q	\overline{Q}	
0	–		Q_p	\overline{Q}_p	Valore precedente
1	0		0	1	Memorizza 0
1	1		1	0	Memorizza 1

Nel diagramma temporale seguente si osserva l'effetto dell'ingresso E , in corrispondenza dei fronti attivi del $Clock$. I fronti di salita del $Clock$ in cui il flip-flop memorizza il nuovo valore sono i (2), (4) e (7):



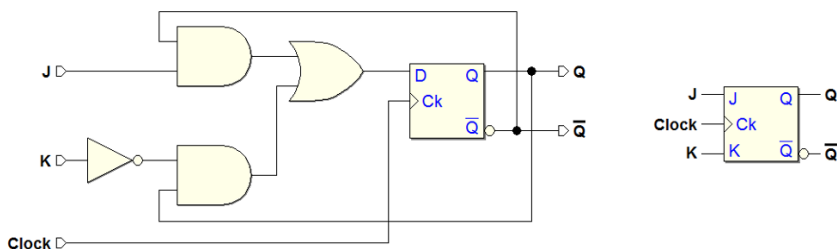
Infine, consideriamo il flip-flop E-PET nella versione completata dagli ingressi di inizializzazione. Qui sotto il simbolo logico e la tabella di funzionamento:



Flip-flop E-PET (con \overline{Clear} e \overline{Preset})							
\overline{Clear}	\overline{Preset}	E	D	$Clock$	Q	\overline{Q}	
0	1	–	–	–	0	1	Azione di Clear
1	0	–	–	–	1	0	Azione di Preset
0	0	–	–	–	1	1	(non ammessa)
1	1	0	–		Q_p	\overline{Q}_p	Valore precedente
1	1	1	0		0	1	Memorizza 0
1	1	1	1		1	0	Memorizza 1

5.6.3 Flip-flop JK-PET

Nella figura seguente, a sinistra, si osserva una rete logica che realizza il flip-flop di tipo JK-PET *a comando abilitato sul fronte di salita*. La struttura è ottenuta da quella del D-PET, in cui l'ingresso D è pilotato da una rete combinatoria AND-OR che ha il compito di calcolarne il valore sulla base degli ingressi J e K e delle uscite Q e \bar{Q} . A destra è rappresentato il simbolo logico corrispondente: anche qui, il triangolo sull'ingresso di $Clock$ indica la sua sensibilità alla transizione:



La tabella di funzionamento che descrive il flip-flop JK-PET è simile a quella delle strutture JK esaminate in precedenza, ma in questo caso i cambiamenti delle uscite avvengono sul fronte di salita del $Clock$:

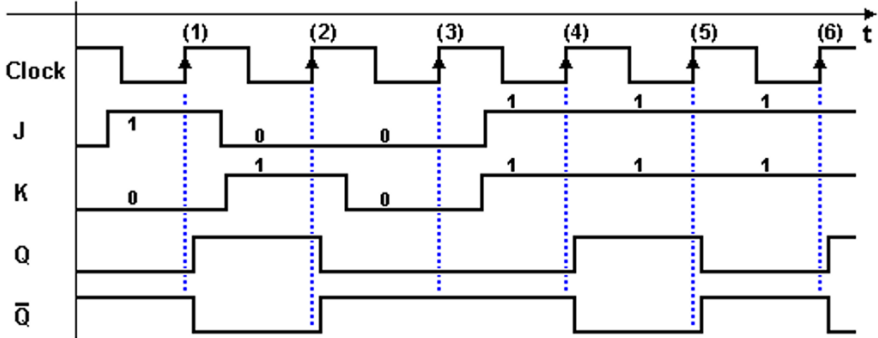
Flip-flop JK-PET					
J	K	$Clock$	Q	\bar{Q}	
0	0		Q_p	\bar{Q}_p	Stato precedente
1	0		1	0	Comando di <i>SET</i>
0	1		0	1	Comando di <i>RESET</i>
1	1		\bar{Q}_p	Q_p	Inverte le uscite

La rete combinatoria AND-OR che genera il valore di D si ricava analizzando la tabella di funzionamento. Da questa analisi si ottiene la tabella riportata qui sotto, da cui è immediato passare alla sintesi della rete:

J	K	Q	D
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

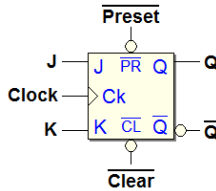
Da cui si ricava: $D = (J\bar{Q} + \bar{K}Q)$

Il diagramma temporale nella figura seguente descrive il comportamento tipico del flip-flop JK-PET. In corrispondenza del fronte di salita del *Clock*, il flip-flop risponde agli ingressi e, dopo il tempo di propagazione della rete, aggiorna le sue uscite. La sequenza di attivazione degli ingressi è analoga a quella già proposta per le versioni precedenti del flip-flop:



Sul fronte (1) *Q* è attivato in conseguenza dell'acquisizione di *J*, sul (2) è disattivato (perché $K = 1$), mentre sul (3) il suo valore è mantenuto (infatti $J = K = 0$). Infine, sui fronti successivi (4,5,6) le uscite invertono il proprio valore (dato che $J = K = 1$).

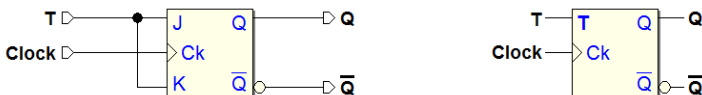
Il flip-flop JK-PET, come gli altri tipi logici, nelle sue realizzazioni commerciali o di libreria, è sovente completato dagli ingressi di *Clear* e *Preset*. Ecco il suo simbolo logico e la tabella di funzionamento:



Flip-flop JK-PET (con \overline{Clear} e \overline{Preset})							
\overline{Clear}	\overline{Preset}	<i>J</i>	<i>K</i>	<i>Clock</i>	<i>Q</i>	\overline{Q}	
0	1	—	—	—	0	1	Azione di Clear
1	0	—	—	—	1	0	Azione di Preset
0	0	—	—	—	1	1	(non ammessa)
1	1	0	0	\downarrow	Q_p	\overline{Q}_p	Valore precedente
1	1	1	0	\downarrow	1	0	Comando di SET
1	1	0	1	\downarrow	0	1	Com.do di RESET
1	1	1	1	\downarrow	\overline{Q}_p	Q_p	Inverte le uscite

5.6.4 Flip-flop T-PET

Se si collegano insieme i due ingressi di un flip-flop JK si ottiene la funzionalità del flip-flop di tipo T (*Toggle*). Nella figura qui sotto, la sua rete basata sul JK-PET ed il corrispondente simbolo logico:



La tabella di funzionamento del T-PET è ricavabile dalla tabella del JK-PET eliminando le righe in cui J e K sono differenti:

Flip-flop T-PET				
T	$Clock$	Q	\overline{Q}	
0		Q_p	\overline{Q}_p	Stato precedente
1		\overline{Q}_p	Q_p	Inverte le uscite

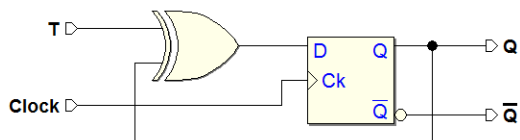
Il flip-flop T di può ottenere anche a partire dal tipo D, con la stessa procedura vista in precedenza per ricavare il JK dal D. In quel caso avevamo ottenuto:

$$D = (J\overline{Q} + \overline{K}Q)$$

dato che $T = J = K$, l'espressione si riduce a:

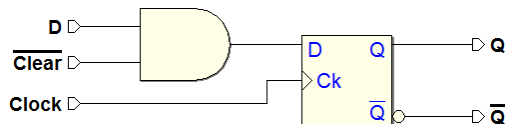
$$D = T\overline{Q} + \overline{T}Q = T \oplus Q$$

Da cui otteniamo la rete qui sotto (che richiameremo più volte nel seguito):



5.6.5 Inizializzazione sincrona dei flip-flop

Nei circuiti moderni di una certa complessità è anche usata una struttura di *inizializzazione sincrona*, nella quale le azioni di \overline{Clear} e \overline{Preset} sono sincronizzate dal clock. Ad esempio, nella figura seguente è rappresentata una possibile realizzazione di \overline{Clear} sincrono, per un flip-flop di tipo D-PET:



L'ingresso \overline{Clear} , quando attivato (a 0), prevale sull'ingresso D e produce l'azzeramento dell'uscita, in corrispondenza del fronte attivo del $Clock$.

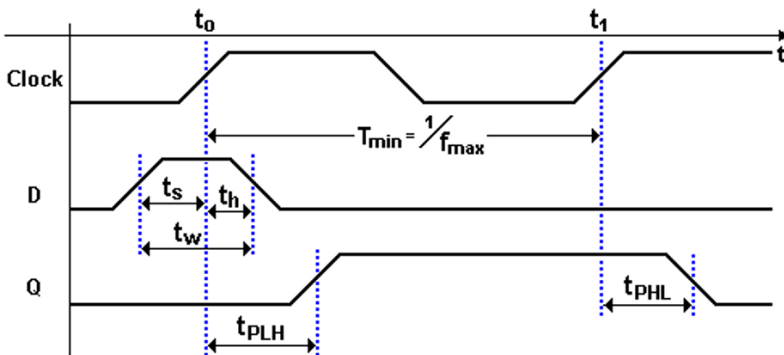
5.7 Tempi caratteristici dei flip-flop

I flip-flop, come i componenti combinatori di cui sono costituiti, sono soggetti a *ritardi di propagazione*. Come già visto per le reti combinatorie, i tempi di propagazione sono misurati prendendo come riferimento il 50% dei fronti dei segnali. I tempi di propagazione sono rilevati dal costruttore su base statistica, e ne vengono dichiarati i valori *minimi*, *tipici* e *massimi*.

Oltre ai tempi di propagazione, esistono altri tempi caratteristici, rilevati anch'essi su base statistica, che quantificano dei *margini di sicurezza* che devono essere rispettati per garantire il corretto funzionamento del flip-flop. La tabella qui sotto fornisce una sintetica definizione dei principali tempi caratteristici:

t_{PLH}	Tempo di propagazione, misurato dall'attivazione del <i>Clock</i> alla transizione dell'uscita dal basso verso l'alto (L-H)
t_{PHL}	Tempo di propagazione misurato dall'attivazione del <i>Clock</i> alla transizione dell'uscita dall'alto verso il basso (H-L)
t_s	Tempo di predisposizione (<i>Setup Time</i>): l'intervallo in cui il valore di un ingresso sincrono deve essere stabile <i>prima</i> del fronte attivo del <i>Clock</i> affinché il suo valore venga acquisito correttamente
t_h	Tempo di mantenimento (<i>hold time</i>): l'intervallo in cui il valore di un ingresso sincrono deve essere stabile <i>dopo</i> il fronte attivo del <i>Clock</i> affinché il suo valore venga acquisito correttamente
t_w	Durata minima (<i>width</i>) di un segnale di ingresso
T_{min}	Minimo periodo del <i>Clock</i>
F_{max}	Massima frequenza di <i>Clock</i>

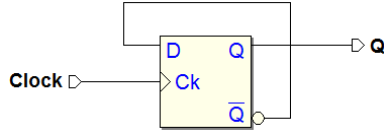
Il diagramma temporale nelle figura qui sotto mostra i tempi caratteristici di un flip-flop D-PET, in una sequenza di acquisizione di un 1 seguito da uno 0. Il t_w esemplificato in figura si riferisce all'ingresso *D* e, in questo esempio, è dato dalla somma dei tempi di t_s e di t_h :



Se i *margini di sicurezza* dati dai tempi dichiarati dal costruttore del flip-flop non sono rispettati, si possono verificare comportamenti non prevedibili, compresa la generazione di segnali anomali. Tale fenomeno prende il nome di *metastabilità* e sarà esaminato nel prossimo capitolo.

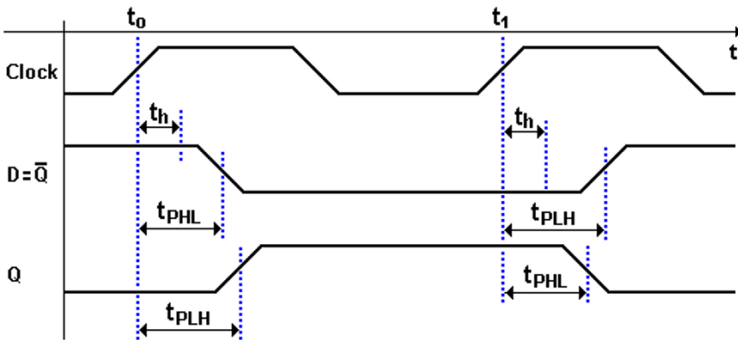
5.7.1 Relazione tra i tempi di propagazione e di mantenimento

Utilizziamo l'esempio riportato nella figura qui sotto per ragionare sulla relazione che esiste tra i tempi di propagazione e di mantenimento:



Il circuito è composto da un flip-flop D-PET con l'ingresso D pilotato dalla sua uscita negata \bar{Q} . Ad ogni fronte attivo del $Clock$ l'uscita invertirà il suo valore, come avverrebbe in un flip-flop tipo T-PET con l'ingresso $T = 1$.

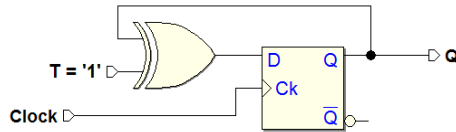
Affinché il valore dell'ingresso D sia acquisito correttamente, deve rimanere stabile per un tempo maggiore di t_h dopo il fronte di salita del $Clock$:



La situazione è esemplificata nel diagramma temporale qui sopra, nel quale si osserva che il t_h deve essere più breve di ciascuno dei due tempi di propagazione (t_{PHL} e t_{PLH}): tutte i flip-flop soddisfano questa specifica.

5.7.2 Massima frequenza del clock di una rete con flip-flop

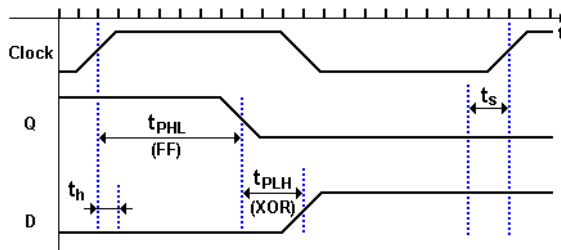
I tempi caratteristici dei flip-flop entrano in gioco nel determinare la *massima frequenza* del $Clock$ della rete che lo impiega. In precedenza abbiamo visto il flip-flop T-PET nella sua realizzazione a partire da un D-PET. Nella rete mostrata nella figura seguente, ne impostiamo l'ingresso T sempre a 1, imponendo la configurazione in cui l'uscita *cambia valore* ad ogni fronte di salita del $Clock$. Lo scopo di questo esempio è di avere, lungo il percorso di retroazione, una semplice rete combinatoria, con il suo ritardo di propagazione:



L'ingresso D è pilotato dall'uscita Q , tramite la porta EXOR, la quale, avendo un ingresso ad 1, funziona come una porta NOT. Per un corretto funzionamento della rete, il segnale presente su D deve essere stabile almeno da un tempo t_s , nell'istante in cui il $Clock$ presenta il fronte di salita, e deve rimanere tale per un tempo t_h dopo la transizione.

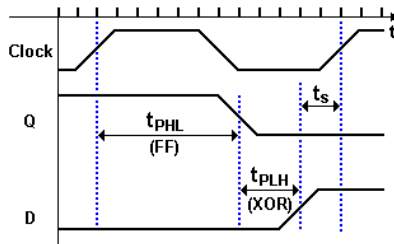
Si osservi che la scala dei tempi, nei seguenti tre diagrammi temporali, corrisponde ad **1 nS** (tra due tacche sull'asse dei tempi); si sono assunti i valori: **$t_s = 2 \text{ nS}$** , **$t_h = 1 \text{ nS}$** , **$t_{PHL}(FF) = 7 \text{ nS}$** , **$t_{PLH}(XOR) = 3 \text{ nS}$** .

Le tre situazioni hanno in comune il fatto che il segnale D è stabile dopo un tempo $t_{PHL}(FF) + t_{PLH}(XOR)$, indipendentemente dalla durata del periodo del $Clock$. Nel primo diagramma, qui sotto, la sequenza è caratterizzata da un $Clock$ con periodo $T = 20 \text{ nS}$ (corrispondente alla frequenza di **50 MHz**):



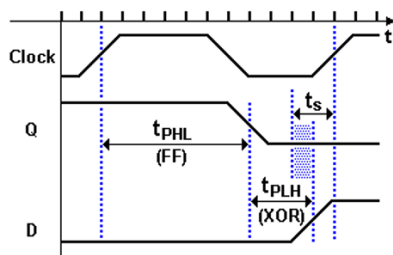
Il successivo fronte di salita campiona un dato D stabile da un tempo maggiore di t_s : la rete funziona quindi correttamente. Inoltre, la condizione su t_h è rispettata, poiché t_h è minore del tempo di propagazione del flip-flop (nelle successive figure la rappresentazione di t_h è omessa).

Nella figura qui sotto, con un periodo di $Clock$ di **12 nS**, si osserva la situazione limite, nella quale D è stabile da un tempo t_s prima del fronte. Il funzionamento è ancora corretto:



Il periodo di **12 nS** corrisponde ad una frequenza di circa **83 MHz**, la *frequenza massima* del $Clock$ della rete.

Invece, nella figura seguente, con un *Clock* di **11 nS** di periodo (corrispondente a circa **91 MHz**), il successivo fronte di *Clock* si verifica quando *D* è già stabile, ma non è rispettato il t_s . In questo caso, il funzionamento non è garantito: siamo quindi oltre la *frequenza massima* di lavoro della rete.



5.8 Flip-flop: simboli grafici e tabelle

All'inizio della trattazione abbiamo distinto i flip-flop per *tipo logico* e per *tipo di comando*. Questa sezione presenta in modo riassuntivo le caratteristiche e i simboli grafici solo dei flip-flop di impiego più comune.

5.8.1 Tipi logici

Le *tabelle di funzionamento* seguenti riepilogano il comportamento dei *tipi logici* dei flip-flop presentati in questo capitolo (SR, JK, D, E e T). Poiché non descrivono le modalità temporali del funzionamento, riportano soltanto gli *ingressi logici* e non quelli di abilitazione o di clock, e neppure quelli di inizializzazione. In queste tabelle, tutti gli ingressi sono *attivi alti*.

SR			
<i>SET</i>	<i>RESET</i>	<i>Q</i>	
0	0	Q_p	Stato precedente
1	0	1	Comando di <i>SET</i>
0	1	0	Comando di <i>RESET</i>
1	1	1	Non ammesso

JK			
<i>J</i>	<i>K</i>	<i>Q</i>	
0	0	Q_p	Stato precedente
1	0	1	Comando di <i>SET</i>
0	1	0	Comando di <i>RESET</i>
1	1	$\overline{Q_p}$	Inverte (<i>Toggle</i>)

D		
D	Q	
0	0	Memorizza 0
1	1	Memorizza 1

E			
E	D	Q	
0	–	Q_p	Valore precedente
1	0	0	Memorizza 0
1	1	1	Memorizza 1

T		
T	Q	
0	Q_p	Stato precedente
1	$\overline{Q_p}$	Inverte (<i>Toggle</i>)

5.8.2 Tipi di comando

Riepiloghiamo qui le modalità temporali con cui i flip-flop acquisiscono gli ingressi e aggiornano le uscite:

- **Comando diretto:**

Il comando è *diretto* quando l'azione degli *ingressi logici* non è subordinata a nessun altro ingresso, controllando direttamente il comportamento del flip-flop, che in questo caso è chiamato *asincrono*.

- **Comando abilitato a livello:**

Definiamo il comando come *abilitato a livello* quando è presente un ingresso aggiuntivo avente la funzione di abilitare/disabilitare gli ingressi in funzione del suo livello logico. Si parla anche di *comando di tipo Latch*.

- **Comando abilitato sul fronte:**

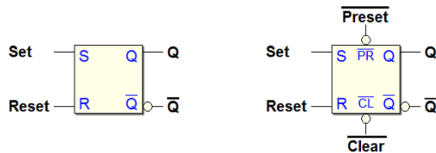
Si ha il comando *abilitato sul fronte* quando è presente un ingresso aggiuntivo di sincronizzazione, usualmente denominato *Clock*, che ha il compito di abilitare la funzione degli ingressi in corrispondenza dei suoi fronti di salita e/o di discesa. Questi flip-flop sono chiamati *sincroni*.

In questa sintesi vengono riportati soltanto i flip-flop di uso più comune.

Comando diretto

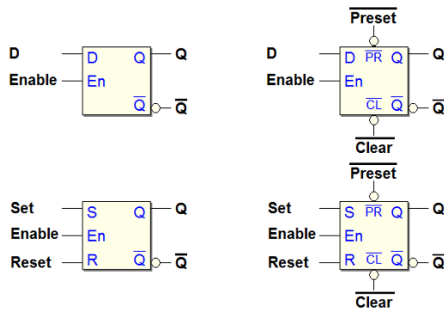
La struttura a comando diretto è correntemente impiegata soltanto per il tipo logico *Set-Reset*. A sinistra è rappresentato il simbolo senza ingressi di inizializzazione; sulla destra, invece, quello che include il *Clear* ed il *Preset*.

La funzione di questi ultimi può sembrare superflua, visto che la loro funzione è identica a quella degli ingressi logici, ma in un progetto è spesso utile separare la rete di inizializzazione dal resto:



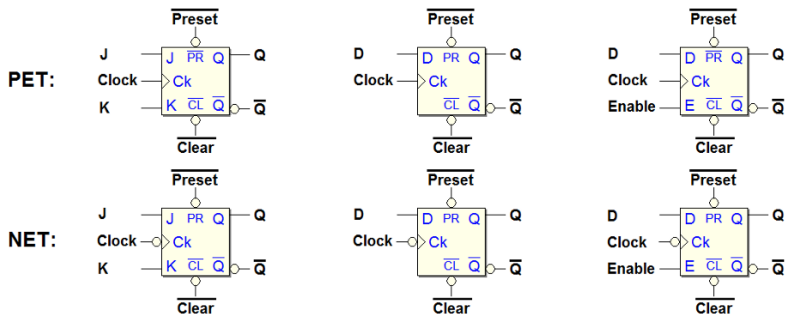
Comando abilitato a livello

I flip-flop a comando abilitato a livello sono realizzati principalmente nel tipo logico D (*D-Latch*), e trovano impiego anche in strutture multiple (come le *memorie di lettura/scrittura*). Il Set-Reset abilitato a livello è spesso invece un blocco interno di flip-flop più complessi. Valgono le stesse considerazioni di prima circa gli ingressi di inizializzazione:



Comando abilitato sul fronte

La struttura a comando abilitato sul fronte (*edge-triggered*, PET o NET) è di gran lunga la più utilizzata, soprattutto nei tipi logici D ed E. Quest'ultimo si va diffondendo nelle reti complesse, mentre l'impiego del JK è in declino. La figura qui sotto riporta i simboli di flip-flop con abilitazione sia PET che NET, di vari tipi logici. In questa categoria sono generalmente presenti gli ingressi di inizializzazione *Clear* ed il *Preset* (talvolta il solo *Clear*):



5.8.3 Tabelle di eccitazione

Un altro modo di descrivere il comportamento dei flip-flop è rappresentato dalle “*tabelle di eccitazione*”. Le tabelle di funzionamento, incontrate in precedenza, indicano il valore assunto dall’uscita in funzione degli ingressi. La tabella di eccitazione, invece, fornisce i valori da assegnare agli ingressi logici del flip-flop in funzione della *transizione dell’uscita* che si vuole ottenere.

Qui sotto riportiamo, sulla sinistra, la tabella di funzionamento (già vista) del flip-flop *Set-Reset*. Sulla destra la corrispondente tabella di eccitazione, nella quale, per ognuna delle 4 possibili transizioni dell’uscita, è riportata la configurazione degli ingressi necessaria per ottenerla:

Set-Reset			
Tabella di funzionamento			
<i>Set</i>	<i>Reset</i>	<i>Q</i>	
0	0	Q_p	Stato precedente
1	0	1	Comando di SET
0	1	0	Comando di RESET
1	1	1	Non ammesso

Set-Reset		
Tabella di eccitazione		
$Q_p \rightarrow Q$	<i>Set</i>	<i>Reset</i>
$0 \rightarrow 0$	0	–
$0 \rightarrow 1$	1	0
$1 \rightarrow 0$	0	1
$1 \rightarrow 1$	–	0

La tabella di eccitazione si ricava facilmente da quella di funzionamento. Ad esempio, la transizione dell’uscita Q da 0 a 0 (si scrive: $0 \rightarrow 0$) può essere ottenuta mantenendo lo stato precedente del flip-flop ($Set = 0$, $Reset = 0$, prima riga della tabella di funzionamento), oppure con un comando di Reset ($Set = 0$, $Reset = 1$, terza riga).

Ciò si traduce nella condizione espressa dalla prima riga della tabella di eccitazione: *Set* deve essere a 0, mentre il valore di *Reset* è indifferente. Situazione analoga per la transizione da $1 \rightarrow 1$, mentre i pilotaggi delle altre due transizioni non presentano indifferenze.

Le tabelle di funzionamento e di eccitazione del flip-flop JK sono riportate qui sotto. Quella di eccitazione presenta una indifferenza per ciascuna delle quattro transizioni. Ad esempio, la transizione da $0 \rightarrow 1$ si ottiene o con un comando di Set ($J = 1$, $K = 0$), o con uno di *Toggle* ($J = 1$, $K = 1$). In sintesi, il valore di K è indifferente, mentre J deve essere impostato ad 1:

JK			
Tabella di funzionamento			
<i>J</i>	<i>K</i>	<i>Q</i>	
0	0	Q_p	Stato precedente
1	0	1	Comando di SET
0	1	0	Comando di RESET
1	1	$\overline{Q_p}$	Inverte (<i>Toggle</i>)

JK		
Tabella di eccitazione		
$Q_p \rightarrow Q$	<i>J</i>	<i>K</i>
$0 \rightarrow 0$	0	–
$0 \rightarrow 1$	1	–
$1 \rightarrow 0$	–	1
$1 \rightarrow 1$	–	0

Per completezza, riportiamo qui sotto le tabelle per il flip-flop D, anche se, in questo caso, la tabella di eccitazione è di derivazione immediata. La tabella del flip-flop E, quando è abilitato dall'ingresso E , è identica.

D		
Tabella di funzionamento		
D	Q	
0	0	Memorizza 0
1	1	Memorizza 1

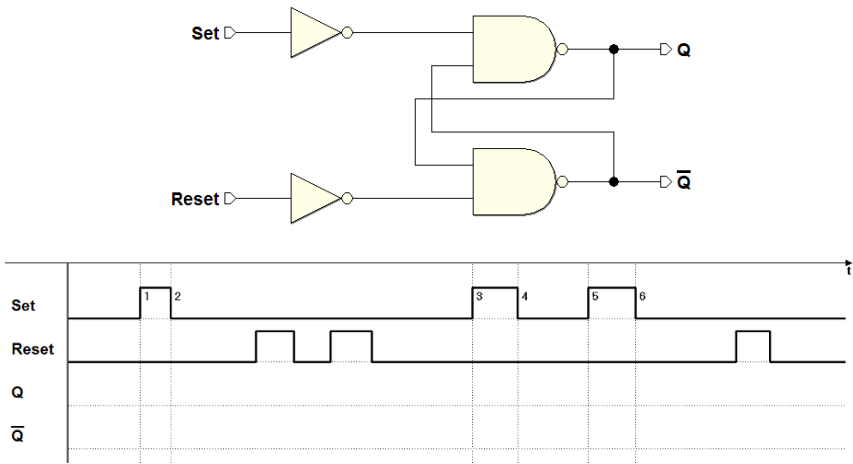
D	
Tabella di eccitazione	
$Q_p \rightarrow Q$	D
0 \rightarrow 0	0
0 \rightarrow 1	1
1 \rightarrow 0	0
1 \rightarrow 1	1

5.9 Esercizi

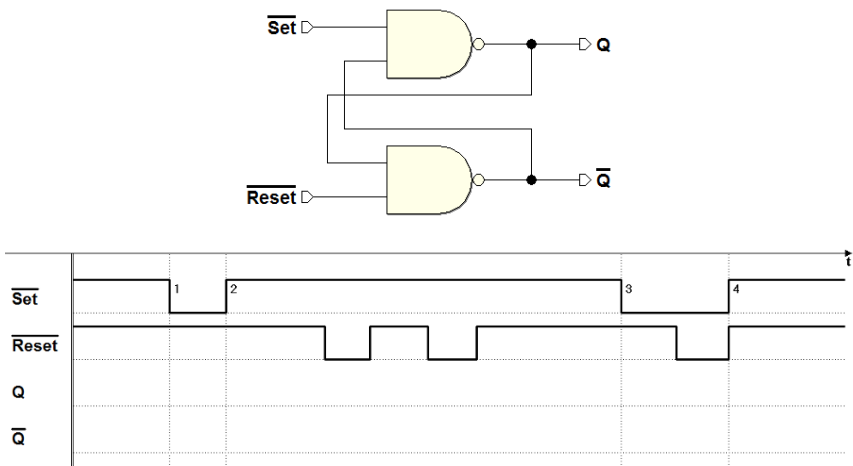
Completare, per ciascuna rete, il tracciato temporale proposto. Abbiamo reso disponibili le tracce qui proposte anche sul sito del simulatore, in formato *PDF*, nelle pagine dei *contenuti digitali* del libro.

Si suggerisce di completare sulla carta i tracciati *senza* l'ausilio di *Deeds*, che può essere successivamente utilizzato per verificare la propria soluzione (sul sito sono disponibili anche i file delle reti proposte).

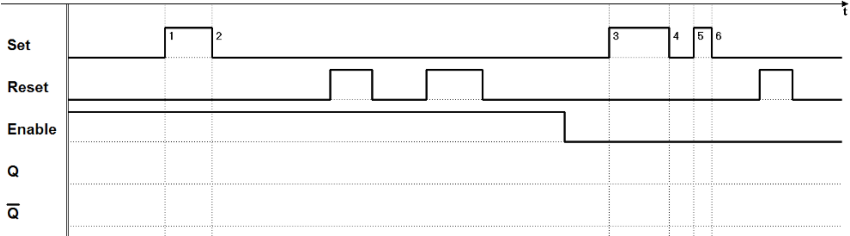
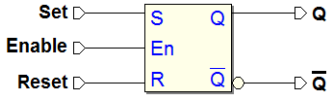
1. Flip-flop *Set-Reset* a comando diretto:



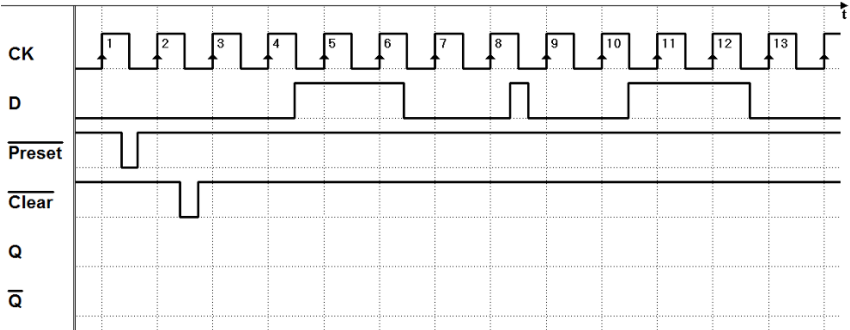
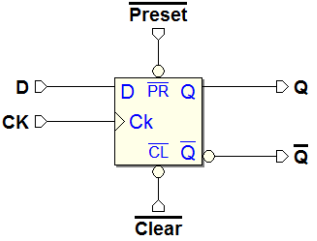
2. Flip-flop *Set-Reset* (cella base 'nand'):



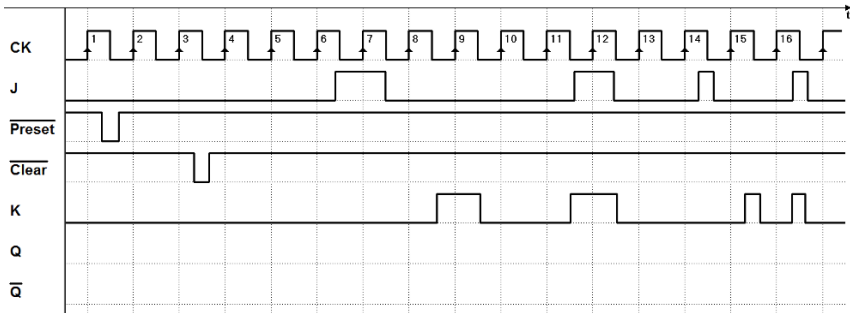
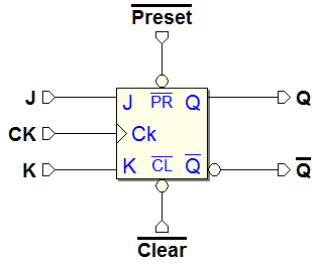
3. Flip-flop *Set-Reset* (con Enable):



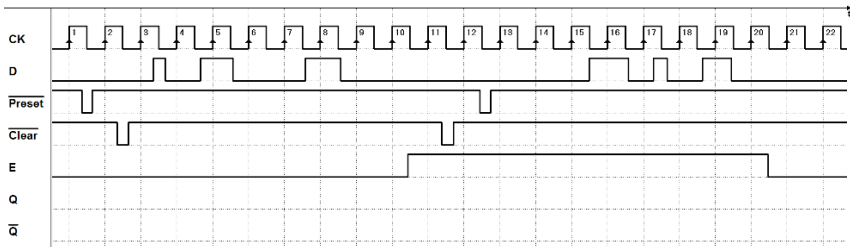
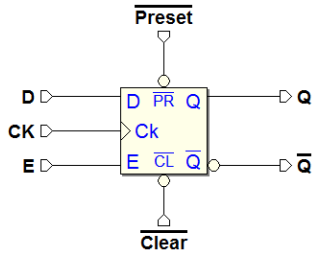
4. Flip-flop D - PET (con Preset e Clear):



5. Flip-flop JK-PET (con Preset e Clear):



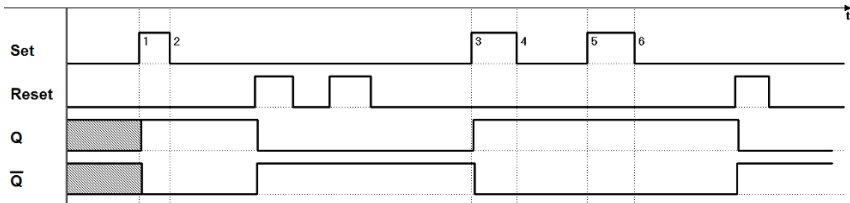
6. Flip-flop E-PET (con Preset e Clear):



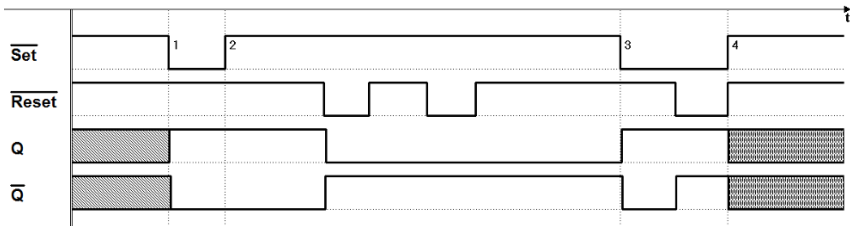
5.10 Soluzioni

I tracciati qui riportati sono stati ottenuti tramite la simulazione temporale di *Deeds*. Sono disponibili, sul sito del simulatore, i file delle reti assegnate, in modo tale che le soluzioni possano essere verificate mediante simulazione.

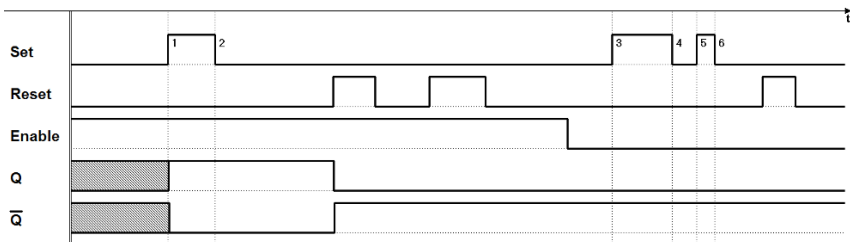
1. Flip-flop *Set-Reset* a comando diretto:



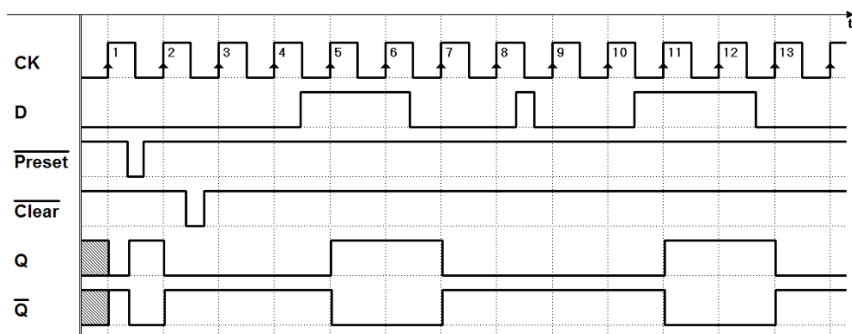
2. Flip-flop *Set-Reset* (cella base 'nand'):



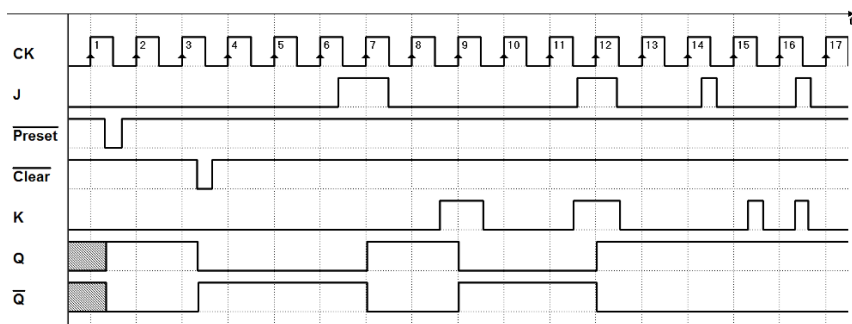
3. Flip-flop *Set-Reset* (con Enable):



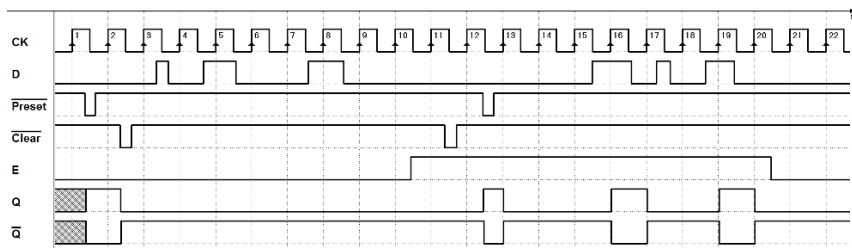
4. Flip-flop D - PET (con Preset e Clear):



5. Flip-flop JK-PET (con Preset e Clear):



6. Flip-flop E-PET (con Preset e Clear):



Reti sincrone di flip-flop

Nel presente capitolo passeremo in rassegna le reti di flip-flop più usate, e che a loro volta possono essere utilizzate come *blocchi funzionali standard* per la realizzazione di reti digitali più complesse. Affronteremo l'analisi di queste reti *in un modo intuitivo*, mentre nei prossimi capitoli si procederà alla loro progettazione con metodi sistematici. Rappresenteremo le reti mediante i consueti schemi logici, cioè sotto forma di un insieme di componenti, come porte logiche e flip-flop, con le relative interconnessioni.

Nel precedente capitolo sono state esaminate le reti sequenziali più semplici, generalmente formate da reti combinatorie retro-azionate, con lo scopo di comprendere le strutture ed il funzionamento dei vari tipi di celle elementari di memoria, i flip-flop. Sul piano teorico, una rete sequenziale di complessità più elevata potrebbe essere progettata in modo analogo, sulla base di reti combinatorie retro-azionate. Tale approccio, però, presenterebbe numerosi problemi, specie sul piano della progettazione e della testabilità.

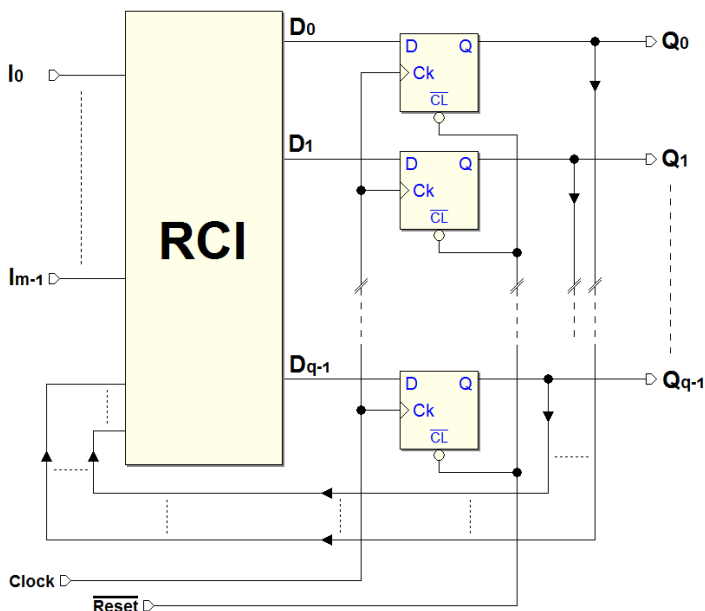
E' preferibile progettare una rete sequenziale complessa in modo più strutturato, utilizzando flip-flop come elementi base, interconnessi da *reti puramente combinatorie*. In questo modo si riesce a dividere nettamente la funzione di memorizzazione, caratteristica tipica di una rete sequenziale (affidata ai flip-flop), da quella che determina il suo comportamento logico e la sua evoluzione nel tempo (affidata alle reti combinatorie).

Come si è visto nel capitolo precedente, l'uscita di una rete sequenziale è funzione non soltanto degli ingressi in quel momento, ma anche dei valori assunti da questi in precedenza.

In una rete sequenziale costituita di flip-flop e reti combinatorie, la storia precedente della rete lascia traccia soltanto nei valori assunti dai flip-flop, dal momento che le reti combinatorie, in quanto tali, non hanno capacità di memoria. L'insieme dei *valori memorizzati dai flip-flop* viene chiamato *stato* della rete. Le uscite della rete sequenziale saranno quindi una funzione degli ingressi e dello stato.

Costituisce una *rete sincrona di flip-flop* la rete in cui questi condividono lo stesso *clock*. Gli ingressi asincroni dei flip-flop, quando presenti, sono usati *soltanto per la loro inizializzazione*. Se queste due condizioni non sono soddisfatte, la rete rientra nell'ampia casistica delle *reti asincrone*.

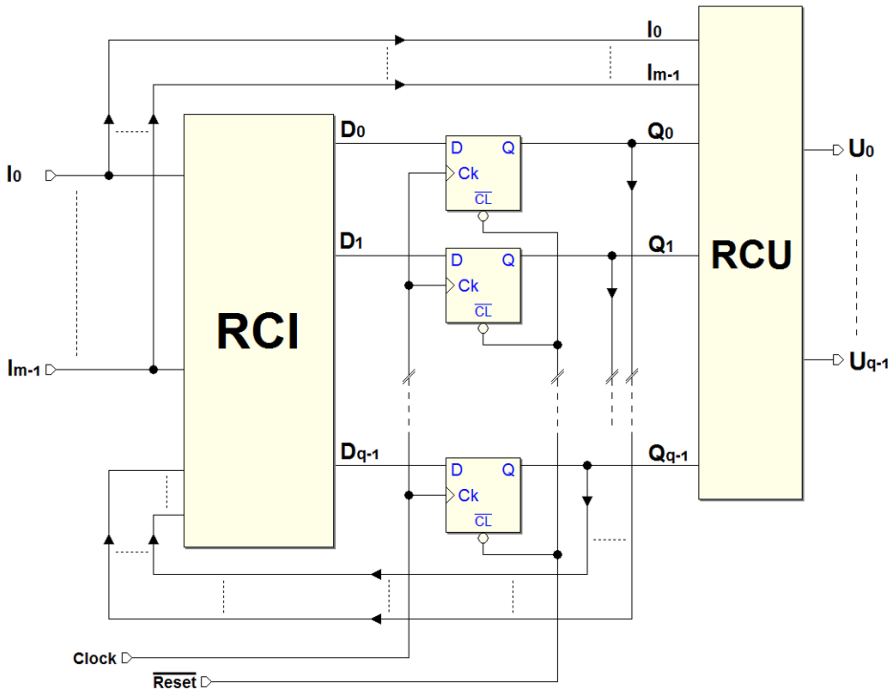
Le reti sincrone presentano, per la loro regolarità, notevoli vantaggi dal punto di vista della progettazione e della testabilità. La progettazione delle reti di struttura più semplice può essere inizialmente affrontata senza ricorrere a procedure formali di sintesi. Nelle reti esaminate in questo capitolo sono utilizzati i flip-flop di tipo logico D, E e JK, con abilitazione sul fronte (PET), studiati nel capitolo precedente.



La figura rappresenta lo schema di principio di una rete sequenziale sincrona di flip-flop, in versione semplificata. I flip-flop hanno in comune il segnale di *Clock*: questo garantisce una delle proprietà delle reti sequenziali sincrone, cioè il cambiamento contemporaneo delle uscite dei flip-flop (entro i limiti delle tolleranze dei componenti reali).

Gli ingressi *D* dei flip-flop sono generati dalla *rete combinatoria degli ingressi RCI* che elabora gli ingressi provenienti dall'esterno ($I_0..I_{m-1}$) e le uscite ($Q_0..Q_{q-1}$) dei flip-flop stessi. Ad ogni fronte attivo del *Clock*, i valori delle uscite $Q_0..Q_{q-1}$ sono sostituiti da quelli forniti dalla RCI. Si capisce quindi come la rete combinatoria determini il *comportamento* della rete sequenziale, sulla base dei *ingressi esterni* e dei *valori memorizzati* nei flip-flop (lo stato della rete). Per questa ragione, la RCI viene anche chiamata *rete dello stato successivo*. Il \overline{Reset} , attivo basso, consente di inizializzare tutti i flip-flop.

La struttura generale di una rete sincrona è quella della figura seguente:



Si differenzia dalla precedente struttura semplificata per la presenza della *rete combinatoria delle uscite RCU* che consente una maggiore flessibilità nella generazione delle uscite. In questo modo, ciascuna delle uscite ($U_0..U_{p-1}$) può essere generata come funzione combinatoria dello stato memorizzato nei flip-flop ($Q_0..Q_{q-1}$) e degli ingressi provenienti dall'esterno ($I_0..I_{m-1}$).

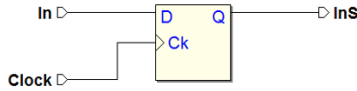
6.1 Segnali sincroni e asincroni

In una rete sincrona, come abbiamo visto, i flip-flop condividono lo stesso clock. Le loro uscite, quindi, cambiano in corrispondenza dei fronti del clock, ossia sono *sincrone* con il clock. Tutte le *uscite della rete*, e i segnali *interni* alla stessa, ottenuti tramite reti combinatori dalle uscite dei flip-flop, conservano, sia pure con l'aggiunta di ritardi, una relazione temporale con il clock.

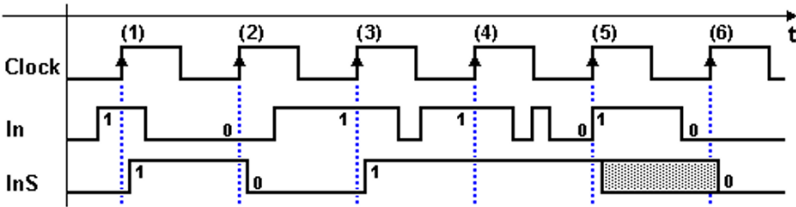
I *segnali di ingresso*, invece, in linea di principio, non hanno alcuna relazione con il clock della rete, essendo generati da sistemi esterni. In generale, un segnale di ingresso è quindi *asincrono*, a meno che non provenga da un'altra *rete sincrona* che utilizzi lo stesso clock. In generale, prima di utilizzare un segnale asincrono, è conveniente *sincronizzarlo* mediante una opportuna rete di sincronizzazione.

6.1.1 Sincronizzatore

Consideriamo ora la rete seguente, costituita da un semplice flip-flop di tipo D-PET, e che riceve in ingresso un segnale *asincrono*:



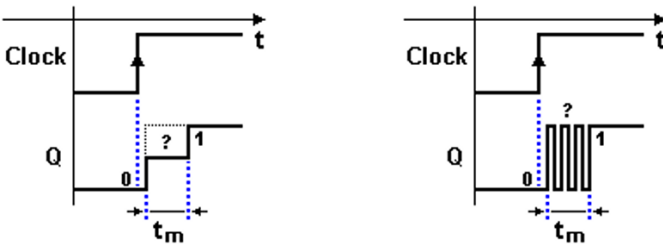
Si vuole esaminare il suo funzionamento come *sincronizzatore* del segnale di ingresso: l'uscita *InS* rappresenta infatti la versione *sincronizzata* del segnale applicato all'ingresso *In*. Nel diagramma temporale qui sotto è rappresentato l'andamento dell'ingresso. Si faccia attenzione che le sue caratteristiche di *asincronicità* sono state messe in particolare evidenza:



Sui fronti di salita del *Clock*, dal primo al numero (4), l'uscita *InS* assume il valore dell'ingresso *In* e lo mantiene sino al successivo fronte attivo. Non sono rilevate dal flip-flop, invece, le eventuali *variazioni intermedie* dell'ingresso, come si osserva negli intervalli di tempo tra i fronti (3) e (5).

Sui fronti dal primo al (4) si suppongono rispettati i *tempi di setup* (t_s) e di *hold* (t_h) del flip-flop. Questo non è vero nel caso del fronte (5), dove l'ingresso presenta una transizione verso l'alto *contemporanea* a quella del *Clock*. Per questo motivo, non possiamo prevedere a priori l'uscita del flip-flop, che è stata rappresentata in questa figura come *indeterminata*.

Un circuito reale, in queste condizioni, ha una *piccola probabilità* di generare, per un breve intervallo di tempo, dei livelli logici non validi o delle oscillazioni: questo fenomeno è chiamato "*metastabilità*":



A causa dei ritardi interni del flip-flop, si possono occasionalmente verificare uscite *anomale*, di cui due possibili esempi sono riportati nella figura qui sopra.

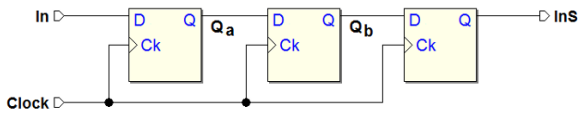
Nell'esempio a sinistra, l'uscita Q del flip-flop si porta su di un livello logico non valido, per un certo tempo t_m , prima di stabilizzarsi sul valore 1. Nel caso a destra, prima di assestarsi, l'uscita compie alcuni cicli di oscillazione tra i due livelli.

La rete logica che legge questi tipi di segnale può quindi cadere in *errore*. Si tratta di un comportamento *casuale*: sia per quanto riguarda la probabilità che esso si verifichi, sia per quanto concerne la durata del tempo t_m . Questi errori dipendono dalle caratteristiche fisiche dei flip-flop e dalla frequenza con cui variano i segnali in gioco. Tanto più ci si avvicina ai limiti di velocità del particolare componente in uso, tanto più la probabilità di comportamento metastabile diventa significativa (la relazione è di tipo esponenziale).

Tuttavia, in condizioni non limite, per un flip-flop ben progettato, gli errori dovuti a metastabilità risultano molto rari: il tempo medio tra due errori consecutivi può essere dell'ordine delle centinaia di anni. Tenendo conto che la probabilità che si verifichi un errore in un circuito digitale per altre cause (un guasto del circuito o un disturbo di natura elettromagnetica, per esempio) è molto più alta, possiamo in linea di massima considerare affidabile, per molte applicazioni, la sincronizzazione di un segnale ottenuta con questa tecnica.

6.1.2 Sincronizzatore a più stadi

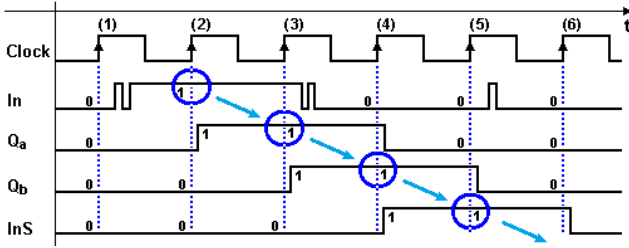
In condizioni critiche, o quando si vuole aumentare il margine di sicurezza, per ottenere la sincronizzazione di un segnale risulta necessario utilizzare una configurazione più complessa della precedente, ricorrendo a più flip-flop sincronizzatori collegati in cascata:



Iterando, in questo modo, la procedura di sincronizzazione, si riesce a ridurre la probabilità di errore dovuta alla metastabilità entro limiti estremamente piccoli e accettabili, in quanto l'eventuale comportamento metastabile del primo flip-flop è filtrato dal secondo, e via di seguito. A parte la questione della metastabilità, è interessante considerare la relazione temporale tra il segnale di ingresso In e l'uscita InS , come mostrato nella figura successiva.

Il primo flip-flop, ad ogni fronte di salita del clock, trasferisce sulla sua uscita Q_a il valore dell'ingresso In , come nel caso del precedente sincronizzatore. Q_a riproduce, quindi, l'andamento del segnale di ingresso In , sincronizzato e ripulito. A causa del fatto che il primo flip-flop genera la propria uscita Q_a dopo il fronte del clock, il secondo non può fare altro che leggere il nuovo valore di Q_a sul fronte del clock successivo a quello che lo ha generato. Risulta quindi che il tracciato temporale di Q_b è identico a quello di Q_a , ma ritardato

di un periodo di clock. Il flip-flop successivo reitera il processo, per cui l'uscita InS è ancora identica a Q_a , ma ritardata di un ulteriore ciclo di clock.



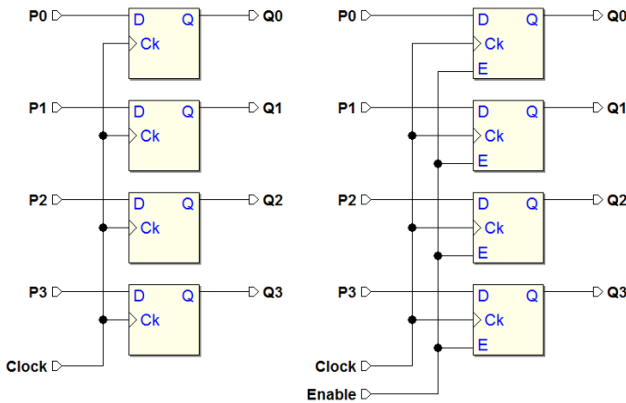
In figura abbiamo evidenziato che il valore generato da un flip-flop sul fronte (n) viene letto dal flip-flop successivo al fronte ($n + 1$). Nel seguito incontreremo molto spesso il concetto della lettura del valore al ciclo di clock successivo, nello scambio di informazione tra due reti sincrone.

6.2 Registri

Il *registro* è una importante struttura logica utilizzata per la memorizzazione temporanea dei dati. In un registro è possibile “scrivere” dei dati binari, mantenerveli per un certo tempo, e poi “leggerli” tutte le volte che sia necessario. Da questo punto di vista, per esempio, il flip-flop D è anch’esso un *registro*, essendo in grado di eseguire le operazioni descritte su di *un solo bit*. In generale, un registro è costituito da un numero di flip-flop pari al *numero dei bit* del dato che si vuole memorizzare. I registri consentono due diverse modalità di immagazzinamento e restituzione dei dati (il modo *parallelo* o il modo *seriale*), come vedremo nei prossimi paragrafi.

6.2.1 Registro parallelo

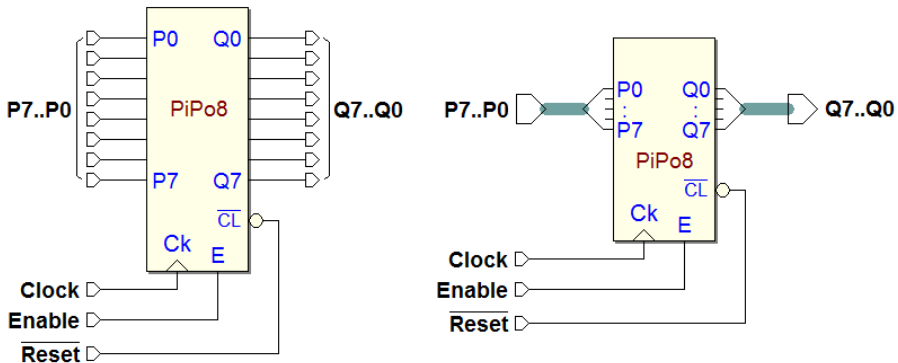
Le reti rappresentate nella figura seguente sono due esempi di *registri paralleli* in grado di memorizzare 4 bit di informazione:



Il registro a sinistra impiega flip-flop di tipo D, mentre l'altro utilizza il tipo E. Si tratta di reti *sincrone*, in quanto i flip-flop ricevono lo stesso clock. Secondo una vecchia denominazione, questo tipo di registri è anche riferito come “PIPO” (Parallel Input - Parallel Output). Normalmente i registri dispongono anche di un ingresso di inizializzazione: per semplicità, non è stato evidenziato in questa figura.

Nella versione di tipo D, il dato in ingresso $P3..P0$, all'occorrenza di un fronte di salita del *Clock*, è memorizzato *in parallelo* nei flip-flop. I dati sono mantenuti sulle uscite $Q3..Q0$, disponibili fino alla successiva scrittura: in assenza di questa, permangono indefinitamente nel registro, finché la rete è alimentata. Nella variante di tipo E, abbiamo in più l'opportunità di abilitare/disabilitare la scrittura, sotto controllo dell'ingresso *Enable*: questa è la versione più usata nei sistemi complessi, dove possono essere presenti, ad esempio, molti registri ed è necessario *selezionare* di volta in volta il registro nel quale immagazzinare l'informazione.

Qui sotto due esempi di registri paralleli sincroni da 8 bit, tratti dalla libreria del simulatore *Deeds*. Si tratta dello stesso componente, ma in due versioni differenti per quanto riguarda i collegamenti:

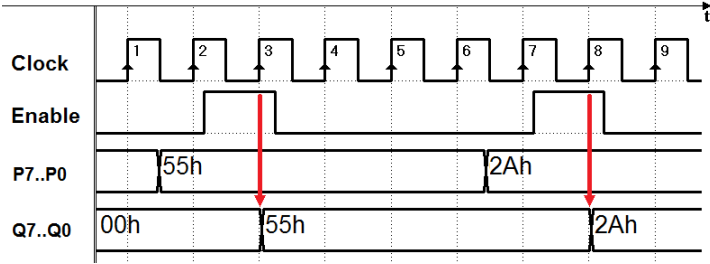


In quello sulla sinistra le terminazioni sono rappresentate singolarmente: gli 8 ingressi di dato $P7...P0$, le 8 uscite $Q7...Q0$, e gli ingressi di *Clock*, l'abilitazione *E*, nonché l'ingresso di inizializzazione asincrona \overline{CL} (*Clear*). Sulla destra, invece, gli ingressi $P7...P0$ e le 8 uscite $Q7...Q0$ sono mostrate sotto forma di *connessioni multifilari*, denominate più brevemente “di tipo *bus*”¹.

Nella figura seguente è mostrata una tipica sequenza di utilizzo di un registro da 8 bit, nella versione con connessioni di tipo *bus* e con abilitazione. Nel diagramma temporale, l'ipotesi è che il registro contenga inizialmente 0 in tutti i flip-flop. Come si osserva, la rappresentazione grafica dei valori è “*cumulativa*”, cioè si rappresentano su di una sola traccia tutti i bit del regi-

¹ Rappresentare connessioni multifilari come *bus* consente di semplificare e rendere più leggibile lo schema logico, specialmente se complesso.

stro, rappresentando solo le transizioni e, sinteticamente, all'interno, il valore numerico assunto tra una transizione e l'altra (in questo caso rappresentato in esadecimale):

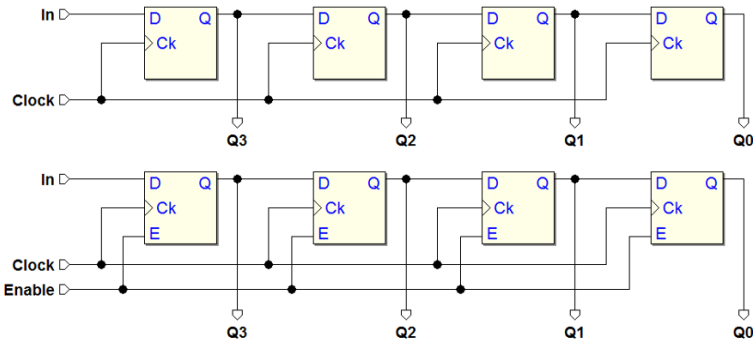


Nel nostro esempio le linee $P7..P0$ vengono impostate, nel tempo, ai valori 01010101 (= 55h) e poi 00101010 (= 2Ah). L'abilitazione *Enable* è fornita in modo da caricare il nuovo dato sulle uscite $Q7..Q0$ solo in corrispondenza dei fronti 3 e 8 del *Clock*.

I registri paralleli sono di larghissimo impiego nei sistemi digitali, dove i dati sono generalmente organizzati in 'parole' composte da più bit e immagazzinati in registri. Si osservi poi che, nella struttura della *generica rete sequenziale sincrona*, esaminata all'inizio del capitolo, lo *stato della rete* è memorizzato proprio in un registro.

6.2.2 Registro a scorrimento

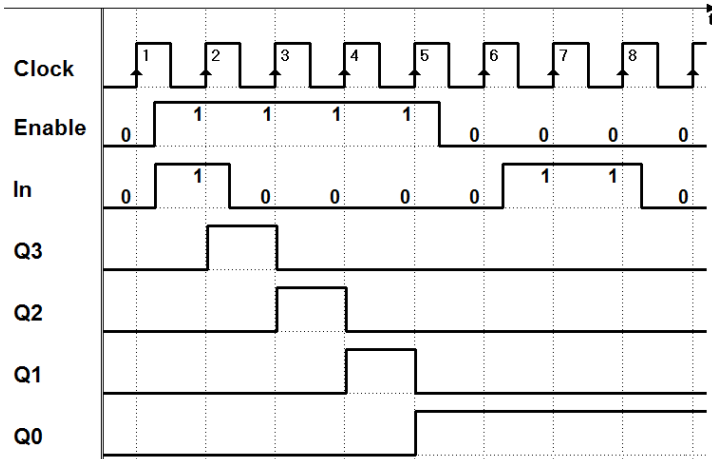
L'altra modalità d'ingresso dei dati nel registro è quella *seriale*, in cui i bit costituenti la parola da memorizzare sono presentati in successione, uno alla volta, ad un apposito ingresso. Un registro che consente tale modalità è chiamato *registro a scorrimento* (*shift register*, o SHR) e può essere ottenuto utilizzando flip-flop di tipo logico D, E o JK (abbiamo già incontrato la medesima struttura, con il tipo D, impiegata come sincronizzatore).



Nella figura qui sopra vediamo due esempi di registro a scorrimento a 4 bit, il primo realizzato con flip-flop di tipo D, il secondo con il tipo E. Si vede

come l'uscita di ciascun flip-flop è collegata all'ingresso del successivo mentre il *Clock* è comune a tutti, trattandosi di una rete sincrona (per semplicità, anche qui è stata omessa la rete di inizializzazione). L'ingresso "seriale" del registro è *In*. In entrambi i casi, le uscite sono $Q3..Q0$; nella seconda versione, è disponibile anche l'ingresso di abilitazione *Enable*.

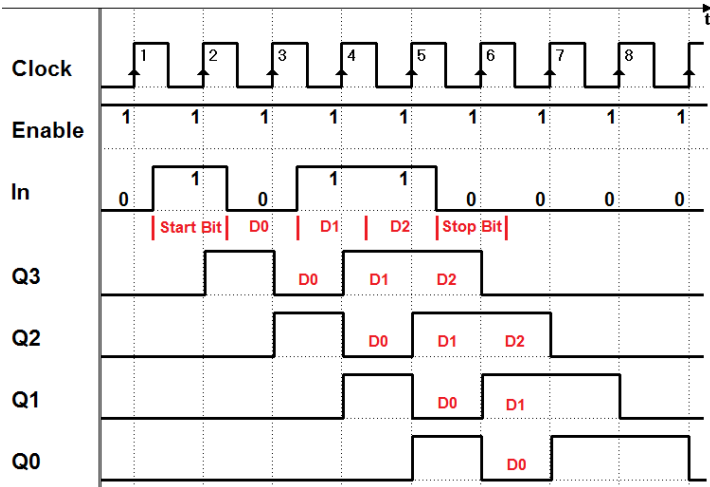
Il diagramma temporale qui sotto mostra un esempio di funzionamento del registro di tipo E. Si ipotizza che le uscite $Q3..Q0$ si trovino dapprima tutte a 0, e che l'ingresso *In* sia attivato in corrispondenza del fronte 2 del *Clock*. Inoltre, *Enable* è attivato per 4 cicli di *Clock*, dal 2 al 5:



Come già visto nel caso del sincronizzatore, il flip-flop più a sinistra, ad ogni fronte di salita del *Clock*, trasferisce sull'uscita $Q3$ il valore dell'ingresso *In*; quindi, l'uscita $Q3$ riproduce l'andamento del segnale di ingresso *In*, ma ritardato e sincronizzato con il *Clock*. Il secondo flip-flop legge i cambiamenti di $Q3$ sul successivo fronte di salita, per cui $Q2$ risulta identico a $Q3$, ma in ritardo di un ciclo di *Clock* e, analogamente, le stesse considerazioni valgono per tutti gli altri flip-flop.

Dal fronte 6 in avanti, poi, *Enable* è letto a 0. Mancando l'abilitazione, il contenuto del registro rimane inalterato, dal fronte 5 in avanti: la nuova attivazione dell'ingresso *In* è quindi ignorata.

Consideriamo ora un altro esempio di utilizzo di questo tipo di registro a scorrimento, questa volta impiegato in un caso reale, come elemento base di un "ricevitore di sequenza seriale". Nel diagramma temporale riportato nella figura seguente, si ipotizza di ricevere sull'ingresso *In* una sequenza di bit, uno dopo l'altro, secondo un formato convenuto. Il formato prevede, in questo esempio, che i bit siano organizzati in un gruppo (o "pacchetto") di cinque bit, composto da un bit iniziale a 1, chiamato *bit di start*, tre bit di *informazione* ($D0$, $D1$, $D2$) ed un bit finale a 0, denominato *bit di stop*. Inoltre, ciascun bit ha la durata pari ad un periodo del clock (come evidenziato in figura):

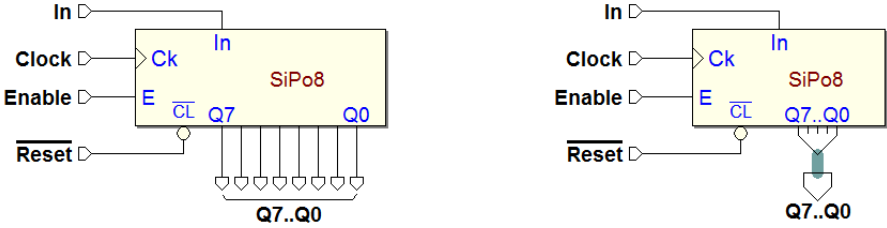


Si suppone che la linea *IN*, a riposo, sia normalmente a 0, per cui il *bit di start* a 1 ha la funzione di segnalare l'inizio della sequenza; il *bit di stop* serve per separare, con uno 0, due pacchetti consecutivi.

Nel nostro esempio, il *bit di start* è letto dal primo flip-flop sul fronte 2, e quindi i bit di informazione ($D0 = 0, D1 = D2 = 1$) sui fronti 3, 4 e 5, ed infine il *bit di stop* sul fronte 6. Nel ricevere la sequenza, il registro memorizza e fa scorrere, uno dopo l'altro, i bit ricevuti. In particolare, dopo il fronte 6 i bit di informazione $D0, D1$ e $D2$ sono resi disponibili, *in parallelo*, sulle uscite $Q0, Q1$ e $Q2$ (in questo esempio semplificato, sui successivi cicli di *Clock* lo scorrimento prosegue e quindi si ha la progressiva perdita del dato).

La *conversione seriale/parallelo* è utilizzata largamente nei sistemi digitali di telecomunicazione (dove la comunicazione seriale è utilizzata soprattutto sulle grandi distanze) e in tutti i sistemi di elaborazione in cui sia necessario (per ragioni di costo, usabilità e praticità) ridurre il numero di fili utilizzati per le connessioni tra i moduli del sistema. Le connessioni USB (*Universal Serial Bus*) sono un esempio di comunicazione in formato seriale. Il formato parallelo, invece, permette una più efficiente e veloce elaborazione dei dati, ed è generalmente usato all'interno dei sistemi di calcolo.

Qui sotto due esempi di registro a scorrimento da 8 bit, presi dalla libreria del simulatore *Deeds*:



Il componente è lo stesso, la differenza tra le due versioni è nelle connessioni: quello a destra utilizza il tipo *bus* per le uscite $Q7..Q0$. Sul componente si legge la sigla “SIPO”, ossia *Serial Input - Parallel Output*, secondo la tradizionale denominazione.

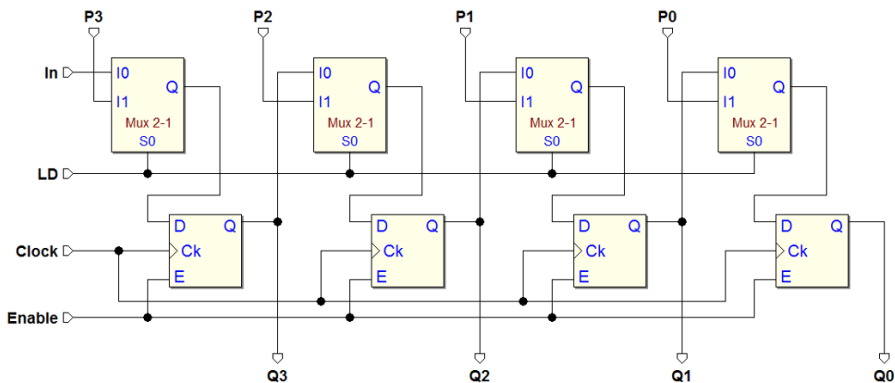
Nella classificazione troviamo anche i registri “SISO” (*Serial Input - Serial Output*), la cui struttura interna è tuttavia identica ai SIPO. È facile constatare che, di principio, è sufficiente scegliere una qualunque delle uscite Q per ottenere un’uscita seriale.

Tuttavia, i registri SISO propriamente detti, cioè effettivamente dotati di *un’unica uscita seriale*, sono normalmente realizzati con un elevato numero di flip-flop (anche migliaia), e vengono utilizzati per ottenere un segnale ritardato di altrettanti cicli di clock. A causa dell’elevato numero di collegamenti richiesti, non risulta pratico rendere disponibili all’esterno le uscite intermedie.

6.2.3 Registro a scorrimento con caricamento parallelo

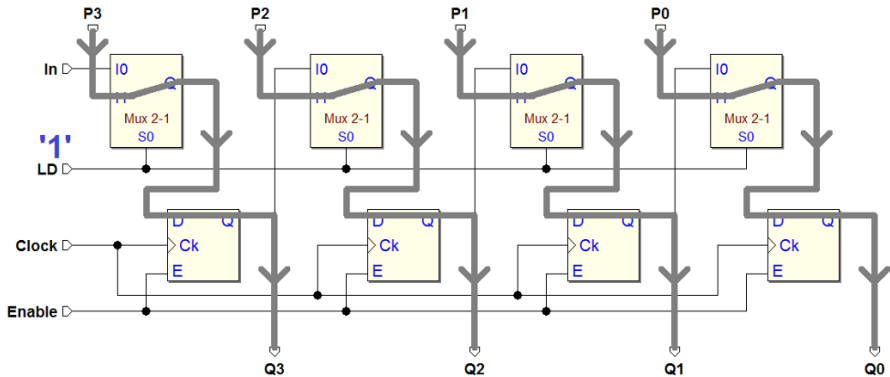
Abbiamo visto che il registro con ingresso seriale ed uscite parallele effettua direttamente la conversione seriale-parallelo. Per ottenere la conversione contraria, dal *formato parallelo a quello seriale* è necessario prima caricare nel registro il dato, *in formato parallelo*, per poi farlo scorrere *serialmente*.

Tale funzione può essere ottenuta combinando insieme le strutture già viste del registro parallelo e del registro a scorrimento, con l’aggiunta di alcuni selettori (multiplexer). La figura qui sotto mostra lo schema di un registro a scorrimento a quattro bit con caricamento parallelo (è stata omessa la rete di inizializzazione anche in questo esempio, per semplicità):

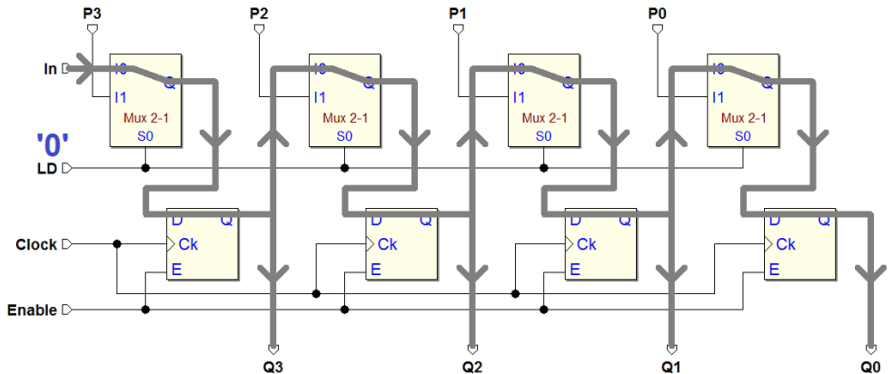


Si riconoscono in figura i quattro flip-flop di tipo E, con le loro uscite $Q3..Q0$, come nei precedenti tipi di registro. Sono presenti però sia l’ingresso seriale In , che gli ingressi paralleli $P3..P0$. Il nuovo ingresso LD (“Load”) comanda il caricamento parallelo, ottenuto tramite i quattro selettori “Mux 2-1”, che permettono di scegliere il dato da mandare in ingresso ai flip-flop.

Come visibile nella figura qui sotto, quando $LD = 1$, i selettori instradano gli ingressi paralleli $P3..P0$ verso i flip-flop. Se $Enable = 1$, al successivo fronte di salita del $Clock$ il dato parallelo in ingresso è caricato nei flip-flop, e compare sulle uscite $Q3..Q0$:



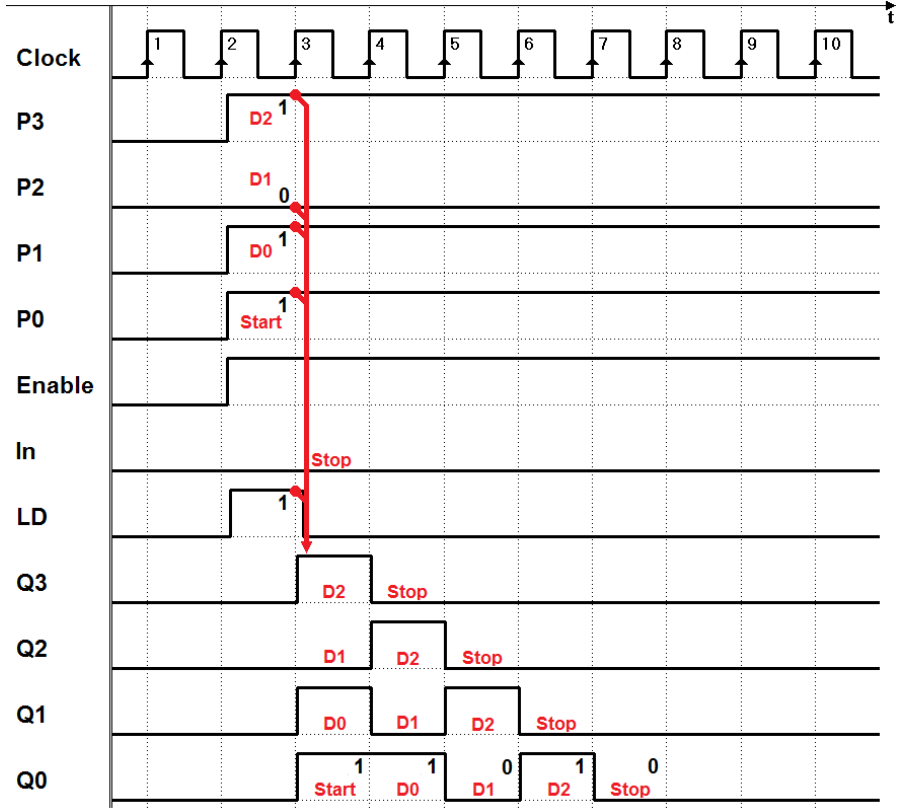
Se $LD = 0$, invece, il percorso del dato è finalizzato allo scorrimento, come si vede nella figura seguente. L'ingresso seriale In è diretto al primo flip-flop, l'uscita di questo è portata al secondo, e così via. L'operazione di scorrimento è eseguita, ovviamente, sul successivo fronte di salita del $Clock$. Si presti attenzione al fatto che le operazioni di caricamento parallelo e scorrimento seriale sono mutuamente esclusive:



Abbiamo visto in precedenza un esempio di *ricezione di sequenza seriale*. Un registro a scorrimento con caricamento parallelo, come quello esaminato ora, si presta perfettamente al compito di *trasmettere una sequenza seriale*.

Nel diagramma temporale riportato nella figura seguente, ipotizziamo di trasmettere sull'uscita $Q0$ una sequenza di bit secondo il formato definito nell'esempio della ricezione seriale. Ricordiamo che il formato prevedeva un *pacchetto* di cinque bit: un bit iniziale a 1 (il *bit di start*), poi tre bit di *informazione* $D0$, $D1$, $D2$ ed infine un *bit di stop* a 0. La durata di ciascun bit era stata definita pari ad un ciclo di clock.

Si vuole trasmettere un pacchetto con $D_0 = 1$, $D_1 = 0$ e $D_2 = 1$. Come evidenziato in figura, impostiamo questi valori sugli ingressi paralleli del registro: $P_1 = D_0$, $P_2 = D_1$ e $P_3 = D_2$. Si presti attenzione al fatto che P_0 è stato impostato a 1, in qualità di *bit di start* da trasmettere per primo, mentre l'ingresso seriale In è predisposto a 0 (per inviare, in ultimo, il *bit di stop*):

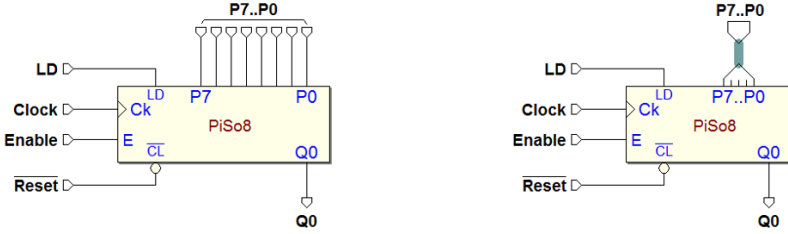


Insieme al dato, abilitiamo il registro attivando l'ingresso *Enable*, e portiamo *LD* a 1 per un ciclo di clock, in modo che sul fronte 3 di questo avvenga il caricamento parallelo nel registro. Sull'uscita Q_0 si presenta quindi il *bit di start*, che viene mantenuto per un ciclo di clock.

All'arrivo del fronte 4, dato che $LD = 0$, il registro scorre a destra, presentando sull'uscita Q_0 questa volta il valore di D_0 , mantenuto per un ciclo di clock, e così via, fino a trasmettere tutti i bit.

Si noti che l'ingresso seriale In , nel frattempo, ad ogni passo di scorrimento, ha inserito nel registro uno 0, da sinistra, per cui il registro progressivamente si azzerava e, in ultimo, viene trasmesso uno 0, il *bit di stop* (in questo esempio semplificato, sui successivi cicli di *Clock* lo scorrimento prosegue, e il registro continua a inviare 0 sulla linea Q_0).

Qui sotto due esempi di registro a scorrimento con caricamento parallelo, da 8 bit, presi dalla libreria del simulatore *Deeds*:



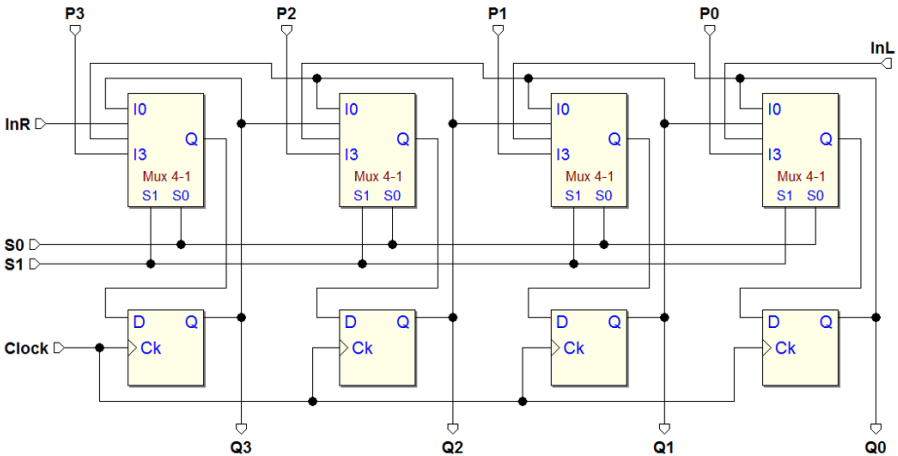
Anche in questo caso, la differenza tra i due componenti è solo nelle connessioni: quello a destra utilizza il tipo *bus* per gli ingressi *P7..P0*. Il componente è etichettato come “PISO” (*Parallel Input - Serial Output*), sigla che completa la casistica della denominazione classica.

Si noti però che questo componente di libreria presenta in uscita solo la linea *Q0*. Inoltre vi sono alcune piccole differenze nella logica di abilitazione (infatti qui l'ingresso *E* controlla l'abilitazione del solo scorrimento; per caricare in parallelo il dato è sufficiente attivare l'ingresso *LD*).

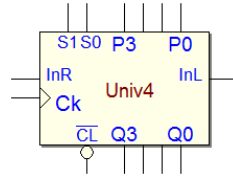
6.2.4 Registro a scorrimento universale

I registri presentati fino ad ora permettono di effettuare sia il caricamento che la lettura dei dati, in formato seriale o parallelo, ma lo scorrimento avviene solo in una sola direzione (a destra: $Q3 \rightarrow Q0$). Esistono anche registri nei quali lo scorrimento può avvenire in entrambe le direzioni.

Un registro che, oltre allo scorrimento bidirezionale, può essere caricato e letto in formato seriale o parallelo, è chiamato *registro universale*. Qui un esempio di registro universale a quattro bit:



Nello schema non è stata riportata la rete di inizializzazione solo per semplificare la comprensione generale dello schema. Per riassumerne le terminazioni e la relativa funzionalità, conviene fare riferimento al simbolo del corrispondente componente, tratto dalla libreria di *Deeds*:



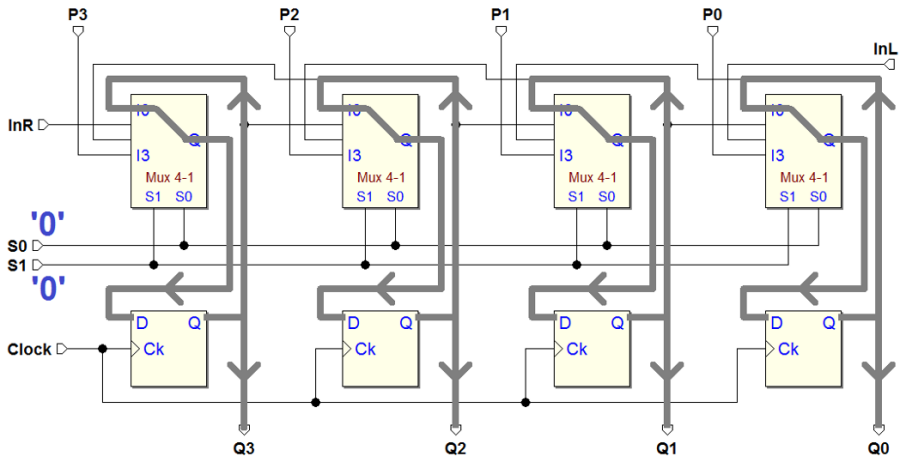
Come in un registro parallelo, sono presenti gli ingressi $P3..P0$ e le uscite $Q3..Q0$. Sono ovviamente presenti gli ingressi di clock (CK) e di inizializzazione (\overline{CL}). InR e InL sono gli ingressi seriali, rispettivamente per lo scorrimento verso destra e verso sinistra.

Gli ingressi $S1$ e $S0$ controllano la funzione svolta dal componente, secondo la tabella qui sotto:

$S1$	$S0$	Funzione
0	0	Mantenimento dell'informazione
0	1	Scorrimento a "destra"
1	0	Scorrimento a "sinistra"
1	1	Caricamento parallelo

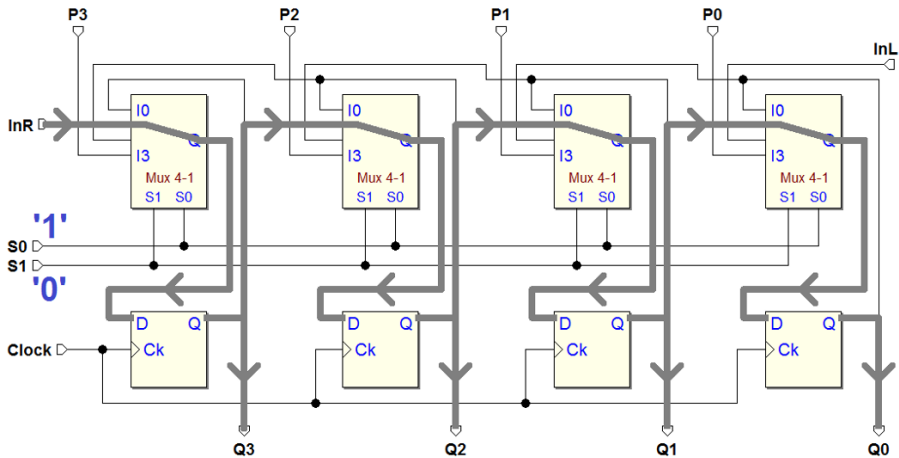
Questo avviene perché $S1$ e $S0$ controllano la selezione dei "Mux 4-1" visibili nello schema, sopra ai rispettivi flip-flop.

La configurazione ($S1 = 0, S0 = 0$) collega ciascun ingresso D con l'uscita Q del medesimo flip-flop. All'occorrenza del fronte di salita del clock CK il registro ricarica nei flip-flop i valori precedenti, per cui l'informazione è mantenuta invariata, come è evidenziato nella figura qui sotto:



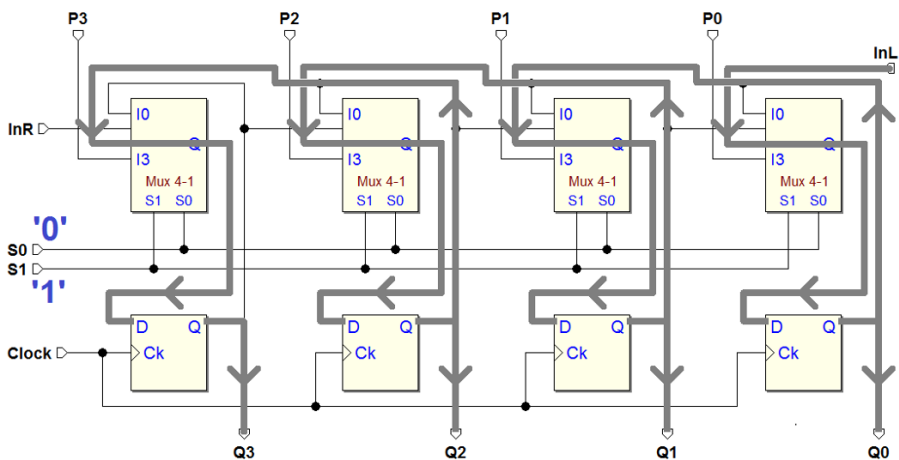
La combinazione ($S1 = 0, S0 = 1$) configura il registro per lo scorrimento a destra (vedi figura seguente).

Al fronte di salita del clock, il dato presente su $Q1$ è caricato nell'ultimo flip-flop a destra, e compare su $Q0$, quello su $Q2$ è trasferito su $Q1$, e $Q3$ su $Q2$. Infine, InR è copiato nel primo flip-flop e compare su $Q3$:

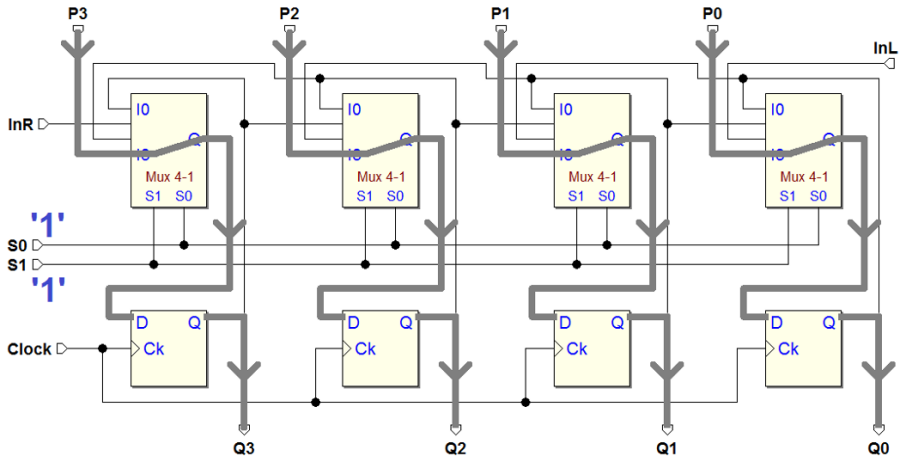


Lo scorrimento a sinistra è visibile nella figura qui in basso, e si ottiene con la combinazione ($S1 = 1, S0 = 0$).

Nonostante l'apparente complessità dei percorsi rappresentati in figura, si tratta di un instradamento analogo al precedente, ma nella direzione contraria. In questo caso, l'ingresso seriale è InL : all'arrivo del fronte di salita del clock, il valore di InL è caricato su $Q0$, mentre le altre uscite scalano a sinistra di un posto:



Infine, la configurazione ($S1 = 1, S0 = 1$) instrada su ciascun ingresso D il valore del corrispondente ingresso P di caricamento parallelo. All'occorrenza del fronte di salita del clock, il registro eseguirà quindi un caricamento parallelo, come evidenziato nella figura seguente:



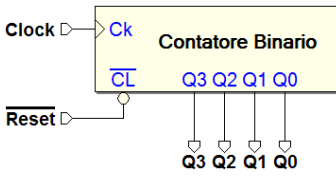
6.3 Contatori

Un'altra classe di reti sequenziali di uso comune è costituita dai *contatori*. Con tale termine si indica una rete che genera una sequenza numerica in un particolare codice (si pensi, ad esempio, ad una sequenza crescente formata da numeri binari, rappresentati con un certo numero di bit). Il passaggio da un elemento della sequenza al successivo è determinato dal *fronte attivo* dell'ingresso di clock della rete. Un contatore è *sincrono* quando è *sincrona* la rete di flip-flop che lo realizza.

6.3.1 Contatore binario

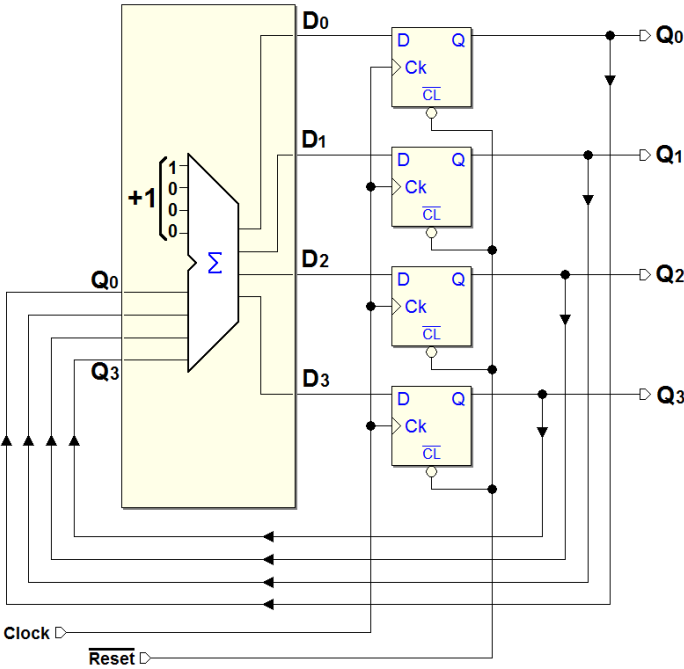
Nella figura seguente è raffigurato, come blocco funzionale, un esempio di *contatore in codice binario naturale a 4 bit*. La tabella sulla destra riporta le *16 combinazioni* delle sue uscite. Si tratta di una *sequenza crescente* e, per questo, il contatore è detto "*in avanti*".

Un *ciclo di conteggio* è formato dalla sequenza delle 16 differenti configurazioni generabili: si dice anche che il conteggio avviene "*modulo 16*". Arrivato al massimo numero rappresentabile 1111, il conteggio prosegue ciclicamente dal valore 0000. L'avanzamento del conteggio è controllato dal fronte di salita del clock. L'ingresso di \overline{CL} (*Reset*) consente di inizializzare la rete al valore 0000.



Q3	Q2	Q1	Q0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

La struttura interna di un contatore può essere ricondotta a quella della rete sequenziale sincrona definita all’inizio del capitolo, composta da un registro parallelo di flip-flop D-PET e da una rete combinatoria che ne determina il comportamento. Nel caso del contatore qui proposto, la funzione richiesta alla rete combinatoria è l’incremento di una unità del numero binario presente sulle uscite, come rappresentato nella figura seguente:



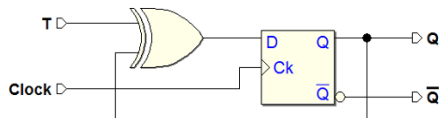
Intuitivamente, quindi, il conteggio può essere ottenuto con un sommatore, aggiungendo la costante $+1$ al numero attualmente presente alle uscite dei flip-flop. Il risultato viene proposto all'ingresso dei flip-flop: questi caricheranno il nuovo numero sul successivo fronte di salita del clock. Si noti che il riporto dal quarto bit della somma è ignorato, dal momento che i bit memorizzabili sono 4: si ottiene così un conteggio modulo 16.

Si vuole adesso procedere in modo più sistematico, descrivendo, nella tabella di verità qui sotto, il comportamento che la rete combinatoria deve produrre:

$Q3$	$Q2$	$Q1$	$Q0$	$D3$	$D2$	$D1$	$D0$
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	1	0	1	0
1	0	1	0	1	0	1	1
1	0	1	1	1	1	0	0
1	1	0	0	1	1	0	1
1	1	0	1	1	1	1	0
1	1	1	0	1	1	1	1
1	1	1	1	0	0	0	0

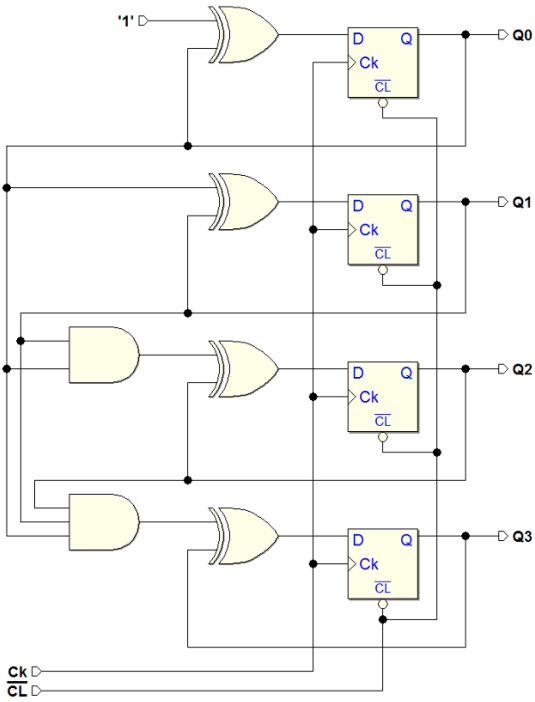
Nella parte sinistra della tabella sono rappresentate le 16 possibili combinazioni delle uscite dei flip-flop $Q3..Q0$ e, in corrispondenza, nella parte a destra, i valori degli ingressi $D3..D0$ che devono essere prodotti dalla rete combinatoria. Dal momento che il fronte di salita del clock caricherà nei flip-flop il valore elaborato dalla rete combinatoria, di fatto, la tabella mette in relazione lo *stato attuale* della rete con lo *stato successivo*.

Ricordiamo che un flip-flop di tipo D-PET, collegato con una EXOR in retroazione, riproduce la funzionalità del tipo T, come visto in precedenza:



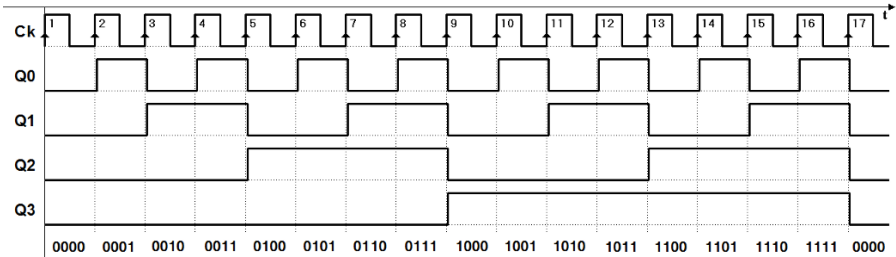
Il flip-flop, se all'ingresso della EXOR poniamo $T = 0$, mantiene il valore precedentemente memorizzato; inverte la sua uscita, invece, quando $T = 1$.

Tenendo conto di questo, dalla sintesi della rete combinatoria descritta nella tabella precedente, omettendo tutti i passaggi intermedi, si ricava lo schema del contatore, basato su flip-flop D-PET collegati come T:



La struttura regolare della rete permette di indagarne il funzionamento anche sul piano intuitivo, con l'aiuto della simulazione temporale qui sotto.

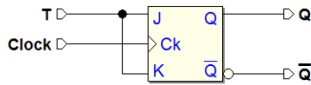
L'uscita Q_0 cambia valore ad ogni fronte attivo di clock poiché il relativo flip-flop riceve in ingresso sempre il negato di Q_0 . Nel caso di Q_1 , invece, la condizione di inversione è vera solo quando Q_0 si presenta a 1 all'EXOR; al contrario, se $Q_0 = 0$, Q_1 mantiene il valore precedente.



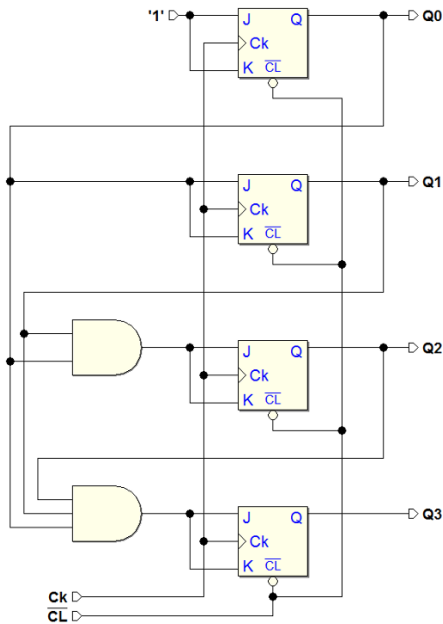
Analogamente, Q_2 cambia quando le uscite Q_0 e Q_1 sono entrambe a 1, grazie alla porta AND a due ingressi; infine, Q_3 commuta solo quando sono a 1 tutte le Q_0 , Q_1 e Q_2 , grazie alla AND a tre ingressi. La simulazione mostra,

in generale, che un'uscita cambia quando tutte quelle meno significative sono alte. Questa osservazione consente di estendere il contatore binario, in modo intuitivo, ad un numero qualsiasi di bit.

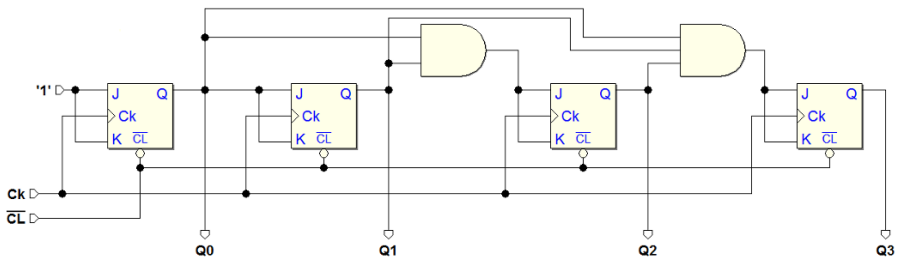
Consideriamo ora la funzionalità di inversione che si può ottenere con il flip-flop di tipo JK-PET (è implicita nel tipo logico ed è sufficiente collegare insieme gli ingressi J e K):



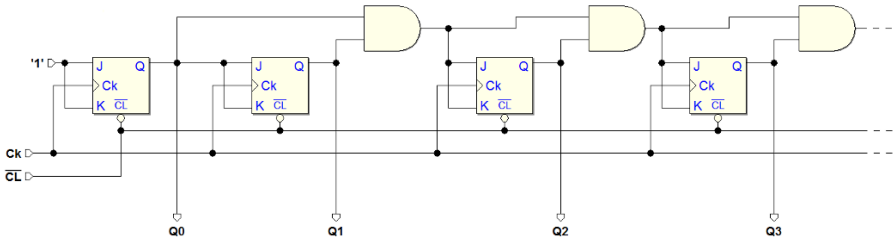
Sostituendo i flip-flop D con altrettanti JK, lo schema precedente si semplifica, come si vede nella figura seguente:



Possiamo anche disegnare la stessa rete disponendo i flip-flop in orizzontale, come nella figura seguente. Si ha il vantaggio di una visualizzazione più intuitiva del pilotaggio in "cascata" degli ingressi dei flip-flop:

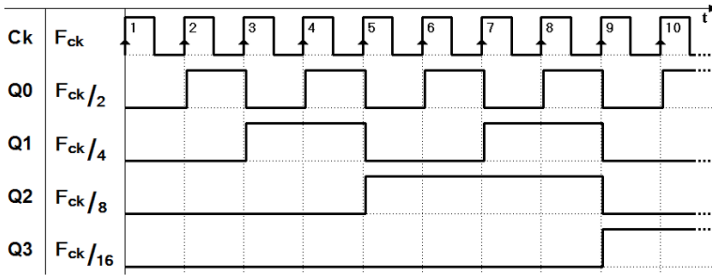


La rete si semplifica ulteriormente se teniamo conto della proprietà associativa della AND, che ci consente di utilizzare delle semplici AND a due soli ingressi, per cui si perviene alla struttura mostrata nella figura seguente:



Una struttura di questo tipo, tuttavia, riduce la massima frequenza di funzionamento all’aumentare del numero dei bit, poiché il numero dei livelli della rete combinatoria cresce linearmente con il numero dei flip-flop presenti.

Torniamo ora nuovamente alla simulazione temporale del contatore binario, di cui osserviamo qui un intervallo:



Se si considera la relazione tra l’andamento temporale dei segnali (le cosiddette “forme d’onda”), si osserva che Q_0 presenta un periodo pari al doppio di quello del clock Ck . Analogamente, il periodo di Q_1 è il doppio di quello di Q_0 e il quadruplo rispetto al clock.

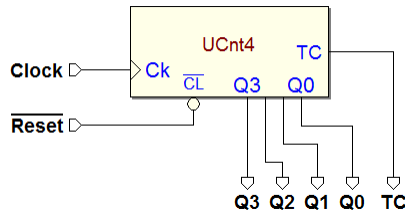
Se F_{ck} è la frequenza del segnale di clock, la frequenza del segnale Q_0 è pari a $F_{ck} / 2$, quella di Q_1 è $F_{ck} / 4$, e così via. Inoltre, nel nostro particolare esempio, le forme d’onda delle uscite sono anche simmetriche, ossia la durata dei due semi-periodi (alto e basso), è identica.

Un contatore, quindi, può essere utilizzato come “divisore di frequenza”, cioè ci consente di ottenere dei segnali periodici derivati dal clock, di frequenza pari ad un sottomultiplo (per potenze del 2) di quella del clock.

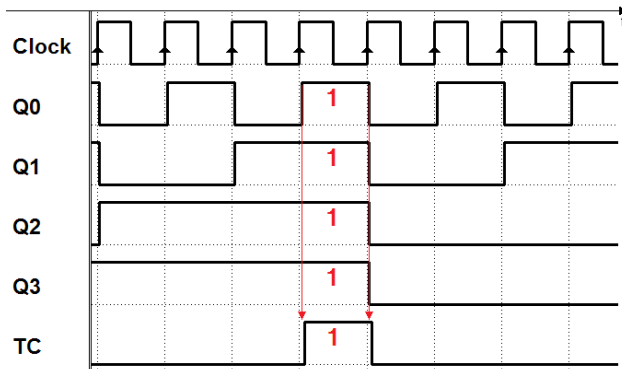
Nella figura seguente troviamo un esempio di contatore binario sincrono, in avanti e ciclico, a 4 bit, tratto dalla libreria del simulatore Deeds: “UCnt4” (“Up Counter 4-bits”). E’ un contatore funzionalmente identico a quello ora descritto, ma con l’aggiunta della uscita TC (“Terminal Count”, o “Conteggio

terminale): TC si attiva quando il numero presente sulle uscite del contatore ha raggiunto il numero più alto del conteggio.

Si presti attenzione che si tratta di una semplice *funzione combinatoria* delle uscite, definita come: $TC = Q3 \cdot Q2 \cdot Q1 \cdot Q0$.

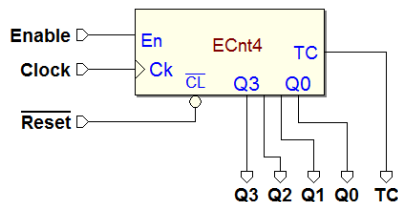


Nella figura seguente, un esempio di simulazione temporale del componente, in cui è evidenziata l'attivazione dell'uscita TC , nel corso del conteggio:



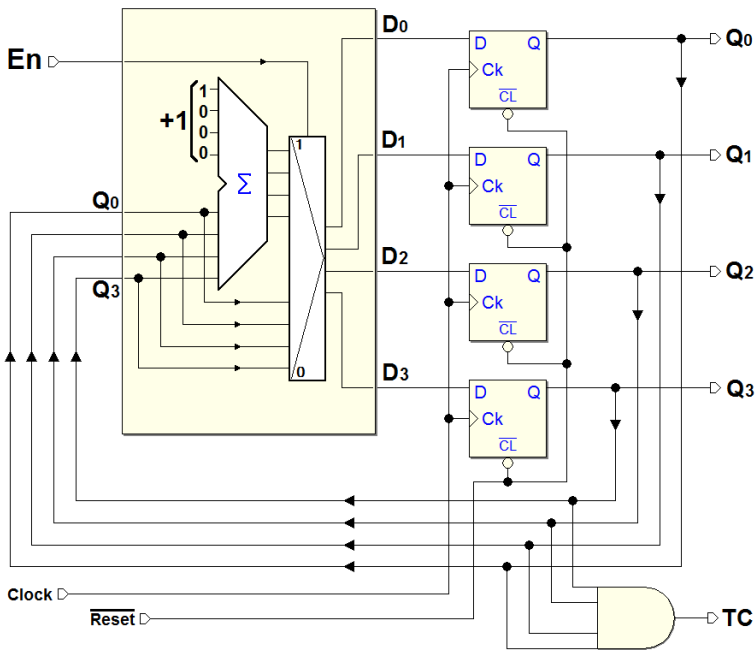
6.3.2 Contatore con abilitazione

In figura è visibile un esempio di contatore simile a quello ora visto. Possiede in più un ingresso di *abilitazione*, che controlla la funzione di conteggio:



Se En è a 1 il conteggio è *abilitato*, altrimenti no. Quando il conteggio è disabilitato, le uscite del contatore non cambiano, nonostante il succedersi dei fronti del clock. Il componente considerato come esempio è il tipo “ECnt4” (“Up Counter with Enable 4-bits”), preso dalla libreria del simulatore *Deeds*.

Esaminiamo, in termini di principio, come funziona l'abilitazione. La figura seguente descrive la funzionalità di un contatore binario in avanti a 4 bit con abilitazione:



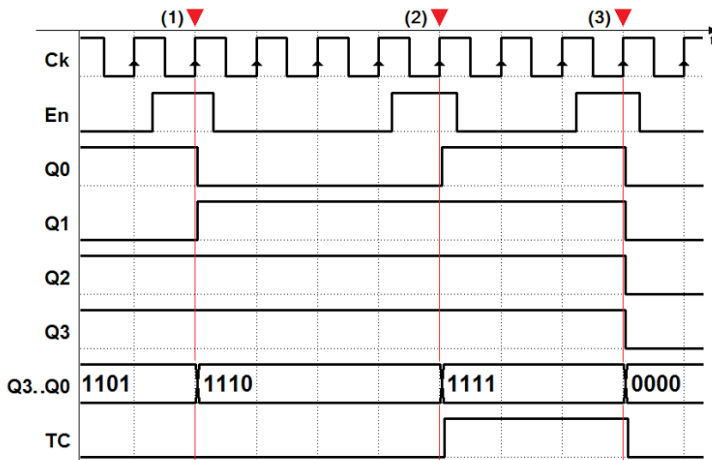
Confrontando la sua struttura con quella precedente vista, qui è stato aggiunto un selettore davanti agli ingressi dei flip-flop, comandato da En . Quando En è alto, il selettore collega l'uscita del sommatore ai flip-flop, per cui rientriamo del normale conteggio, come visto in precedenza.

Quando En è basso, invece, il selettore collega agli ingressi dei flip-flop le uscite in retroazione degli stessi: ad ogni fronte di clock vengono quindi confermati i valori precedenti (e il conteggio rimane fermo). Il TC è generato in funzione del valore assunto dai flip-flop: si attiva se sulle uscite $Q3..Q0$ è presente il valore 1111.

Il diagramma temporale riportato nella figura seguente mostra un esempio di funzionamento del contatore con abilitazione. Il tracciato inizia con En impostato basso, e il conteggio è fermo (per ipotesi al valore 1101). Il conteggio avanza sui fronti di salita del clock Ck , ma solo se $En = 1$, che in questa simulazione è attivato tre volte, per la durata di un ciclo di clock.

Alla prima (1) attivazione di En , il conteggio passa al valore 1110, per poi fermarsi. Un altro incremento del valore si avrà solo alla successiva (2) attivazione di En : arrivando al numero 1111. Si noti che, per tutto il tempo in cui il contatore presenta questa combinazione delle uscite, l'uscita TC risulta attivata, ad indicare che il contatore è arrivato al suo valore "terminale".

Infine, il contatore avanza di un'altra unità in corrispondenza dell'ultima (3) attivazione di En : le sue uscite passano da 1111 a 0000, in quanto il conteggio è ciclico e utilizza solo 4 bit (il riporto è ignorato):

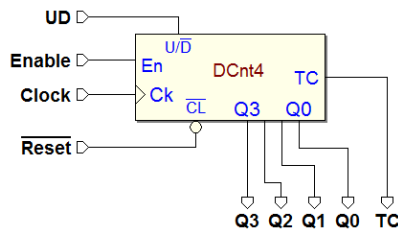


Si noti che $TC = 1$, in presenza della abilitazione $En = 1$, segnala all'esterno che il conteggio, sul fronte attivo del clock, passerà dal valore massimo a zero: questo sarà utile, come vedremo nel seguito, per collegare più contatori “in cascata”, al fine di ottenere un conteggio su di un numero maggiore di bit.

Riassumendo, un contatore con abilitazione permette di *contare il numero di volte* in cui, in corrispondenza del fronte attivo del clock, è attivato l'ingresso En , mantenendo una rigorosa sincronicità delle operazioni.

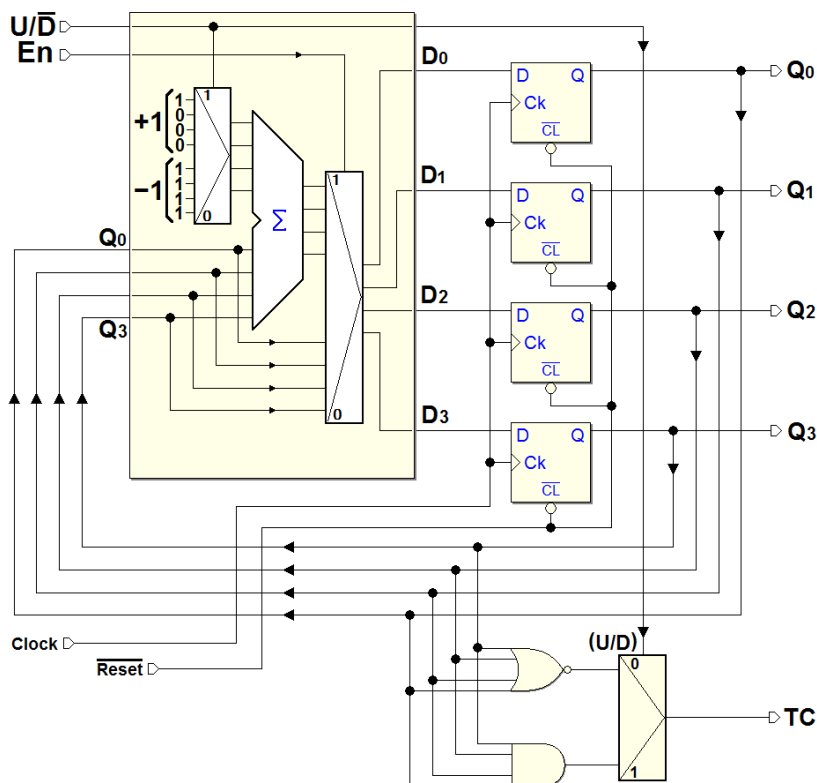
6.3.3 Contatore bidirezionale

Un contatore si dice *bidirezionale* quando la sequenza di conteggio può essere eseguita *in avanti* oppure *all'indietro*. Nella figura seguente si osserva, a titolo di esempio, il contatore binario di tipo “DCnt4” (“Up/Down Counter with Enable 4-bits”), tratto dalla libreria del simulatore *Deeds*:



Come si vede in figura, il *contatore bidirezionale* possiede l'ulteriore ingresso U/\overline{D} , con il compito di impostare la direzione del conteggio.

Il contatore bidirezionale (vedi figura qui sotto) può essere ricondotto alla struttura dei contatori esaminati in precedenza, ma con la differenza principale che il contenuto del registro può essere sia incrementato che decrementato:

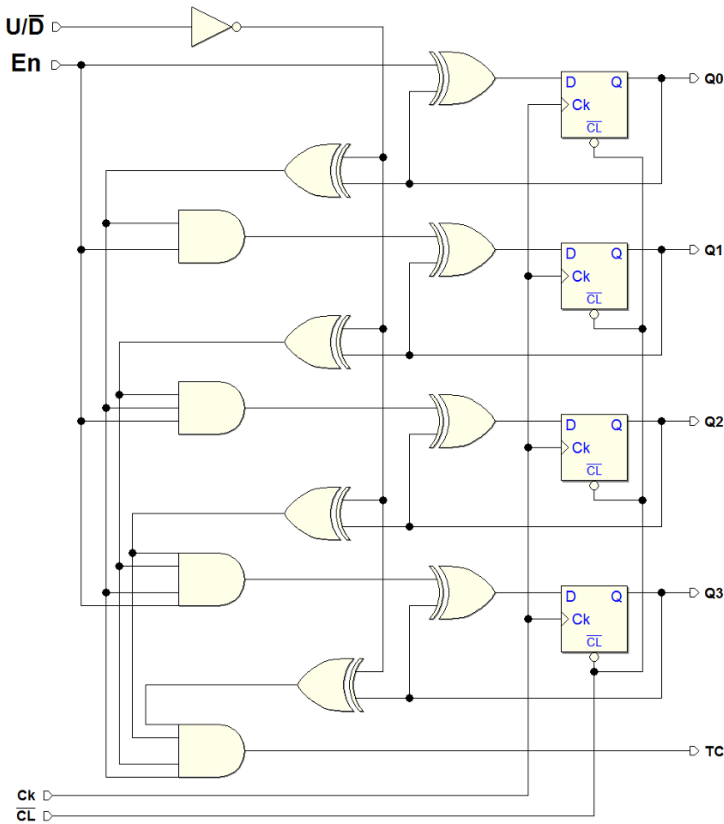


Questo avviene presentando al sommatore, tramite un ulteriore selettore (quello in alto a sinistra), le costanti $+1$ oppure -1 (rappresentate in *codice complemento a due*), sulla base del valore dell'ingresso di controllo della direzione del conteggio U/\overline{D} .

L'ingresso U/\overline{D} controlla anche un altro selettore (quello in basso a destra in figura); questo permette di generare il *Terminal Count* in modo coerente con la direzione del conteggio. Infatti, nel conteggio in avanti, come abbiamo visto, TC è attivato quando è raggiunto il massimo numero (1111); al contrario, quando $U/\overline{D} = 0$, nel conteggio all'indietro, TC è posto a 1 se in uscita abbiamo il numero più piccolo (0000).

Lo schema di principio ora visto trova una possibile sintesi circuitale nella figura seguente. La rete è molto simile, nella struttura, a quella già esaminata per il contatore in avanti senza abilitazione. Per esercizio, tenendo conto delle similitudini, proviamo a “interpretare” gli elementi di questa nuova rete da un punto di vista intuitivo. Anche in questo caso, i flip-flop D-PET sono

collegati in modo da riprodurre la funzionalità del tipo T, grazie agli EXOR in retroazione (nella figura, quelli che pilotano il D dei flip-flop):

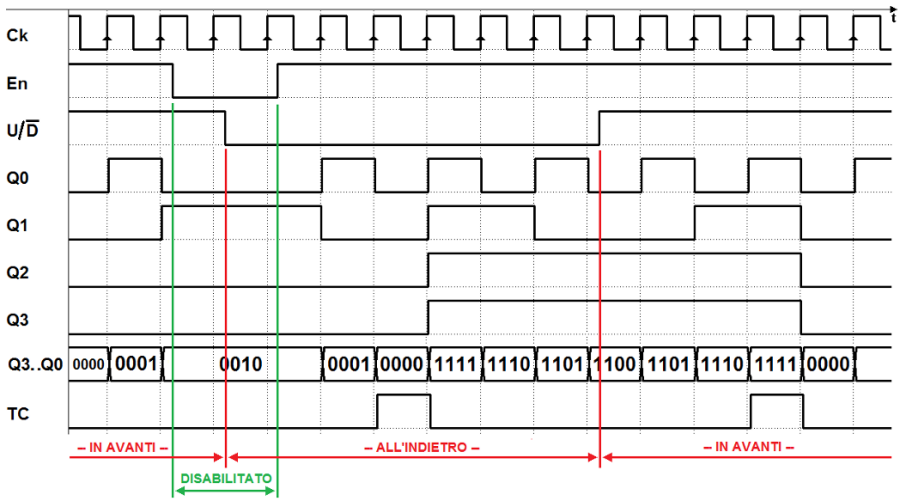


Sono stati aggiunti quattro EXOR nel percorso di retroazione. Sotto comando dell'ingresso U/\overline{D} , è possibile invertire, oppure no, il valore letto dalle uscite dei flip-flop. Se $U/\overline{D} = 1$, gli EXOR citati *non invertono*, per cui è come se non ci fossero, e la rete funzionerà come contatore in avanti; altrimenti, se $U/\overline{D} = 0$, la rete funzionerà come contatore all'indietro.

L'ingresso En agisce in parallelo su tutta la retroazione, come si vede nello schema. Si può dimostrare facilmente che se $En = 0$, i flip-flop sono obbligati nella condizione di ricaricare il proprio stesso valore (ad ogni fronte attivo del clock), per cui le uscite non cambiano ed il conteggio è fermo. Se $En = 1$, invece, tutto va come se tale ingresso non ci fosse, e il conteggio è abilitato.

L'uscita TC , indipendente da En , è generata da una porta AND. I quattro ingressi della AND provengono dalle uscite dei flip-flop, invertite o meno, in funzione dell'ingresso U/\overline{D} . TC sarà attivata quando le uscite dei flip-flop arriveranno a 1111, se il conteggio è *in avanti* (oppure a 0000, se *all'indietro*).

Nella figura seguente è riportato un esempio di simulazione temporale del contatore bidirezionale con abilitazione:

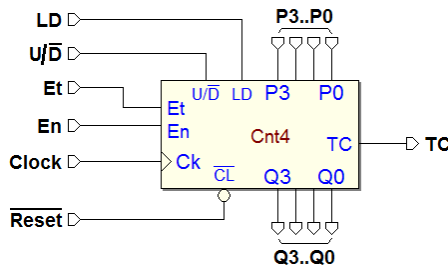


Nella prima parte del tracciato temporale, il contatore è abilitato ($En = 1$) e sta contando in avanti ($U/\bar{D} = 1$). Poi, per la durata di due cicli di clock, il contatore viene disabilitato ($En = 0$) ed il conteggio rimane fermo, mantenendo il numero a cui era arrivato (0010). Nel frattempo viene chiesto al contatore di cambiare la direzione di conteggio ($U/\bar{D} = 0$). Ricevuta nuovamente l'abilitazione ($En = 1$), riprende a contare, ma all'indietro. Giunto a 0000, attiva il TC e continua il conteggio (ripartendo da 1111), fino a scendere a 1100, quando gli ordiniamo di contare di nuovo in avanti ($U/\bar{D} = 1$). Arrivato a 1111, attiva ancora il TC , e riparte da 0000 (e così via).

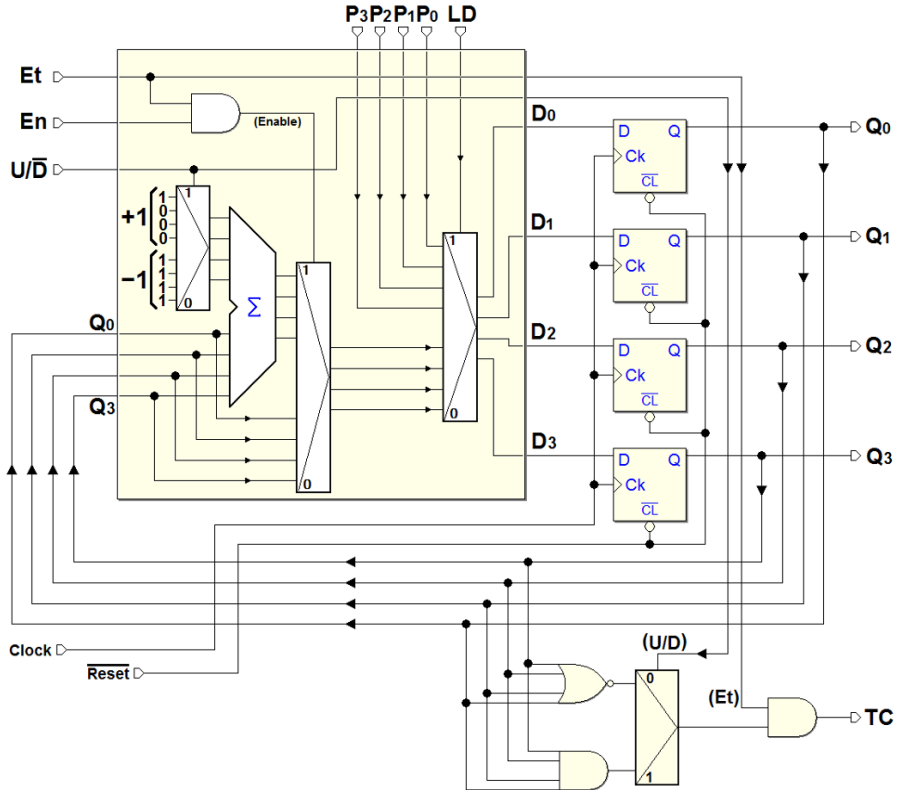
6.3.4 Contatori “universali”

Il contatore più completo, definito “universale”, aggiunge la possibilità di *pre-impostare* il numero contenuto del contatore, come in un registro parallelo, oltre ad una migliorata gestione della abilitazione e dell'uscita TC .

Nella figura che segue vediamo un esempio di contatore universale, il “Cnt4” (“Counter 4-bits”), presente nella libreria di *Deeds*:



Il contatore possiede gli ingressi per la *pre-impostazione* $P3..P0$, in numero pari a quello delle uscite $Q3..Q0$, e il relativo comando di caricamento LD (*Load*). Come si osserva nella figura seguente, che descrive a blocchi il contatore universale, il caricamento avviene in modo sincrono sul fronte di salita del clock, quando LD è impostato a 1:



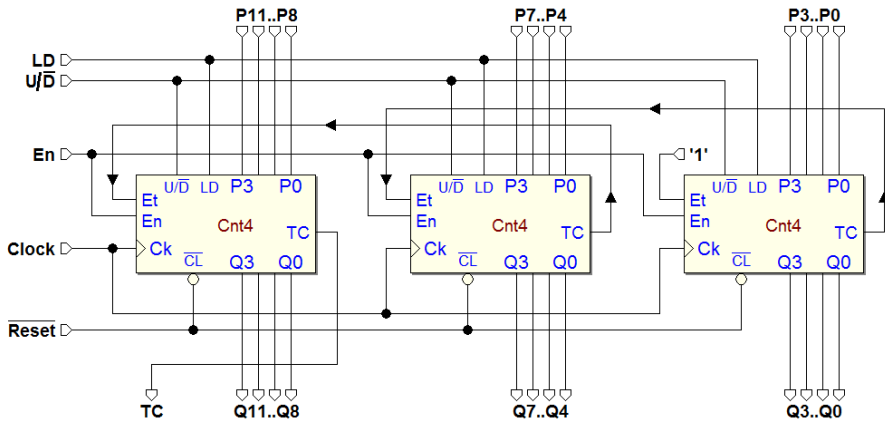
Infatti, alla struttura dei precedenti contatori è stato aggiunto un ulteriore selettore, controllato dall'ingresso LD . Se questo è a 1, il numero impostato sugli ingressi $P3..P0$ è instradato verso i D dei flip-flop. Se $LD = 0$, invece, il funzionamento del contatore è riconducibile a quello dei tipi precedenti. Ad esempio, la funzionalità dell'ingresso di direzione U/\bar{D} è identica.

Gli ingressi di abilitazione ora sono due: oltre al già noto En , abbiamo anche Et ("Enable Terminal Count"). L'abilitazione al conteggio richiede che entrambi En e Et siano attivati. Dato che Et abilita la generazione dell'uscita TC , è usato separatamente da En quando si mettono più contatori in cascata, come esaminato poco più avanti.

Ricordiamo che in tutti i contatori visti in precedenza, la funzionalità del TC non si poteva disabilitare e la sua generazione dipendeva soltanto dalla direzione del conteggio e dal valore presente sulle uscite.

Estensione dei contatori

La struttura sincrona del contatore universale si presta all'*estensione del numero di bit* utilizzando più dispositivi collegati tra di loro in "cascata", come si vede nella figura seguente, dove si è ottenuto un unico contatore da 12 bit utilizzando 3 da 4:



Come si vede, i tre dispositivi condividono il segnale di clock, per cui l'intera struttura è *sincrona*. Inoltre anche gli ingressi di $U/Reset$ e di U/D sono condivisi, per cui i tre contatori saranno inizializzati insieme e saranno sempre impostati nella stessa direzione di conteggio; anche l'abilitazione En è comune, così come l'ingresso di pre-impostazione LD . Il contatore più a destra in figura è utilizzato per i bit meno significativi ($Q3..Q0$), quello più a sinistra per i più significativi ($Q11..Q8$).

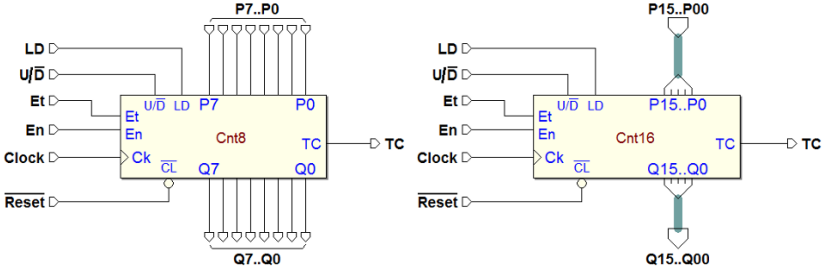
Supponiamo ora che l'abilitazione complessiva En sia impostata a 1, e la direzione in avanti. Il contatore che genera $Q3..Q0$ risulterà abilitato, poiché anche il suo ingresso Et è a 1. Per fare in modo che il contatore di mezzo, che produce $Q7..Q4$, conti solo quando dovuto, colleghiamo il suo ingresso Et al TC del contatore che genera $Q3..Q0$.

In questo modo, TC è utilizzato nel significato di "riporto": quando il contatore che più a destra è arrivato al massimo, ordina a quello intermedio di incrementare di una unità, abilitandolo con $Et = 1$.

Un collegamento analogo è presente tra il TC del contatore in mezzo e l'ingresso Et di quello più a sinistra. Il TC complessivo, quello prodotto dal contatore più a sinistra, risulterà attivo solo quando *tutti e tre* i contatori avranno raggiunto 1111.

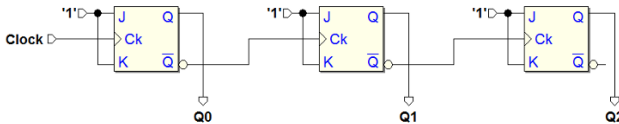
La struttura è estendibile ad un numero maggiore di bit, ma occorre tenere presente che, per ogni contatore che aggiungiamo, aumenta il ritardo di propagazione complessivo della rete combinatoria che propaga i segnali di abilitazione Et attraverso i TC dei vari componenti.

In tutta la trattazione abbiamo utilizzato contatori a 4 bit solo per semplicità. Ovviamente, nelle librerie dei sistemi CAD troviamo componenti di varie dimensioni, come ad esempio i contatori a 8 e a 16 bit visibili nella figura qui sotto. Il primo è riportato nella versione normale (a sinistra), il secondo nella versione con piedini di tipo *bus* (a destra):



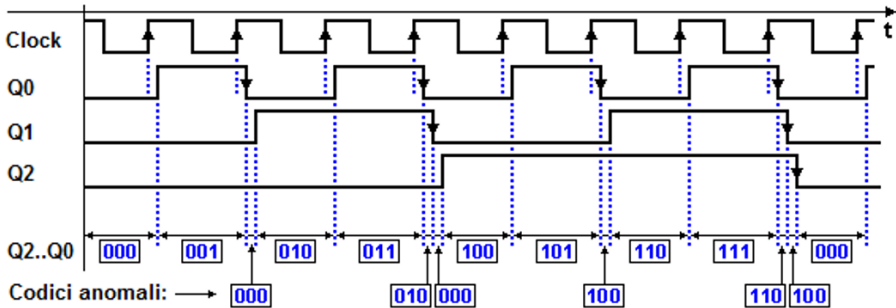
6.3.5 Contatori asincroni

Un contatore si dice *asincrono* quando i flip-flop di cui è composto *non condividono* tutti lo stesso segnale di clock. La rete nella figura seguente rappresenta un contatore asincrono binario avanti, ove ogni flip-flop JK, a parte il primo, utilizza come segnale di clock l'uscita \bar{Q} di quello che lo precede.



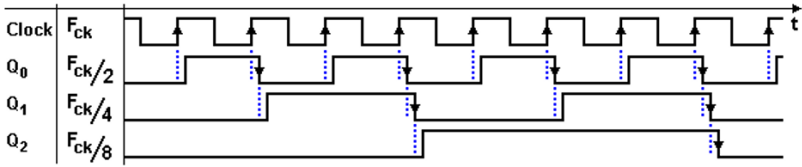
In questa rete ogni flip-flop (attivo sul fronte di salita del *proprio* ingresso di clock), cambia stato quando l'uscita Q del precedente presenta una transizione *da uno a zero*. A causa di questo comportamento, questo contatore è denominato in inglese “*ripple counter*”, con un termine che lo paragona alla propagazione di un'onda.

L'andamento delle sue uscite è descritto nella figura qui sotto, dove sono stati messi in evidenza (in modo indicativo) i tempi di propagazione dei flip-flop. Si noti che il ritardo tra il clock in ingresso al contatore e una data uscita cresce *proporzionalmente* alla posizione dell'uscita stessa:



Se trascuriamo i ritardi, questo contatore segue una *sequenza binaria crescente*, ma la asincronicità di commutazione genera codici anomali che ne alterano la sequenza, come evidenziato in figura. Tali codici hanno durata breve (pari al tempo di propagazione dei flip-flop), ma possono creare problemi in una rete che ne elabori le uscite, ad esempio una rete di decodifica. Per questo, i contatori asincroni sono utilizzati solo in casi particolari.

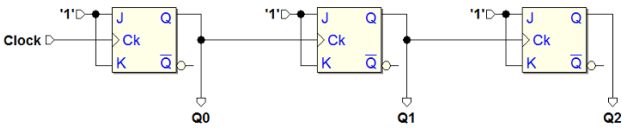
Come si è già visto, un contatore può essere considerato come *divisore di frequenza*, quando si consideri la relazione temporale tra le forme d'onda del clock e di una qualunque delle uscite. Un contatore asincrono può essere vantaggiosamente impiegato a questo scopo: come si vede nella figura seguente, la frequenza dell'uscita Q_0 è pari ad $1/2$ di quella del Clock, e le uscite Q_1 e Q_2 forniscono un segnale di frequenza rispettivamente di $1/4$ e $1/8$:



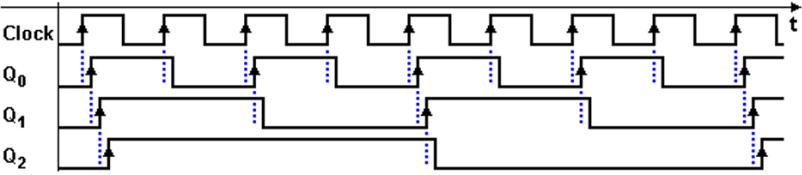
Nell'utilizzo del contatore come divisore di frequenza, l'asincronicità delle uscite non rappresenta un problema, in quanto i segnali generati sono sovente utilizzati in modo indipendente tra di loro. Inoltre, la semplicità del contatore asincrono, per "rapporti di divisione" di frequenza particolarmente alti, rappresenta un notevole vantaggio rispetto ad una realizzazione sincrona.

I divisori di frequenza trovano applicazione negli apparati di telecomunicazione, ove sono utilizzati per generare segnali la cui frequenza è un sottomultiplo di quella di un generatore di clock che lavora a frequenza più elevata.

Per completezza, rappresentiamo nella prossima figura un contatore asincrono *indietro*. In questa rete, l'ingresso Ck di ogni flip-flop, a parte il primo, è collegato all'uscita Q del precedente:



Nel diagramma temporale qui sotto, si osserva che, nel conteggio all'indietro, il cambiamento dell'uscita di un flop-flop avviene quando il precedente effettua una transizione *da zero a uno*.



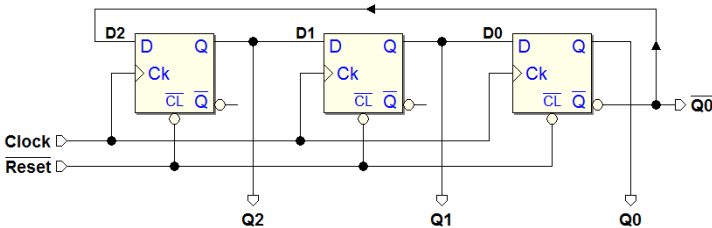
6.4 Analisi di reti

Un passo essenziale nel percorso verso la progettazione di sistemi digitali è la comprensione del comportamento di una rete sequenziale data. L'analisi temporale di una rete, di cui è dato lo schema, consente di acquisire dimestichezza con alcuni aspetti generali dell'interazione tra i componenti combinatori e sequenziali di un circuito logico. Questa familiarità con le dinamiche e il comportamento *a basso livello* delle reti sequenziali è un passo importante nel rendere il progettista consapevole nel percorso del suo lavoro, dalle specifiche di progetto al prodotto finale.

Nel seguito presentiamo una serie di esempi di analisi, mediante *diagrammi temporali*, di semplici reti sequenziali di cui sarà dato lo schema logico, associato ad opportune sequenze dei segnali di ingresso. Eseguiremo un esame *funzionale* delle reti: in altre parole studieremo l'andamento delle uscite in dipendenza dagli ingressi.

6.4.1 Esempio n. 1

In questo paragrafo ci poniamo l'obiettivo di analizzare il funzionamento della rete di flip-flop rappresentata nella figura seguente. La semplice osservazione dello schema fornisce indicazioni utili per la sua analisi. Si tratta di una rete sincrona composta da flip-flop di tipo D-PET, provvista dell'ingresso di inizializzazione asincrona *Reset* (che agisce sugli ingressi di *Clear* dei flip-flop). La rete genera le tre uscite Q_2 , Q_1 e Q_0 :

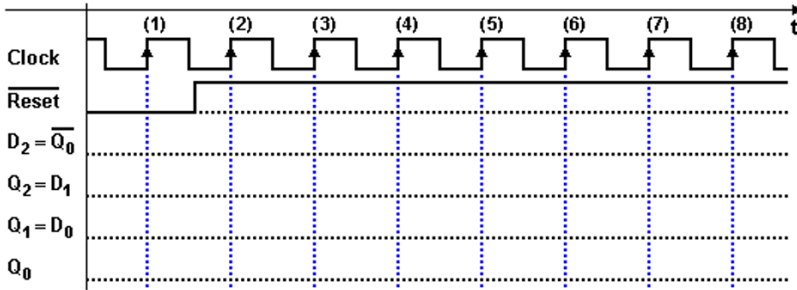


Si può anche facilmente identificare la struttura della rete: un registro a scorrimento, in cui l'ingresso seriale D_2 è stato collegato all'uscita negata $\overline{Q_0}$ dell'ultimo flip-flop. L'identificazione della particolare struttura, tuttavia, non è necessaria ai fini dell'analisi: le procedure che presentiamo sono valide per qualunque rete sincrona di flip-flop.

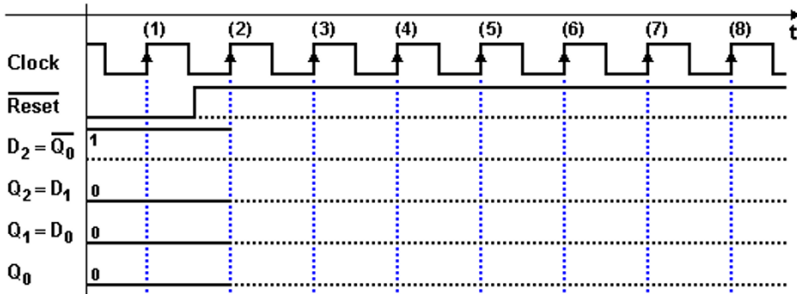
Per cominciare, è necessario predisporre un *diagramma temporale* nel quale rappresentare l'andamento del clock, degli ingressi e delle uscite della rete. Nel caso in esame, per comodità di analisi, è utile inserire nel diagramma anche il segnale $D_2 (= \overline{Q_0})$.

La figura seguente mostra la traccia del diagramma temporale da costruire, nella quale è rappresentato il *Clock* ed è stato impostato il segnale di inizia-

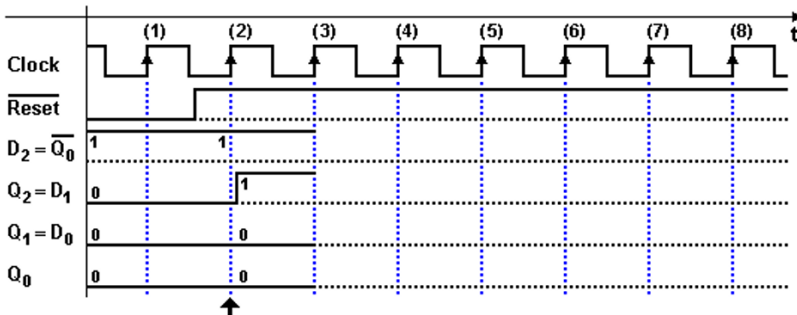
lizzazione \overline{Reset} , attivo all'origine del diagramma e disattivato nell'intervallo tra i fronti (1) e (2) del $Clock$:



Fin tanto che l'ingresso \overline{Reset} è mantenuto attivo (basso), le uscite Q dei flip-flop sono *forzate* a zero, ed il fronte (1) del $Clock$ non può produrre cambiamenti. Si faccia attenzione che dobbiamo disegnare D_2 al valore 1, in questa fase di inizializzazione, in quanto è collegato all'uscita negata di Q_0 . Ricordiamoci che la disattivazione del \overline{Reset} non altera lo stato della rete, come si vede nel tracciato qui sotto, e i segnali rimangono inalterati fino al fronte (2):

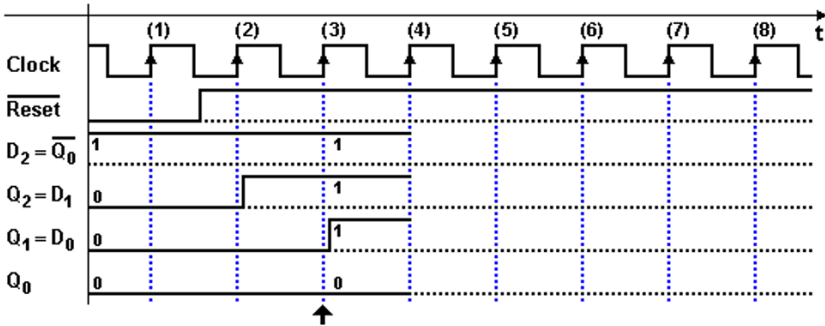


All'occorrenza di ogni fronte attivo del $Clock$, i flip-flop D-PET trasferiscono sulla uscita Q il valore logico presente in quell'istante sul proprio ingresso D . Sul fronte (2), nella figura precedente, gli ingressi D_2 , D_1 e D_0 dei flip-flop valgono rispettivamente 1, 0 e 0. Nella figura seguente, quindi, andiamo a rappresentare le uscite dei flip-flop dopo il fronte (2):

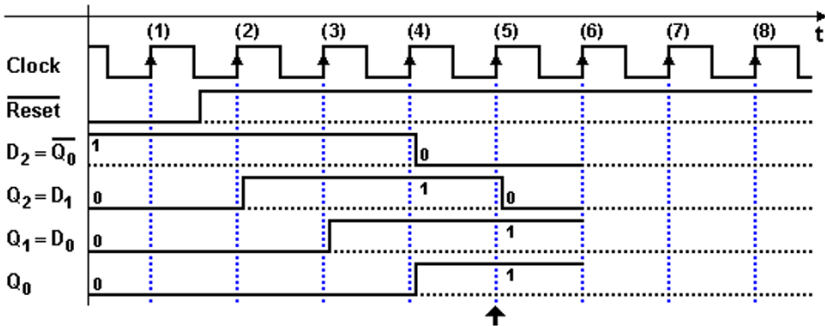
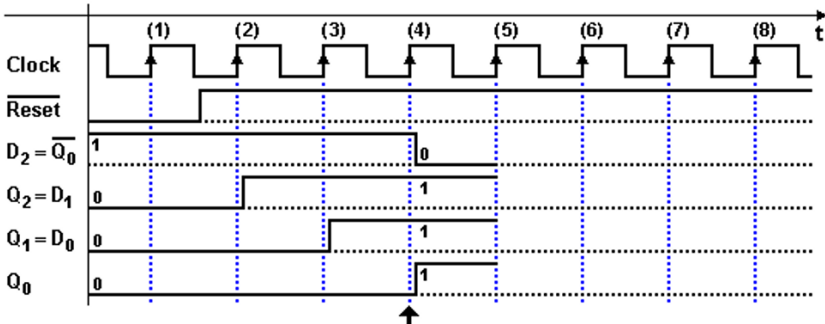


Nel tracciato della figura precedente si è preferito evidenziare, in modo qualitativo, il ritardo di propagazione tra il fronte del *Clock* e il cambiamento della uscita Q_2 . Si noti che il nuovo valore assunto da Q_2 (e quindi da D_1) verrà acquisito dal flip-flop Q_1 sul fronte successivo (3).

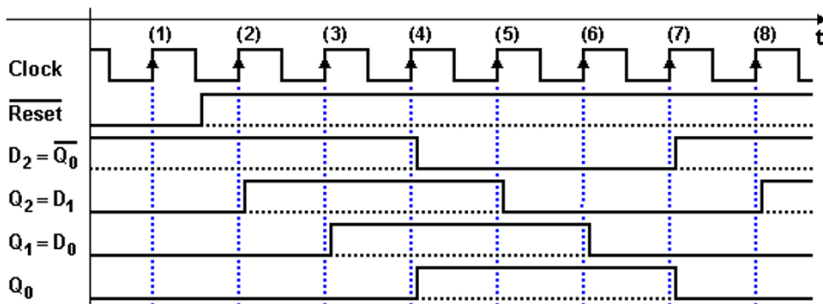
Sino al fronte (3) la situazione rimane invariata, dato che i flip-flop cambiano stato solo sul fronte di salita del *Clock*. Sul fronte (3), in modo analogo al fronte (2), i valori sugli ingressi D sono trasferiti alle uscite Q . Nella figura qui sotto abbiamo disegnato la situazione dopo il fronte (3), prestando attenzione a indicare qualitativamente un po' di ritardo nella attivazione di Q_1 :



Nei due diagrammi che seguono, si vede come continuare a tracciare il diagramma, in relazione ai fronti (4) e (5), applicando gli stessi criteri:



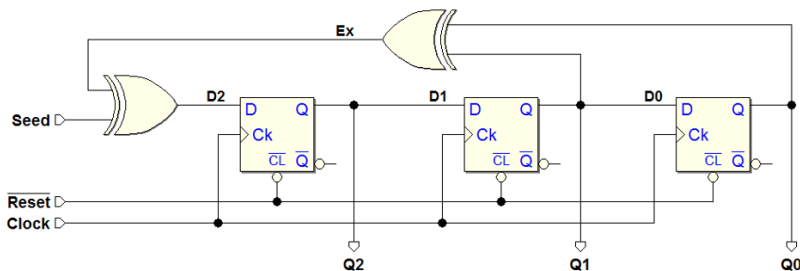
Nella figura seguente, infine, vediamo completato il diagramma temporale: è evidente nella figura il comportamento tipico del *registro a scorrimento*:



La rete che abbiamo esaminato è un “*Contatore Johnson*” a tre bit. Si può realizzare con un registro a scorrimento di un numero qualsiasi di bit, collegando l’uscita negata dell’ultimo flip-flop all’ingresso del primo. Presenta il vantaggio di una struttura semplice, a cui corrisponde lo svantaggio di un codice di conteggio non in binario puro (ma facilmente decodificabile).

6.4.2 Esempio n. 2

La rete nella figura seguente è composta di tre flip-flop D-PET, con i segnali di *Clock* e di *Reset* in comune: è facile identificare nella rete la struttura base del registro a scorrimento, ma l’ingresso del flip-flop più a sinistra, *D2*, è ottenuto da una rete di EXOR che elabora le uscite *Q1*, *Q0* e l’ingresso *Seed*:

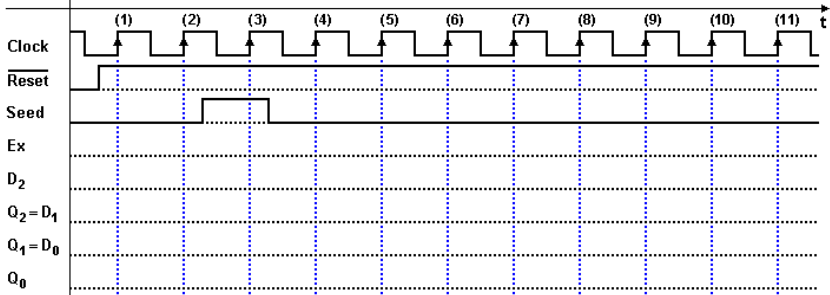


Sappiamo che, ad ogni fronte attivo del *Clock*, i flip-flop trasferiscono sulla uscita *Q* il valore logico presente in quell’istante sull’ingresso *D*: l’analisi della rete consiste quindi nel valutare gli ingressi *D2*, *D1* e *D0* in corrispondenza di tali fronti. Qui di seguito, dall’esame della figura precedente, scriviamo le relative espressioni booleane. Si noti che soltanto la prima di esse rappresenta una rete di porte logiche, mentre le altre descrivono semplici collegamenti:

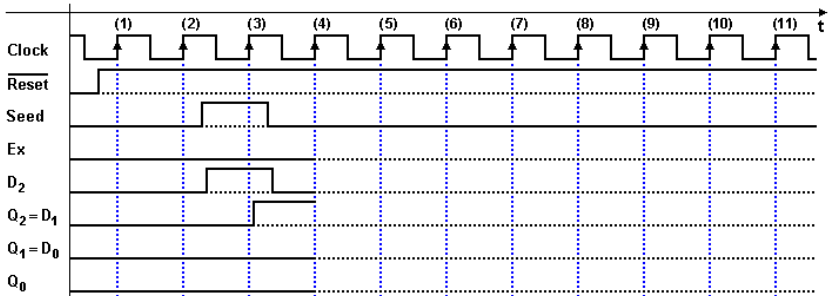
$$D2 = Seed \oplus Q1 \oplus Q0; \quad D1 = Q2; \quad D0 = Q1$$

Queste tre espressioni, in base ai valori dell’ingresso *Seed* e delle uscite dei flip-flop *Q2*, *Q1* e *Q0* (lo *stato attuale* della rete), forniscono i valori degli

ingressi D_2 , D_1 e D_0 che, una volta caricati nei flip-flop, costituiranno lo *stato successivo* della rete. Anche in questo esempio, tracciamo il comportamento della rete in un diagramma temporale, che impostiamo nella figura seguente. Prendiamo i tracciati per tutti gli ingressi e le uscite della rete, gli ingressi dei flip-flop e, per comodità di analisi, anche del segnale intermedio Ex :



La figura seguente presenta l'analisi temporale fino al fronte (4) escluso. Il \overline{Reset} , attivo prima del fronte attivo (1) del $Clock$, impone a 0 le uscite dei flip-flop. Anche gli ingressi D_1 e D_0 risultano a 0, essendo collegati alle uscite Q_2 e Q_1 . D_2 è a 0, come si vede dalla sua espressione booleana, essendo l'ingresso esterno $Seed$ impostato a 0. Il fronte del $Clock$ (1), quindi, non modifica le uscite, che rimangono a 0. Stesso dicasi per il fronte (2):

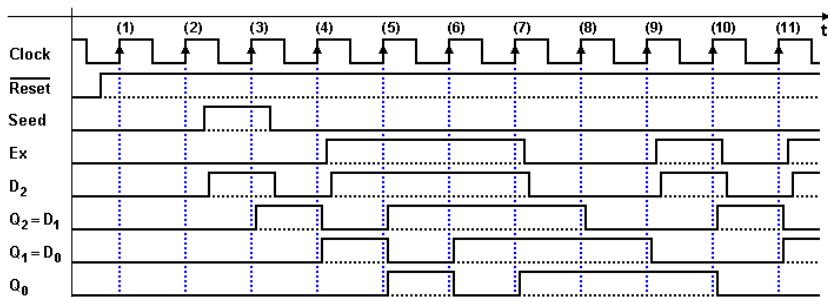


Nell'impostazione iniziale del diagramma si è supposto che l'esterno attivi l'ingresso $Seed$ per la durata un ciclo del $Clock$, tra i fronti (2) e (3). La conseguenza immediata è l'attivazione di D_2 . Sul fronte (3) l'uscita Q_2 passa a 1, mentre gli altri flip-flop non cambiano le uscite (infatti $D_1 = D_0 = 0$).

Si capisce come l'attivazione di $Seed$ (“*Seme*”) sia necessaria per far evolvere la rete, che altrimenti rimarrebbe indefinitamente nella situazione impostata dalla attivazione del \overline{Reset} . Dopo il fronte (3), l'esterno riporta l'ingresso $Seed$ a 0, per cui la valutazione di D_2 si semplifica, venendo ora a dipendere soltanto dalle variazioni delle uscite Q_1 e Q_0 .

Proseguendo nell'analisi, dopo l'azzeramento di $Seed$, le reti che generano D_2 , D_1 e D_0 producono rispettivamente i valori 0, 1 e 0, che sono trasferiti sulle

uscite dei flip-flop al fronte (4). Con lo stesso metodo, proseguiamo l'analisi fino a completare il diagramma, che riportiamo qui:

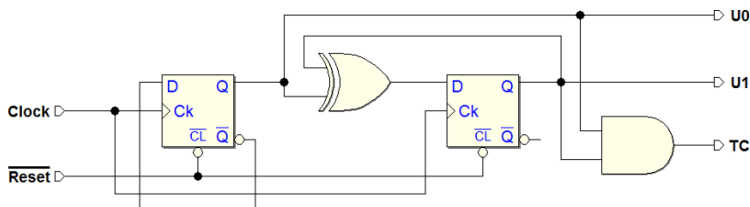


Il diagramma completo si presta ad alcune osservazioni di carattere generale. Tutte le uscite dei flip-flop commutano in corrispondenza dei fronti attivi del *Clock*, con un ritardo pari al tempo di propagazione. I segnali *Ex* e *D2* presentano un ulteriore ritardo, dovuto alla rete combinatoria che li genera. Come si osserva nel diagramma temporale, tra i fronti (2) e (4), il segnale *D2* può variare in modo asincrono rispetto al *Clock*, a causa della sua dipendenza dall'ingresso esterno *Seed*.

Per informazione, la rete che abbiamo analizzato è un esempio, ridotto, di *generatore di numeri pseudo-casuali*. Il numero generato, nel nostro caso, è composto dalle uscite *Q2*, *Q1* e *Q0*. Un generatore pseudo-casuale è normalmente realizzato con un numero elevato di flip-flop (per es. 32), perché la sequenza generata è solo *apparentemente* casuale, ma in realtà si ripete ciclicamente.

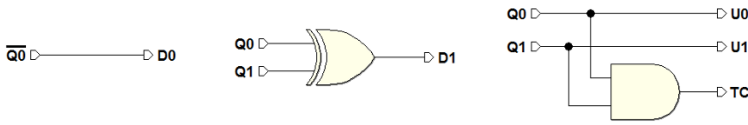
6.4.3 Esempio n. 3

La rete di questo esempio impiega due flip-flop D-PET, con i segnali di *Clock* e di *Reset* in comune, e non presenta ingressi di comando. Le uscite *U0* e *U1* sono prelevate direttamente dalle uscite *Q0* e *Q1* dei flip-flop, mentre *TC* è ottenuta attraverso una porta logica:



Applichiamo la stessa procedura di analisi impiegata in precedenza: si tratta di valutare *D0* e *D1* in corrispondenza del fronte attivo del *Clock*, sapendo che, all'arrivo di quel fronte, saranno caricati nei flip-flop. Può essere utile “*separare*” dallo schema complessivo le reti combinatorie che producono *D0*, *D1* e le uscite della rete *U0*, *U2* e *TC*.

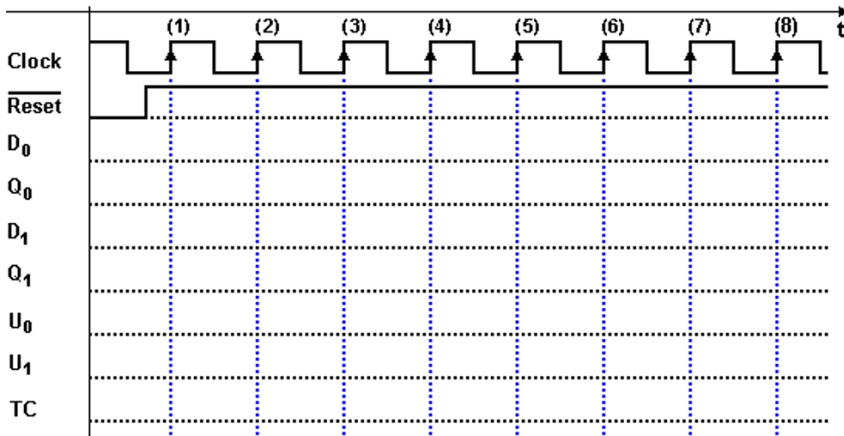
Le ridisegniamo a parte in forma circuitale, quindi, oppure (o in aggiunta) le esprimiamo come espressioni booleane:



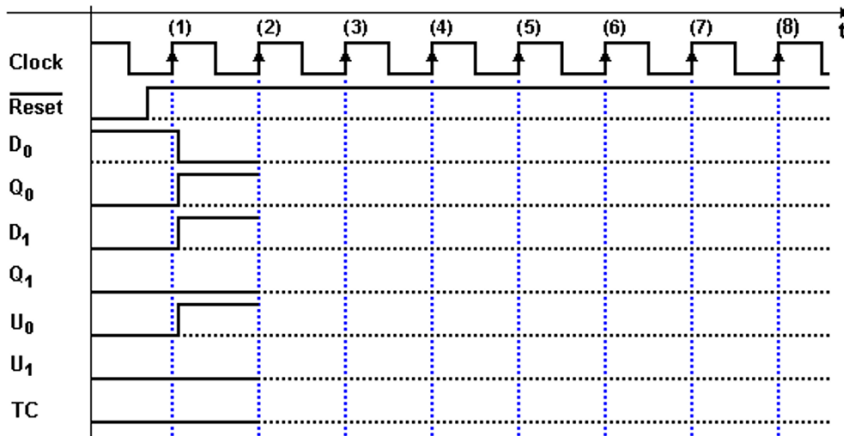
$$D0 = \overline{Q0}; \quad D1 = Q0 \oplus Q1;$$

$$U0 = Q0; \quad U1 = Q1; \quad TC = Q0 \cdot Q1$$

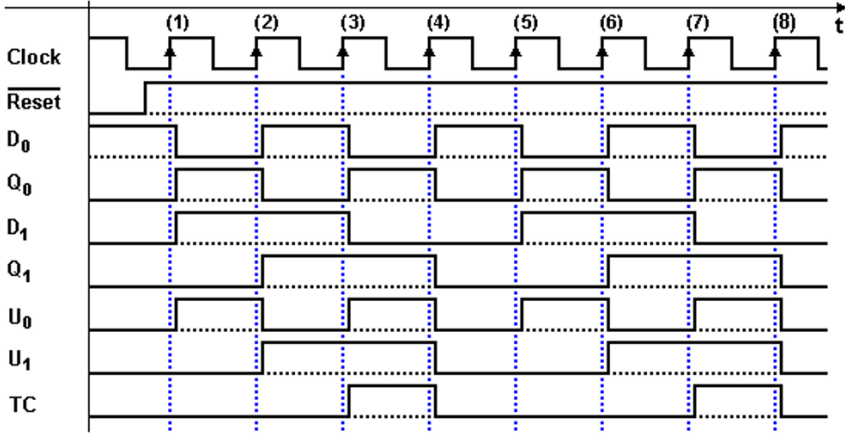
Utilizzando le figure o le espressioni descritte tracciamo più agevolmente il diagramma temporale. Impostiamo nel diagramma, oltre al *Clock*, il segnale di *Reset*, attivo dall'inizio e disattivato poco prima del fronte (1) del *Clock*:



Come primo passo, consideriamo il \overline{Reset} , che all'inizio forza i flip-flop a 0 e poi viene rimosso. Lo stato della rete cambia sul fronte (1) del *Clock*, quando i due flip-flop acquisiscono i valori di $D0$ e $D1$:



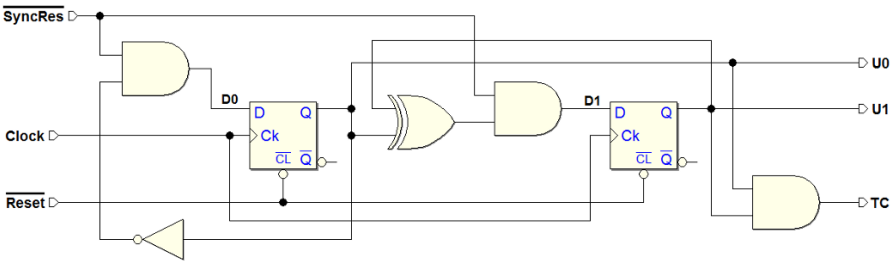
Suggeriamo ora di proseguire autonomamente nell'analisi, per esercizio, secondo i criteri fin qui suggeriti. Il diagramma temporale risulterà come riportato nella figura seguente, dove tra l'altro è evidente la *ciclicità* della sequenza, che si ripete ogni 4 fronti del *Clock*.



Il dispositivo esaminato si comporta come *contatore*, infatti i valori assunti dalle uscite *U1* (MSB) e *U0* (LSB) seguono un ordinamento *binario naturale in avanti, modulo 4*; *TC* segnala che le uscite hanno raggiunto il valore massimo.

6.4.4 Esempio n. 4

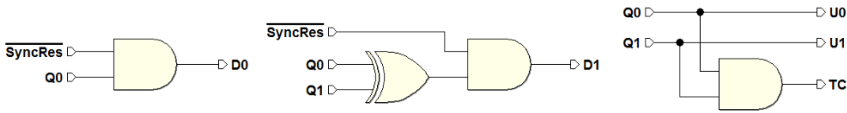
La rete nella figura seguente è molto simile a quella dell'esempio precedente: impiega due flip-flop D-PET, ha le stesse uscite *U0*, *U1* e *TC*, ma presenta in più l'ingresso di comando *SyncRes* ("Synchronous Reset") e la logica associata a questo. Identica al caso precedente, la rete del *Clock* e del *Reset*:



Qualche considerazione intuitiva può aiutare nell'analisi della rete: osserviamo che, se l'ingresso *SyncRes* vale 1, i due AND condizionati da questo segnale trasmettono alle rispettive uscite il valore dell'altro loro ingresso, e la rete risulta funzionalmente identica a quella dell'esercizio precedente.

Se $\overline{SyncRes}$ vale 0, entrambi gli AND hanno le uscite a 0, imponendo *D0* e *D1* a 0 e, quindi, l'azzeramento *sincrono* dei due flip-flop.

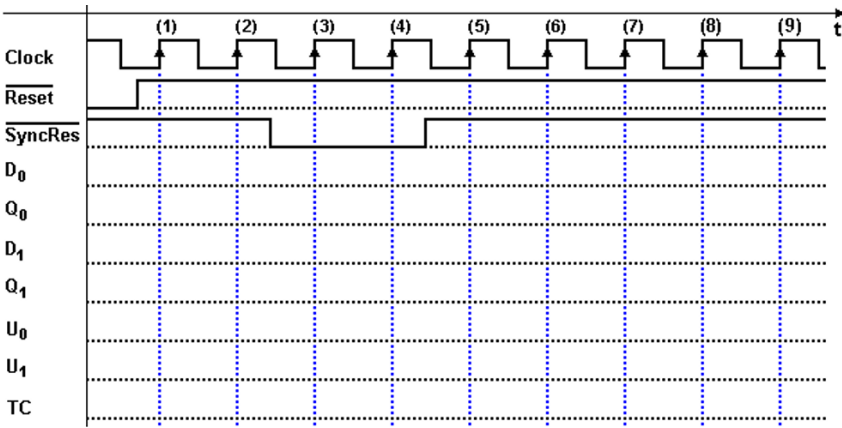
Ricaviamo le reti combinatorie che producono D_0 e D_1 , e/o le espressioni. La rete di U_0 , U_1 e TC è identica a quella del precedente esercizio:



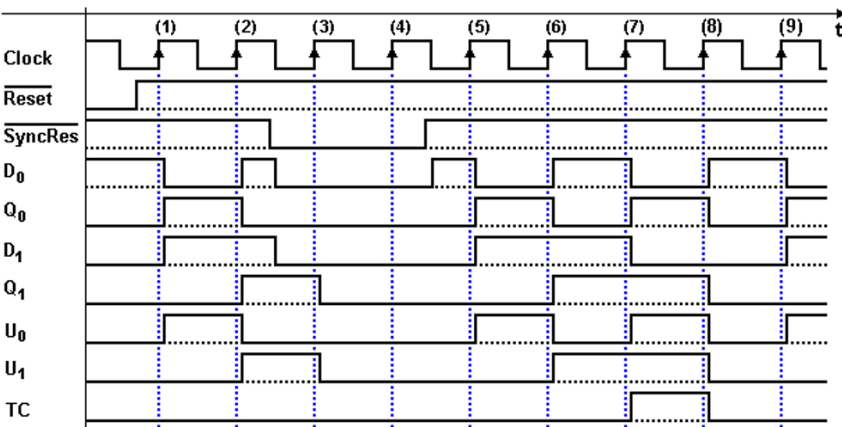
$$D_0 = \overline{SyncRes} \cdot \overline{Q_0}; \quad D_1 = \overline{SyncRes} \cdot (Q_0 \oplus Q_1);$$

$$U_0 = Q_0; \quad U_1 = Q_1; \quad TC = Q_0 \cdot Q_1;$$

Predisponiamo il diagramma temporale come nella figura seguente, con i segnali di $Clock$ e di \overline{Reset} impostati come nel precedente esercizio. Ipotizziamo che il comando $\overline{SyncRes}$ venga attivato per due cicli di clock, come disegnato:

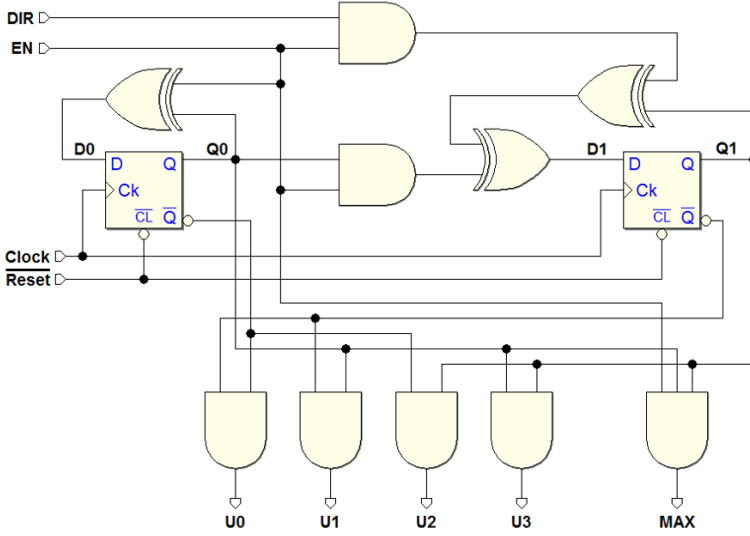


Lasciamo al lettore il compito di completare il tracciato temporale, che risulterà come nella figura qui sotto. Si faccia attenzione ai segnali D_0 e D_1 che, come si può osservare, rispondono immediatamente (a parte i ritardi di propagazione) alle variazioni del comando $\overline{SyncRes}$:

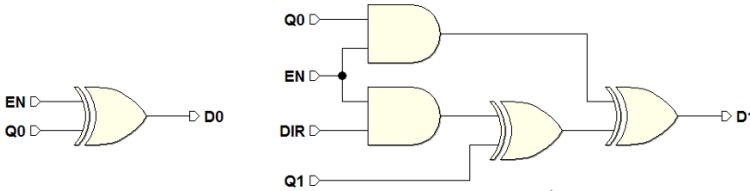


6.4.5 Esempio n. 5

Nella figura seguente è riportata una rete che impiega due flip-flop D-PET che condividono lo stesso *Clock*. Entrambi i flip-flop sono connessi all'ingresso di inizializzazione asincrona *Reset*. Sono presenti due ingressi *EN* e *DIR*; le uscite generate dalle rete sono denominate *U0*, *U1*, *U2*, *U3* e *MAX*.



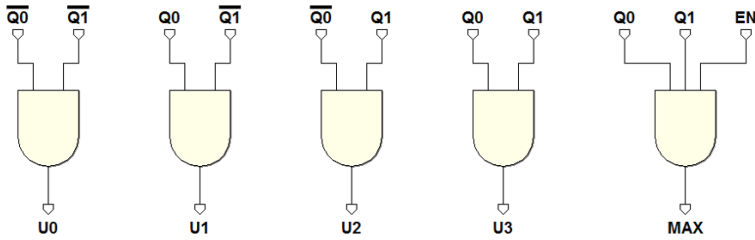
Come nei casi precedenti, un passo fondamentale per l'analisi della rete è la valutazione di *D0* e *D1* in corrispondenza dei fronti attivi del *Clock*. Estraiamo dal disegno le reti combinatorie che producono *D0* e *D1*, e/o le esprimiamo in termini di espressioni booleane:



$$D0 = EN \oplus Q0; \quad D1 = (EN \cdot Q0) \oplus (Q1 \oplus (EN \cdot DIR))$$

Queste reti combinano il valore degli ingressi *EN* e *DIR* con il valore delle uscite *Q0* e *Q1* dei flip-flop (lo “*stato*” della rete), e producono un nuovo valore agli ingressi *D0* e *D1* degli stessi. Questi valori saranno caricati all'arrivo del fronte attivo del *Clock*, e costituiranno lo “*stato successivo*” della rete.

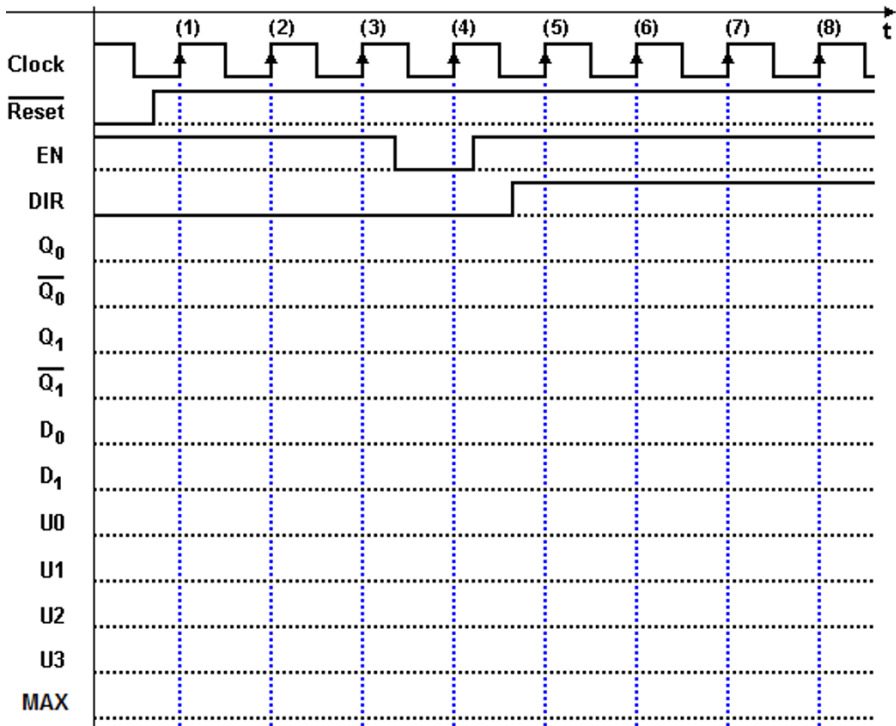
Le uscite della rete *U0*, *U1*, *U2*, *U3* e *MAX* sono funzioni combinatorie delle uscite dei flip-flop e dell'ingresso *EN*, come evidenziato qui di seguito, sia come schema della rete, sia in termini di espressioni booleane.



$$U_0 = \overline{Q_0} \cdot \overline{Q_1}; \quad U_1 = Q_0 \cdot \overline{Q_1}; \quad U_2 = \overline{Q_0} \cdot Q_1; \quad U_3 = Q_0 \cdot Q_1;$$

$$MAX = Q_0 \cdot Q_1 \cdot EN$$

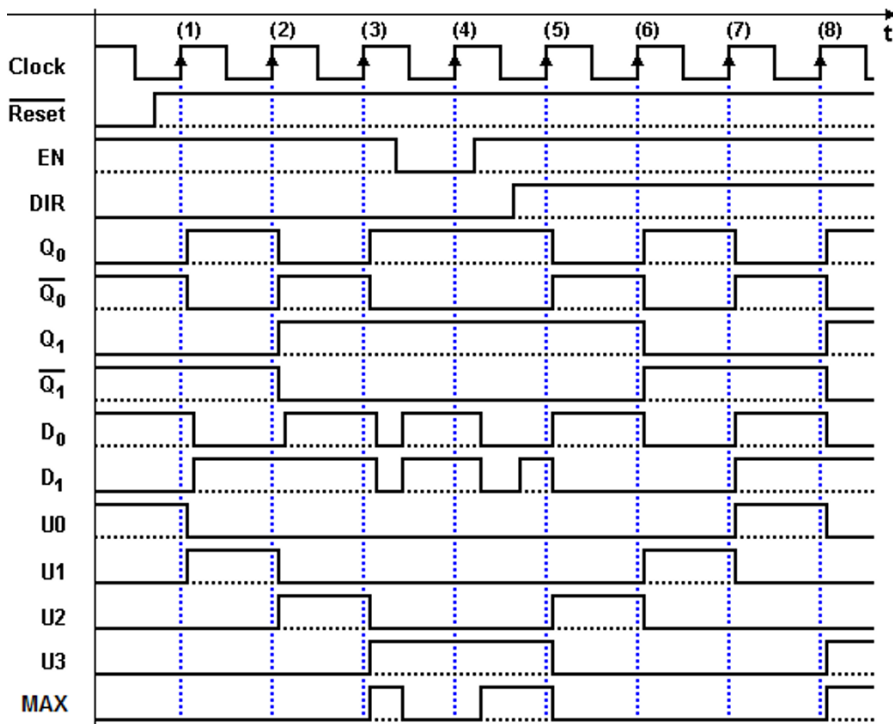
Ai fini della analisi temporale, è bene che l'andamento degli ingressi *EN* e *DIR* sia impostato in modo ragionato, per non ottenere un tracciato poco rappresentativo. Impostiamo gli ingressi come visibile nella figura qui sotto, in modo da evidenziarne la funzione, in modo che sia chiara alla conclusione del processo di analisi. Si è scelto di includere nel diagramma anche gli ingressi e le uscite dei flip-flop, per facilitare il lavoro di analisi:



I criteri di analisi visti in precedenza sono sufficienti per analizzare la rete: si suggerisce al lettore di eseguire in prima persona l'analisi e, allo scopo, qui di seguito proponiamo alcuni consigli.

1. È opportuno concentrare inizialmente l'attenzione sulla evoluzione dello stato della rete, per occuparci in un secondo momento della generazione delle uscite. Conviene quindi tracciare prima l'andamento dei segnali Q_0 e Q_1 (diretti e negati) e di D_0 e D_1 .
2. Si assuma che i ritardi di propagazione siano piccoli rispetto al periodo del clock, ma è molto utile rappresentarli ugualmente nel diagramma, sia pure in maniera qualitativa.
3. Si noti che, dopo l'azionamento del \overline{Reset} , la rete inizia ad evolvere con il fronte (1) del $Clock$.
4. Dopo ogni fronte attivo del $Clock$, dobbiamo ricalcolare i valori di D_0 e D_1 , che saranno caricati nei flip-flop al successivo fronte attivo.
5. Tra i fronti (3) e (4), EN cambia. Come risultato, D_0 e D_1 seguono immediatamente questo cambiamento, ed i valori di D_0 e D_1 che saranno trasferiti su Q_0 e Q_1 sono quelli campionati dai flip-flop sul fronte (4).
6. Analogamente, tra i fronti (4) e (5), cambiano sia EN che DIR .
7. Terminata l'analisi dello stato successivo, si tracciano le uscite U_0, U_1, U_2 e U_3 . Tali uscite possono cambiare soltanto in corrispondenza dei fronti del $Clock$; questo non vale per MAX che dipende anche da un ingresso.

Nella figura seguente riportiamo il risultato dell'analisi. Dal diagramma temporale si nota che le uscite Q_0 e Q_1 dei flip-flop seguono, nei primi cicli dopo l'attivazione del \overline{Reset} , una sequenza di conteggio binario avanti.



Quando l'ingresso EN è a 0, la rete ricarica nei due flip-flop i valori già presenti: sul fronte (4), $Q0$ e $Q1$ non cambiano. Dal fronte (5) il conteggio cambia direzione poiché il valore sull'ingresso DIR cambia.

Valutato sulle uscite $Q0$ e $Q1$, il circuito si comporta come un contatore binario bidirezionale. L'ingresso EN agisce come abilitazione al conteggio (se $EN = 1$), mentre DIR imposta la direzione (all'indietro se $DIR = 1$).

Le uscite $U0$, $U1$, $U2$ e $U3$, come si vede nel diagramma temporale, sono attivate, rispettivamente, dalle combinazioni 00, 01, 10 e 11 di $Q1$ e $Q0$. L'uscita MAX decodifica la stessa combinazione di $U3$, ma si attiva soltanto se $EN = 1$, per cui, come si vede tra i fronti (3) e (5), tale uscita va a 0 in modo asincrono rispetto al $Clock$, seguendo l'andamento di EN .

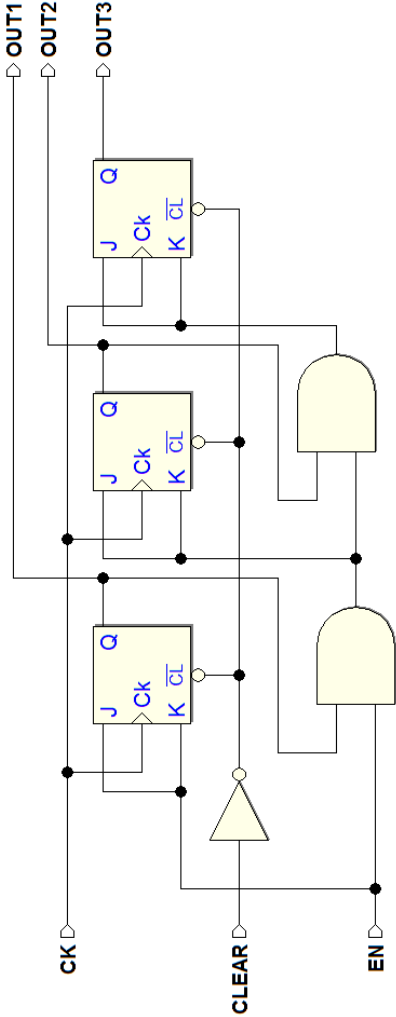
Infine, si noti che $D0$ e $D1$ presentano un andamento asincrono tra i fronti (3) e (5), poiché seguono i cambiamenti degli ingressi EN e DIR , ma ciò che conta è che il loro valore sia stabile al momento di essere acquisito (cioè, in corrispondenza del fronte di salita del $Clock$).

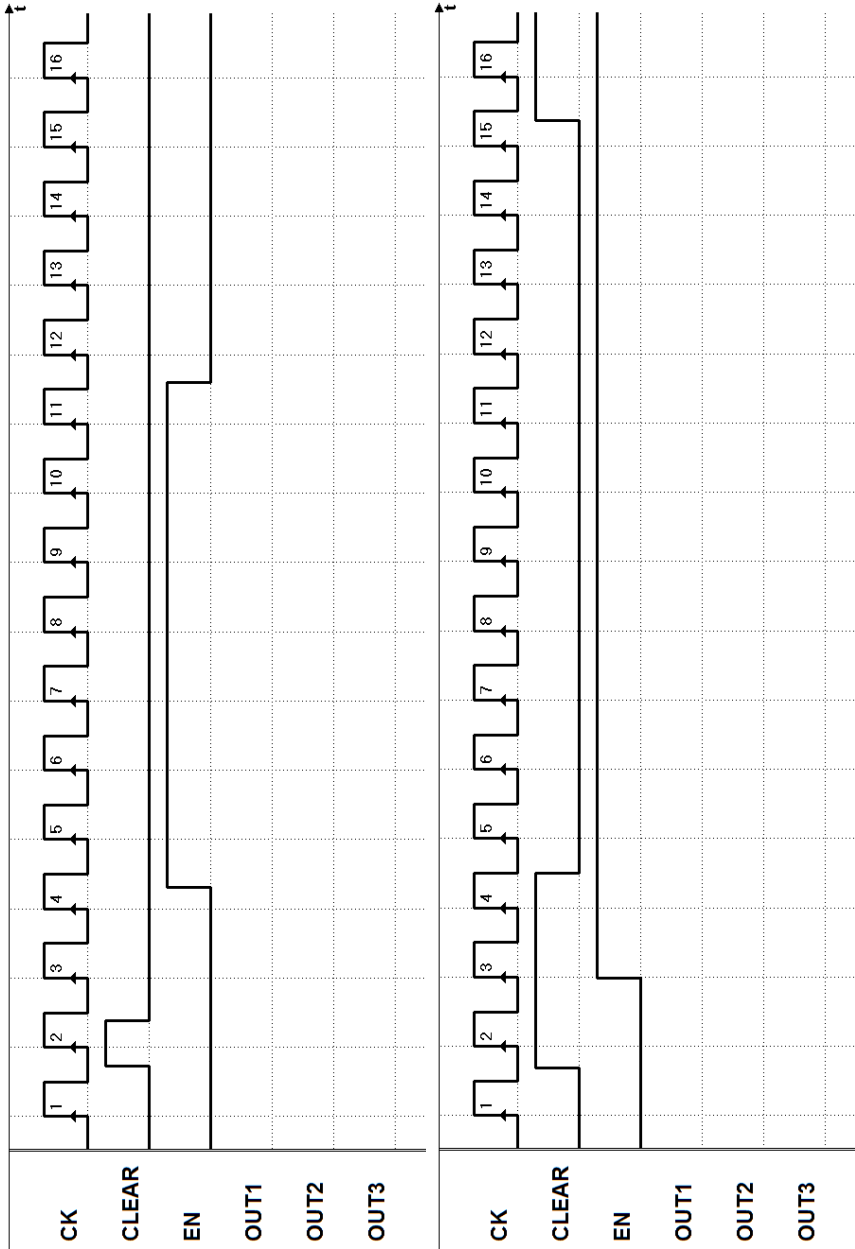
6.5 Esercizi

Analizzare le seguenti reti sequenziali sincrone, completando le tracce temporali proposte a lato. Abbiamo reso disponibili le tracce anche sul sito del simulatore, in formato *PDF*, nelle pagine dei *contenuti digitali* del libro.

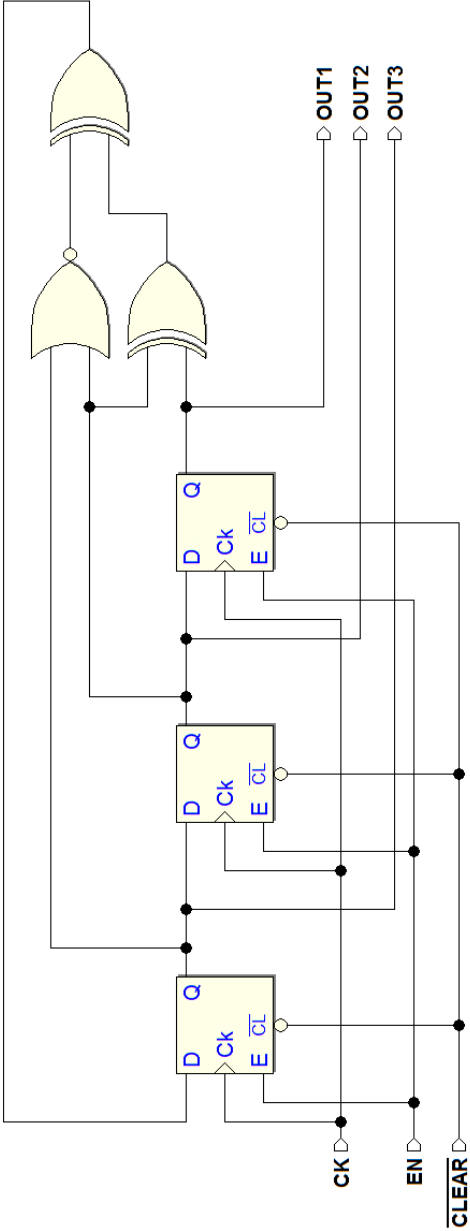
Si suggerisce di completare sulla carta i tracciati *senza* l'ausilio di *Deeds*, che può essere successivamente utilizzato per verificare la propria soluzione (sul sito sono disponibili anche i file delle reti proposte).

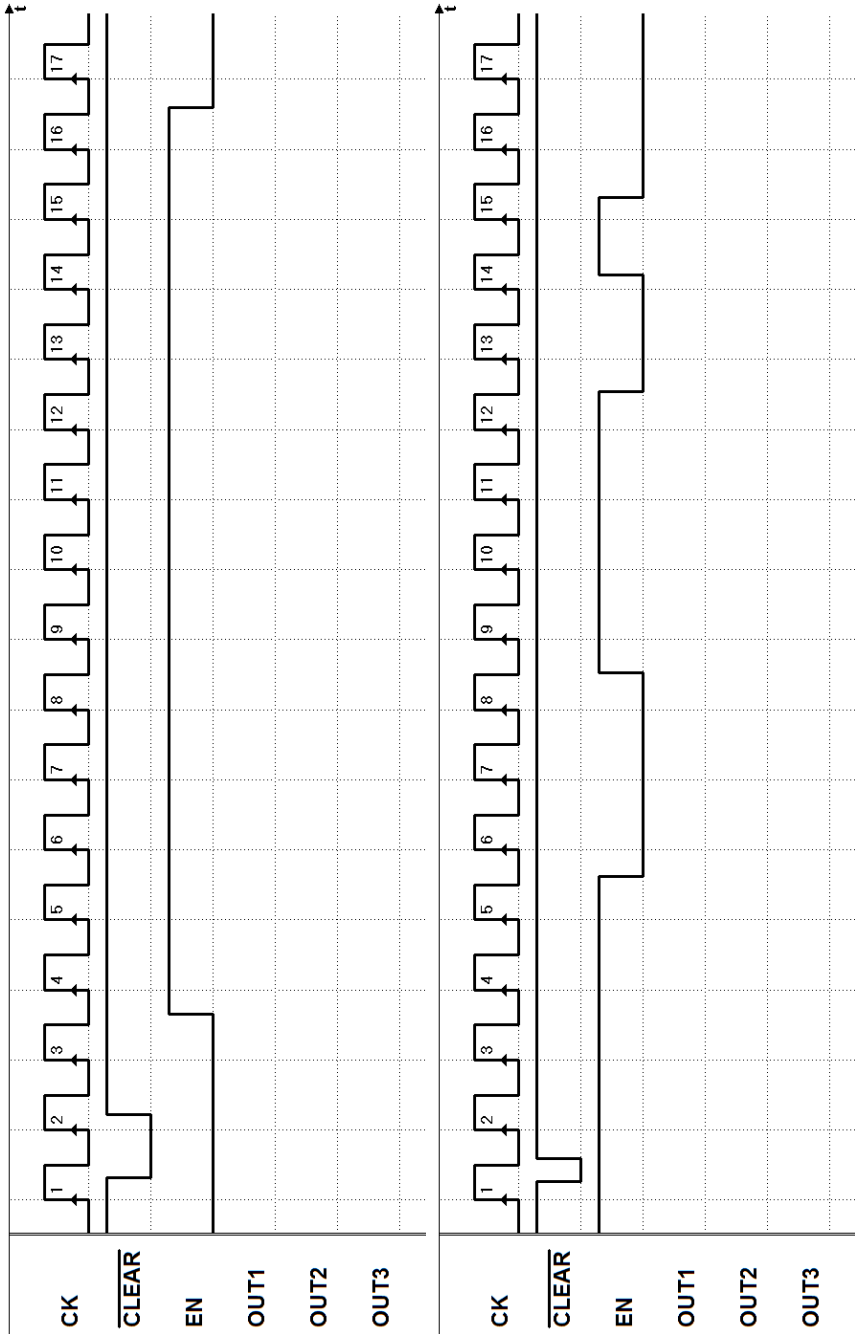
- 1. Esercizio 1 - (diagrammi temporali a fronte)



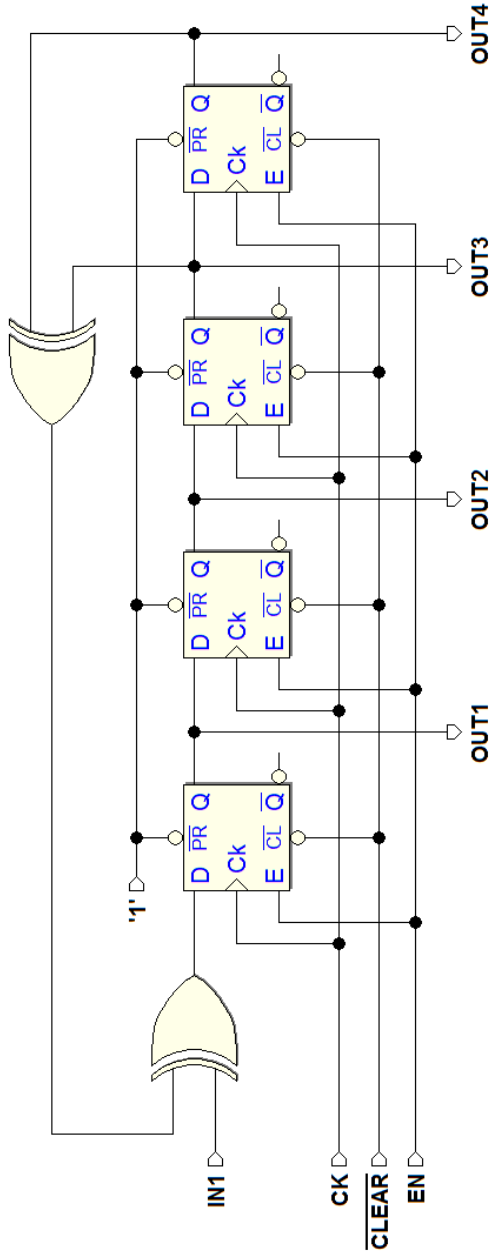


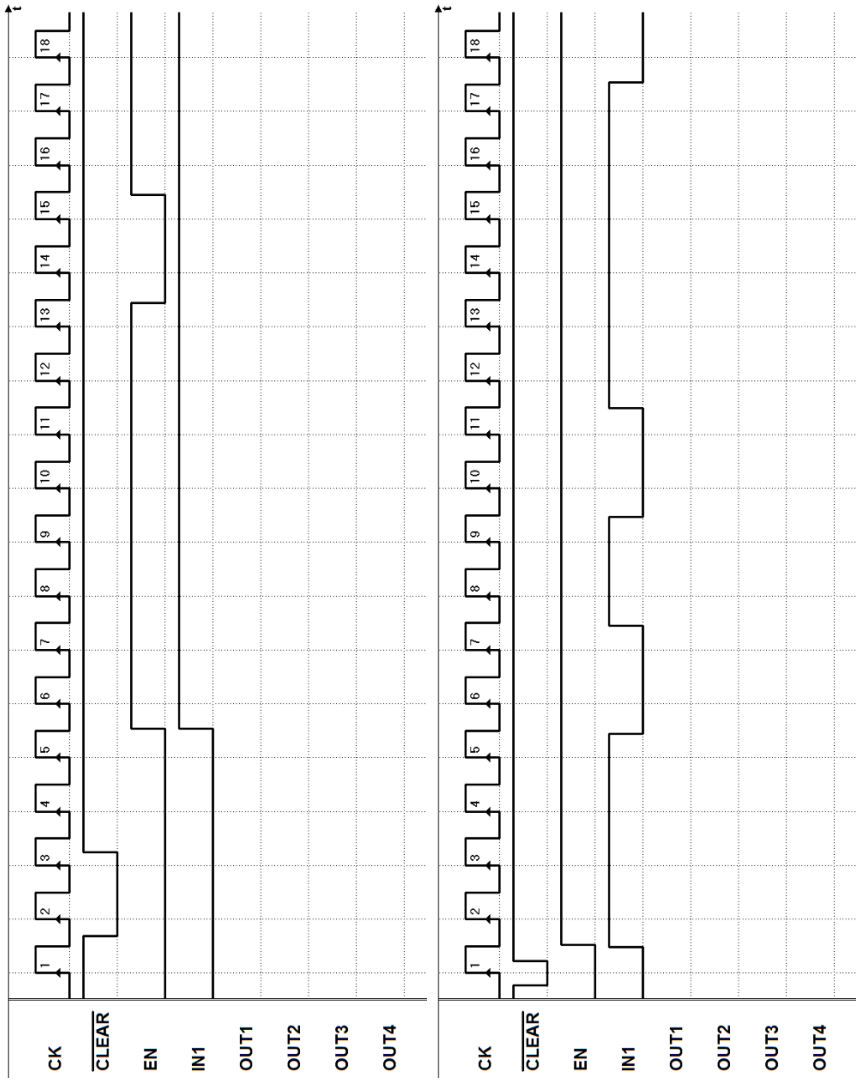
2. Esercizio 2 - (diagrammi temporali a fronte)



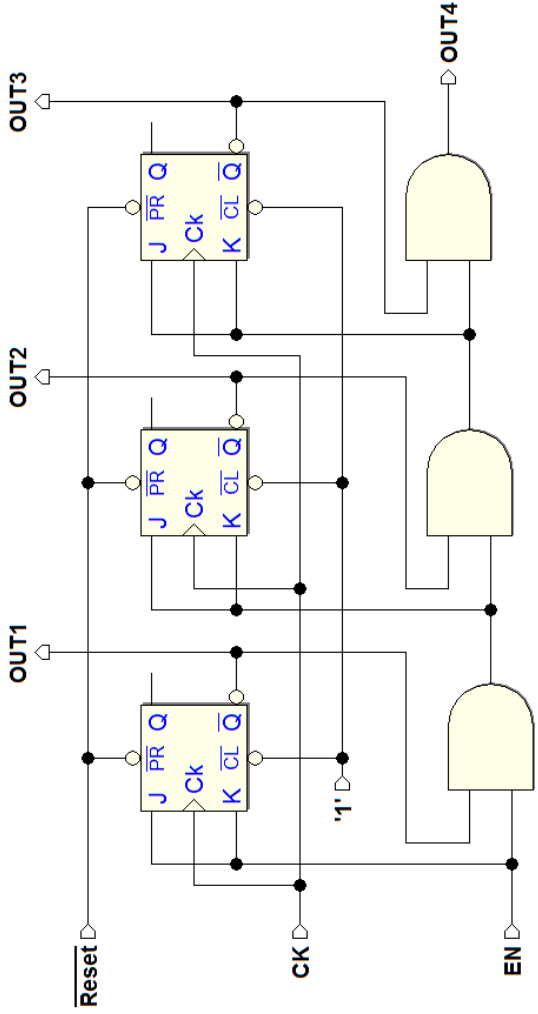


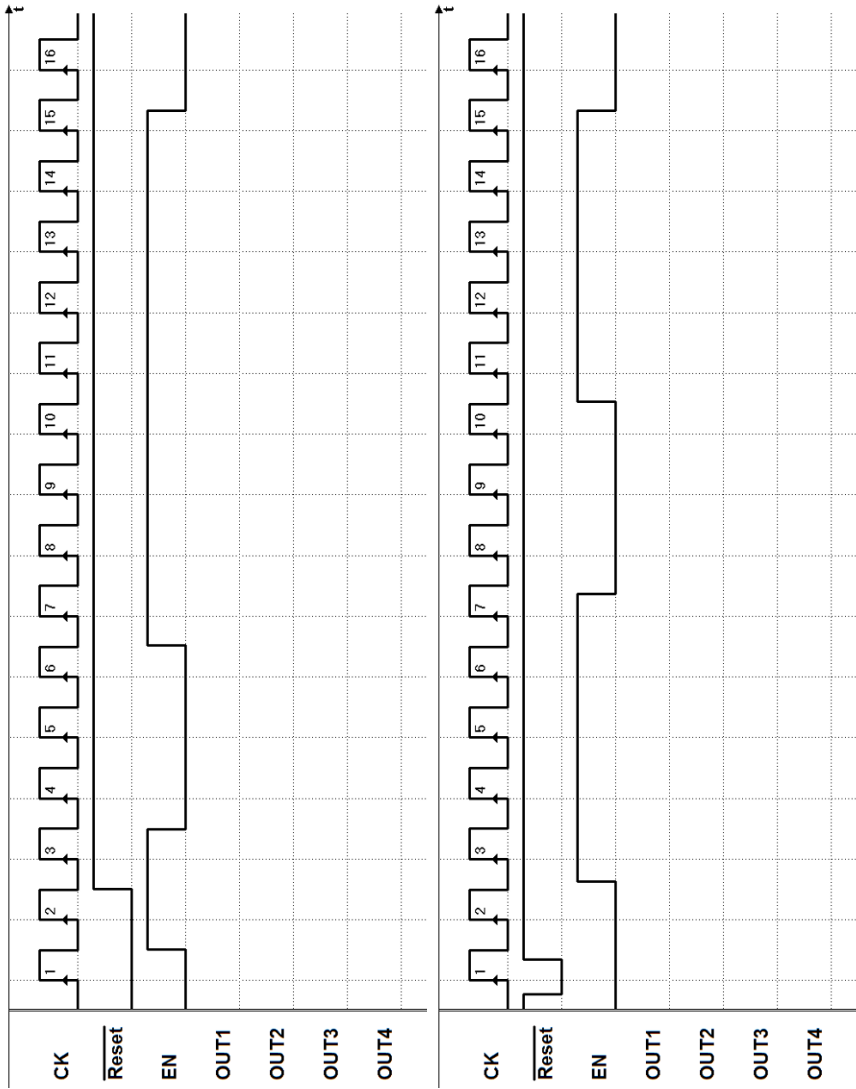
3. Esercizio 3 - (diagrammi temporali a fronte)



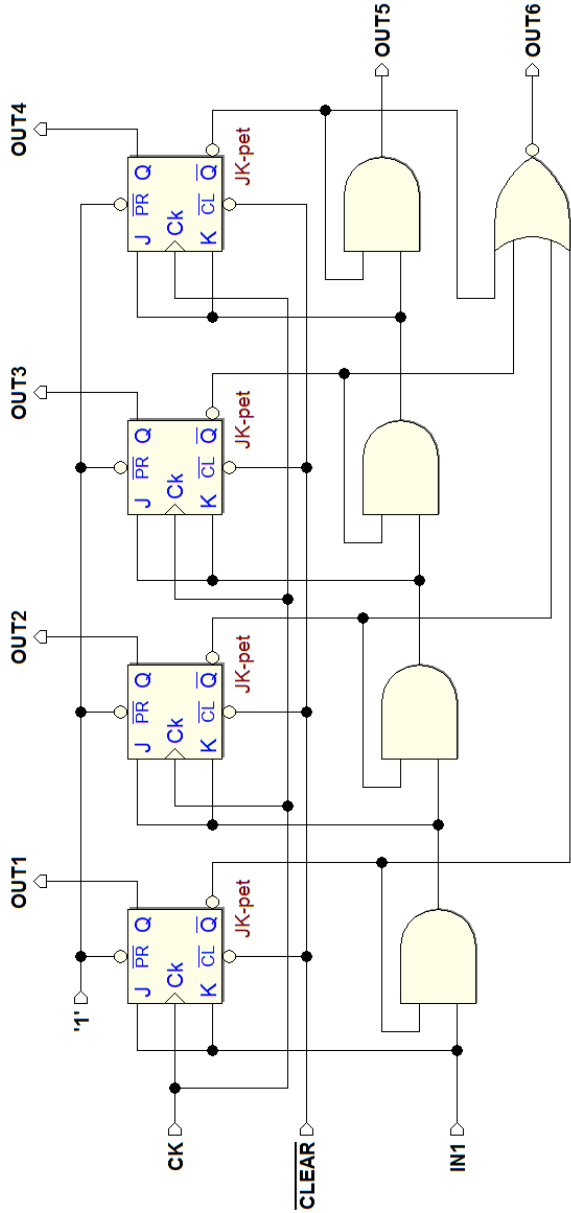


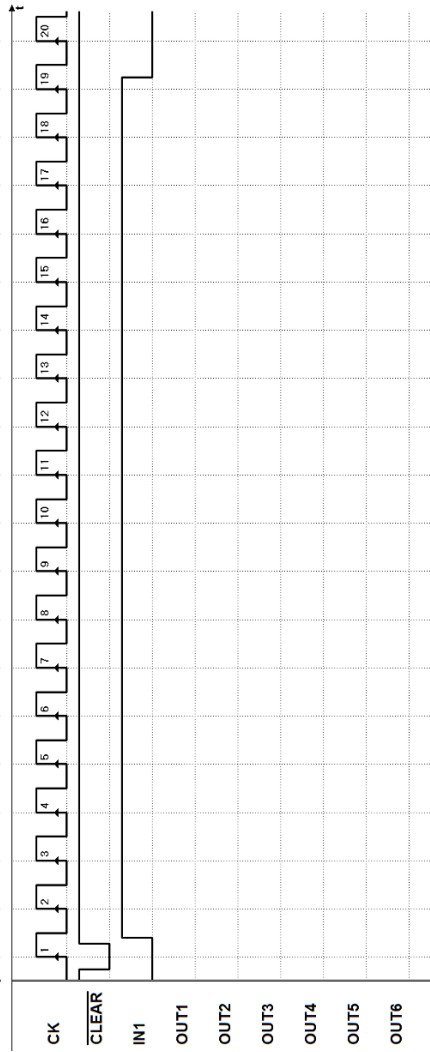
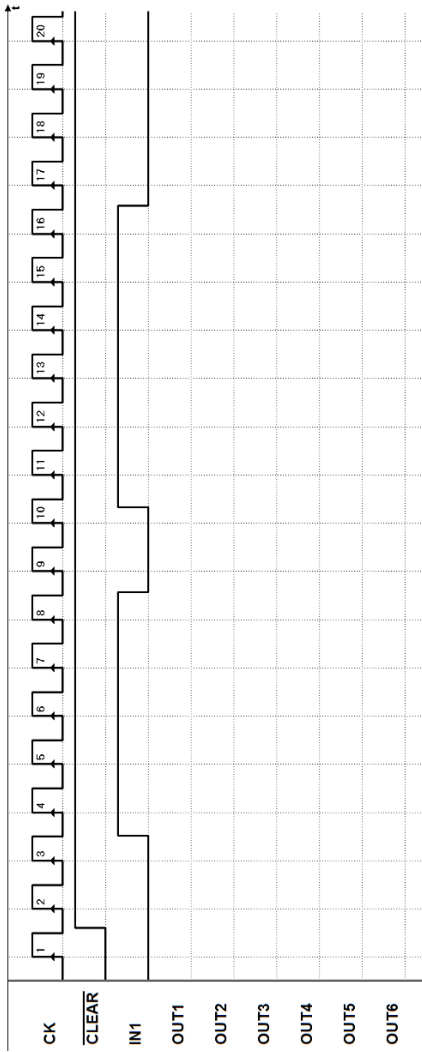
4. Esercizio 4 - (diagrammi temporali a fronte)



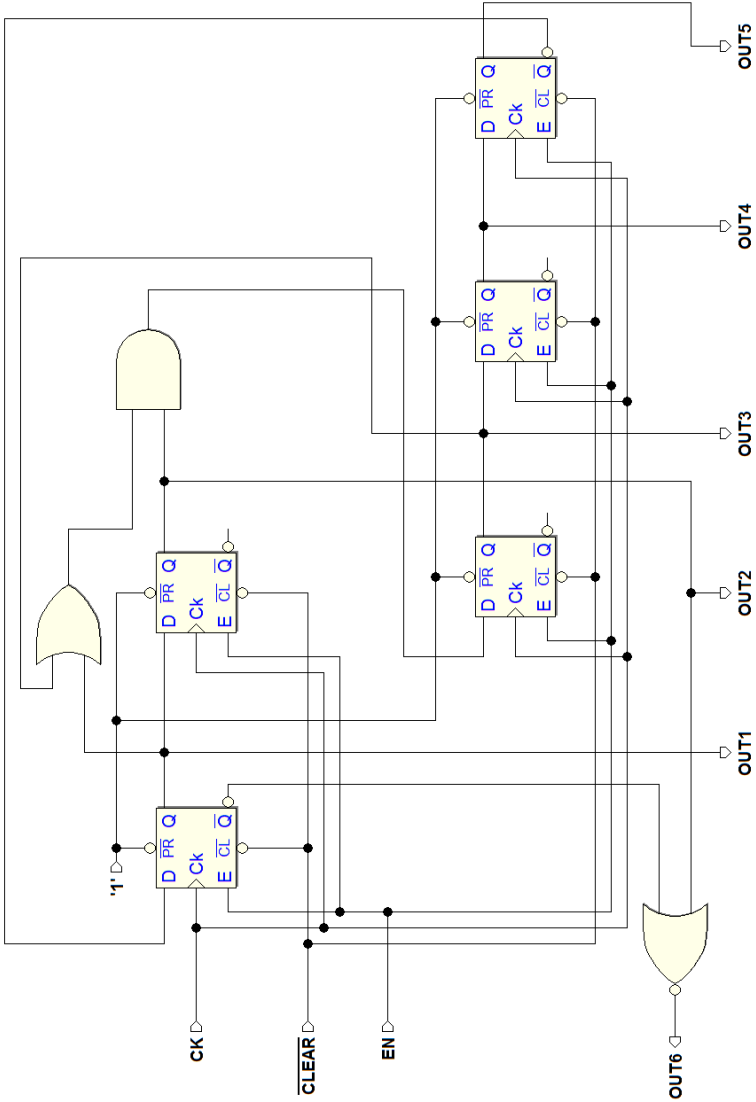


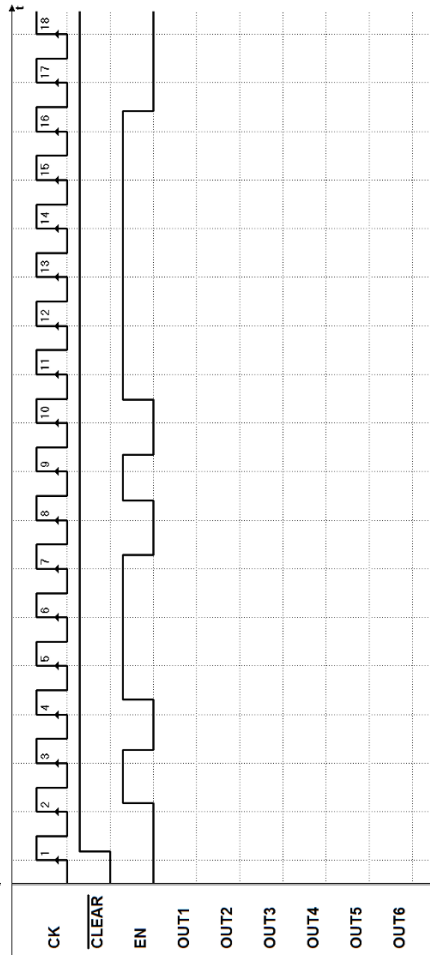
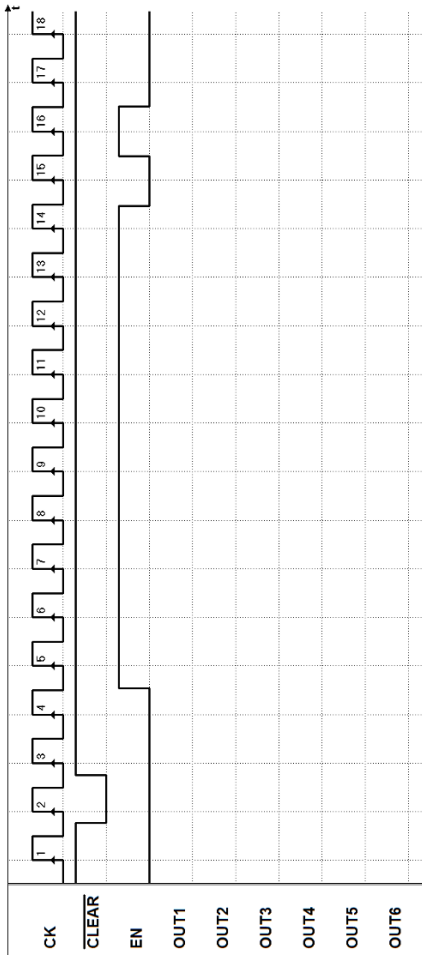
5. Esercizio 5 - (diagrammi temporali a fronte)



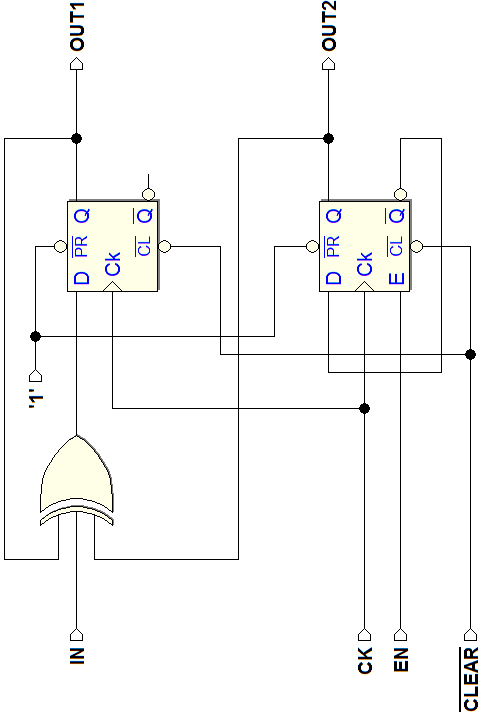


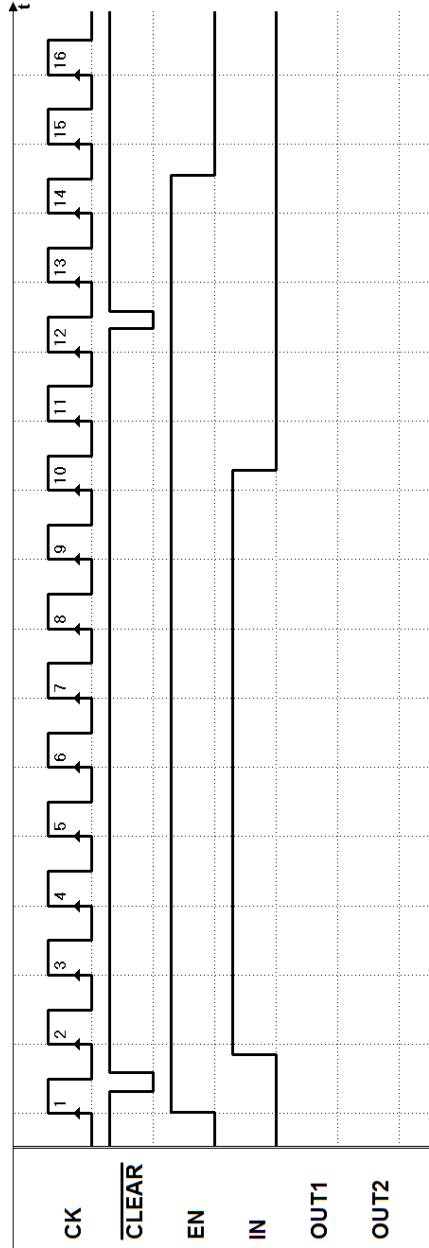
6. Esercizio 6 - (diagrammi temporali a fronte)



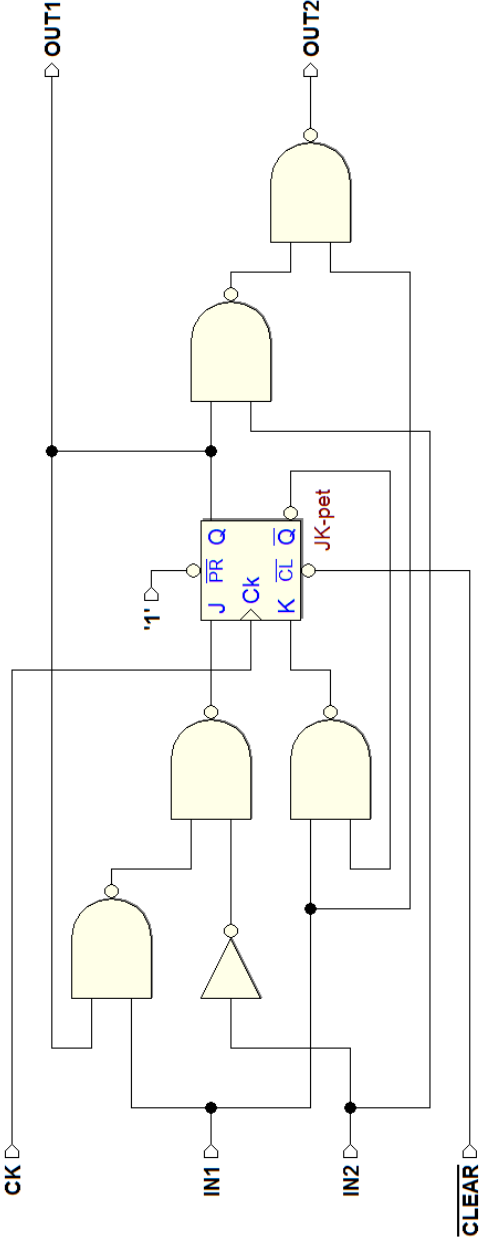


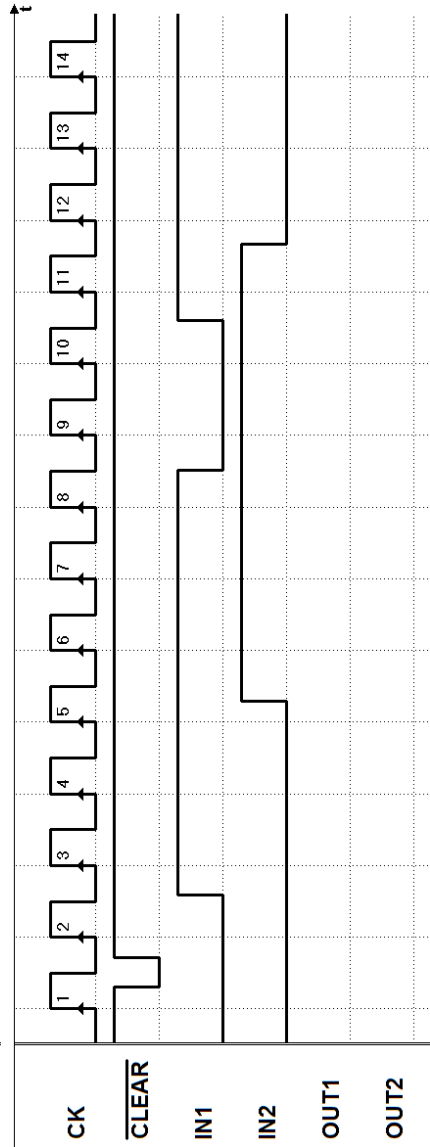
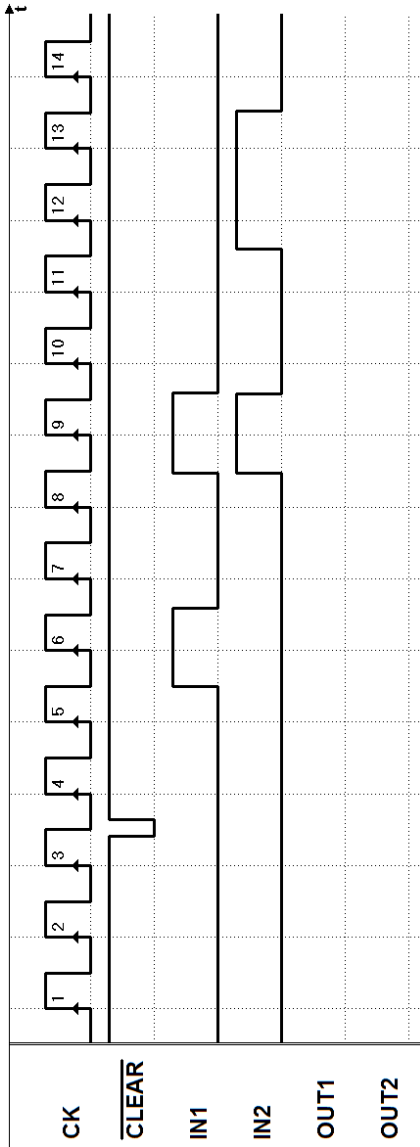
7. Esercizio 7 - (diagrammi temporali a fronte)



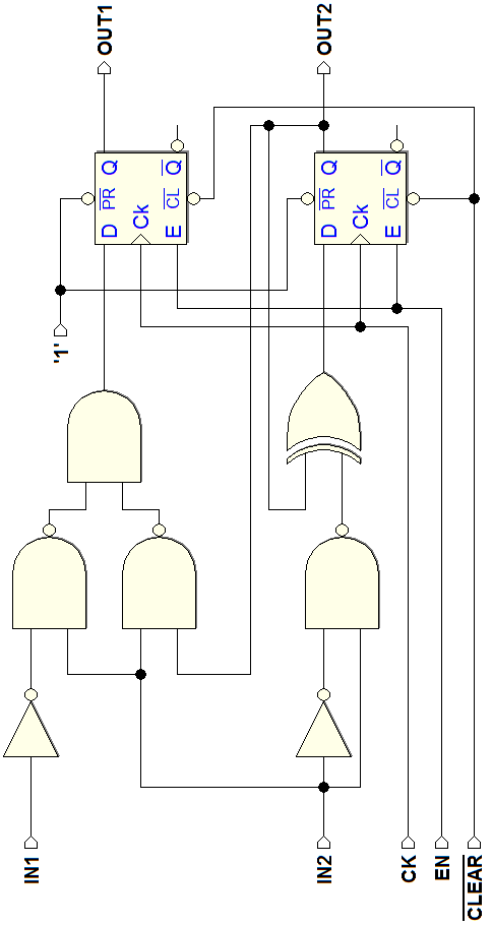


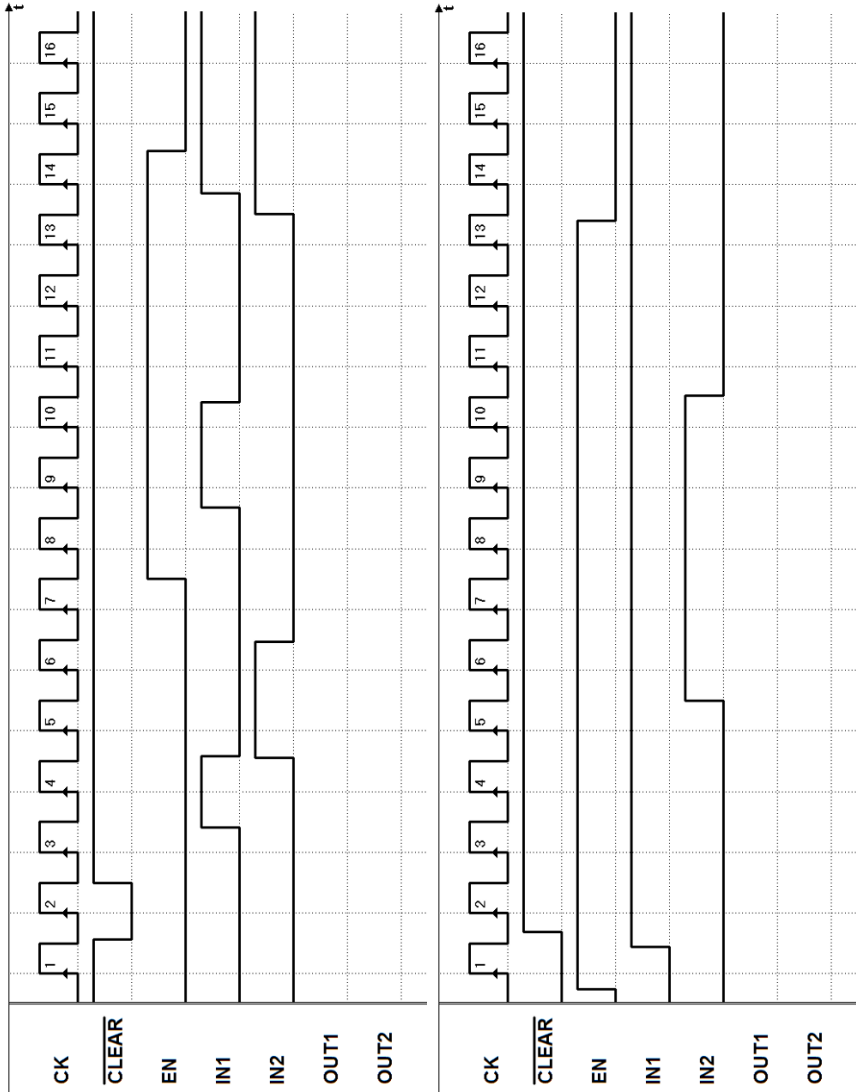
8. Esercizio 8 - (diagrammi temporali a fronte)



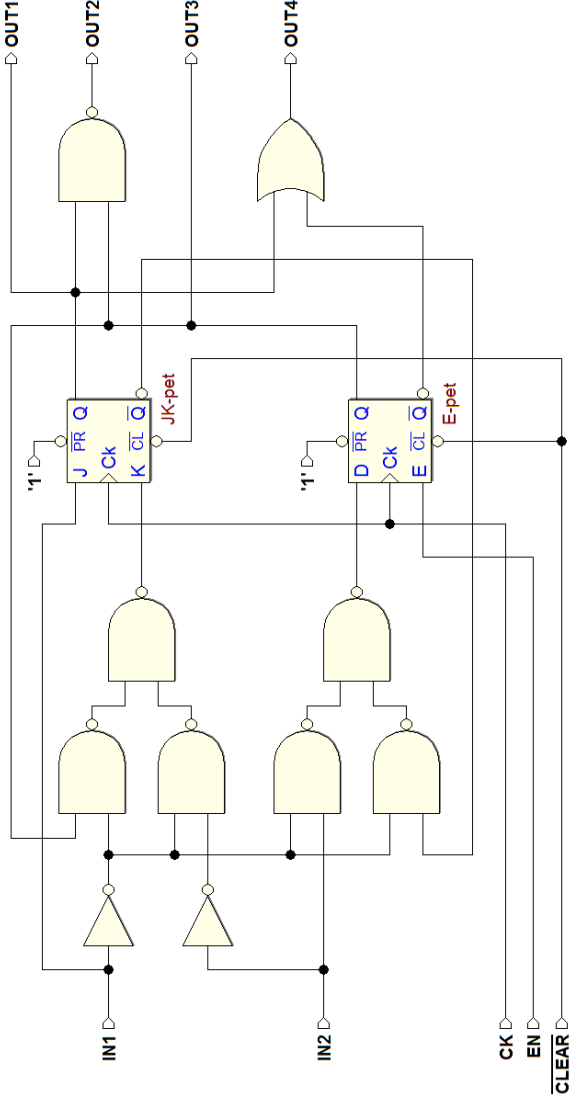


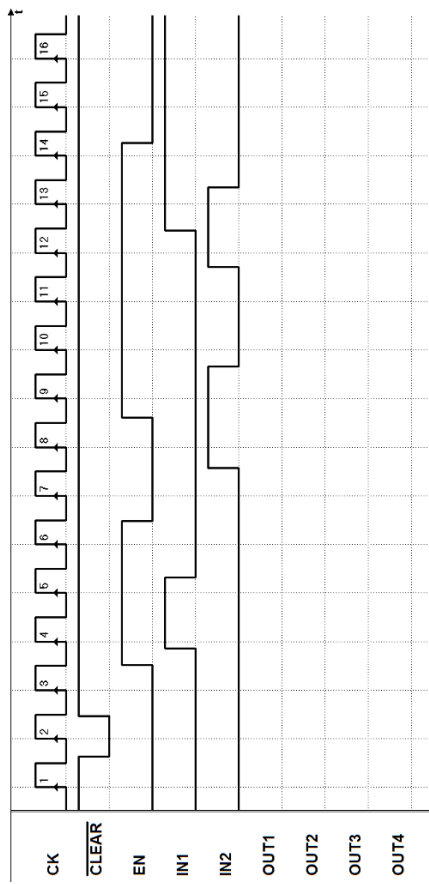
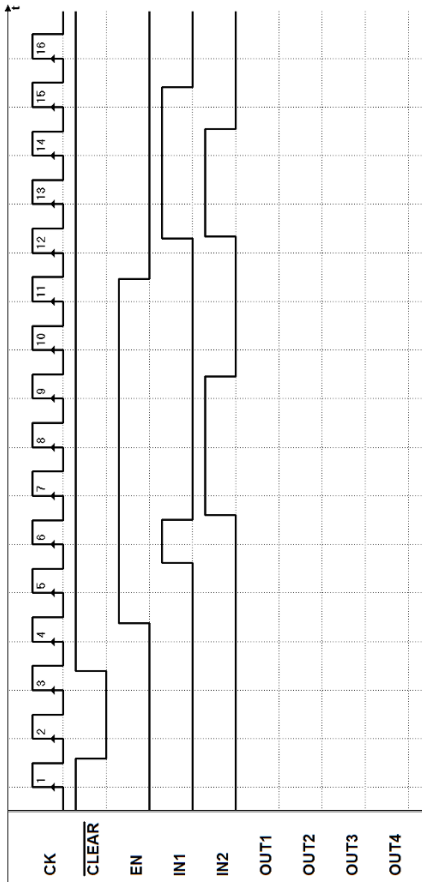
9. Esercizio 9 - (diagrammi temporali a fronte)





10. Esercizio 10 - (diagrammi temporali a fronte)

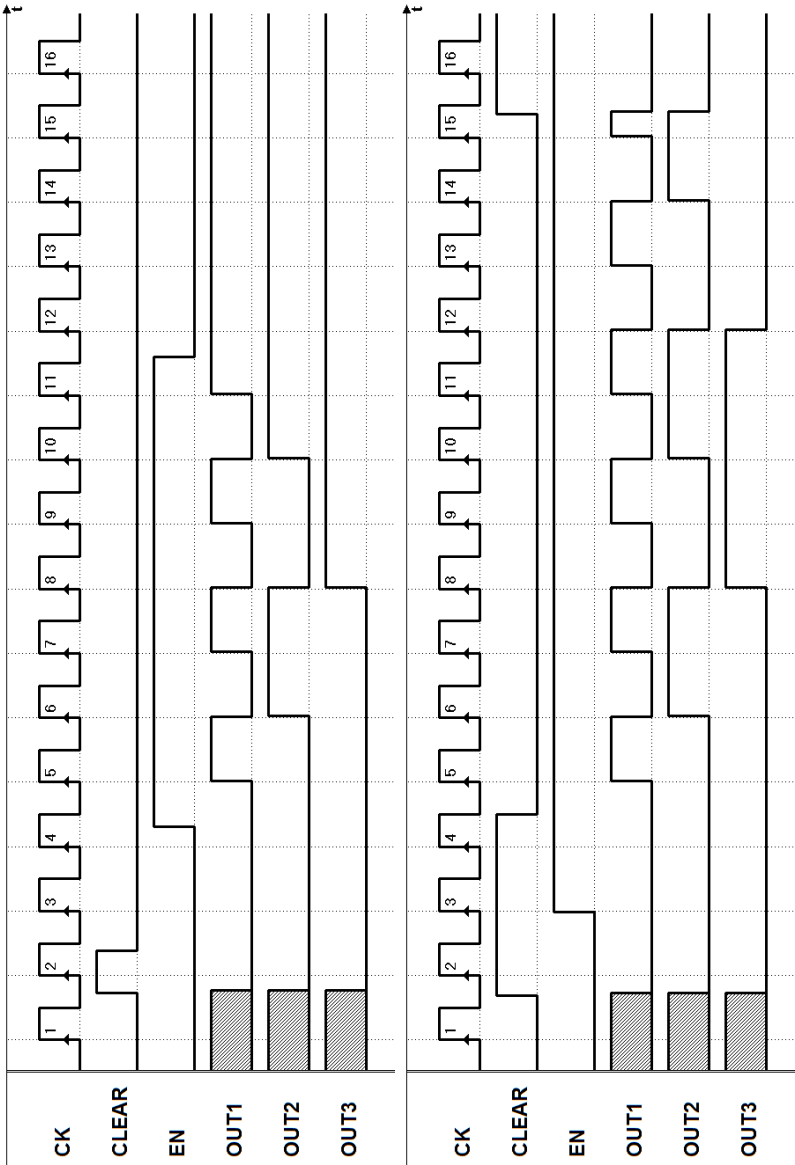




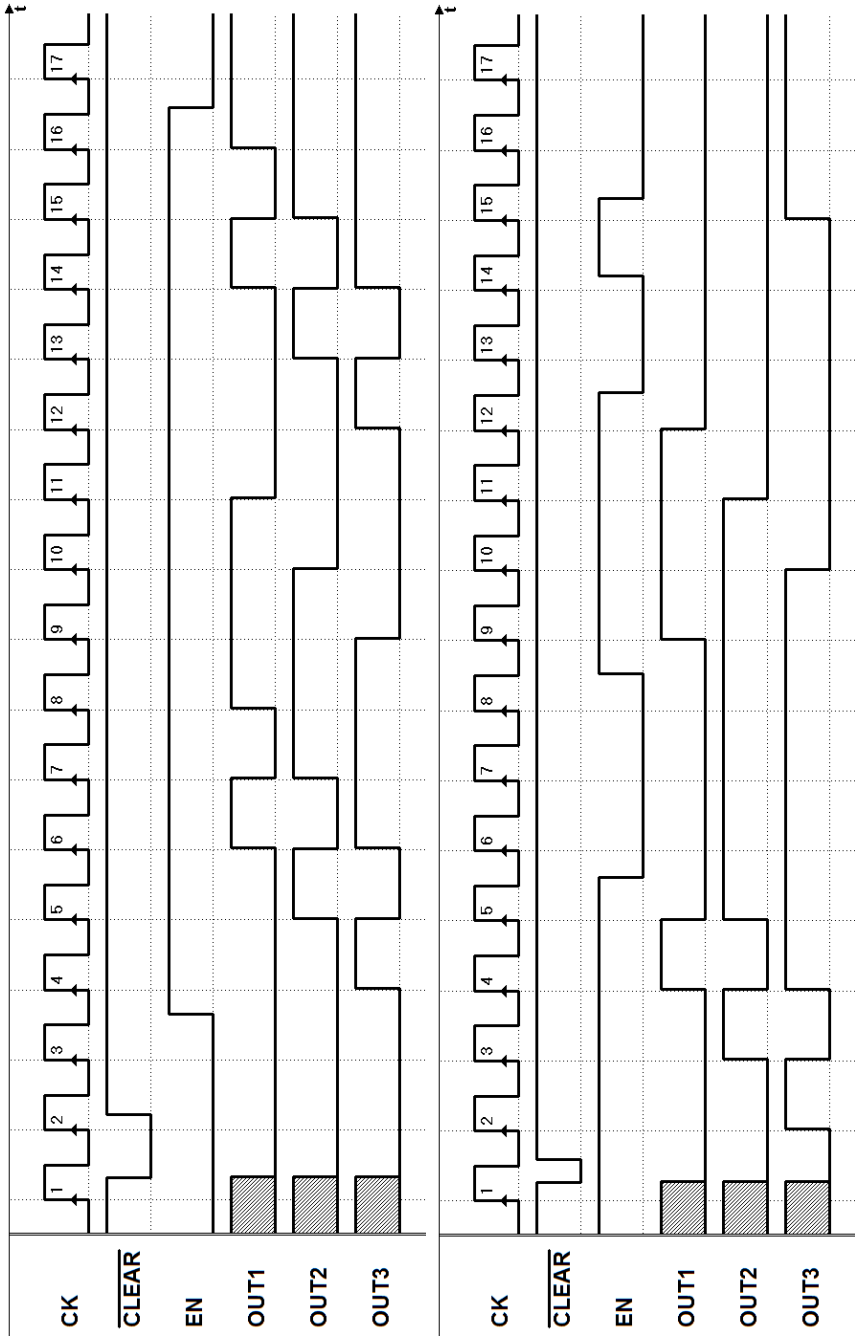
6.6 Soluzioni

I tracciati qui riportati sono stati ottenuti tramite la simulazione temporale di *Deeds*. Sul sito del simulatore, nelle pagine dei *contenuti digitali* del libro, sono disponibili i file delle reti assegnate; le soluzioni potranno quindi essere verificate anche mediante la simulazione.

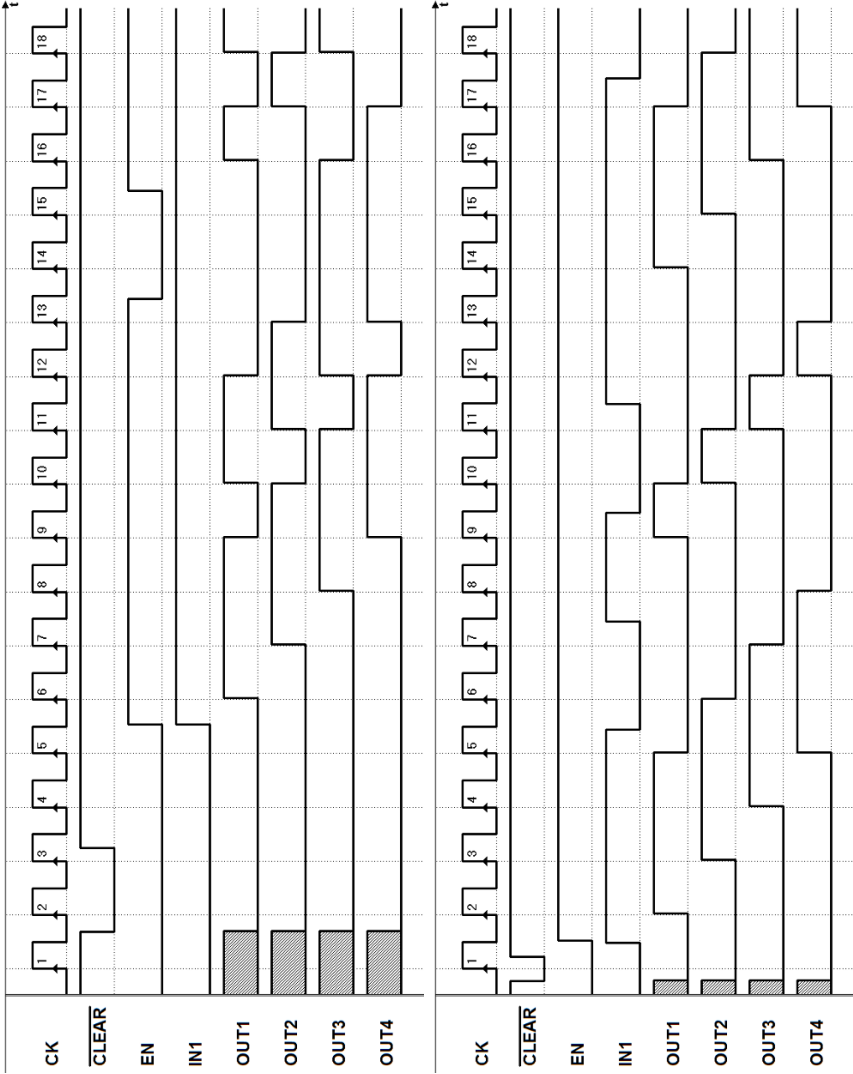
1. Esercizio 1 (soluzioni)



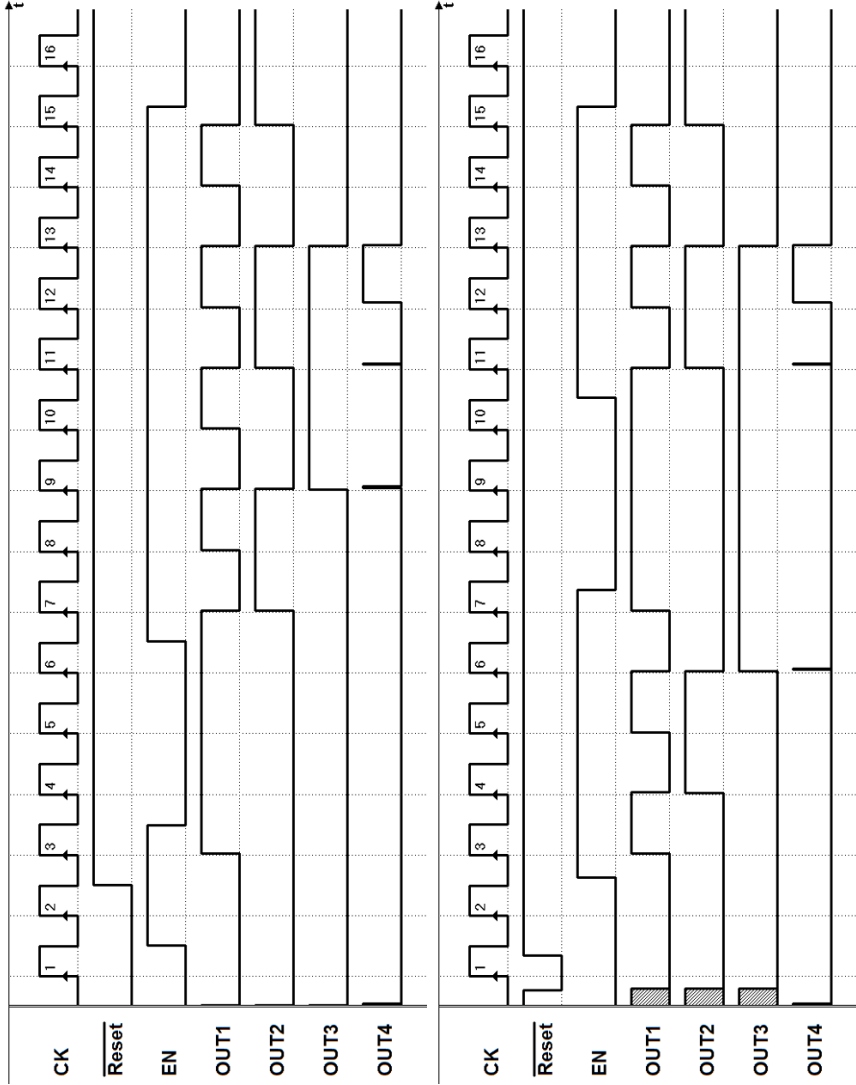
2. Esercizio 2 (soluzioni)



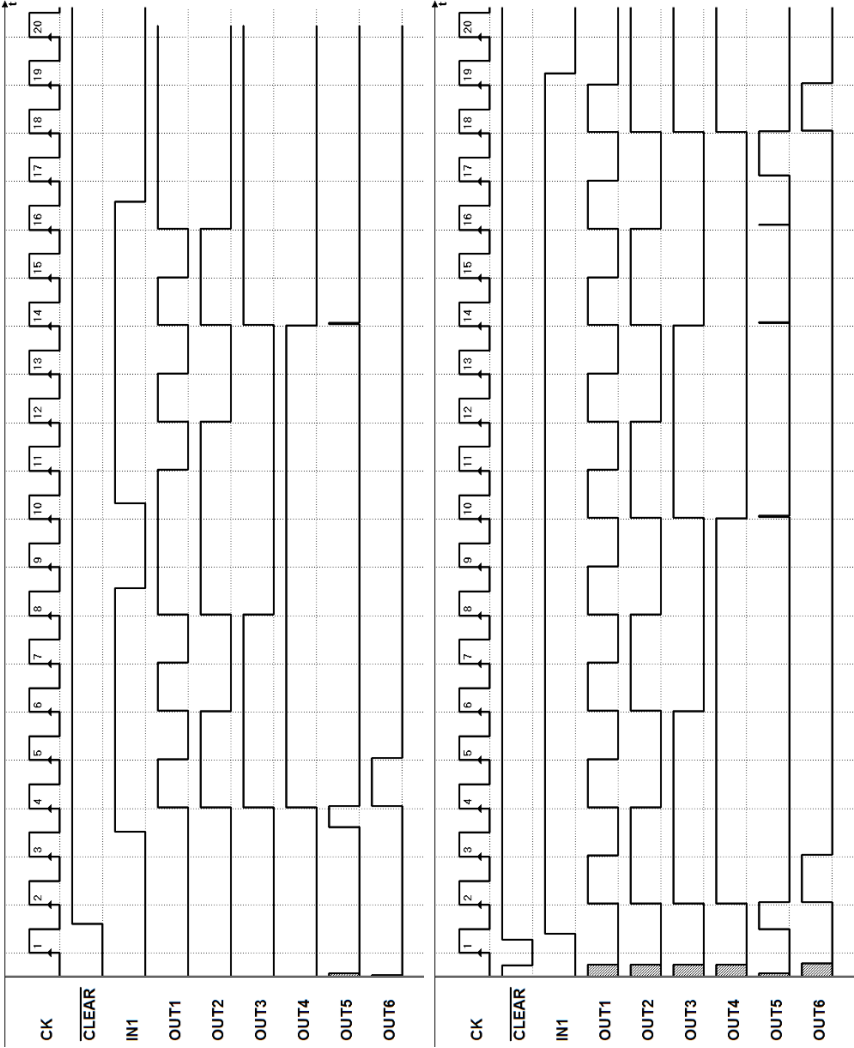
3. Esercizio 3 (soluzioni)



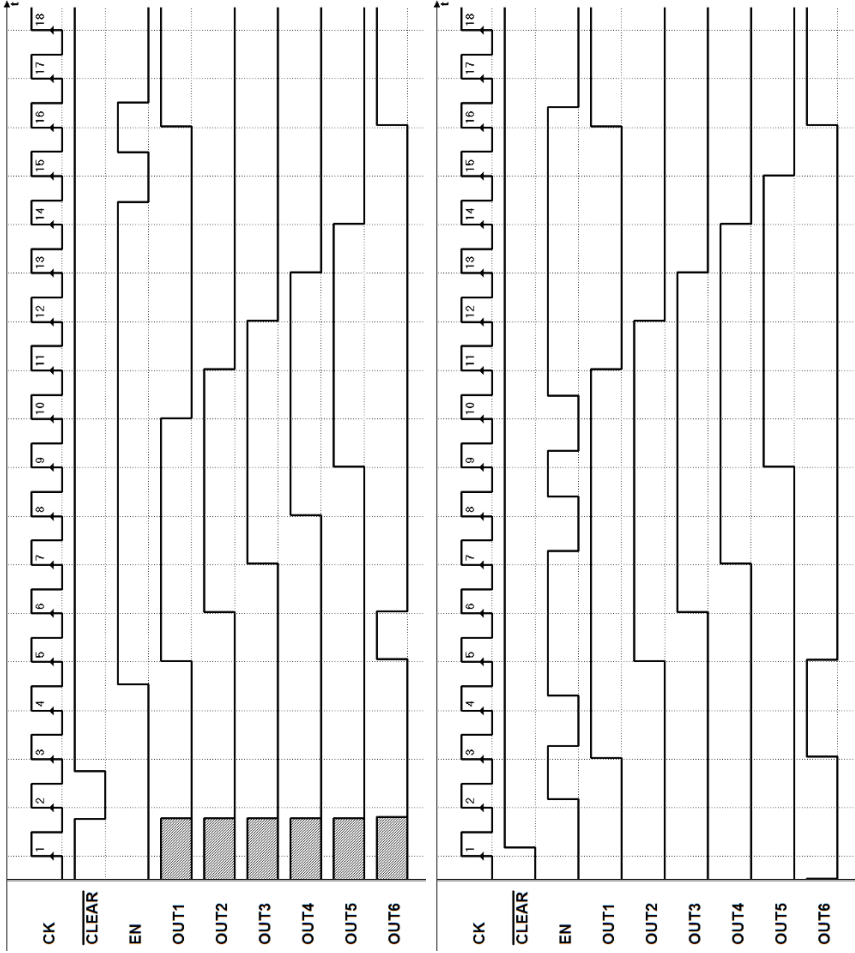
4. Esercizio 4 (soluzioni)



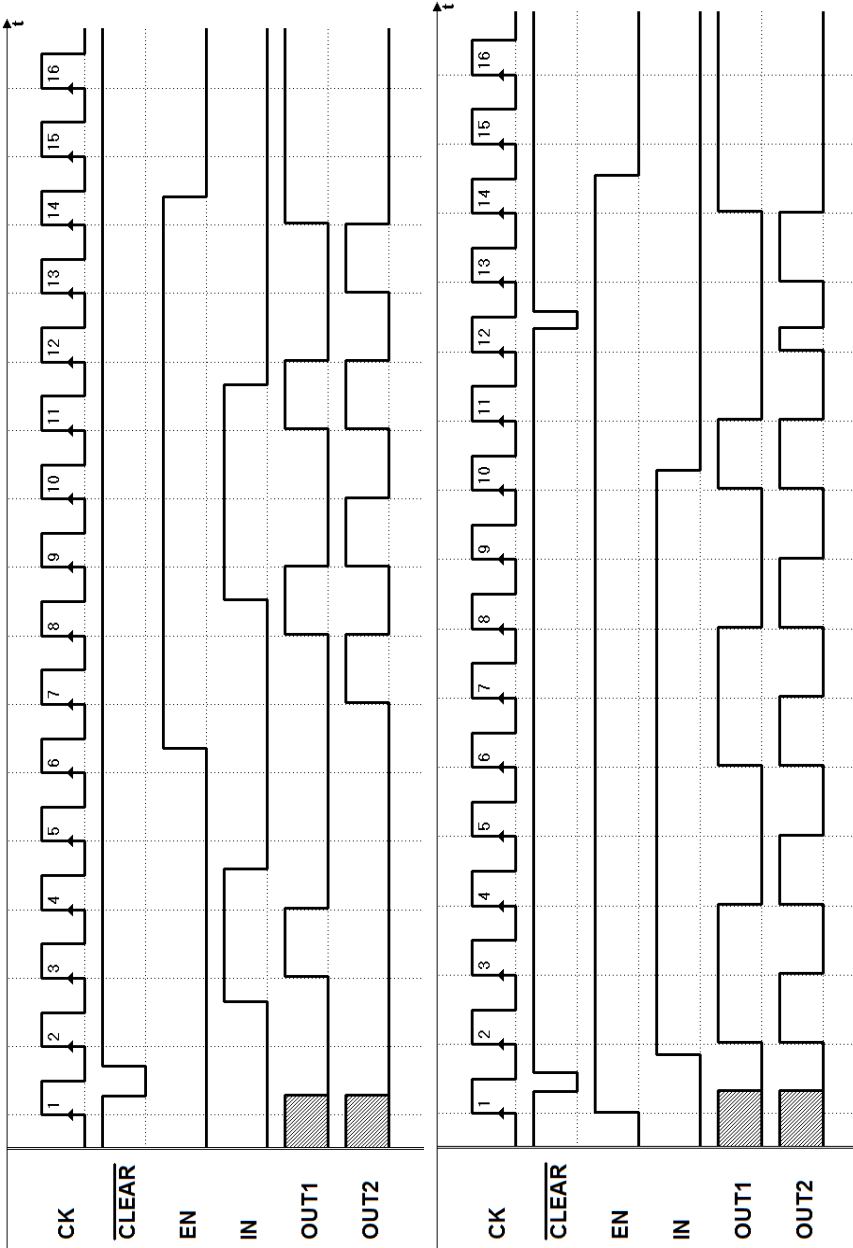
5. Esercizio 5 (soluzioni)



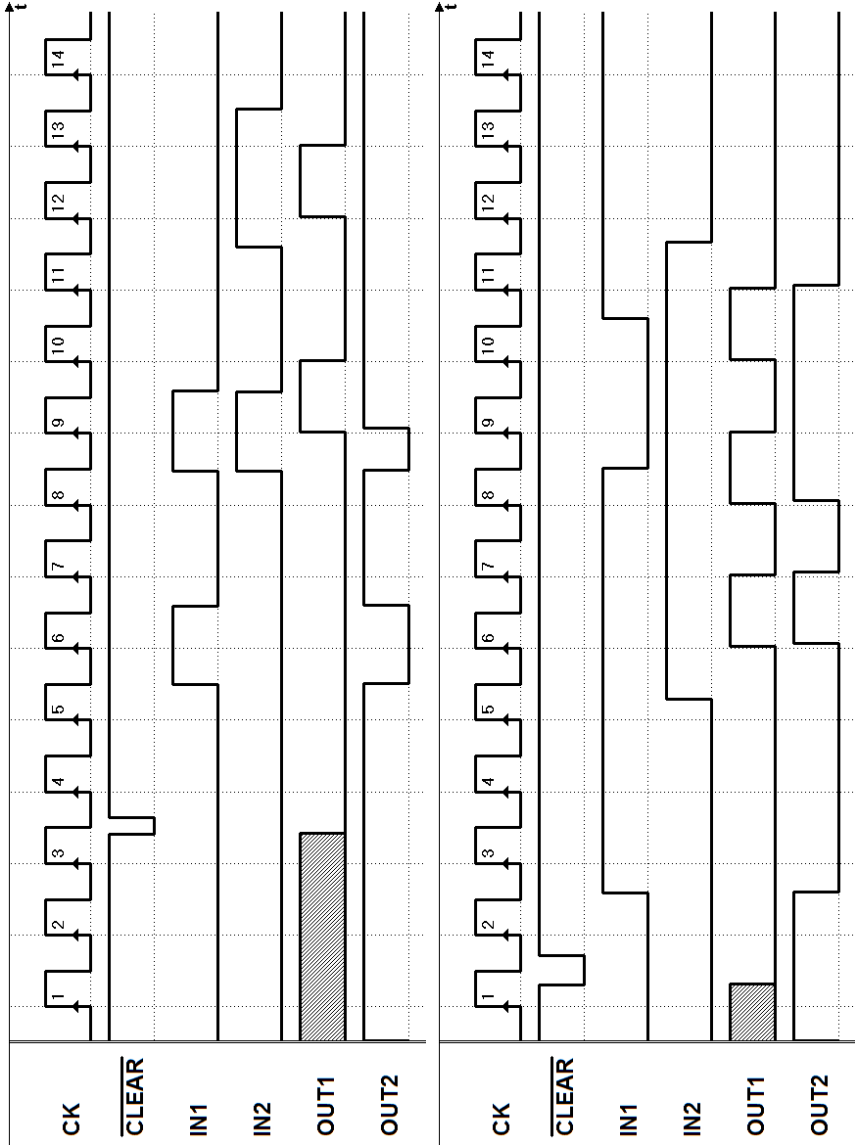
6. Esercizio 6 (soluzioni)



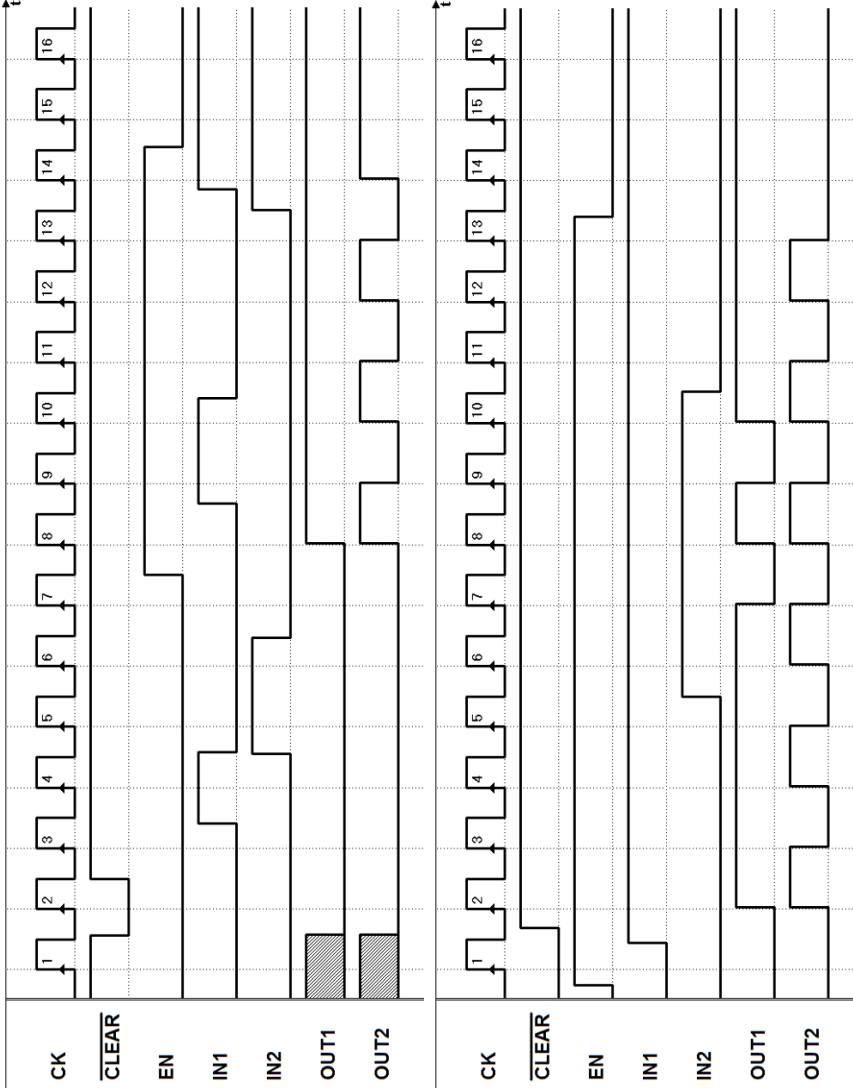
7. Esercizio 7 (soluzioni)



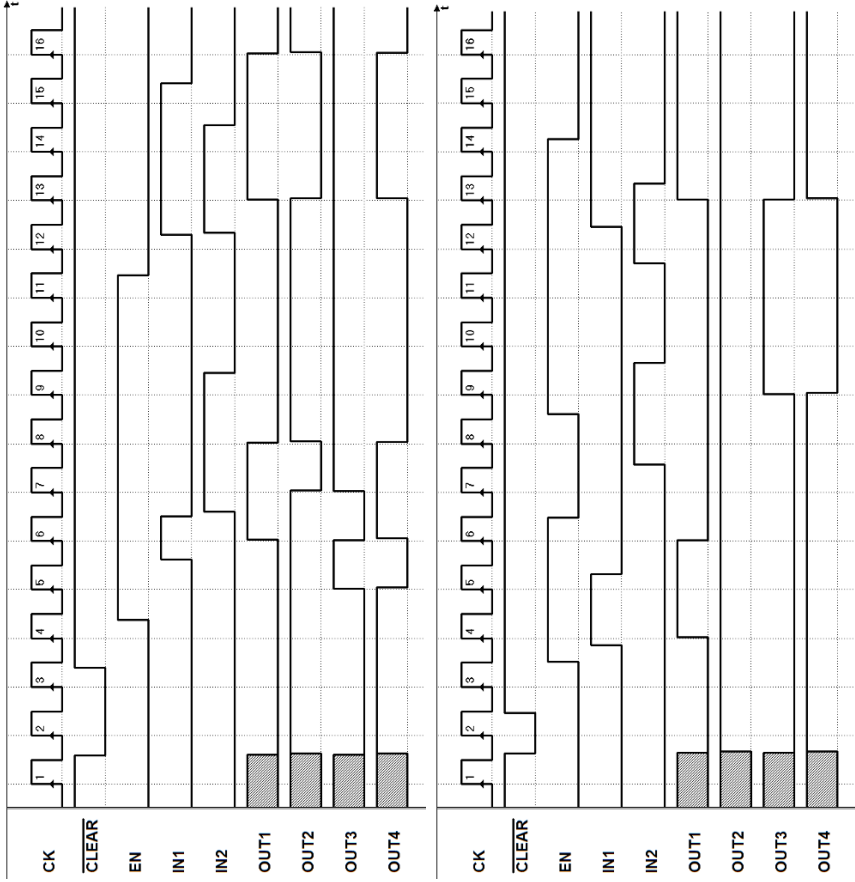
8. Esercizio 8 (soluzioni)



9. Esercizio 9 (soluzioni)

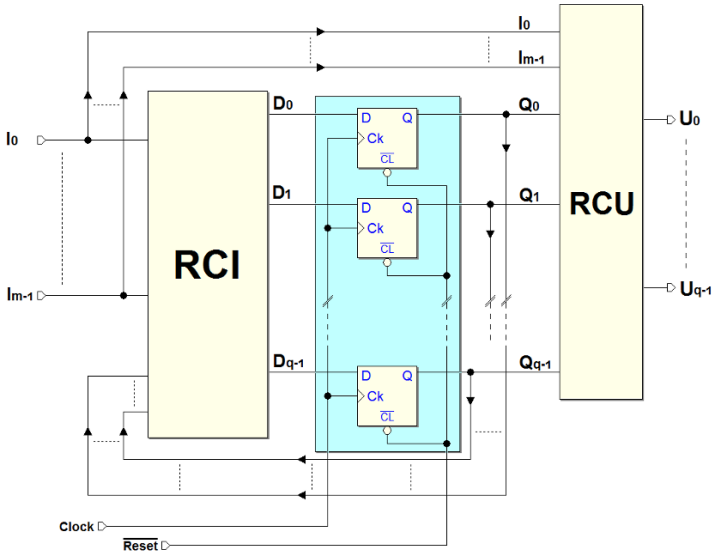


10. Esercizio 10 (soluzioni)



Reti sequenziali come Macchine a Stati Finiti

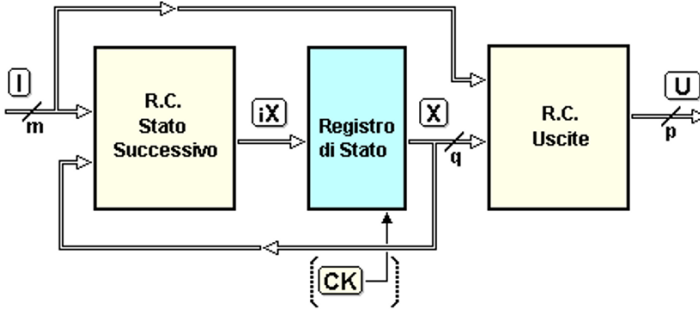
Nei precedenti capitoli abbiamo visto diversi esempi di reti logiche sequenziali, il cui funzionamento è stato analizzato mediante diagrammi temporali. Tutte le reti sincrone esaminate sono descrivibili dalla struttura generale vista in precedenza, che riproponiamo nelle figura seguente:



Abbiamo chiamato *stato* della rete l'insieme dei valori memorizzati dai flip-flop, che abbiamo raggruppato in un *registro parallelo*, che prende quindi il nome di *registro di stato* della rete (evidenziato in figura). In questo capitolo, il nostro obiettivo è di imparare a progettare le reti sequenziali sincrone, utilizzando una *metodologia* del tutto generale. A questo scopo, introduciamo qui il modello generale di *Macchina a Stati Finiti* (MSF), con il quale è possibile realizzare una rete sequenziale in grado di eseguire qualunque algoritmo logico che richieda un numero finito di operazioni.

7.1 Modello generale di Macchina a Stati Finiti

Il modello generale che descriviamo nel seguito è valido non solo per le reti sequenziali sincrone, ma anche per quelle asincrone. Una generica rete sequenziale può essere rappresentata come composta da *tre blocchi*, come visibile nella figura seguente:



Indichiamo con \mathbf{I} , \mathbf{X} e \mathbf{U} , rispettivamente, l'insieme delle variabili di *ingresso*, di *stato* e di *uscita*. Un insieme di variabili è denominato *vettore*: per esempio, nel nostro caso $\mathbf{I} = \{\mathbf{I}_0, \mathbf{I}_1, \mathbf{I}_2 \dots \mathbf{I}_{m-1}\}$, dove m è il numero di variabili di ingresso. Nella figura i vettori \mathbf{I} , \mathbf{X} e \mathbf{U} sono rappresentati indicando con un *taglio* il numero di linee che raggruppano.

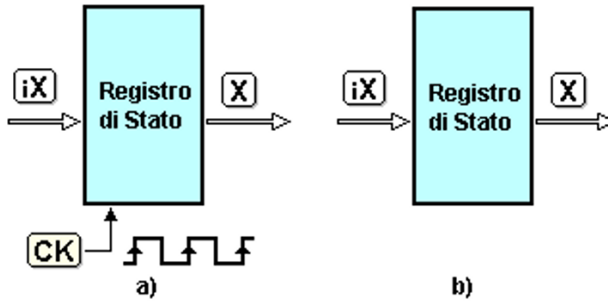
Il blocco centrale è, come abbiamo già visto, il *registro di stato*, che memorizza il *vettore di stato* \mathbf{X} , definito dall'insieme delle q variabili di stato $\{\mathbf{X}_0, \mathbf{X}_1, \mathbf{X}_2 \dots \mathbf{X}_{q-1}\}$. Ciascuna delle 2^q combinazioni delle variabili identifica un particolare *stato* della rete. Il clock \mathbf{CK} è rappresentato *tra parentesi*, perchè, come vedremo, è presente solo per le reti sincrone.

La *Rete Combinatoria dello Stato Successivo*, che in precedenza abbiamo chiamato RCI, riceve il vettore degli ingressi \mathbf{I} e dello stato \mathbf{X} e, in base a queste informazioni, genera il vettore $i\mathbf{X}$, portato in ingresso al registro. Come si è visto, il vettore \mathbf{X} assumerà, al momento opportuno, il valore di $i\mathbf{X}$.

La *Rete Combinatoria delle Uscite*, la RCU dell'esempio precedente, ha in ingresso il vettore di stato \mathbf{X} ed il vettore degli ingressi \mathbf{I} , e genera il *vettore delle uscite* della rete $\mathbf{U} = \{\mathbf{U}_0, \mathbf{U}_1, \mathbf{U}_2 \dots \mathbf{U}_{p-1}\}$.

7.1.1 Macchine Sincrone e Asincrone

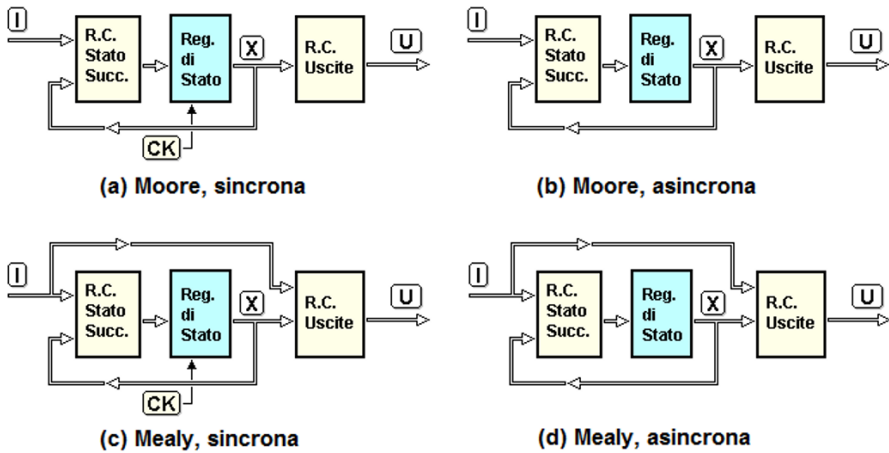
L'evoluzione temporale della rete è condizionata alla particolare struttura del registro di stato. Se il registro di stato è realizzato con *flip-flop sincroni*, temporizzati dallo stesso clock, la rete sequenziale è descritta dal modello della *MSF sincrona*. Il clock scandisce l'*evoluzione nel tempo* della sequenza degli stati assunti dalla macchina (vedi figura seguente, a sinistra):



Invece, se il registro di stato impiega *flip-flop asincroni*, non dispone di un ingresso di clock (figura sopra, a destra) ed è descritta dal modello della *MSF asincrona*. In questo caso, l'evoluzione temporale della rete dipende dalla commutazione degli ingressi e dai ritardi interni della rete.

7.1.2 Macchine di Moore e di Mealy

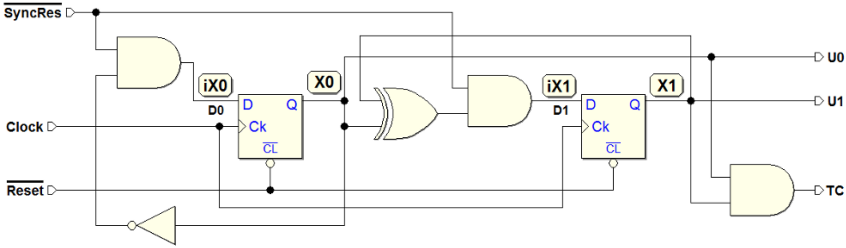
Esistono due varianti architetture, che riguardano la modalità di generazione delle uscite. Il modello generale, presentato prima, dove le uscite sono *funzione sia dello stato che degli ingressi*, è normalmente denominato *macchina di Mealy*. Nel modello chiamato *macchina di Moore*, invece, le uscite sono *solo funzione dello stato* e non dipendono direttamente dagli ingressi. Nella figura qui sotto riassumiamo le quattro possibili varianti:



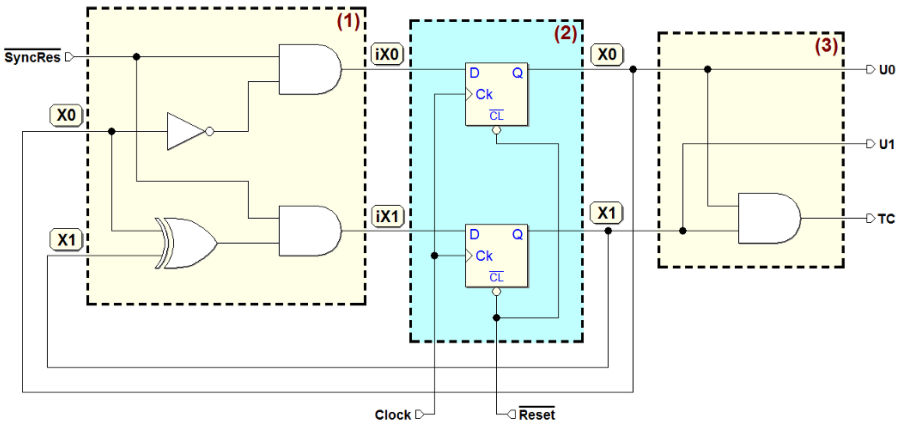
I modelli a sinistra (a), (c) sono *sincroni*: l'azione di sincronizzazione del clock impone che lo stato cambi sui fronti attivi del clock. I due modelli sulla destra (b), (d), invece, sono *asincroni*. Nelle varianti di *Moore* (a), (b) non è presente il collegamento tra gli ingressi e la rete combinatoria delle uscite, che invece è presente nelle macchine di *Mealy* (c), (d).

7.1.3 Esempio di Macchina a Stati Finiti Sincrona

Nel capitolo precedente abbiamo analizzato il funzionamento di una rete sequenziale sincrona, in grado di contare in avanti, modulo 4, con la possibilità di azzerare il conteggio in modo sincrono. La riesaminiamo ora in termini di Macchina a Stati Finiti. Come si osserva nello schema, riproposto nella figura seguente, la rete impiega due flip-flop di tipo D-PET. Possiamo considerarli come gli elementi di memoria che compongono il *registro di stato*, considerando le loro uscite come variabili di stato (X_0 , X_1):



Nella figura si osserva un insieme di porte logiche che calcolano i valori degli ingressi D dei flip-flop, a partire dall'ingresso esterno $\overline{SyncRes}$ e dalle variabili di stato X_0 e X_1 . Possiamo considerare questa parte del circuito come *rete dello stato successivo*. Analogamente osserviamo che le uscite U_0 , U_1 e TC sono generate in funzione delle variabili di stato X_0 e X_1 . Alla luce di queste osservazioni, proviamo a ridisegnare la rete, organizzando lo schema secondo il modello generale della *MSF Sincrona di Moore*:



Il blocco (1) si occupa di generare lo stato successivo, proponendolo all'ingresso dei flip-flop (iX_0 e iX_1), sulla base dell'ingresso $\overline{SyncRes}$ e dello stato attuale X_0 e X_1 . Ricaviamone le espressioni booleane:

$$iX_0 = (\overline{SyncRes} \cdot \overline{X_0})$$

$$iX_1 = (\overline{SyncRes} \cdot (X_0 \oplus X_1))$$

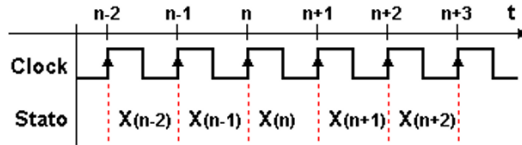
Il blocco (2), il registro di stato, memorizza i valori di iX e $iX1$ ad ogni fronte attivo del *Clock*, aggiornando lo stato rappresentato da $X0$ e $X1$. Il segnale \overline{Reset} è utilizzato per l'inizializzazione asincrona dei flip-flop.

Il blocco (3) genera le uscite della rete $U0$, $U1$ e TC , che dipendono dalle variabili di stato $X0$ e $X1$, nel modo seguente:

$$U0 = X0; \quad U1 = X1; \quad TC = (X0 \cdot X1).$$

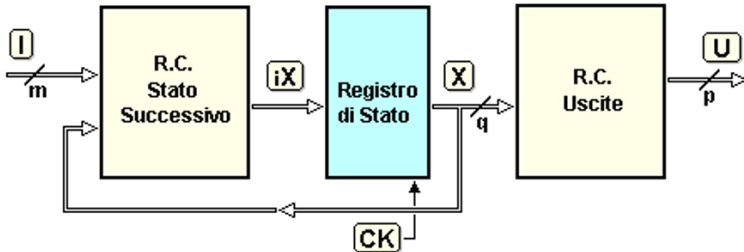
7.1.4 Equazioni generali dello stato successivo e delle uscite

Nelle MSF *sincrone*, il clock ha la funzione di imporre gli istanti $n-1$, n , $n+1$... nei quali la MSF può cambiare stato. Ciclicamente, ad ogni evento del clock il vettore iX è trasferito nel registro di stato, aggiornando il valore di X . Il tempo si riduce ad una successione di eventi numerabili:



Indichiamo con $X(n)$ lo *stato* della MSF nell'intervallo $[n, n+1]$. In corrispondenza degli eventi $n-1$, n , $n+1$, ..., lo stato assume i valori $X(n-1)$, $X(n)$, $X(n+1)$... (in figura, gli eventi corrispondono ai fronti di salita del clock).

Facciamo ora riferimento alla MSF *sincrona di Moore*, che riproponiamo qui:



Esprimiamo il valore dello stato nell'intervallo $[n, n+1]$, cioè $X(n)$, come funzione dello stato precedente $X(n-1)$ e degli ingressi $I(n-1)$, presenti nell'intervallo $[n-1, n]$:

$$X(n) = f(X(n-1), I(n-1))$$

Questa espressione è denominata “*equazione dello stato successivo*”. Ricordando che X è un vettore composto da q variabili, avremo in effetti q equazioni scalari, una per ogni variabile di stato. Più sinteticamente, l'equazione precedente si può anche indicare come:

$$X \leftarrow f(X, I)$$

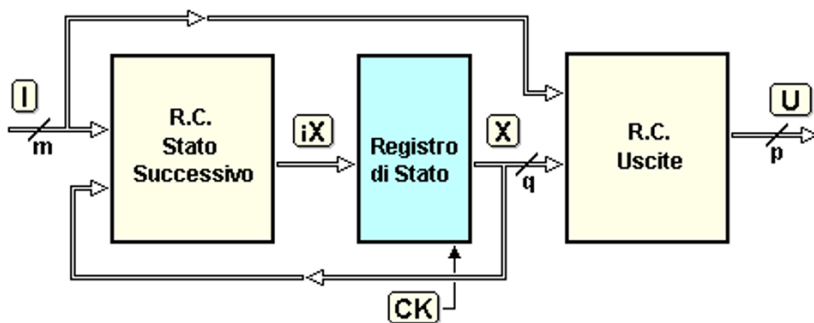
La freccia “←” ricorda che X è “funzione ritardata” degli ingressi e dello stato precedente. La funzione $U(n)$, che rappresenta l’uscita del sistema nell’intervallo $[n, n+1]$, è:

$$U(n) = g(X(n))$$

Quest’ultima espressione è l’equazione delle uscite per la MSF di Moore, e rappresenta la dipendenza delle uscite dallo stato che assume nell’intervallo $[n, n+1]$. Dato che lo stato $X(n)$ è costante per tutto l’intervallo, anche le uscite saranno costanti per tutta la durata di tale intervallo.

Teniamo presente, inoltre, che se p è il numero di variabili che costituiscono il vettore delle uscite, avremo p funzioni scalari, una per ogni uscita.

Per la MSF sincrona di Mealy, invece, la funzione $U(n)$ è differente, dal momento che qui è presente il collegamento tra gli ingressi della MSF e la rete combinatoria delle uscite (vedi figura qui sotto):



L’equazione seguente è la “equazione delle uscite” per la MSF di Mealy, e descrive il legame di queste non soltanto con lo stato $X(n)$ della macchina, ma anche con gli ingressi $I(n)$ della MSF:

$$U(n) = g(X(n), I(n))$$

Si noti che, mentre lo stato $X(n)$ è costante per tutto l’intervallo $[n, n+1]$, altrettanto non si può dire degli ingressi $I(n)$ che, potendo in generale variare in qualunque istante, condizioneranno il valore delle uscite $U(n)$ durante il tempo di detto intervallo.

Le uscite che dipendono anche dal valore degli ingressi sono denominate “uscite condizionate”.

Nel seguito, per semplicità, si parlerà di *intervallo* n o di *stato* n , riferendosi, rispettivamente, all’intervallo temporale $[n, n+1]$ ed allo stato $X(n)$.

7.2 Diagrammi ASM

Nella progettazione di una rete sequenziale conviene adottare un approccio *comportamentale*, piuttosto che circuitale. Questo significa iniziare il progetto dalla definizione dell’algoritmo di funzionamento della rete, e procedere lungo un percorso di tipo *top-down*. Solo in un secondo tempo l’algoritmo sarà realizzato in forma circuitale, attraverso un procedimento chiamato *sintesi*.

Tra i vari metodi che possiamo utilizzare per costruire un algoritmo, il metodo ASM (“*Algorithmic State Machine*”) consente la descrizione e la sintesi delle reti sequenziali in modo semplice ed intuitivo.

I *diagramma ASM* sono apparentemente simili ai *diagrammi di flusso* utilizzati in programmazione, ma sono strutturati in modo da descrivere *l’evoluzione degli stati* di una rete: sono stati introdotti negli anni settanta da Christopher R. Clare nel suo “*Designing Logic Systems Using State Machines*” (McGraw-Hill, 1973, non più ristampato). I diagrammi ASM possono descrivere sia le MSF *sincrone* che quelle *asincrone*. Qui di seguito, i diagrammi ASM vengono applicati alle *macchine sincrone*.

7.2.1 Descrizione degli stati

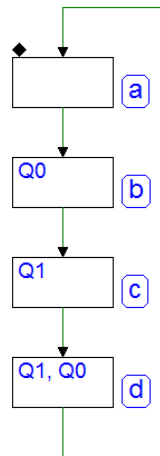
Presentiamo qui di seguito un primo esempio di diagramma ASM, nel quale sono presenti quattro *blocchi di stato* (i rettangoli), interconnessi da linee che rappresentano la *successione logica* degli stati.

Proviamo ad interpretare insieme la figura qui a lato. Il diagramma descrive *l’algoritmo di funzionamento* di un dispositivo che può assumere *quattro stati diversi*, rappresentati dai blocchi (a), (b), (c) e (d).

Le lettere rappresentano i *nomi simbolici* assegnati agli stati, e si indicano sulla destra dei blocchi, racchiusi in un ovale. Come indicano *le frecce*, questi quattro stati si succedono, *uno dopo l’altro*, nell’ordine indicato.

Agli stati possono essere associate delle uscite: il diagramma mostra solo le uscite *attive*, stato per stato, indicandole all’interno dei rettangoli. Ad esempio, nello stato (c), risulta attiva l’uscita *Q1*, mentre nello stato (d) risultano attive le uscite *Q1* e *Q0*. Per convenzione, si indicano all’interno di un blocco di stato *solo le uscite attive*, mentre le uscite *non riportate* si intendono *inattive*.

Nel nostro esempio, *Q1* e *Q0* sono le due sole uscite del dispositivo, in quanto il diagramma non ne menziona altre. Nel diagramma descritto *non compaiono ingressi*. Infatti, il nostro dispositivo non possiede ingressi che intervengono sulla *successione degli stati* (come invece vedremo nei prossimi esempi).

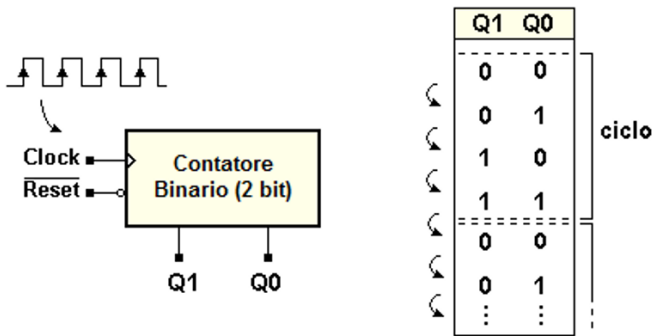


Nella descrizione ASM della rete non sono mai rappresentati, in quanto non partecipano alla sua descrizione comportamentale, gli ingressi di inizializzazione e di sincronizzazione (\overline{Reset} e $Clock$).

Inoltre, un diagramma ASM non esplicita le *modalità temporali* con cui la macchina *passa di stato in stato*, che dipendono dalla struttura del registro di stato (ossia se la rete è *sincrona* o *asincrona*).

Il progettista, nel momento in cui passa a descrivere la MSF in termini algoritmici, ha già deciso se la rete sarà *sincrona* o *asincrona*. Nel nostro esempio, il dispositivo è stato pensato *sincrono*, e quindi sarà il $Clock$ a governare l'*avanzamento degli stati*. Nel seguito daremo per scontato che tutti i dispositivi che analizzeremo e progetteremo siano *sincroni*.

Nella figura seguente, sulla sinistra, è rappresentato uno schema a blocchi che descrive il dispositivo. Possiede gli ingressi di $Clock$ e di inizializzazione \overline{Reset} , e le due uscite $Q1$ e $Q0$:

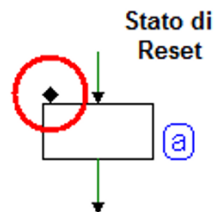


Nella parte a destra, invece, è riportata la sequenza delle uscite, di stato in stato, come indicata dal diagramma ASM. Da questa sequenza è immediato osservare che il nostro dispositivo si comporta come *contatore binario sincrono* a due bit: conta da 0 a 3, in modo ciclico (modulo 4).

L'attivazione del Reset

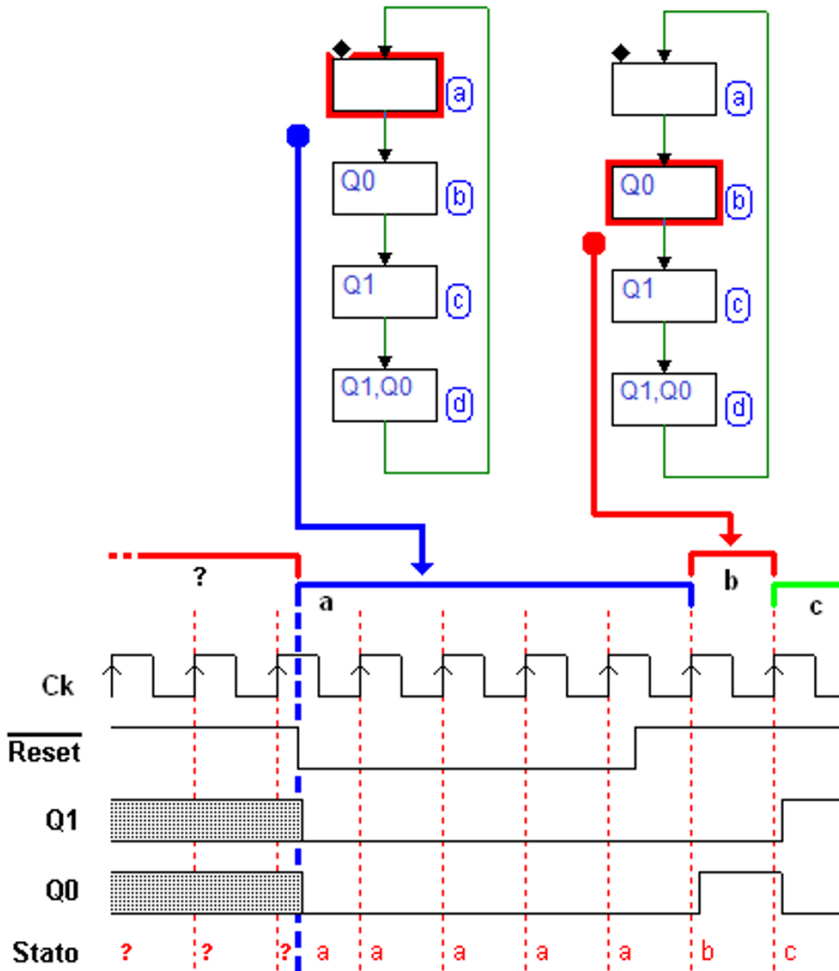
Quando l'ingresso di inizializzazione \overline{Reset} è attivato, i flip-flop del registro di stato sono forzati ad un valore noto. La rete si porta cioè in uno stato predefinito, che in sede di progetto chiamiamo *stato di Reset*.

Per nostra convenzione, come evidenziato qui a lato, lo *stato di Reset* è indicato nel diagramma con un *piccolo rombo* in alto a sinistra del rettangolo.



Nella figura successiva è evidenziata una *sequenza* nella quale esaminiamo il comportamento della rete all'inizializzazione. Come si vede nel tracciato temporale, si è scelto di iniziare la simulazione con il \overline{Reset} non attivo.

Dato che non ipotizziamo nulla circa lo stato della rete *prima* della attivazione del \overline{Reset} , rappresentiamo lo stato come inizialmente *ignoto* (tramite dei punti interrogativi), e le uscite come *indefinite* (vedi figura).



Quando l'ingresso di \overline{Reset} è *attivato*, la rete si porta nello stato (a), lo *stato di Reset* specificato dal diagramma ASM. Si ricordi che il \overline{Reset} ha un comportamento *asincrono* (può cambiare valore in *un istante qualunque, indipendente dal clock*), e la rete permane forzatamente nello stato (a) fino a quando il \overline{Reset} non viene *disattivato*.

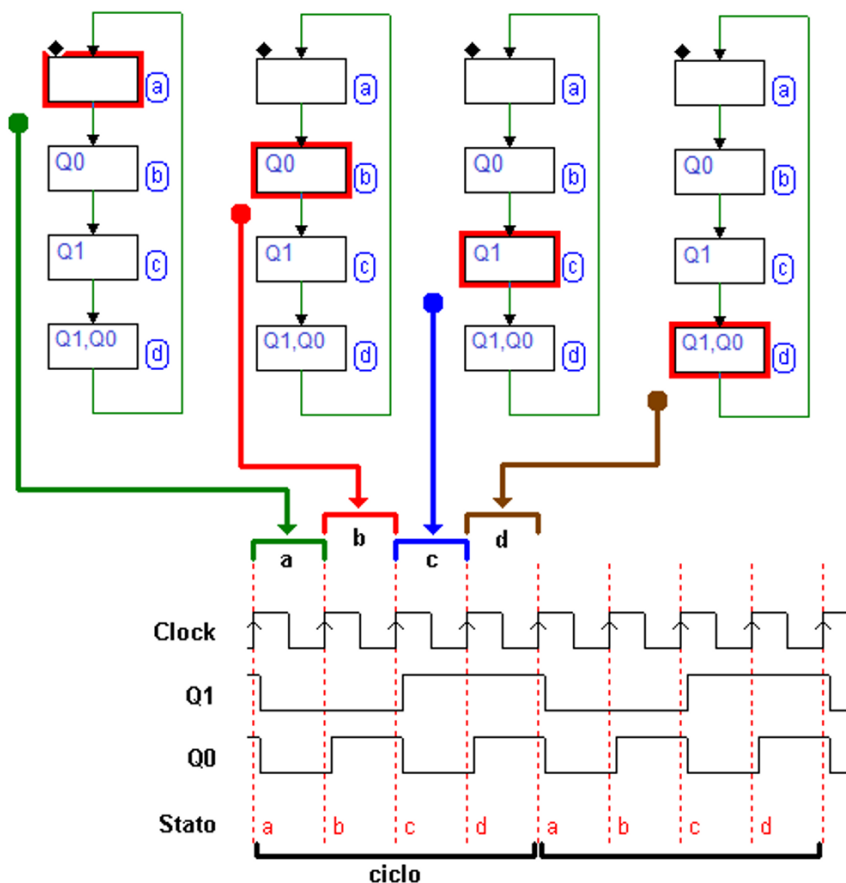
Quando questo avviene, al *primo fronte di salita* del clock, successivo alla disattivazione del \overline{Reset} , come si osserva in figura, la rete si porta nello stato (b), per poi proseguire lungo la sequenza prevista dal diagramma ASM.

Relazione tra diagramma ASM e diagramma temporale

Nella figura seguente è messa in evidenza la relazione tra *successione degli stati*, *diagramma temporale delle uscite* ed il clock. La macchina è sincrona: permane in ogni stato per la durata di un *ciclo di clock*.

Supponendo che, in un dato momento, la MSF si trovi nello stato (a), il diagramma ASM ci dice che il prossimo stato sarà (b): nel diagramma temporale si osserva il passaggio dallo stato (a) allo stato (b), che avviene con il *fronte di salita* del clock.

Con il passaggio allo stato (b), si attiva l'uscita Q_0 . Quindi, sul prossimo *fronte di salita*, la MSF passa nello stato (c), ove attiva l'uscita Q_1 , e così via:

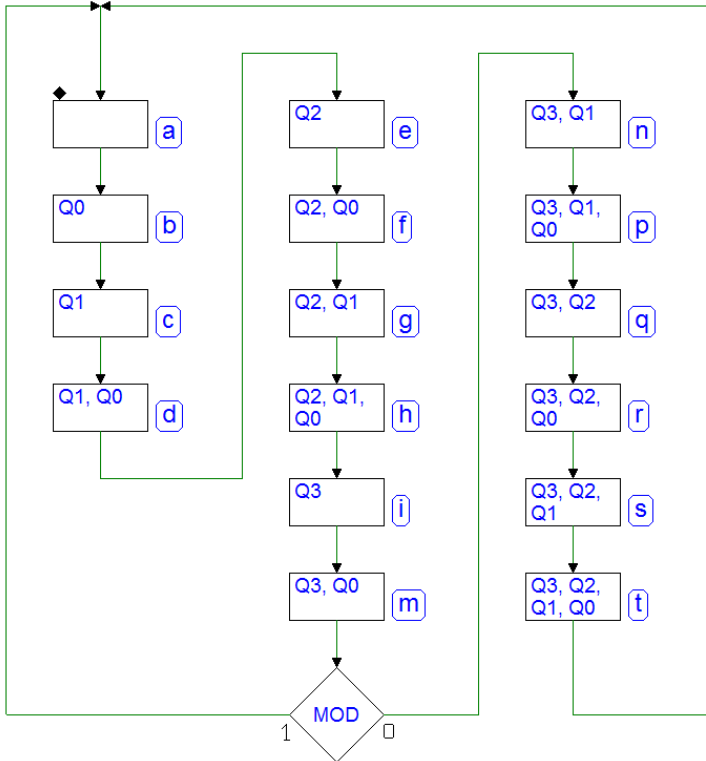


Il piccolo ritardo evidenziato nel diagramma temporale rappresenta i *ritardi fisici* della rete logica: il cambiamento delle uscite avviene “*dopo*” il fronte di salita del clock. Inoltre, la *ciclicità* della rete risalta bene dal diagramma temporale, ove si osserva la ripetizione, nel tempo, della sequenza generata.

7.2.2 Ingressi

Nella figura seguente presentiamo un secondo esempio di diagramma ASM, più esteso del precedente. Il diagramma rappresenta un *contatore a modulo variabile*, che può contare in *binario puro* (modulo 16), oppure in *codice BCD 8421*, (*Binary Coded Decimal*).

Nel diagramma sono presenti 16 stati, e un *blocco decisionale*. Il blocco decisionale, che incontriamo ora per la prima volta, tiene conto dell'ingresso denominato *MOD*:

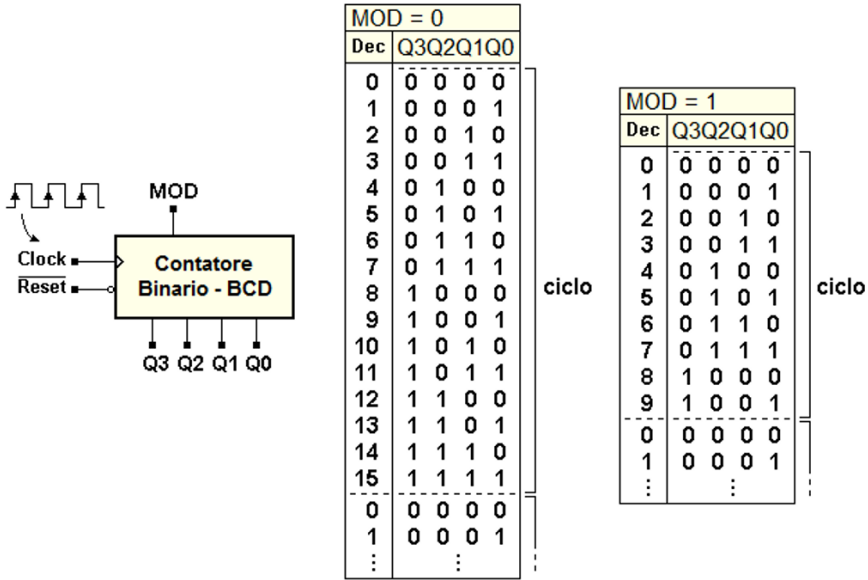


Se il valore di questo è 0, la macchina percorre, nell'ordine riportato nel diagramma, tutti i 16 stati, mentre, se il suo valore è 1, dopo lo stato (m), la macchina torna ad (a), percorrendo così una sequenza di 10 stati.

Nel primo caso, la successione degli stati produce sulle uscite $Q3$, $Q2$, $Q1$ e $Q0$ una sequenza *binaria crescente da 0 a 15*. Invece, nel secondo caso, le uscite assumono i valori dei codici BCD corrispondenti alle cifre decimali da 0 a 9.

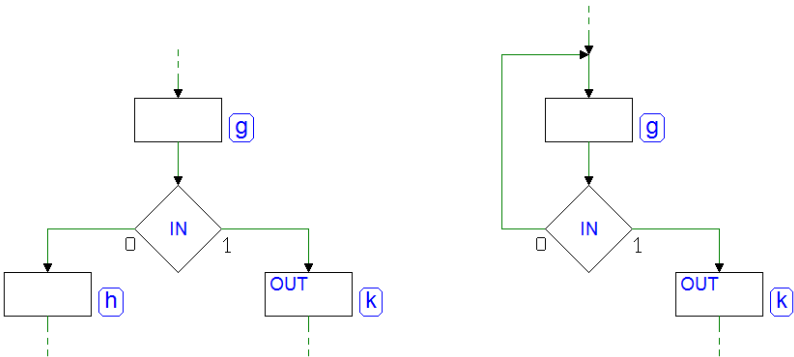
Nella figura seguente, sulla sinistra, è riportato lo schema a blocchi del contatore a modulo variabile. Oltre agli ingressi *Reset* e *Clock*, lo schema a blocchi mostra l'ingresso di impostazione della sequenza *MOD* e le quattro uscite $Q3$,

Q_2 , Q_1 e Q_0 . Nella figura sono anche mostrate, in forma di tabella, le due sequenze generate dal contatore, al variare della impostazione dell'ingresso:



Uso dei blocchi decisionali

Nei diagrammi ASM, gli ingressi della rete intervengono sulla *successione degli stati* attraverso i *blocchi decisionali*. Argomento di un blocco decisionale è normalmente una *variabile di ingresso*, oppure una *funzione booleana di più variabili di ingresso*. Un blocco decisionale consente di decidere tra *due percorsi logici*, a seconda del valore booleano del suo argomento. Nella figura qui sotto sono riportati due esempi tratti da generici diagrammi ASM:



Nell'esempio a sinistra, il blocco decisionale consente all'ingresso IN di definire quale sarà lo stato successivo allo stato (g): (h) se $IN = 0$, oppure (k) se

$IN = 1$ (si tratta di un caso analogo a quello dell'esempio precedente). Invece, sulla destra della stessa figura, lo stato (g) ha due possibili stati successivi: *se stesso*, oppure lo stato (k). Se $IN = 0$, la macchina resta nello stato (g): l'arrivo del prossimo fronte di clock non produrrà cambiamenti di stato. Se $IN = 1$, all'arrivo del fronte attivo la macchina passerà nello stato (k). La macchina permane nello stato (g) *in attesa* che l'ingresso IN passi a 1, prima di continuare le operazioni. Si dice che, in (g), la rete è in uno *stato di attesa*.

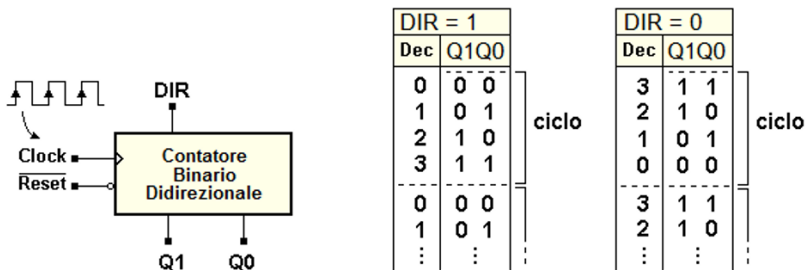
Significato temporale dei blocchi di stato e decisionali

Al *blocco di stato* è associato un significato temporale importante. Lo scorrere del tempo è infatti associato agli stati: alla permanenza in un determinato stato corrisponde una certa *durata temporale*, e la successione degli stati *evolve nel tempo*. Nelle macchine sincrone gli stati si succedono *cadenzati dal clock*, per cui la permanenza in un determinato stato avrà la durata, a seconda dell'algoritmo, di *un ciclo di clock* o di un suo *multiplo*.

Al *blocco decisionale*, invece, non è associato un significato temporale, ma soltanto *logico*. Facendo riferimento all'esempio precedente del contatore a modulo variabile, quando il contatore si trova nello stato (m), l'ingresso MOD decide quale sarà lo stato in cui spostarsi al successivo fronte attivo del clock. L'operazione svolta dal blocco decisionale è quella logica di *discriminare il percorso* verso lo stato successivo: (a) oppure (n). Gli effetti della decisione si concretizzano *all'atto della transizione* nello stato successivo.

Esempio: contatore binario avanti/indietro

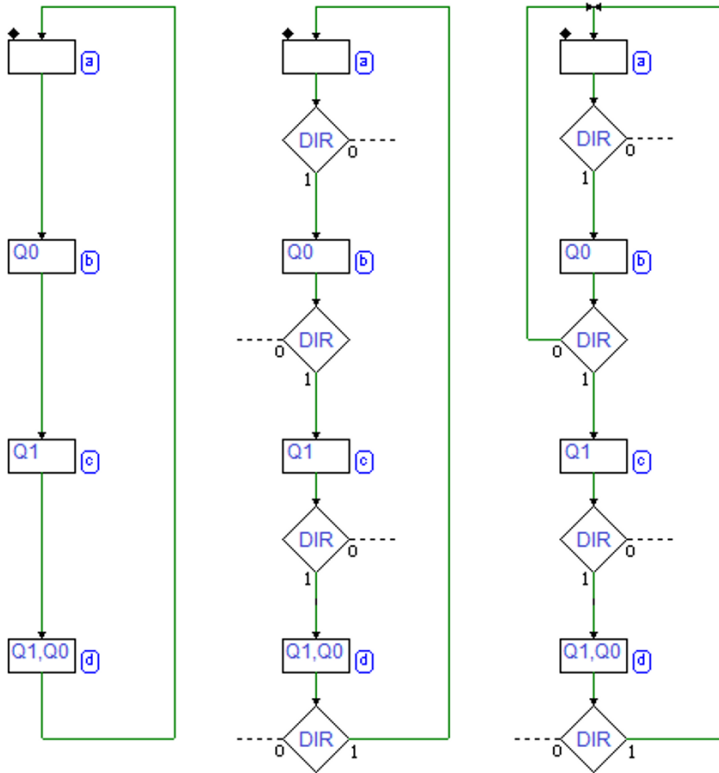
Abbiamo visto in precedenza la MSF di un semplice *contatore binario a 2 bit*. Il contatore era progettato per generare una sequenza binaria crescente. Vogliamo ora descrivere, in modo comportamentale, un dispositivo *bidirezionale*, in grado di contare non solo *in avanti*, ma anche *all'indietro*. Aggiungiamo, al disegno del componente visto in precedenza, un ingresso di comando, che denominiamo DIR , come mostrato nella figura qui sotto:



Tale ingresso avrà il compito di imporre il conteggio in avanti, se a 1, o indietro, se a 0: nella figura sono riportate, per chiarezza, anche le sequenze richieste, al

variare dell'ingresso *DIR*. Si vuole, come ulteriore specifica, che il conteggio possa *cambiare direzione* in qualunque stato il dispositivo si trovi.

Procediamo per gradi, modificando il precedente contatore in avanti, il cui diagramma ASM è riportato nella figura seguente, sulla sinistra. La sequenza dei quattro stati corrisponde al conteggio *in avanti*. Per permettere anche il conteggio *all'indietro*, inseriamo un blocco decisionale per ogni stato, in modo da condizionare la sequenza al valore dell'ingresso *DIR*:

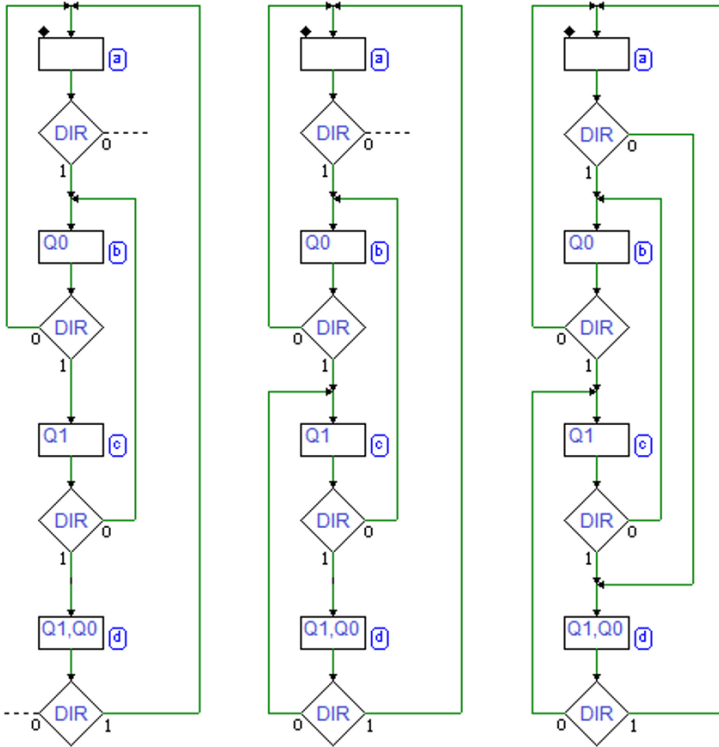


Nella figura, al centro, si vedono i quattro blocchi condizionali che sono stati aggiunti, ma il diagramma è ancora incompleto: non sono stati ancora disegnati i percorsi logici relativi al valore 0 di *DIR*. Si vede che, se il valore di *DIR* è pari a 1, la sequenza degli stati procede *contando in avanti*.

Aggiungiamo i percorsi logici mancanti: dobbiamo fare in modo che, se *DIR* = 0, il conteggio *decrementi* di una unità ad ogni ciclo di clock. In altre parole, si vuole che la sequenza degli stati proceda in senso inverso. Ad esempio, nella parte a destra della figura, si è tracciato il percorso logico che, se *DIR* = 0, porta la macchina nello stato (a) a partire dallo stato (b).

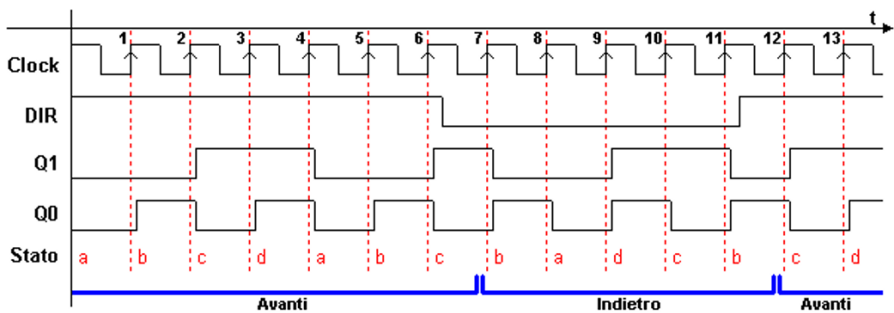
Nella figura seguente vengono aggiunti, uno dopo l'altro, i percorsi relativi al conteggio all'indietro: a sinistra, da (c) a (b) e, al centro, da (d) a (c). Per

completare la sequenza, è necessario ora individuare il percorso, per $DIR = 0$, relativo allo stato (a): se nel conteggio avanti si passa da (d) ad (a), nel caso della sequenza all'indietro dovremo passare, ovviamente, da (a) a (d).



Il diagramma ASM completo è riportato nella parte destra della figura. Si noti che il comportamento della macchina nello stato (a), da un punto di vista topologico, è analogo a quello di tutti gli altri stati, nonostante che, da un punto di vista grafico, possa sembrare diverso.

Nella figura qui sotto è mostrata una simulazione nel tempo del contatore:

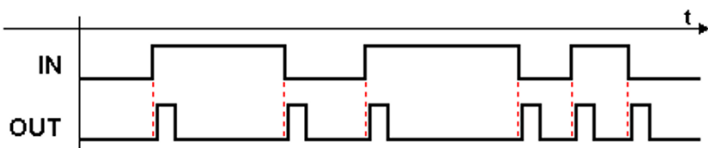


Si osservano gli ingressi di *Clock*, *DIR*, e le uscite *Q1* e *Q0*, oltre all'indicazione dello *Stato* della MSF. In basso, il conteggio è rappresentato sotto forma di scala, dove l'altezza dei gradini è proporzionale al numero generato. I fronti attivi del clock sono stati numerati per facilitare la lettura della figura.

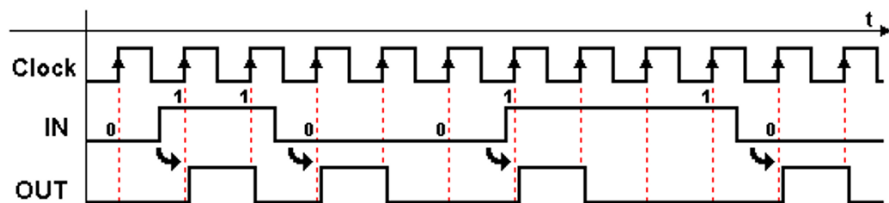
Sui primi fronti attivi del clock, numerati in figura da (1) a (6), l'ingresso *DIR* è alto, per cui la sequenza di conteggio risulta *in avanti*. Dal fronte (7), fino al fronte (11), l'ingresso *DIR* è basso, per cui il contatore procederà *all'indietro*, per poi tornare in avanti a partire dal fronte (12).

Esempio di progetto: un rivelatore di fronti

Un dispositivo *rivelatore di fronti* (*Edge Detector*), come dice il suo nome, è in grado di segnalare che il suo ingresso ha cambiato valore (da '0' è passato a '1', o viceversa). Nella figura qui sotto è esemplificato, con un tracciato temporale, il suo comportamento. Tutte le volte che l'ingresso *IN* presenta un *fronte* (di salita o di discesa), il rivelatore segnala l'evento generando un impulso sull'uscita *OUT* (un dispositivo di questo genere è anche chiamato, sia pur impropriamente, *derivatore*):

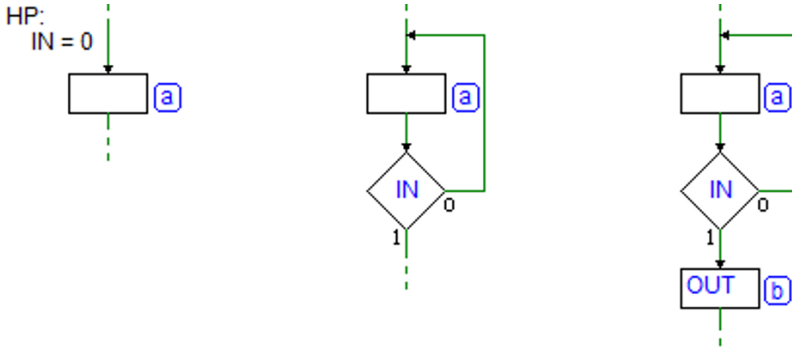


Il tracciato qui rappresentato è solamente indicativo: infatti, il preciso andamento temporale dell'uscita dipenderà dalla particolare realizzazione che sceglieremo. Se ci riferiamo al modello della macchina di *Moore*, la sua uscita *OUT* sarà sincrona con il clock e, di conseguenza, l'impulso generato avrà la durata minima di un ciclo di clock. Nella figura seguente è mostrato il tracciato temporale, corretto alla luce delle scelte ora impostate:



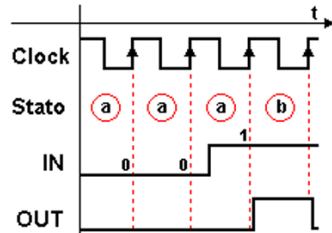
Ad ogni transizione dell'ingresso *IN* ($0 \rightarrow 1$ oppure $1 \rightarrow 0$), viene generato un impulso della durata di *un ciclo* di clock. Notiamo che, a causa della asincronicità dell'ingresso rispetto al clock della macchina, dovremo accettare l'inevitabile e casuale tempo di attesa tra l'effettiva transizione dell'ingresso e la generazione dell'impulso in uscita. Il tempo di attesa massimo, comunque, non supererà mai la durata di un ciclo di clock. Ricaviamo ora, a partire dalle specifiche qui discusse, il diagramma ASM del rivelatore di fronti.

Per iniziare, è necessario ipotizzare che la MSF si trovi “in un certo stato”, e che l’ingresso abbia “un preciso valore”, in modo consistente con i dati di progetto. A tale fine supponiamo che, in un certo momento, l’ingresso IN valga 0 e la macchina si trovi in uno stato (a) in cui l’uscita non è attiva (vedi figura seguente, a sinistra):



In questo stato la macchina deve attendere l’unico evento possibile, cioè la variazione dal basso verso l’alto di IN : aggiungiamo quindi un blocco decisionale che coinvolga tale ingresso (al centro nella figura), disegnandolo in modo che la macchina rimanga nello stato (a), in assenza di cambiamenti dell’ingresso. Quando l’ingresso cambia (passando a 1), la macchina deve segnalarne l’avenuta transizione, attivando l’uscita OUT : per ottenere questo è necessario aggiungere un nuovo stato (b), che genererà per un ciclo di clock l’uscita OUT (a destra in figura).

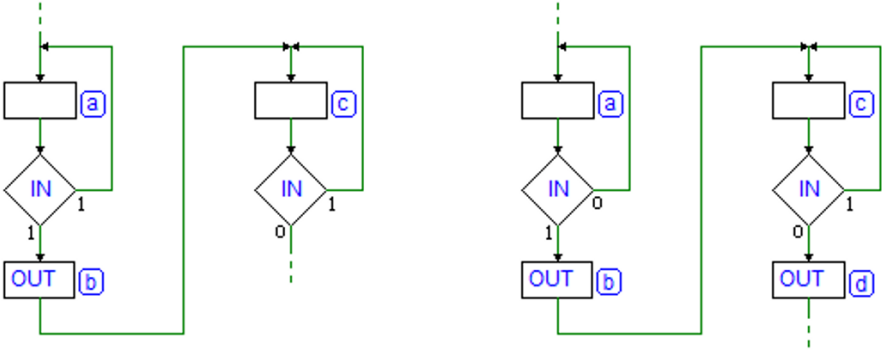
E’ utile soffermarci un momento sull’evoluzione nel tempo dei segnali coinvolti, in relazione con il diagramma ASM che stiamo costruendo. Nella figura qui a lato è riportata la sequenza temporale dei segnali relativi al passaggio tra gli stati (a) e (b). Nel tracciato è messa in evidenza la permanenza nello stato (a) per $IN = 0$, nei primi cicli di clock.



Si osserva poi la transizione in (b), sul fronte di salita del clock, dopo che IN è passato al valore 1; l’uscita OUT è attivata durante la permanenza in (b).

Nel continuare a disegnare il diagramma ASM, dobbiamo tenere presente che l’uscita OUT deve durare solo un ciclo di clock, per cui lo stato (b) deve essere seguito necessariamente da un altro stato, privo dell’uscita OUT . Inoltre, dato che l’ingresso IN è ora ad 1, in questo nuovo stato la macchina dovrà anche attendere il successivo passaggio di IN a 0.

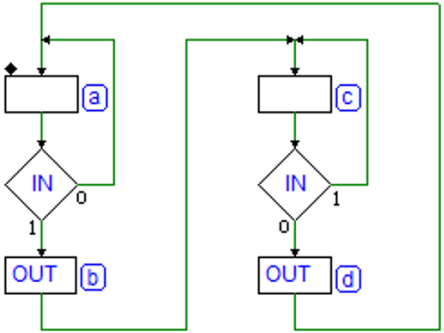
Aggiungiamo quindi lo stato (c), nel quale la macchina attenderà, questa volta, la transizione dell’ingresso verso il basso (vedi figura seguente, a sinistra).



Quando l'ingresso passerà a 0, la macchina dovrà produrre nuovamente l'uscita *OUT*. Come nel caso precedente, per attivare l'uscita dovremo cambiare stato, per cui aggiungiamo lo stato (d) a seguire lo stato (c) per *IN* pari a 0 (a destra, in figura).

Si noti la forte analogia tra la coppia di stati (a), (b) e la coppia (c), (d): il comportamento della macchina è simile, ma la differenza nel valore dell'ingresso impone *due sequenze distinte* di stati.

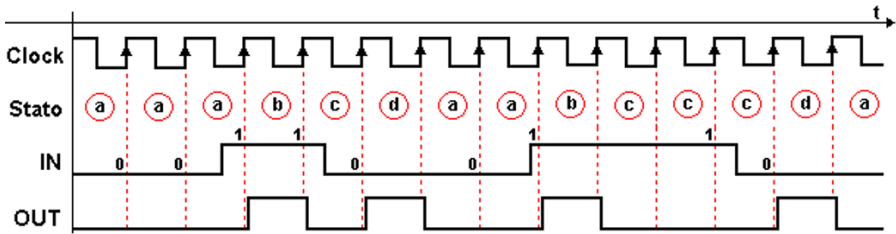
Dopo la generazione di *OUT* in (d), la macchina dovrà nuovamente attendere la transizione di *IN* verso l'alto: questo comportamento corrisponde proprio a quello dello stato (a) che avevamo inizialmente introdotto partendo dall'ipotesi di avere l'ingresso a 0. Richiudiamo quindi (d) sullo stato (a), come nella figura qui sotto, completando quindi il diagramma. Non ci sono più *percorsi da richiudere* e l'ipotesi di partenza circa lo stato (a) è stata soddisfatta:



Resta da decidere quale sarà lo *stato di Reset* in cui andrà la macchina al momento della inizializzazione. Appare intuitivo che gli stati (a) e (c) possano essere i più adatti allo scopo, in quanto in essi la MSF non attiva l'uscita e rimane semplicemente in attesa dell'ingresso.

Gli stati in cui si attendono eventi senza generare uscite sono anche chiamati "*stati di quiete*" ("*idle state*"). In mancanza di ulteriori specifiche, si è scelto lo stato (a), indicandolo con il rombo in alto a sinistra (vedi figura).

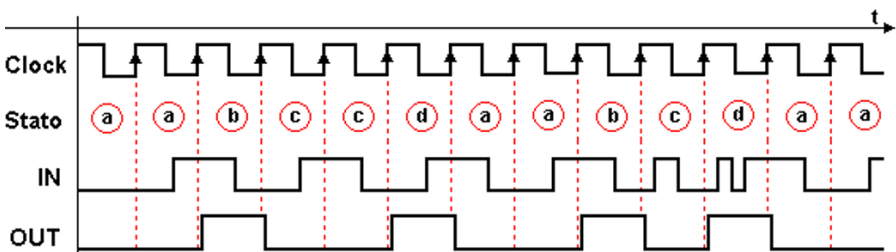
Qui sotto riproponiamo il tracciato temporale del funzionamento del rivelatore di fronti, riprendendo la sequenza di ingresso proposta in precedenza, ma il tracciato è ora completato dalla sequenza degli stati assunti dalla MSF:



Se osserviamo attentamente il tracciato temporale, e il diagramma ASM, ci accorgiamo che l'andamento del segnale di ingresso deve *sottostare ad alcune restrizioni*, affinché la MSF funzioni correttamente. In breve, le variazioni dell'ingresso devono essere sufficientemente distanziate tra di loro.

Nella prima parte del tracciato qui sopra, dove il segnale di ingresso viene letto alto per due cicli consecutivi, la macchina, dopo avere rilevato il fronte di salita, fa ancora in tempo ad accorgersi del fronte di discesa, e l'andamento dell'uscita *OUT* è corrispondente con quanto ci si aspetta.

E' una situazione al limite delle possibilità di questa MSF: infatti, se il segnale di ingresso varia più frequentemente, la macchina non riesce più ad inseguire il suo andamento, producendo una sequenza di uscita errata (cioè, non rispetta le specifiche date), come nell'esempio della figura seguente:



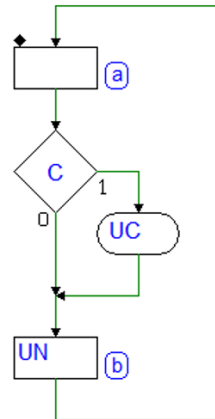
Si osserva che non è più generato l'impulso su *OUT* per ogni fronte del segnale di ingresso. Addirittura, nell'ultima parte del tracciato, dove l'esempio mostra un segnale di ingresso che cambia più volte all'interno dello *stesso ciclo di clock*, le transizioni risultano *totalmente invisibili*.

Questo genere di problemi sono normali nella progettazione digitale, e sono legati al tipo di operazione da svolgere, all'algoritmo utilizzato e, principalmente, alla frequenza del clock. In questo caso, il rivelatore di fronti funzionerebbe regolarmente se la frequenza di clock fosse aumentata adeguatamente.

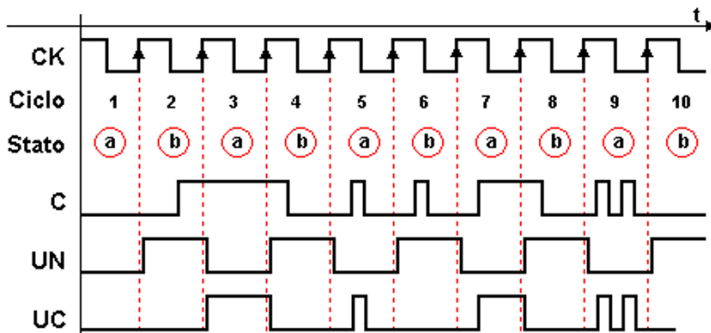
7.2.3 Uscite condizionate

Oltre al blocco di stato e al blocco condizionale, sufficienti per descrivere le macchine di Moore, nei diagrammi ASM si usa anche un terzo blocco: *l'uscita condizionata*. Nella macchina di Mealy, come abbiamo visto, le uscite possono dipendere non soltanto dallo stato, come in quella di Moore, ma anche *direttamente dagli ingressi*, in modo combinatorio.

I blocchi di uscita condizionata consentono di descrivere, nel diagramma ASM, questo tipo di dipendenza. La macchina descritta dal diagramma ASM qui a lato possiede due soli stati, (a) e (b). Il blocco di uscita condizionata è rappresentato da un rettangolo arrotondato sui fianchi e contiene, al suo interno, il nome dell'uscita (*UC* nell'esempio). Il blocco decisionale che valuta l'ingresso *C* non è coinvolto, in questo esempio, nel passaggio di stato: se osserviamo i percorsi che fuoriescono dal blocco, qualunque sia il valore dell'ingresso, il prossimo stato di (a) sarà sempre (b), così come il prossimo stato di (b) è (a).



Il blocco decisionale è funzionale, invece, al *blocco di uscita condizionata*: l'uscita *UC* è generata quando la macchina si trova nello stato (a), se e solo se $C = 1$. Il diagramma temporale mostrato nella figura seguente esemplifica questo comportamento:



Nel tracciato si osserva, coerentemente con il diagramma ASM, che nei cicli di clock 2, 4, 6, 8 e 10 l'uscita *UC* non è generata, poiché in quei cicli la macchina si trova nello stato (b). Negli altri cicli, invece, la macchina si trova nello stato (a), per cui l'attivazione di *UC* dipenderà dal valore dell'ingresso:

1. nel ciclo 1, l'ingresso *C* è basso per tutto il ciclo, per cui *UC* non si attiva;
2. nel 3, invece, *C* è alto per tutto il ciclo, per cui *UC* è attiva per tutta la durata dello stato;

3. nel ciclo 5, l'ingresso C è attivo solo per un breve intervallo di tempo, e l'uscita UC è attivata di conseguenza (come si osserva, l'andamento nel tempo di UC ricopia quello dell'ingresso C);
4. nel ciclo 7 il comportamento è simile al precedente, con la differenza che UC si attiva quando l'ingresso C va alto, per poi azzerarsi con il passaggio nello stato (b), anche se C permane alto, poiché in (b) l'uscita UC non è prevista;
5. nel ciclo 9, infine, il comportamento è analogo a quello del ciclo 5.

Per confronto, nel tracciato compare anche l'uscita di stato UN (non condizionata), attivata ogni volta che la macchina si trova nello stato (b). Poiché nel tempo i due stati (a) e (b) si alternano ad ogni ciclo di clock, UN mostra un andamento periodico.

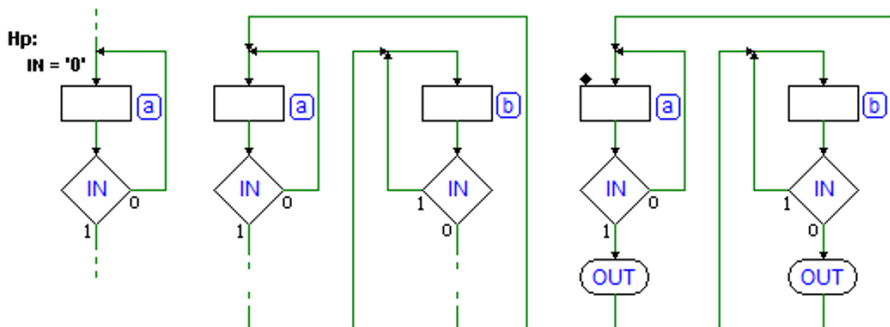
Esempio: un nuovo rivelatore di fronti

Abbiamo in precedenza progettato il *rivelatore di fronti* come macchina di Moore, utilizzando 4 stati. Proviamo a ripensare tale dispositivo alla luce delle possibilità offerte dalle macchine di Mealy, utilizzando cioè le *uscite condizionate*. Vedremo i vantaggi e gli svantaggi di questo nuovo approccio.

Con il precedente dispositivo, l'uscita veniva attivata in sincronia con il clock, per cui si produceva un'attesa tra l'effettivo arrivo del fronte del segnale di ingresso e la generazione dell'uscita. Tale attesa era pari, al massimo, alla durata di un ciclo di clock.

Con la macchina di Mealy, invece, abbiamo la possibilità di controllare le uscite direttamente dagli ingressi: vogliamo sfruttare questa caratteristica per ottenere un dispositivo che sia in grado di generare l'impulso in uscita non appena l'ingresso varia, senza aspettare il fronte attivo del clock.

Nella figura qui sotto (a sinistra), iniziamo a impostare la soluzione, ipotizzando uno stato (a) nel quale si entra con l'ingresso IN a 0, e nel quale si rimane in attesa del suo passaggio a 1. Al centro, in figura, aggiungiamo un secondo stato (b), simmetrico all'altro, per gestire il caso opposto, lasciando tratteggiati, per il momento, i percorsi di passaggio tra i due stati.

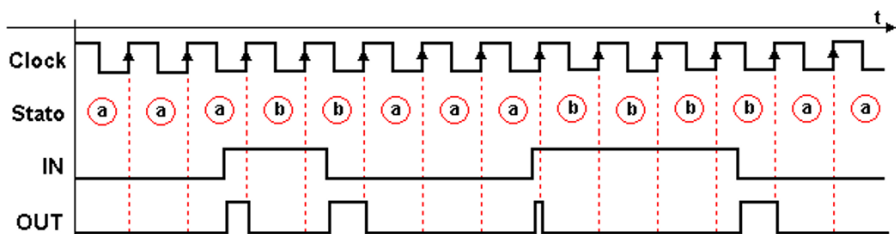


Per quanto sia ancora priva di uscite, la macchina, così impostata, svolge già un compito importante, che ritroveremo spesso in altri casi: è in grado di *inseguire* il segnale di ingresso. La macchina, infatti, permane in uno di due stati distinti, ciascuno corrispondente ad un preciso valore dell'ingresso, e passa dall'uno all'altro, in modo sincrono, al suo variare.

Aggiungiamo allo stato (a) l'uscita condizionata *OUT* (lato a destra della figura precedente). Nello stato (a), in attesa che *IN* vada alto, l'uscita non è generata: ma, non appena *IN* passa ad 1, l'uscita *OUT* è attivata. Completiamo il diagramma ASM introducendo l'uscita condizionata *OUT* in (b), attivata però dalla condizione $IN = 0$.

E' importante osservare che i blocchi *decisionali* esprimono sia le condizioni per la transizione tra gli stati, che la funzione logica che lega *OUT* all'ingresso *IN*. In effetti, un diagramma ASM che contenga blocchi di uscita condizionata descrive, con un unico disegno, *due differenti aspetti* del comportamento della MSF. Il primo è quello del *passaggio di stato*, che non è influenzato dai blocchi di uscita condizionata. Il secondo è quello delle *uscite condizionate*, che vengono espresse sotto forma di funzioni combinatorie degli ingressi e dello stato in cui sono istanziate.

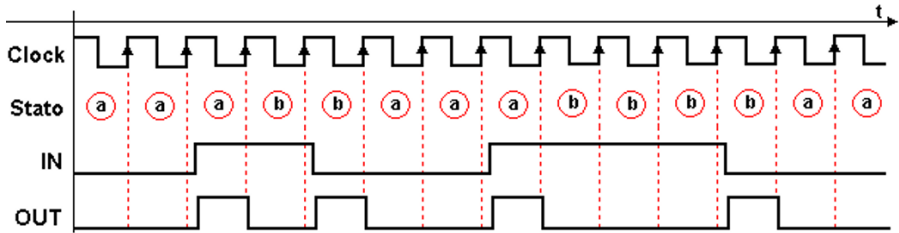
A titolo di confronto, il seguente diagramma temporale esemplifica il comportamento della uscita condizionata *OUT* in funzione della stessa sequenza di ingresso utilizzata per il test del rivelatore di fronti con uscite di stato:



All'arrivo della prima transizione, la macchina è nello stato (a): il cambiamento del valore dell'ingresso produce l'attivazione immediata dell'uscita *OUT*. Questa permane attiva fino al passaggio nello stato (b). In (b) l'uscita *OUT* è condizionata ad $IN = 0$, per cui in tale stato si attiverà solo nel momento in cui l'ingresso ritornerà a zero. Un comportamento analogo si ha per le altre transizioni dell'ingresso rappresentate nella figura.

Tuttavia, nella sequenza temporale che abbiamo utilizzato, l'ingresso *IN* è *asincrono* rispetto al clock della macchina: la *durata* dell'uscita *OUT* risulta *non controllabile*, come messo in evidenza dal tracciato. Infatti, la terza transizione dell'ingresso dà origine ad un impulso di durata molto breve e, se la transizione avviene troppo a ridosso del fronte del clock, l'impulso potrebbe non essere generato (a causa dei ritardi fisici dei componenti), o essere troppo breve per essere leggibile (a causa dei tempi minimi di funzionamento).

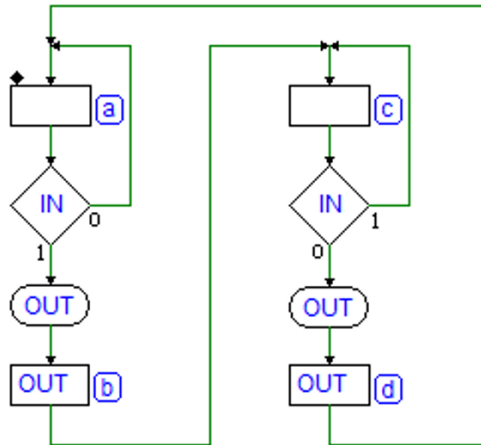
E' evidente che questa versione della macchina non si può utilizzare con ingressi di tipo *asincrono*. Un ingresso sincrono, invece, consente di evitare i casi limite, e di ottenere un segnale di uscita di durata sufficiente. Nel diagramma temporale della figura seguente il segnale *IN* è sincrono con il clock della macchina, e quindi la durata dell'uscita è costante:



In ultimo, osserviamo che la macchina impiega due soli stati. Generalmente una macchina di Mealy può avere bisogno di *meno stati* della corrispondente macchina di Moore, e risulta più compatta.

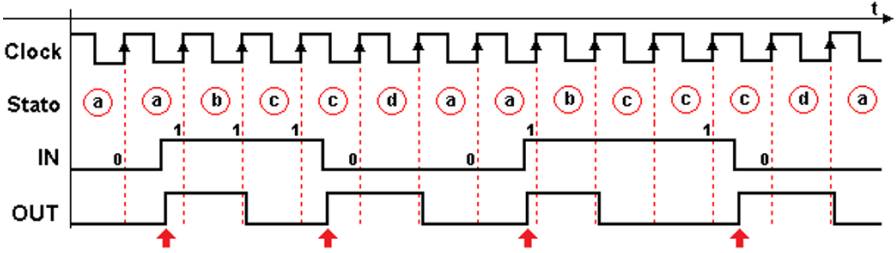
Esempio: terza versione del rivelatore di fronti

Proviamo ora a riconsiderare i vantaggi e gli svantaggi discussi, al fine di ottenere un'altra versione del rivelatore, che mantenga la *prontezza di risposta* delle uscite condizionate, ma che generi l'uscita per la durata di *almeno di un ciclo* di clock. Partiamo dal diagramma ASM della versione con quattro stati, ricavato in precedenza, e aggiungiamo due uscite condizionate, come si vede nella figura qui sotto:



Facciamo riferimento allo stato (a): in attesa che *IN* vada alto, l'uscita condizionata *OUT* non è attivata. Non appena *IN* passa ad 1, l'uscita condizionata *OUT* è portata ad 1 immediatamente, mentre siamo ancora in (a). Con il successivo fronte di clock, quando la MSF passa in (b), continuerà a generare *OUT*, in quanto definita come uscita di stato.

Nel diagramma temporale seguente, le frecce indicano gli istanti di attivazione dell'uscita *OUT*, *anticipati* rispetto alla versione di Moore (nella quale avremmo atteso il fronte di clock per attivare l'uscita):



Questa nuova realizzazione consente di generare un'uscita leggibile, di durata minima garantita, anche se l'ingresso *IN* ha un comportamento *asincrono*.

7.3 Esempi di costruzione di diagrammi ASM

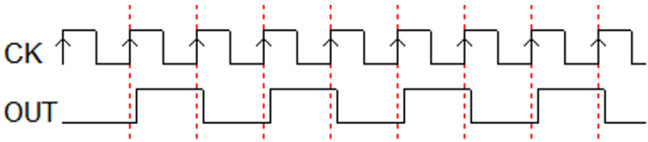
Presentiamo qui una raccolta di esempi di progetto di MSF sincrone, allo scopo di chiarire i concetti base che riguardano la costruzione dei diagrammi ASM, nel rispetto delle specifiche assegnate. Per adesso, negli esempi di questo capitolo, la MSF gestirà direttamente gli ingressi e le uscite del sistema che si vuole progettare.

7.3.1 Esempi introduttivi

Affrontiamo per ora il progetto di semplici MSF sincrone, le cui specifiche sono definite, oltre che a parole, anche tramite *diagrammi temporali*. Questo ci consentirà di comprendere la relazione che esiste tra *l'andamento nel tempo* dei segnali di ingresso/uscita e *l'evoluzione degli stati*.

Esempio 1

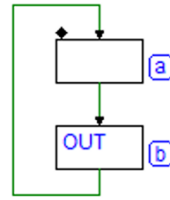
Progettare, usando i diagrammi ASM, una MSF che generi un segnale periodico *OUT*, sincrono con il clock *CK*, il cui valore sia alto per un ciclo e basso per il successivo, come si vede nella figura:



Soluzione

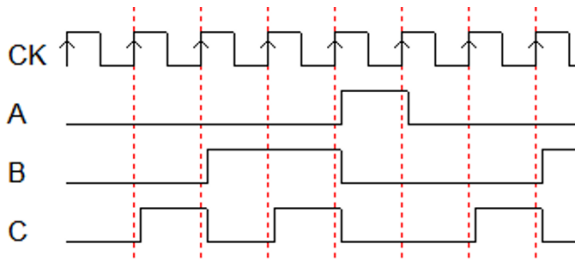
In una generica MSF sincrona *senza ingressi*, l'uscita cambia solo se cambia lo stato (modello di *Moore*).

Nel nostro caso, per alternare i due valori 0 e 1, ad ogni ciclo di clock, dovremo ciclicamente cambiare stato (vedi figura qui a lato).



Esempio 2

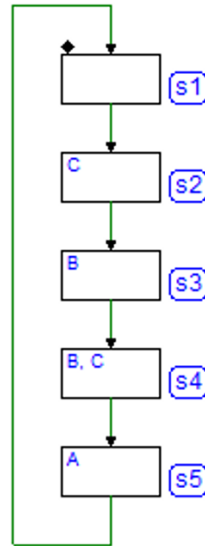
Tracciare il diagramma ASM di un contatore sincrono che generi indefinitamente la sequenza (ABC) 000, 001, 010, 011, 100, 000, etc.



Soluzione

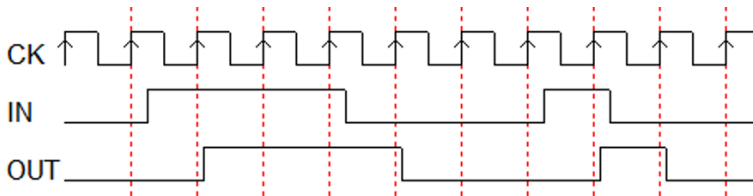
Anche in questo secondo esempio l'uscita sarà funzione solo dello stato (non ci sono ingressi), e di ciclo in ciclo le combinazioni richieste sono tutte diverse.

Quindi, dovremo introdurre nel diagramma ASM una sequenza di stati che avranno tutti una differente combinazione di uscite. La sequenza si ripeterà in modo ciclico.



Esempio 3

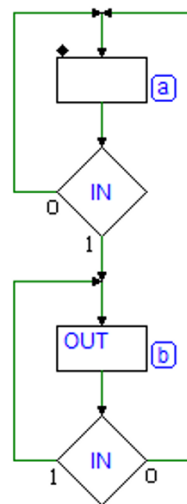
Progettare, usando i diagrammi ASM, una MSF con un ingresso *IN* (sincrono), che generi un segnale *OUT* sincrono, per tutto il tempo in cui l'ingresso *IN* è a 1, come esemplificato nella figura seguente:



Soluzione

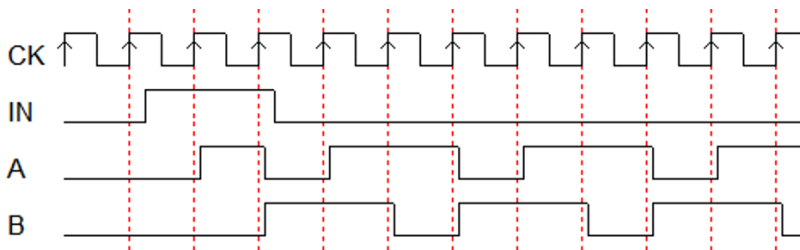
Affinchè l'uscita *OUT* sia sincrona con il clock, dovrà essere una uscita di stato, e utilizzeremo quindi il modello di Moore. L'ingresso sarà quindi valutato dalla MSF sul fronte di salita del clock. La soluzione è mostrata nella figura qui accanto.

Nello stato (a) attendiamo che l'ingresso vada a 1, senza attivare l'uscita *OUT*. Al cambiamento dell'ingresso, si cambia stato, generando l'uscita *OUT* e aspettando, questa volta, che l'ingresso ritorni a 0.



Esempio 4

Progettare, usando i diagrammi ASM, una MSF che alla transizione basso-alto dell'ingresso *IN*, sincrono con il clock *CK*, generi ininterrottamente la seguente sequenza di segnali: *A*, *B*, *AB* (dove *A* e *B* sono le due uscite della macchina):

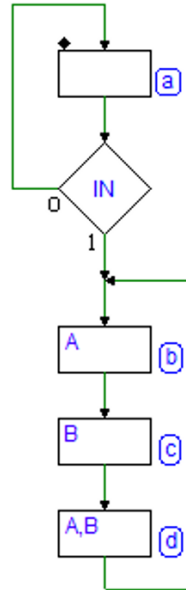


Soluzione

La sequenza di segnali, generata da una MSF sincrona, sarà ovviamente *sincrona* con il clock, come si osserva nel tracciato temporale: utilizzeremo quindi il modello di *Moore*.

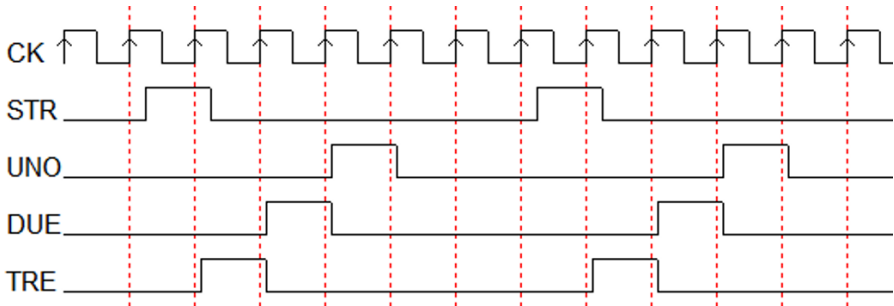
L'ingresso è valutato nello stato di attesa (a). Non appena l'ingresso si porta ad 1, si entra nella sequenza ciclica di stati (b), (c), (d). In ogni stato attiviamo una combinazione differente delle uscite, come richiesto dalle specifiche.

Si noti che la macchina può ritornare nello stato (a) soltanto a seguito della attivazione del Reset asincrono.



Esempio 5

Progettare una MSF usando i diagrammi ASM con un ingresso *STR* (*Start*) sincrono con il clock e tre uscite *UNO*, *DUE* e *TRE*. Si vuole che, all'arrivo di un impulso su *STR*, della durata di un ciclo di clock, la macchina conti alla rovescia da tre a uno attivando le omonime uscite. Il conteggio non può essere interrotto e termina con tutte le uscite a zero.



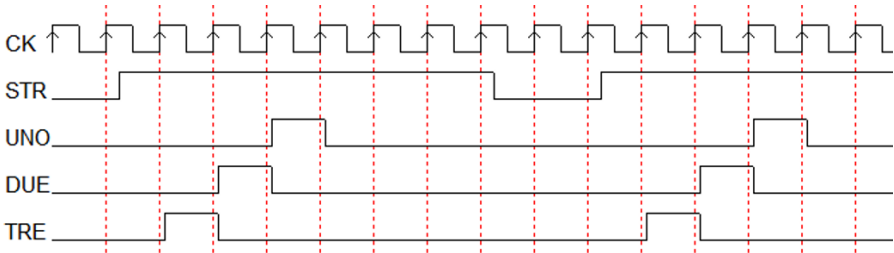
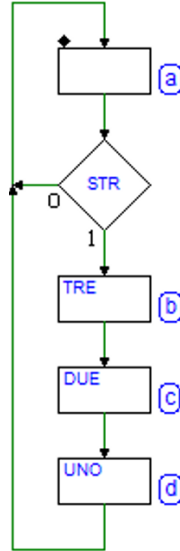
Soluzione

La soluzione è simile a quella dell'esempio precedente, con la differenza che questa volta lo stato iniziale (a) è compreso nella sequenza ciclica degli stati.

Vedi figura qui a lato. L'ingresso, di natura impulsiva, è valutato nello stato (a): si noti che se l'ingresso *STR* avesse un andamento non impulsivo ma *continuo*, la macchina funzionerebbe lo stesso, ma continuerebbe a generare la sequenza ciclica nel caso in cui *STR* rimanesse a uno per lungo tempo, per poi fermarla, nello stato (a), dopo la disattivazione dell'ingresso.

Esempio 6

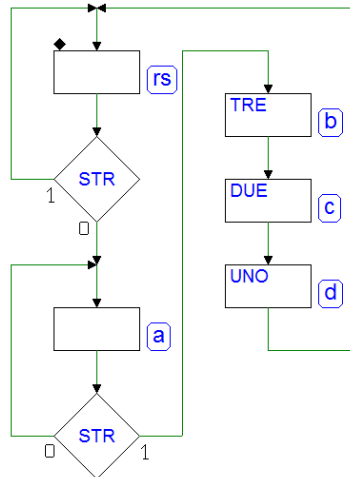
Progettare una MSF avente la stessa funzionalità di quella al punto precedente, ma con una diversa specifica di attivazione della sequenza, ad opera dell'ingresso *STR* (rappresentata nella figura qui sotto). Si vuole che la sequenza parta all'arrivo di una *transizione basso-alto* dell'ingresso *STR*. Anche in questo caso il conteggio non può essere interrotto e termina con tutte le uscite a zero.



Soluzione

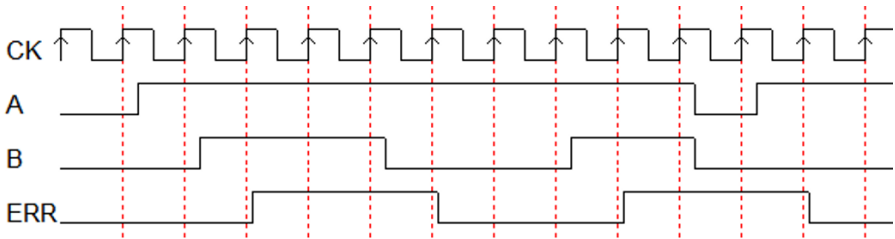
Ad un primo sguardo, la soluzione proposta per l'esempio precedente sembrerebbe funzionare anche per questo, ma non è così. Infatti, qui è richiesto che sia *solo la transizione dal basso ad alto* dell'ingresso ad attivare la sequenza.

Per essere sicuri di rivelare tale transizione, dobbiamo verificare prima che *STR* sia (o sia già tornato) a livello basso. Quindi aggiungiamo al diagramma dell'esempio precedente uno stato in più (rs), prima dello stato (a). Nello stato (rs) attendiamo (o verifichiamo) che *STR* sia già a 0, prima di passare al controllo opposto, nello stato (a).



Esempio 7

Si progetti, usando i diagrammi ASM, una MSF sincrona con due ingressi A e B sincroni con il clock, che generi un'uscita ERR ogni volta che entrambi gli ingressi vengono visti contemporaneamente alti. L'uscita ERR deve essere azzerata quando gli ingressi assumono valori differenti tra loro.

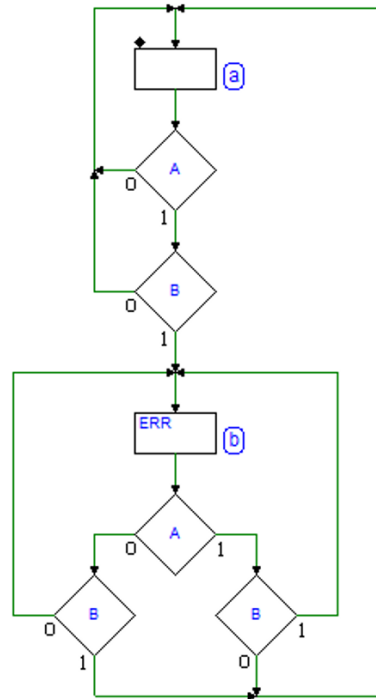


Soluzione

Scegliamo di progettare una MSF sincrona secondo il modello di Moore. In una macchina sincrona, la valutazione della *contemporaneità* di due ingressi si traduce nel fatto che saranno esaminati *nello stesso ciclo di clock*.

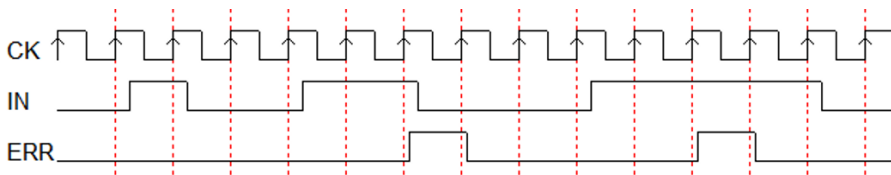
Nello stato di attesa (a) valutiamo *contemporaneamente* gli ingressi A e B : ci spostiamo nello stato (b) se entrambi valgono 1. Nello stato (b) generiamo l'uscita ERR e rivalutiamo nuovamente insieme gli ingressi A e B .

Per come sono stati collegati i percorsi dei blocchi decisionali, si rimane in (b) solo se A e B sono tra loro uguali; altrimenti, si torna nello stato (a).



Esempio 8

Progettare, usando i diagrammi ASM, una MSF che controlli un ingresso IN sincrono e che generi un segnale ERR della durata di un periodo del clock se IN resta alto per più di un ciclo di clock (vedi il tracciato temporale seguente).



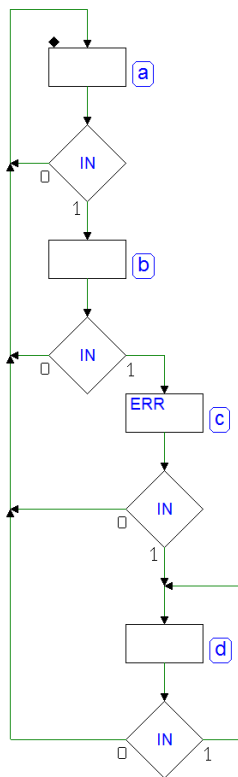
Soluzione

Scegliamo di progettare una MSF sincrona secondo il modello di Moore (vedi figura qui a lato). Nello stato di attesa (a) valutiamo l'ingresso *IN*, e ci spostiamo nello stato (b) se è a 1.

Nello stato (b) eseguiamo lo stesso controllo. Se anche nello stato (b) l'ingresso è letto a 1, generiamo l'uscita *ERR*, spostandoci nello stato (c), per segnalare che l'ingresso è rimasto alto per più di un ciclo di clock; altrimenti si torna nuovamente ad attendere l'arrivo di un prossimo 1.

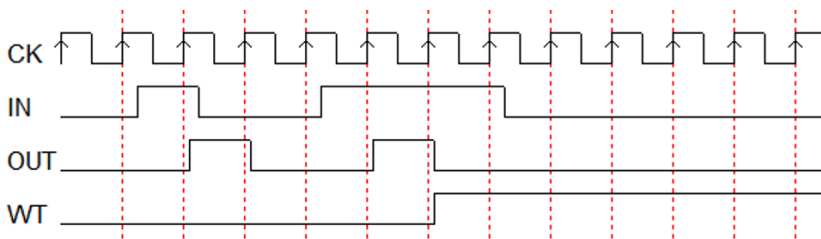
L'uscita *ERR* deve essere attivata per un solo ciclo di clock, per cui la generiamo solo nello stato (c).

In questo stato, controlliamo l'ingresso *IN*: se è tornato nel frattempo a 0, ritorniamo nello stato (a). Se *IN* fosse ancora a 1, invece, dovremo attendere che finalmente ritorni a 0, prima di tornare nello stato (a): infatti, se vi andassimo direttamente, rivaluteremmo nuovamente lo stesso segnale.



Esempio 9

Si progetti una MSF che generi un'uscita *OUT* alta quando l'ingresso *IN* è alto, ma per la durata di un ciclo di clock (vedi figura):



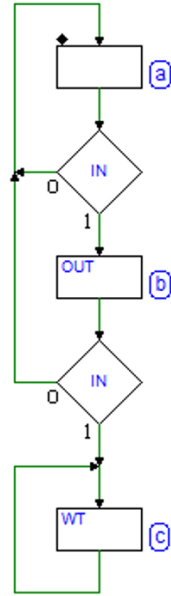
Se l'ingresso *IN* dura di più, la MSF attiva in modo stabile l'uscita *WT* (*Wait*), entrando in un ciclo infinito da cui esce solo se resettata.

Soluzione

Scegliamo di progettare una MSF sincrona secondo il modello di *Moore*. Come si vede nella figura qui accanto, nello stato di attesa (a), valutiamo l'ingresso *IN*, e ci spostiamo, se è alto, nello stato (b).

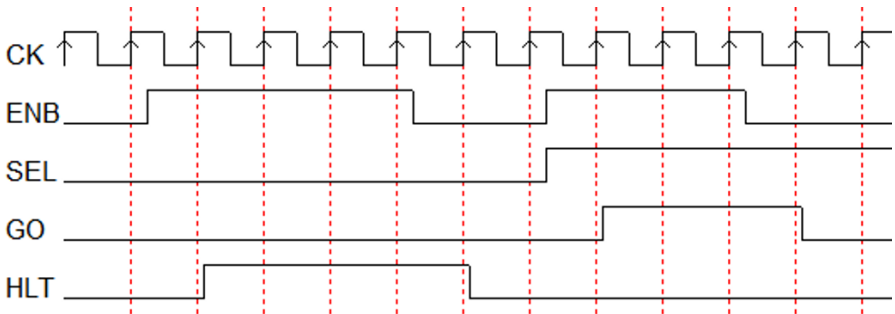
Nello stato (b) generiamo l'uscita *OUT* e, se l'ingresso è nuovamente cambiato, si torna nello stato (a) ad aspettare la prossima transizione.

Se l'ingresso *IN* è rimasto basso, la MSF si sposta definitivamente nello stato (c), generando l'uscita *WT*, in ciclo infinito.



Esempio 10

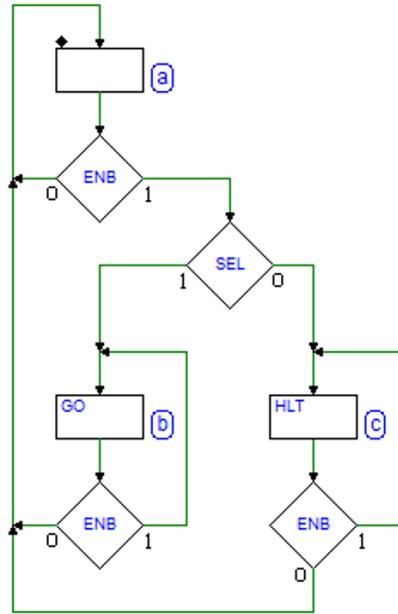
Progettare una MSF avente in ingresso due segnali *ENB* (*Enable*) e *SEL* (*Select*), sincroni con il clock *CK*, e due uscite *GO* e *HLT* (*Halt*). La MSF attende che *ENB* assuma il valore 1. Se *ENB* è a 1, la macchina sceglie se attivare, a seconda che *SEL* sia 1 o 0, rispettivamente *GO* o *HLT* per tutto il tempo in cui *ENB* rimane a 1. Quando *ENB* ritorna a 0, la macchina si rimette in attesa, senza generare uscite attive.



Soluzione

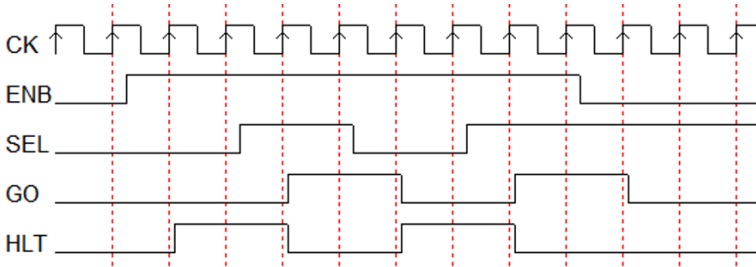
Realizziamo una MSF sincrona di Moore. Nello stato di attesa (a) valutiamo l'ingresso *ENB* e ci spostiamo, se è a 1, nello stato (b), se *SEL* è a 1, altrimenti in (c). Nello stato (b) generiamo l'uscita *GO*, rimanendo in attesa che *ENB* ritorni a 0. Nello stato (c), in modo simile, attiviamo *HLT*, e aspettiamo che *ENB* cambi valore.

Infine, al ritorno a 0 di *ENB*, la macchina ritorna nello stato (a).



Esempio 11

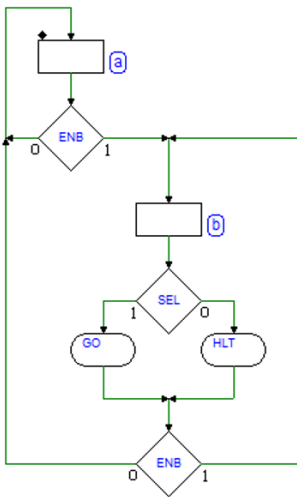
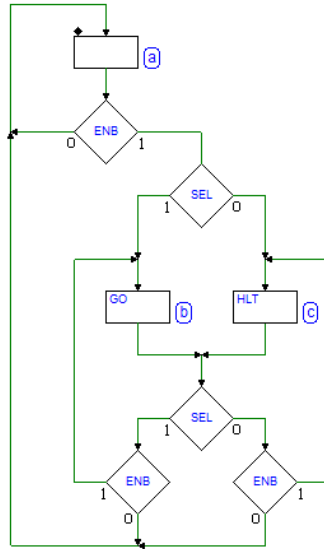
Stesse specifiche del precedente esempio, ma con una variante: se *ENB* è a 1, il valore dell'ingresso *SEL* consentirà di scegliere quale segnale viene attivato tra *GO* e *HLT*, mentre li stiamo generando:



Soluzione

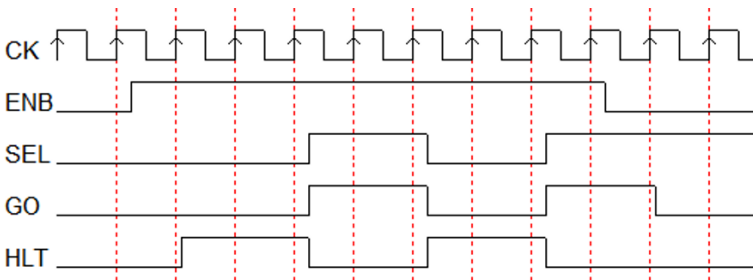
Ne realizziamo una prima versione utilizzando il modello di *Moore* (figura a destra). La macchina è in pratica identica alla precedente, ma con l'aggiunta, negli stati (b) e (c), del controllo dell'ingresso *SEL*.

A seconda del valore di *SEL*, il ciclo viene effettuato sullo stato (b), oppure (c).



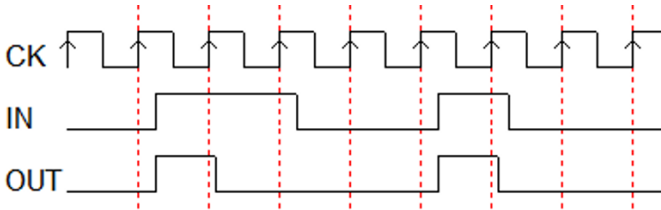
Nella figura a sinistra una seconda soluzione, che utilizza il modello di *Mealy*. La macchina possiede solo due stati e attiva *GO*, oppure *HLT*, a seconda del valore di *SEL*, mentre si trova nello stato (b).

Il funzionamento risulterà un poco diverso dal punto di vista dei tempi (si veda la figura qui sotto). Infatti, per ottenere la stessa sequenza vista nel diagramma precedente, il controllo di *SEL* deve essere posticipato di un ciclo di clock:

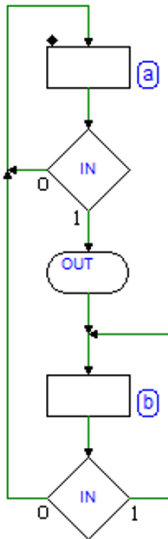


Esempio 12

Progettare, usando i diagrammi ASM, una MSF con un ingresso *IN* (sincrono rispetto al clock *CK*) ed un'uscita *OUT*, che si attiva alla salita di *IN* e rimane tale fino al fronte successivo di *CK*:



Soluzione



Dal momento che l'uscita deve attivarsi alla salita di *IN*, occorre utilizzare il modello di Mealy (vedi figura qui a sinistra).

Nello stato (a) attendiamo che *IN* passi da 0 a 1; l'uscita condizionata consente di generare *OUT* non appena *IN* diventa alto, mentre siamo ancora nello stato (a). All'arrivo che prossimo fronte di *CK*, la macchina si porta nello stato (b), dove non è istanziata alcuna uscita, per cui *OUT* viene disattivata non appena usciti dallo stato (a).

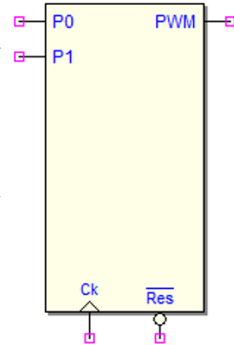
Dallo stato (b) si torna ad (a) sul fronte di *CK* successivo al ritorno a 0 di *IN*.

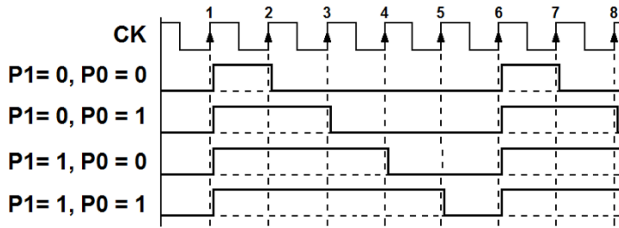
7.3.2 Generatore di impulsi a rapporto pieno/vuoto regolabile

Si vuole progettare una rete sequenziale sincrona (in termini di MSF) che generi, sull'uscita *PWM* ("Pulse Width Modulation"), una successione continua di impulsi aventi un rapporto pieno/vuoto (*Duty Cycle*) regolabile (vedi figura qui a destra).

Il periodo dell'uscita è fisso e pari a 5 volte quello del clock *CK*; il rapporto pieno/vuoto è regolato tramite gli ingressi *P1* e *P0*.

La figura seguente descrive l'andamento atteso dell'uscita, in dipendenza dagli ingressi *P1* e *P0*.

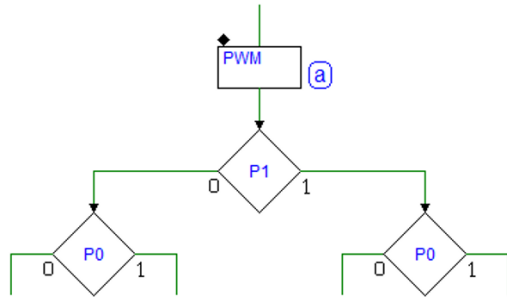




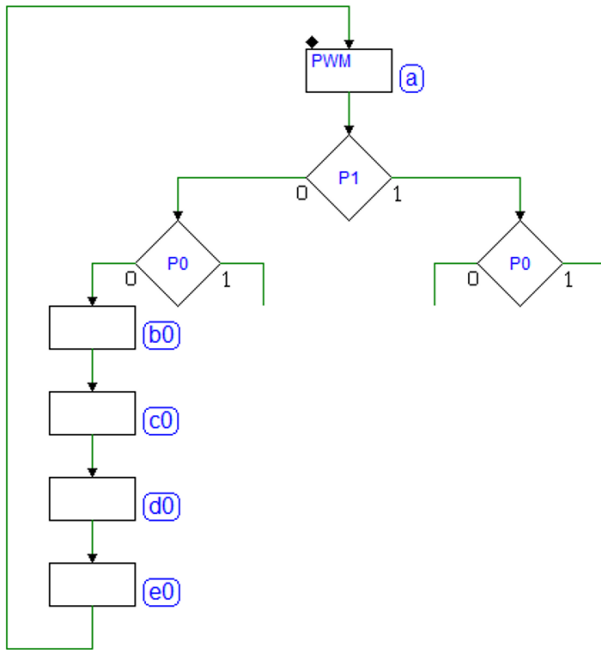
Soluzione

Impostiamo il diagramma introducendo uno stato (a) in cui attiviamo l'uscita *PWM* (figura a lato).

Questo stato è comune a tutte le sequenze e corrisponde, nella figura precedente, al ciclo di clock tra il fronte 1 e il 2.

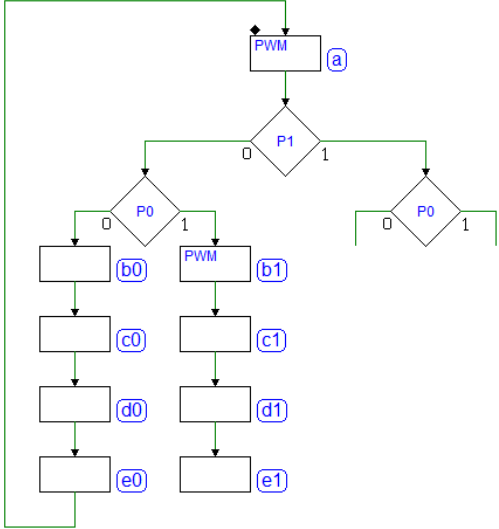


Dallo stato (a) si diramano 4 percorsi logici, corrispondenti alle 4 combinazioni di *P1* e *P0*. Completiamo il diagramma, per adesso, in relazione alla prima sequenza rappresentata nel diagramma temporale ($P1 = 0$ e $P0 = 0$):

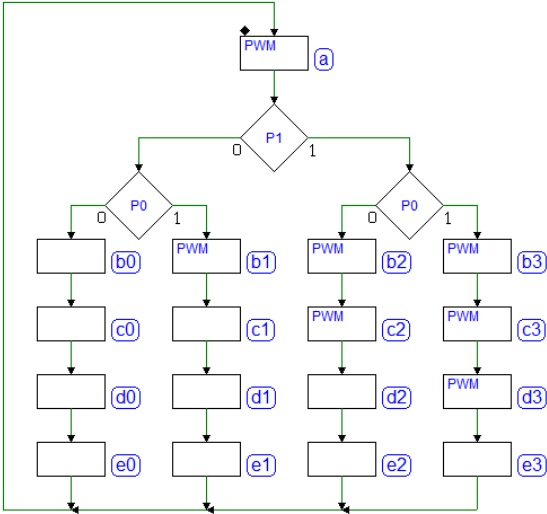


Abbiamo aggiunto 4 stati, senza uscite attive, che faranno trascorrere 4 cicli di clock prima di ritornare nello stato (a), in modo che il periodo sia in totale di 5 cicli di clock, come richiesto.

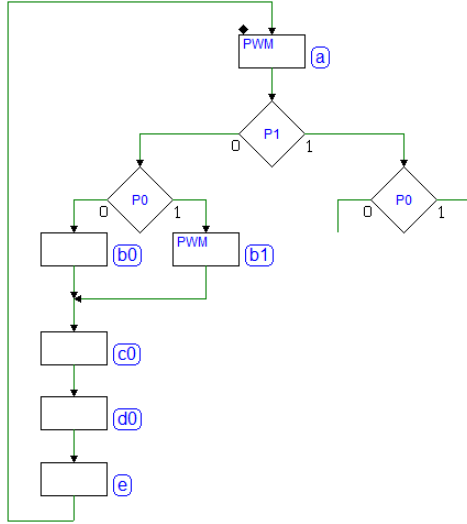
Se i due ingressi $P1$ e $P0$ valgono entrambi 0, quindi, la macchina genera PWM attiva per un ciclo di clock, nello stato (a), e non attiva per altri 4, negli stati (b0), (c0), (d0) ed (e0). Completiamo la seconda sequenza. Differisce dalla prima solo per uno stato (b1), nel quale attiviamo PWM :



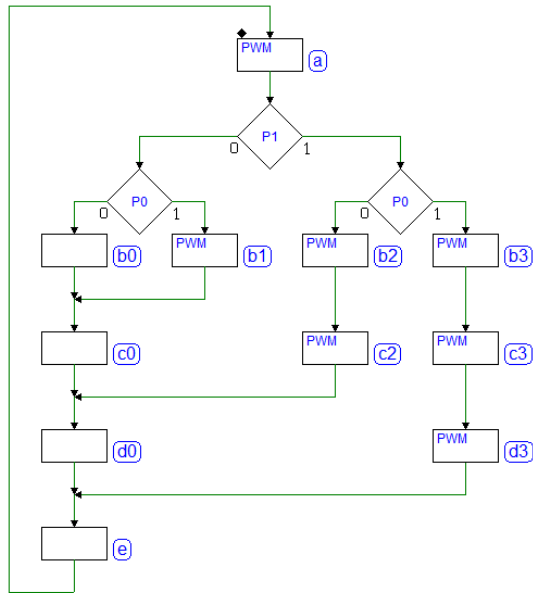
Richiudiamo questa sequenza e completiamo in modo simile le altre due, prestando attenzione a richiudere tutti i percorsi sullo stato comune (a):



Ci accorgiamo, a questo punto, di avere introdotto delle sequenze di stati inutili in quanto in parte eseguono le stesse funzioni. Per esempio, la sequenza di stati (c1)(d1)(e1) produce lo stesso risultato della sequenza (c0)(d0)(e0), ed entrambe terminano sullo stato (a) per cui, nel definire gli stati della seconda sequenza, potevamo procedere in questo modo:



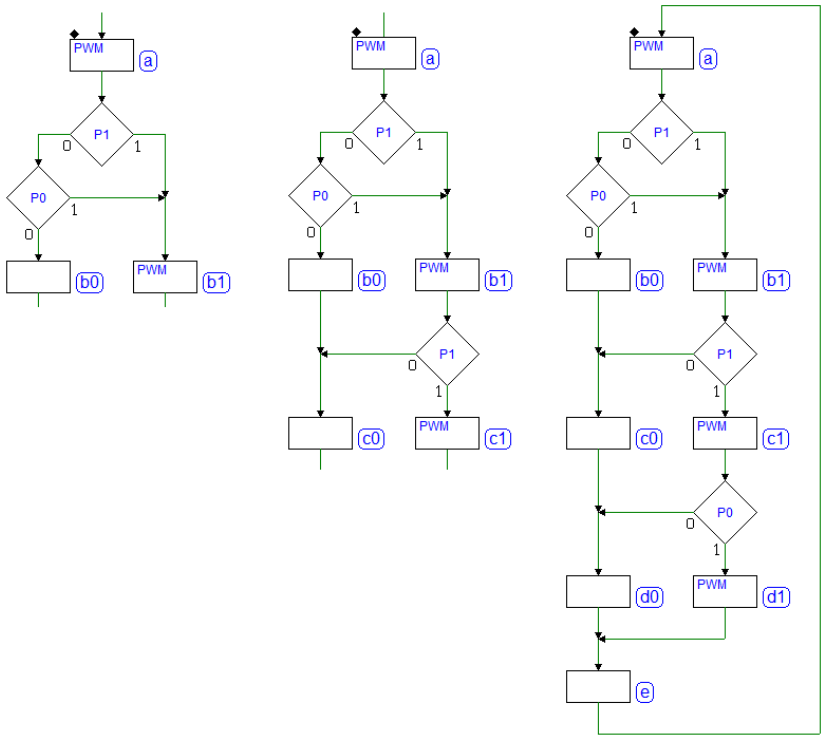
Abbiamo riutilizzato gli stati (c0)(d0)(e0) della prima sequenza. Procediamo quindi in modo simile per la terza e la quarta sequenza, ottenendo il risultato finale di questa seconda versione, decisamente migliore rispetto alla prima:



Potremmo fermarci qui. Si noti che il valore degli ingressi $P1$ e $P0$ è valutato solo nello stato (a). Un eventuale cambiamento degli ingressi, durante la generazione dell'impulso di larghezza variabile, non produrrà effetti se non all'avvio della generazione del successivo impulso: una scelta progettuale corretta.

Ipotizziamo quindi che le specifiche del problema ci dicano che $P1$ e $P0$ possano cambiare soltanto durante lo stato (a). Possiamo quindi permetterci di testare i loro valori anche negli stati successivi ad (a), dato che nel frattempo non cambiano. Partendo da questo presupposto, e analizzando le quattro possibili sequenze da generare, potremmo riuscire a ridurre ancora il numero di stati. Infatti, dopo lo stato (a), nel secondo ciclo della sequenza (tra i fronti 2 e 3) l'uscita PWM non deve essere generata se entrambi gli ingressi $P1$ e $P0$ valgono 0 (figura seguente, a sinistra).

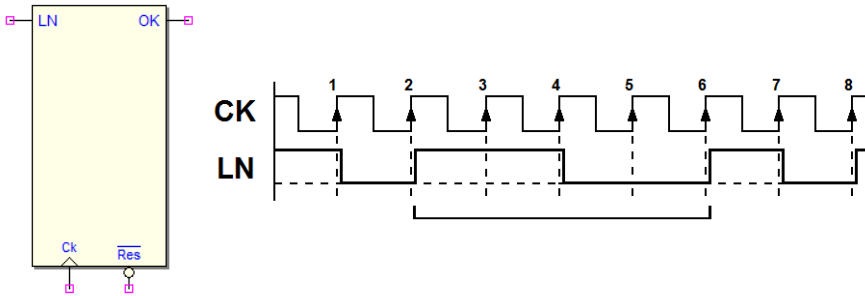
Successivamente, nel terzo ciclo di clock della sequenza (tra i fronti 3 e 4), l'uscita PWM deve essere generata solo se $P1$ è a 1, per cui proseguiamo il disegno come visibile in figura, al centro:



Grazie a considerazioni analoghe, completiamo il diagramma (a destra in figura) testando solo $P0$ nel quarto ciclo della sequenza (tra i fronti 4 e 5), poiché proveniamo da un percorso in cui siamo sicuri che anche $P1$ è a 1. Inseriamo infine l'ultimo stato (e), corrispondente all'ultimo ciclo della sequenza (tra i fronti 5 e 6), e abbiamo terminato.

7.3.3 Riconoscitore di sequenza

Una rete sincrona ha un ingresso LN e un'uscita OK . L'ingresso LN legge una sequenza continua di zeri e uni, come visibile nella figura:



L'uscita OK è attivata per la durata di *un ciclo* di clock ogni volta che è riconosciuta la sequenza '1100' (evidenziata in figura).

Soluzione

Definiamo uno stato di attesa (a) nel quale si attende di leggere un 1 sull'ingresso LN (vedi figura a lato). Ci spostiamo nello stato (b) non appena è stato letto $LN = 1$.

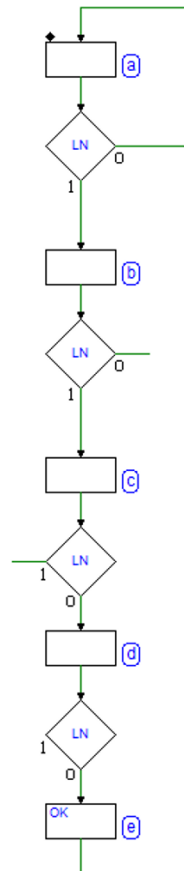
Dato che il valore dell'ingresso LN può cambiare ad ogni ciclo di clock, anche nello stato (b), e nei successivi, dovremo leggerne il valore.

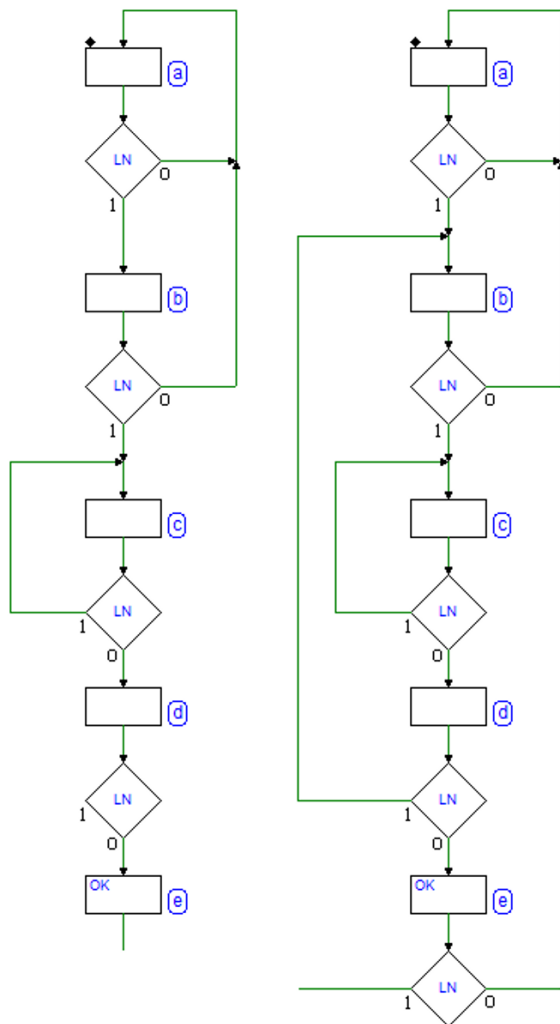
Proseguiamo il disegno del diagramma ASM considerando, per adesso, soltanto la sequenza attesa "1100", tralasciando momentaneamente gli altri percorsi.

Si arriva quindi nello stato (e) dopo avere letto la sequenza dei quattro valori 1, 1, 0 e 0, a distanza di un ciclo di clock uno dall'altro (dato che la MSF è sincrona). Se facciamo riferimento, a titolo di esempio, al diagramma temporale mostrato prima, i quattro valori sono stati letti, nell'ordine, sui fronti 3, 4, 5 e 6.

Nello stato (e) viene generata l'uscita 'OK', come da specifiche, per segnalare la rivelazione della sequenza cercata.

Completiamo ora, uno dopo l'altro, i percorsi lasciati in sospeso.





Consideriamo la figura qui accanto, a sinistra. Se nello stato (b) viene letto uno 0, dopo avere letto un primo 1 nello stato (a), dobbiamo ritornare nello stato (a), perché stiamo ricercando due 1 consecutivi.

Li avremo invece trovati se, nello stato (b), verrà letto un 1: ci spostiamo quindi nello stato (c). In questo stato, o leggiamo uno 0, che corrisponde al successivo bit della sequenza cercata, o leggiamo un 1.

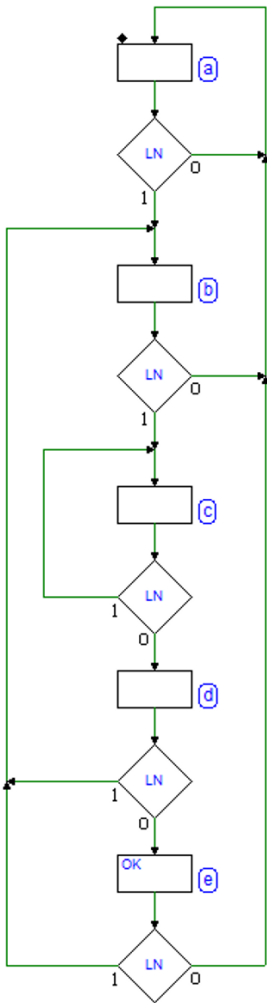
Leggere un 1, nello stato (c), non implica che la sequenza non corrisponda a quella ricercata: infatti le specifiche non richiedono che prima della sequenza ci sia necessariamente uno 0.

LN potrebbe rimanere a 1 anche per lungo tempo. Quello che conta è che dopo i due ultimi 1, vengano letti 0 e poi 1: rimaniamo quindi nello stato (c) in attesa dello 0.

Riassumendo, ci spostiamo nello stato (d) se abbiamo individuato lo 0 successivo ai due 1. Dobbiamo decidere cosa fare se, nello stato (d), non sarà letto l'atteso secondo 0.

Come si vede nella parte destra della figura, a proposito dell'eventuale 1 letto nello stato (d), l'unica ipotesi che possiamo fare è che, dato che segue uno 0, si tratti del primo 1 della sequenza da ricercare, per cui ci spostiamo nello stato (b), dove si giunge anche dallo stato (a) per lo stesso motivo.

Infine, dobbiamo valutare come uscire dallo stato (e), in cui la MSF si trova dopo avere individuato la sequenza cercata.



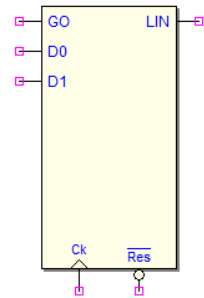
Completiamo quindi il diagramma (vedi figura a sinistra). Ipotizziamo di avere riconosciuto la sequenza, e di trovarci nello stato (e).

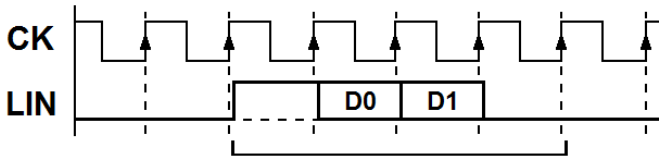
Se troviamo ancora uno 0, torniamo nello stato iniziale (a), ad attendere nuovamente una nuova sequenza; ma, nel caso venga letto un 1, questa potrebbe essere ricominciata proprio adesso, per cui si passerà nello stato (b), come abbiamo fatto nello stato (d).

7.3.4 Trasmettitore Seriale Sincrono (2 bit)

Si progetti un trasmettitore di sequenza seriale sincrona a 2 bit (in termini di MSF). Il dispositivo legge tre ingressi sincroni *GO*, *D0* e *D1*, e genera la sequenza sull'uscita *LIN*.

La trasmissione dei due bit di dato *D0* e *D1* è avviata da una transizione basso-alto dell'ingresso di comando *GO*. La figura seguente descrive il formato della sequenza trasmessa.

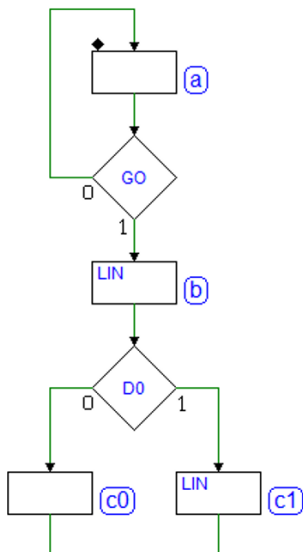
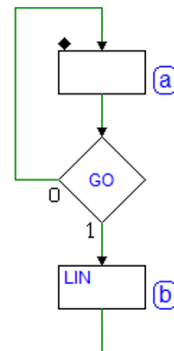




Il tempo di bit è pari al ciclo di clock *CK*; la sequenza inizia con un *bit di start* a 1, prosegue con i *due bit di dato* *D0* e *D1*, nell'ordine, ed infine termina con un *bit di stop* a 0:

Soluzione

Partendo dall'ipotesi che la linea *GO* sia a 0 inizialmente, attendiamo in (a) che *GO* vada ad 1. Non appena questo avviene, si passa nello stato (b), nel quale attiviamo l'uscita *LIN* (cioè iniziamo la trasmissione generando il *bit di start*), per la durata di un ciclo di clock:



Nel successivo ciclo di clock dobbiamo trasmettere il valore di *D0*, per cui leggiamo questo ingresso e decidiamo se andare nello stato (c0) oppure nello stato (c1).

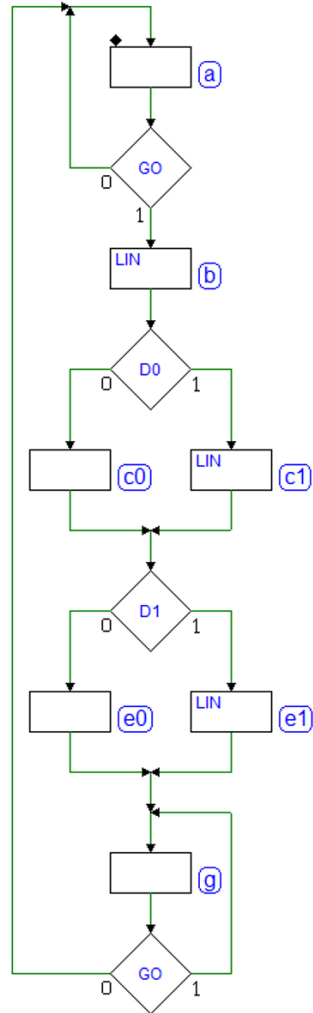
Nel primo l'uscita *LIN* non è attivata, nel secondo sì. Si noti che i due stati (c0) e (c1) rappresentano un'alternativa mutuamente esclusiva nel percorso degli stati, ma che dal punto di vista del tempo corrispondono al medesimo ciclo di clock, successivo a quello in cui abbiamo generato il *bit di start*.

Con lo stesso criterio (vedi figura a lato), nel ciclo di clock successivo trasmettiamo il valore di $D1$, imponendo $LIN = 0$ oppure $LIN = 1$, rispettivamente negli stati (e0) o (e1).

La sequenza seriale si deve concludere con un *bit di stop* a 0, per cui passiamo nello stato (g) dove l'uscita non è attivata.

Infine, dobbiamo verificare che l'ingresso GO sia tornato a 0, prima di tornare nello stato (a). Se non facessimo questa verifica, se GO fosse ancora alto, tornando nello stato (a) faremmo ripartire subito la trasmissione di una nuova sequenza (mentre questo deve avvenire solo sulla transizione basso-alto di GO).

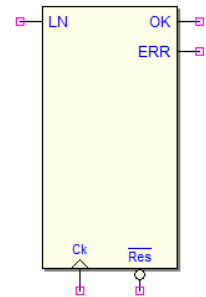
Invece di inserire un altro stato dove eseguire questo controllo, sfruttiamo direttamente lo stato (g) già definito, dato che non contiene uscite, trasformandolo in stato di attesa.

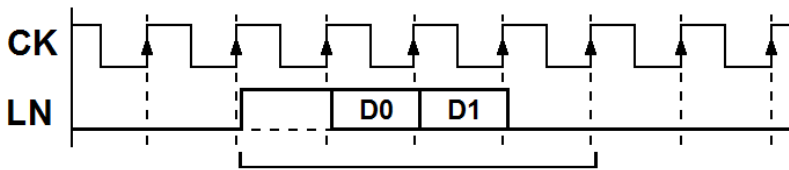


7.3.5 Ricevitore di comando in formato seriale sincrono

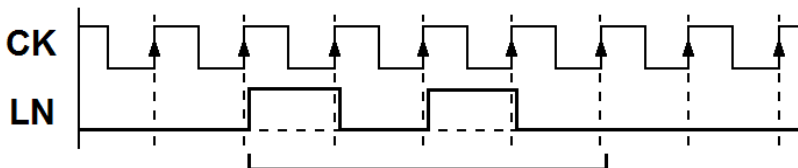
Si vuole progettare un ricevitore di comando in formato seriale sincrono, a 2 bit (come Macchina a Stati Finiti). Il dispositivo riceve le sequenze seriali sull'ingresso LN , e genera le uscite OK ed ERR .

Il tempo di bit è pari al ciclo di clock CK ; le sequenze iniziano tutte con un *bit di start* a 1, proseguono con i due bit di dato $D0$ e $D1$, ed infine terminano tutte con un *bit di stop* a 0. Nella figura seguente osserviamo il formato delle sequenze.





Tra le quattro combinazioni possibili, il nostro ricevitore deve riconoscere quella con $D0 = 0$ e $D1 = 1$, visibile nella figura seguente:



Al termine della ricezione di una sequenza, il ricevitore effettua un controllo della *correttezza del bit di stop*, per individuare la presenza di eventuali errori. Questo controllo consiste nel verificare che nel tempo di bit corrispondente al *bit di stop* il segnale seriale LN sia a 0. Questa verifica è effettuata in ogni caso, anche per le combinazioni *non di nostro interesse*.

Se il *bit di stop* è corretto (cioè è uguale a 0), e se la sequenza ricevuta corrisponde a quella cercata, il ricevitore attiva l'uscita *OK* per la durata di un ciclo di *CK*, segnalando quindi di avere *ricevuto il comando atteso*. Se la sequenza non è quella voluta, il ricevitore non attiva nessuna delle uscite.

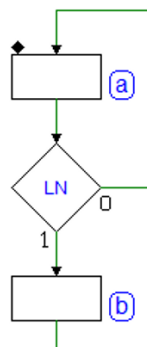
Se il *bit di stop* non è corretto, il ricevitore attiva l'uscita *ERR* (sempre per la durata di un *CK*). In ogni caso, dopo avere ricevuto una qualunque sequenza, il ricevitore ritorna sempre ad attendere la prossima.

Soluzione

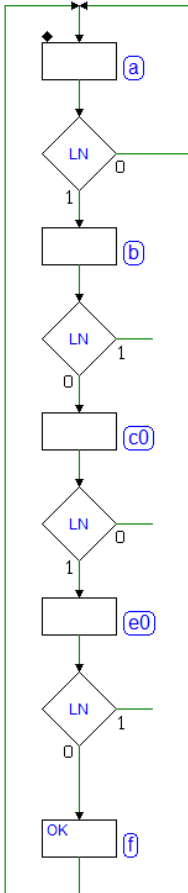
Iniziamo il disegno del diagramma ASM impostando un ciclo di attesa sullo stato (a), in cui aspettiamo l'arrivo di un 1 (il *bit di start*) sull'ingresso LN (vedi figura qui a destra).

Una volta individuato il *bit di start*, gli altri bit saranno letti di seguito, ad ogni ciclo di clock.

Ipotizzando di ricevere la sequenza attesa, senza errori, proseguiamo il disegno degli stati senza completarlo, per adesso, nelle altre diramazioni.



Si noti (figura sotto a sinistra) che gli stati (b), (c0) ed (e0) sono stati inseriti per testare LN ad ogni ciclo di clock, in corrispondenza dei bit ricevuti, uno dopo l'altro.



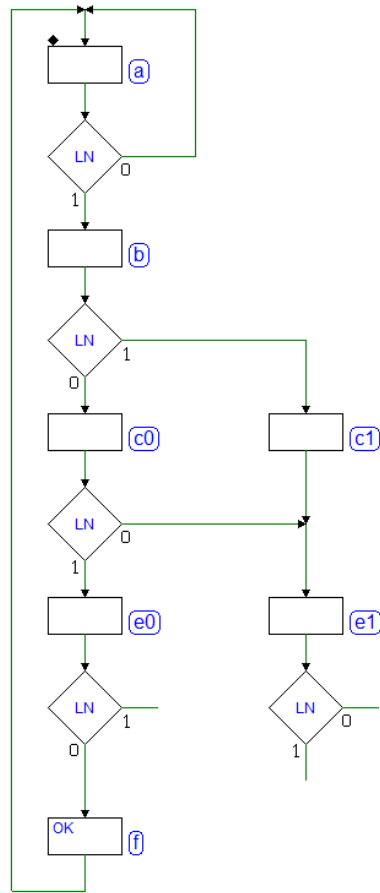
Si arriva in (f), quindi, se abbiamo letto $D0 = 0$, $D1 = 1$ e il *bit di stop* correttamente a 0.

L'attivazione di OK in (f) conclude la ricezione della sequenza attesa, e si ritorna in (a) ad attendere la prossima.

Completiamo adesso gli altri percorsi lasciati in sospeso (vedi figura a destra).

Se in (b) viene letto 1, stiamo ricevendo una sequenza che non è quella attesa.

Dobbiamo tuttavia completare la sua ricezione, perché dobbiamo verificare, al tempo giusto, che il *bit di stop* sia corretto.

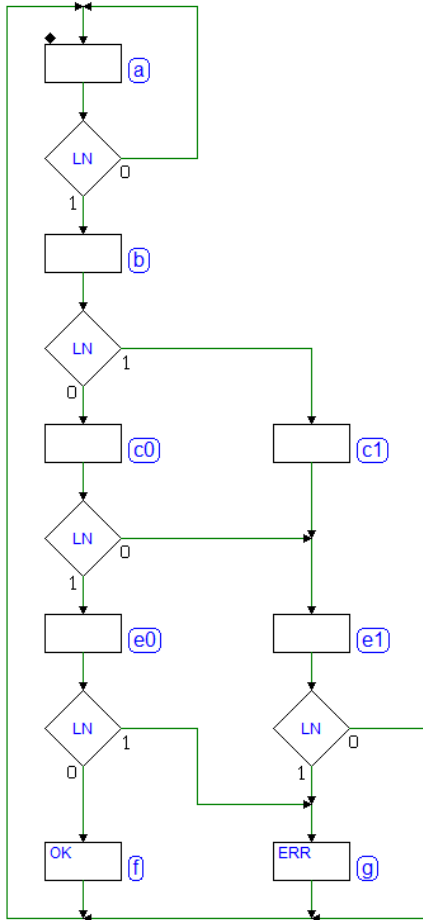


Sono stati quindi introdotti gli stati (c1) ed (e1): per permettere che il test sia effettuato allo stesso tempo della sequenza attesa.

Si noti che lungo questo percorso non viene controllato il valore del secondo bit, perché ormai sappiamo che questa sequenza *non* è quella attesa, e *non attiveremo* l'uscita OK .

Torniamo a considerare lo stato (b). Se il primo bit viene letto a 0 (quello atteso), si prosegue nello stato (c0) per controllare il secondo. Se questo è uguale a 0, la sequenza non è quella attesa e passiamo nello stato (e1), perché *non attiveremo* l'uscita OK .

Completiamo infine il diagramma (vedi figura seguente), considerando la valutazione del *bit di stop* che avviene nello stato (e1).



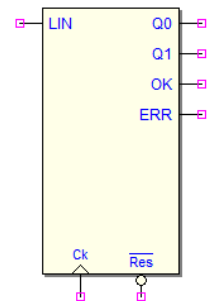
Se il bit è *corretto*, torniamo nello stato (a) ad attendere una nuova sequenza; ma se il bit *non è corretto*, dobbiamo attivare l'uscita *ERR*, per un ciclo di clock, nello stato (g), e successivamente tornare in (a).

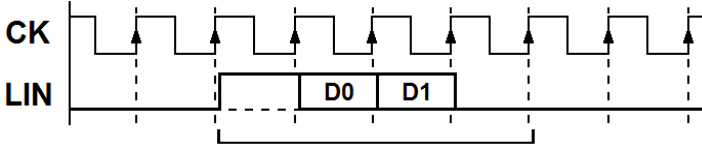
Infine, completiamo il diagramma utilizzando lo stato (g) anche per l'altro percorso, quando la sequenza ricevuta risulta uguale a quella attesa, ma con il *bit di stop* sbagliato, per cui attiviamo *ERR*, invece che *OK*.

7.3.6 Ricevitore Seriale Sincrono (2 bit)

Si progettano, in termini di MSF, un *ricevitore seriale sincrono*, a 2 bit. Il dispositivo riceve le sequenze seriali sulla linea *LIN*, e genera le uscite *Q0*, *Q1*, *OK* ed *ERR*.

Il tempo di bit è pari al ciclo di clock *CK*; le sequenze iniziano tutte con un *bit di start* a 1, proseguono con i due bit di dato *D0* e *D1*, ed infine terminano tutte con un *bit di stop* a 0. Nella figura seguente osserviamo il formato delle sequenze ricevute.



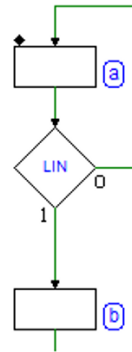


Si osservi che il segnale *LIN* è sincrono con il clock *CK*, con un ritardo di propagazione rispetto al fronte di salita (situazione tipica nel caso in cui il sistema che genera *LIN* abbia lo stesso clock del dispositivo che stiamo progettando). *LIN* è quindi campionato sul fronte di clock *successivo* a quello che l'ha generato.

Al termine della ricezione di una sequenza, il ricevitore controlla la *correttezza del bit di stop*, per individuare la presenza di eventuali errori. Se il *bit di stop* è correttamente a zero, il ricevitore attiva l'uscita *OK*, segnalando di avere ricavato dalla sequenza i due bit di dato *D0* e *D1*, e di averli reso disponibili, rispettivamente, sulle uscite *Q0* e *Q1*. Le uscite *Q0*, *Q1* e *OK* sono mantenute attive fino alla ricezione di una nuova sequenza. Se il *bit di stop* non è corretto, il ricevitore attiva solo l'uscita *ERR*, per la durata di un ciclo di *CK*, e quindi attende una nuova sequenza.

Soluzione

Iniziamo il disegno del diagramma ASM impostando un ciclo di attesa sullo stato (a), in cui aspettiamo l'arrivo di un 1 (il *bit di start*) sull'ingresso *LIN* (figura a destra). Una volta individuato il *bit di start*, gli altri bit saranno letti di seguito, uno per ogni ciclo di clock.

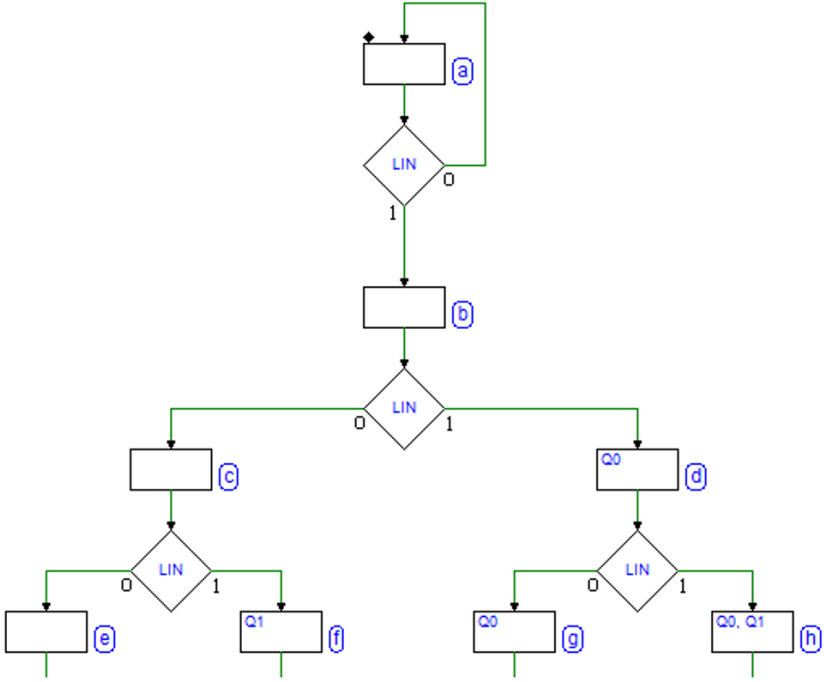


Consideriamo quindi lo stato (b): durante la permanenza in questo stato, il valore sull'ingresso *LIN* corrisponde al bit *D0*. A seconda del valore letto, dobbiamo percorrere strade diverse, come mostrato nella prima figura della pagina successiva: andiamo nello stato (c) se $D0 = 0$, altrimenti nello stato (d) se $D0 = 1$.

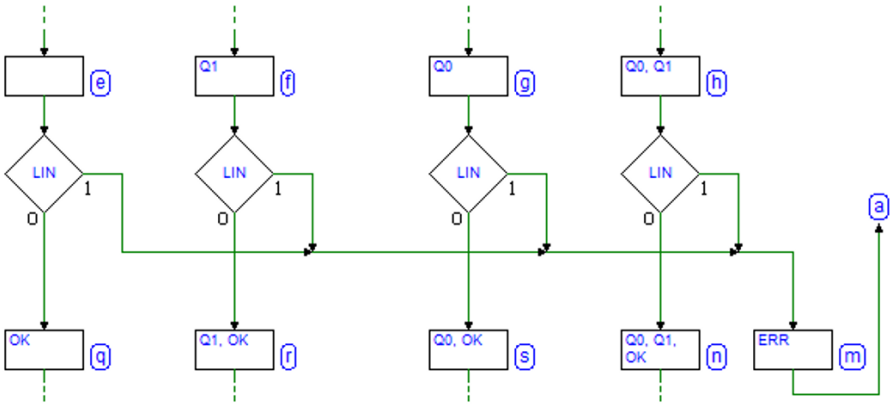
Ripetiamo lo stesso ragionamento per *D1*, il secondo bit di dato della sequenza, separando ulteriormente i percorsi, in base al valore ricevuto. A valle della ricezione dei bit *D0* e *D1*, la MSF si troverà in uno degli stati (e), (f), (g) oppure (h), corrispondenti alle quattro possibili combinazioni di valori.

Separare i *percorsi* è necessario perché in questo modo la MSF è in grado di *ricordare*, grazie allo stato in cui si trova, la storia precedente degli ingressi (nel nostro caso ci interessano *D0* e *D1*, letti nella sequenza).

L'attivazione delle uscite *Q0* e *Q1* lungo i quattro percorsi visti non sarebbe necessaria: però ci è utile per avere un riscontro *esterno* del valore ricevuto, utile a scopo di verifica.



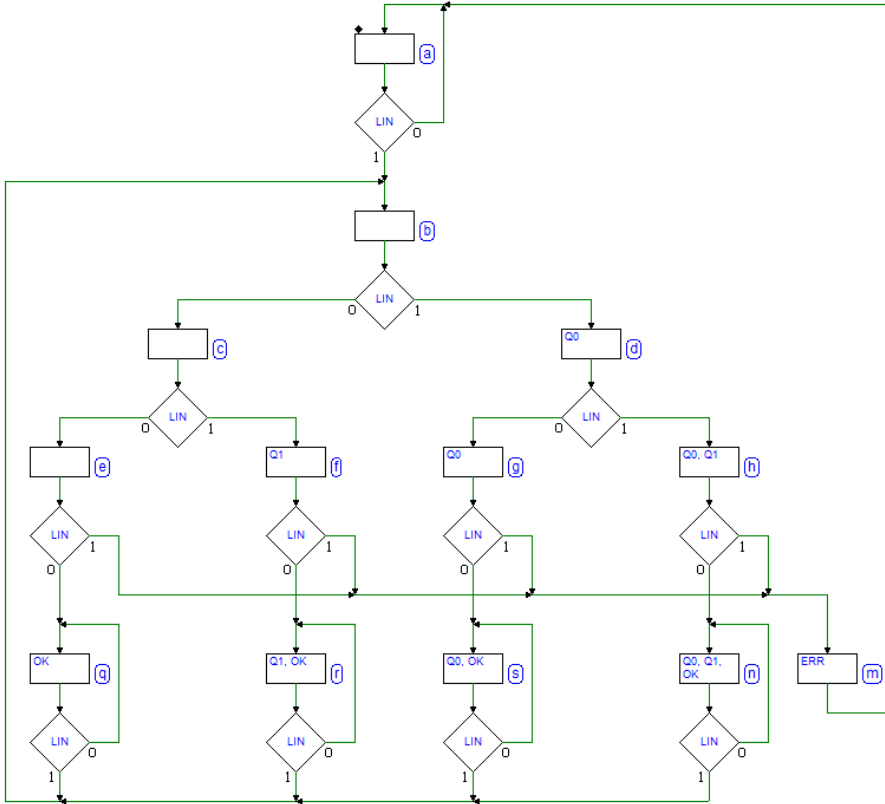
Dopo i bit di dato, dobbiamo testare il *bit di stop*, presente sulla linea *LIN* proprio quando la MSF si trova in uno degli stati (e), (f), (g) o (h):



Quindi, come si vede nella figura qui sopra, in ognuno di questi stati controlliamo *LIN* e, nel caso il *bit di stop* non sia corretto, riuniamo tutti i percorsi (non ci interessa più ricordare i valori del bit di dato) e generiamo l'uscita *ERR* nello stato (m), per un ciclo di clock, per poi tornare in (a).

Invece, se il *bit di stop* è corretto, per ognuno dei quattro percorsi, andremo in un corrispondente stato (q), (r), (s) oppure (n), in cui attiveremo le uscite *Q0* e *Q1* a seconda del dato ricevuto e l'uscita *OK*.

Completiamo il diagramma, come visibile nella figura qui sotto, tenendo conto della specifica che richiede che i valori sulle uscite $Q0$, $Q1$ e OK debbano essere mantenuti fino all'arrivo della successiva sequenza: per ottenere questo, ognuno degli stati (q), (r), (s) e (n) viene chiuso in ciclo su se stesso. Dal ciclo si esce solo se dalla linea LIN arriva un nuovo *bit di start*.

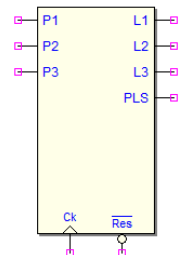


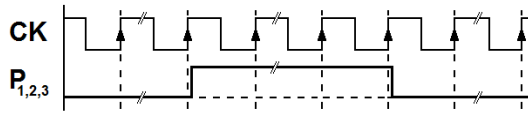
Ognuno degli stati (q), (r), (s) o (n) si sostituisce, quindi, alla funzione svolta da (a), per quanto riguarda l'attesa della prossima sequenza. Si noti che, come avviene per lo stato (a), per ognuno di essi, lo stato successivo sarà (b).

7.3.7 Gestione di pulsanti

Si progetti una rete sequenziale sincrona (come MSF) che gestisca lo stato dei pulsanti $P1$, $P2$ e $P3$ e generi le uscite $L1$, $L2$ e $L3$ (da collegare ad altrettanti LED) e una uscita PLS (*pulse*).

La figura successiva descrive l'andamento tipico del segnale agli ingressi $P1$, $P2$ o $P3$: inizialmente un pulsante è *a riposo*, poi viene *premuto* e, infine, *rilasciato*.

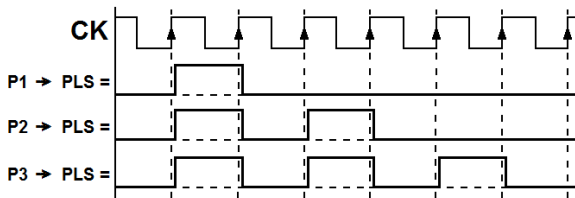




Si faccia l'ipotesi che nei pulsanti sia già inclusa la necessaria circuiteria per risolvere il problema dei *rimbalzi elettro meccanici*, affinché il segnale da loro generato possa essere considerato esente da rimbalzi e sincrono rispetto al clock.

I pulsanti devono essere gestiti tenendo conto della *priorità*: P_3 , P_2 ed infine P_1 . Questo significa che, se viene premuto P_3 , lo stato degli altri pulsanti viene ignorato; se non è premuto P_3 , ma è premuto P_2 , lo stato di P_1 è ignorato; P_1 è preso in considerazione solo se nessuno degli altri pulsanti è premuto.

Quando un pulsante P_i è premuto, la MSF attiva la corrispondente uscita L_i per tutto il tempo in cui il pulsante rimane premuto. Al *rilascio* del pulsante, il LED viene spento, ed è attivata l'uscita PLS , che genererà 1, 2 o 3 impulsi (a seconda del pulsante premuto), come mostrato nella figura seguente:

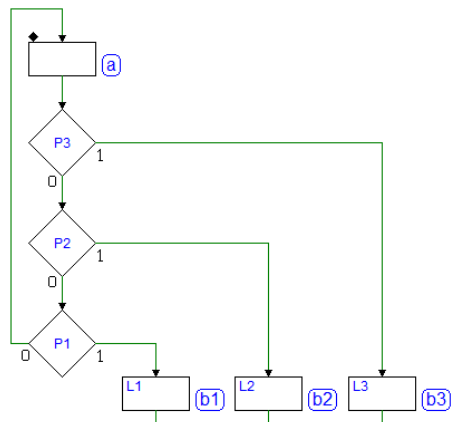


Soluzione

Come si osserva nella figura a lato, impostiamo un ciclo di attesa sullo stato (a), ove rimaniamo se nessun pulsante è premuto.

Aggiungiamo poi al diagramma gli stati (b1), (b2) e (b3), dove viene attivata l'uscita corrispondente al pulsante premuto (L_1 , L_2 o L_3).

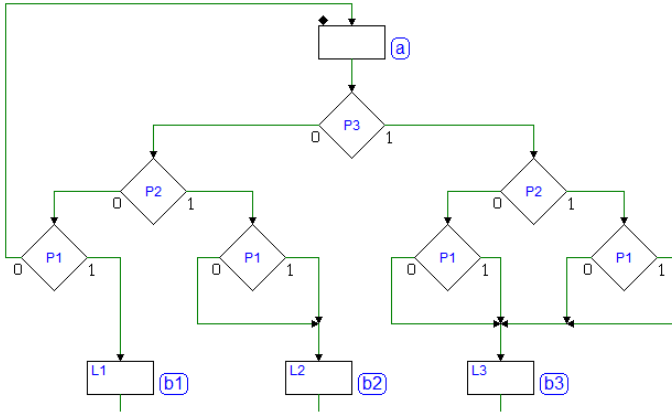
Dal punto di vista dell'algoritmo, ci spostiamo in questi stati per "ricordare" che un determinato pulsante (P_1 , P_2 o P_3) è stato premuto.



Si noti la logica con cui sono testate le linee provenienti dai pulsanti. Rispetchia le specifiche circa la priorità: se P_3 è premuto, non dobbiamo controllare gli altri pulsanti; se P_3 non è premuto, e lo è P_2 , non controlliamo P_1 .

Si faccia attenzione che la figura potrebbe trarre in inganno, in quanto potrebbe sembrare che il test dei pulsanti sia eseguito “uno dopo l’altro”, cioè in ordine di tempo, ma *non è così*. La figura descrive invece una *logica combinatoria*, dove gli ingressi sono *valutati congiuntamente e allo stesso tempo*, ossia durante lo stato (a): P_2 e P_1 sono *indifferenti* se P_3 è a 1; P_1 è *indifferente* se $P_3 = 0$ e $P_2 = 1$.

La descrizione della figura precedente è equivalente alla seguente, meno sintetica, dove si considerano tutte le possibili combinazioni di P_1 , P_2 e P_3 :

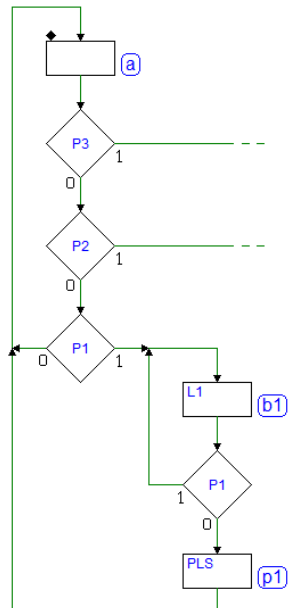


Alcuni degli otto possibili percorsi sono stati riuniti insieme, seguendo il criterio della priorità prima descritta. Eliminando i blocchi decisionali inutili, e cioè che confluiscono nello stesso percorso, si ottiene la figura di prima.

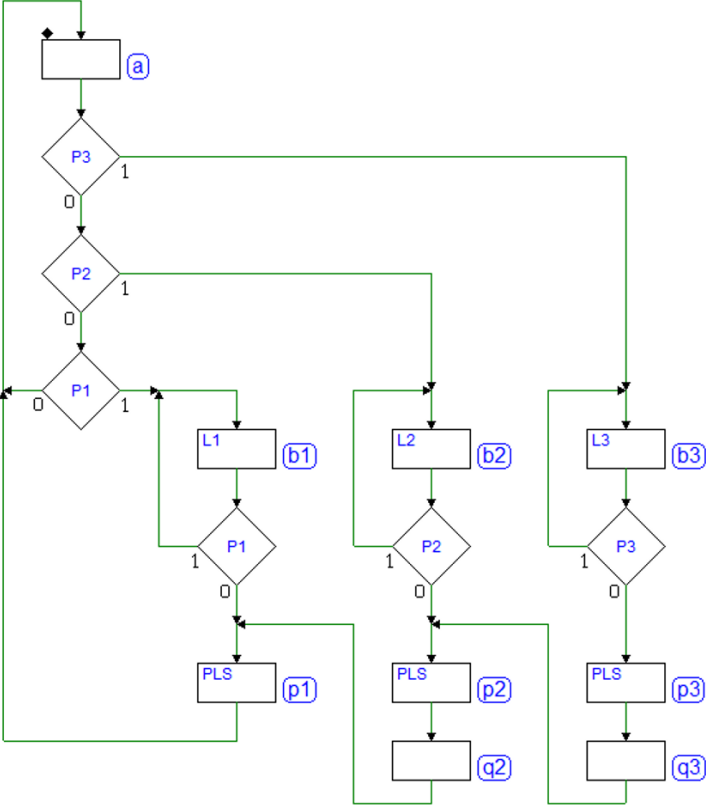
Torniamo ora a ragionare sulla funzionalità degli stati (b1), (b2) e (b3): le specifiche richiedono che l’uscita L_i corrispondente al pulsante P_i sia *mantenuta attiva* per tutto il tempo in cui è *premuto*, e che debba essere poi attivata l’uscita PLS al suo *rilascio*.

Questo significa che (b1), (b2) e (b3) devono essere impostati come *stati di attesa* (che il relativo pulsante sia rilasciato), come mostrato nella figura accanto, dove per adesso è stata impostata la logica del solo pulsante P_1 .

Impostiamo la generazione degli impulsi sull’uscita PLS . Il caso più semplice è quello del rilascio del pulsante P_1 : come si vede, attiviamo PLS per un ciclo di clock, nello stato (p1), per poi ritornare nuovamente in attesa della pressione di un pulsante, in (a).



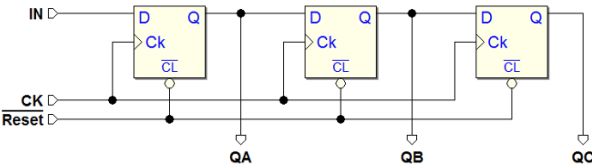
La figura seguente mostra il diagramma ASM completato. La generazione di *due impulsi* su *PLS*, a seguito del rilascio di *P2*, è ottenuta con una sequenza di *tre stati*: (p2), in cui attiviamo l'uscita *PLS*; (q2), in cui la disattiviamo; infine riutilizziamo (p1), lo stato già introdotto per il pulsante *P1*.



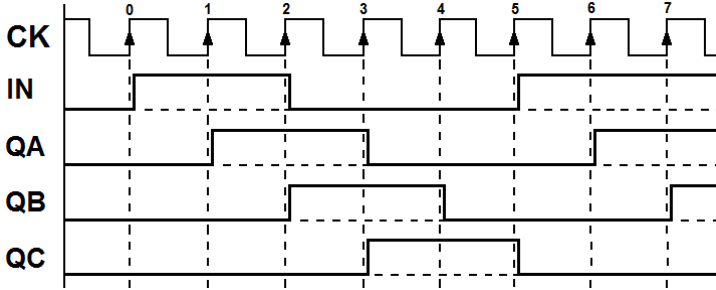
Nella figura è stato completato anche il percorso relativo a *P3*, utilizzando lo stesso criterio di riutilizzo di stati già presenti.

7.3.8 Registro a Scorrimento (3 bit)

Si vuole descrivere, in termini di MSF, il funzionamento di un *registro a scorrimento* (a 3 bit). Riceve un ingresso *IN* e genera le uscite, nell'ordine di scorrimento *QA*, *QB* e *QC*:



La figura seguente mostra un esempio di funzionamento del registro, per una particolare sequenza applicata all'ingresso *IN*:



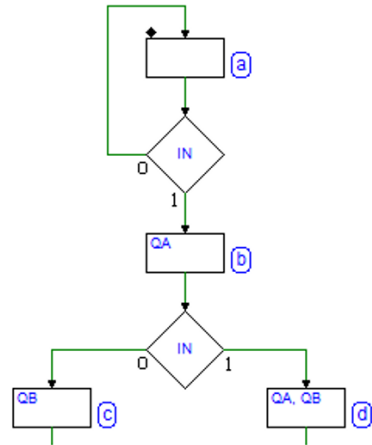
Per sua caratteristica, il diagramma temporale non si presta per mostrare tutti i casi possibili, che dipendono dalla grande variabilità di sequenze che si possono presentare all'ingresso. Invece, il diagramma ASM, una volta completato, descriverà in modo completo il comportamento del registro.

Soluzione

Ipotizziamo che il registro sia inizialmente tutto azzerato, nello stato (a) (vedi figura qui accanto). Attendiamo che arrivi un 1 all'ingresso *IN*: se questo avviene, sul prossimo fronte attivo del clock *CK*, il registro ne carica il valore sull'uscita *QA*.

Formalmente descriveremo questa nuova situazione cambiando stato: la MSF si porta in (b), ove viene attivata l'uscita *QA*.

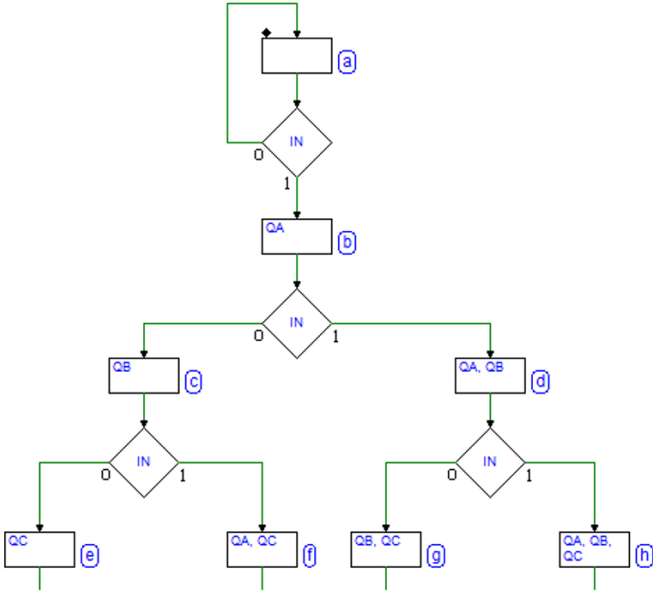
Si noti che l'ingresso dovrà sempre essere controllato, in qualunque stato, perché l'uscita *QA* dipenderà sempre dal valore di *IN*: valutiamo ora l'ingresso nello stato (b).



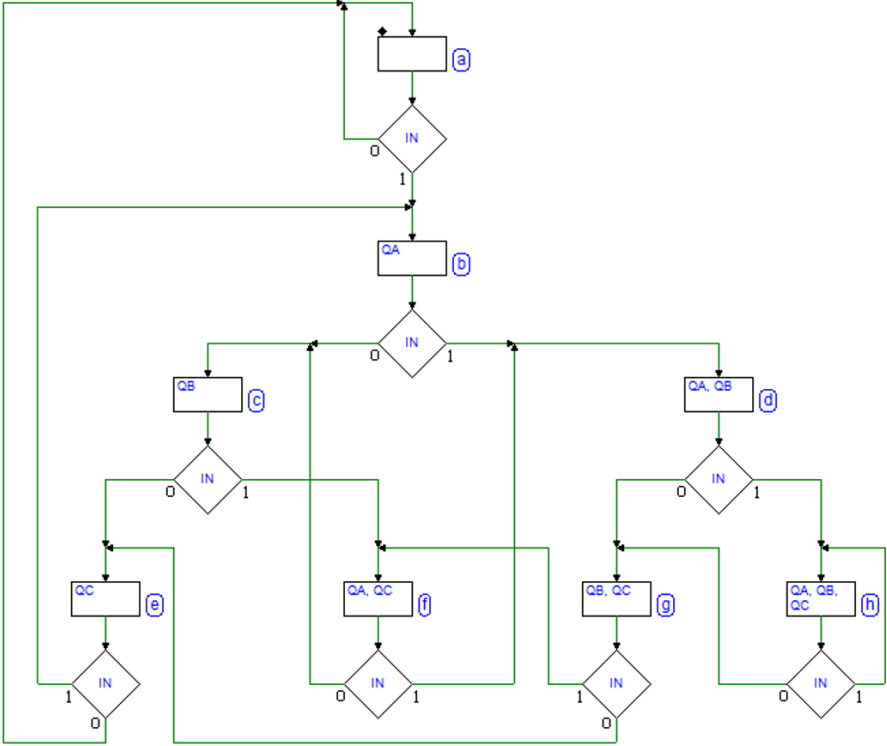
Sul prossimo fronte attivo del clock la MSF cambia stato: il valore memorizzato in *QA* si sposta in *QB*, mentre *QA* acquisisce il valore di *IN*. Gli stati (c) e (d) descrivono i due casi possibili, in dipendenza dal valore acquisito.

Ripetiamo lo stesso ragionamento per il prossimo fronte attivo del clock, quando il valore memorizzato in *QB* si sposta in *QC*, *QA* in *QB*, mentre *QA* ricopia *IN*, ottenendo il diagramma visibile nella pagina seguente (in alto).

Non è ancora finito, ma include tutti gli otto stati possibili (le combinazioni di valori delle uscite *QA*, *QB* e *QC*). Quindi, sicuri di non dovere introdurre altri stati, dovremo valutare in quali stati la MSF si sposterà, in dipendenza dal valore di *IN*, a partire dagli stati (e), (f), (g) e (h).



Completiamo il diagramma, quindi, ottenendo quello finale, visibile qui sotto:



Qui alcune considerazioni che hanno permesso di completare il diagramma, a partire dagli stati (e), (f), (g) e (h).

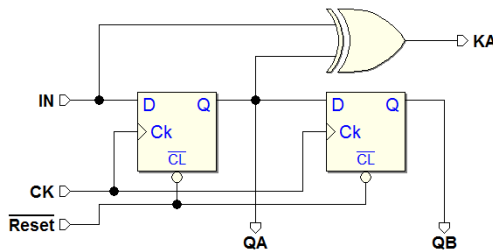
Se la MSF si trova nello stato (e), dopo lo scorrimento sul prossimo fronte attivo del clock, QC e QB saranno a 0 quale che sia il valore di IN , poiché caricheranno rispettivamente i contenuti di QB e QA , che in questo stato sono a 0. Gli stati già esistenti con QB e QA pari a 0 sono gli stati (a) e (b). Quindi, se IN è a 0, andremo in (a), altrimenti se è a 1, andremo in (b).

Analogo ragionamento per quanto riguarda lo stato (f), il cui stato successivo non potrà essere che (c), se IN è a 0, oppure lo stato (d), se IN è a 1.

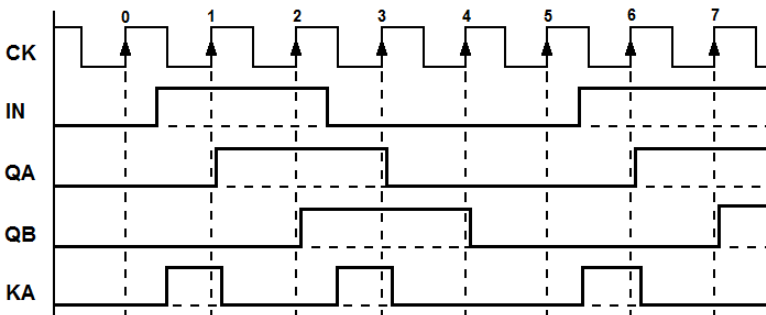
Seguendo ancora lo stesso criterio per gli stati (g) e (h), si completa il diagramma richiudendo tutti i percorsi logici rimanenti.

7.3.9 Rete sequenziale con uscita condizionata

Si vuole descrivere, come MSF, il funzionamento di una rete sequenziale, basata su di un *registro a scorrimento da 2 bit*, con l'aggiunta di una uscita che è funzione non solo dello stato, ma anche dell'ingresso (modello di *Mealy*):



La rete legge IN e genera le uscite QA e QB . Una porta EXOR genera l'uscita KA , funzione dell'ingresso IN e dell'uscita QA . La figura qui sotto seguente ne mostra il funzionamento, per una particolare sequenza di ingresso:



Come si vede, KA è generata quando IN e QA sono diversi. Il diagramma ASM descriverà la funzionalità della rete, inclusa l'uscita *condizionata* KA .

Note sulle uscite condizionate

Anche se ipotizziamo che il segnale in arrivo sull'ingresso IN sia generato da un altro sistema *sincrono* con lo stesso clock della MSF, alcuni fattori in gioco (la lunghezza dei collegamenti, tecnologia eventualmente diversa, o altro) potrebbero ritardare l'arrivo del segnale che, pur *rimanendo sincrónico*, risulterà traslato di una certa quantità nel tempo (se si osserva la sequenza temporale proposta nella figura precedente, IN è stato disegnato un poco più a destra degli altri segnali).

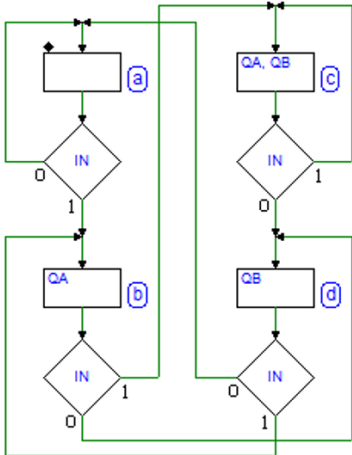
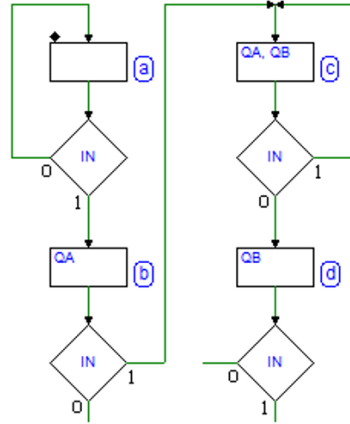
La conseguenza di questo è che la durata degli impulsi sull'uscita KA , essendo questa una funzione combinatoria di IN , potrà risultare significativamente minore rispetto alla durata di un ciclo di clock (come esemplificato sempre nella stessa figura). In molti progetti, tuttavia, questo particolare potrebbe essere irrilevante, nell'ipotesi che sia sufficiente che KA sia correttamente leggibile sul fronte attivo del clock.

Soluzione

Inizialmente, per semplificare, disegniamo il diagramma ASM ignorando l'uscita KA .

Nello stato (a), QA e QB sono a 0. All'arrivo di un 1 su IN , l'uscita QA passerà a 1 (sul prossimo fronte attivo del clock CK); ci saremo spostati nello stato (b), dove l'uscita QA è attiva (vedi figura a lato).

Supponendo di ricevere all'ingresso IN alcuni 1 di seguito, proseguiamo nel completare il diagramma: dallo stato (b) ci spostiamo in (c), dove sono attive tutte e due le uscite QA e QB . In (c) rimaniamo se su IN continua a presentarsi un 1.



Se $IN = 0$, invece, ci spostiamo in (d): il precedente QA si sposta in QB , mentre QA acquisisce lo 0.

Completiamo ora i percorsi mancanti, come nella figura a sinistra. Se siamo in (b), nel caso in cui IN valga 0, QA si sposta in QB , e in QA entra uno 0, per cui andremo in (d).

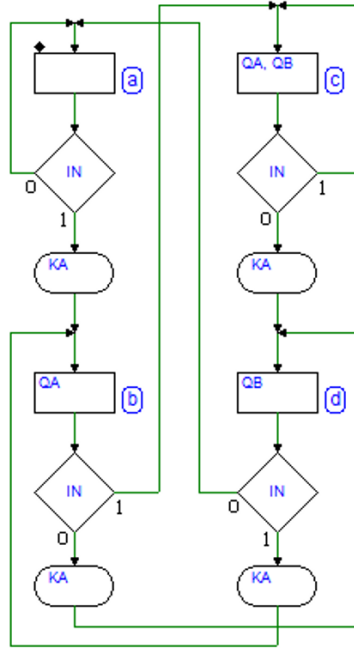
Nello stato (d) è attiva solo QB ; qualunque sarà il prossimo stato, essendo $QA = 0$, QB si azzererà. Quindi, in dipendenza dal valore dell'ingresso, ci sposteremo in (a) oppure in (b).

Aggiungiamo la descrizione dell'uscita condizionata. Dallo schema della rete, si vede che KA si attiva se QA e IN sono *diversi*.

Consideriamo lo stato (a), dove $QA = 0$: per quanto osservato ora, KA si attiverà *durante* la permanenza in (a), se $IN = 1$; aggiungiamo quindi un blocco *uscita condizionata* lungo il percorso logico per $IN = 1$ (vedi figura qui accanto).

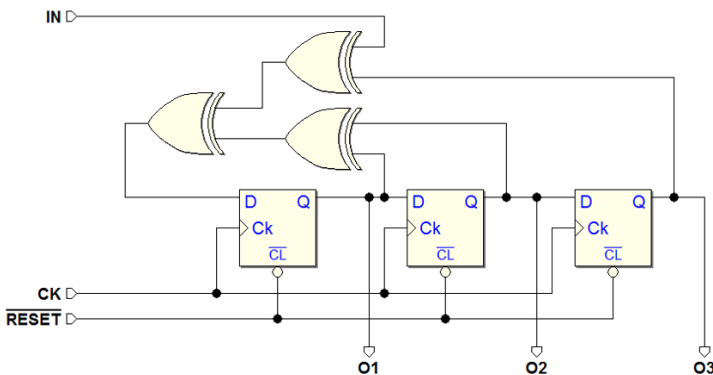
Anche nello stato (d) abbiamo $QA = 0$: l'*uscita condizionata* andrà aggiunta con lo stesso criterio (per $IN = 1$).

Completiamo infine il diagramma, in relazione agli stati (b) e (c), dove $QA = 1$. L'uscita KA si attiverà *durante* la permanenza in questi stati, se $IN = 0$: il blocco sarà posto lungo i percorsi per $IN = 0$.



7.3.10 Registro a scorrimento con albero di EXOR

Si vuole descrivere, come MSF, il funzionamento di una rete sequenziale, composta da un *registro a scorrimento* a 3 bit e da un albero di EXOR *in retroazione* che *valuta la parità* delle uscite $O1$, $O2$ e $O3$ dei flip-flop e la confronta con il valore dell'ingresso IN . Il risultato del confronto è reintrodotta nel registro a scorrimento:



Una utile osservazione, ai fini della descrizione della rete in termini di MSF, è che se il numero di uno presenti sulle uscite $O1$, $O2$ e $O3$ è pari, nel primo flip-flop sulla sinistra è inserito il valore dell'ingresso IN ; se il numero di uno è dispari, viene inserito il suo negato.

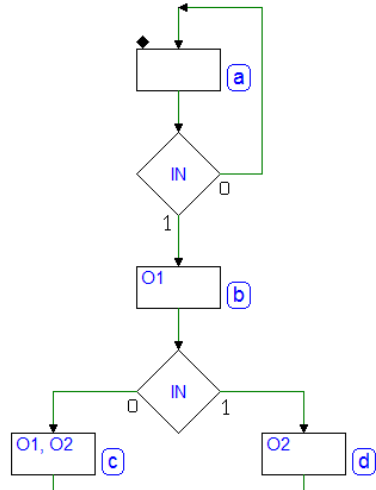
Soluzione

Nella figura seguente osserviamo i primi passi di costruzione del diagramma ASM. Nello stato (a) le uscite $O1$, $O2$ e $O3$ sono impostate a 0.

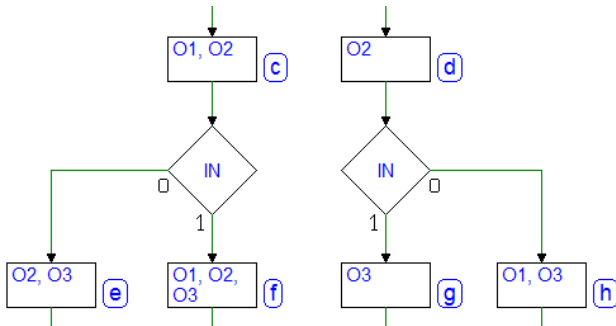
Quindi, nello stato (a) l'albero di EXOR non farà altro che ricopiare l'ingresso IN , per cui sul prossimo fronte attivo del clock, se $IN = 1$, si passerà nello stato (b), attivando l'uscita $O1$.

Nello stato (b) abbiamo un numero dispari di uscite a 1, per cui l'uscita $O1$ sul fronte attivo del clock caricherà il negato di IN , mentre $Q2$ ricopierà semplicemente il valore attuale di $Q1$.

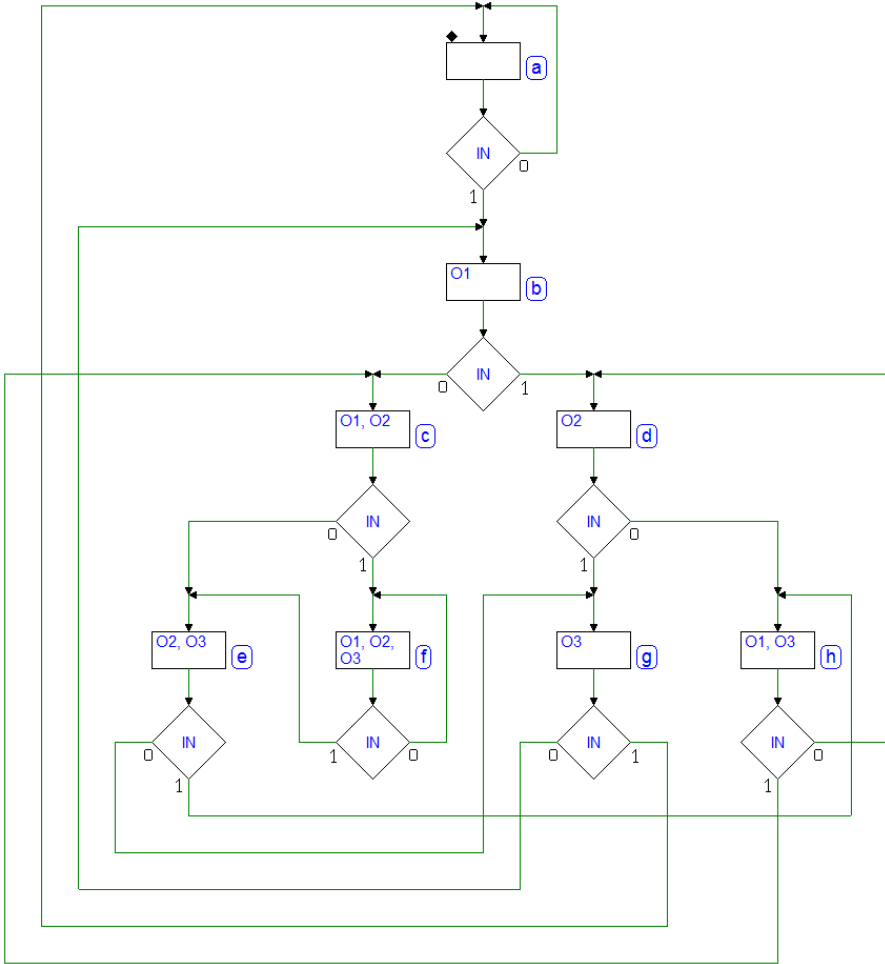
Introduciamo quindi due nuovi stati: (c) e (d). In entrambi $Q2$ è attiva, mentre $Q1$ sarà attiva solo in (c), quello in cui la MSF va, partendo da (b), se $IN = 0$.



Applicando lo stesso criterio agli stati (c) e (d), vedi figura seguente, introduciamo altri quattro stati: (e), (f), (g) e (h):



A questo punto il diagramma include otto stati, tutti quelli possibili in una rete con tre flip-flop. Nel completare il diagramma, dovremo stare attenti a non aggiungerne altri, ma a ricongiungerci con quelli già esistenti, in dipendenza dall'ingresso IN , a partire dagli ultimi stato introdotti (e), (f), (g) e (h). Il diagramma completo è visibile nella figura successiva.

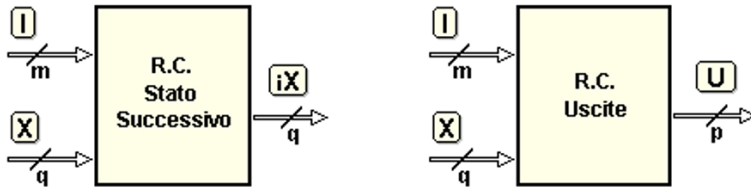


7.4 Sintesi della MSF sincrona

In questo paragrafo ci occuperemo del processo di *sintesi* della *MSF sincrona*, mediante cui si realizza una rete sequenziale sincrona a partire dalla corrispondente descrizione algoritmica. Un sistema di CAD normalmente ci consente di effettuare la sintesi in modo automatico. Tuttavia, è utile affrontare l'argomento della sintesi in modo sistematico, esaminandone le regole ed i concetti, come se dovessimo procedere in modo manuale.

Riferendoci al modello di Mealy, il problema della sintesi si riduce alla definizione di due reti, la *Rete Combinatoria dello Stato Successivo* e la *Rete Combinatoria delle Uscite*.

Nella figura seguente si osserva che entrambe le reti ricevono in ingresso il vettore I (gli ingressi della rete) ed il vettore X (lo stato della rete). La prima produce il vettore iX (lo stato successivo *proposto* al registro di stato), la seconda il vettore U (le uscite della rete).

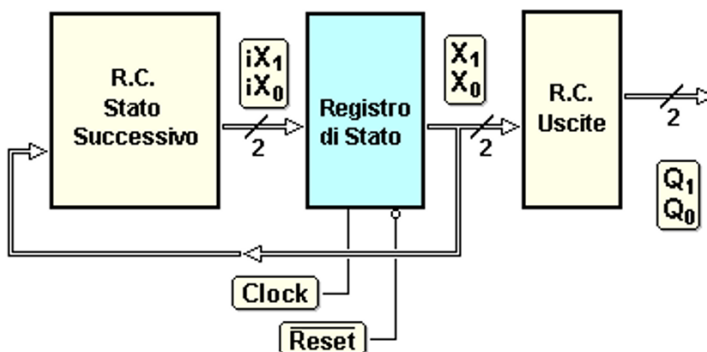


Gli ingressi I e le uscite U sono definiti dalle specifiche della rete da realizzare; il numero delle variabili che compongono i vettori iX e X viene a dipendere dal numero di stati previsti dall'algoritmo definito in sede di progetto (e dal tipo di flip-flop che si sceglie per il registro di stato). Nei paragrafi precedenti abbiamo semplicemente assegnato un nome ad ogni stato: per sintetizzare la rete è indispensabile associare ad ogni stato anche *un codice binario unico*, e questo processo è detto *assegnazione degli stati*.

7.4.1 Assegnazione degli stati

In una MSF sincrona ci sono varie tecniche per l'assegnazione dei codici agli stati: alcune tendono a ridurre la complessità della realizzazione circuitale, altre a migliorare le prestazioni in termini di velocità. Generalmente si ricerca un compromesso tra dimensioni del circuito e prestazioni. La prima tecnica che prendiamo in considerazione minimizza il numero delle variabili di stato: con q variabili si possono codificare fino a 2^q stati.

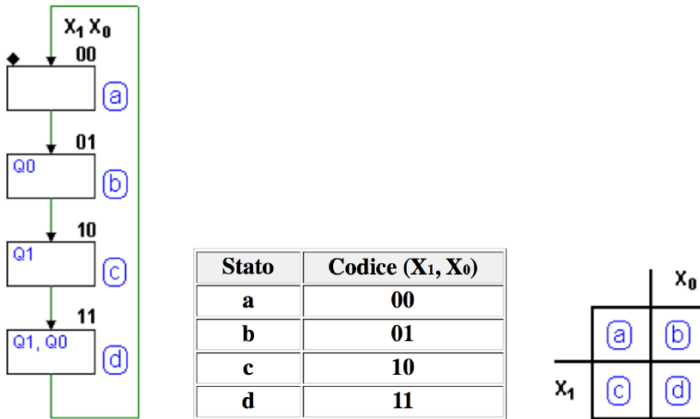
Il contatore a quattro stati esaminato in precedenza è una rete *sincrona senza ingressi* (escludendo il *Clock* e il *Reset*). Il minimo numero di variabili che permette di rappresentare *quattro stati* è due (X_1 e X_0). Riferendoci al modello di MSF sincrona di Moore, tale contatore può essere rappresentato dal modello ridotto proposto nella figura seguente:



Grazie alla struttura sincrona del sistema e all'assenza di ingressi, la scelta dei codici di stato può essere del tutto *arbitraria*. Con due variabili di stato, abbiamo quattro possibili combinazioni e quindi 24 possibili diverse assegnazioni dei codici ai quattro stati ($24 = 4!$, ossia il numero di possibili *permutazioni* di quattro combinazioni).

Nel nostro caso, potendo scegliere, associamo ad ogni stato il codice che coincide con la *combinazione binaria delle uscite che lo stesso stato genera*. In altre parole, è conveniente far corrispondere alle uscite $Q1$ e $Q0$ direttamente le variabili di stato $X1$ e $X0$, in modo che la rete combinatoria delle uscite degeneri in due semplici collegamenti ($Q1 = X1$ e $Q0 = X0$).

L'assegnazione degli stati può essere rappresentata in vari modi, tutti equivalenti. Negli esempi riportati nella figura seguente: a sinistra, l'assegnazione è riportata direttamente nel *diagramma ASM*, annotata sopra ad ogni blocco di stato; al centro, la stessa assegnazione è descritta da una *tabella*; a destra, si è utilizzata una *mappa*:



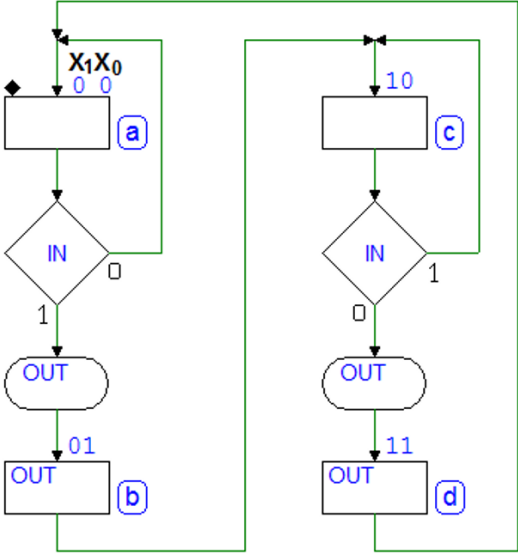
7.4.2 Descrizione della MSF mediante tabella degli stati

La *tabella degli stati* è una descrizione formalmente *equivalente* al diagramma ASM, e fornisce esattamente la stessa informazione: per ogni stato, in funzione degli eventuali ingressi, indica le uscite corrispondenti e lo stato successivo. Nel caso del contatore senza ingressi di prima, la tabella è la seguente:

Stato Attuale					Stato Successivo		
Stato		Uscite			Stato		
Nome	X1	X0	Q1	Q0	Nome	iX1	iX0
a	0	0	0	0	b	0	1
b	0	1	0	1	c	1	0
c	1	0	1	0	d	1	1
d	1	1	1	1	a	0	0

La tabella è divisa in due campi: quello a sinistra descrive lo *stato attuale* (nome e codice) e le relative *uscite*. Sulla destra, invece, per ogni possibile stato attuale, è indicato il nome e il codice dello *stato successivo*.

Un altro esempio di tabella degli stati si ricava da uno dei rivelatori di fronti descritti in precedenza, quello nel quale sono presenti quattro stati, un ingresso e un'uscita. Procediamo ad assegnare i codici degli stati direttamente sul diagramma, come si vede nella figura seguente. Se facciamo l'ipotesi che *l'ingresso sia sincrono*, anche in questo caso l'assegnazione può essere *arbitraria*.



Ricaviamo da questo diagramma ASM la corrispondente tabella degli stati: il risultato è mostrato qui sotto. Si noti che la tabella contiene *una riga per ogni percorso* di collegamento tra i blocchi di stato, perché descrive completamente, come il diagramma ASM, l'evoluzione degli stati della macchina.

		Stato Attuale			Stato Successivo	
Stato		Ingresso	Uscita	Stato		
Nome	X1	X0	IN	OUT	iX1	iX0
a	0	0	0	0	0	0
a	0	0	1	1	0	1
b	0	1	-	1	1	0
c	1	0	0	1	1	1
c	1	0	1	0	1	0
d	1	1	-	1	0	0

La tabella non è ordinata in sequenza temporale, e potrebbe essere scritta in qualsiasi ordine. Ciò che esprime deve essere letto riga per riga, indipenden-

temente: elenca, stato per stato, quali sono le uscite attuali (condizionate e non), e quale sarà lo stato successivo, in funzione del valore degli ingressi. Ad esempio, per gli stati (a) e (c) sono presenti in tabella due righe per ciascuno, per descrivere la dipendenza dall'ingresso *IN*. Per gli altri due stati, invece, una sola riga è sufficiente poiché le uscite e il prossimo stato non dipendono dal valore dell'ingresso.

La descrizione tabellare di una MSF è un metodo che si presta molto bene alla sua rappresentazione su di un calcolatore. Tuttavia è evidente che il diagramma ASM fornisce una migliore e più comprensibile visione di insieme dell'algoritmo della MSF e, per questo motivo, è la rappresentazione che usiamo in questo testo per comprendere al meglio la progettazione delle reti sequenziali. Tuttavia, da un punto di vista pratico, tale metodo cade in difficoltà quando si tratta di descrivere macchine con un grande numero di stati, poiché a quel punto diventa impossibile cogliere la visione di insieme. In questi casi diventa preferibile l'utilizzo di un linguaggio di descrizione dell'hardware (come ad esempio il VHDL e il VERILOG).

7.4.3 Sintesi dalla tabella degli stati

Dalla tabella degli stati è possibile estrarre le tavole di verità delle reti dello stato successivo e delle uscite, che riportiamo qui di seguito:

Rete dello Stato Successivo				
Stato Attuale		Ingresso	Stato Successivo	
X1	X0	IN	iX1	iX0
0	0	0	0	0
0	0	1	0	1
0	1	-	1	0
1	0	0	1	1
1	0	1	1	0
1	1	-	0	0

Rete delle Uscite			
Stato Attuale		Ingresso	Uscita
X1	X0	IN	OUT
0	0	0	0
0	0	1	1
0	1	-	1
1	0	0	1
1	0	1	0
1	1	-	1

Da queste tavole è possibile passare, con il metodo che più ci è comodo, alla sintesi delle reti in questione e, quindi, alla realizzazione circuitale della rete. Se utilizziamo le mappe, da queste tabelle ne ricaviamo due per lo stato successivo ($iX1$, $iX0$), e una per l'uscita (OUT), come riportato qui sotto:

		$X0$			
		0	1	0	1
IN	0	1	0	1	
	1	1	0	1	
		$X1$			

		$X0$			
		0	1	0	0
IN	0	1	0	0	
	1	0	0	0	
		$X1$			

		$X0$			
		0	1	1	1
IN	0	1	0	1	
	1	0	1	1	
		$X1$			

In alternativa, possiamo disegnare delle mappe a variabili riportate. Le variabili di stato sono utilizzate come coordinate della mappa, mentre gli ingressi (qui solo IN) saranno riportati al suo interno:

		$X0$	
		0	1
$X1$	0	1	
	1	0	

		$X0$	
		IN	0
$X1$	0	IN	
	1	\overline{IN}	

		$X0$	
		IN	1
$X1$	0	IN	
	1	\overline{IN}	

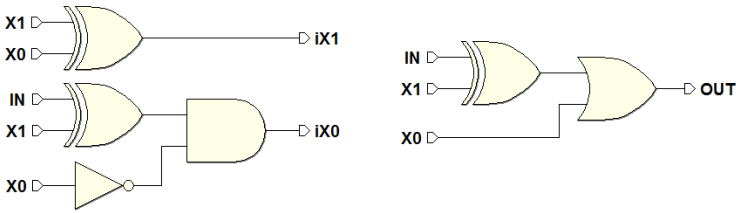
Raggruppate opportunamente queste mappe, otteniamo le equazioni che descrivono le funzioni cercate:

$$\begin{aligned}
 iX1 &= (\overline{X0} \cdot X1) + (X0 \cdot \overline{X1}) \\
 iX0 &= (IN \cdot \overline{X1} \cdot \overline{X0}) + (\overline{IN} \cdot X1 \cdot \overline{X0}) \\
 OUT &= X0 + (\overline{IN} \cdot X1) + (IN \cdot \overline{X1})
 \end{aligned}$$

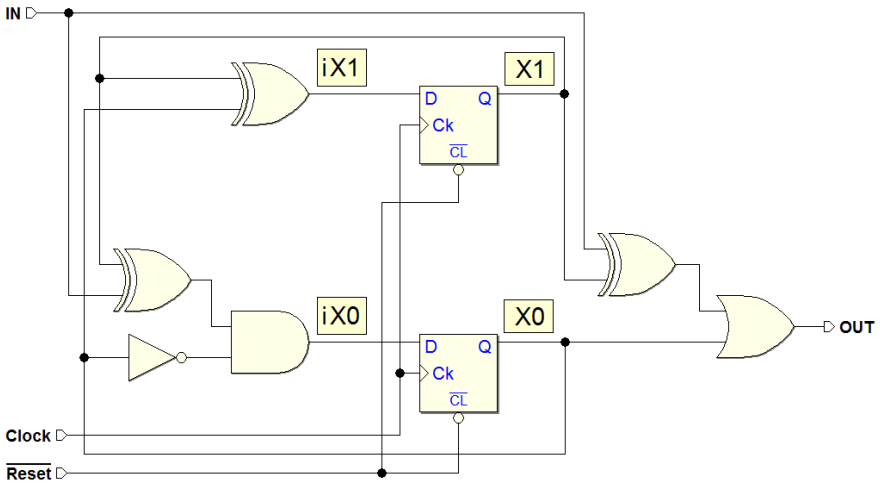
Utilizzando le porte EXOR e svolgendo alcuni passaggi, rendiamo più concise queste espressioni:

$$\begin{aligned}
 iX1 &= X0 \oplus X1 \\
 iX0 &= (IN \oplus X1) \cdot \overline{X0} \\
 OUT &= (IN \oplus X1) + X0
 \end{aligned}$$

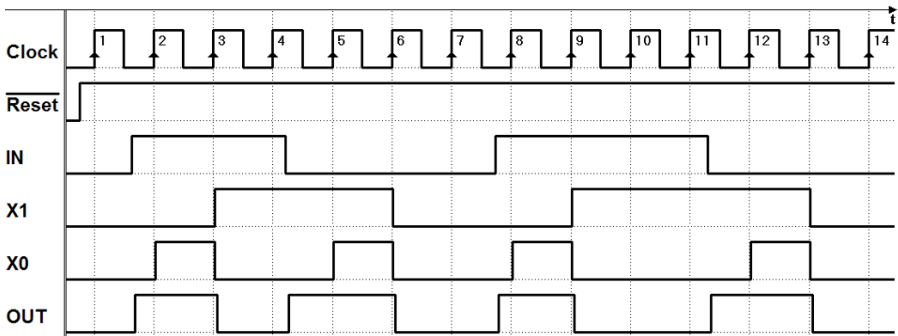
Rappresentate nella forma circuitale, ecco le tre funzioni:



La rete complessiva è quella visibile nella figura seguente, completa dei circuiti di inizializzazione asincrona (l'ingresso *Reset* che agisce sui \overline{CL} dei flip-flop):



Per completezza, esaminiamo una simulazione temporale di questa rete:

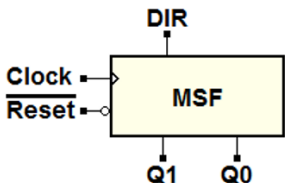


Nel tracciato si osserva il comportamento dell'uscita *OUT* che, come desiderato, si attiva a seguito del cambiamento dell'ingresso *IN*, senza aspettare il fronte di salita del *Clock*. L'uscita *OUT* è poi essere mantenuta attiva nello stato successivo, per un intero ciclo.

7.4.4 Esempi di sintesi di MSF sincrone

Esempio 1 (contatore bidirezionale a due bit)

In questo esempio eseguiamo la sintesi di una MSF sincrona, con un ingresso sincrono, avente le terminazioni descritte dalla figura qui sotto:

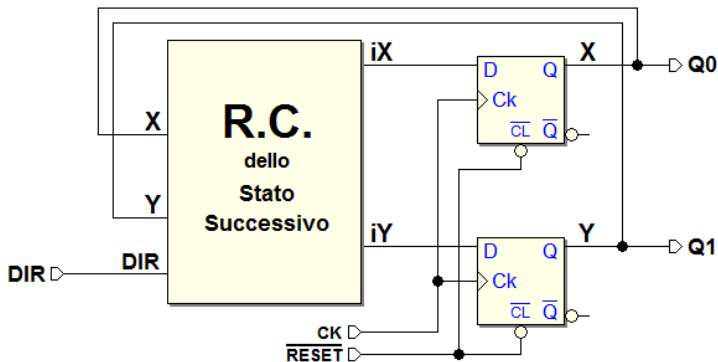
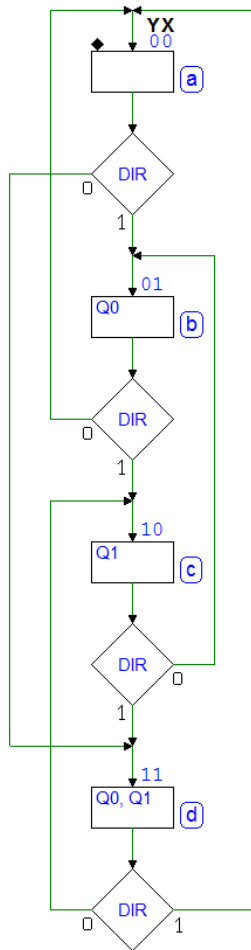


Il diagramma ASM è riportato nella figura qui a destra: si tratta di un *contatore binario bidirezionale* ed è stato analizzato e definito a pag. 279. Non sono presenti uscite condizionate, per cui si tratta di una *macchina di Moore*.

Ai fini della sintesi, nel diagramma è stata assegnata una codifica agli stati (visibile nell'angolo in alto a destra degli stessi), utilizzando i criteri qui descritti. Con 4 stati, possiamo utilizzare solo 2 variabili di stato (X e Y). Inoltre, dato che le uscite $Q0$ e $Q1$ compaiono nei 4 stati in tutte le loro possibili combinazioni, una buona scelta è stata di far coincidere le variabili di stato con le uscite:

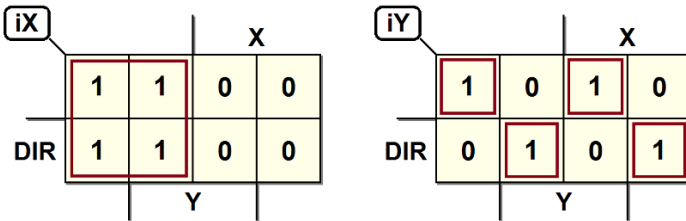
$$Q0 = X \quad Q1 = Y$$

Nella figura qui sotto è rappresentato lo schema a blocchi della rete che si vuole ottenere: sono evidenziate le variabili di stato X e Y (le uscite dei flip-flop) e le variabili iX e iY che la rete combinatoria dello stato successivo deve produrre, in funzione dello stato X, Y e dell'ingresso DIR :



Dal diagramma ASM ricaviamo la tabella dello stato successivo (quella delle uscite è inutile, per come le abbiamo definite), e poi le mappe:

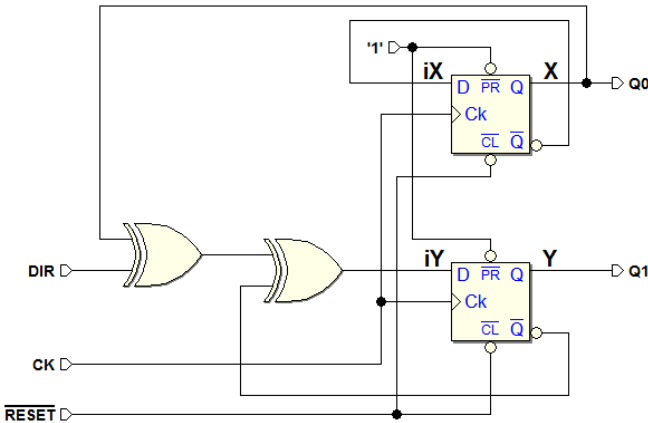
Stato	Y	X	DIR	Stato Succ.	iY	iX
a	0	0	0	d	1	1
a	0	0	1	b	0	1
b	0	1	0	a	0	0
b	0	1	1	c	1	0
c	1	0	0	b	0	1
c	1	0	1	d	1	1
d	1	1	0	c	1	0
d	1	1	1	a	0	0



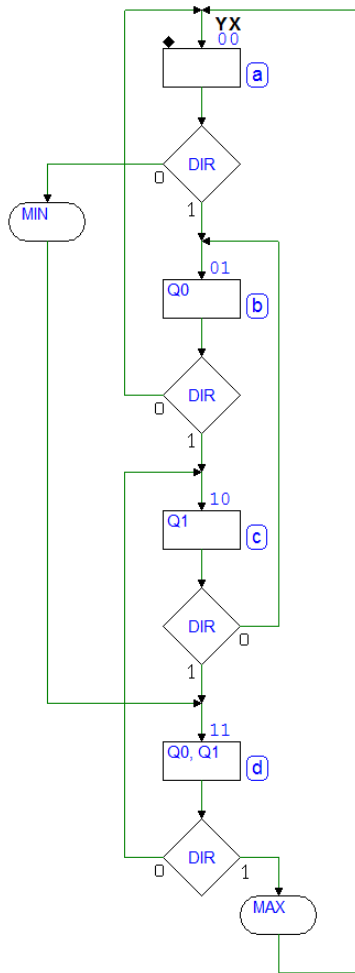
Da cui si ricava (si noti la trasformazione di iY in albero di EXOR, possibile in quanto la mappa è a scacchiera):

$$\begin{aligned}
 iX &= \bar{X} \\
 iY &= \overline{DIR} \cdot \bar{X} \cdot \bar{Y} + DIR \cdot \bar{X} \cdot Y + \overline{DIR} \cdot X \cdot Y + DIR \cdot X \cdot \bar{Y} \\
 &= (\overline{DIR} \cdot \bar{X} + DIR \cdot X) \cdot \bar{Y} + (DIR \cdot \bar{X} + \overline{DIR} \cdot X) \cdot Y \\
 &= (\overline{DIR \oplus X}) \cdot \bar{Y} + (DIR \oplus X) \cdot Y = \overline{(DIR \oplus X \oplus Y)} \\
 &= DIR \oplus X \oplus \bar{Y}
 \end{aligned}$$

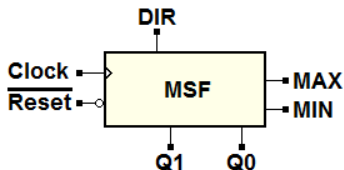
Ed infine disegniamo lo schema logico della rete che ne risulta:



Esempio 2 (contatore bidirezionale, con uscite di massimo e minimo)



In questo esempio aggiungiamo, al contatore binario bidirezionale dell'esempio precedente, due uscite per indicare il raggiungimento del numero massimo (se in conteggio *avanti*) o di quello minimo (se *indietro*).



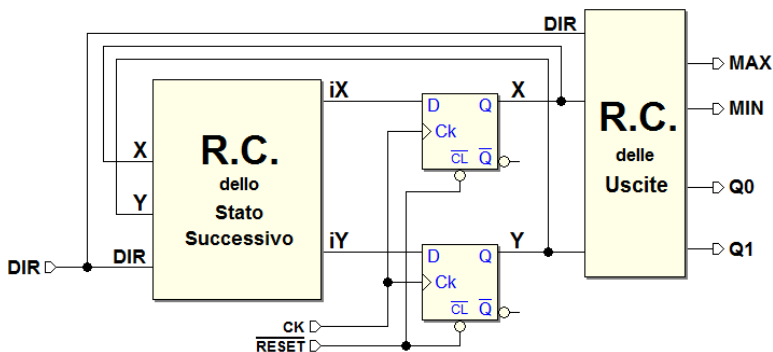
Il diagramma ASM, qui a sinistra, è identico a quello dell'esempio precedente per quanto riguarda l'evoluzione degli stati. Sono presenti però due *uscite condizionate* e, quindi, è una MSF di *Mealy*.

Come si vede, *MIN* si attiva *mentre* siamo in (a), purché *DIR* = 0. *MIN* indica quindi la condizione di *conteggio terminale* nella direzione *all'indietro*. In modo del tutto analogo, *MAX* è attivato in (d) se *DIR* = 1 e indica la *conteggio terminale* nell'altra direzione.

Gli stati sono assegnati in modo identico all'esempio precedente (variabili *X* e *Y*), con le *uscite di stato* coincidenti con esse:

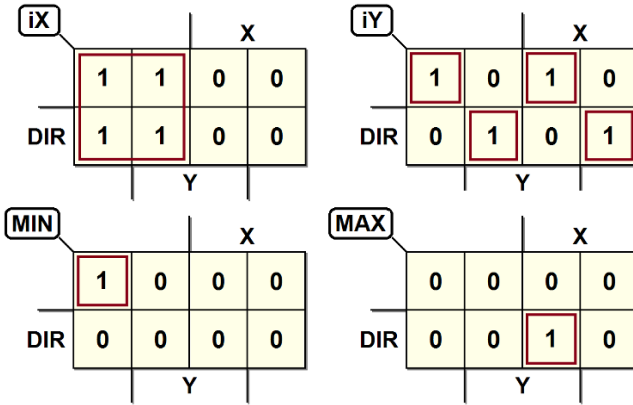
$$Q0 = X \quad Q1 = Y$$

Qui sotto lo schema a blocchi della rete che si vuole sintetizzare. Si noti la struttura di *Mealy*, con le uscite *MAX* e *MIN* che dipendono anche dall'ingresso *DIR*:



Dal diagramma ASM ricaviamo la tabella degli stati, che include stato successivo e uscite; dalla tabella, poi, ricaviamo le mappe:

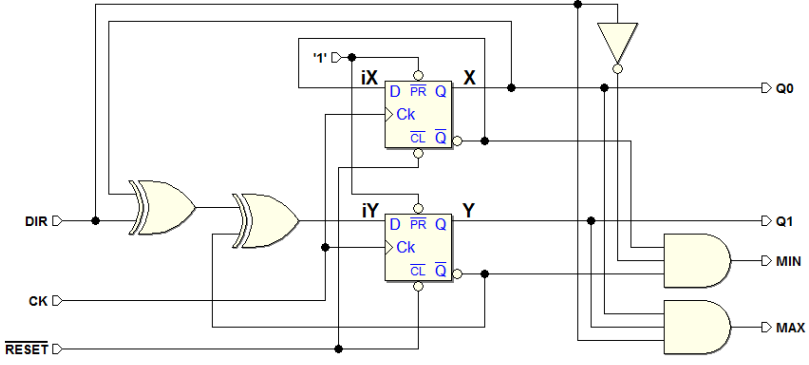
Stato	Y	X	DIR	Stato S.	iY	iX	Q1	Q0	MIN	MAX
a	0	0	0	d	1	1	0	0	1	0
a	0	0	1	b	0	1	0	0	0	0
b	0	1	0	a	0	0	0	1	0	0
b	0	1	1	c	1	0	0	1	0	0
c	1	0	0	b	0	1	1	0	0	0
c	1	0	1	d	1	1	1	0	0	0
d	1	1	0	c	1	0	1	1	0	0
d	1	1	1	a	0	0	1	1	0	1



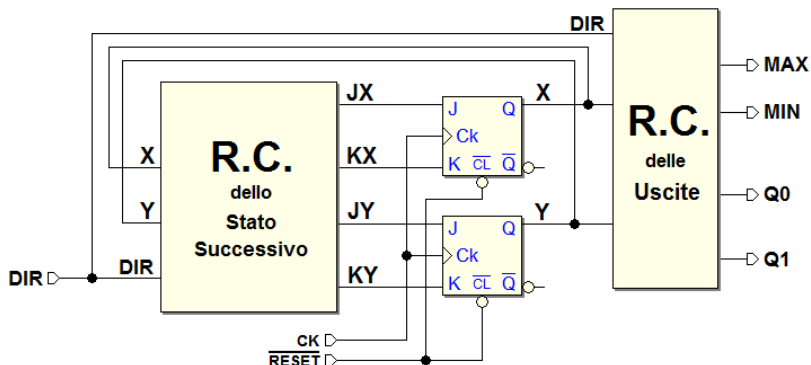
Le mappe di iX e iY sono identiche all'esempio precedente, e le mappe di $Q0$ e $Q1$ non servono per come le abbiamo definite; ricaviamo le espressioni:

$$\begin{aligned}
 iX &= \bar{X} & iY &= DIR \oplus X \oplus \bar{Y} \\
 Q0 &= X & Q1 &= Y \\
 MIN &= \overline{DIR} \cdot \bar{X} \cdot \bar{Y} & MAX &= DIR \cdot X \cdot Y
 \end{aligned}$$

La rete che risulta dalla sintesi è come quella dell'esempio precedente, con l'aggiunta della logica che genera le uscite MIN e MAX :

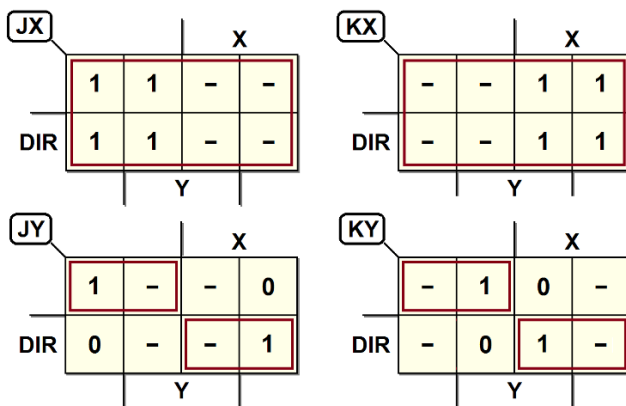


Ora, per esercizio, proviamo a rifare la sintesi, questa volta utilizzando flip-flop di tipo JK-PET invece che D-PET. La rete che ci aspettiamo di ricavare è descritta dal seguente schema a blocchi:

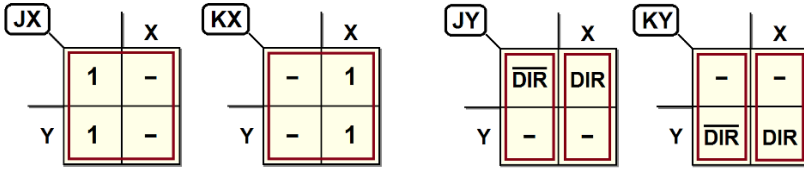


L'assegnazione degli stati e la rete delle uscite sono identiche alla versione con i D-PET. Cambia invece la rete dello stato successivo, che deve generare quattro funzioni invece che due: JX , KX , JY e KY . Dal diagramma ASM ricaviamo la tabella dello stato successivo, e da questa ricaviamo le mappe:

Stato	Y	X	DIR	Stato S.	JY	KY	JX	KX
a	0	0	0	d	1	-	1	-
a	0	0	1	b	0	-	1	-
b	0	1	0	a	0	-	-	1
b	0	1	1	c	1	-	-	1
c	1	0	0	b	-	1	1	-
c	1	0	1	d	-	0	1	-
d	1	1	0	c	-	0	-	1
d	1	1	1	a	-	1	-	1



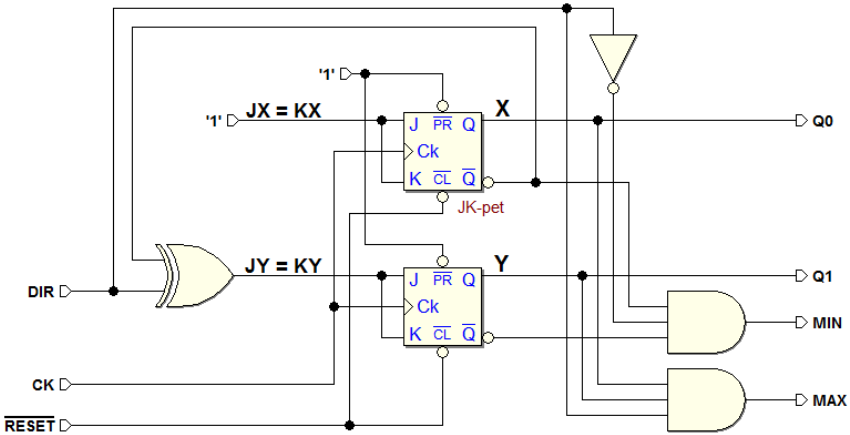
In alternativa, potremmo impostare le mappe con la tecnica delle variabili riportate, utilizzando X e Y come coordinate.



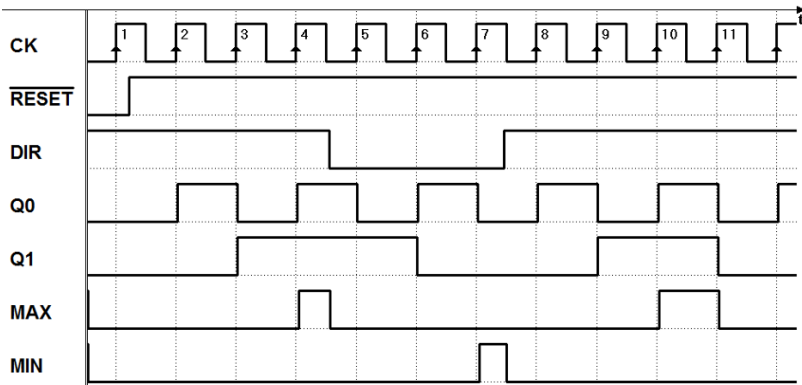
Raggruppate opportunamente queste mappe (o le prime, o le seconde), otteniamo le espressioni che descrivono le quattro funzioni cercate:

$$\begin{aligned}
 JX &= 1 & JY &= \overline{DIR} \cdot \overline{X} + DIR \cdot X = DIR \oplus \overline{X} \\
 KX &= 1 & KY &= \overline{DIR} \cdot \overline{X} + DIR \cdot X = DIR \oplus \overline{X}
 \end{aligned}$$

La rete che risulta dalla sintesi con i Flip-flop JK-PET è quindi la seguente:



Esaminiamo una simulazione temporale di questa rete:

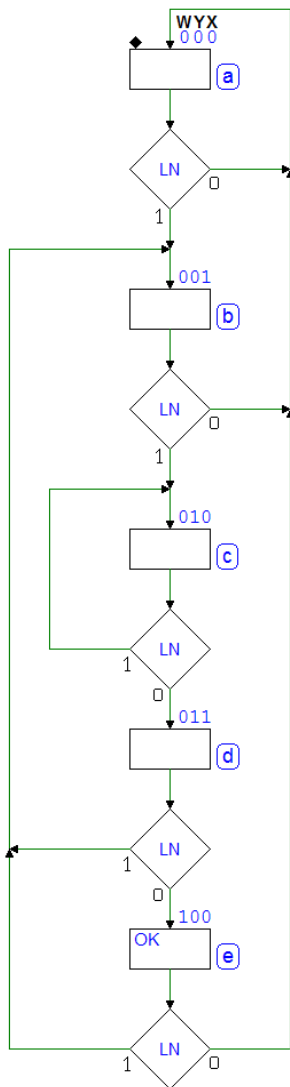


Osserviamo, ad esempio, cosa succede tra il fronte 4 e il 5 del clock: viene chiesto al contatore di cambiare la direzione di conteggio. L'uscita *MAX*, attiva nel conteggio avanti, viene disattivata subito, al cambiamento di *DIR*, essendo *MAX* una funzione combinatoria di *DIR* (oltre che dello stato).

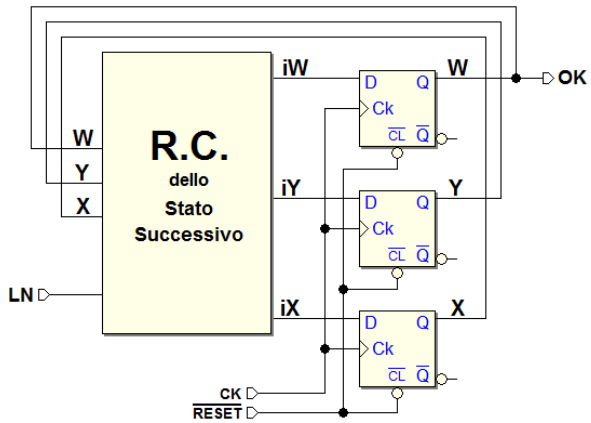
Esempio 3 (riconoscitore di sequenza)

In questo esempio eseguiamo la sintesi del *riconoscitore di sequenza* analizzato a pag. 303. Il suo diagramma ASM è riportato nuovamente qui sotto, sulla sinistra, con l'aggiunta della assegnazione degli stati.

Dato che l'ingresso *LN* è sincrono e la *MSF* è sincrona, l'assegnazione dei codici può essere arbitraria; qui si è scelto di utilizzare una delle variabili di stato per generare direttamente l'uscita *OK*. Definiamo tre variabili di stato: *W*, *Y* e *X*, con il presupposto di definire $OK = W$; nel diagramma non ci sono uscite condizionate, per cui dalla sintesi otterremo una macchina di *Moore*.



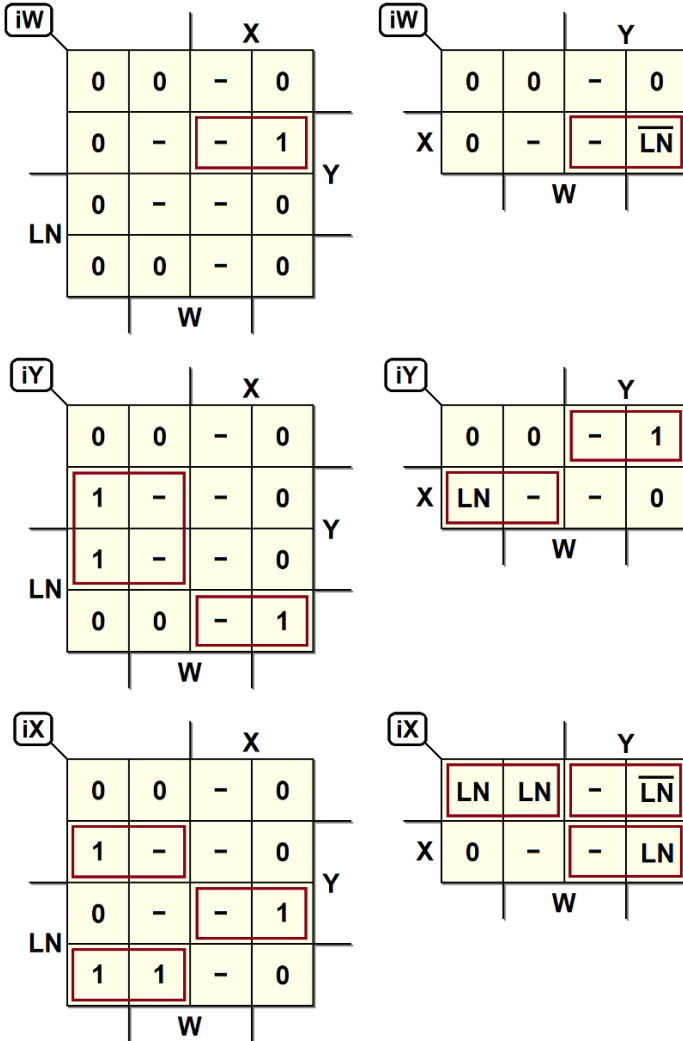
Lo schema a blocchi della rete è visibile qui sotto:



Nella rete sono evidenziate le variabili di stato *W*, *Y*, *X* e le uscite della rete dello stato successivo *iW*, *iY* e *iX*. Ricaviamo dal diagramma la tabella:

Stato	W	Y	X	LN	Stato Succ.	iW	iY	iX
a	0	0	0	0	a	0	0	0
a	0	0	0	1	b	0	0	1
b	0	0	1	0	a	0	0	0
b	0	0	1	1	c	0	1	0
c	0	1	0	0	d	0	1	1
c	0	1	0	1	c	0	1	0
d	0	1	1	0	e	1	0	0
d	0	1	1	1	b	0	0	1
e	1	0	0	0	a	0	0	0
e	1	0	0	1	b	0	0	1
-	1	0	1	-	-	-	-	-
-	1	1	-	-	-	-	-	-

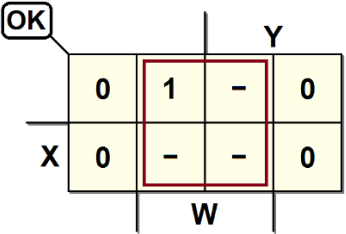
Dalla tabella dello stato successivo ricaviamo le mappe. Per confronto, potremmo compilare delle mappe *ordinarie* (sulla sinistra), oppure delle mappe *a variabili riportate*, più compatte (sulla destra):



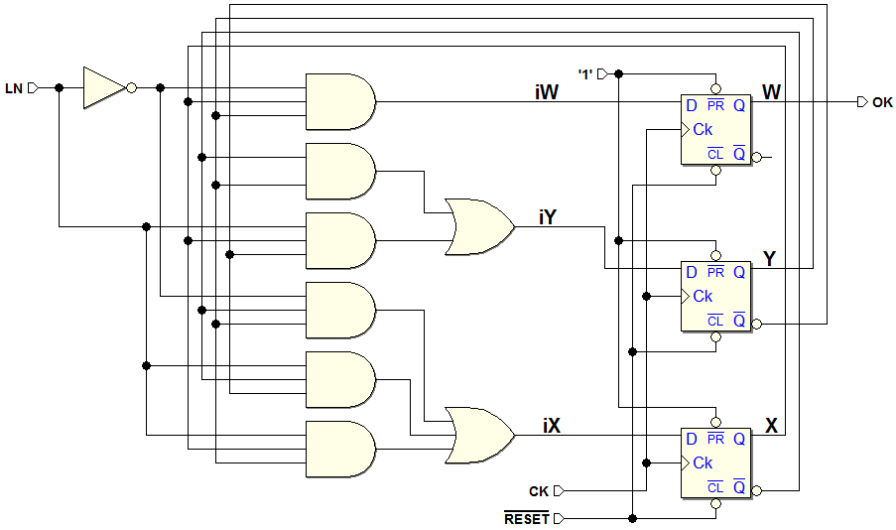
Procedendo con la lettura delle mappe, si ricavano le espressioni:

$$\begin{aligned}
 iW &= \overline{LN} \cdot X \cdot Y \\
 iY &= \overline{X} \cdot Y + LN \cdot X \cdot \overline{Y} \\
 iX &= \overline{LN} \cdot \overline{X} \cdot Y + LN \cdot \overline{X} \cdot \overline{Y} + LN \cdot X \cdot Y
 \end{aligned}$$

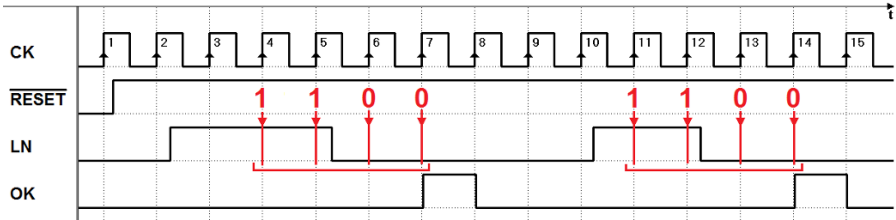
Infine, per completezza, anche se non ce ne sarebbe bisogno in quanto avevamo già definito in precedenza $OK = W$, ecco la mappa dell'uscita OK :



Dalla sintesi risulta quindi la rete:

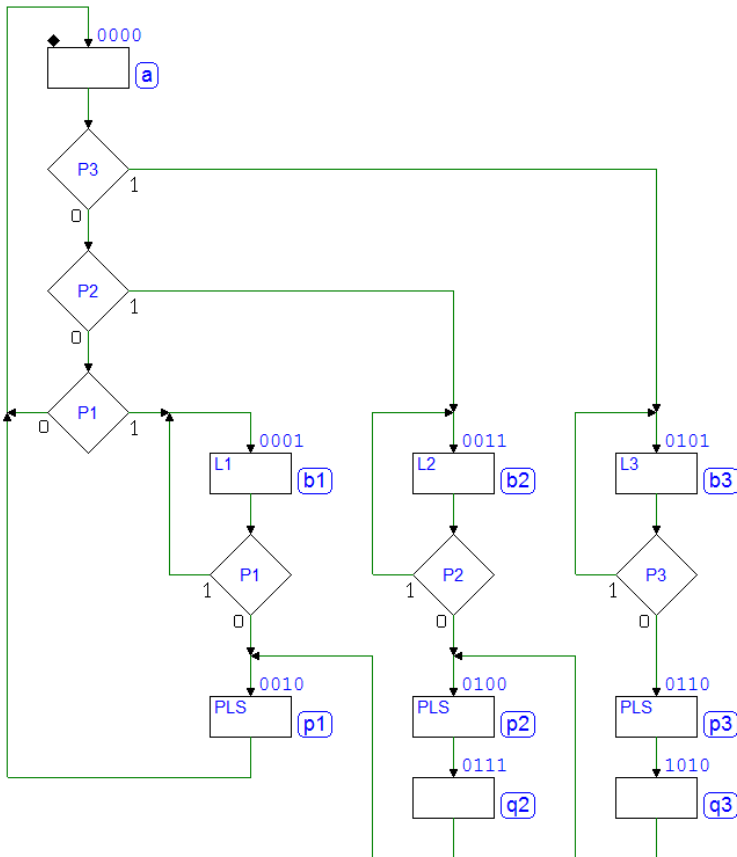


Qui sotto una simulazione temporale di verifica della rete. Sono evidenziate le due sequenze 1-1-0-0 riconosciute dall'algoritmo:



Esempio 4 (gestione di pulsanti)

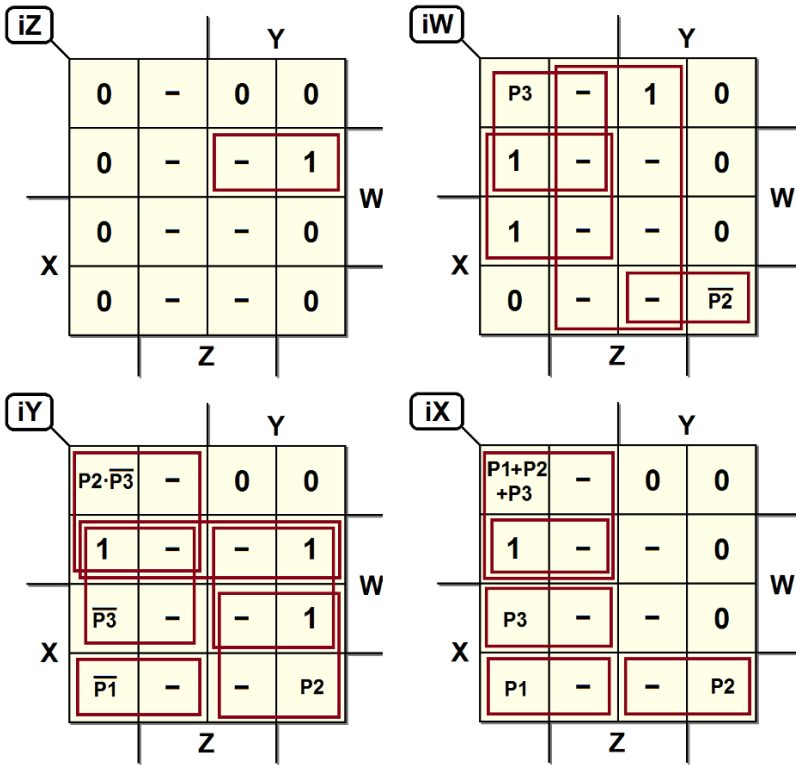
Nella figura seguente è riportato il diagramma ASM costruito nell'esercizio "Gestione di pulsanti", a pag. 313). Nel diagramma sono presenti 9 stati, per cui dobbiamo utilizzare 4 variabili di stato (Z , W , Y e X). L'assegnazione degli stati è stata eseguita in modo del tutto arbitrario, in quanto MSF sincrona con ingressi che ipotizziamo sincronizzati con il clock della MSF.



Anticipiamo che l'assegnazione scelta conduce ad una sintesi ed ad una rete decisamente complessa, come vedremo nelle pagine seguenti. Dalla particolare assegnazione possono dipendere reti tutte perfettamente equivalenti, ma di notevoli diversità in termini di complessità, e quindi di costo.

Grazie alla disponibilità dei sistemi CAD per la progettazione, da moltissimi anni questo tipo di scelta e la stessa sintesi viene delegata ad algoritmi automatici, dove il progettista interviene normalmente solo nell'impostare delle regole e dei vincoli, all'inizio, e nel verificare la funzionalità del sistema alla fine della sintesi. Qui l'esempio è portato avanti in ogni caso con modalità *manuali*, ai fini della comprensione generale dell'argomento trattato.

Dalla tabella otteniamo le mappe di iZ , iW , iY e iX . Dobbiamo disegnarle a *variabili riportate*, perché abbiamo troppi ingressi ($P1$, $P2$, $P3$ e le 4 variabili di stato):



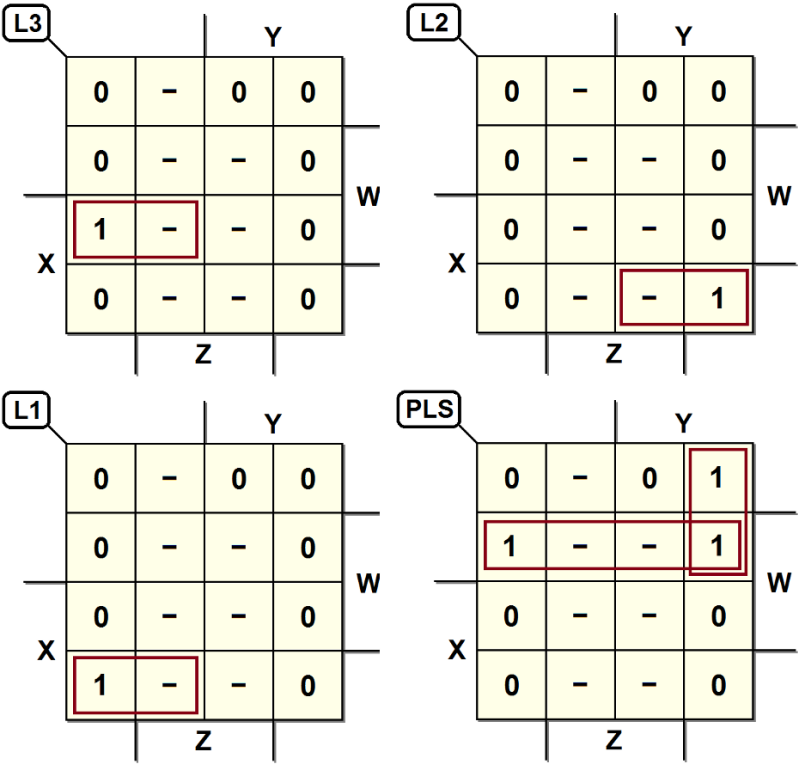
Raggruppando opportunamente le mappe, ricaviamo le espressioni:

$$\begin{aligned}
 iZ &= \bar{X} \cdot Y \cdot W \\
 iW &= W \cdot \bar{Y} + Z + \\
 &\quad P3 \cdot \bar{X} \cdot \bar{Y} + \bar{P2} \cdot X \cdot Y \cdot \bar{W} \\
 iY &= \bar{X} \cdot W + Y \cdot W + \\
 &\quad P2 \cdot \bar{P3} \cdot \bar{X} \cdot \bar{Y} + \bar{P3} \cdot W \cdot \bar{Y} + \\
 &\quad \bar{P1} \cdot X \cdot \bar{Y} \cdot \bar{W} + P2 \cdot X \cdot Y \\
 iX &= \bar{X} \cdot \bar{Y} \cdot W + (P1 + P2 + P3) \cdot \bar{X} \cdot \bar{Y} + \\
 &\quad P3 \cdot X \cdot \bar{Y} \cdot W + P1 \cdot X \cdot \bar{Y} \cdot \bar{W} + \\
 &\quad P2 \cdot X \cdot Y \cdot \bar{W}
 \end{aligned}$$

Torniamo al diagramma ASM e ricaviamo la tabella delle uscite:

Stato	Z	W	Y	X	L3	L2	L1	PLS
a	0	0	0	0	0	0	0	0
b1	0	0	0	1	0	0	1	0
p1	0	0	1	0	0	0	0	1
b2	0	0	1	1	0	1	0	0
p2	0	1	0	0	0	0	0	1
b3	0	1	0	1	1	0	0	0
p3	0	1	1	0	0	0	0	1
q2	0	1	1	1	0	0	0	0
-	1	0	0	-	-	-	-	-
q3	1	0	1	0	0	0	0	0
-	1	0	1	1	-	-	-	-
-	1	1	-	-	-	-	-	-

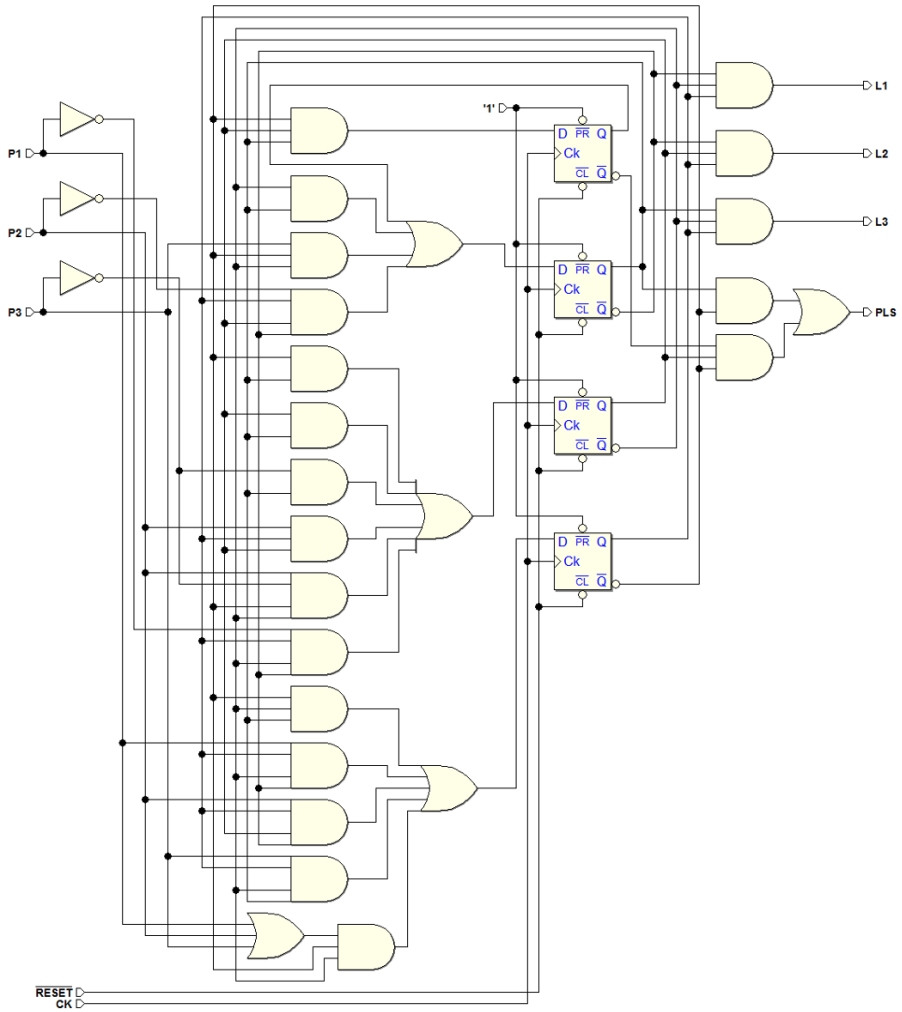
Traduciamo la tabella nelle quattro mappe di L3, L2, L1 e PLS. In questo caso non è utile disegnarle a *variabili riportate*, in quanto le uscite della rete sono funzione di quattro sole variabili, quelle di stato:



Dai raggruppamenti nelle mappe delle uscite ricaviamo le espressioni:

$$\begin{aligned}
 L1 &= X \cdot \bar{Y} \cdot \bar{W} \\
 L2 &= X \cdot Y \cdot \bar{W} \\
 L3 &= X \cdot \bar{Y} \cdot W \\
 PLS &= \bar{X} \cdot W + \bar{X} \cdot Y \cdot \bar{Z}
 \end{aligned}$$

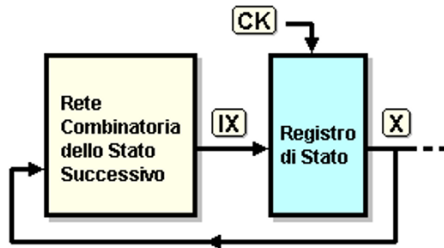
Infine, dalla sintesi risulta la seguente rete:



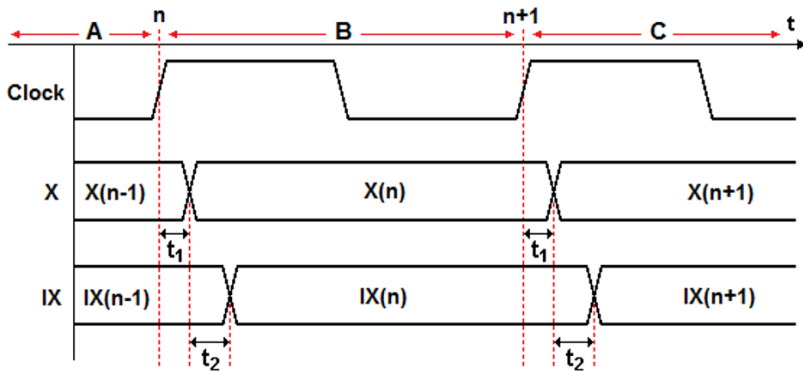
7.5 Comportamento nel tempo della MSF sincrona

Comportamento nel tempo della MSF senza ingressi

Per semplicità, iniziamo a considerare una MSF sincrona priva di ingressi (vedi figura seguente), e trascuriamo, per il momento, le uscite, concentrandoci sulla logica di generazione dello stato successivo:



Il diagramma temporale qui sotto ne rappresenta l'evoluzione degli stati nel tempo. Il vettore X rappresenta lo stato della MSF, con i suoi cambiamenti, conseguenti all'azione del clock:



Alla fine del ciclo di clock “A”, la macchina si trova nello stato $X(n-1)$, mentre sono già disponibili gli ingressi del registro di stato $IX(n-1)$ relativi al successivo stato $X(n)$.

Al verificarsi del fronte di salita del clock n , si passa nel ciclo di clock “B”, ove il vettore di stato $X(n-1)$ viene *sostituito* con quello generato dalla rete dello stato successivo $IX(n-1)$, ottenendo $X(n)$.

E' evidenziato il ritardo t_1 tra il fronte di clock e l'istante del cambiamento del vettore X (tale ritardo è causato dal tempo di propagazione degli elementi di memoria che costituiscono il registro di stato).

Dopo il ritardo t_1 è quindi attivo lo stato $X(n)$ e, grazie alla retroazione, $X(n)$ viene riportato all'ingresso della rete combinatoria dello stato successivo. Questa renderà disponibile, dopo il ritardo t_2 , il nuovo valore del vettore

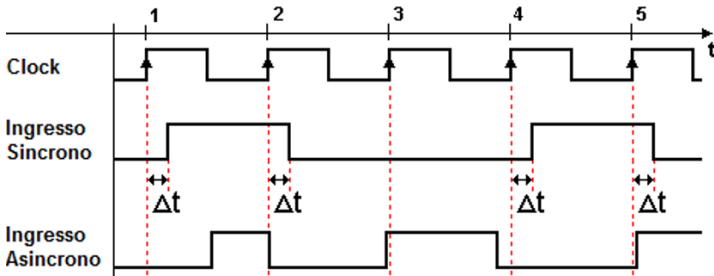
$IX(n)$, che rappresenta il nuovo stato successivo, predisposto per il prossimo ciclo di *Clock* (questo ulteriore ritardo è dovuto al tempo di propagazione della rete combinatoria).

Tutta la sequenza esaminata si ripeterà a partire dal prossimo fronte di salita del clock, quando il valore $IX(n)$ sarà caricato nel registro di stato, producendo nel ciclo di clock “C” lo stato successivo $X(n+1)$... e via di seguito. Si noti che, affinché gli aventi possano succedersi come descritti, il periodo del clock T deve essere maggiore della somma dei ritardi considerati ($T > t_1 + t_2$).

Comportamento nel tempo della MSF con ingressi sincroni e asincroni

Occorre distinguere, tra gli ingressi della MSF, tra “*sincroni*” e “*asincroni*”. Ricordiamo che gli *ingressi sincroni* hanno la proprietà di cambiare livello in *intervalli di tempo ben definiti e costanti rispetto all’evento attivo del clock*, in modo la rete li legga senza ambiguità.

Ciò implica che siano *stabili* nel momento in cui lo stato successivo viene caricato nel registro di stato. Tipico esempio di *ingresso sincrono* è quello che proviene da un’altra rete sequenziale che condivide lo stesso clock, e che quindi genera tale segnale con un ritardo costante rispetto al clock (vedi figura):



Gli *ingressi asincroni*, invece, potendo cambiare in qualunque momento, rischiano di essere letti dalla rete *quando non sono stabili* (ossia quando stanno cambiando), causando quindi errori nella transizione di stato. Un esempio di ingresso asincrono è dato dal segnale proveniente da un interruttore azionato manualmente, e che quindi, per sua natura, del tutto slegato dal clock.

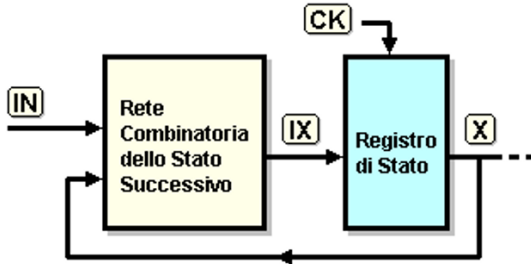
Nella figura qui sopra sono rappresentate alcune *transizioni* di un *ingresso asincrono* da considerare critiche, poiché avvengono in corrispondenza o quasi di un fronte di clock (istanti 2, 3 e 5). La criticità è dovuta ai tempi di propagazione della rete dello stato successivo e del registro di stato.

In generale, il cambiamento del valore di un ingresso si propaga attraverso la rete dello stato successivo con *ritardi leggermente diversi*, variabile per variabile. Se la transizione dell’ingresso avviene troppo *in prossimità del fronte attivo del clock*, a causa delle differenze di ritardo, alcune delle variazioni

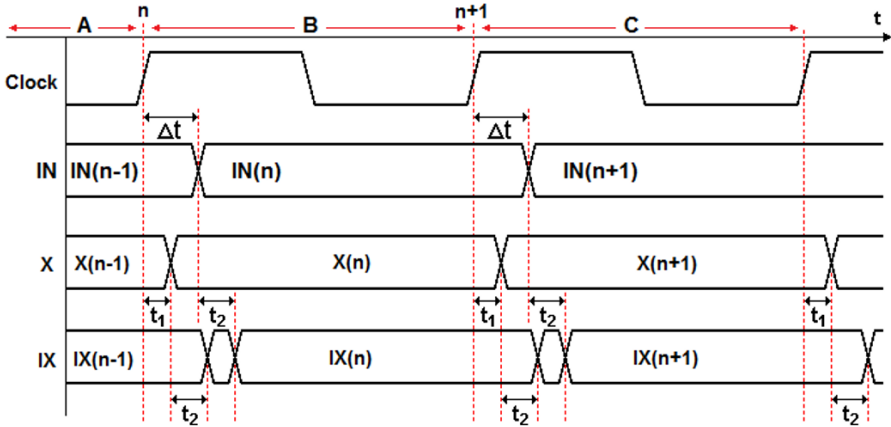
potrebbero essere acquisite dal registro di stato, ma altre no, dando origine a transizioni di stato errate, come spiegato più sotto.

Ingressi sincroni

Esaminiamo ora, nei dettagli, i tempi di una MSF sincrona con ingressi *sincroni*, continuando a ignorare, per il momento, le uscite (vedi figura):



Come abbiamo visto, in questo caso, lo stato successivo è ottenuto combinando lo stato attuale X con gli ingressi IN . Il diagramma temporale qui sotto esemplifica l'evoluzione degli stati, in presenza di ingressi sincroni:



Al termine del ciclo di clock “A”, la macchina si trova nello stato $X(n - 1)$, sono disponibili gli ingressi del registro di stato $IX(n - 1)$, e sono stabili gli ingressi $IN(n - 1)$. Al fronte di salita del clock n , si passa nel ciclo di clock B , ove il vettore di stato $X(n - 1)$ viene sostituito con quello generato dalla rete dello stato successivo $IX(n - 1)$, ottenendo $X(n)$.

Come nel caso della rete senza ingressi, trascorre un tempo di ritardo t_1 tra il fronte di clock e l’istante del cambiamento del vettore X (dovuto, ricordiamo, al registro di stato). Il nuovo stato $X(n)$, ad opera del collegamento in retroazione, viene riportato all’ingresso della rete combinatoria dello stato successivo. Questa, trascorso il suo tempo di propagazione t_2 , produrrà

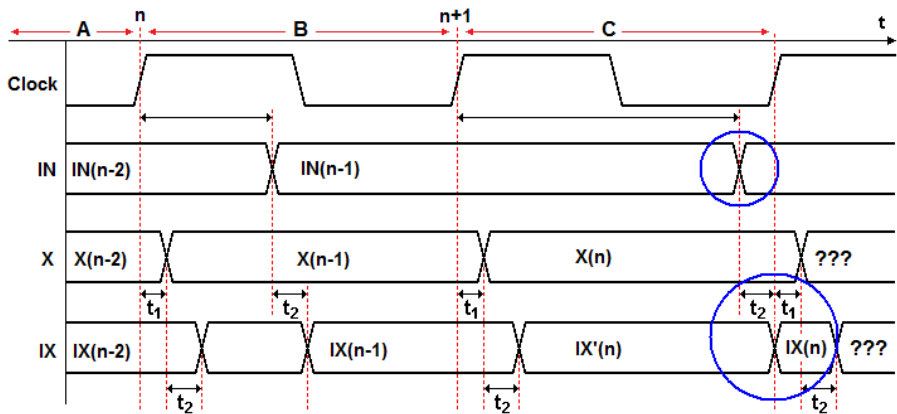
un nuovo valore per il vettore $IX(n)$, per il momento non ancora definitivo, tuttavia, perché nel frattempo *sono cambiati gli ingressi*.

Il nuovo valore degli ingressi $IN(n)$ (avvenuto in relazione sincrona con il clock), provoca un ulteriore lavoro per la rete combinatoria dello stato successivo che, con il ritardo di propagazione t_2 , produrrà il valore definitivo di $IX(n)$, funzione del nuovo valore degli $IN(n)$ e dello stato corrente $X(n)$.

Ad ogni fronte attivo di clock, la sequenza si ripeterà in modo analogo. Si noti che la natura sincrona degli ingressi consente di produrre un vettore IX sempre stabile in corrispondenza del successivo fronte attivo del clock.

Ingressi asincroni

Nel caso di una MSF sincrona con ingressi *asincroni* la transizione di stato può avvenire in modo errato. Il diagramma temporale qui sotto riportato esemplifica due casi possibili, in relazione al cambiamento *asincrono* degli ingressi. Nel primo esempio, durante il ciclo di clock "B", gli *ingressi asincroni* cambiano in un istante che risulta ancora compatibile con i tempi di ritardo delle reti coinvolte (in modo simile al caso degli ingressi sincroni):



Invece, nel secondo esempio, durante il ciclo di clock "C", gli *ingressi asincroni* cambiano troppo in ritardo: come evidenziato in figura, la rete combinatoria dello stato successivo produce il vettore $IX(n)$ in concomitanza con il fronte attivo del clock.

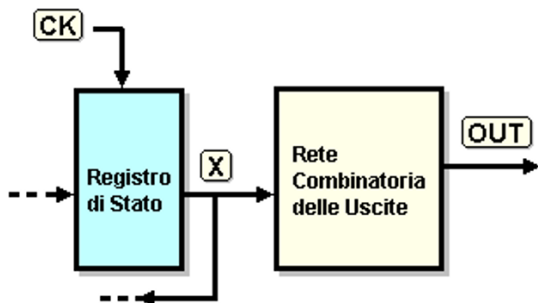
Questo fronte ordina al registro di stato di caricare il nuovo stato ma, come accennato prima, a causa dei *ritardi leggermente diversi* tra le singole reti che generano ciascuna variabile, alcune delle variazioni potrebbero giungere poco *prima*, e altre poco *dopo* il fronte di clock.

Il registro di stato potrebbe quindi caricare un vettore errato, e portare *imprevedibilmente* la macchina in *uno stato non previsto*. Per ovviare a questo grave inconveniente, occorre ricorrere ad una *particolare assegnazione degli*

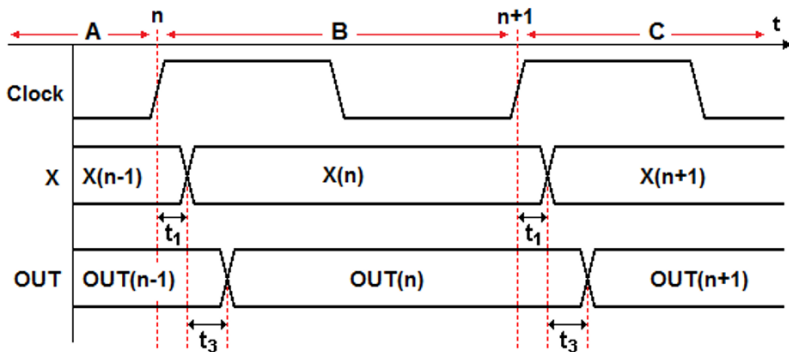
stati, tale che *una sola variabile di stato* dipenda dall'ingresso *asincrono*. Nella pratica progettuale, la soluzione standard consiste nella eliminazione degli ingressi asincroni mediante la sincronizzazione di questi.

Le uscite della MSF (modello di Moore)

Per quanto riguarda le uscite di una *MSF sincrona di Moore* non ci sono particolari osservazioni da fare, dal momento che sono generate dalla rete combinatoria delle uscite come semplice funzione dello stato (vedi figura):

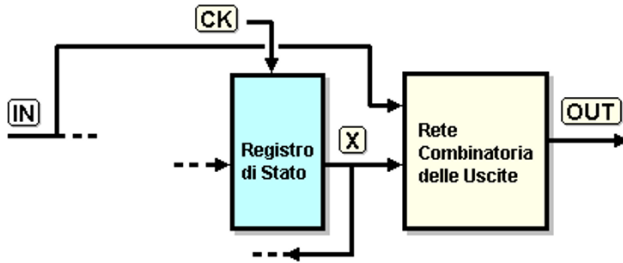


Come mostrato nel diagramma temporale qui sotto, le uscite cambiano quando cambia lo stato, con un tempo di ritardo $t_1 + t_3$, rispetto al fronte di clock, dove t_1 è il tempo tempo di propagazione del registro di stato, e t_3 è quello della rete combinatoria delle uscite:

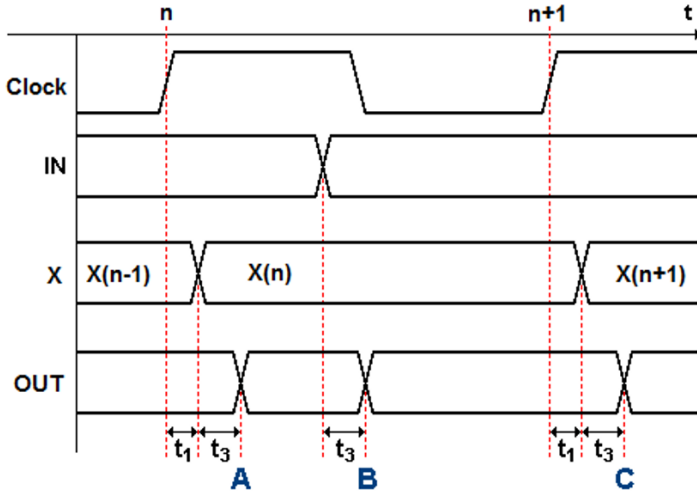


Le uscite della MSF (modello di Mealy)

Nel caso delle *MSF sincrone di Mealy*, l'andamento delle uscite condizionate dipenderà, in generale, dal comportamento nel tempo degli ingressi, per cui i casi esemplificabili sarebbero moltissimi. Nella figura seguente è messa in evidenza la generazione delle uscite condizionate, come funzione combinatoria dello stato e degli ingressi attuali.



Nel diagramma temporale qui sotto ne osserviamo solo una rappresentazione generale. Le uscite condizionate sono funzione dello stato e degli ingressi, per cui, come rappresentato in figura, le uscite potranno cambiare sia a seguito della transizione di stato (casi “A” e “C”), che di una variazione degli ingressi (caso “B”), all’interno dell’intervallo tra i due fronti attivi di clock:



Per quanto riguarda i tempi di propagazione, come evidenziato in figura, il ritardo dell’uscita va misurato rispetto alla causa che ha provocato il suo cambiamento. Esaminando il percorso dei segnali, si verifica facilmente che è pari a $t_1 + t_3$, se riferito al clock, oppure al solo t_3 , se riferito ad un ingresso.

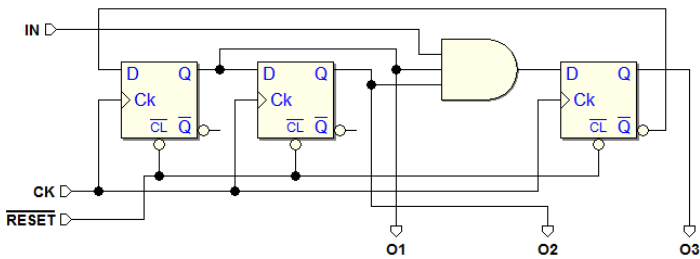
7.6 Esercizi

7.6.1 Analisi di reti sequenziali in termini di MSF

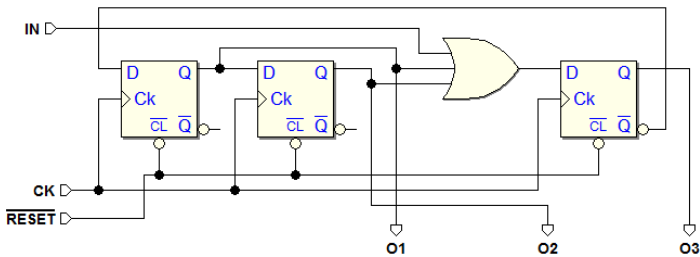
Analizzare le seguenti reti sequenziali sincrone mediante la costruzione di un diagramma ASM che ne descriva funzionalmente il comportamento.

Abbiamo reso disponibili, sul sito del simulatore, la traccia da completare di ognuna delle MSF, e lo schema *Deeds* con la rete assegnata. Gli schemi sono predisposti per inserire la vostra MSF accanto alla rete data, in modo da consentire un agevole confronto del loro comportamento mediante la simulazione.

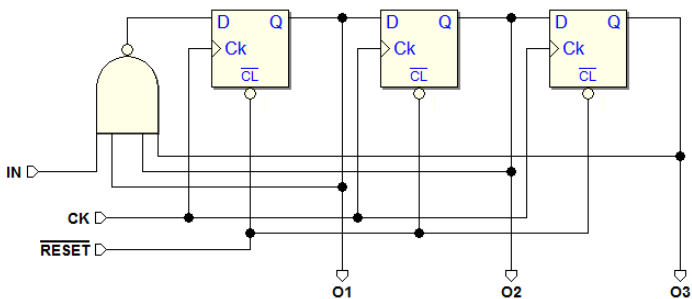
Rete 1



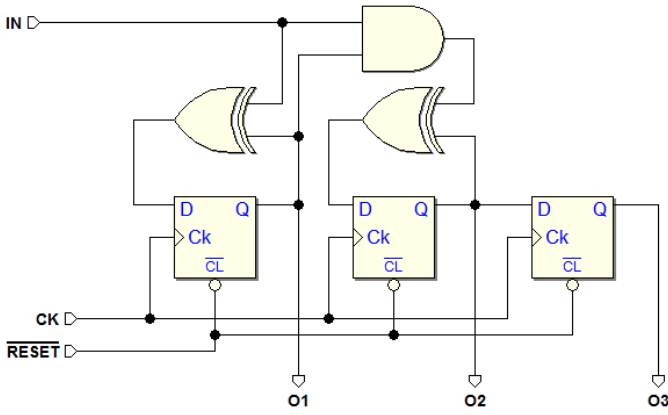
Rete 2



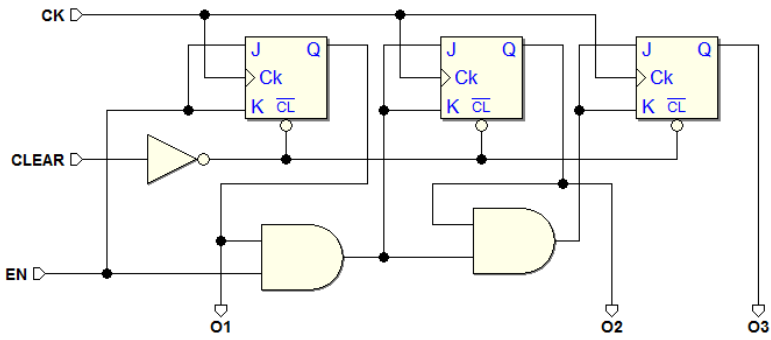
Rete 3



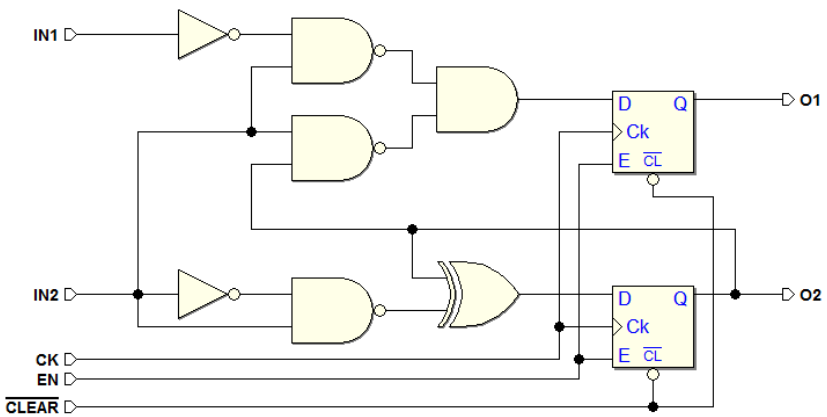
Rete 4



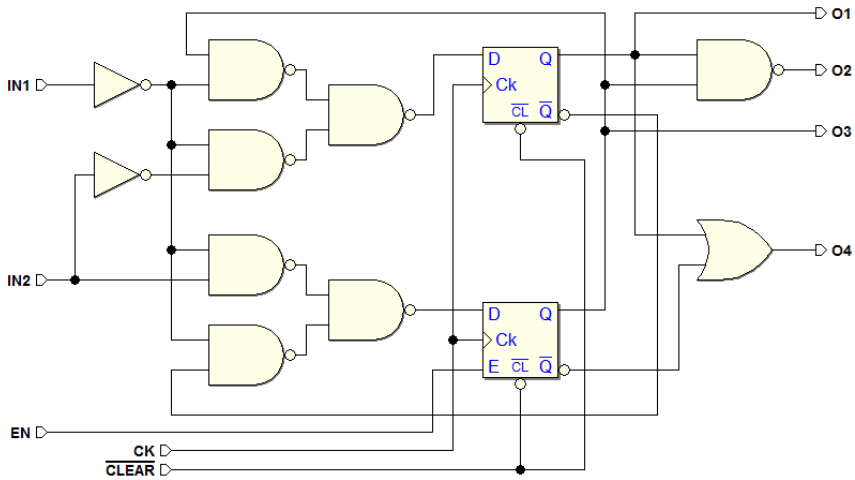
Rete 5



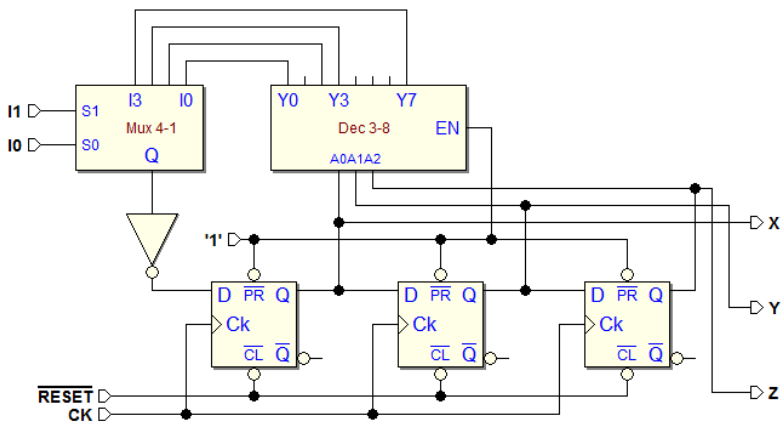
Rete 6



Rete 7



Rete 8



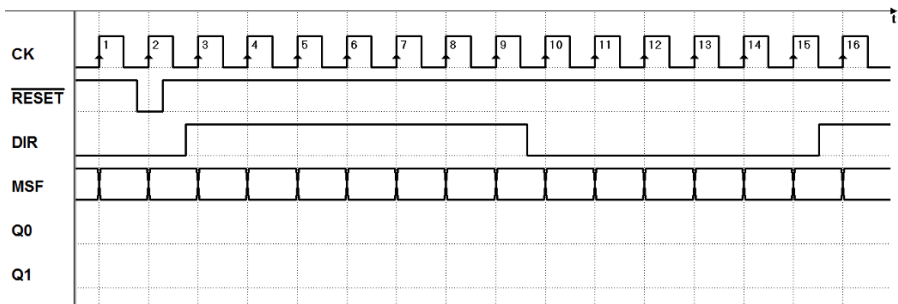
7.6.2 Progetto di MSF a partire da specifiche testuali

Per ognuno degli esercizi seguenti di progetto di MSF, si richiede di costruire il diagramma ASM sulla base delle specifiche date, e di completare il diagramma temporale proposto, includendo l'indicazione dello stato in cui si trova la MSF, in ogni ciclo di clock. Non è richiesto di effettuare la sintesi della MSF.

Sono disponibili, sul sito di *Deeds*, per ogni esercizio: una traccia da completare del diagramma ASM; un file PDF con il tracciato temporale qui proposto, da completare su carta senza usare il simulatore; uno schema in cui inserire la vostra MSF, per verificarne il corretto comportamento nel tempo.

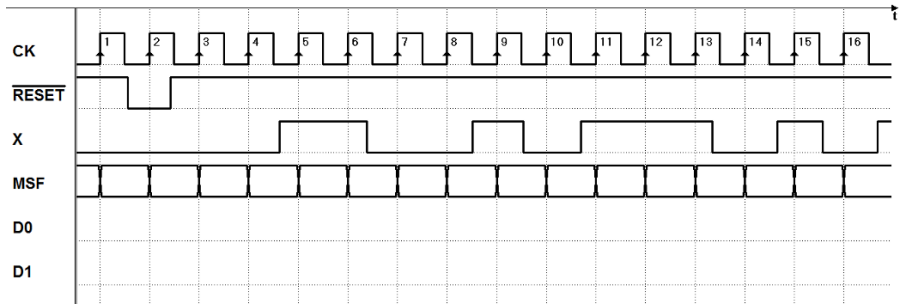
Esercizio 1

Si progetti, usando il metodo ASM, una MSF sincrona dotata di un ingresso sincrono *DIR* e due uscite *Q1* e *Q0*. Le uscite *Q1* (*MSB*) e *Q0* (*LSB*) rappresentano, in binario, i numeri decimali da 0 a 3. Il dispositivo si comporta come un contatore binario avanti-indietro non ciclico. L'ingresso *DIR* determina la direzione di conteggio (*DIR* = 1, conteggio avanti; *DIR* = 0, conteggio indietro), ed è sincrono con il clock *CK*. Una volta raggiunto uno degli estremi del conteggio, il dispositivo si ferma su tale valore. Il dispositivo può ripartire nella direzione di conteggio opposta, invertendo il valore di *DIR*.



Esercizio 2

Si progetti, usando il metodo ASM, una MSF sincrona avente un ingresso *X*, sincrono con il segnale di clock *CK*, e due uscite *D0*, *D1* il cui valore rappresenta in codice binario il numero di 1 letti sull'ingresso negli ultimi tre cicli di clock.

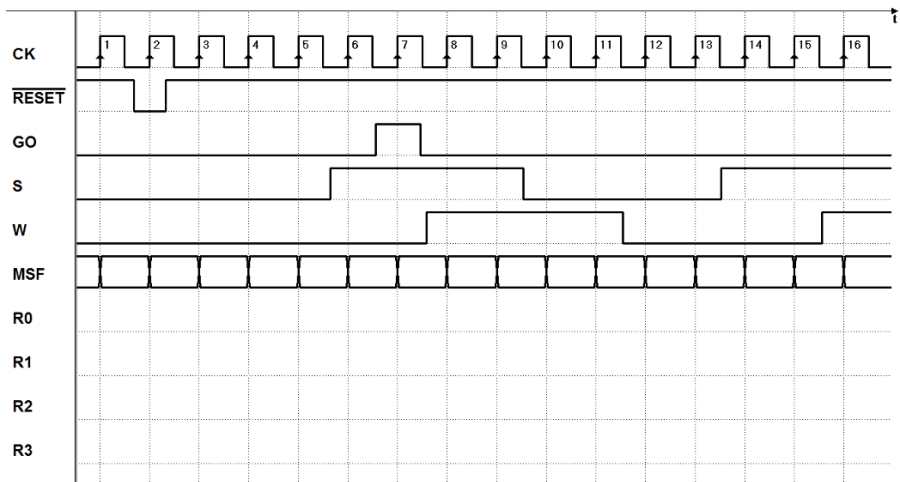


Esercizio 3

Si progetti, usando il metodo ASM, una MSF sincrona in grado di misurare il ritardo tra due segnali S e W presentati agli ingressi, secondo le specifiche:

1. Possiede un clock master CK e riceve in ingresso due segnali S e W , ad *onda quadra simmetrica*, di periodo pari a 4 volte quello del clock.
2. I cambiamenti di livello dei due ingressi S e W sono sincroni, ma i due segnali possono trovarsi tra loro in ritardo di 0, 1, 2 o 3 cicli di clock.
3. Quattro uscite $R0$, $R1$, $R2$ ed $R3$ indicheranno la misura del ritardo di W rispetto a S .
4. Un segnale sincrono di GO , della durata di un solo ciclo di clock, ordina alla macchina la sequenza di misura.

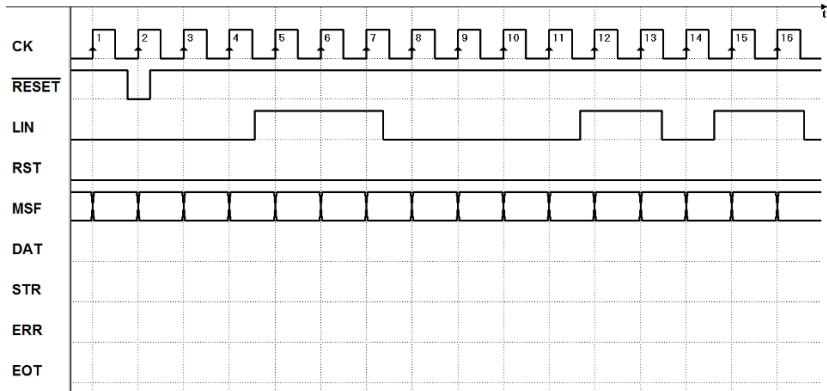
Tenere presente che la macchina deve rimanere inattiva fino all'arrivo di questo segnale, e che S e W possono essere ritenuti validi soltanto dal ciclo di GO (compreso) in avanti, ossia nulla si può ipotizzare sulla loro precedente posizione temporale, sia rispetto al segnale di GO , sia tra di loro.



Esercizio 4

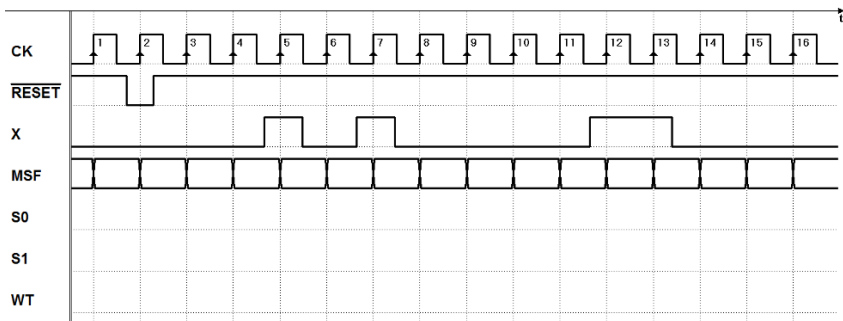
Si progetti, usando il metodo ASM, una MSF in grado di ricevere una sequenza seriale di dati binari *LIN*, sincrona con un clock *CK* disponibile alla MSF stessa. La macchina possiede le seguenti uscite: *DAT* (data), *STR* (strobe), *ERR* (error), *EOT* (end of transmission). È previsto anche un ingresso sincrono di *RST* (restart). La sequenza *LIN* è interpretata secondo le regole:

- un livello basso costante significa assenza di trasmissione.
- due livelli alti consecutivi indicano la presenza di un bit di dato nella terza posizione, che deve essere copiato sull'uscita *DAT*, contemporaneamente all'attivazione del segnale di *STR* (non è richiesto che vengano mantenuti validi per più di un ciclo di *CK*).
- dopo il bit di dato deve essere sempre presente un livello basso: in sua mancanza la MSF attiva *ERR*, aspettando un *RST* da parte dell'utente. Tale *RST*, attivo alto, sarà considerato solo in assenza di trasmissione.
- un livello alto isolato indica la fine delle trasmissioni, che la MSF segnala attivando *EOT*, mantenuta attiva fino a ricezione di una nuova sequenza.



Esercizio 5

Si progetti, usando il metodo ASM, una MSF sincrona avente un ingresso *X*, sincrono con un segnale di clock *CK*, e tre uscite *S0*, *S1* e *WT* (wait).



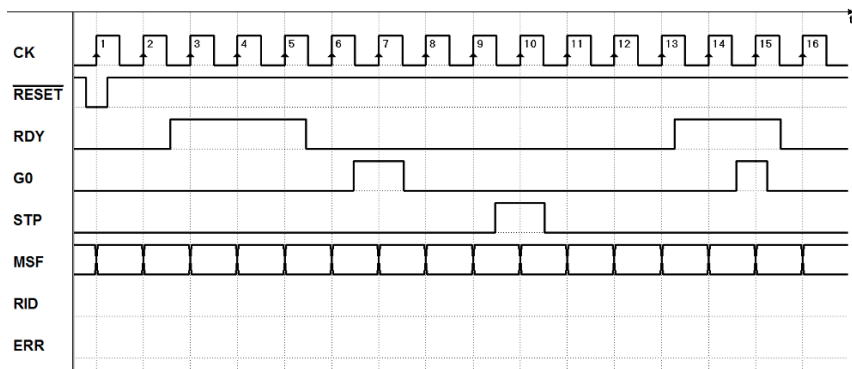
L'ingresso X riceve sequenze di 3 bit, caratterizzate dal primo sempre a 1. Tra una sequenza e l'altra sono presenti, sempre, almeno 3 bit a 0. Le due uscite $S0$, $S1$ dovranno indicare, in un codice a scelta, quale sequenza è stata ricevuta per ultima, mantenendo tale codice fino alla ricezione del primo bit di una nuova sequenza. L'uscita WT dovrà invece indicare i periodi di non validità delle uscite $S0$ e $S1$.

Esercizio 6

Si progetti, usando il metodo ASM, una MSF sincrona che rispetti le seguenti specifiche:

1. Possiede tre ingressi provenienti da tre pulsanti RDY (ready), GO e STP (stop); due uscite RID (ride), ERR (error). Gli ingressi sono sincroni.
2. L'uscita RID viene attivata alla ricezione della corretta sequenza dei segnali di ingresso, che si verifica quando viene premuto e poi rilasciato il pulsante di RDY , e quindi premuto il pulsante di GO . L'uscita RID deve attivarsi subito, non appena GO viene attivato, senza attendere il fronte attivo del clock; sarà disattivata successivamente, alla pressione del pulsante di STP .
3. L'uscita ERR viene attivata, invece, nel caso in cui il pulsante di GO sia premuto prima o insieme a quello di RDY . Anche l'uscita ERR sarà disattivata dalla pressione del pulsante di $STOP$.

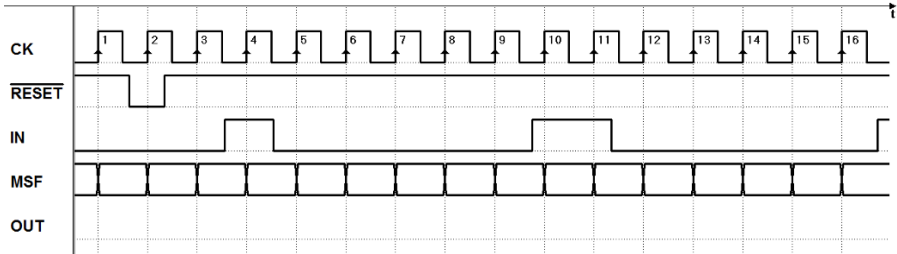
In ogni caso, ciascuna uscita deve permanere attiva almeno due interi cicli di clock, indipendentemente dalla attivazione del pulsante di STP .



Esercizio 7

Si progetti, usando il metodo ASM, una MSF sincrona dotata di un ingresso sincrono IN ed un'uscita OUT . L'ingresso, normalmente a zero, può essere ad uno per intervalli di uno, due o tre periodi consecutivi del clock CK .

Si vuole che l'uscita assuma il valore uno e lo mantenga per tre periodi di clock se IN è rimasto ad uno per un periodo, per due periodi se IN è rimasto ad uno per due periodi, per un periodo se IN è rimasto ad uno per tre periodi. Si faccia l'ipotesi che, dopo la disattivazione dell'uscita, l'ingresso IN rimanga inattivo ancora per alcuni cicli di clock.



Esercizio 8

Si progettino, usando il metodo ASM, una MSF sincrona dotata di:

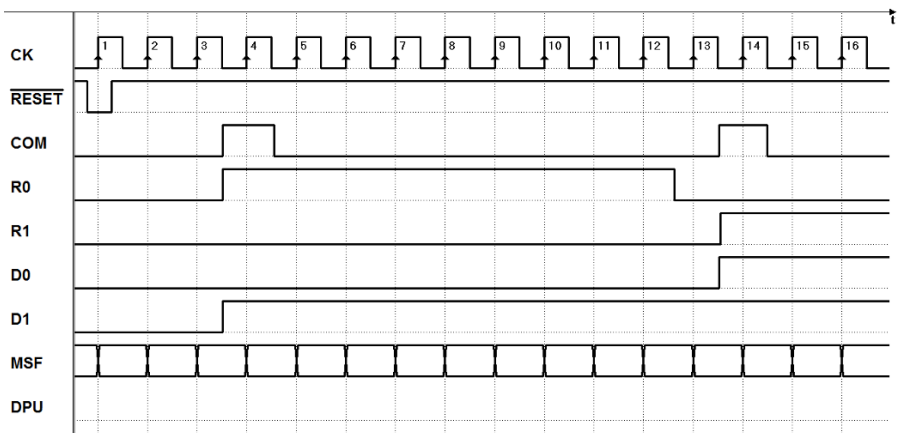
1. un ingresso sincrono COM ;
2. due gruppi di due ingressi $R1, R0$ e $D1, D0$;
3. un'uscita DPU

Il dispositivo deve generare un impulso aperiodico DPU , la cui durata e ritardo rispetto al segnale di comando siano separatamente controllabili, a passi di periodo T_{ck} del clock CK .

Ad ogni fronte di salita di COM , il sistema attende un tempo $r \cdot T_{ck}$ e quindi genera un impulso di durata $d \cdot T_{ck}$.

Gli ingressi $R1, R0$ definiscono l'entità r del ritardo ($0, 1$ e $2 \cdot T_{ck}$) e $D1, D0$ la durata d dell'impulso ($1, 2$ e $3 \cdot T_{ck}$): il ritardo può quindi essere nullo e l'impulso deve durare sempre almeno un ciclo di clock.

Il sistema genera una nuova sequenza di uscita solo dopo che la precedente è terminata.

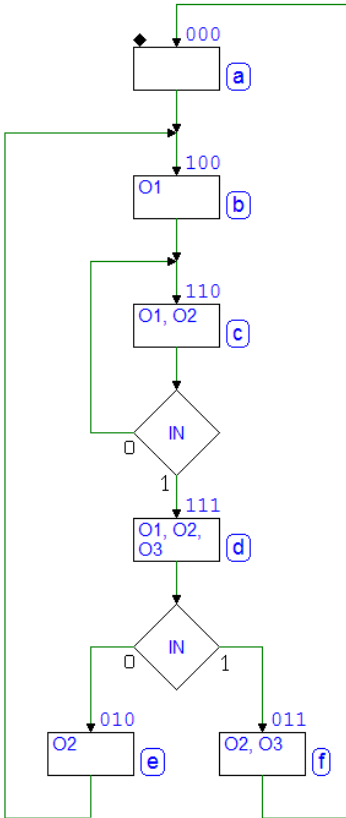


7.7 Soluzioni

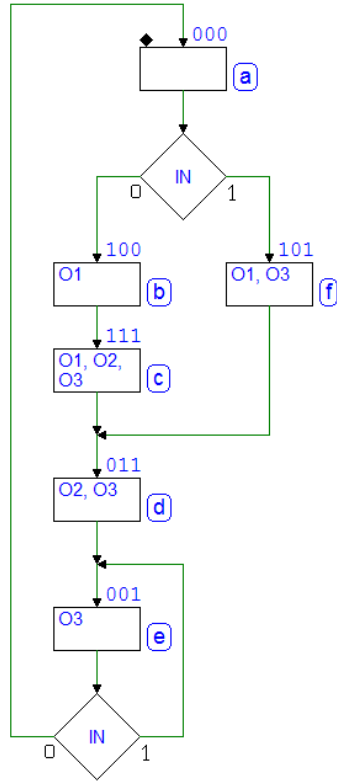
7.7.1 Analisi di reti sequenziali in termini di MSF

I file delle MSF qui rappresentate si possono scaricare Dal sito del simulatore *Deeds*, nelle pagine dei *contenuti digitali* del libro.

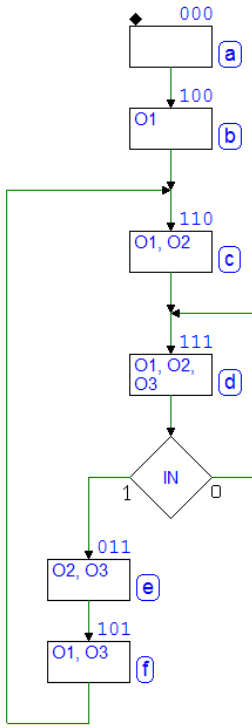
Rete 1:



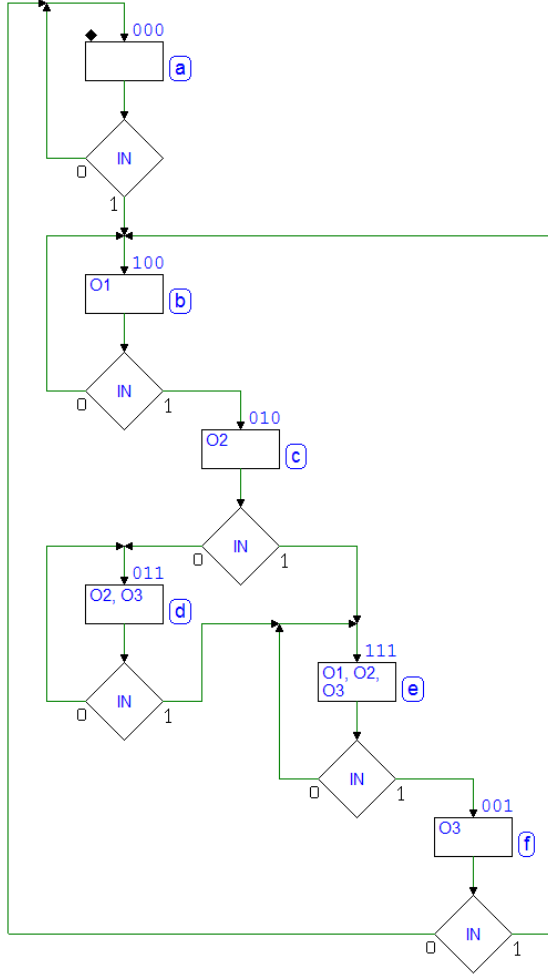
Rete 2:



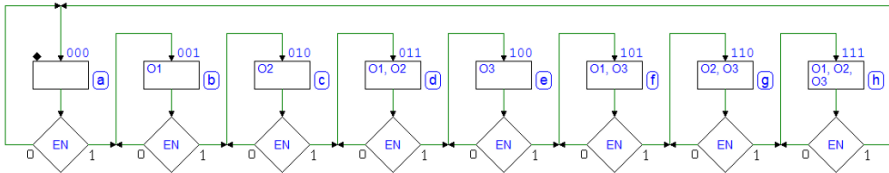
Rete 3:



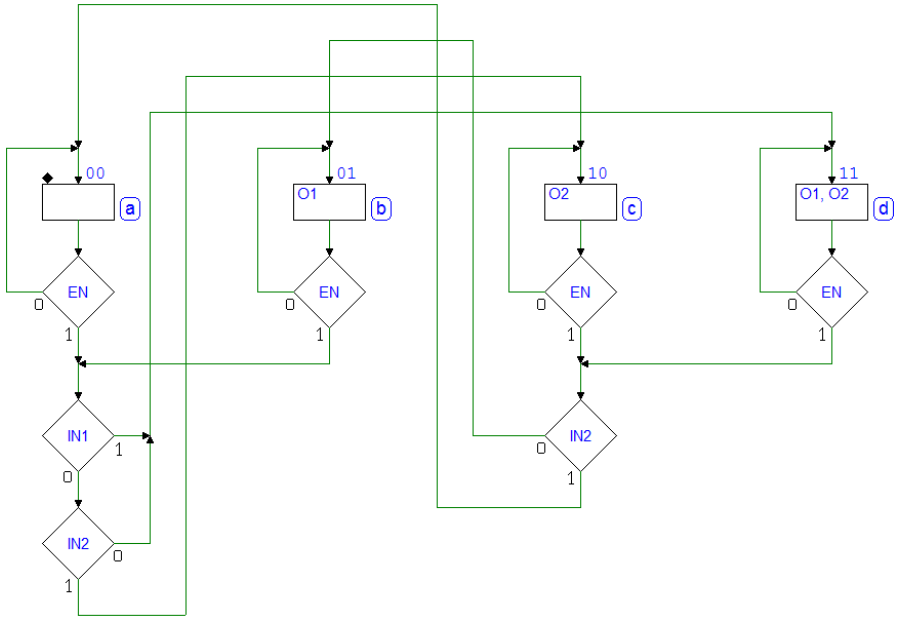
Rete 4:



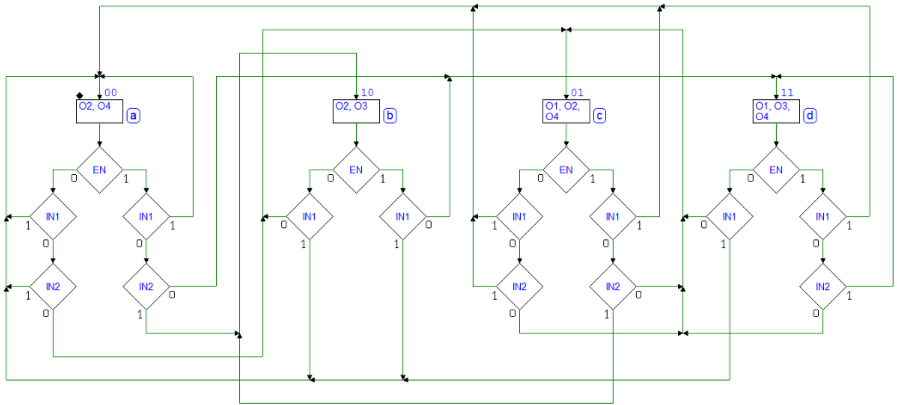
Rete 5:



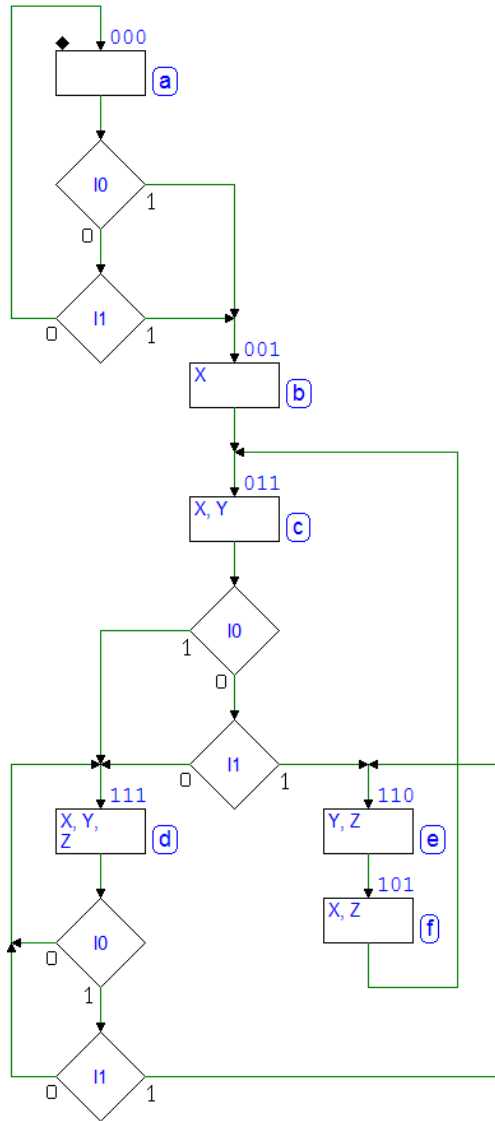
Rete 6:



Rete 7:



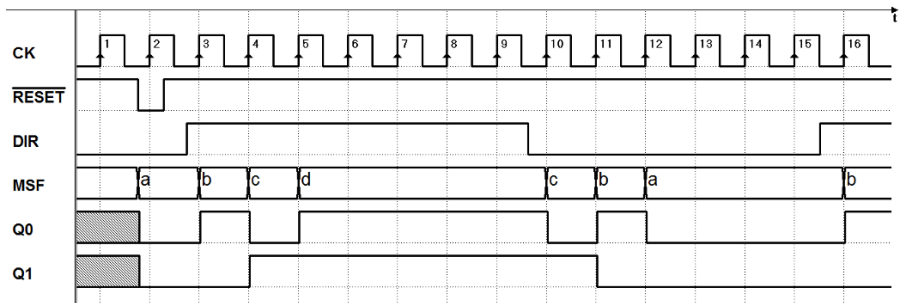
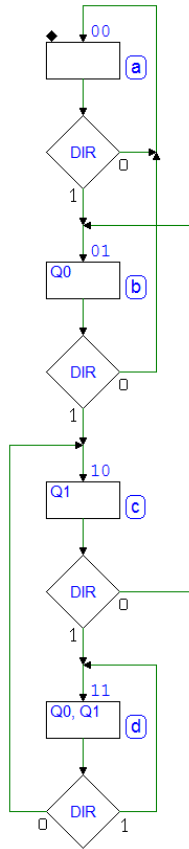
Rete 8:



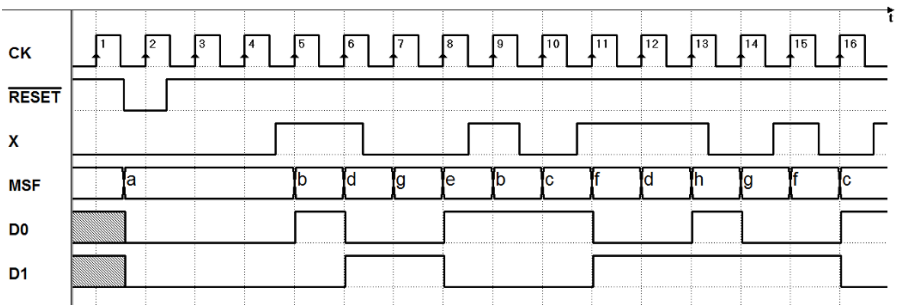
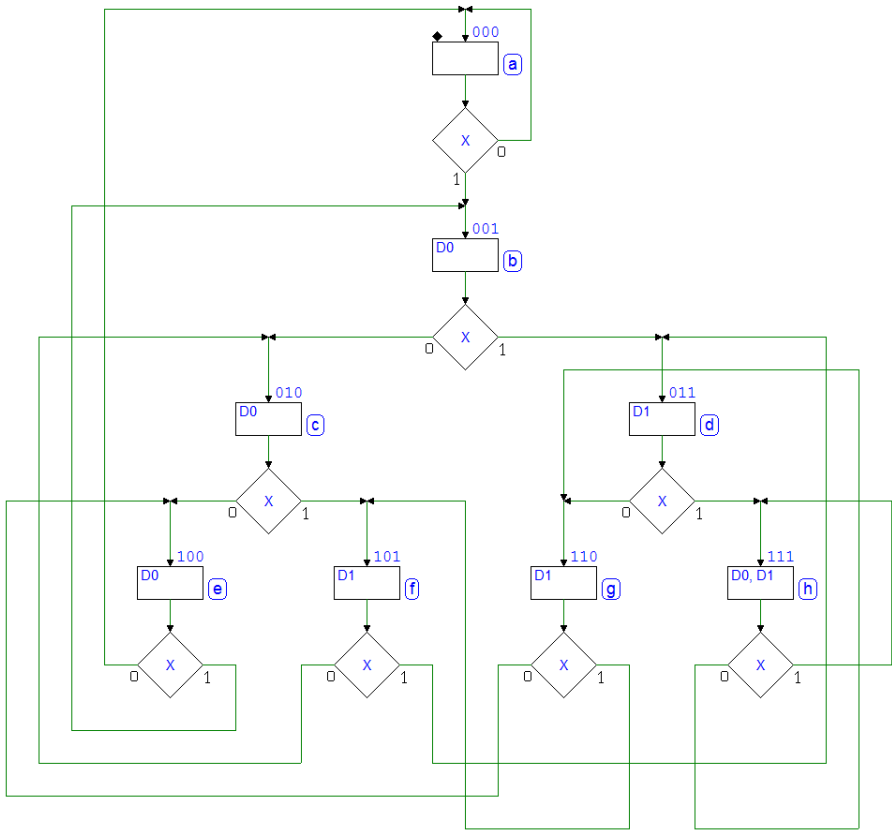
7.7.2 Progetto di MSF a partire da specifiche testuali

Le soluzioni delle MSF sono riportate anche sul sito del simulatore *Deeds*; per ognuna di esse è scaricabile anche un file di circuito con il quale è possibile verificare il comportamento della rete mediante simulazione temporale.

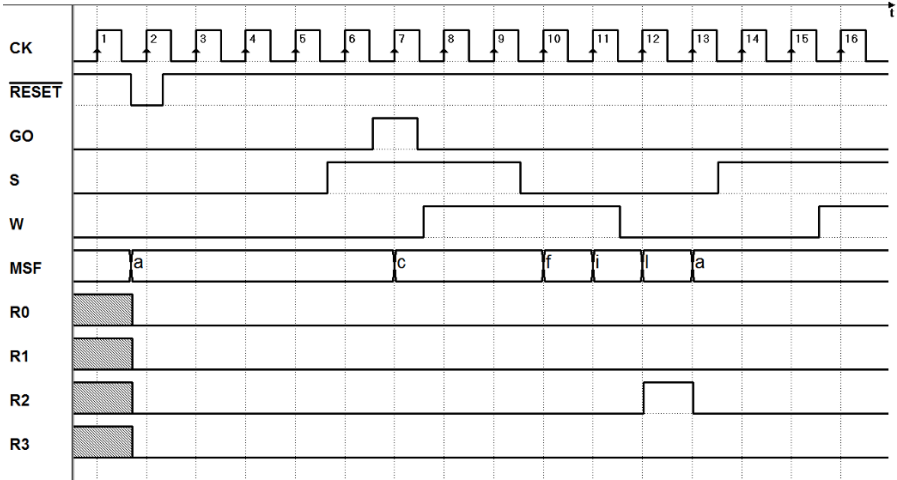
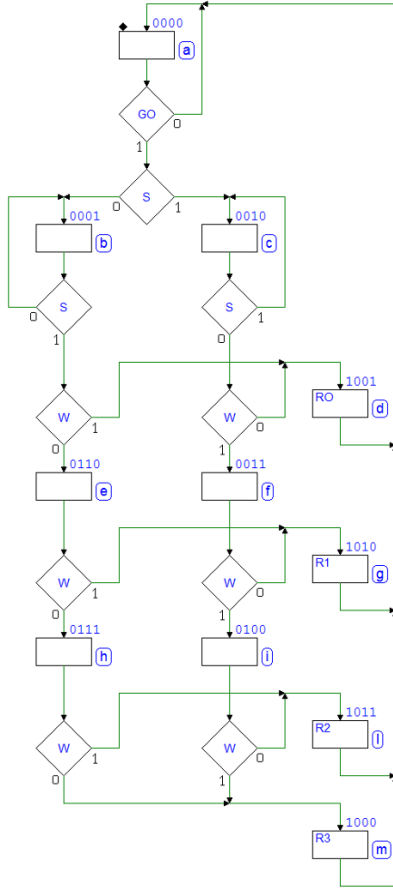
Soluzione esercizio 1:



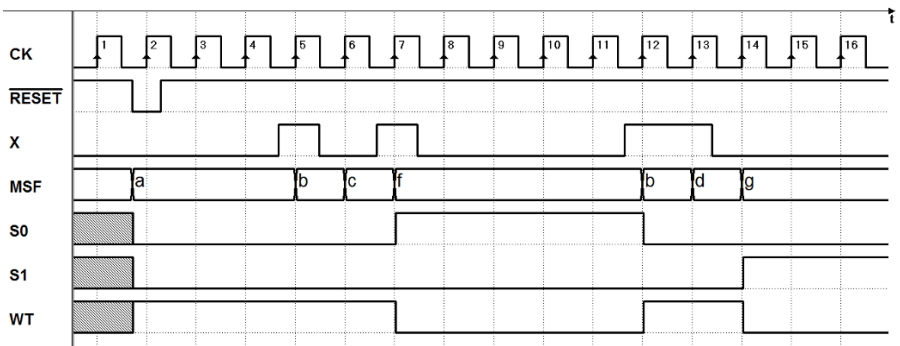
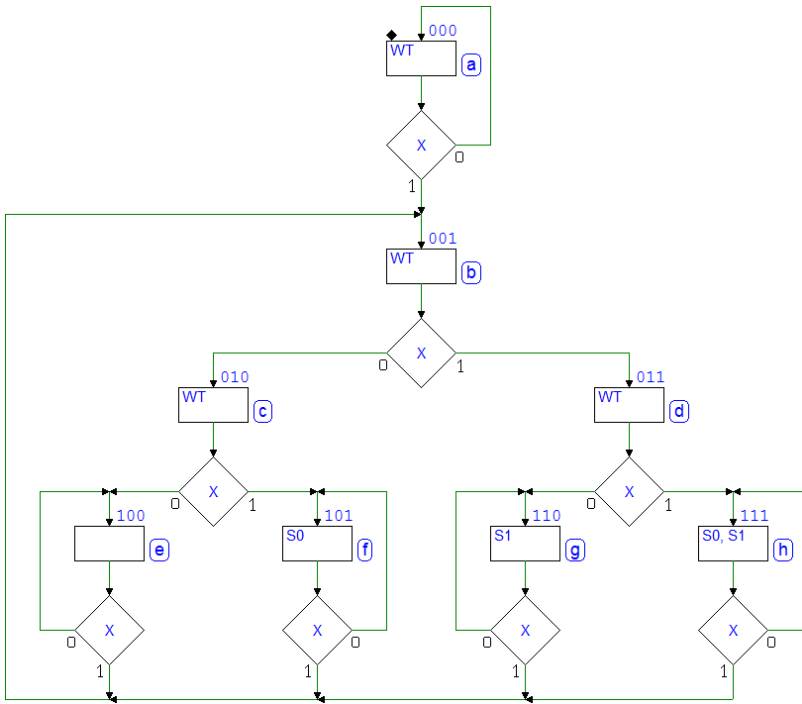
Soluzione esercizio 2:



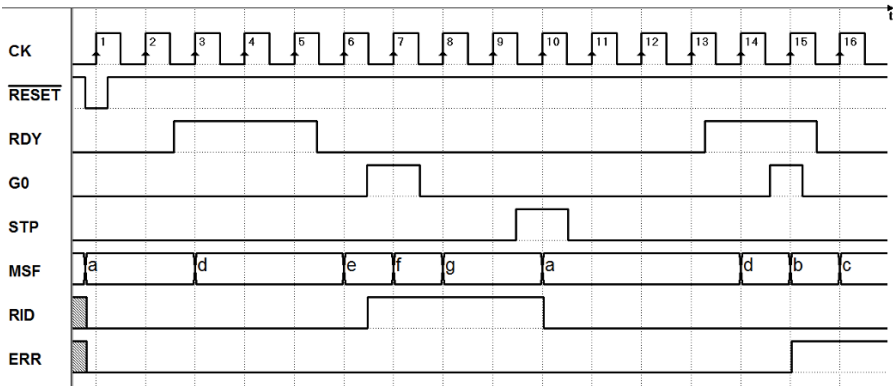
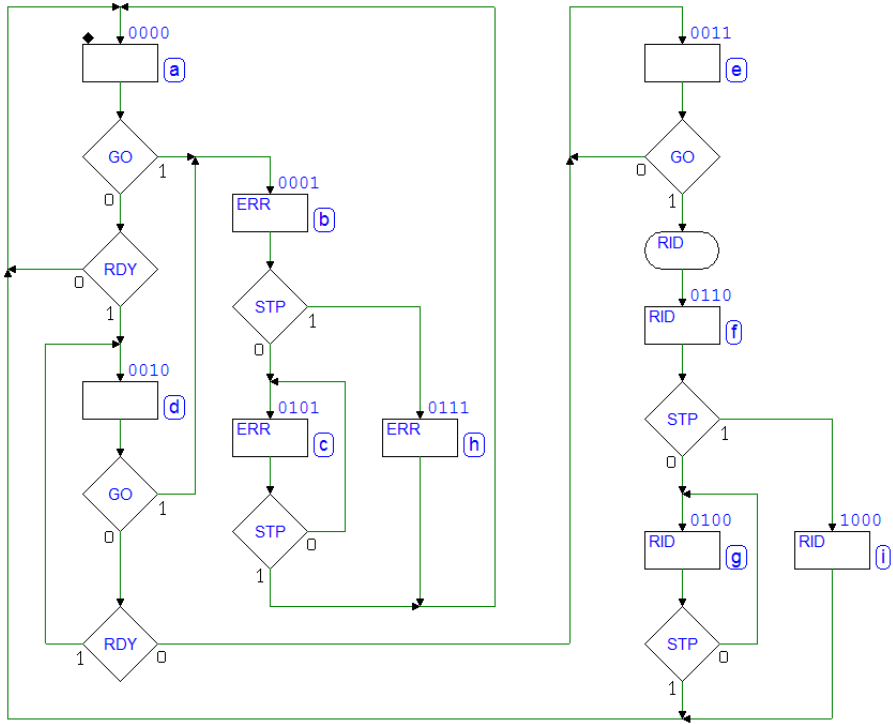
Soluzione esercizio 3:



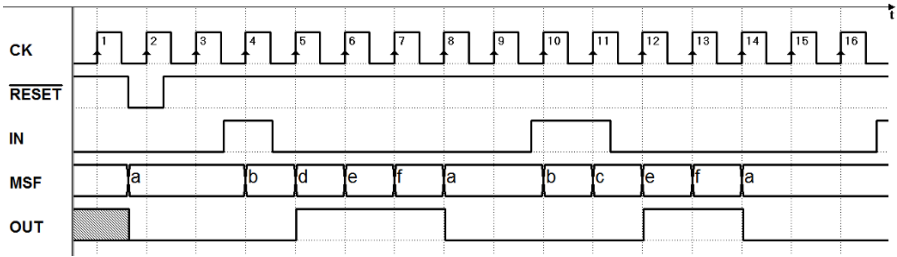
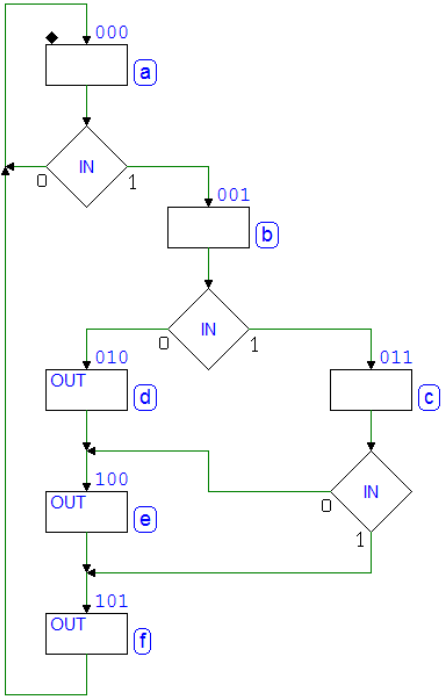
Soluzione esercizio 5:



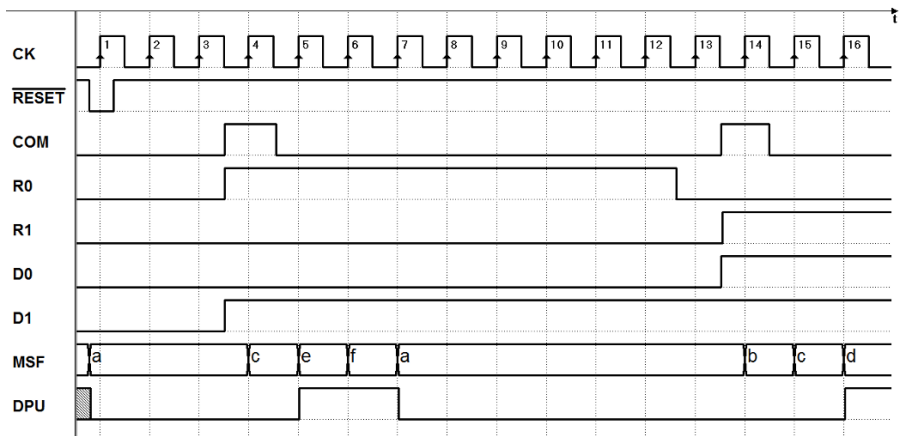
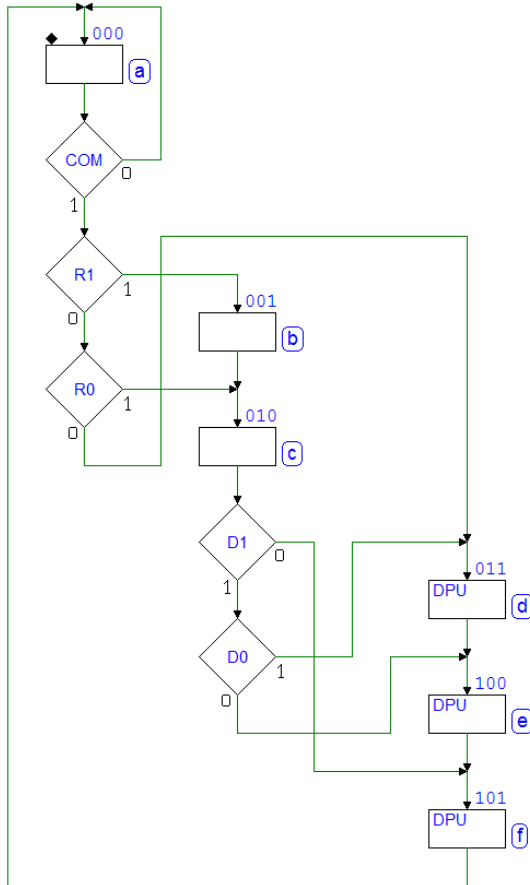
Soluzione esercizio 6:



Soluzione esercizio 7:



Soluzione esercizio 8:



La Macchina a Stati Finiti come controllore di sistema

Presentiamo qui una raccolta di esempi di progetto di *sistemi digitali sincroni* impieganti una MSF come controllore del sistema.

8.1 I sistemi digitali

Si è visto nei capitoli precedenti che la MSF è in grado di modellare e di progettare svariati algoritmi. La rappresentazione di un sistema come MSF lo descrive in termini di evoluzione degli stati e delle uscite in funzione degli ingressi. Da un punto di vista teorico, la MSF può rappresentare qualunque sistema discreto che evolve nel tempo passando da uno stato ad un altro ed ha un numero finito (di qui il nome) di ingressi, uscite e stati. Tuttavia la descrizione ed il progetto di un sistema in termini di “stati”, benché molto generale e versatile possono diventare troppo complessi o impraticabili quando il numero di stati diventa molto grande.

Come si è visto nel caso del registro a scorrimento, nei sistemi che gestiscono dati il numero degli stati è dipendente da tutte le possibili configurazioni che i dati possono assumere. Ad esempio, un registro a scorrimento da 8 bit sarà descritto da una MSF (nel nostro caso un *diagramma ASM*) formato da almeno 256 stati. Se si volesse descrivere in termini di stati un microprocessore, basta osservare che un solo registro da 32 bit (un microprocessore può contenerne molti) richiederebbe 2^{32} stati.

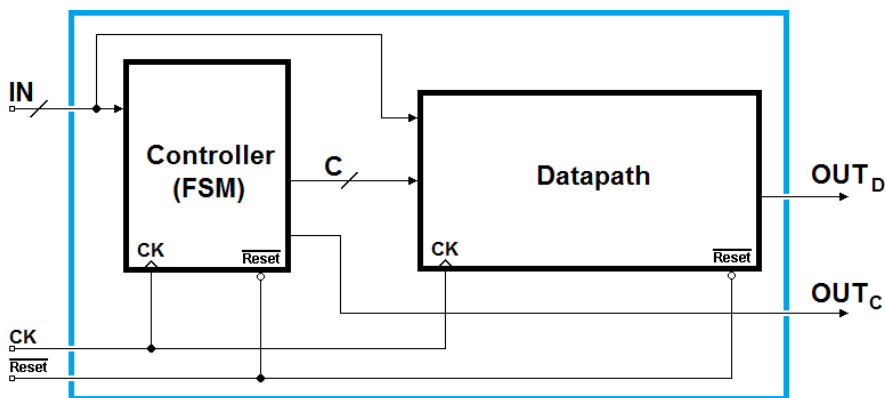
D'altra parte esistono strutture digitali regolari per la gestione dei dati, sia combinatorie, (ad esempio selettori, deselettori, codificatori, decodificatori, reti aritmetiche) che sequenziali (flip-flop, registri, contatori, memorie e molti altri). Benché la MSF sia in grado di effettuare operazioni aritmetiche e logiche, tali operazioni possono essere eseguite in un modo più efficace usando dispositivi aritmetici dedicati.

In linea di massima si può affermare che un sistema digitale si ottimizza dividendo le sue mansioni tra un *controllore* e dei *componenti dedicati*.

Il *controllore* (*controller*), che progetteremo in termini di MSF, ha il compito di gestire il funzionamento complessivo del sistema, mentre i *componenti dedicati* ne definiscono la *architettura* (*architecture*), che interagisce direttamente con i dati. Il controllore del sistema è alle volte denominato *sequenziatore* (*sequencer*) o anche *temporizzatore* (*timing generator*); l'architettura è sovente denominata *percorso dei dati* (*datapath*). Nel seguito useremo i termini *controllore* e *datapath*.

8.2 Sistemi a controllo aperto

Il caso più semplice di sistema è quello in cui il *controllore* fornisce comandi al *datapath* senza ricevere da quest'ultimo *nessuna retroazione* (*feedback*), come si vede nella figura seguente. Le uscite del sistema possono essere generate sia dal *controllore* che dal *datapath*. Anche gli ingressi possono interessare entrambi i costituenti il sistema:



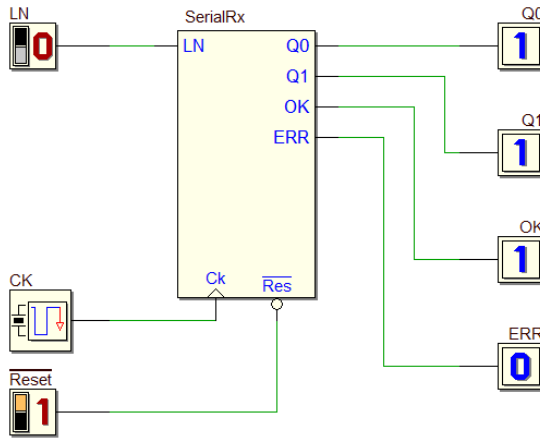
Nel prossimo paragrafo esaminiamo un esempio di questa struttura: un *ricevitore seriale* a due bit, già stato visto in precedenza in una versione impiegante solo la MSF.

L'uso del *datapath* permetterà di semplificare la MSF e di rendere la struttura facilmente *scalabile*. Si pensi infatti che un sistema basato solo su MSF avrebbe bisogno, per gestire un segnale seriale contenente *8 bit di dato*, di almeno *256 stati*, mentre con l'aggiunta di un *datapath*, composto da tanti flip-flop quanti sono i bit seriali, come vedremo, la MSF aumenterà solo moderatamente la sua complessità. In sintesi si può dire che con il secondo approccio la *complessità progettuale* aumenta *linearmente* col numero di bit da gestire mentre nel primo aumenta in modo *esponenziale*.

Negli esempi successivi, poi, mostreremo come, con un adeguato *datapath*, sarà possibile progettare un controllore la cui complessità è *indipendente* dal numero di bit seriali.

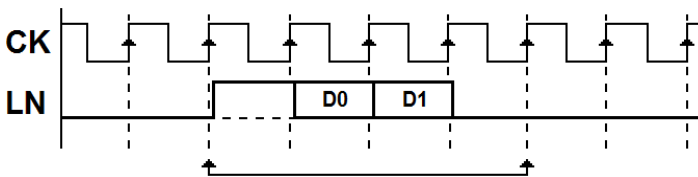
8.2.1 Ricevitore seriale (2 bit)

Riassumiamo in breve le specifiche: si tratta di progettare un *ricevitore seriale sincrono* a 2 bit. Il dispositivo riceve le sequenze seriali sulla linea *LN*, e genera le uscite *Q0*, *Q1*, *OK* ed *ERR*, come si osserva nella figura seguente, dove richiamiamo la precedente realizzazione impiegante la sola MSF (ingressi e uscite della rete appaiono come componenti attivi, come impostati nel simulatore *Deeds*, durante la simulazione):



Il formato delle sequenze ricevute è riassunto qui a parole e visibile nel diagramma temporale qui sotto:

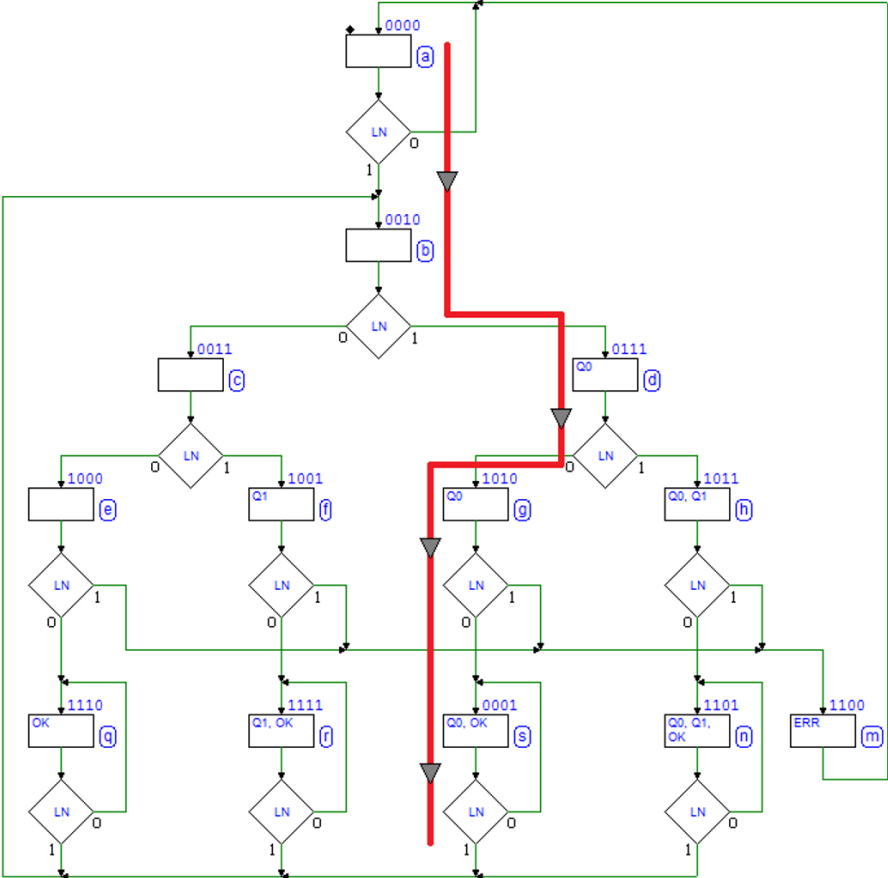
- il tempo di bit è pari al ciclo del clock *CK*;
- le sequenze iniziano con un *bit di start* a 1;
- proseguono con i *due* bit di dato *D0* e *D1* (nell'ordine);
- terminano con un *bit di stop* a 0.



Si osservi che il segnale *LN* è sincrono con il clock *CK*, e per questo motivo lo rappresentiamo con un ritardo di propagazione, indicativo, rispetto al fronte di salita. Il sistema esegue le seguenti operazioni:

- alla ricezione di una sequenza, rende disponibili i due bit di dato *D0* e *D1* sulle uscite *Q0* e *Q1*;
- controlla che il *bit di stop* sia correttamente a 0 e, in tal caso, attiva l'uscita *OK*, e la mantiene attiva fino all'arrivo della prossima sequenza; invece, se è a 1, il ricevitore attiva *solo* l'uscita *ERR* (Errore), per la durata di un ciclo di clock, e si mette in attesa di una nuova sequenza.

In figura si osserva l'ASM della precedente versione, basata sulla sola MSF:

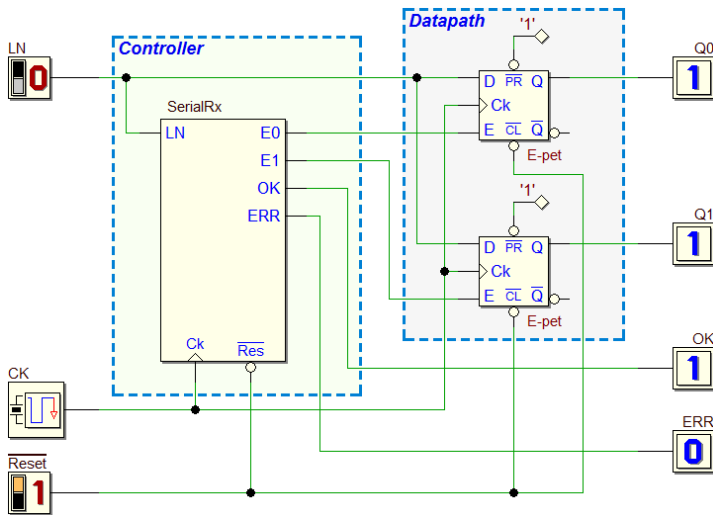


Il percorso evidenziato ci ricorda che la MSF, per ricordare quali bit ha ricevuto, deve separare i percorsi, dal momento che non possiamo memorizzare i dati in altro modo. Nell'esempio, ha ricevuto una sequenza con $D0 = 1$ e $D1 = 0$, seguita da un regolare bit di stop; inoltre, per mantenere attive le uscite (qui $Q0 = 1$ e $Q1 = 0$) fino alla ricezione della prossima sequenza, necessita di uno stato di attesa.

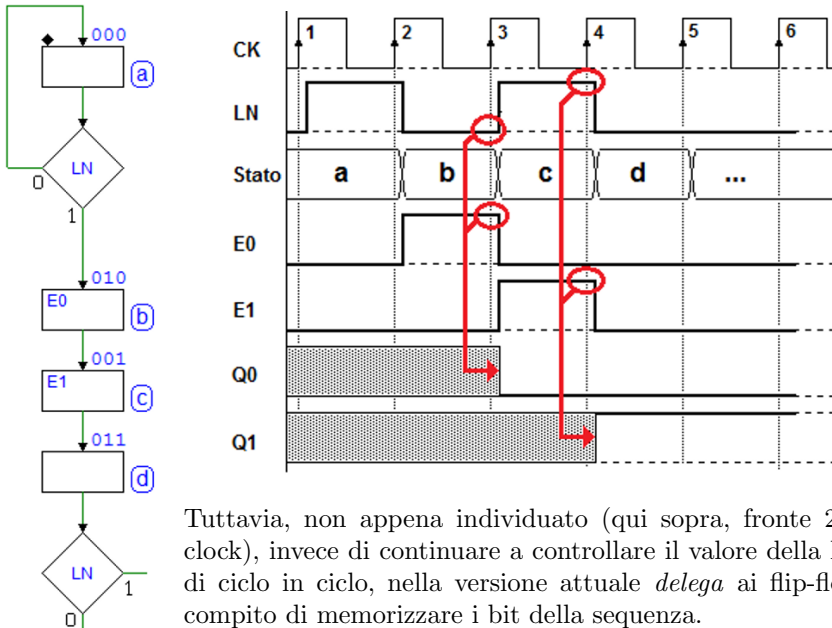
Questa soluzione richiede un numero di stati che dipende da tutte le possibili configurazioni di dato. E' evidente che l'estensione ad un numero più elevato di bit ne aumenta la complessità in progressione geometrica (13 stati nel caso in esame che aumentano a 25 se il segnale seriale contiene tre bit e a 49 con quattro bit; 97 con 5 bit, ecc.).

Risulta quindi poco pratico l'utilizzo di questa struttura del progetto a casi di interesse reale (il tipico segnale seriale contiene otto bit).

La situazione cambia radicalmente introducendo un *datapath* in grado di memorizzare i dati, costituito in questo caso da *due flip-flop* di tipo E-PET:



Si noti che l'ingresso di inizializzazione \overline{Reset} che agisce non solo sul *controllore*, forzando la MSF nello stato predisposto, ma anche sulle componenti sequenziali del *datapath*. Troveremo questo tipo di collegamento in tutti i progetti successivi, salvo casi particolari che saranno messi in evidenza. Iniziamo a tracciare il diagramma ASM (vedi figura sotto). La MSF attende il *bit di start* con un ciclo di attesa sullo stato (a), come nella soluzione precedente:

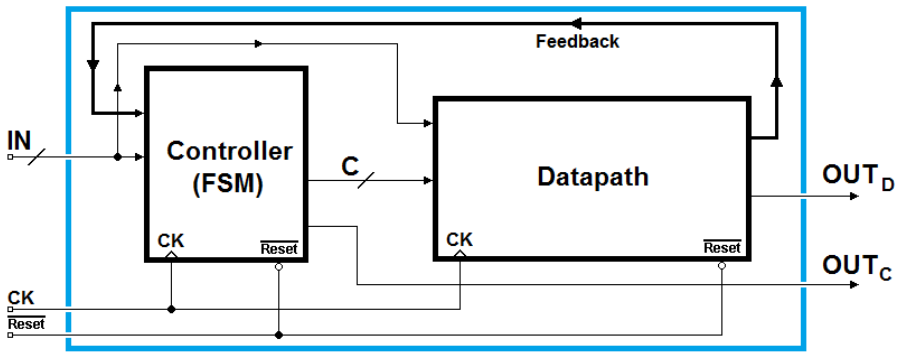


Tuttavia, non appena individuato (qui sopra, fronte 2 del clock), invece di continuare a controllare il valore della linea di ciclo in ciclo, nella versione attuale *delega* ai flip-flop il compito di memorizzare i bit della sequenza.

Se volessimo estendere il ricevitore ad un numero maggiore di bit, sarebbe sufficiente introdurre nel diagramma ASM altri stati come (b) e (c) e, ovviamente, modificare il *datapath* con l'aggiunta dei flip-flop necessari.

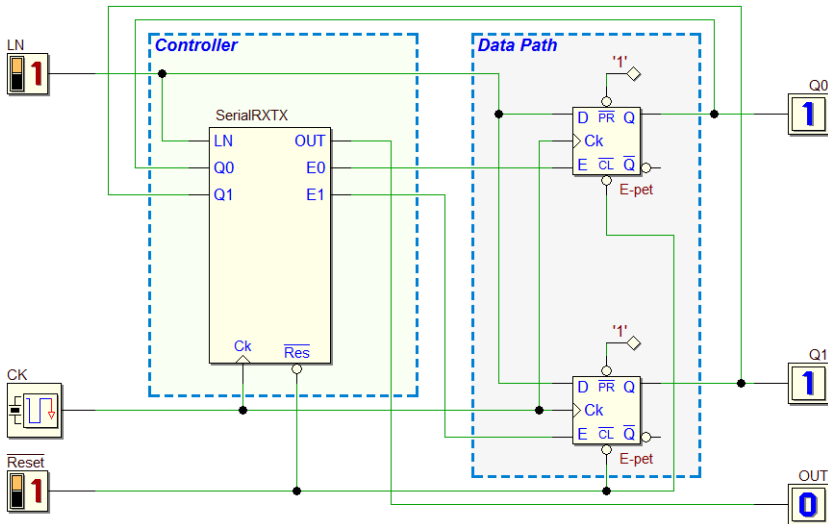
8.3 Sistemi a controllo retroazionato

In generale, la MSF ha necessità di avere informazioni di ritorno dal *datapath*, sia per verificarne il funzionamento che per utilizzarle nel suo algoritmo. Il collegamento che riporta tali informazioni come ingressi della MSF, detto *retroazione (feedback)* aumenta di molto le possibilità del sistema:



8.3.1 Ricevitore e trasmettitore seriale (2 bit)

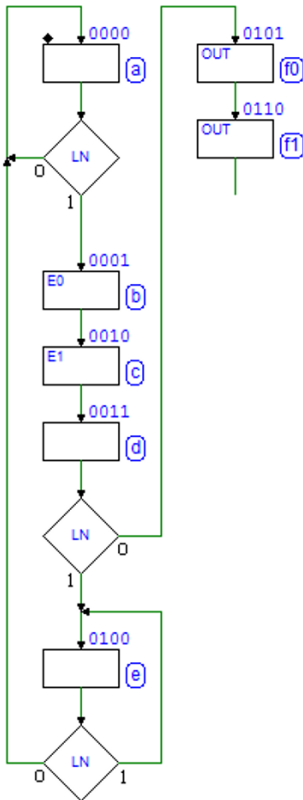
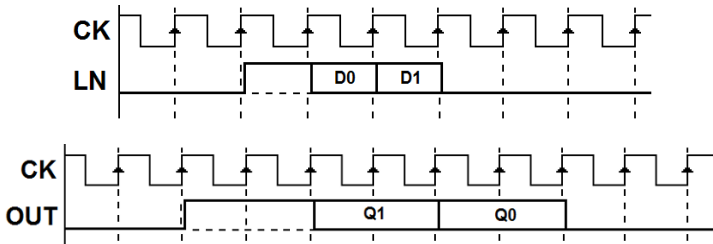
Il progetto qui presentato introduce un esempio di sistema in cui la MSF acquisisce come ingressi dei segnali generati dal *datapath* e li impiega per ritrasmettere il dato seriale in un differente formato:



Lo schema del sistema è a prima vista simile a quello dell'esempio precedente, in quanto utilizza due flip-flop tipo E-PET per immagazzinare i dati seriali ricevuti attraverso la linea *LN*.

Compare però un'uscita *OUT* che ha il compito di ritrasmettere in un diverso formato il segnale seriale. Il collegamento tra le uscite *Q0* e *Q1* dei flip-flop e gli ingressi con lo stesso nome della MSF permette a quest'ultima di conoscere i valori dei bit ricevuti, necessari per la ritrasmissione.

In condizioni di riposo, le linee *LN* e *OUT* si trovano al valore logico 0. Il dispositivo attende l'arrivo, sulla linea *LN*, di un pacchetto nel formato già incontrato (*bit di start* → *D0* → *D1* → *bit di stop*):



Il dispositivo trasmette su *OUT* una sequenza simile (*bit di start* → *Q1* → *Q0* → *bit di stop*), al termine della ricezione, ma con un tempo di bit doppio (due cicli di clock) rispetto alla sequenza ricevuta. Questa operazione dimezza il numero di bit trasmessi in un secondo (*“bit rate”*), come visibile nel tracciato qui sopra.

La trasmissione su *OUT* deve avvenire solo se la sequenza ricevuta su *LN* è corretta (il *bit di stop* deve essere a 0). In caso contrario il dispositivo non deve generare alcuna uscita su *OUT*, ma *attendere* che *LN* ritorni a zero prima di ritornare in attesa di una nuova sequenza.

Tracciamo la prima parte del diagramma ASM (vedi figura a lato). Il compito del ricevitore è quello di salvare i dati nei flip-flop, e lo abbiamo già incontrato nell'esercizio precedente.

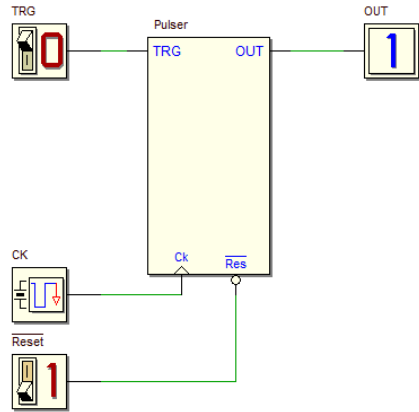
Si noti lo stato (e) di attesa che la linea ritorni a 0, nel caso che il *bit di stop* sia errato. Se questo è corretto, invece, la MSF procede alla generazione del segnale seriale sull'uscita *OUT*, e i due stati consecutivi (f0) e (f1) con uscita attiva fanno sì che su *OUT* sia trasmesso un *bit di start* di durata pari a due cicli di clock.

8.3.2 Generatore di impulso

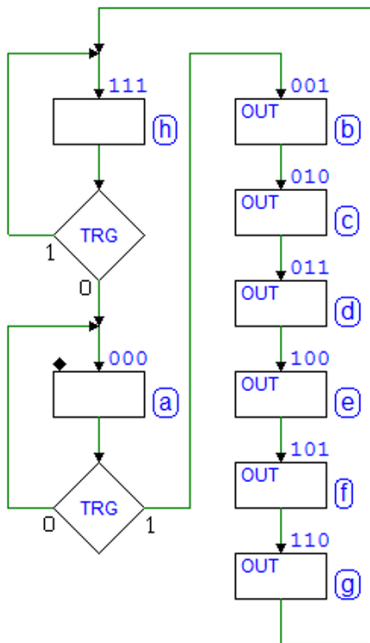
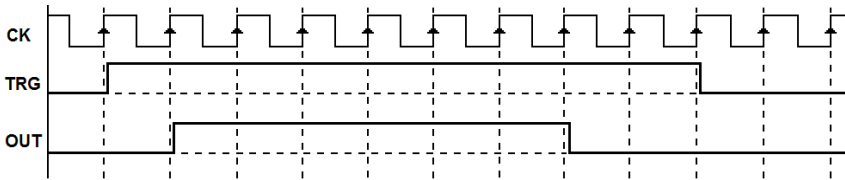
Questo progetto introduce un altro esempio di sistema realizzato in due modi: con la sola MSF e con la struttura *controllore - datapath*.

Il “generatore di impulso”, all’occorrenza di un *fronte di salita* sull’ingresso *TRG*, produce sull’uscita *OUT* un segnale alto (un impulso) di durata multipla del ciclo di clock.

In questa prima versione utilizziamo solo la MSF (figura a lato).



La durata dell’impulso è fissa e di pochi cicli di clock (solo 6), come visibile nel tracciato temporale qui sotto, in cui si vede anche un esempio di attivazione del comando *TRG*:



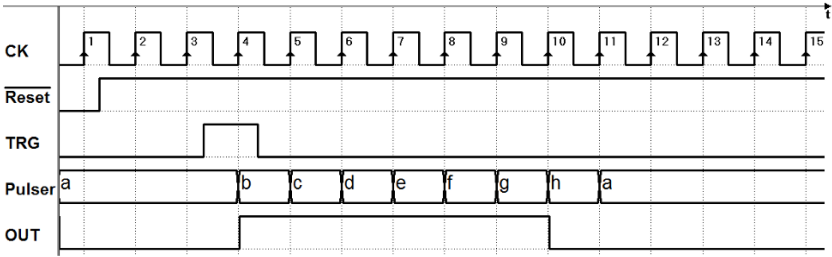
Il diagramma ASM non presenta particolari difficoltà: la MSF, in (a), lo *stato al Reset*, attende che *TRG* vada a 1.

Quando questo avviene, la MSF passa nello stato (b) e successivi, nei quali genera $OUT = 1$: la durata dell’impulso è pari al numero di stati con *OUT* attiva.

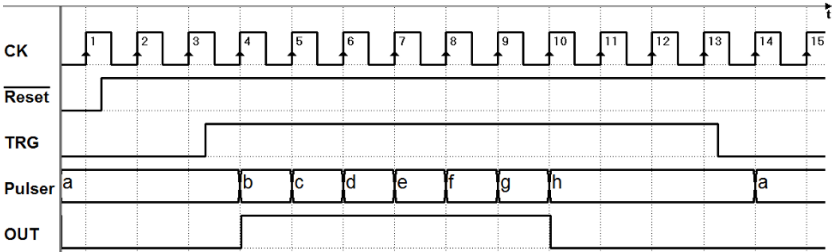
Terminata la generazione dell’impulso, nello stato (h), attendiamo che *TRG* sia tornato a 0.

Questa attesa in (h) è necessaria: se non ci fosse, e se *TRG* fosse ancora a 1 nel tornare nello stato (a), la MSF genererebbe un nuovo impulso subito, mentre questo deve avvenire, come da specifiche, all’arrivo del *fronte di salita* di *TRG*.

Infine, qui il risultato della simulazione temporale prodotta dal simulatore, nel caso di un ingresso *TRG* di breve durata:



E, quindi, nel caso di *TRG* di lunga durata, con l'attesa del suo ritorno a 0:



Passiamo alla seconda versione, basata sulla struttura *controllore - datapath*. Affianchiamo alla MSF un contatore (il "Cnt4", vedi riquadro qui sotto).

Il contatore "Cnt4" (dalla libreria del simulatore *Deeds*)

L'ingresso \overline{CL} , quando a 0, azzerava in modo *asincrono* le uscite $Q3..Q0$.

Se \overline{CL} non è attivo, gli altri ingressi, *sincroni*, controllano il contatore: *LD* (*Load*), *En* (*Enable*), *Et* (*EnableTc*) e U/\overline{D} (*Up/Down*).

L'ingresso *LD*, attivo alto, sul fronte di salita del clock *CK*, carica gli ingressi $P3..P0$ sulle uscite $Q3..Q0$.

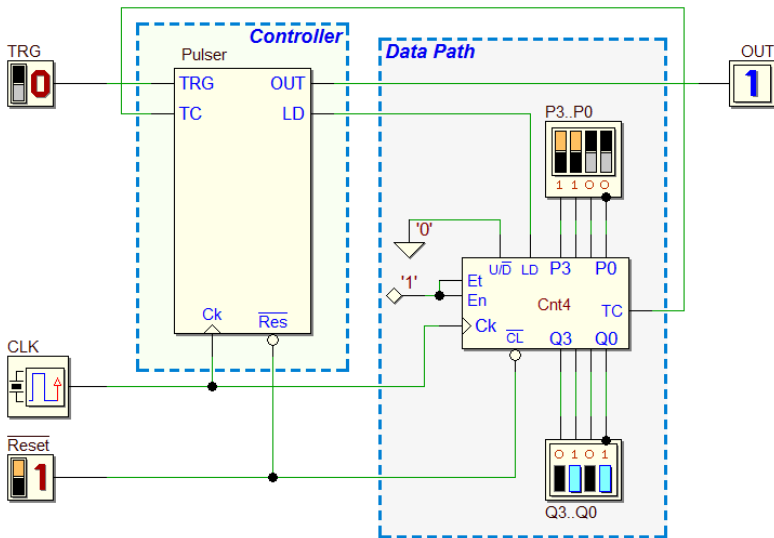
Quando *LD* non è attivo, gli ingressi *En* ed *Et*, se entrambi a 1, abilitano il conteggio sul fronte di salita di *CK*.

Il conteggio è in avanti (*Up*) se $U/\overline{D} = 1$, altrimenti è all'indietro (*Down*).

L'uscita *TC* (*Terminal Count*) si attiva ad 1 quando *Et* = 1 e le uscite del contatore raggiungono il valore 1111 nel conteggio in avanti, o il valore 0000 nel conteggio all'indietro. *TC* è sempre = 0 se *Et* = 0.

Si consiglia di approfondirne il comportamento del contatore “Cnt4”, prima di proseguire, simulando un circuito di prova (come quello suggerito nel riquadro), verificando le modalità operative del conteggio avanti/indietro, del caricamento parallelo e dell’azzeramento.

Come si vede nella figura qui sotto, il *datapath* è costituito da un *contatore*, che ha il compito di contare il numero di cicli di clock in cui *OUT* deve restare attiva. Grazie a questa estensione, riusciamo a realizzare specifiche più flessibili, come la possibilità di programmare la durata dell’impulso:

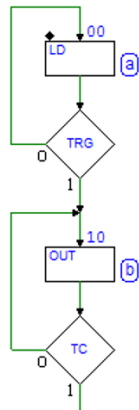


Il contatore è predisposto per un *conteggio all’indietro* ($U/\overline{D} = 0$); il conteggio è *sempre abilitato* ($EN = 1; ET = 1$). La MSF definisce l’inizio del conteggio mediante la linea *LD* del contatore e ne verifica il termine tramite lettura del *TC*. Nel diagramma ASM, attendiamo il comando *TRG*, nello stato (a).

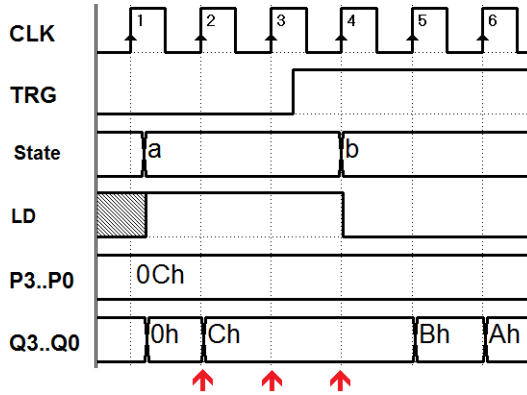
Nello stato di attesa (a) conviene attivare *LD*. In questo modo, mentre siamo in (a), obblighiamo il contatore a caricare, in modo sincrono, il numero impostato sugli interruttori (linee *P3..P0*).

Si noti che in questo modo manteniamo fermo il conteggio, con l’effetto positivo di ridurre i consumi di energia della rete (i circuiti usano una certa quantità di energia ogni volta che cambiano di livello).

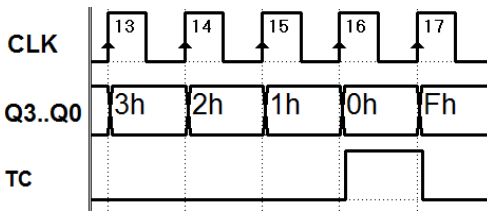
Dato che il comando *LD* è sincrono (agisce sul fronte di salita del clock), nella transizione tra gli stati (a) e (b) il contatore non conta ancora e continua ad essere caricato, dal momento che *LD* è attivo in (a).



Nella figura che segue, le frecce indicano i fronti del clock 2, 3 e 4. Questi sono gli istanti in cui il numero $P3..P0$ è caricato nel contatore, compreso il fronte 4, quando si ha il passaggio della MSF nello stato (b). Il risultato è che il conteggio inizierà effettivamente solo sul fronte successivo (il 5):



Lo stato (b) è ripetuto fino a quando il contatore segnala il termine del conteggio. Ad ogni fronte di clock il contatore decrementa il suo valore fino a quando arriva a zero, attivando TC , come nel tracciato temporale qui sotto.

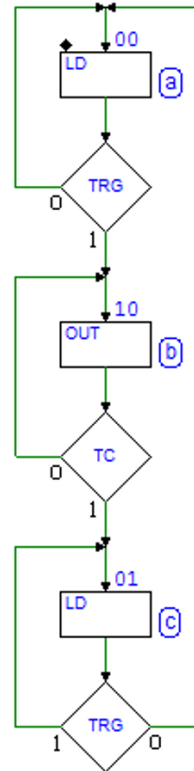


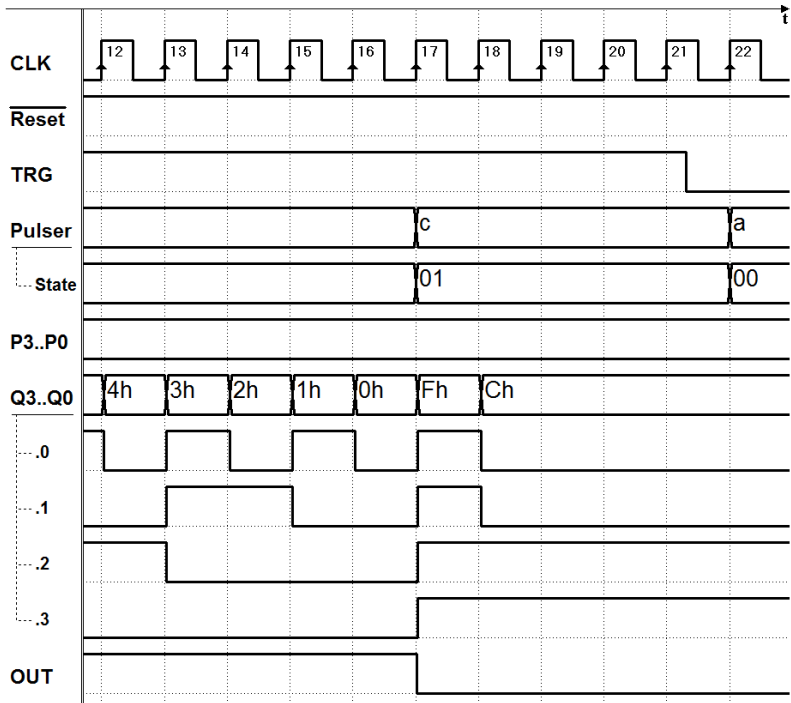
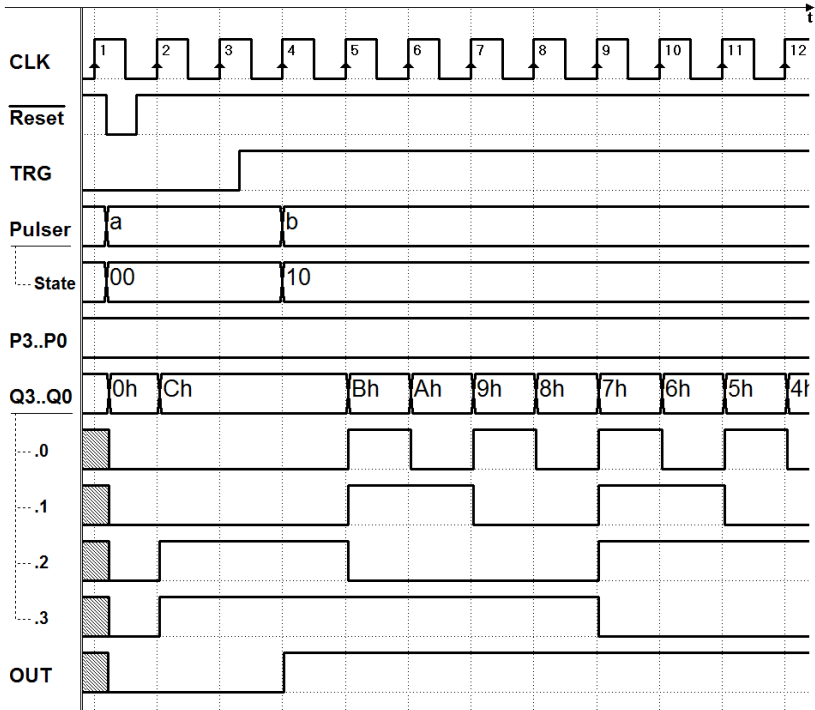
Di conseguenza, la MSF esce dallo stato (b). Si noti un particolare, che nel presente caso non crea problemi: il fronte 17 del clock consente alla MSF di uscire da (b) ma, dato che il contatore è abilitato, sullo stesso fronte il conteggio all'indietro prosegue (da 0000 passa a 1111).

Il disegno del diagramma ASM si conclude aggiungendo lo stato (c), dove l'uscita OUT non è più attiva e si resta in attesa che TRG torni a zero nel caso fosse ancora a 1, come esaminato nella versione precedente del generatore. Nello stato (c) attiviamo anche LD , per fermare il conteggio.

Infine, eseguiamo in *Deeds* una simulazione completa del sistema. L'analisi del suo comportamento (vedi il diagramma temporale nella pagina successiva) ci consentirà anche di rispondere a domande di dettaglio del tipo:

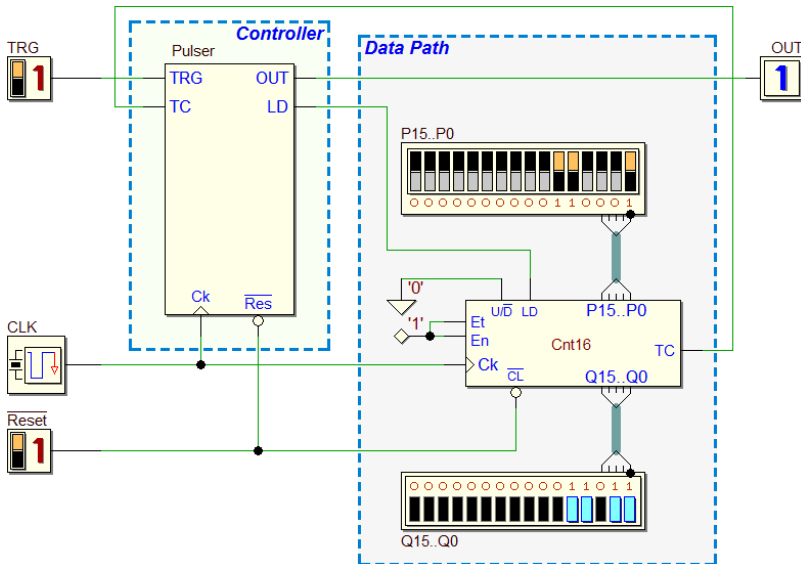
Quante volte abbiamo eseguito lo stato (b)? Quanti cicli di clock sono trascorsi dal caricamento del contatore?





Come si ricava dal diagramma, *OUT* dura 13 cicli di clock, a fronte di un 12 impostato sugli ingressi *P3..P0* del contatore (viene contato anche lo 0).

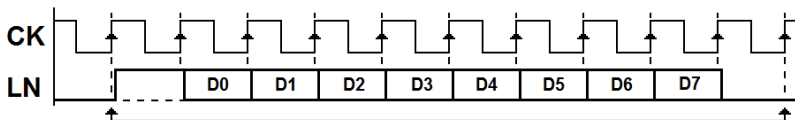
Infine, si noti che il sistema progettato può essere facilmente modificato per generare un impulso di durata più lunga, semplicemente impiegando un contatore con il numero adeguato di bit. Nel sistema mostrato nella figura qui sotto, è impiegato un contatore da 16 bit (il “Cnt16” di *Deeds*):



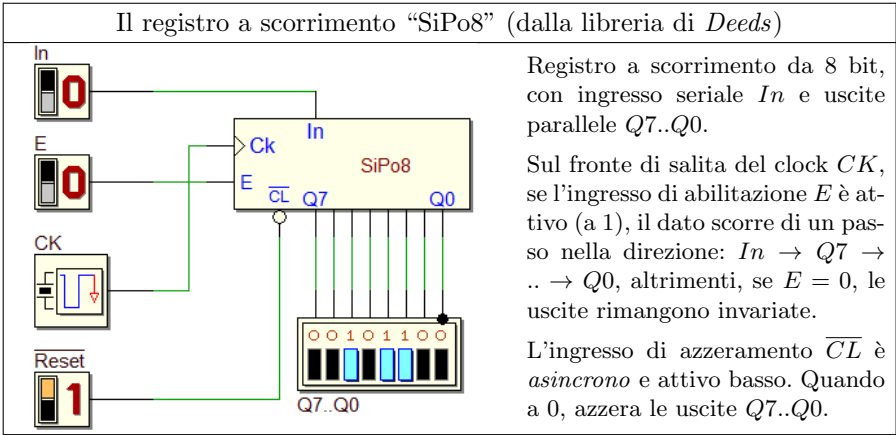
La MSF resta esattamente la stessa, ma il nuovo circuito è in grado di generare impulsi di durata fino a $2^{16} = 65536$ cicli di clock.

8.3.3 Ricevitore seriale (8 bit)

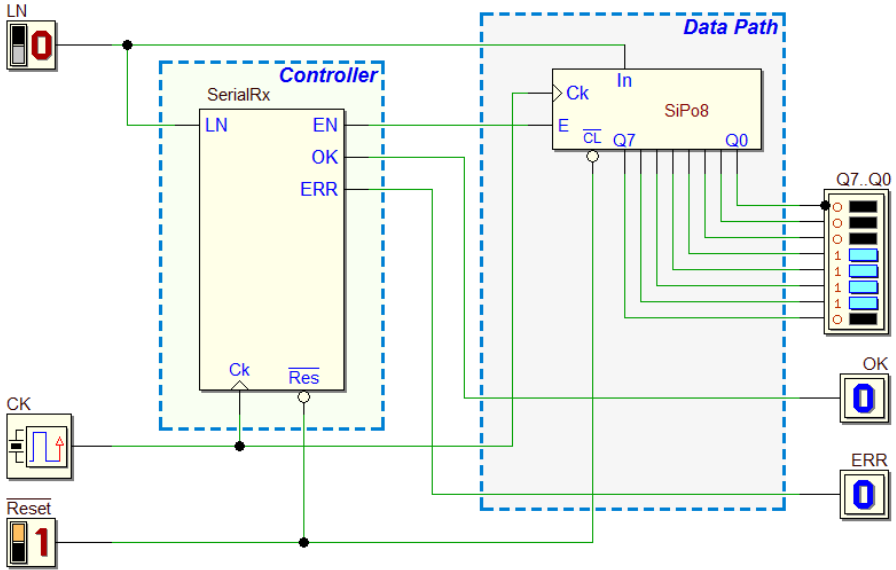
I due progetti qui presentati introducono due varianti del ricevitore seriale già visto: per memorizzare i bit, invece dei flip-flop, impiegano un *registro a scorrimento*. La sequenza seriale ha lo stesso protocollo di quello visto per il ricevitore a due bit di dato, salvo il fatto che ne contiene otto, da D0 a D7. Le specifiche rimangono le stesse a riguardo della attivazione di *OK* o *ERR* al termine della ricezione della sequenza.



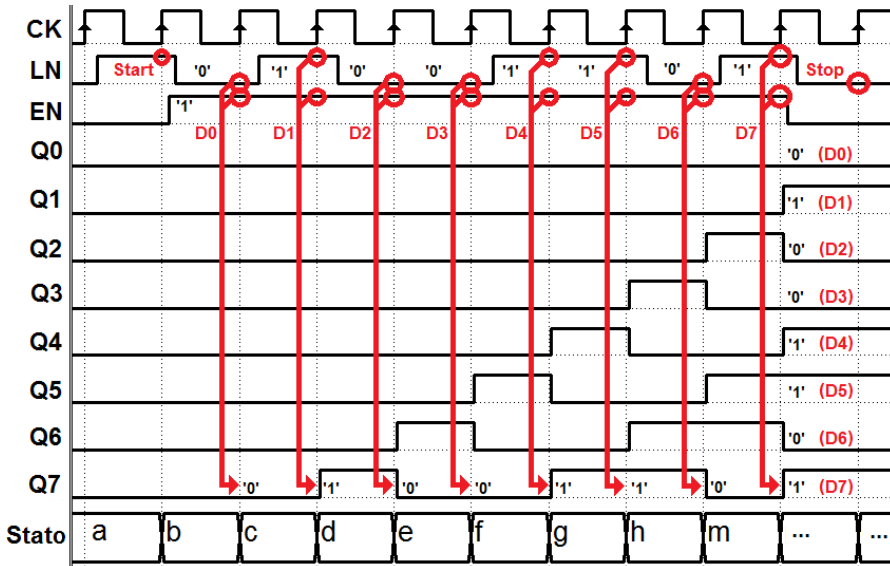
Esaminiamo la prima delle due nuove versioni del ricevitore. Il *datapath* è formato dal solo registro a scorrimento: il componente “SiPo8” del simulatore *Deeds* (vedi il riquadro seguente).



Come si vede nella figura qui sotto, l'ingresso IN del *registro a scorrimento* è pilotato direttamente dalla linea LN , senza mediazione da parte della MSF. La linea E del registro, e cioè l'abilitazione allo scorrimento, è controllata dalla linea EN della MSF. Il reset del sistema azzerà, oltre alla MSF, anche il registro.



Per effettuare correttamente il progetto della MSF è necessario aver chiara la temporizzazione dei segnali in gioco. Spesso conviene, prima di disegnare il diagramma ASM, tracciare uno schizzo dell'evoluzione dei segnali nel tempo. La figura che segue ne suggerisce un esempio, dove si è ipotizzata una sequenza seriale che trasporta, nell'ordine, i bit $D0..D7 = 01001101$.

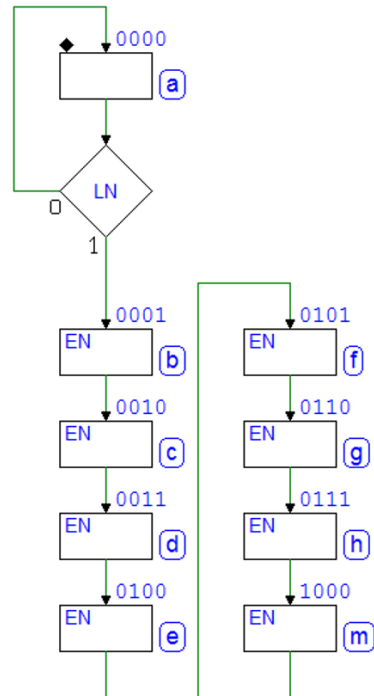


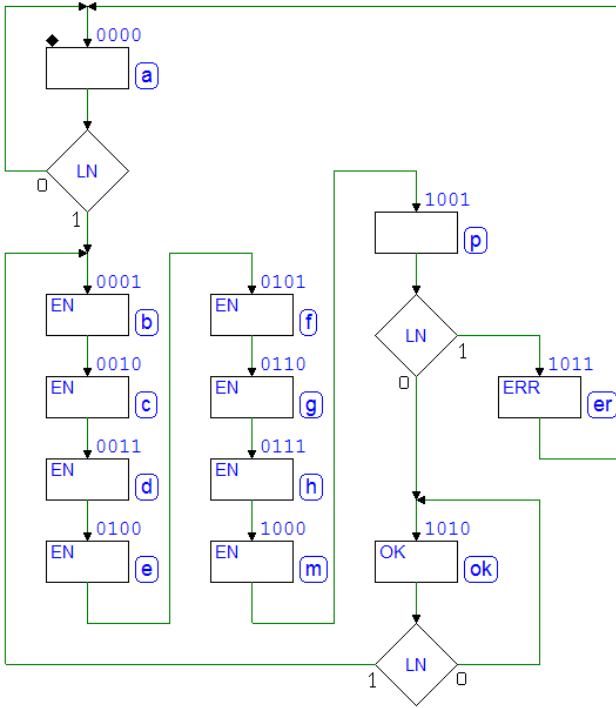
Vogliamo caricare i bit, uno dopo l'altro, nel registro a scorrimento: come si osserva nel tracciato preliminare qui sopra, dobbiamo attivare *EN* non appena individuato il bit di start, e mantenerlo attivo per 8 cicli di clock.

I bit entrano in *Q7* e, clock dopo clock, scorrono verso *Q0*. Quando disattiviamo *EN*, dopo che i bit di dato sulla linea sono terminati, sulle uscite del registro *Q0..Q7* saranno presenti tutti i bit ricevuti, *in parallelo*.

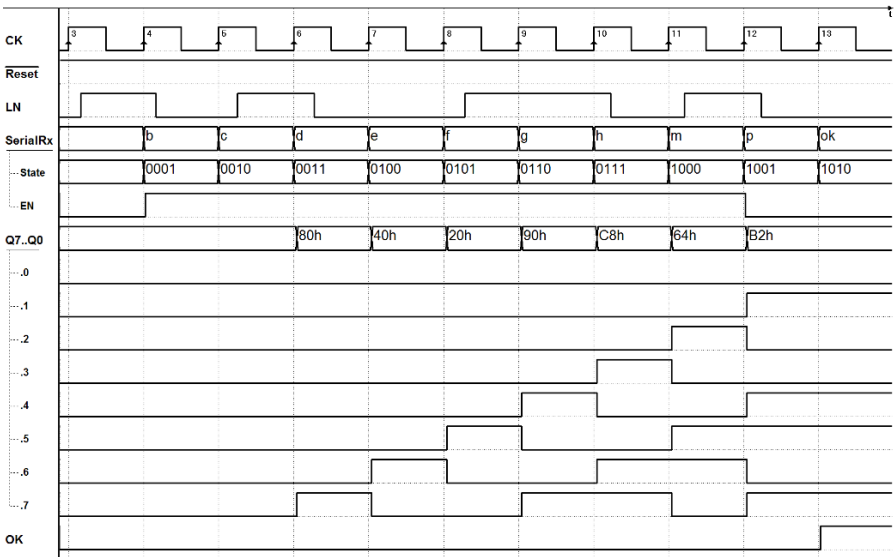
Nel diagramma ASM, quindi, dopo l'attesa del *bit di start* nello stato (a), a partire dal prossimo stato (b) attiveremo *EN* (vedi figura qui a lato). Per mantenerlo attivo per 8 cicli, dovremo in totale inserire 8 stati con *EN* attivo (b)..(m).

Per finire, si esegue il controllo *del bit di stop* e si generano *OK* o *ERR* secondo le specifiche, come nei precedenti esercizi, ottenendo il diagramma ASM completo (vedi figura successiva).

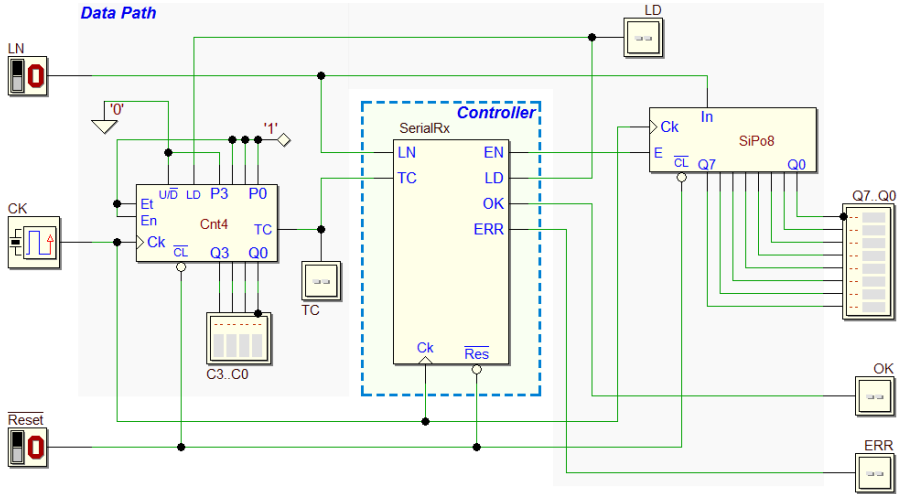




Concludendo, con una simulazione temporale completa, potremo verificare il corretto funzionamento del sistema:



La versione del ricevitore ora esaminata utilizza una struttura esterna per memorizzare i dati seriali ma usa ancora la MSF per contare il *numero dei bit*. È ovviamente possibile delegare al *datapath* anche questa funzione, utilizzando un contatore, predisposto per contare all'indietro, da 7 a 0. Il contatore è il "Cnt4", utilizzato in un precedente esempio (pag. 380). Lo schema del nuovo sistema contiene ora due elementi nel *datapath* (registro e contatore) ed ottimizza la distribuzione dei compiti tra questo e il *controllore*:

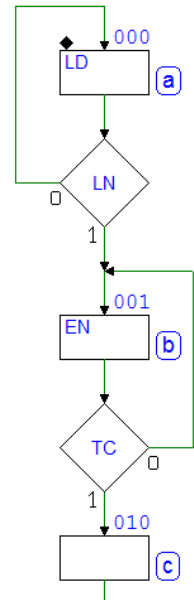


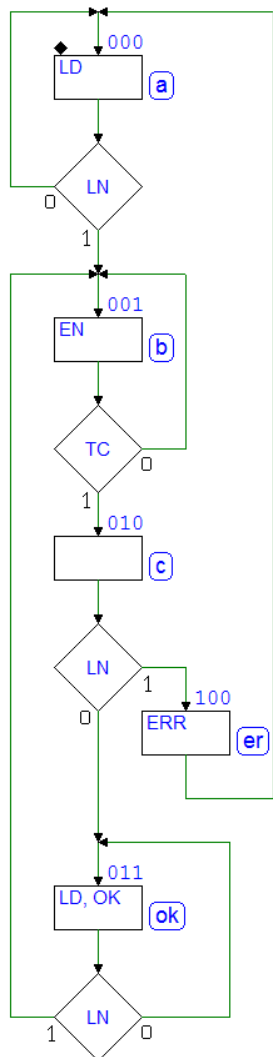
L'inizio del diagramma ASM è uguale alla precedente versione (vedi figura a destra). A riposo, la macchina attende in (a) il bit di start sulla linea *LN*, mantenendo bloccato il contatore tramite la linea *LD* (per i motivi già visti).

Alla rilevazione del bit di start la MSF, passando nello stato (b), disattiva *LD* ed attiva l'ingresso *EN* del registro, come nel caso precedente, con la differenza che utilizza un solo stato, invece di introdurre tanti stati quanti sono i bit da inserire nel registro.

Il contatore è sempre abilitato ($En = Et = 1$), per cui non appena *LD* è disattivato, ad ogni successivo fronte di salita del clock le sue uscite decrementeranno, a partire dal numero 0111 ($P3..P0$ sono impostati a questo valore, vedi schema della rete, qui sopra).

Durante l'esecuzione del ciclo di attesa nello stato (b) avvengono quindi sia la memorizzazione dei dati (ad opera del registro), che il conteggio dei bit (ad opera del contatore), in attesa della attivazione del *TC*.





Quando il conteggio arriva a 0000, il contatore attiva TC , per cui la MSF esce dal ciclo sullo stato (b) e passa in (c), al tempo giusto per controllare il bit di stop della sequenza.

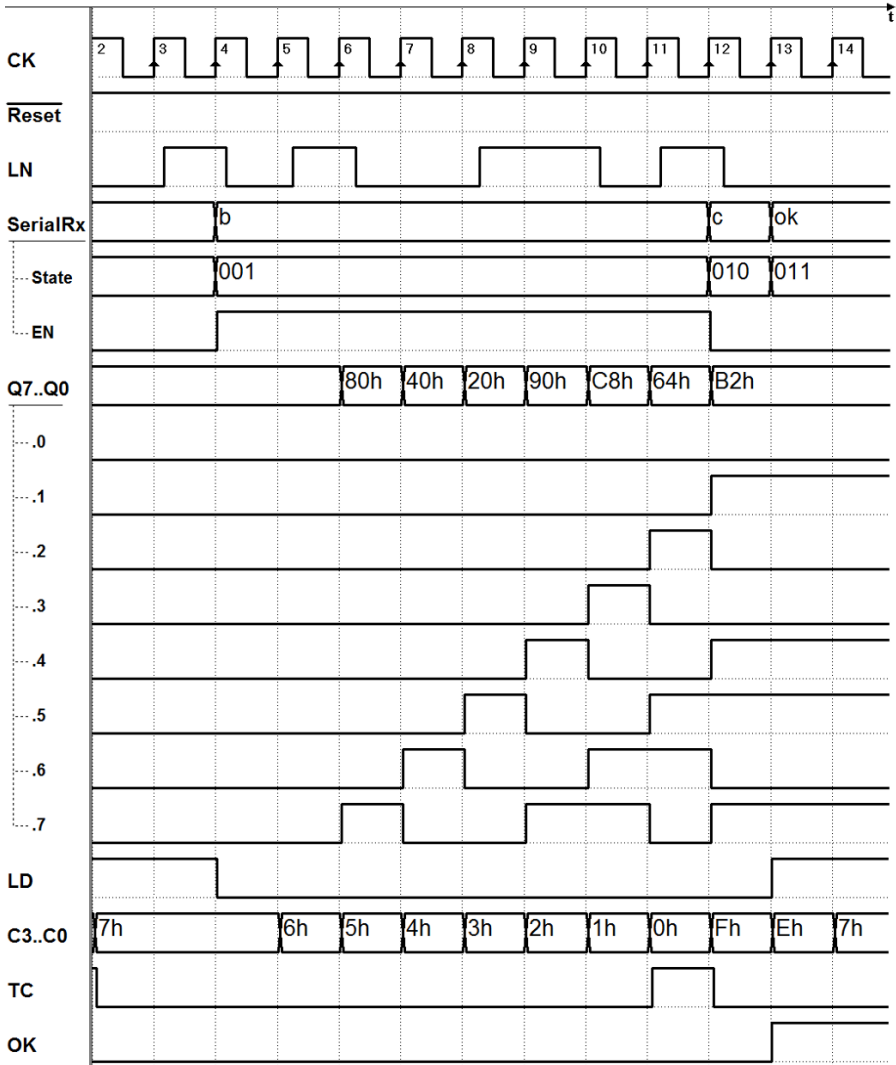
Il seguito è quasi identico alla precedente soluzione, salvo l'attivazione di LD nello stato (ok), come avviene in (a).

Abbiamo così ottenuto una MSF rimarcabilmente semplice e facile da analizzare.

Si noti che anche qui risulta di uso più generale, in quanto lo *stesso identico algoritmo* può controllare, senza cambiare i componenti nel *datapath*, sequenze seriali con un numero di bit ridefinibile (cambiando solo il valore costante applicato agli ingressi $P3..P0$ del contatore).

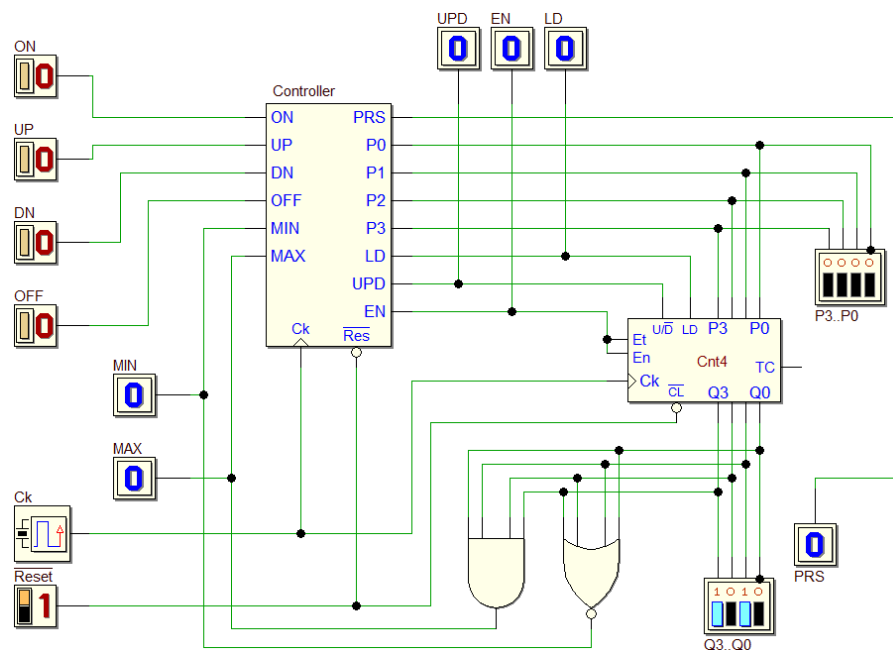
La simulazione temporale del progetto completo ci consente adesso di verificare nel suo complesso la relazione tra l'andamento degli stati e il funzionamento del registro e del contatore (vedi la figura seguente). In particolare, osserviamo:

- il decremento del contatore da 7 a 0, a partire dal fronte 5 del clock, e cioè *alla fine* del ciclo in cui la MSF si trova nello stato (b) per la prima volta;
- la permanenza nello stato (b) fino a quando TC non viene letto a 1 (sul fronte 12);
- che il contatore, dopo avere raggiunto lo zero (0000), per sua natura prosegue a contare ciclicamente: 1111, 1110... finché la MSF non attiva LD nello stato (ok), per cui sul fronte 14 viene ricaricato a 0111.



8.3.4 Regolatore di Luminosità per Lampada

Il sistema nella figura qui sotto realizza un controllore di luminosità (*light dimmer*), comandato da quattro pulsanti (*ON*, *UP*, *DN* e *OFF*). Le uscite $Q3..Q0$ sono prelevate direttamente dal contatore.



A valle del nostro sistema, le linee $Q3..Q0$ pilotano un circuito che ne esegue la conversione in una grandezza analogica, facendo corrispondere al numero binario sedici possibili valori di intensità luminosa, da 0000 (lampada spenta), a 1111 (intensità massima). Questa componente del sistema non fa parte del nostro progetto e non è rappresentata nello schema.

I pulsanti generano 0 a riposo e 1 per il tempo in cui sono premuti. La pressione ed il successivo rilascio di ciascun pulsante determina il comportamento del sistema:

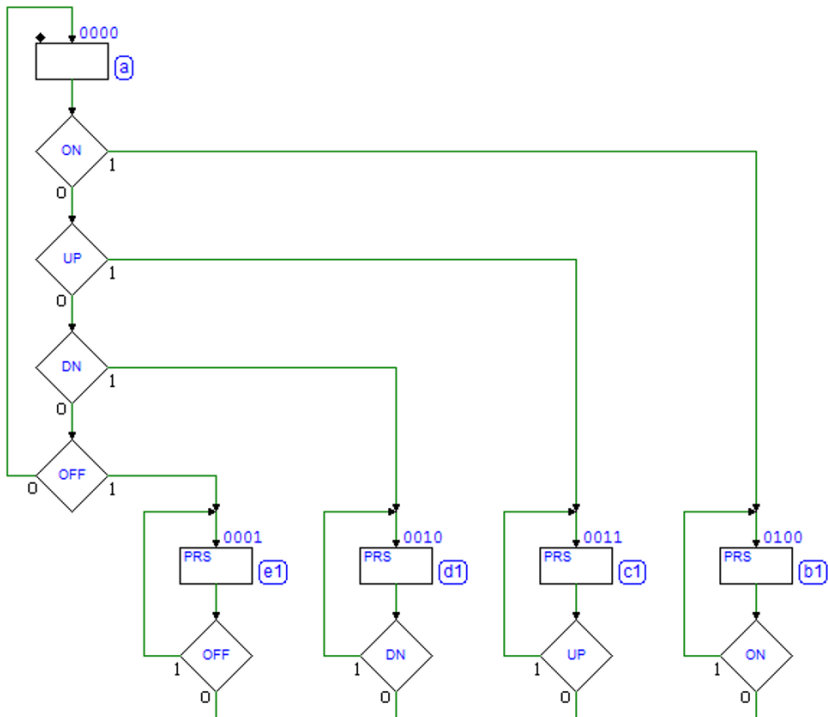
- ON* : accende la lampada al valore massimo di intensità ($Q3..Q0 = 1111$);
- UP* : determina l'incremento di una unità di intensità luminosa;
- DN* : determina il decremento di una unità di intensità luminosa;
- OFF* : spegne la lampada ($Q3..Q0 = 0000$).

L'uscita *PRS* serve come segnalazione e viene attivata per tutto il tempo in cui uno dei pulsanti è premuto. Il numero in uscita $Q3..Q0$ non deve essere incrementato o decrementato se ha già raggiunto, rispettivamente, i valori massimo o minimo.

Il sistema usa il contatore “Cnt4”, che qui è utilizzato in modo più completo rispetto ad altri esempi precedenti (pagine 380 e 385), in quanto deve essere *caricato, abilitato e predisposto al conteggio*, in avanti o all’indietro.

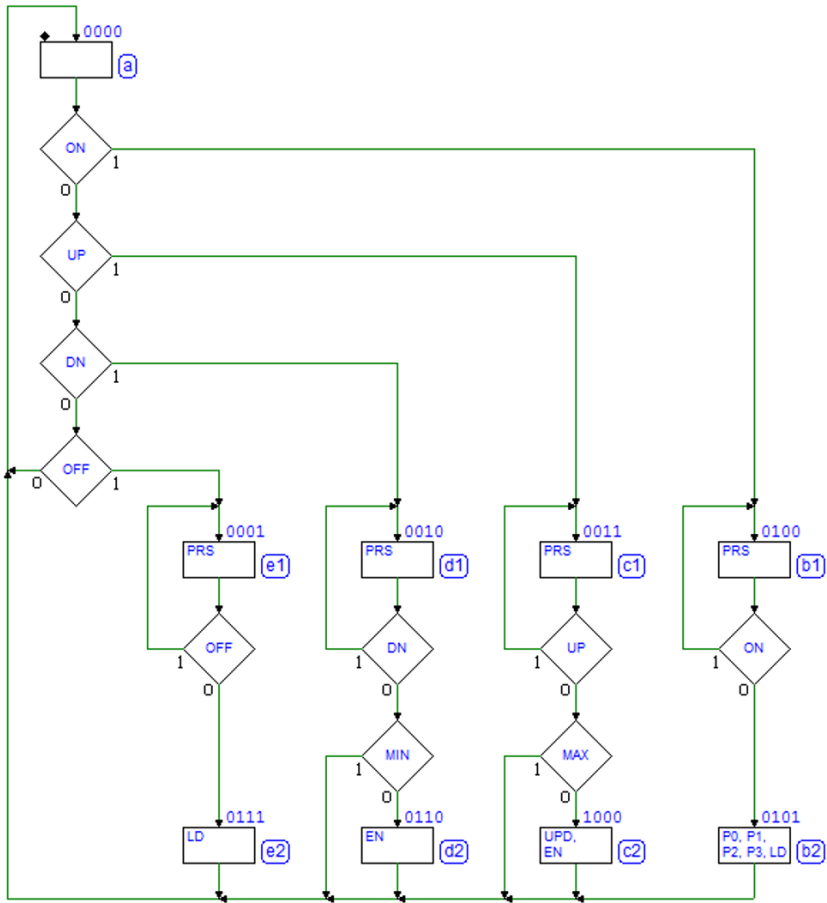
Le due porte logiche hanno per ingresso le uscite del contatore $Q3..Q0$: la AND attiva *MAX* quando il conteggio ha raggiunto il massimo, la NOR attiva *MIN* quando ha raggiunto il minimo.

Per quanto riguarda la lettura dei pulsanti, valgono le considerazioni fatte nel progetto relativo alla “*Gestione di pulsanti*” (pag. 313). La struttura della prima parte del diagramma ASM, riportata qui sotto, fa riferimento a quanto già esaminato in quel progetto: il controllo simultaneo della pressione dei pulsanti e la attesa del rilascio del pulsante che è stato premuto.



Come si vede in figura, il diagramma si dirama in quattro percorsi separati, corrispondenti ai pulsanti *OFF*, *DN*, *UP* e *ON*. Sulla base delle specifiche, l’uscita *PRS* è attiva negli stati (e1), (d1), (c1) e (b1), in cui la MSF si trova quando uno dei pulsanti è premuto e siamo in attesa del suo rilascio.

Completiamo il diagramma lungo i quattro percorsi, tenendo conto dell’operatività richiesta al rilascio dei pulsanti: per ora consideriamo i percorsi relativi al rilascio di *OFF* e di *ON*. Come visibile nella figura successiva, negli stati (e2) e (b2) inviamo al contatore il comando di caricare quanto predisposto ai suoi ingressi $P3..P0$, mediante l’attivazione di *LD*.



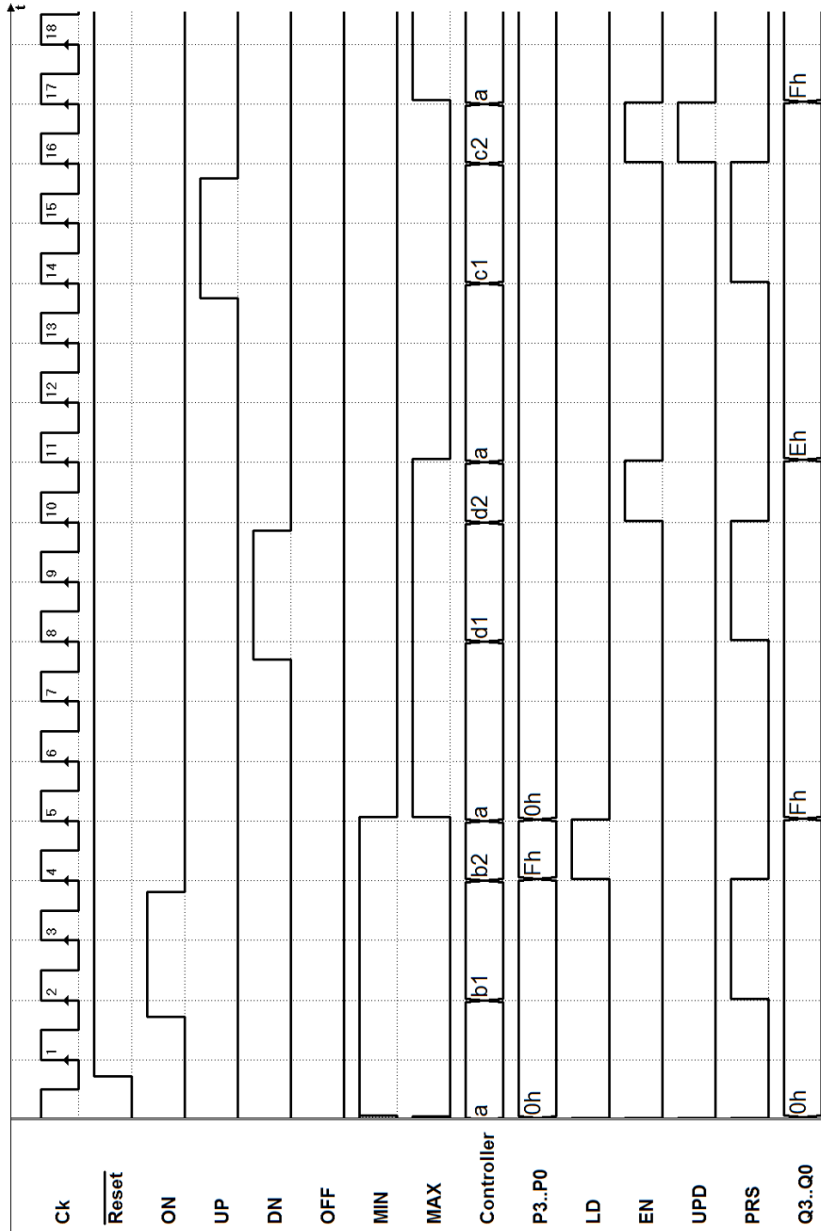
Allo scopo, in (e2) la MSF imposta $P3..P0 = 0000$, mentre in (b2) assegna agli ingressi $P3..P0$ il valore 1111. Nel primo caso, quindi, il contatore sarà caricato al valore minimo (lampada spenta), nel secondo al valore massimo (lampada completamente accesa).

Si osservi che l'impostazione delle linee $P3..P0$ e di LD avviene nel medesimo stato, in contemporanea. L'effettivo caricamento, dato che LD è sincrono, ha luogo sul fronte del clock che fa uscire dallo stato e che fa tornare in (a).

Consideriamo ora il diagramma in relazione al decremento e all'incremento del numero (pulsanti DN e UP). Negli stati (d2) e (c2) hanno luogo, rispettivamente, il decremento e l'incremento del contatore, previo controllo di MIN e MAX , perché il numero non deve essere modificato se ha già raggiunto, rispettivamente, i valori *minimo* o *massimo*. In questi stati occorre attivare EN per abilitare il conteggio, e contestualmente determinarne la direzione con UPD : nello stato (d2) UPD è posto a 0 (*conteggio all'indietro*), mentre in (c2) è messo a 1 (*conteggio in avanti*).

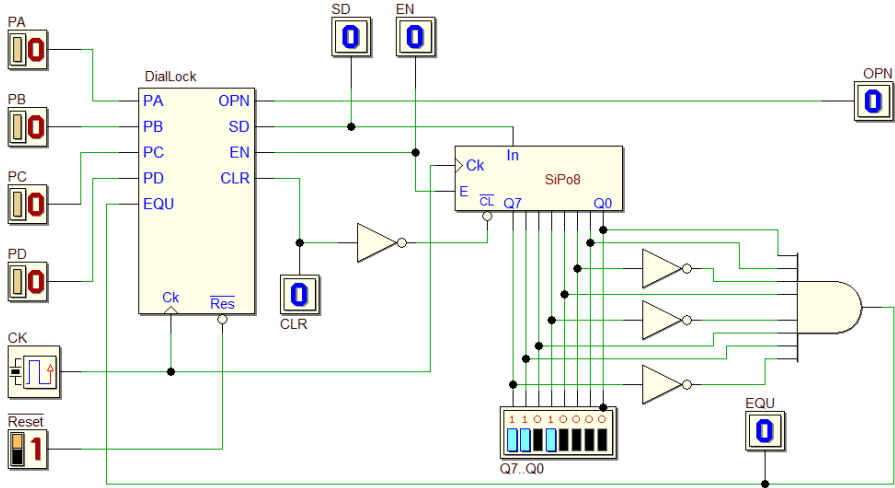
Si faccia attenzione al fatto che il conteggio avviene sul fronte attivo del clock; di conseguenza, anche l'incremento o il decremento avverranno *in uscita* dagli stati (d2) e (c2) in cui è abilitato, ossia nell'istante del rientro in (a).

Qui di seguito, una simulazione temporale del sistema: si faccia attenzione agli istanti di caricamento, incremento e decremento ora discussi.



8.3.5 Serratura a combinazione

Il sistema comanda l'apertura di una porta dotata di serratura elettrica attraverso una *combinazione* inserita dall'utente. La serratura, comandata dall'uscita *OPN*, si apre premendo in una determinata successione i pulsanti di ingresso *PA*, *PB*, *PC* e *PD*.



Il *datapath* è formato da un registro a scorrimento “SiPo8” (è stato descritto in un precedente esempio, vedi pag. 385) e da una semplice logica combinatoria che ne legge le uscite ed attiva l'ingresso *EQU* della MSF quando assumono il valore $Q7..Q0 = 01101011_2$ (il codice interno della *combinazione*).

Gli ingressi *In* ed *E* del registro sono pilotati, rispettivamente, dalle uscite *SD* (*serial data*) e *EN* della MSF. Si osservi che, a differenza dei casi precedenti, il \overline{Reset} agisce soltanto sulla MSF, mentre il comando di \overline{CL} del registro è generato dall'uscita *CLR* della MSF.

Quando l'utente preme un pulsante, *al suo rilascio* il sistema inserisce nel registro a scorrimento “SiPo8” un codice formato da due bit:

- PA* : carica il codice 00;
- PB* : carica il codice 01 (prima 1, poi 0);
- PC* : carica il codice 10 (prima 0, poi 1);
- PD* : carica il codice 11.

Nel nostro caso, la successione di azionamento dei pulsanti (la *combinazione*) che fa scattare la serratura è:

$$PD \rightarrow PC \rightarrow PC \rightarrow PB .$$

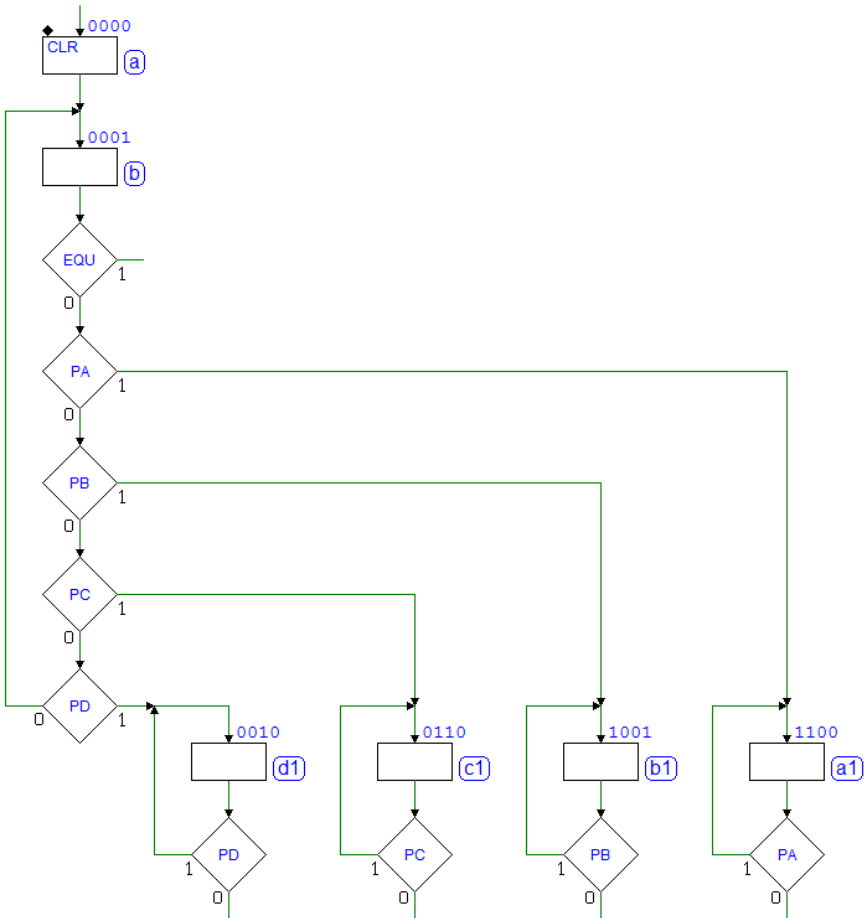
Quando il dato nel registro ha un valore uguale a quello prodotto da questa sequenza di ingressi (01101011_2), la MSF attiva per un ciclo di clock l'uscita *OPN*, e quindi azzerà il dato contenuto nel registro.

Dopo l'apertura della porta, il blocco della serratura avverrà meccanicamente, quando sarà richiusa. Si assume che:

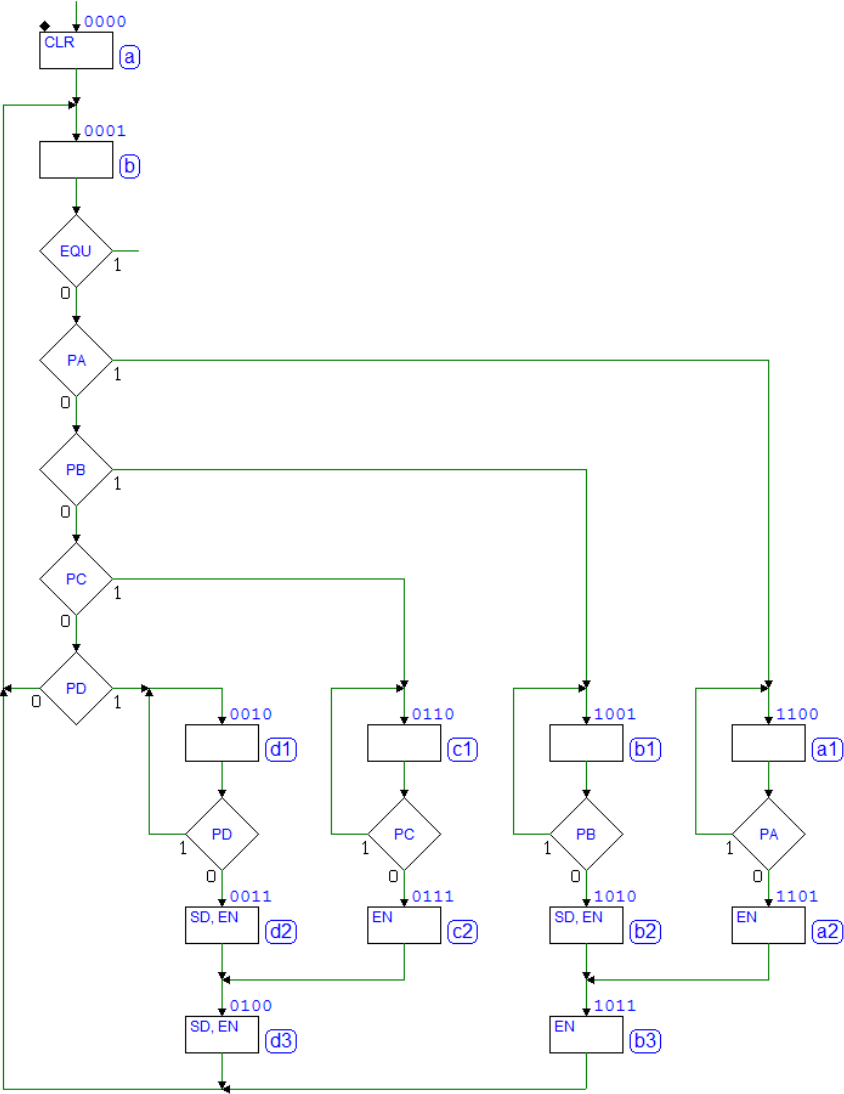
- la pressione di un pulsante attiva ad 1 la corrispondente linea in ingresso alla MSF, mentre, al rilascio del pulsante, la linea ritorna a 0;
- non vengono mai premuti contemporaneamente più pulsanti;
- tra il rilascio di un pulsante e la pressione del successivo trascorre un tempo sufficiente affinché il sistema concluda le sue operazioni.

Come si vede nella figura qui sotto, la sezione del diagramma ASM che controlla i pulsanti è già stata incontrata in esercizi precedenti (p.es. in “*Gestione di pulsanti*”, a pag. 313).

Si osservi che nel nostro caso la MSF, al reset del sistema, si porta nello stato (a), che *precede* il ciclo di controllo dei pulsanti, e dove *CLR*, attivo, azzerava il registro (la ragione di ciò apparirà chiara nel seguito).

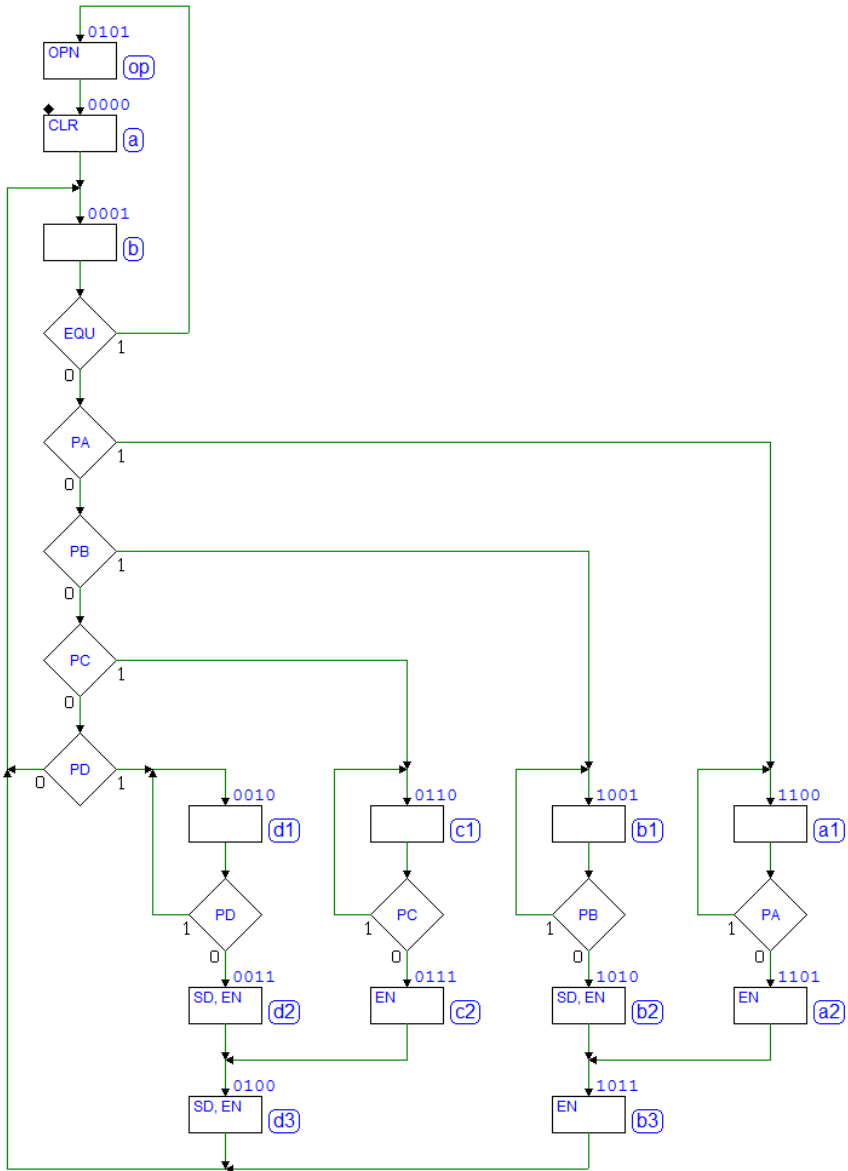


Gli stati successivi hanno il compito di *caricare nel registro* i due bit corrispondenti al pulsante azionato. La figura qui sotto propone un diagramma ASM quasi completato. Lungo il percorso relativo alla pressione e rilascio del pulsante *PD*, sono stati inseriti gli stati (d2) e (d3) che, attivando per due cicli di clock *EN* e *SD*, caricano la coppia di bit 11 nel registro.



Analogamente, nello stato (c2) carichiamo nel registro uno 0, sfruttando poi lo stato già esistente (d3) per caricare il successivo 1. Analogo lavoro è eseguito negli altri stati (b2), (b3) e (a2). Si vede anche qui come si possa usare più volte uno stesso stato, lungo percorsi differenti.

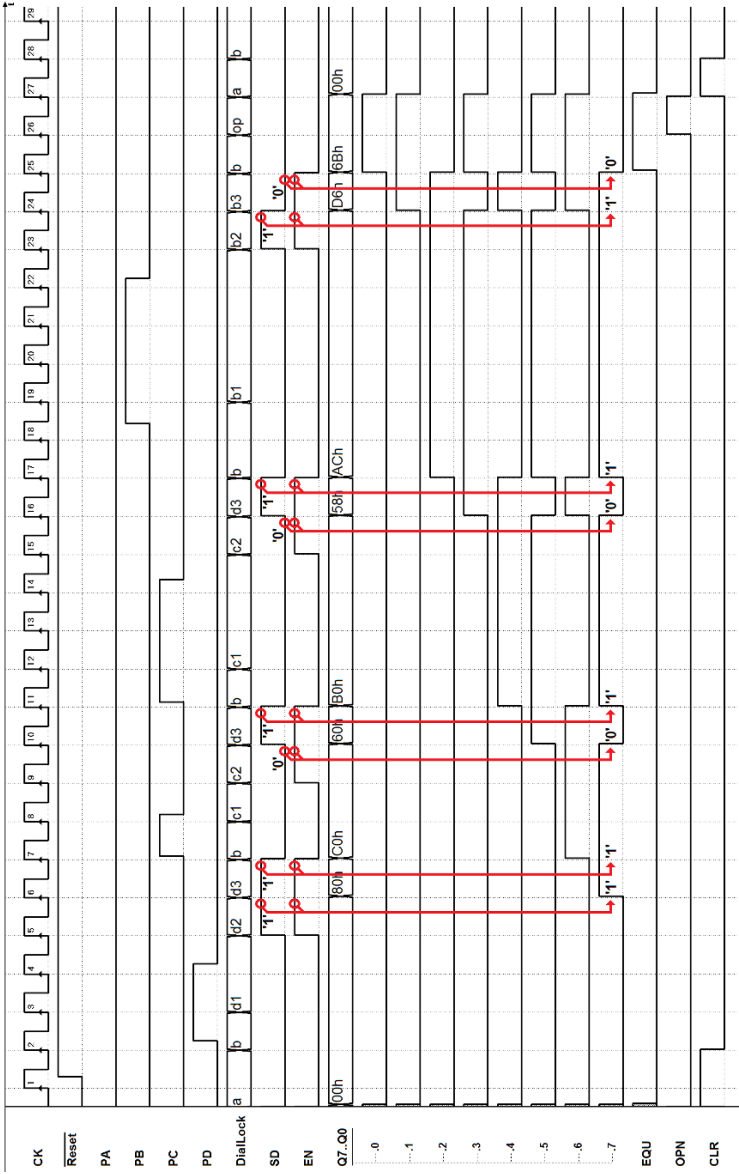
Concludiamo ora il diagramma considerando le specifiche a riguardo alla effettiva apertura della porta (vedi figura qui riportata).



Si noti che il controllo di *EQU* è inserito per semplicità nello stesso ciclo che controlla i pulsanti. Quando *EQU* vale 1, la MSF transisce nello stato (op) attivando *OPN* per un ciclo, come richiesto, e quindi si porta nello stato (a), che ha il compito di azzerare il registro (senza questa operazione, una volta impostata la combinazione corretta, *OPN* sarebbe sempre attiva).

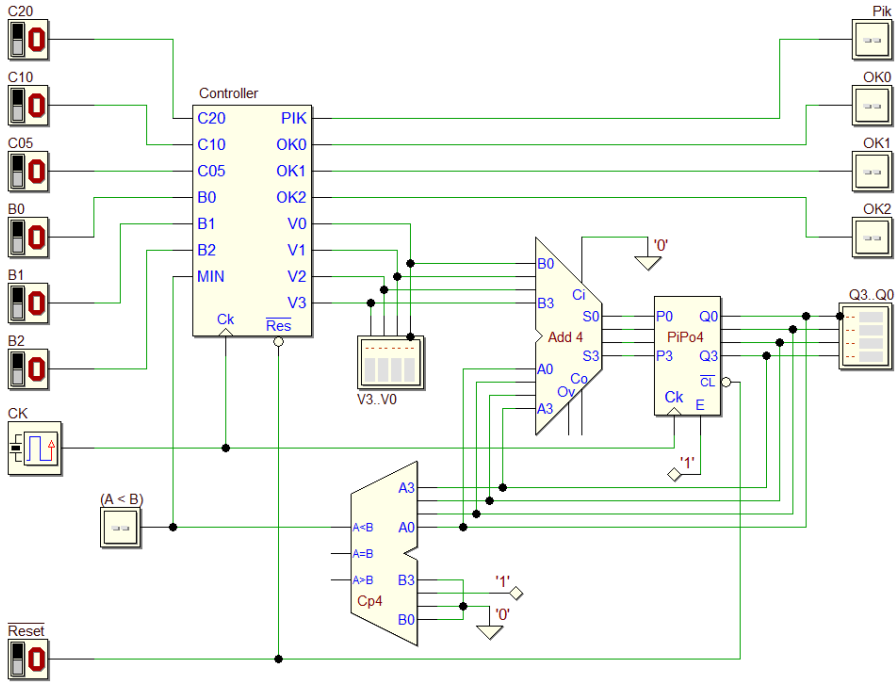
Infine, si osservi che (a) è impostato anche come *stato di reset*, perché il registro, come commentato in precedenza, non è azzerato direttamente dal reset di sistema, ma necessita di essere azzerato prima di entrare nel ciclo di attesa della pressione dei pulsanti.

Nel diagramma temporale che segue, risultato della simulazione del sistema in *Deeds*, sono state messe in evidenza le uscite della MSF *SD* e *EN*, in relazione con i valori caricati nel registro a scorrimento.



8.3.6 Distributore automatico di bevande

Progettiamo un sistema digitale che gestisce un distributore automatico di bevande (semplificato). Il sistema è formato da una MSF sincrona, più un *datapath* composto da un *circuito di calcolo a 4 bit*, che comprende un *sommatore* (“Add4”), un *registro parallelo* (“PiPo4”) ed un *comparatore di grandezza* (“Cp4”). Sia la MSF che il registro sono inizializzati dal *Reset* di sistema.



Le bevande offerte dal distributore sono di *tre tipi* e hanno tutte lo stesso costo: *20 centesimi*. Il distributore attende che l'utente inserisca la somma di denaro per pagare la bevanda. Gli ingressi *C05*, *C10* e *C20*, indicano, rispettivamente, l'introduzione di una moneta da 5, 10 oppure 20 centesimi: la discesa della moneta nel distributore produce, sulla linea corrispondente, un *impulso* a livello alto della durata di *un ciclo di clock* (a riposo, le linee sono a livello basso).

Il sistema calcola il totale introdotto nel distributore, sommando i valori delle monete inserite, memorizzandolo nel registro “PiPo4”. Per convenzione, un'unità di *Q3..Q0* corrisponde a *5 centesimi*: per esempio $Q3..Q0 = 0100_2 = 4_{10}$ corrisponde a $(4 \cdot 5) = 20$ centesimi.

L'ingresso del registro è collegato all'uscita del sommatore “Add4”. Uno dei due ingressi del sommatore è collegato all'uscita del registro, mentre l'altro è fornito direttamente dalla MSF mediante le linee *V3..V0*: di conseguenza, il sommatore genera, istante per istante, la somma del numero presente nel

registro con quello fornito dalla MSF. La MSF aggiunge al registro il *valore della moneta* introdotta, fornendolo al sommatore per un ciclo di clock. Dato che l'ingresso di abilitazione E del registro è sempre attivo, per mantenere inalterato il totale calcolato sarà poi necessario che nel resto del tempo la MSF mantenga le linee $V3..V0$ a zero.

Il comparatore "Cp4" confronta il totale $Q3..Q0$ accumulato nel registro con il prezzo stabilito per l'erogazione ($= 0100_2 = 4 = 20$ centesimi, predisposto sugli ingressi $B3..B0$ del comparatore). Delle tre uscite $A < B$, $A = B$, $A > B$, che si attivano in base al risultato del confronto, se ne utilizza solo una $A < B$, letta dalla MSF all'ingresso MIN ("Minore").

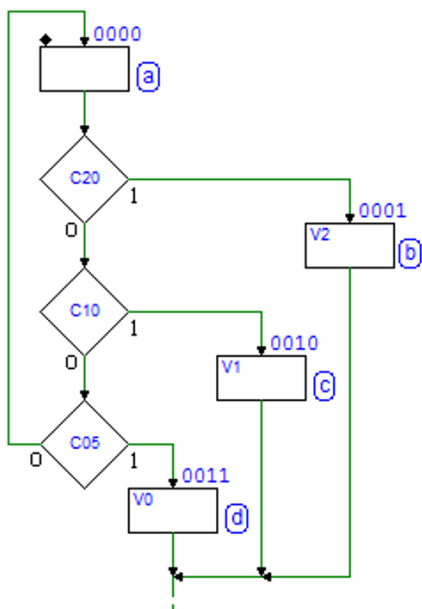
Al raggiungimento di *almeno* 20 centesimi, il distributore attiva l'uscita PIK per invitare l'utente ad effettuare la scelta della bevanda, premendo uno dei tre pulsanti $B0$, $B1$ o $B2$ (PIK accende la retroilluminazione dei pulsanti). Alla pressione di uno dei pulsanti, la MSF disattiva PIK ed attiva, per un ciclo di clock, una delle uscite $OK0$, $OK1$ o $OK2$ corrispondente alla scelta, comandando così l'erogazione della bevanda. I pulsanti $B0$, $B1$ e $B2$ sono normalmente al valore logico 0 e vanno a 1 quando sono premuti.

Infine, il distributore *incassa* il costo della bevanda, ma *accredita* al prossimo utente l'importo introdotto *eccedente* i 20 centesimi. Per fare questo, la MSF deve sottrarre -20 centesimi dal valore immagazzinato nel registro, generando sulle linee $V3..V0$, per un ciclo di clock, il *complemento a due* del codice corrispondente al valore della bevanda. Dopo questa operazione, il registro conterrà il credito, che sarà riutilizzabile alla prossima erogazione.

Come ulteriore specifica, si supponga che tutti i segnali di ingresso siano sincroni con il clock. Inoltre, si ipotizzi che non sia possibile l'introduzione contemporanea di due monete, e che tra una moneta e la successiva trascorra un tempo sufficiente per il corretto funzionamento del sistema.

Attendiamo in (a) l'introduzione di una moneta (vedi figura a lato).

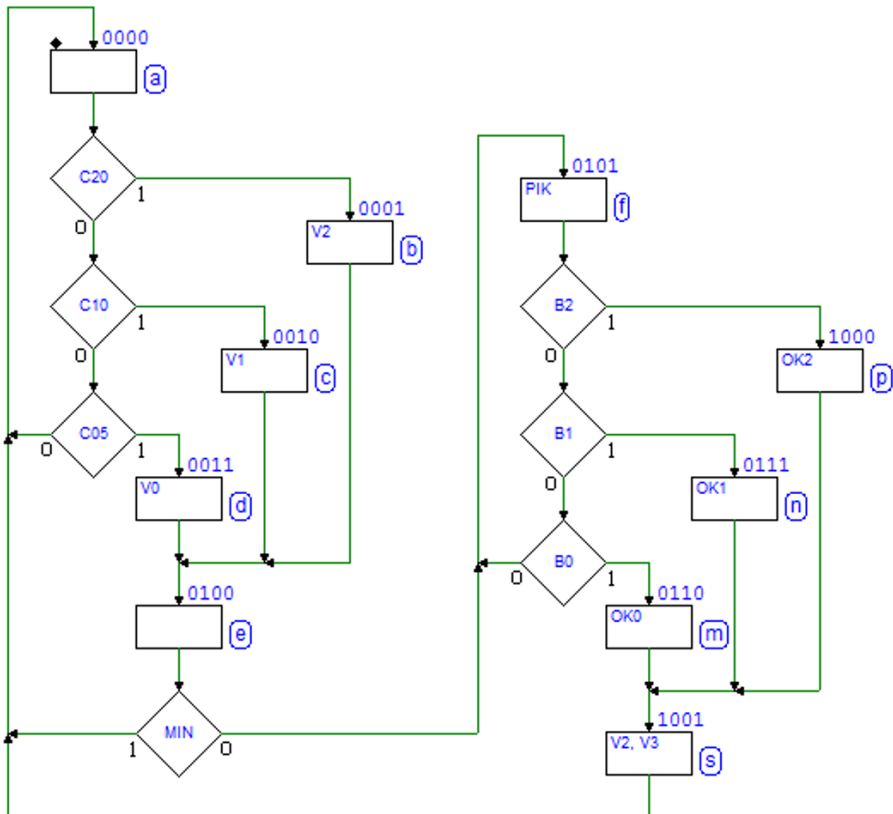
A seconda del suo valore, attiva per un ciclo di clock $V2$, $V1$ oppure $V0$, e cioè fornisce al sommatore, su $V3..V0$, i numeri binari 0100_2 , 0010_2 o 0001_2 , corrispondenti ai valori di 20, 10 o 5 centesimi: negli stati (b), (c) o (d), il sommatore esegue la somma tra il numero $V3..V0$ e quello presente nel registro.



La somma è memorizzata dal registro al *successivo fronte* di clock, e per questo aggiungiamo lo stato (e) per attendere un ciclo di clock prima di controllare *MIN* (vedi il diagramma ASM completo qui sotto).

Si torna in (a) se il costo della bevanda non è stato raggiunto. Quando la somma ha raggiunto un valore uguale o superiore a 20 centesimi, *MIN* si azzerava e la MSF passa nello stato (f), dove attiva *PIK* per invitare l'utente a fare una scelta ed attende la pressione di uno dei pulsanti di erogazione.

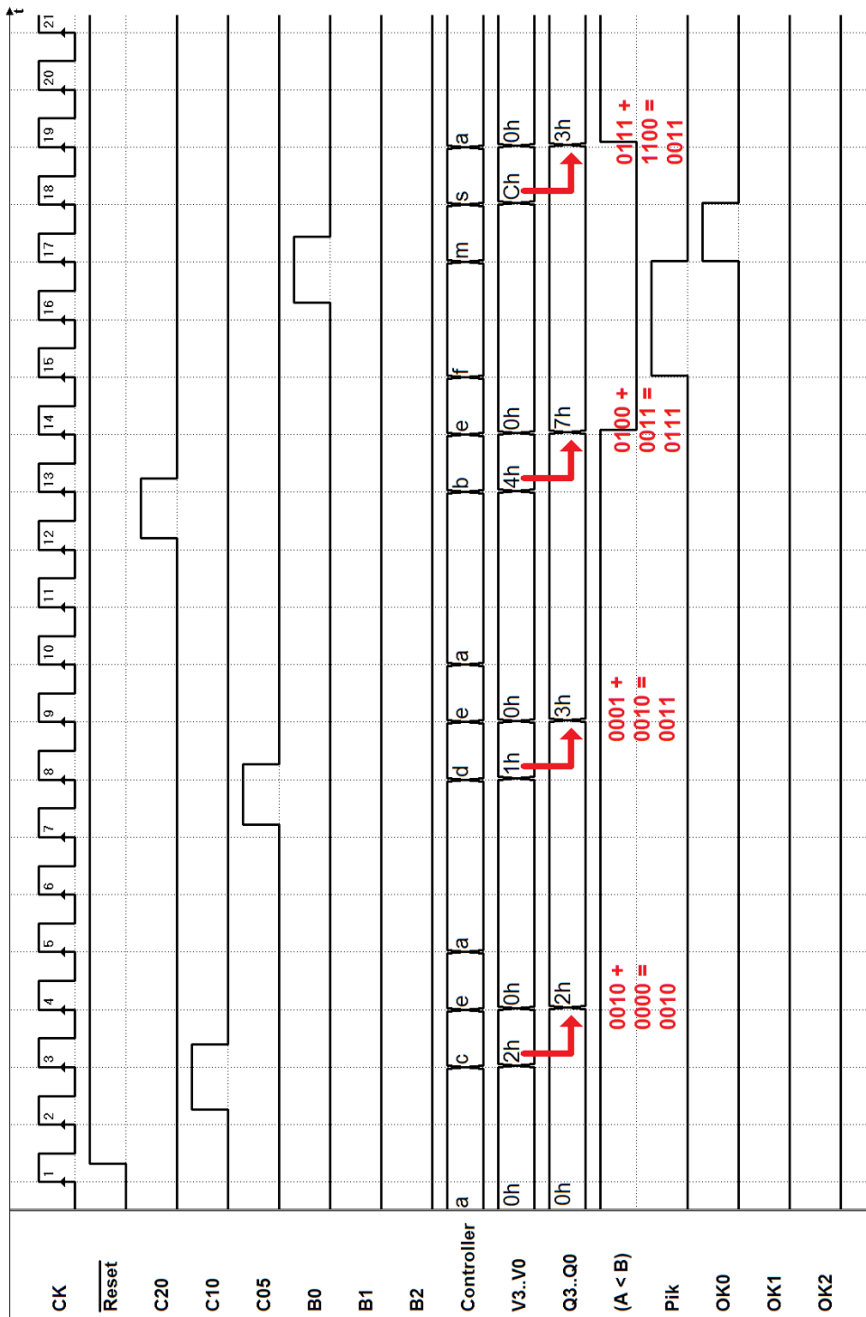
A seconda del pulsante premuto, si attiva per un ciclo di clock *OK2*, *OK1* o *OK0*, per comandare l'erogazione della bevanda corrispondente.



Infine, nel richiudere il diagramma, facciamo in modo che il credito residuo sia a disposizione del prossimo utente: per sottrarre dal registro l'equivalente di 20 centesimi, la MSF imposta nello stato (s) le uscite $V3..V0 = -4_{10} = 1100_2$ (il complemento a due del codice corrispondente ai 20 centesimi).

Nella simulazione temporale del sistema, nella figura successiva, sono state evidenziate le operazioni di somma tra il valore impostato dalla MSF su $V3..V0$ e il valore presente in quel momento nel registro.

In particolare, si osservi l'ultima somma, quando al totale (35 centesimi = $7 = 0111_2$) è sottratto il costo della bevanda, sommandone il complemento a due (-20 centesimi = $-4_{10} = 1100_2$):



8.3.7 Progetto di generatore di onda quadra programmabile

Premessa: in questo caso non viene già fornita la struttura della rete, come negli esempi precedenti, ma deve essere progettata, seguendo le specifiche.

Si vuole progettare un sistema sincrono (*controllore - datapath*) che ha il compito di generare un segnale periodico a 2 livelli (chiamato nel seguito “onda quadra”). Il sistema ha un gruppo di 8 linee di ingresso (che chiamiamo TH), un secondo gruppo di 8 fili di ingresso (TL) e una linea di uscita SW . All’attivazione, senza attendere alcun comando, il sistema genera su SW un’onda quadra in cui la parte alta dura un numero di cicli del clock CK uguale al numero impostato su $TH + 1$ e la parte bassa di durata pari al numero impostato su $TL + 1$. La generazione di SW inizia con la parte alta del periodo.

Soluzione

Leggendo attentamente le specifiche, si ricava che il sistema deve generare, su SW , un segnale periodico che rimane al valore logico 0 per $TL + 1$ cicli di CK (il gruppo di 8 ingressi denominato TL codifica un numero binario a 8 bit) e poi al valore logico 1 per $TH + 1$ cicli di CK (l’altro gruppo di 8 ingressi denominato TH codifica anch’esso un numero binario a 8 bit).

I passi da affrontare per procedere in un progetto di questo tipo sono i seguenti:

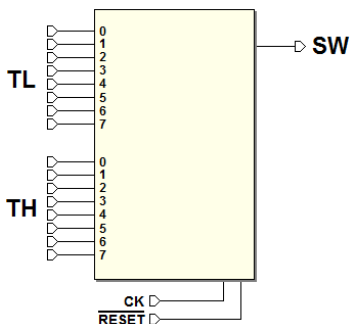
- (a) definizione del sistema come blocco funzionale, evidenziandone ingressi ed uscite;
- (b) scelta dei componenti da usare nel datapath;
- (c) progetto del datapath con i collegamenti del controllore con gli ingressi e le uscite, verso l’esterno ed il datapath stesso;
- (d) progetto del diagramma ASM della MSF che descrive il controllore;
- (e) simulazione temporale del sistema completo, scegliendo sequenze di ingresso opportune a dimostrarne il corretto funzionamento.

Passo (a)

In questa fase è molto importante identificare correttamente gli ingressi e le uscite del sistema, sulla base delle specifiche date; riepiloghiamo quanto definito dal testo:

1. *Ingressi:*
 - a) TL (8 bit) che imposta, in binario puro, la durata della parte bassa del segnale periodico;
 - b) TH (8 bit), come sopra, ma definisce la durata della parte alta;
2. *Uscite:*
 - a) SW (1 bit), che varia nel tempo in modo periodico, alternando il valore logico 0 e il valore logico 1 (tenuti stabili, rispettivamente, per $TL + 1$ e $TH + 1$ cicli di CK).

Il sistema, rappresentato come un unico blocco funzionale, è quindi il seguente:



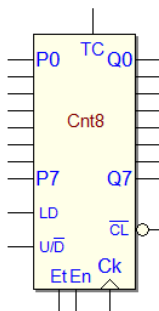
Passo (b)

Per la scelta dei componenti del datapath occorre comprendere quali sono le componentistiche necessarie alla realizzazione del sistema. A tal fine occorre ricordarsi che il controllore deve rimanere tale, e quindi non deve essere usato impropriamente, ad esempio come registro, come contatore, come comparatore o, in generale, per memorizzare e/o elaborare insiemi di dati numerici.

Il controllore deve funzionare da “orchestratore” di componenti combinatori, aritmetici o di memorizzazione, al fine di realizzare il sistema completo finale. Nel caso specifico di questo esempio, il sistema necessita di acquisire *TL* e *TH* al fine di utilizzarli come misura del numero di cicli di *CK* in cui occorre mantenere il segnale *SW*, rispettivamente basso o alto.

Il componente più adatto per fare questo è un contatore universale a 8 bit come il “Cnt8” della libreria di *Deeds*. La sua versione più piccola, il “Cnt4”, è già stata incontrata in precedenza (ad es. a pag. 380).

Permette di essere precaricato con un numero e poi, abilitandone il conteggio all’indietro, di aspettare la generazione del *TC* (*Terminal Count*), che avverrà al raggiungimento del numero di cicli di *CK* impostato +1 (perché conta fino a zero).



Osservando le specifiche quindi, non servono altri componenti specifici se non due contatori universali di questo tipo, uno per *TL* e uno per *TH*.

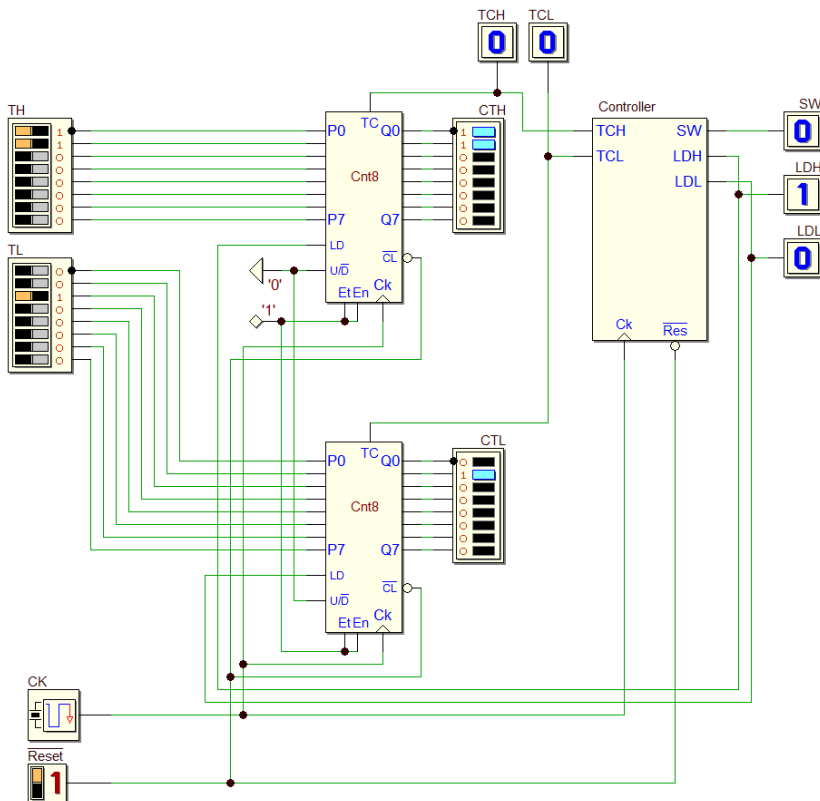
Passo (c)

Questo passo è profondamente legato al passo (b). Come descritto sopra, sono necessari due contatori, abilitati a contare all’indietro, che saranno precaricati dal controllore al momento corretto (ai valori *TL* e *TH*). I *TC* saranno valutati dal controllore per verificare se è trascorso il tempo impostato.

Adesso è necessario capire come collegare i vari ingressi e le varie uscite dei due contatori agli ingressi e alle uscite del sistema e al controllore. In particolare:

- Come descritto in precedenza gli ingressi di precaricamento dei due contatori devono essere collegati ai gruppi di ingressi TL e TH ;
- I contatori devono contare all'indietro: impostiamo l'ingresso $U/D = 0$;
- Gli ingressi LD dei due contatori devono essere comandati dal controllore in modo da essere precaricati al momento giusto. Di conseguenza è possibile impostare al livello logico 1 sia Et che En in modo che il contatore conti sempre e l'unico controllo necessario sia LD . Per esempio, a riguardo di TL , dopo l'attivazione del comando di LD da parte del controllore, il contatore genererà TC dopo $TL + 1$ cicli di CK . Il controllore farà la sua parte, semplicemente aspettando che si attivi TC .
- Si noti che CK e il $Reset$ di sistema vanno collegati sia ai contatori che al controllore; il primo per garantire la sincronia delle operazioni tra contatori e controllore, e il secondo per una comune inizializzazione.
- SW può essere direttamente pilotato dal controllore.
- È opportuno riportare nello schema tutti i vari segnali necessari al sistema: ad esempio LD e TC dei contatori, rispettivamente LDL e TCL per il contatore su TL , LDH e TCH per il contatore su TH , ed infine SW .

Lo schema finale del progetto del datapath con i collegamenti al controllore, agli ingressi e alle uscite, è quindi riportato qui di seguito:



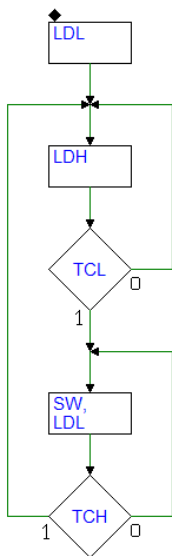
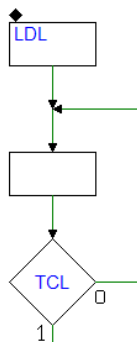
Passo (d)

Una volta definiti i punti (a), (b), e (c), il punto (d) non dovrebbe comportare particolari difficoltà in quanto il controllore dovrà implementare la logica già individuata durante il design del datapath.

Si noti che alla attivazione del \overline{Reset} di sistema i due contatori si azzerano; quando rilasciato, e in assenza di attivazione della LD , i contatori inizierebbero già a contare.

Decidiamo quindi di precaricare, nello stato al reset, uno dei due contatori: quello relativo all'ingresso TL , attivando LDL (vedi figura a destra).

Nello stato successivo non attiviamo più LDL , in modo da lasciarlo libero di contare, ma rimaniamo in attesa del segnale di termine del conteggio TCL . In questo modo manteniamo a zero l'uscita SW per $TL + 1$ cicli di clock.



Tuttavia, per fare in modo che, all'arrivo di TCL , si possa attivare subito SW per $TH + 1$ cicli di clock, possiamo già precaricare il numero TH nell'altro contatore, aggiungendo LDH nello stato di attesa di TCL (vedi figura a sinistra).

Ora occorre disattivare LDH , e attivare l'uscita SW , andando in un nuovo stato. In questo stato il contatore relativo a TH è libero di contare e la MSF rimane in attesa del suo segnale di termine di conteggio, TCH .

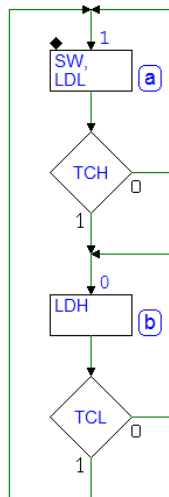
Per lo stesso ragionamento precedente, in questo stato conviene precaricare l'altro contatore, attivando LDL .

In questo modo, all'arrivo di TCH , ritorniamo direttamente nello stato in cui SW non è attivo ad attendere nuovamente l'attivazione di TCL da parte del contatore, e così via ciclicamente.

Se osserviamo il diagramma ASM ora definito, è intuitivo che lo stato al di fuori del ciclo si possa eliminare.

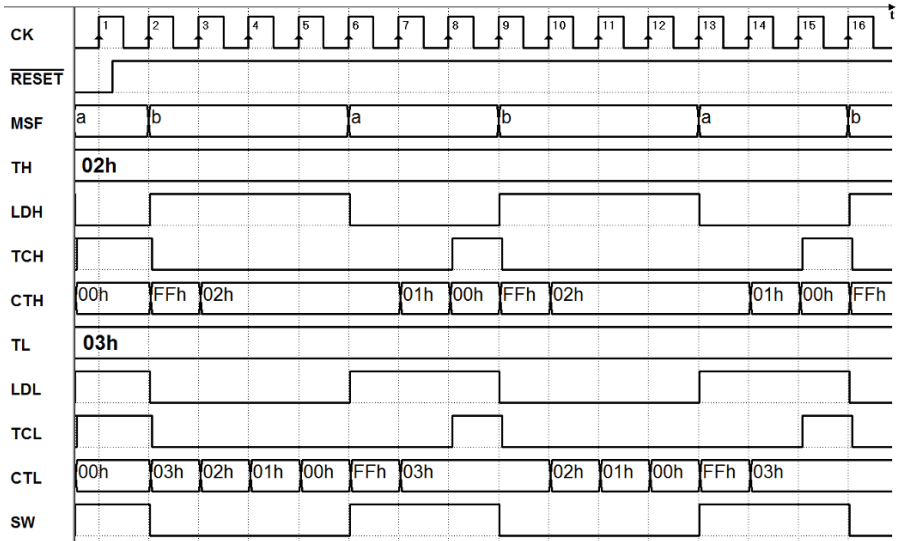
Ricaviamo quindi il diagramma finale, visibile qui sulla destra.

Si noti che, avendo eliminato il pre-caricamento al reset, la prima onda quadra potrà avere un comportamento non corretto; tuttavia, a regime, funzionerà correttamente.



Passo (e)

Per testare il sistema completo in questo caso possiamo usare una traccia temporale molto semplice. È sufficiente definire il reset all'inizio e dei valori opportuni per TL e TH : rispettivamente 2_{10} e 3_{10} . Su SW si ottiene la nostra onda quadra, bassa per 3 cicli di clock e alta per 3. I valori piccoli di TL e TH sono stati scelti per ottenere un grafico ragionevolmente analizzabile.



Come si vede dalla simulazione, il primo tratto del segnale in uscita SW non è corretto (rimane alto per meno del voluto), ma il sistema va poi a regime e produce l'onda quadra prevista dall'impostazione degli ingressi.

8.3.8 Progetto di sistema per luminarie natalizie

Si progetti un sistema per luminarie natalizie: controlla tre strisce di luci natalizie, rosse, verdi e blu. Ogni striscia deve accendersi in sequenza per un tempo controllabile con un dato di 4 bit.

Soluzione

Anche in questo caso occorre leggere attentamente il testo e seguire i cinque passi definiti in precedenza:

- definizione del sistema come blocco funzionale, evidenziandone ingressi ed uscite;
- scelta dei componenti da usare nel datapath;
- progetto del datapath con i collegamenti del controllore con gli ingressi e le uscite, verso l'esterno ed il datapath stesso;

- (d) progetto del diagramma ASM della MSF che descrive il controllore;
- (e) simulazione temporale del sistema completo, scegliendo sequenze di ingresso opportune a dimostrarne il corretto funzionamento.

Passo (a)

In questa fase è molto importante comprendere molto bene il testo al fine di identificare gli ingressi e le uscite del sistema. Leggendo le specifiche si comprende che le tre strisce di luci colorate si accendono ognuna per un numero di cicli di clock diverso, programmabili tramite 4 bit di dato, indipendentemente per ciascun colore. In altre parole, il testo dichiara che la macchina deve avere 12 ingressi e 3 uscite.

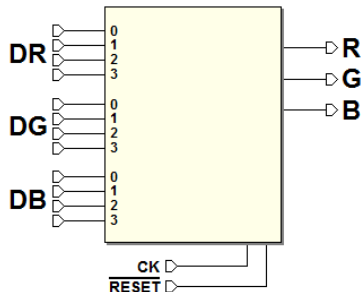
1. *Ingressi:*

- a) DR , composto da 4 bit che rappresentano, in binario puro, la durata di accensione della luce rossa;
- b) DB , 4 bit, come sopra, imposta la durata della luce blu;
- c) DG , 4 bit, analogamente, per la luce verde;

2. *Uscite:*

- a) R , un bit, se a 1 accende la luce rossa per circa DR cicli di CK ;
- b) B , un bit, se a 1 accende la luce blu per circa DB cicli di CK ;
- c) G , un bit, se a 1 accende la luce verde per circa DG cicli di CK .

Il sistema, come un unico blocco funzionale, è quindi questo:



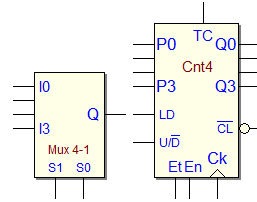
Passo (b)

Per la scelta dei componenti del datapath occorre comprendere quali sono le componentistiche necessarie alla realizzazione del sistema. Valgono le considerazioni fatte nell'esercizio precedente circa l'utilizzo proprio del controllore. Questo deve funzionare da gestore di componenti combinatori, aritmetici o di memorizzazione, e non deve sostituirsi ad essi.

Nel nostro caso, il sistema necessita di acquisire DR , DB e DG al fine di utilizzarli come misura del numero di cicli di clock in cui occorre mantenere attivi i segnali di uscita R , B , G . Potremmo scegliere di utilizzare tre contatori universali, come nell'esempio precedente. Useremo, invece, un solo contatore (a 4 bit, in questo caso), che verrà precaricato al valore corretto grazie a dei

selettori (multiplexer) collegati opportunamente tra gli ingressi DR , DB e DG del sistema e gli ingressi di precaricamento del contatore.

Osservando le specifiche quindi non servono altri componenti specifici se non un contatore “Cnt4” (già utilizzato a pag. 380) e quattro “Mux4-1” (vedi i *selettori* a pag. 50), i cui simboli sono riportati qui a destra.



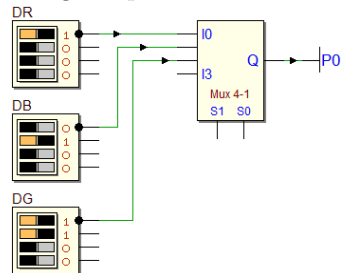
Passo (c)

Come descritto al passo precedente, useremo un contatore che il controllore precaricherà al momento opportuno, rispettivamente al numero letto su DR , DB o DG , grazie ai quattro selettori. Il contatore sarà sempre abilitato ($En = ET = 1$), e predisposto per il conteggio all'indietro. Il relativo TC deve essere letto dal controllore per valutare il tempo trascorso.

Adesso occorre valutare come collegare i vari ingressi e le varie uscite dei selettori e del contatore agli ingressi e alle uscite del sistema e al controllore. In particolare:

- Come descritto in precedenza, gli ingressi di precaricamento del contatore devono essere collegati ai tre gruppi di ingressi DR , DB e DG tramite i selettori, in modo da precaricare il valore corretto nel contatore.
- Per questo motivo possiamo pensare di utilizzare 4 selettori, uno per ogni bit di DR , DB e DG , come nell'esempio nella figura qui sotto:

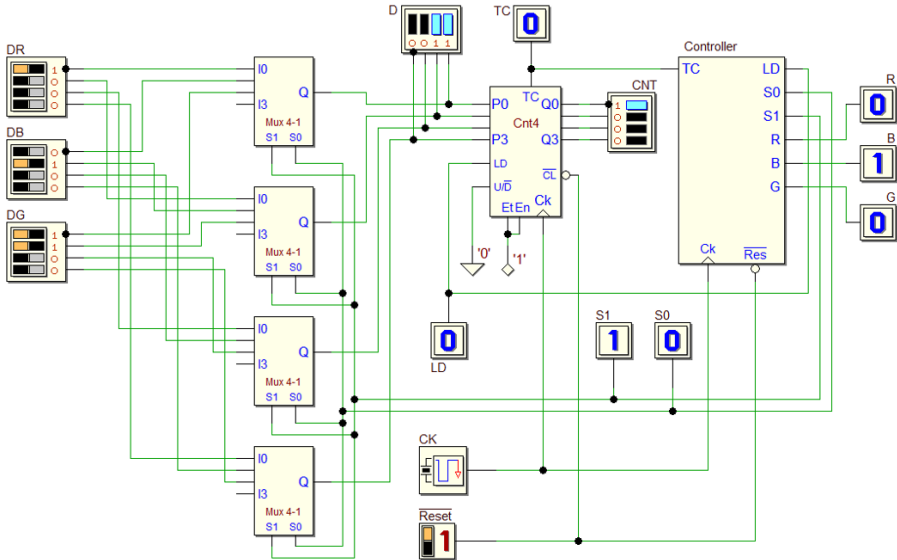
I bit in posizione 0 di DR , DB e DG verranno collegati rispettivamente agli ingressi $I0$, $I1$, $I2$ del primo selettore, come si vede in figura, e così via per gli altri. In questo modo le uscite Q dei 4 selettori ricopieranno DR , DB o DG , sulla base della impostazione dei loro ingressi di selezione $S1$ e $S0$ (rispettivamente a 00, 01, 10).



- Il contatore deve contare all'indietro e per questo il suo ingresso U/D sarà impostato a 0; come nell'esempio precedente, impostiamo $Et = En = 1$, in modo che l'unico controllo necessario sia LD .
- L'ingresso LD del contatore sarà comandato dal controllore in modo da precaricarlo al momento opportuno. Per esempio, se consideriamo DR , il controllore dovrà impostare le linee di selezione $S1$ e $S0$ a 00, e dare e poi disattivare il comando di LD : il contatore attiverà TC dopo $DR + 1$ cicli di CK (in quanto il conteggio include anche lo zero).
- CK e \overline{Reset} di sistema vanno collegati sia al contatore che al controllore, come discusso nell'esempio precedente.
- R , B e G possono essere direttamente generati dal controllore, negli stati in cui questo verifica l'arrivo del TC dal contatore.

- È a questo punto conveniente inserire nello schema tutti i vari segnali: LD e TC del contatore, le linee di selezione $S1$ e $S0$, i selettori, e quindi le uscite R , B e G .

Lo schema finale del progetto del datapath con i collegamenti al controllore, inclusi i collegamenti agli ingressi ed uscite verso l'esterno e l'interno del datapath stesso è riportato qui sotto:



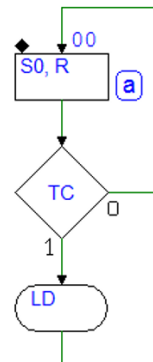
Passo (d)

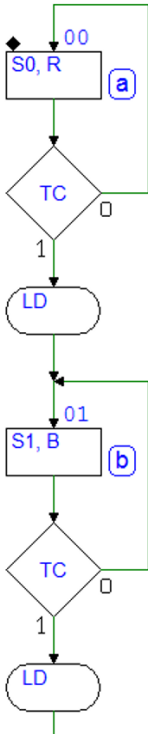
All'arrivo del \overline{Reset} il contatore va a zero e, a meno che non riceva il comando LD , potrebbe cominciare a contare. TC è attivo, in quanto il contatore è impostato all'indietro ed è a zero.

Come si osserva nella figura qui a destra, nello stato al reset (a) scegliamo di attivare l'uscita R , e impostare $S1S0 = 01$ (asserendo solo $S0$ nel blocco di stato), al fine di preparare il numero DB (la durata della successiva luce B) in ingresso al contatore, tramite i selettori.

Nello stato (a) dovremmo permanere un certo tempo, in attesa che il contatore si azzeri, attivando TC . Ma abbiamo visto che TC è già attivo, al reset, per cui da (a) usciamo subito, al primo fronte attivo di clock dopo il \overline{Reset} .

Mentre siamo ancora nello stato (a), impostiamo LD come uscita condizionata: è attiva se $TC = 1$, ossia se il conteggio è arrivato a zero, oppure se lo è già (al reset).





Sullo stesso fronte attivo di CK con cui il controllore passa nello stato successivo (b)(vedi figura a sinistra), il contatore carica il nuovo valore impostato, ossia il numero proveniente dal gruppo di ingresso selezionato (DB , come sopra precisato).

Nello stato (b), l'uscita B è attiva, e impostiamo i selettori in modo che portino il numero DG agli ingressi di precaricamento del contatore, rimanendo in attesa dell'arrivo di TC .

Possiamo continuare quindi il ragionamento iterando anche per l'uscita G (ricordando che dopo la G occorre tornare ad attivare la R) e ottenendo la soluzione finale (figura a destra), che si richiude sullo stato (a).

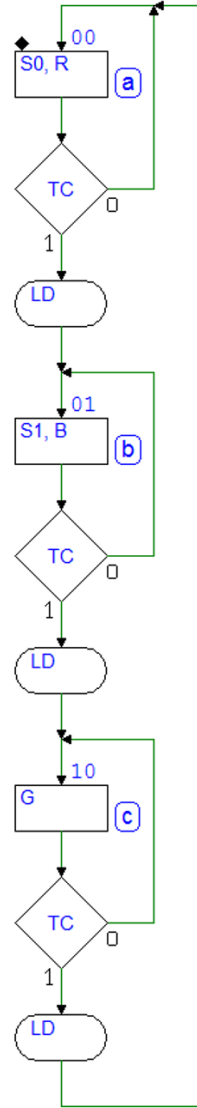
Si noti che al reset la macchina terrà alta l'uscita R per un solo ciclo di CK ma, a partire dalla successiva ripetizione del ciclo degli stati, la sua durata risulterà regolarmente impostata da DR , caricato nel contatore nello stato (c), grazie alla attivazione di TC .

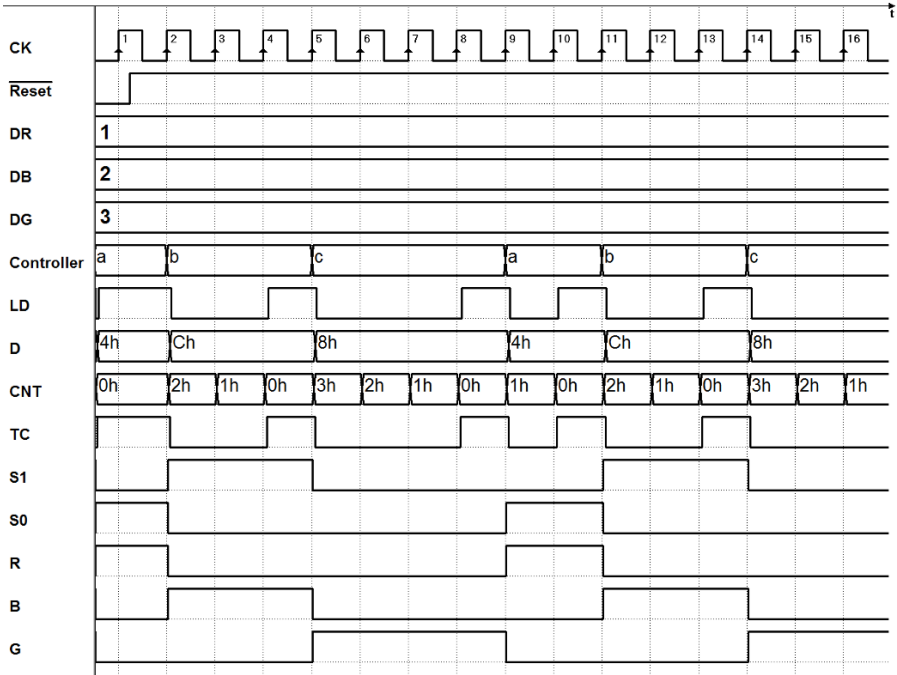
Passo (e)

Per simulare il sistema completo, impostiamo gli ingressi $DR = 1_{10}$, $DB = 2_{10}$, $DG = 3_{10}$ e la usuale sequenza iniziale del reset.

Si ottiene la simulazione temporale visibile nella figura successiva. Si noti come le uscite relative alle luci R , B e G si attivano in sequenza ciclica, rispettivamente per 2_{10} , 3_{10} e 4_{10} cicli del clock CK .

I valori di DR , DB , e DG sono stati scelti per ottenere un grafico ragionevolmente analizzabile. Come si vede dalla simulazione temporale, verificiamo che, all'inizio, la durata di R non è corretta (R rimane alto un solo ciclo di clock) ma successivamente il sistema produce le temporizzazioni previste dall'impostazione degli ingressi DR , DB , e DG .



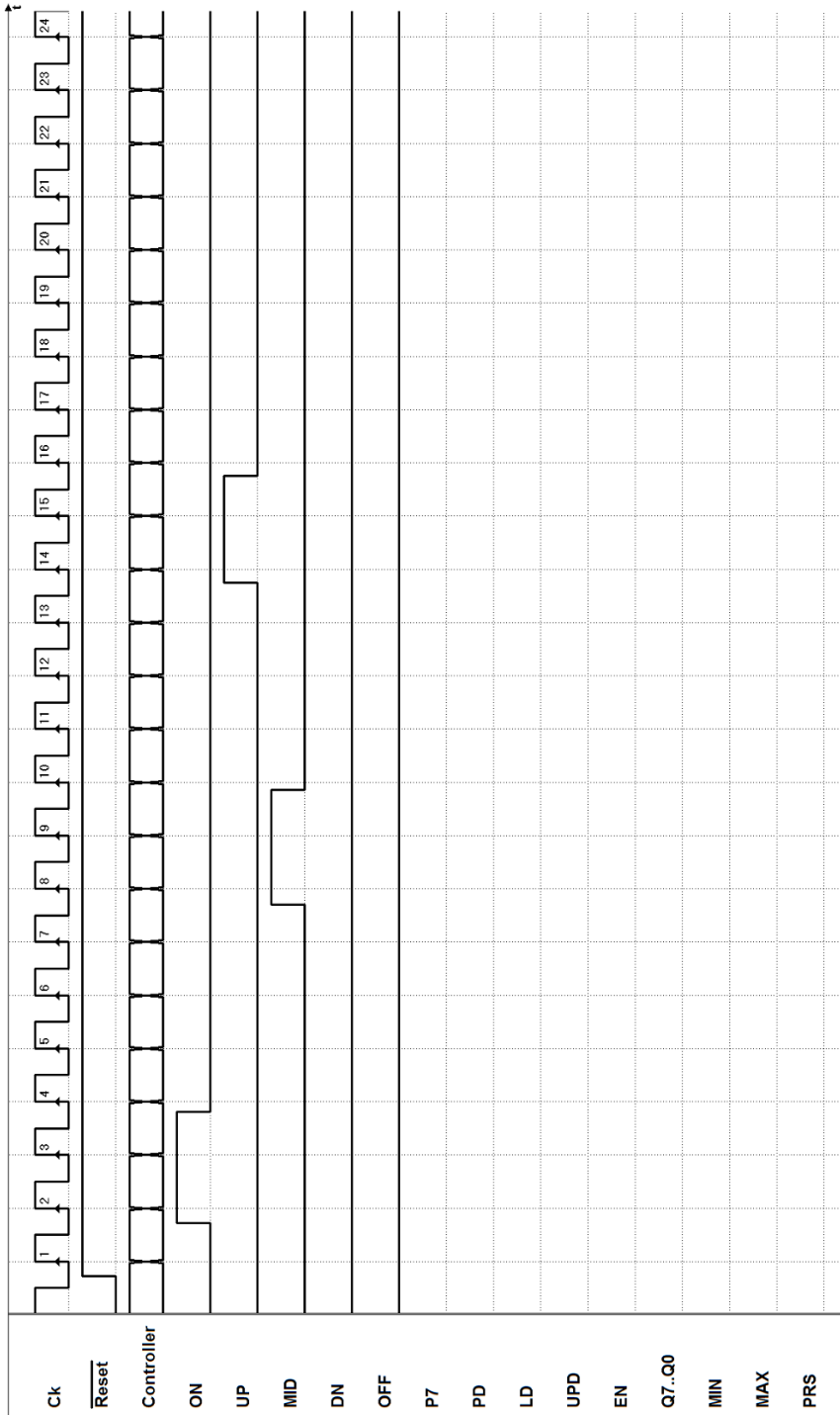


8.4 Esercizi

8.4.1 Progetto di controllore, con datapath assegnato

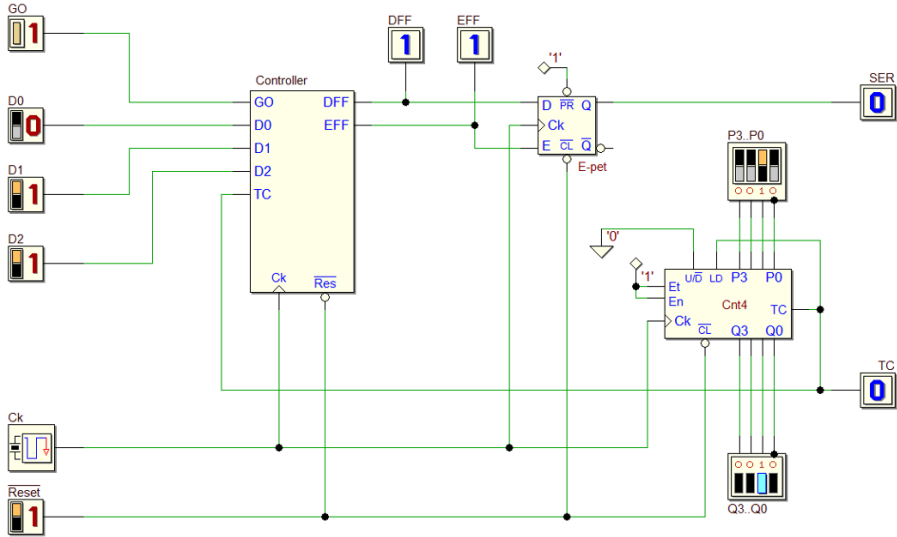
Per ognuno degli esercizi seguenti di progetto di sistema digitale, di cui è fornito lo schema complessivo *controllore - datapath*, si chiede di definire il diagramma ASM del controllore sulla base delle specifiche date (nella pagina sulla sinistra), e quindi di completare il diagramma temporale proposto (nella pagina a fronte), includendo l'indicazione dello stato in cui si trova la MSF, in ogni ciclo di clock. Non è richiesto di effettuare la sintesi della MSF.

Sono disponibili, sul sito del simulatore *Deeds*, per ogni esercizio: una traccia del file della MSF da progettare, dove sono già definite le variabili di stato, gli ingressi e le uscite; un file PDF con il tracciato temporale proposto, da completare su carta senza usare il simulatore; lo schema della rete, predisposto per inserirvi la vostra soluzione di MSF. Lo schema completo sarà utile per verificare il comportamento del sistema, mediante la simulazione temporale.



Esercizio 2

Si consideri il sistema digitale rappresentato in figura, composto dal controllore, dal contatore “Cnt4” e da un flip-flop E-PET:



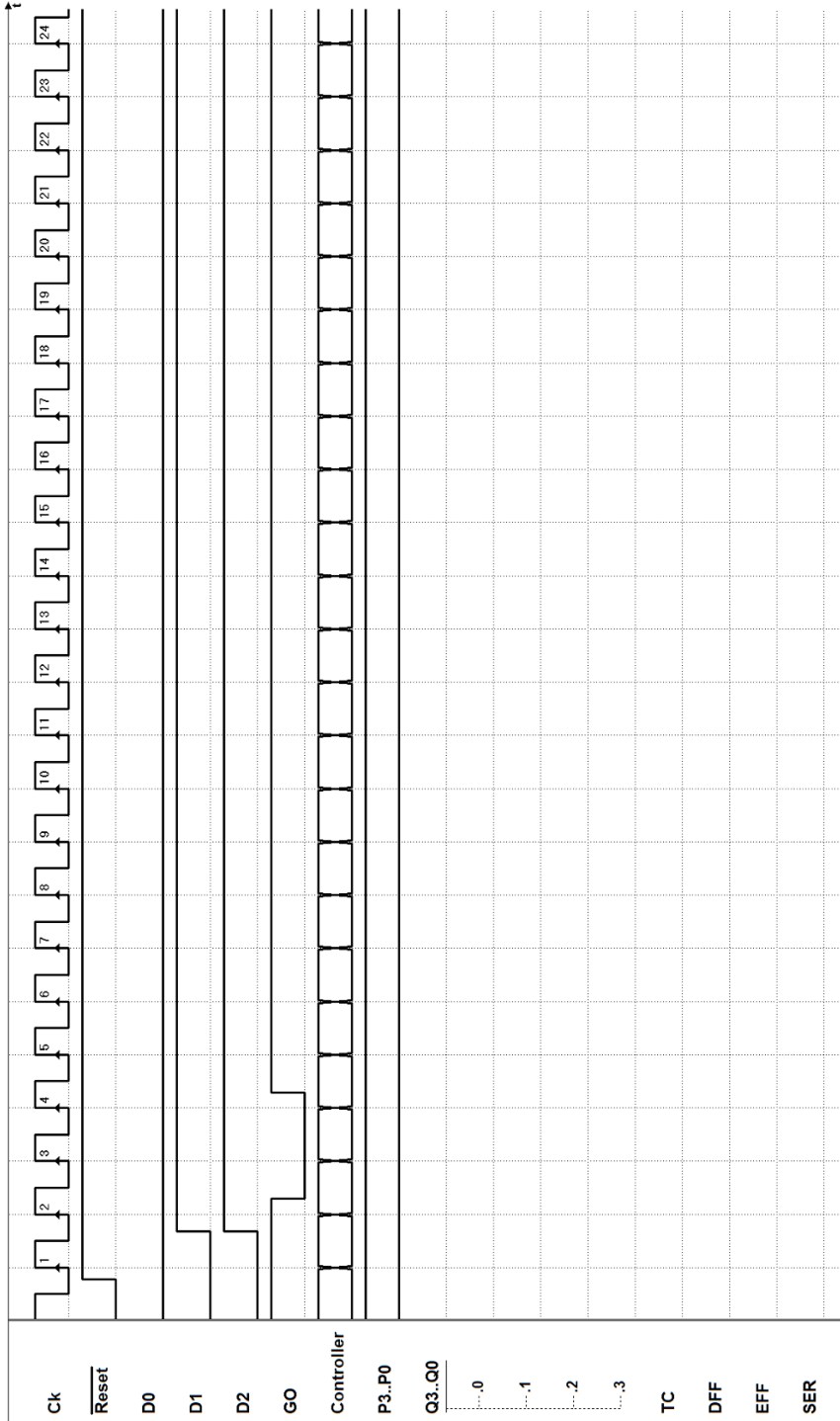
Compito del sistema (un *trasmettitore seriale*) è di generare, sull'uscita *SER*, tramite il flip-flop E-PET, un pacchetto di 5 bit, composto da un bit di start a 1, tre bit di dato (nell'ordine: *D0*, *D1* e *D2*), e un bit di stop a 0.

La durata N di ciascun bit del pacchetto (il tempo di bit) è un multiplo del periodo del clock CK , ed è funzione del numero P da impostare agli ingressi $P3..P0$ del contatore.

Ai fini dell'analisi temporale del sistema, si assegni $P = 2$.

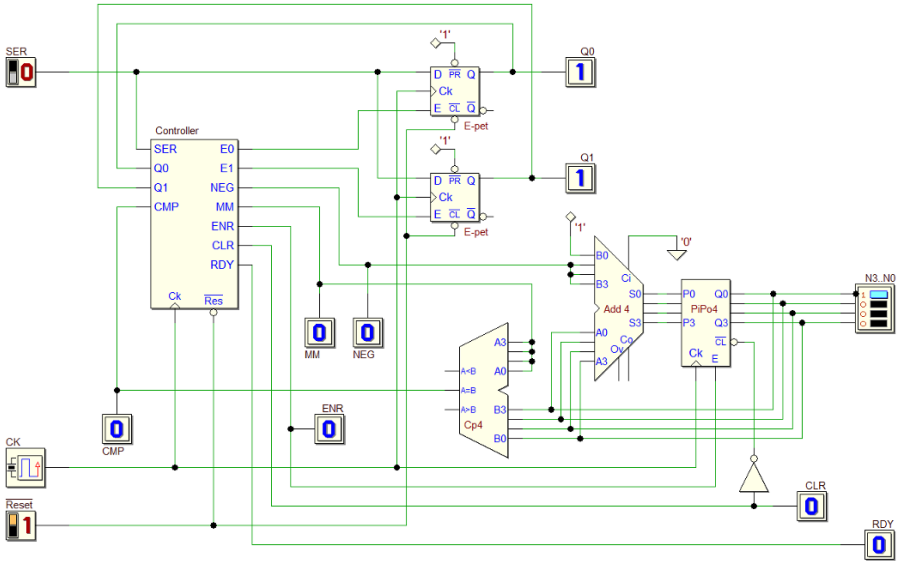
I dati da trasmettere, $D0$, $D1$ e $D2$, sono disponibili agli ingressi del controllore. Un fronte di discesa sull'ingresso GO fa partire la generazione del pacchetto.

Si precisi la relazione risultante tra N e il numero P (dipende dalla propria particolare soluzione).



Esercizio 3

Si consideri il sistema digitale rappresentato in figura, composto dal controllore che riceve dati da una linea seriale e gestisce una rete composta da due flip-flop E-PET, un registro e alcuni circuiti aritmetici.



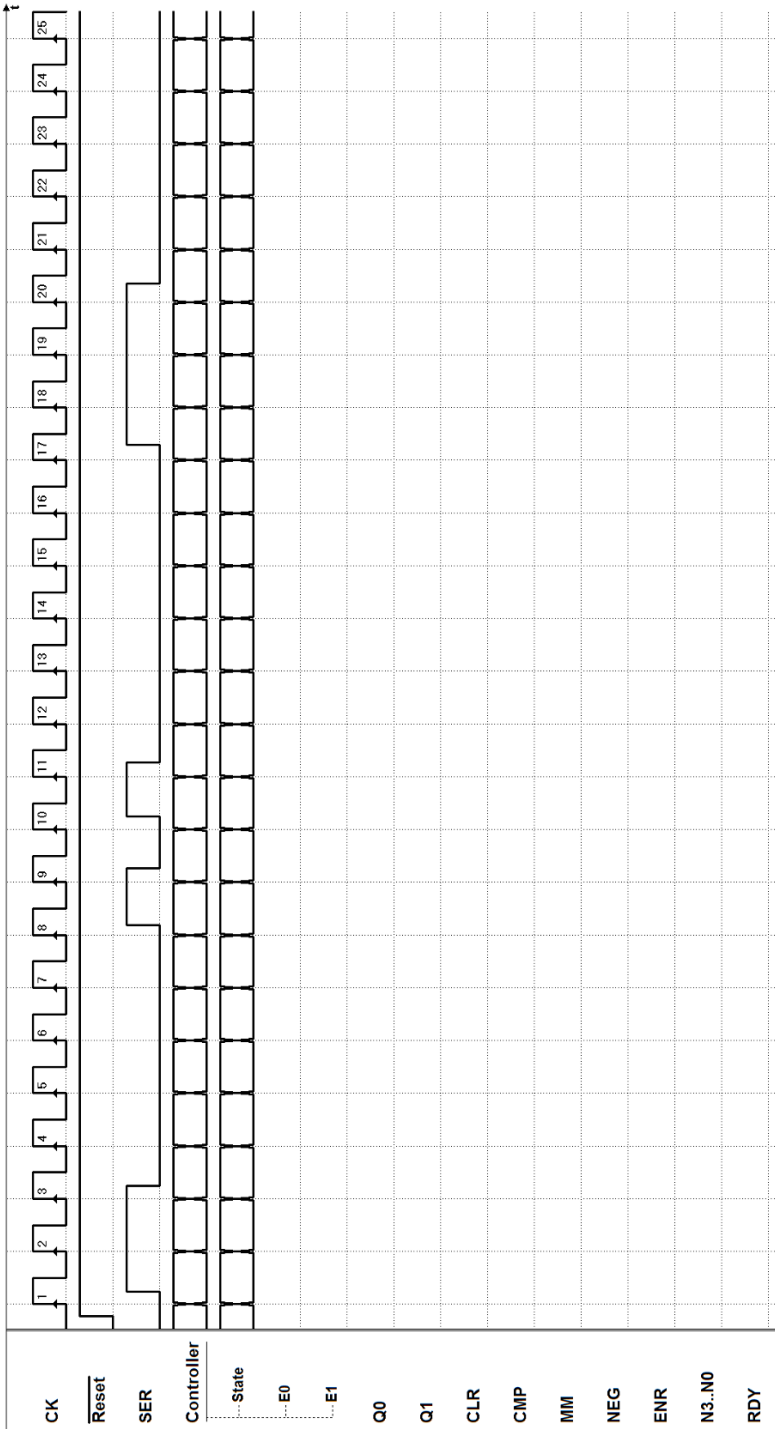
Il controllore riceve, su *SER*, un pacchetto di 4 bit, composto da un bit di start a 1, due bit di dato (nell'ordine *D0* e *D1*), e un bit di stop a 0. La durata del tempo di bit è pari al periodo di *CK*; i due bit di dato codificano un'operazione che il sistema esegue sulle uscite *N3..N0*.

Se è ricevuto un bit di stop errato (= 1), non si eseguono operazioni e si aspetta che *SER* torni a zero, prima di attendere il successivo pacchetto. Alla ricezione di un pacchetto valido (bit di stop = 0), a seconda del valore ricevuto di *D1* e *D0* il sistema esegue le seguenti operazioni:

<i>D1</i>	<i>D0</i>	Operazione
0	0	Nessuna (NOP)
0	1	Azzerata l'uscita <i>N3..N0</i>
1	0	Incrementa di uno l'uscita <i>N3..N0</i>
1	1	Decrementa di uno l'uscita <i>N3..N0</i>

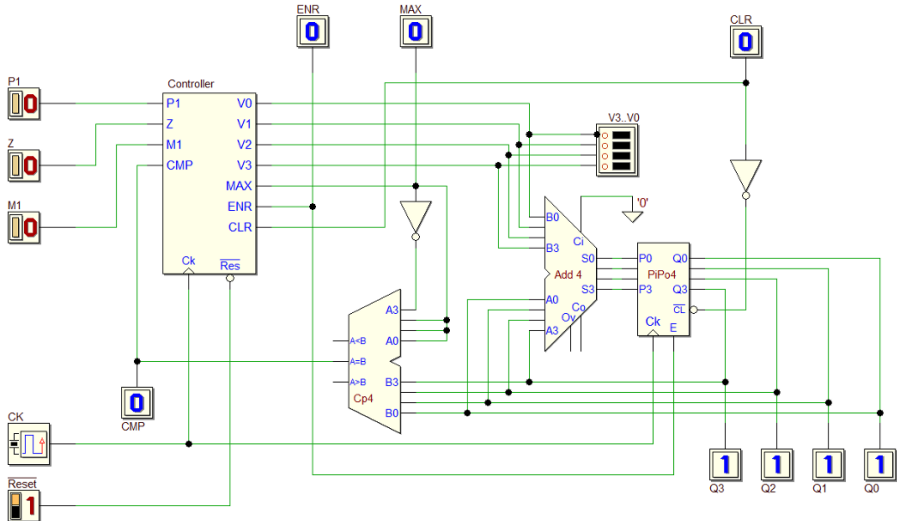
L'uscita *N3..N0* non è incrementata quando ha raggiunto il valore massimo, né decrementata quando ha raggiunto il minimo. L'uscita *RDY* è attivata per un ciclo di clock alla ricezione dei comandi, tranne la NOP.

Si noti che la MSF memorizza i bit ricevuti *D1* e *D0* rispettivamente nei flip-flop *Q1* e *Q0*, in modo da disporre dei valori ricevuti anche dopo il controllo del bit di stop; l'uscita *NEG* della MSF permette di impostare all'ingresso del sommatore *Add4* due diversi valori, mentre l'uscita *MM* consente al comparatore *Cp4* di confrontare il numero *N3..N0* con due diversi valori.



Esercizio 4

Si consideri il sistema digitale rappresentato in figura, composto dal controllore, da un registro e alcuni circuiti aritmetici:



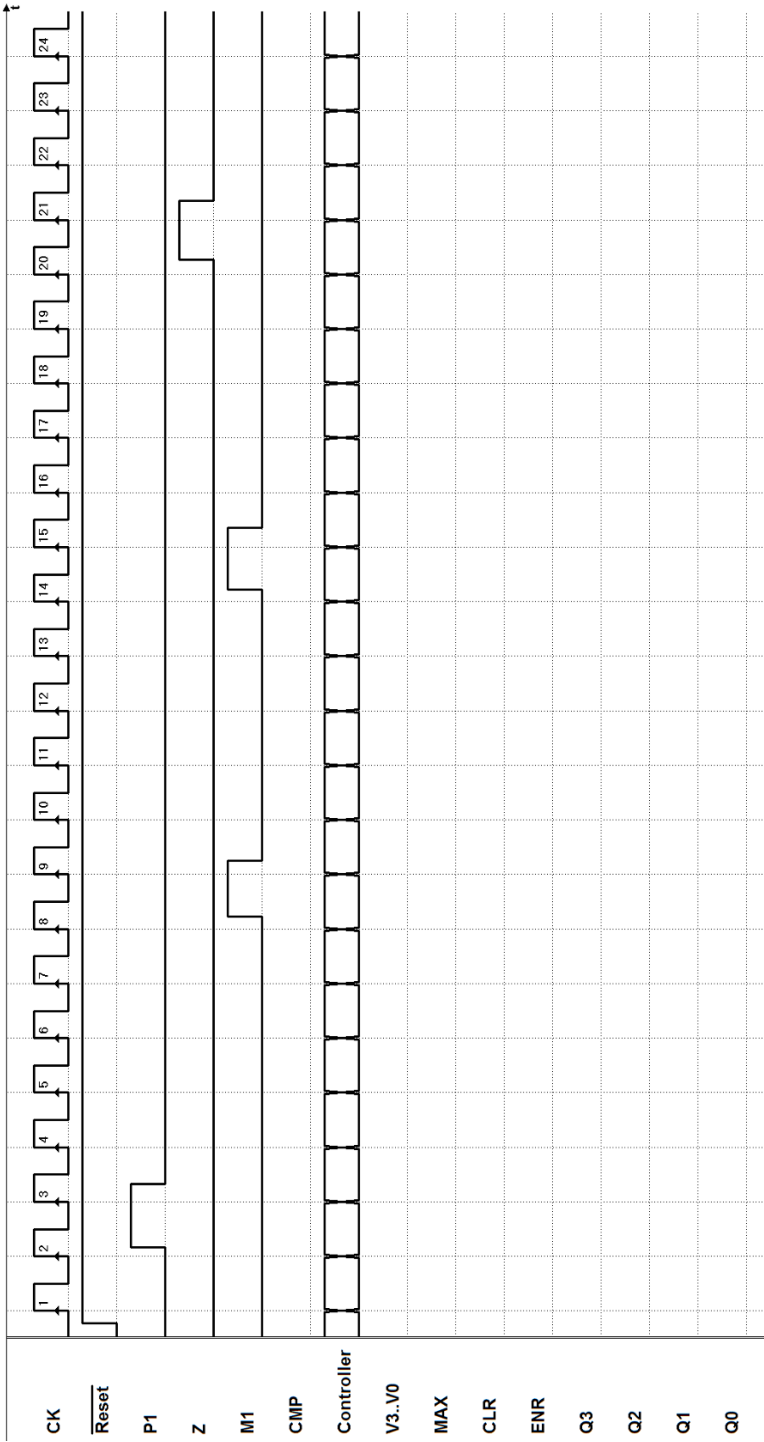
Il sistema realizza un generatore di numero binario, comandato dai tre pulsanti $P1$, Z e $M1$. L'uscita $Q3..Q0$ è prelevata dal registro "PiPo8". Il numero binario in uscita è codificato con segno, in codice complemento a due.

I pulsanti sono a 0 a riposo; a 1 per tutto il tempo in cui sono premuti. I pulsanti si considerano ideali e privi di rimbalzi meccanici. La pressione ed il successivo rilascio di ciascun pulsante determina il comportamento del sistema. La funzione dei pulsanti è la seguente:

- $P1$: produce l'incremento di una unità del numero $Q3..Q0$;
- Z : determina l'azzeramento del numero $Q3..Q0$;
- $M1$: produce il decremento di una unità del numero $Q3..Q0$;

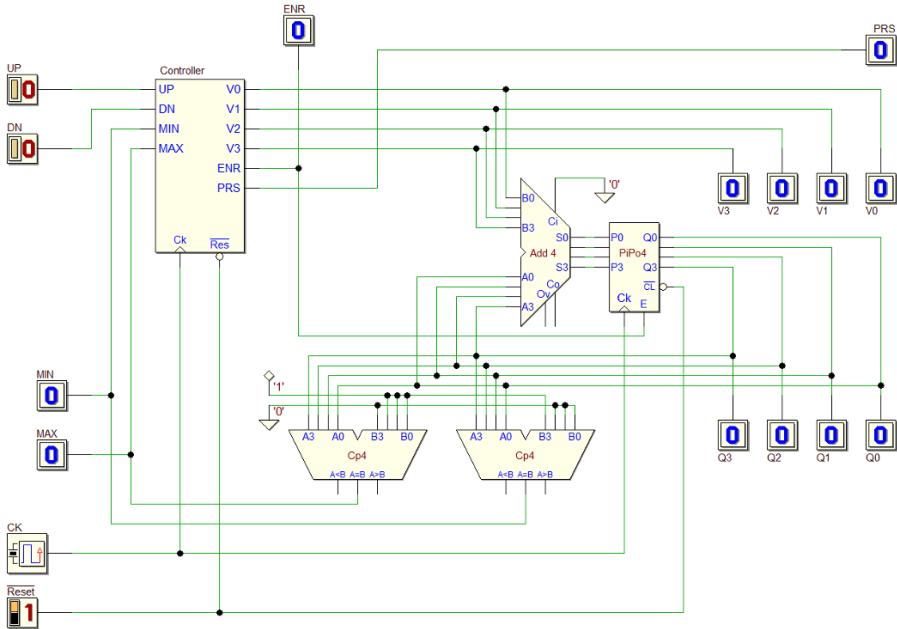
Il numero $Q3..Q0$ non deve essere incrementato o decrementato se ha già raggiunto, rispettivamente, i valori massimo (positivo) e minimo (negativo). All'inizializzazione del sistema il numero $Q3..Q0$ deve essere azzerato.

Si noti che le uscite $V3..V0$ della MSF permettono di impostare all'ingresso del sommatore "Add4" i valori da sommare, mentre l'uscita MAX consente di impostare il comparatore "Cp4" in modo da confrontare il numero $Q3..Q0$ con i valori massimo e minimo.



Esercizio 5

Si consideri il sistema digitale rappresentato in figura, composto dal controllore, da un registro e alcuni circuiti aritmetici:



Il sistema realizza un *generatore di numero binario*, comandato dai due pulsanti *UP* e *DN*. L'uscita $Q3..Q0$ è prelevata dal registro "PiPo4". Il numero binario in uscita è codificato con segno, in codice complemento a due.

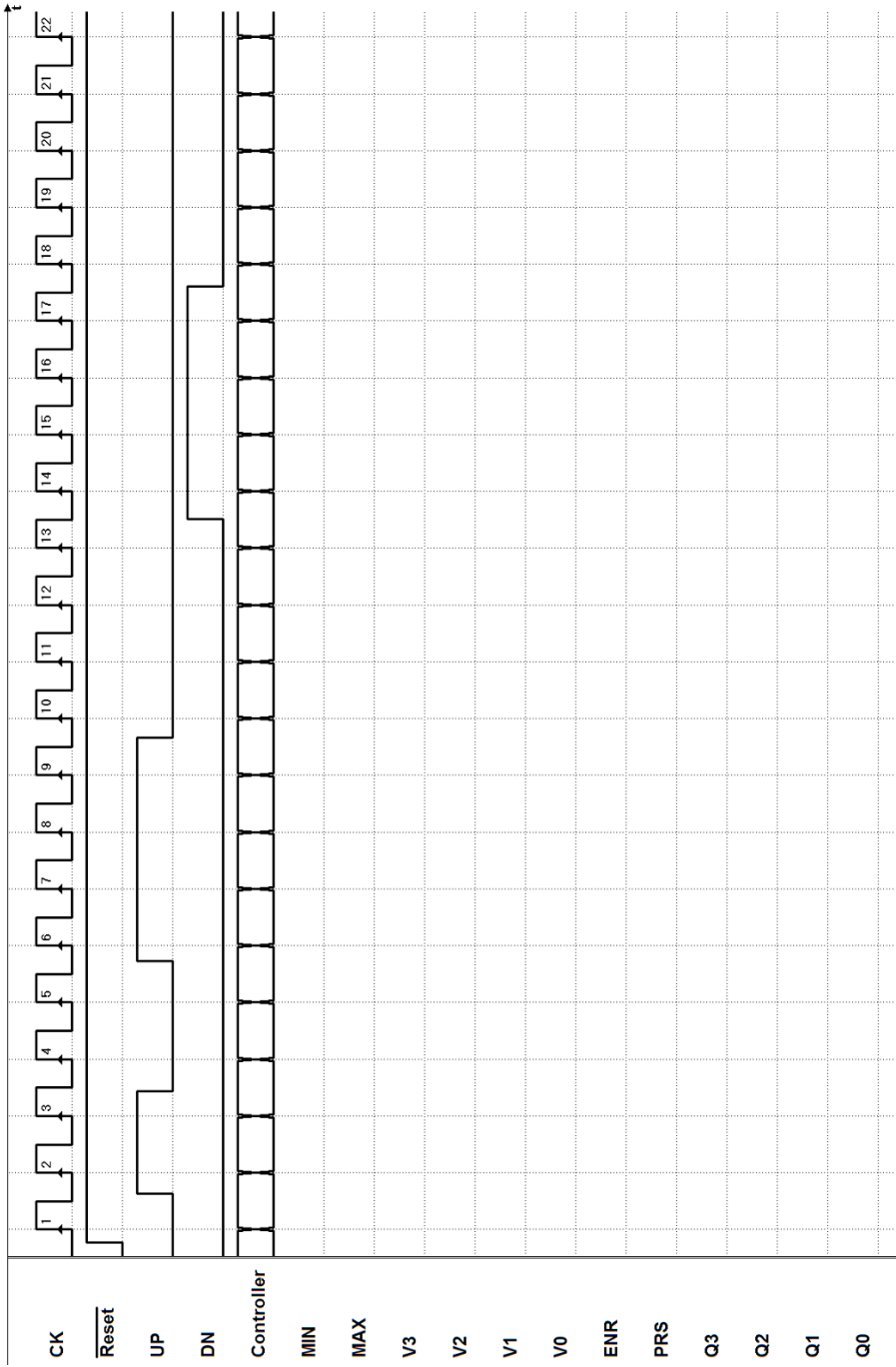
I pulsanti sono a 0 a riposo; a 1 per tutto il tempo in cui sono premuti. I pulsanti si considerano ideali e privi di rimbalzi meccanici. Alla pressione ed il successivo rilascio dei pulsanti il sistema esegue le seguenti funzioni:

- Pulsante *UP* : esegue l'incremento di una unità del numero $Q3..Q0$;
- Pulsante *DN* : esegue il decremento di una unità del numero $Q3..Q0$.

Il rilascio dei pulsanti è ignorato se premuti per un tempo < 3 cicli di clock.

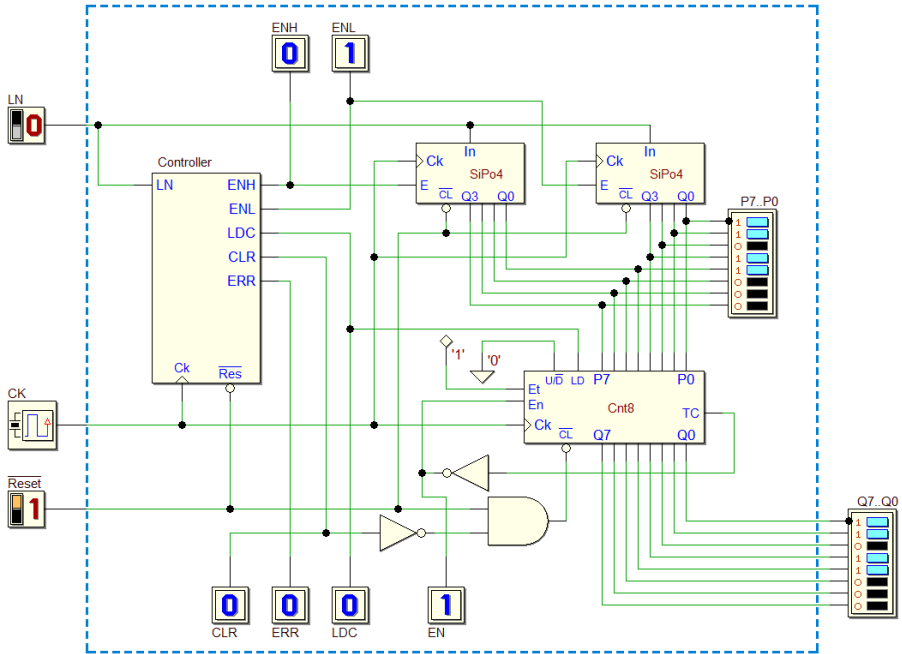
I due comparatori ricevono sugli ingressi $A3..A0$ il valore dell'uscita $Q3..Q0$ e lo confrontano con i valori massimo e minimo impostati su $B3..B0$: il numero $Q3..Q0$ non deve essere incrementato o decrementato se ha già raggiunto, rispettivamente, il valore massimo (positivo) o minimo (negativo).

L'uscita *PRS* è attivata dal sistema per tutto il tempo in cui uno qualunque dei pulsanti è premuto. Si faccia inoltre l'ipotesi che i due pulsanti non possano mai essere premuti nello stesso momento.

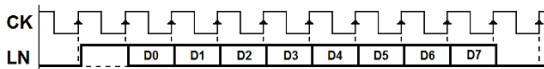


Esercizio 6

Si consideri il sistema digitale rappresentato in figura, composto dal controllore, da due registri a scorrimento “SiPo4”, un contatore a 8 bit “Cnt8” e alcune porte logiche.

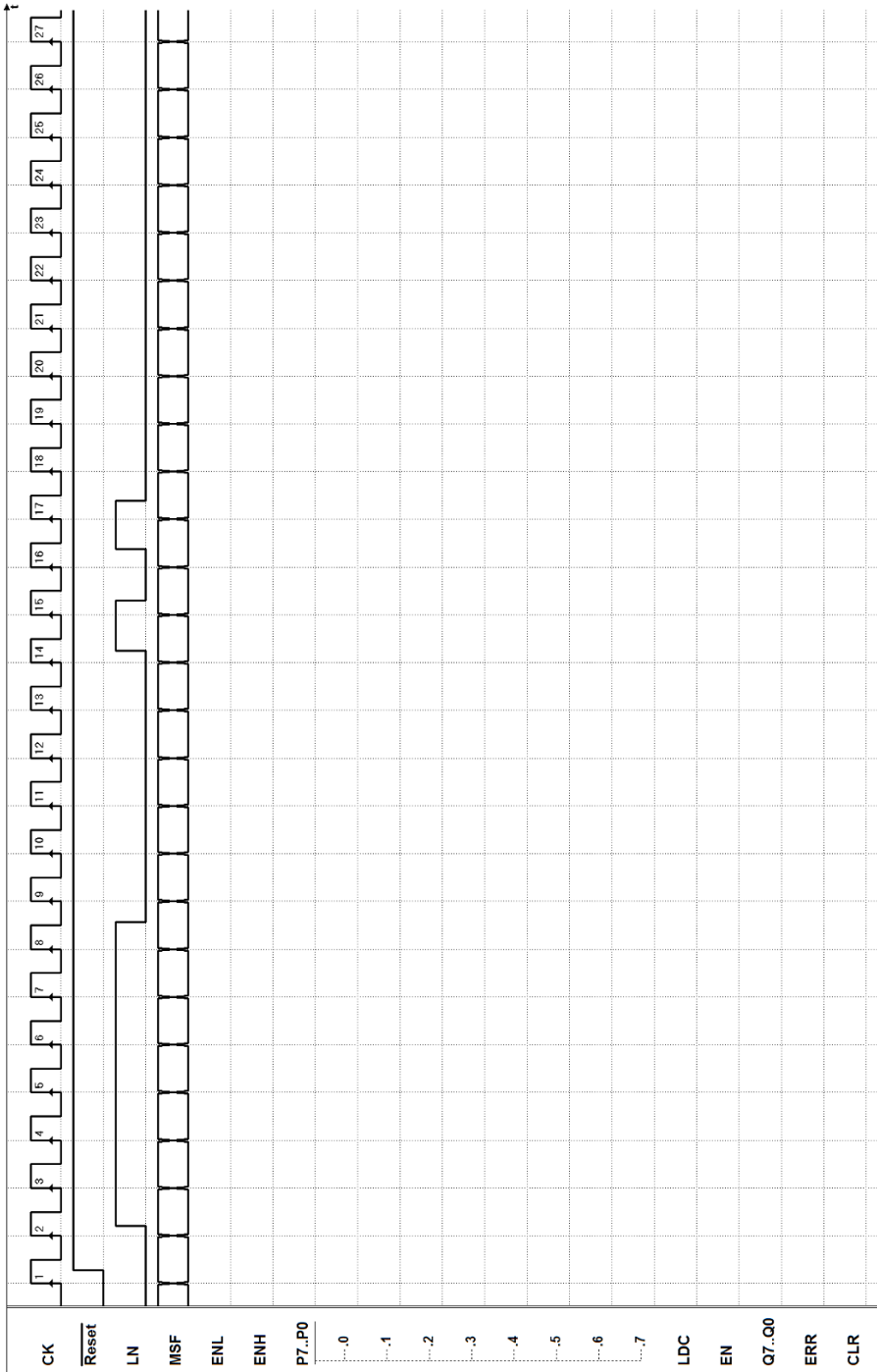


Il sistema riceve sull'ingresso LN un segnale seriale standard (vedi figura), composto da un bit di Start, 8 bit di dato e un bit di Stop:



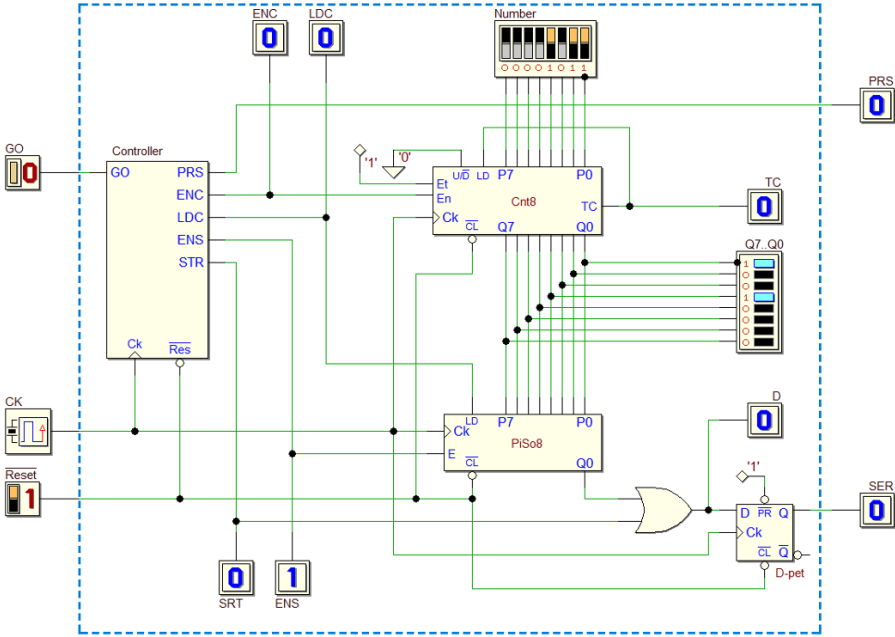
Il controllore comanda l'inserimento dei dati seriali nei due registri con l'ordine che si desume dallo schema (la parte bassa $D3..D0$ sul registro a destra nello schema, la parte alta $D7..D4$ sul registro a sinistra).

Controlla il bit di stop alla fine della sequenza seriale e, se corretto, carica il contatore con il byte ricevuto, tramite le linee $P7..P0$. Invece, se il bit di stop non è valido, il contatore è azzerato e il controllore segnala ERR , mantenendo la segnalazione finché LN non ritorna a zero.

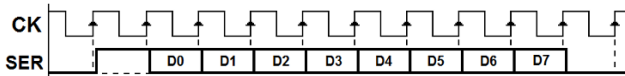


Esercizio 7

Si consideri il sistema digitale rappresentato in figura, composto dal controllore, da un contatore a 8 bit “Cnt8”, un registro a scorrimento “PiSo8”, un flip-flop D-PET e alcune porte logiche.



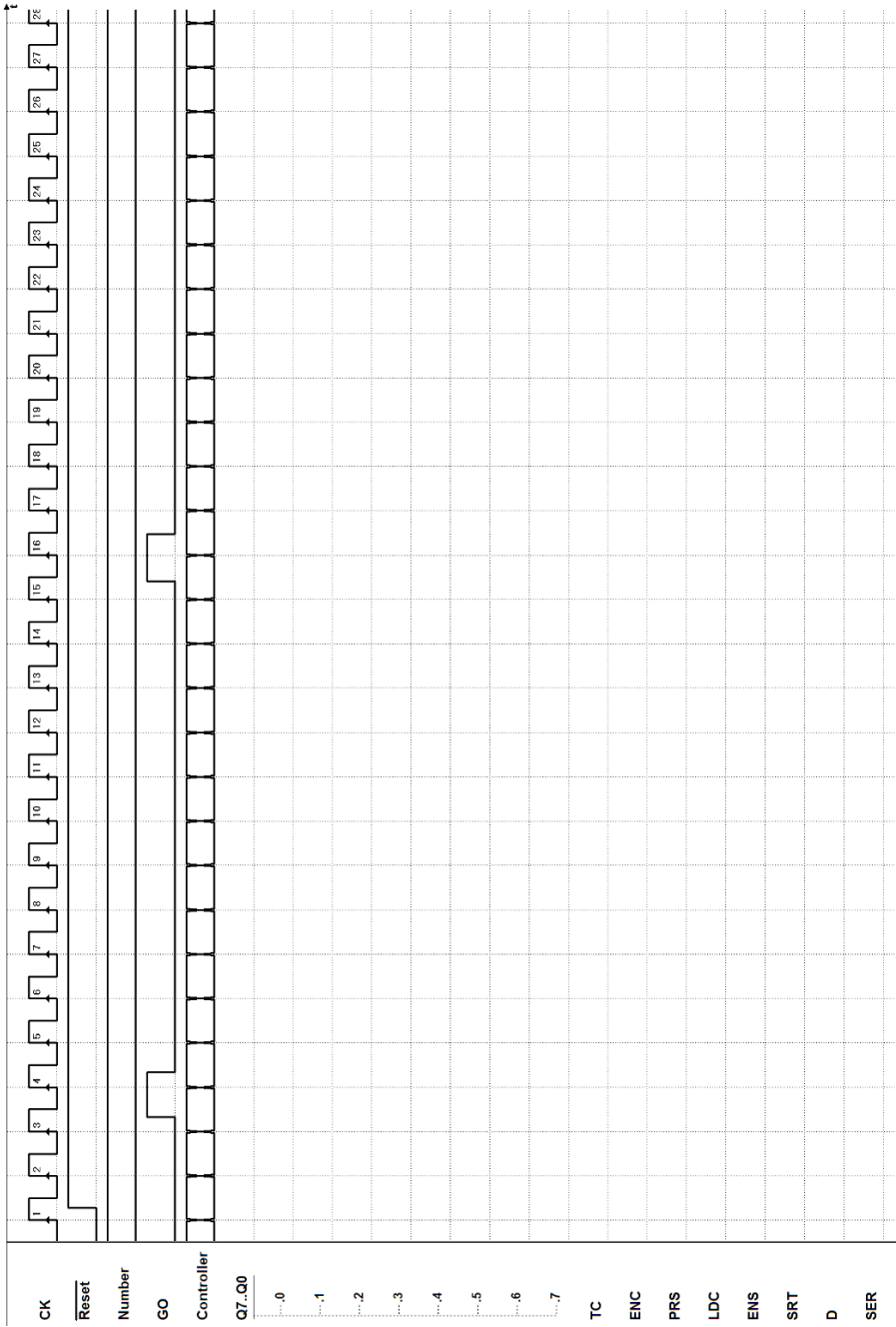
Al rilascio del pulsante *GO*, il sistema trasmette su *SER* un segnale seriale (vedi figura), composto da un bit di start, 8 bit di dato e un bit di stop:



Gli 8 bit di dato da trasmettere sono prelevati dalle uscite del contatore. A riposo, quando il pulsante non è premuto ($GO = 0$), il contatore è abilitato, tramite la linea *ENC*; per tutto il resto del tempo, il conteggio è disabilitato. Alla pressione del pulsante ($GO = 1$), la MSF attiva la linea *PRS*.

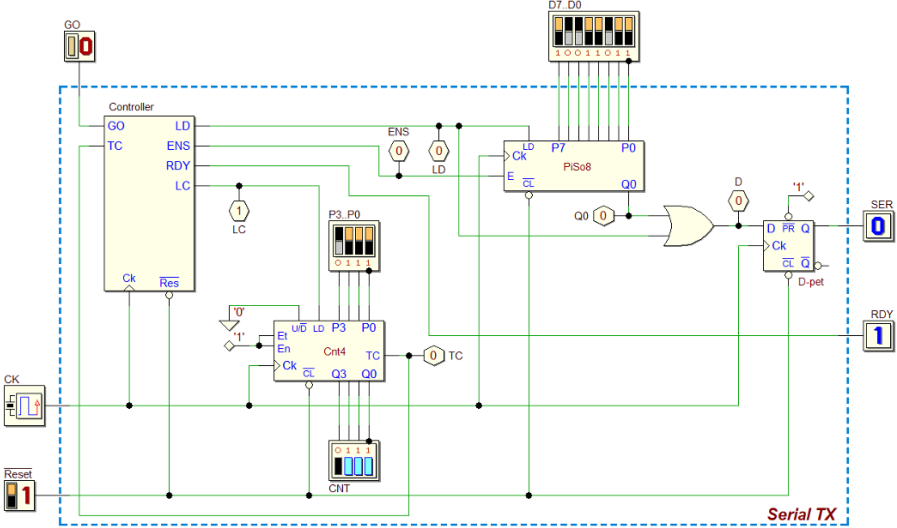
Al rilascio del pulsante, il controllore inizia le operazioni per la trasmissione, caricando le uscite del contatore nel registro a scorrimento, tramite la linea *LDC*. Si noti che la linea *STR* permette di generare il bit di start per un ciclo di clock, e che l'uscita seriale *SER* è generata tramite il flip-flop.

La serializzazione è comandata dal controllore tramite la linea *ENS* che abilita il registro a scorrimento; terminata la generazione di *SER*, il sistema torna nuovamente a controllare il pulsante *GO*. Ai fini della analisi temporale, si ipotizzi che *Number* (vedi schema) sia pari a *0Bh*.

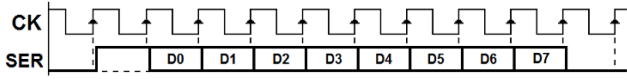


Esercizio 8

Si consideri il sistema digitale rappresentato in figura, composto dal controllore, da un contatore e un registro a scorrimento.



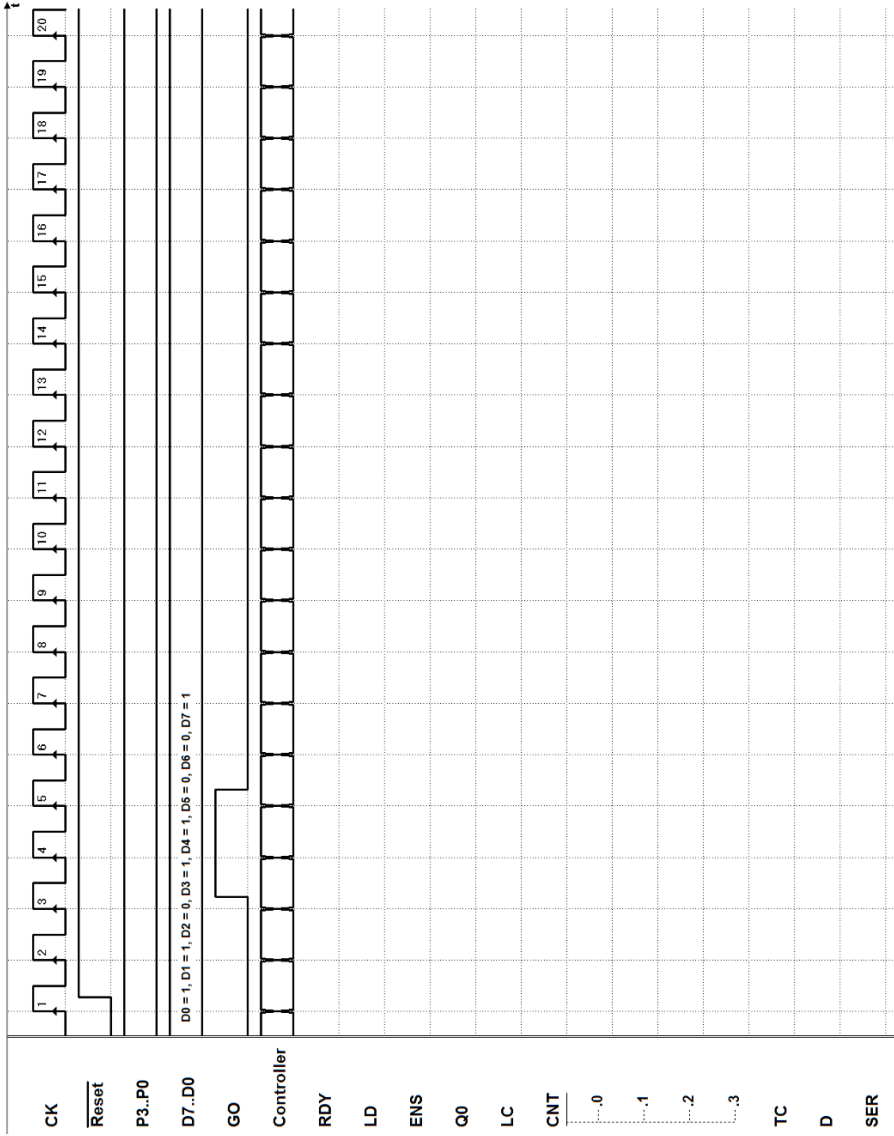
Il sistema realizza un trasmettitore seriale sincrono, con tempo di bit pari al periodo del clock. La trasmissione è avviata sul fronte di salita dell'ingresso *GO*. Il pacchetto trasmesso su *SER* è composto da un bit di start a 1, gli otto bit di dato *D0..D7* (nell'ordine), ed infine un bit di stop a 0:

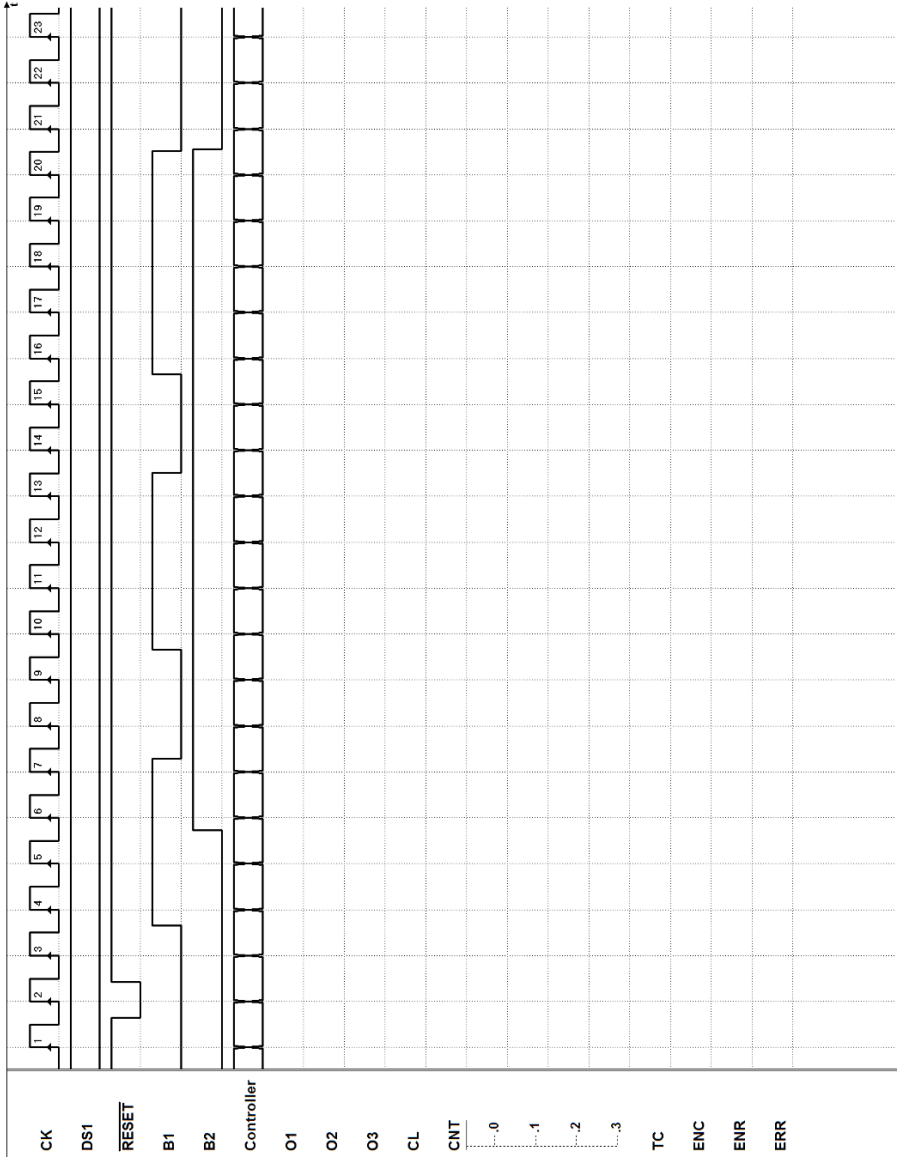


L'uscita *RDY* (ready) viene attivata quando il sistema non trasmette ed è in attesa del comando di avvio.

Il contatore è utilizzato dal controllore per contare il numero di bit trasmessi, a fine di terminare le operazioni dopo la trasmissione del pacchetto.

Si definisca chiaramente il numero *P3..P0* richiesto dal proprio particolare progetto (in figura, 0111_2 è solo indicativo).





8.4.2 Progetto di sistema completo (controllore e datapath)

Ricordiamo che i passi da affrontare per procedere in un progetto di questo tipo sono i seguenti:

- (a) definizione del sistema come blocco funzionale, evidenziandone ingressi ed uscite;
- (b) scelta dei componenti da usare nel datapath;
- (c) progetto del datapath con i collegamenti del controllore con gli ingressi e le uscite, verso l'esterno ed il datapath stesso;
- (d) progetto del diagramma ASM della MSF che descrive il controllore;
- (e) simulazione temporale del sistema completo, scegliendo sequenze di ingresso opportune a dimostrarne il corretto funzionamento.

Esercizio 1

Progettare un *trasmettitore seriale* della misura della *durata di un impulso*, utilizzando la struttura *controllore - datapath*.

Il sistema aspetta che la linea *IM* vada a 1 e quindi inizia a contare quanti cicli di clock il segnale *IM* rimane a 1.

Al primo fronte di clock in cui *IM* torna a zero il sistema spedisce sul canale *PKG* un pacchetto composto da 6 bit: un *bit di start* a 1, 4 *bit di dato* che rappresentano la durata dell'impulso da 0 a 15 in binario puro (è ammesso un errore di più o meno 1), un *bit di stop* a 0.

Se l'impulso dura più di 15 cicli di clock la macchina attiva l'uscita *ERR* fino a che *IM* ritorna a zero, non spedisce nessun pacchetto e poi aspetta l'impulso successivo.

(Suggerimenti a pag. 437)

Esercizio 2

Progettare un *timer per uso di cucina*, utilizzando la struttura *controllore - datapath*.

Al rilascio del pulsante *ST* (a 1 quando premuto) il dispositivo attende per un numero di cicli di clock impostato dall'esterno sugli ingressi di predisposizione *P7..P0* (8 bit), e quindi attiva per *tre cicli* un'uscita acustica *BEP*.

Il clock ha la frequenza di 1 Hz e il tempo deve essere controllabile tra 2 e 256 secondi. È consentito che il tempo effettivo differisca da quello impostato al massimo di un secondo.

(Suggerimenti a pag. 438)

Esercizio 3

Progettare un *convertitore seriale-parallelo*, utilizzando la struttura *controllore - datapath*.

Il sistema dispone di un ingresso *SER* e delle uscite *D0..D5* e *RDY*. Attende il dato seriale nel formato $\{1, D0, D1 .. D5, 0\}$ e lo trasferisce in formato parallelo sulle sei uscite *D0..D5*. Se la verifica del *bit di stop* ha esito positivo attiva *RDY* e lo mantiene fino alla ricezione di un nuovo dato seriale.

Se il *bit di stop* non è corretto il sistema ritorna semplicemente in attesa di un nuovo dato seriale.

(Suggerimenti a pag. 439)

Esercizio 4

Progettare un *ripetitore seriale-seriale*, utilizzando la struttura *controllore - datapath*.

Il sistema riceve sull'unico ingresso *IN* un dato seriale nel formato $\{1, B0, B1, P, 0\}$, dove *P* rappresenta la *parità* (funzione EXOR) di *B0* e *B1*.

Il sistema acquisisce *B0* e *B1* in due flip-flop JK-PET e controlla tramite una porta EXOR che la parità di *B0* e *B1* salvati nei flip-flop corrisponda a quella trasmessa (il bit *P* del dato seriale): in tal caso, se il bit di stop è corretto, trasmette sull'uscita *OUT* il pacchetto $\{1, B0, B1, 0\}$.

Se la verifica della parità ha dato un esito negativo o il bit di stop non è corretto, non c'è trasmissione ed il sistema ritorna in attesa di un altro pacchetto in ingresso.

(Suggerimenti a pag. 440)

Esercizio 5

Progettare un *convertitore parallelo-seriale*, utilizzando la struttura *controllore - datapath*.

Il sistema dispone di 5 ingressi (*GO* e *D0..D3*) e 3 uscite (*SER*, *BSY*, *RDY*). All'arrivo di *GO* il sistema genera un pacchetto seriale composto in questo modo: $\{1, D0, D1, D2, D3, 0\}$.

I segnali *D0..D3* sono garantiti al valore corretto solo alla ricezione di *GO*; *GO* dura un periodo di clock.

Il sistema attiva *BSY* durante la trasmissione del pacchetto e genera un impulso della durata di un ciclo di clock su *RDY* alla fine della trasmissione del pacchetto.

(Suggerimenti a pag. 441)

Esercizio 6

Si progetti un sistema digitale che funzioni come un *accumulatore*, generando in uscita su 4 bit la somma del numero esistente e di quello presentato in ingresso.

Il sistema dispone di 5 ingressi:

- D2..D0* il dato da sommare all'accumulatore (tre bit, senza segno)
- NEG* si attiva per indicare che si deve convertire in negativo il numero su *D2..D0* prima di sommarlo
- GO* comanda l'esecuzione della somma

Il sistema ha 6 uscite:

- Q3..Q0* contenuto dell'accumulatore
- OVF* da attivare quando si verifica un *overflow*
- CO* da attivare quando si verifica un riporto (*Carry*)

(Suggerimenti a pag. 442)

Esercizio 7

Progettare un *controllore per un forno a microonde*, utilizzando la struttura *controllore - datapath*.

Il sistema ha due ingressi a un bit e uno ad otto.

- GO* fa partire la cottura per un tempo *TCC*
- TCC* ingresso a 8 bit per impostare il tempo di cottura
- OPN* attivo per tutto il tempo in cui porta è aperta.

Uscite:

- COK* quando attivo il forno scalda
- BEL* segnala la fine della cottura, dura un ciclo di clock

L'utente imposta il tempo di cottura e preme un pulsante. A seguito di questo, arriva al controllore il segnale di *GO*, attivo per un solo ciclo di clock, che fa partire la cottura per un tempo *TCC*.

Se la porta è aperta il forno non scalda, quando è chiusa inizia a scaldare. Se è aperta prima della fine del tempo impostato interrompe la cottura e la riprende quando la porta è chiusa.

(Suggerimenti a pag. 443)

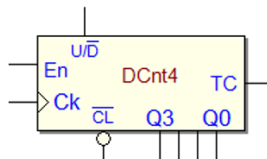
8.5 Suggerimenti

8.5.1 Progetto di sistema completo (controllore e datapath)

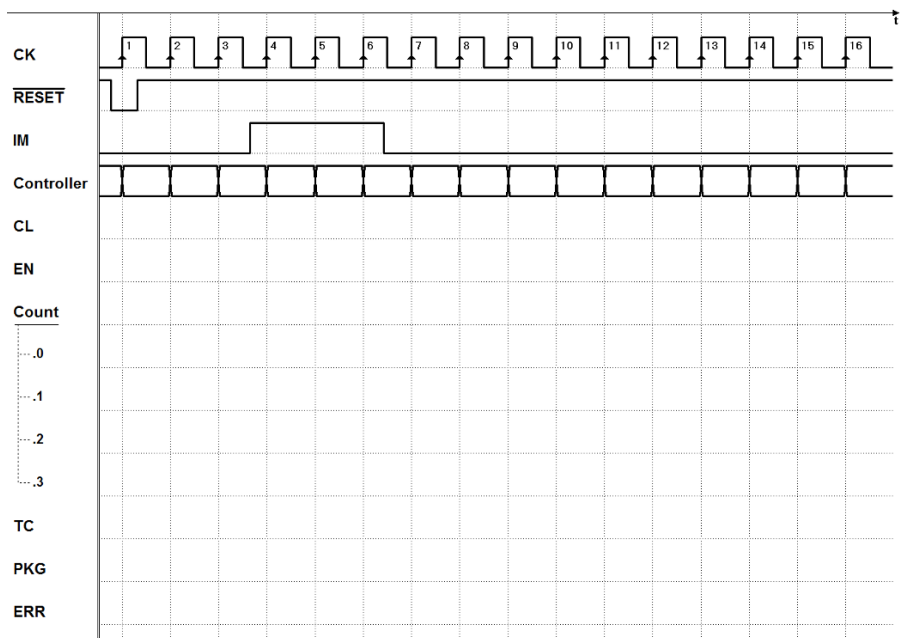
Qui trovate alcuni suggerimenti a riguardo degli esercizi proposti da pagina 434 in avanti.

Esercizio 1

Si suggerisce di utilizzare il contatore “DCnt4”; potete inoltre usare qualsiasi porta logica.

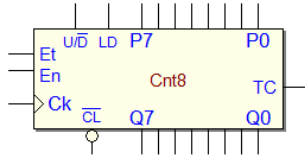


Si consiglia inoltre di definire una traccia temporale come la seguente:

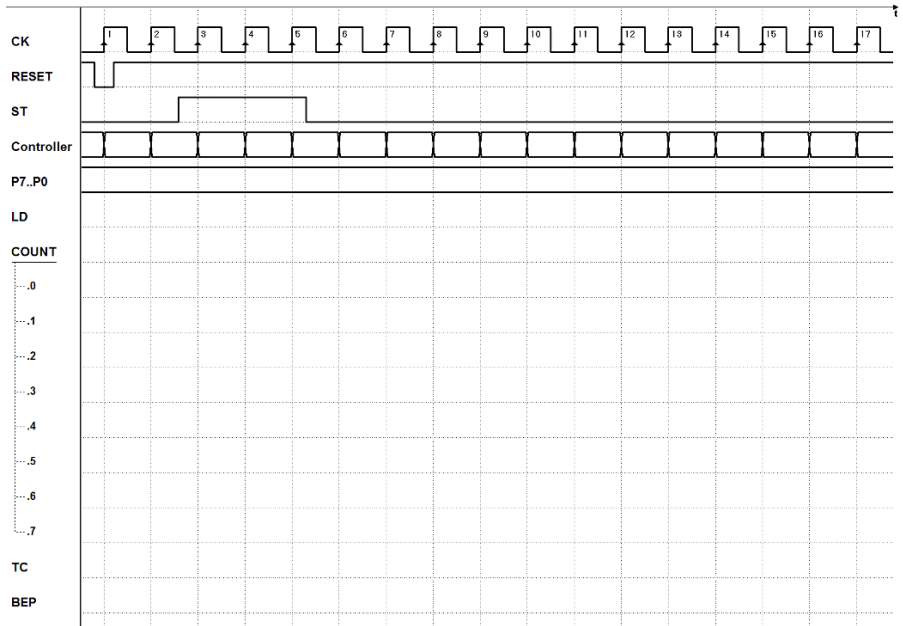


Esercizio 2

Si suggerisce di utilizzare il contatore “Cnt8”; potete inoltre usare qualsiasi porta logica.

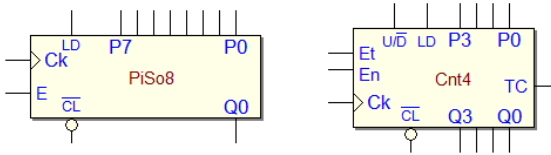


Si consiglia inoltre di definire una traccia temporale come la seguente:

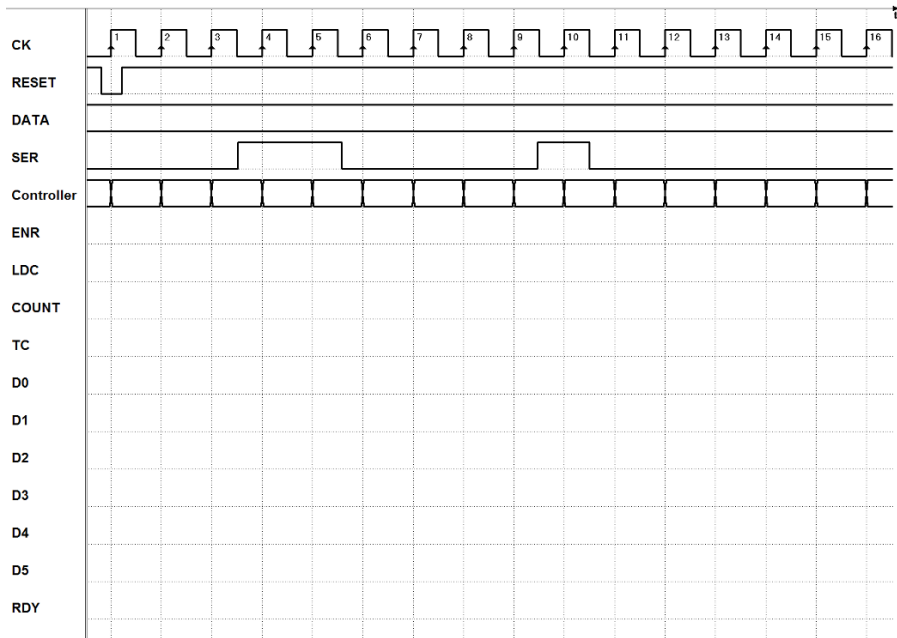


Esercizio 3

Si suggerisce di utilizzare il componenti “PiSo8” e “Cnt4”; potete inoltre usare qualsiasi porta logica.

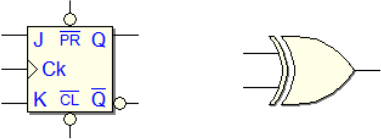


Si consiglia inoltre di definire una traccia temporale come la seguente:

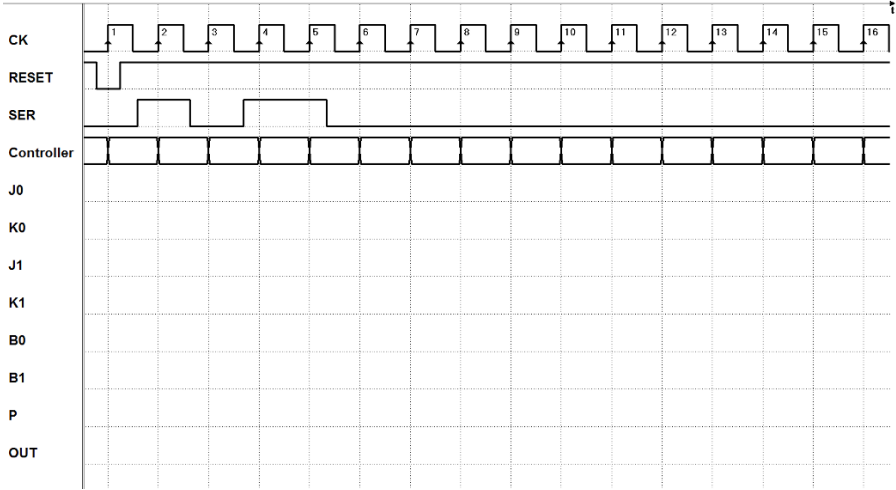


Esercizio 4

Si suggerisce di utilizzare per il datapath due flip-flop JK ed una porta XOR:

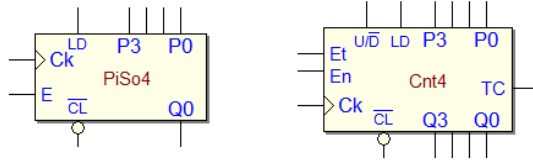


Si consiglia inoltre di definire una traccia temporale come la seguente:

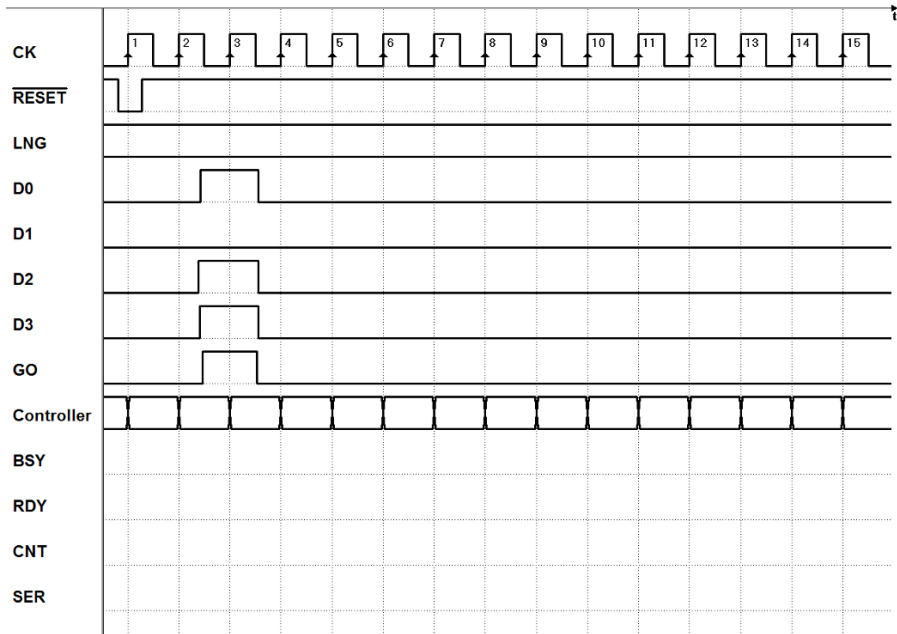


Esercizio 5

Per il datapath si suggerisce di usare i componenti riportati sotto (si tenga presente che si può anche progettare il sistema senza il contatore):

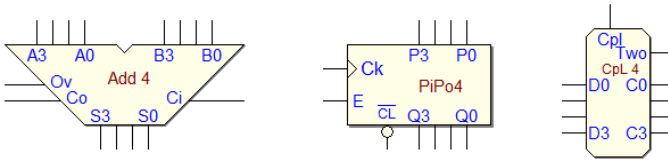


Si consiglia inoltre di definire una traccia temporale come la seguente:

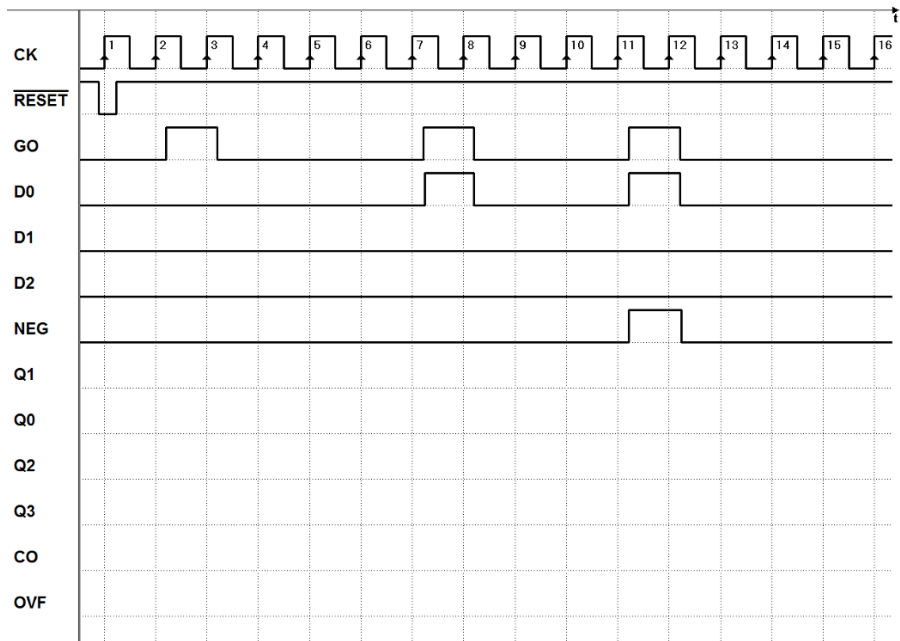


Esercizio 6

Si suggerisce di usare i componenti riportati sotto:

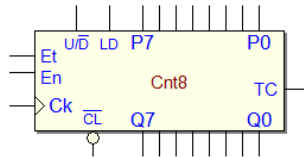


Si consiglia inoltre di definire una traccia temporale come la seguente:

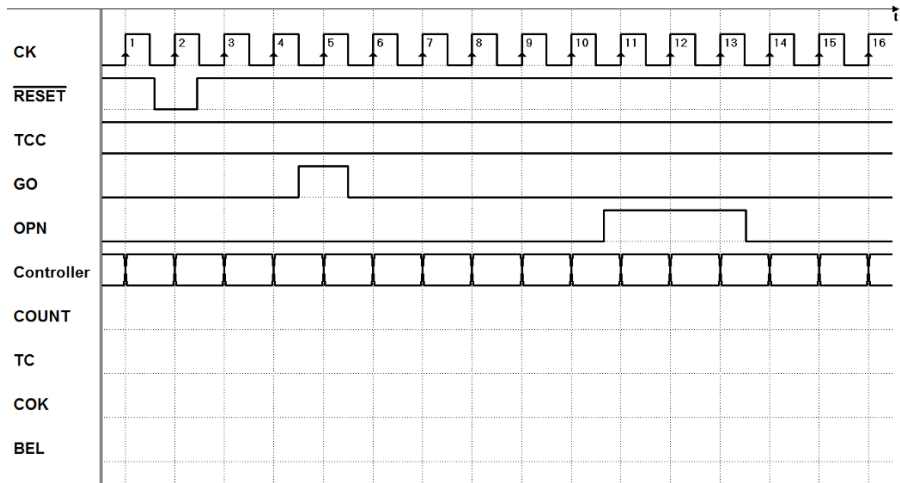


Esercizio 7

Per il datapath si suggerisce di usare il componente riportato qui sotto:



Si consiglia inoltre di definire una traccia temporale come la seguente:

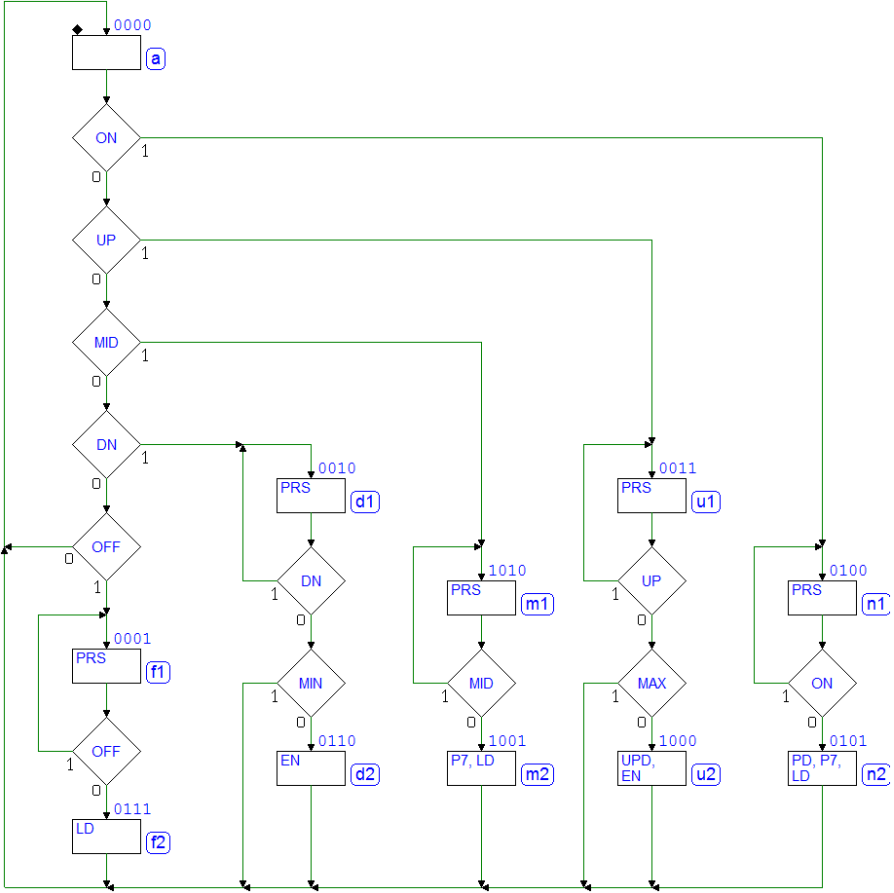


8.6 Soluzioni

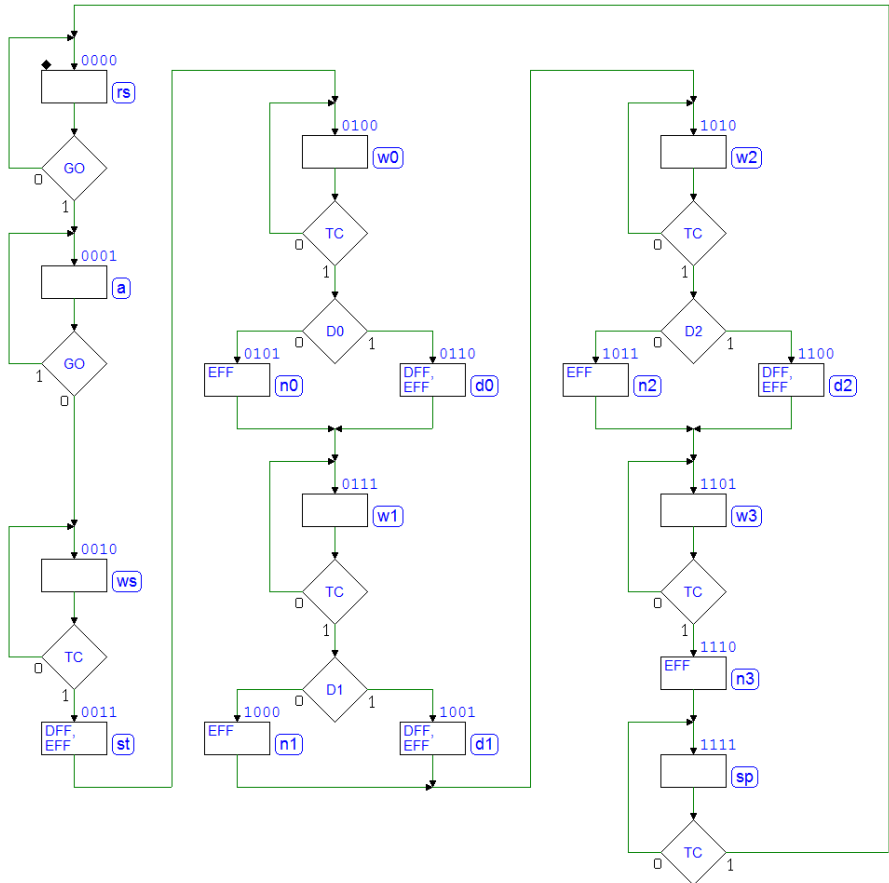
8.6.1 Progetto di controllore, con datapath assegnato

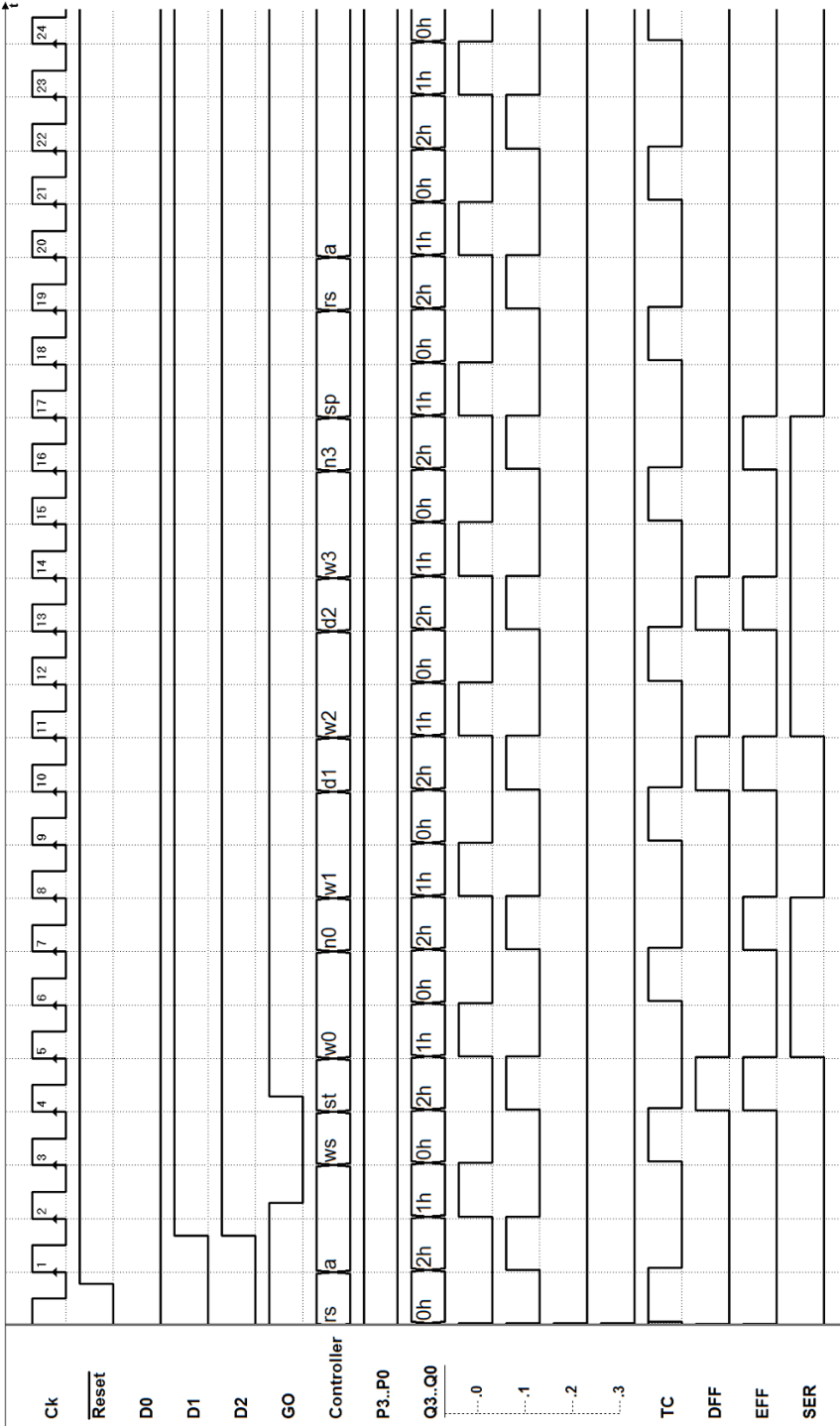
Dal sito del simulatore *Deeds* si possono scaricare i file delle MSF qui rappresentate, e anche i relativi schemi circuitali completi. Sarà utile analizzare le soluzioni proposte utilizzando la simulazione temporale.

Soluzione esercizio 1:

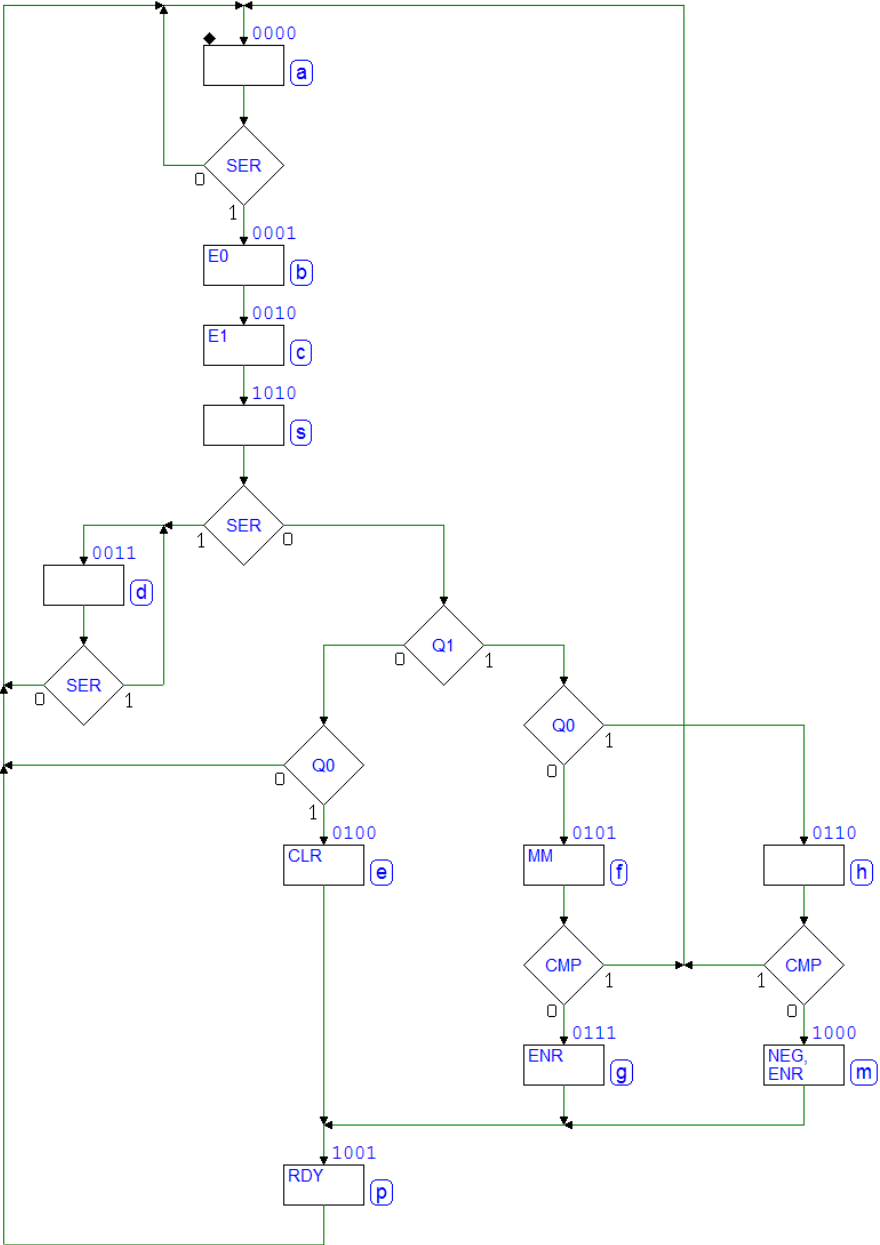


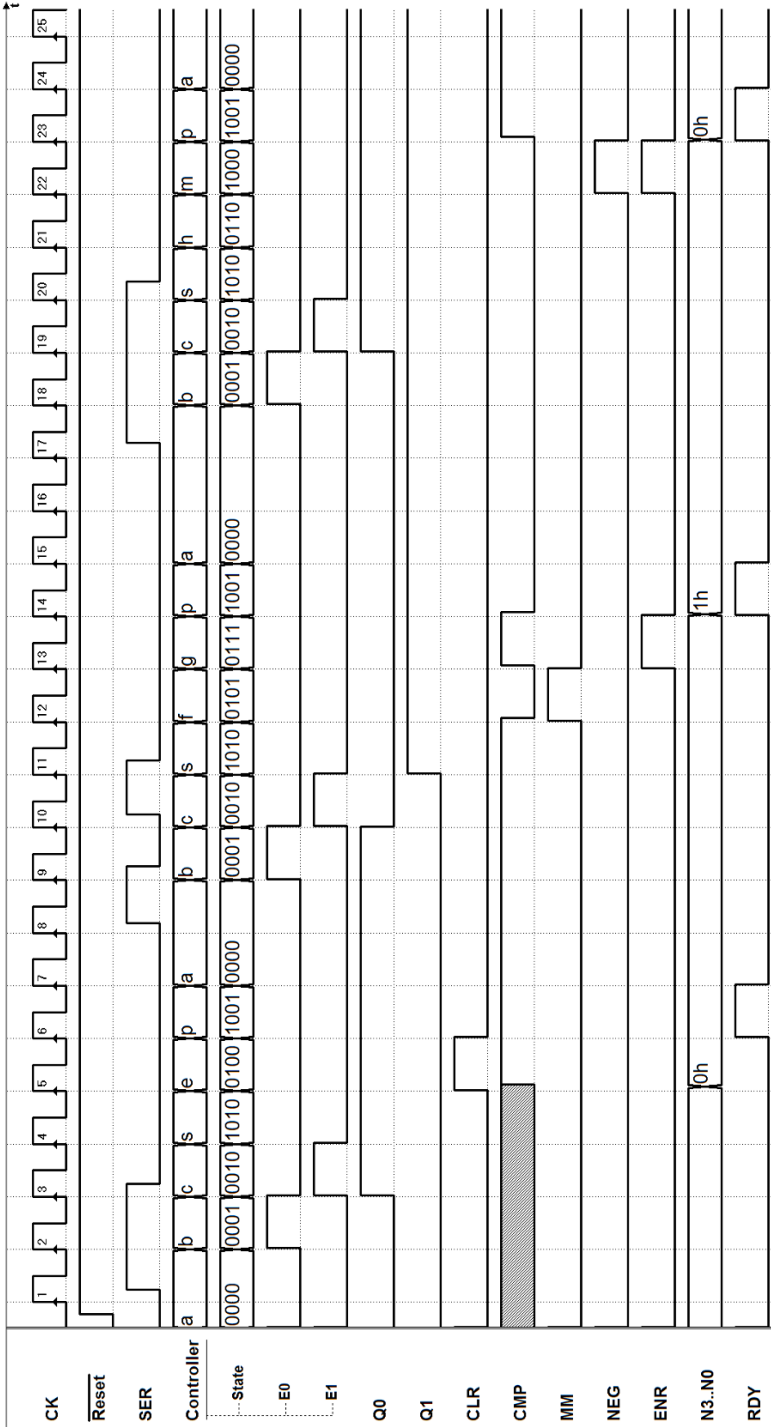
Soluzione esercizio 2:



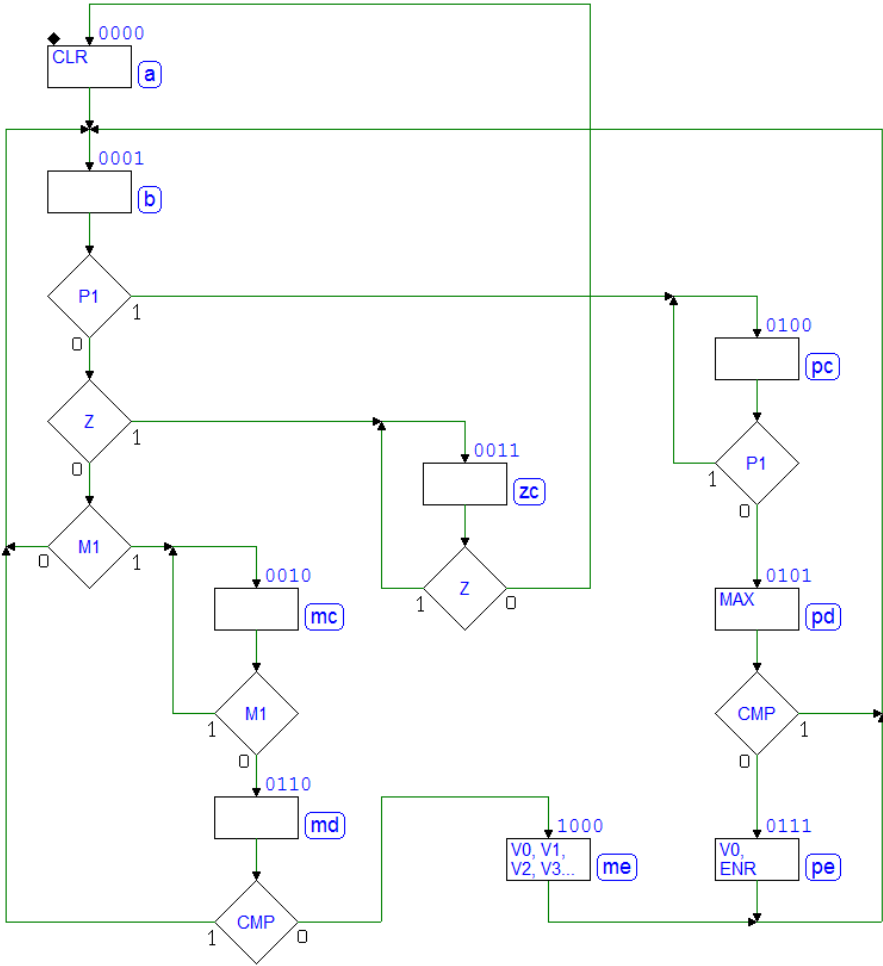


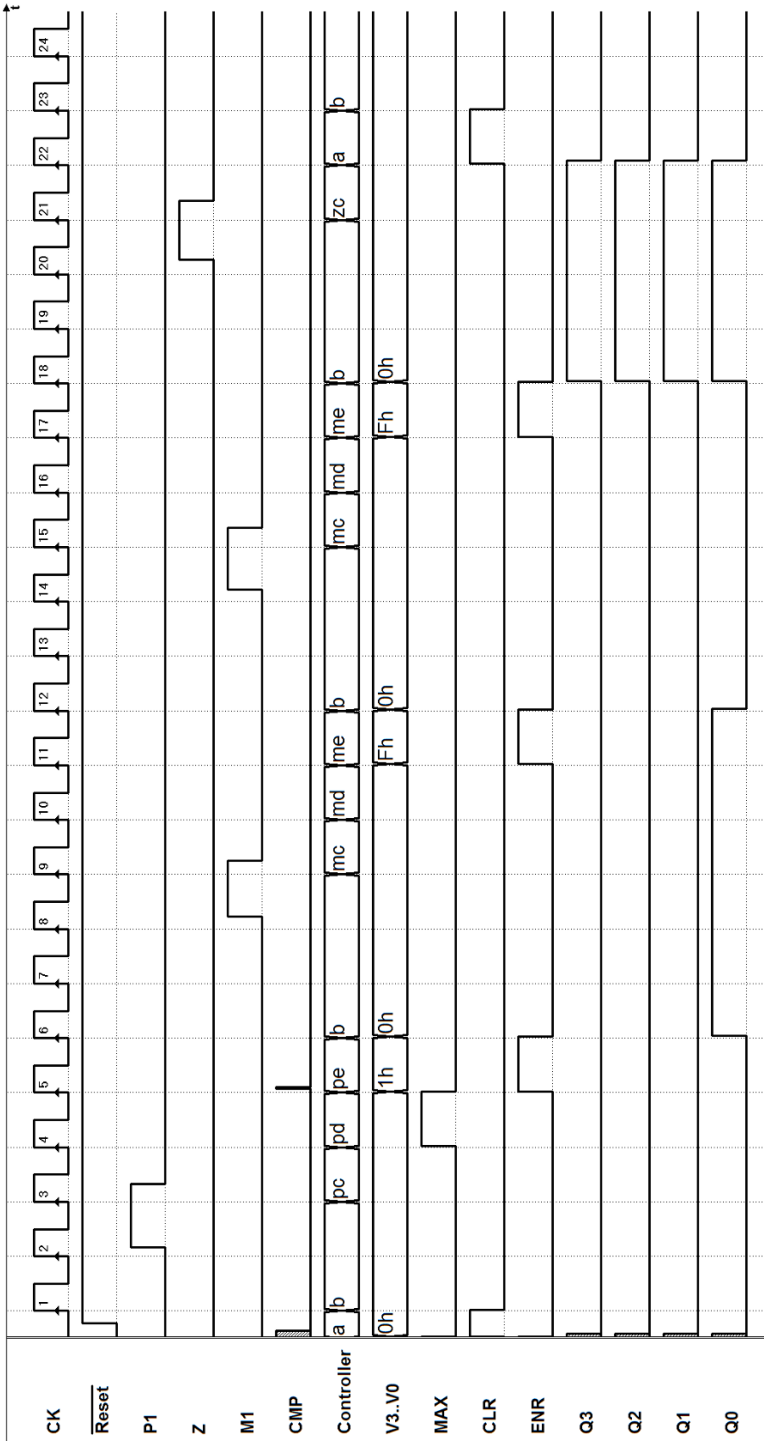
Soluzione esercizio 3:



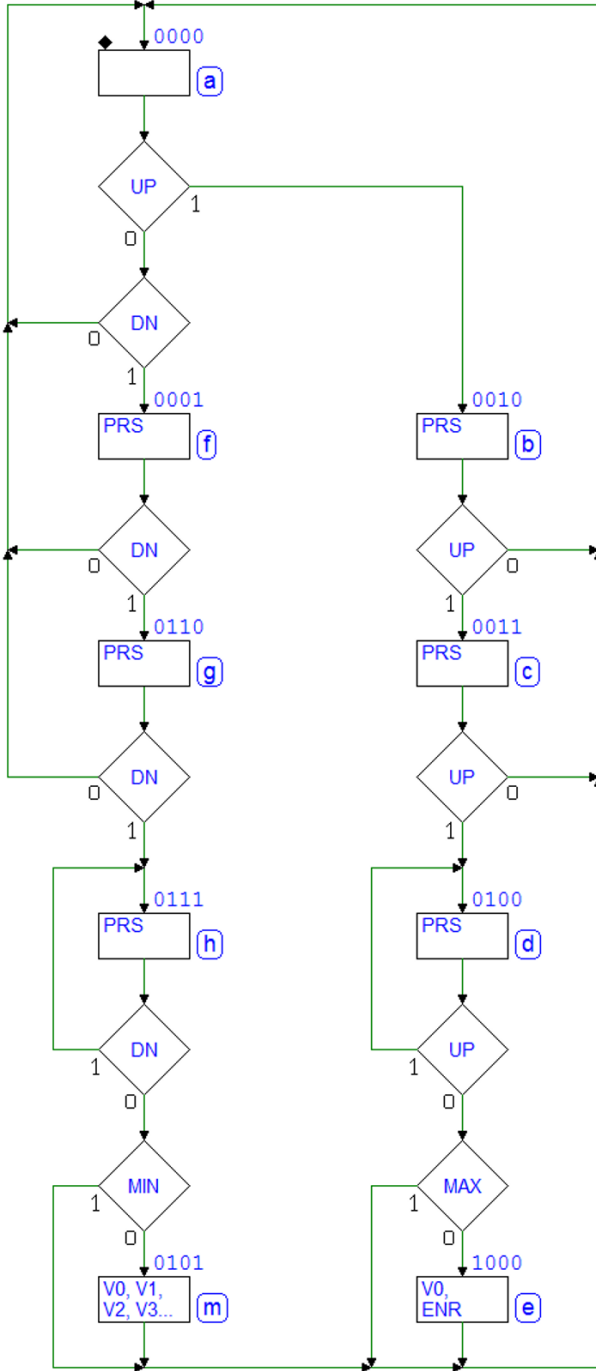


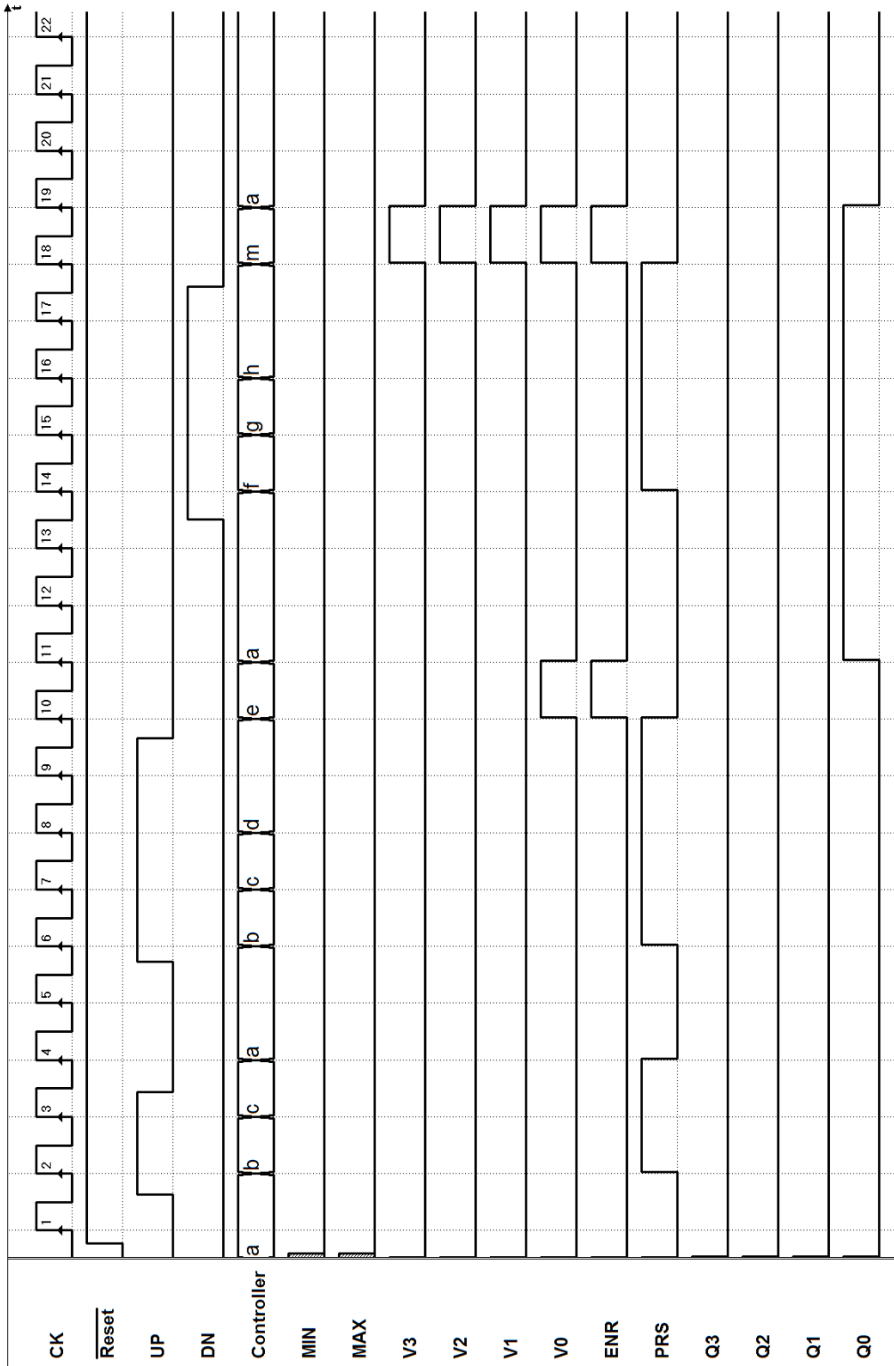
Soluzione esercizio 4:



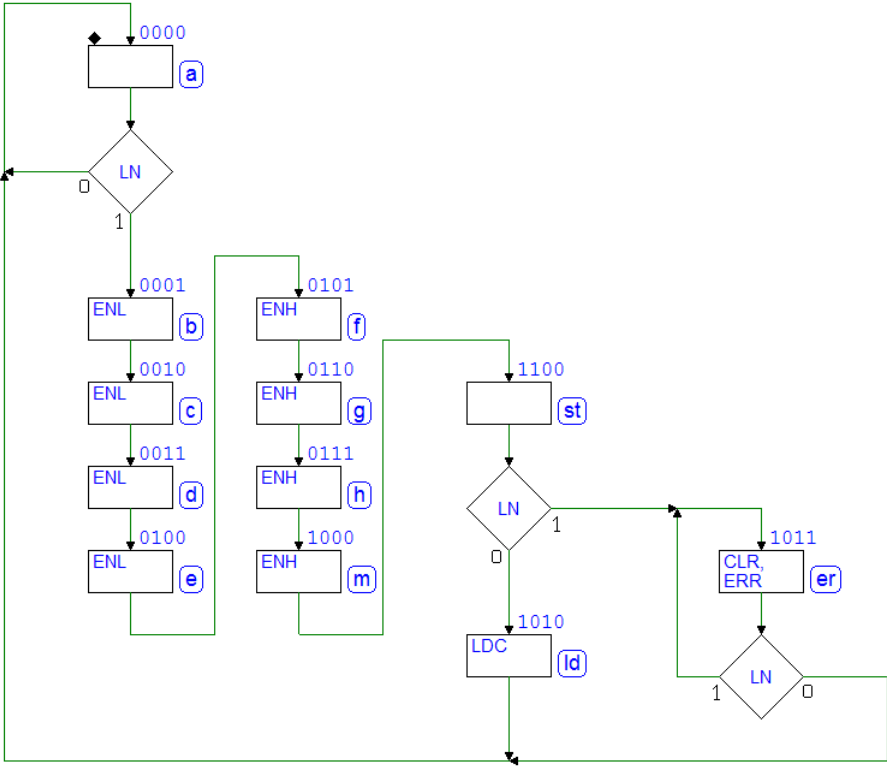


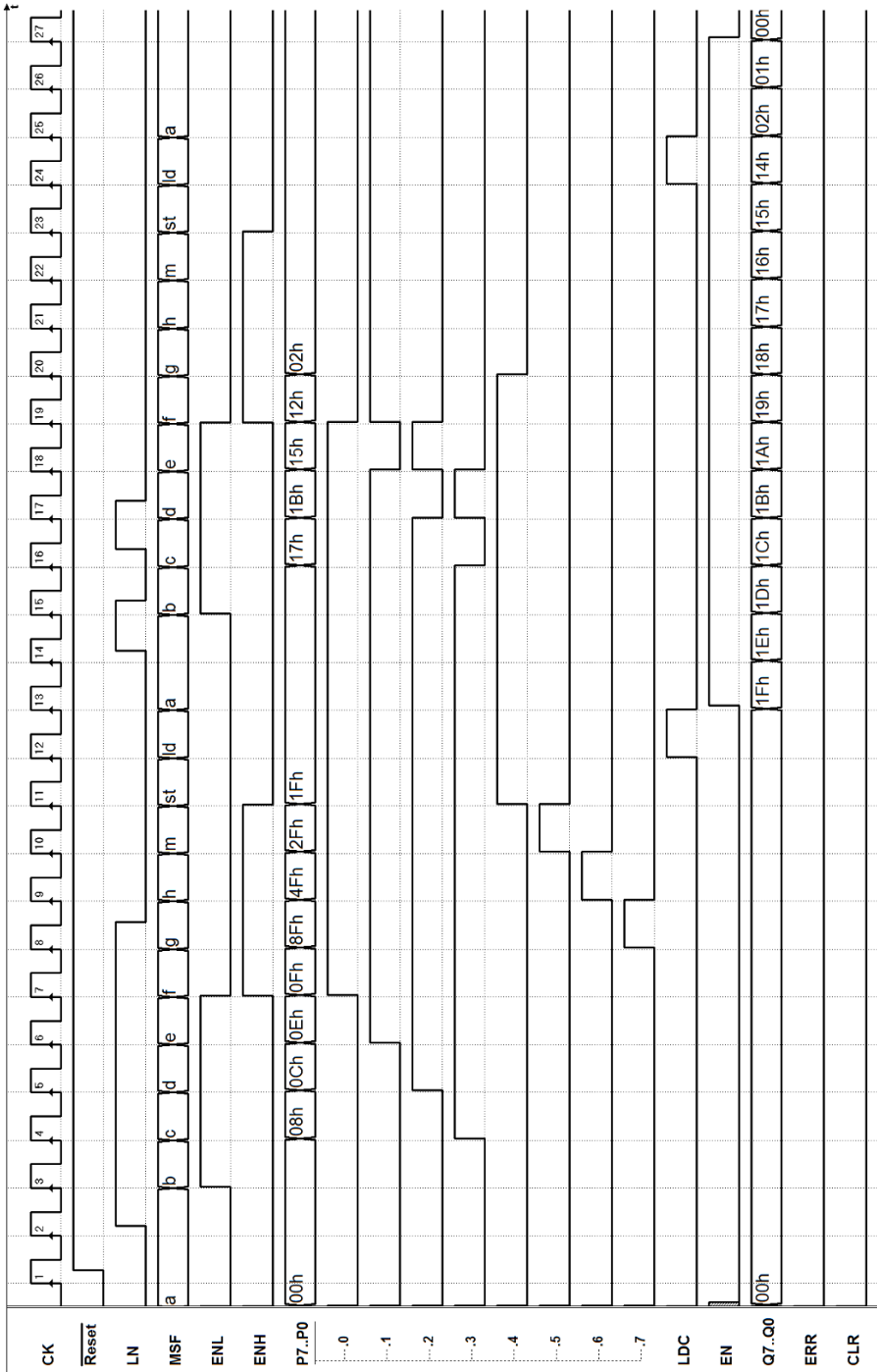
Soluzione esercizio 5:



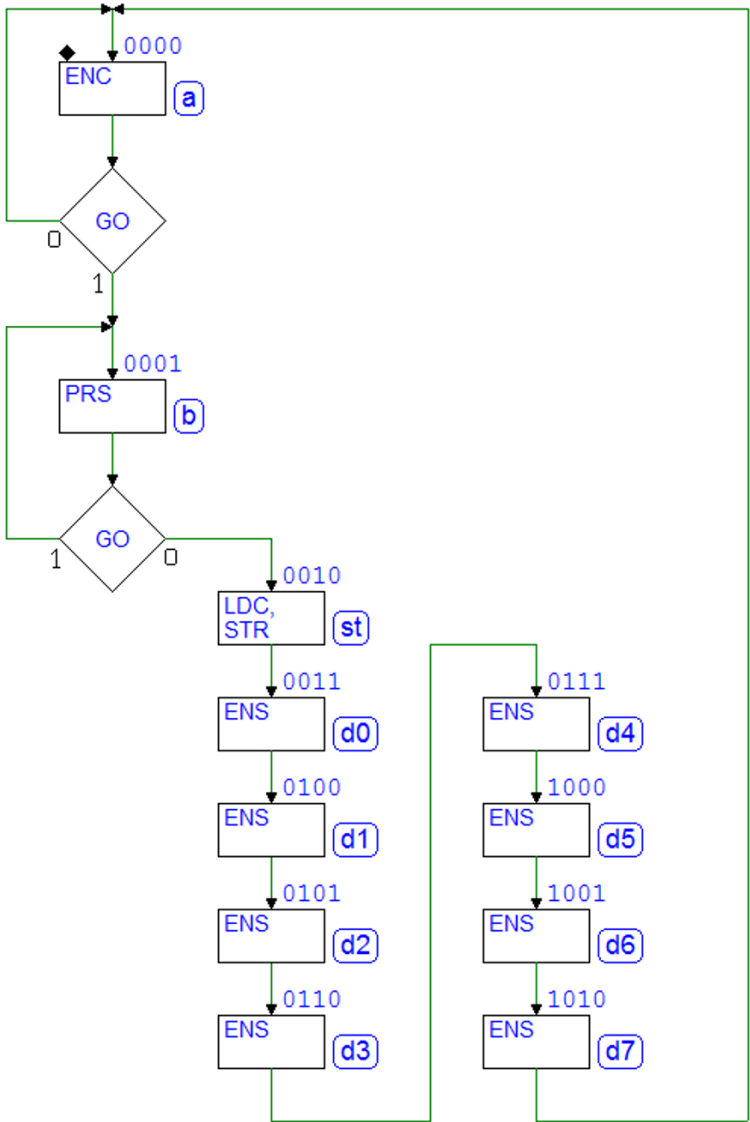


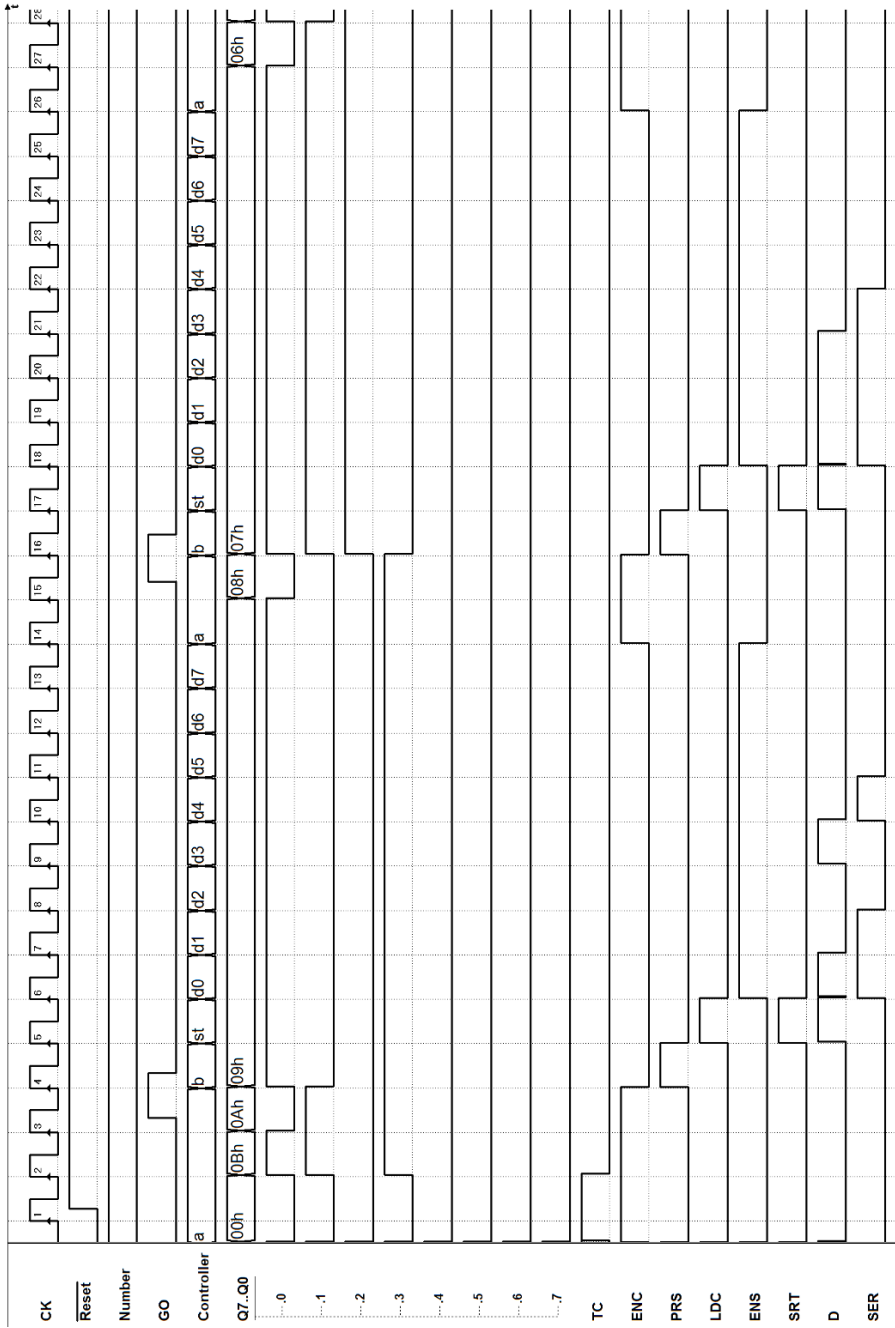
Soluzione esercizio 6:



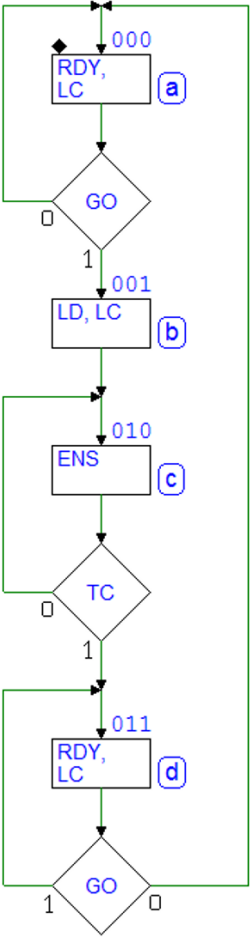


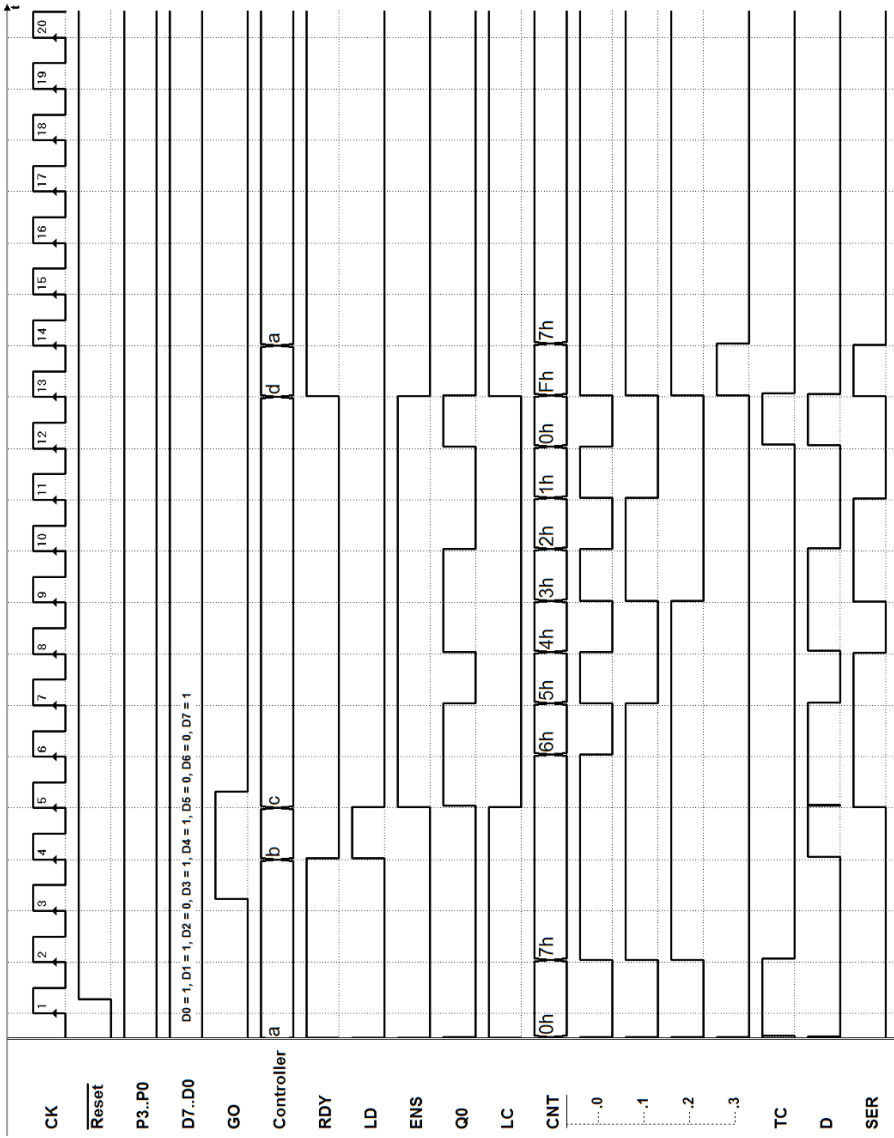
Soluzione esercizio 7:



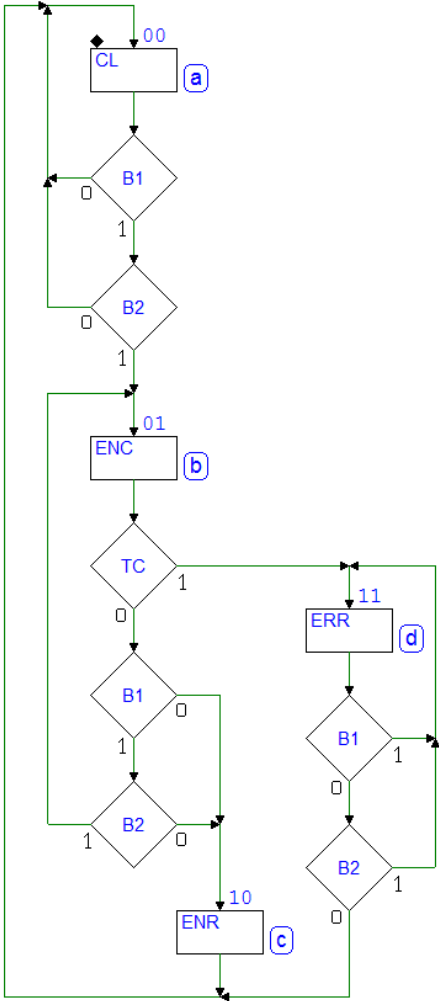


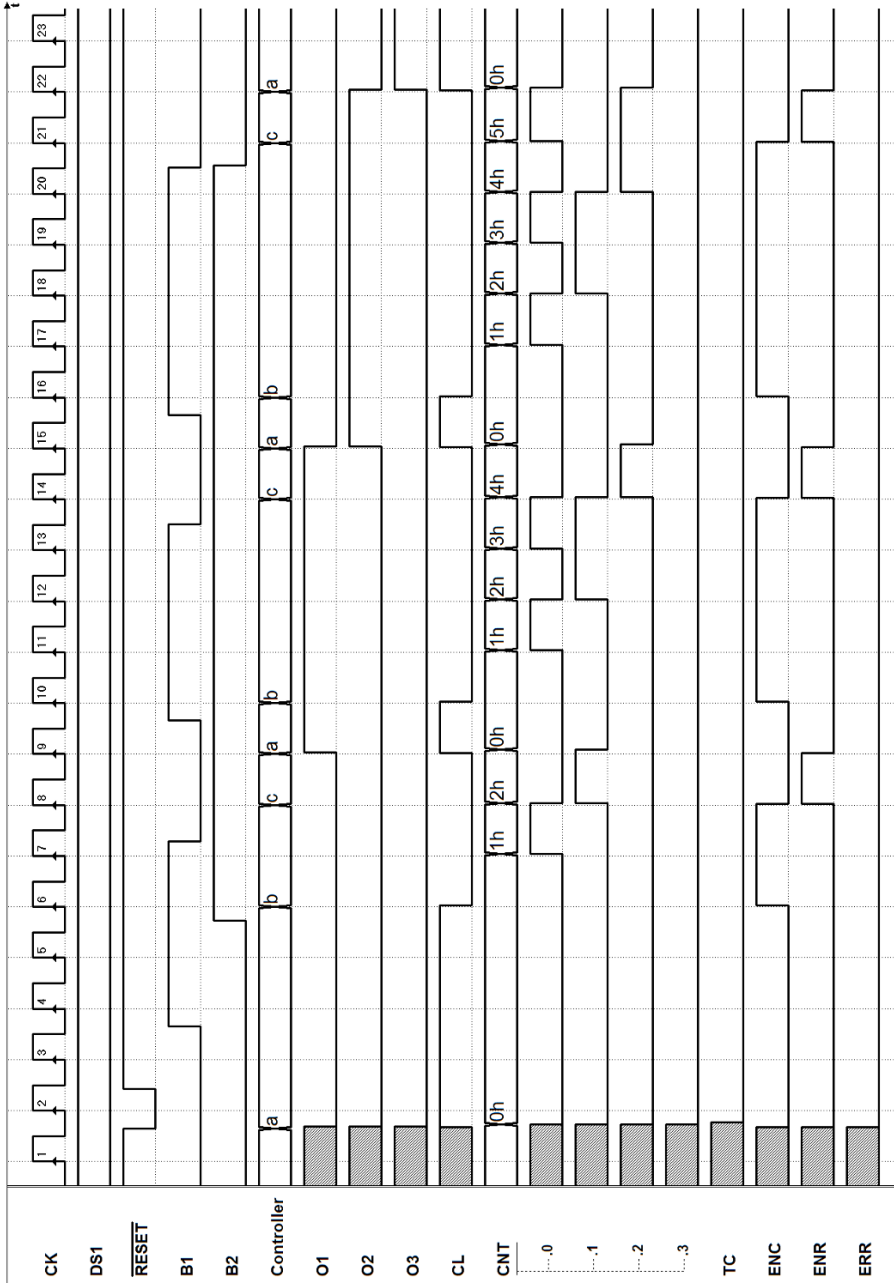
Soluzione esercizio 8:





Soluzione esercizio 9:



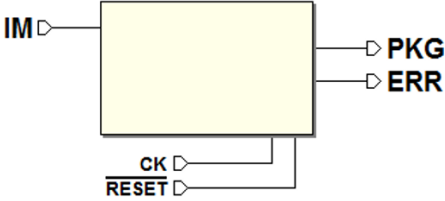


8.6.2 Progetto di sistema completo (controllore e datapath)

Sul sito del simulatore sono disponibili tutti i file corrispondenti alle figure qui rappresentate (schemi circuitali e MSF), in modo che le soluzioni possano essere verificate anche mediante la simulazione.

Soluzione esercizio 1

Ingressi e uscite del sistema:



Schema della rete (controllore + datapath):

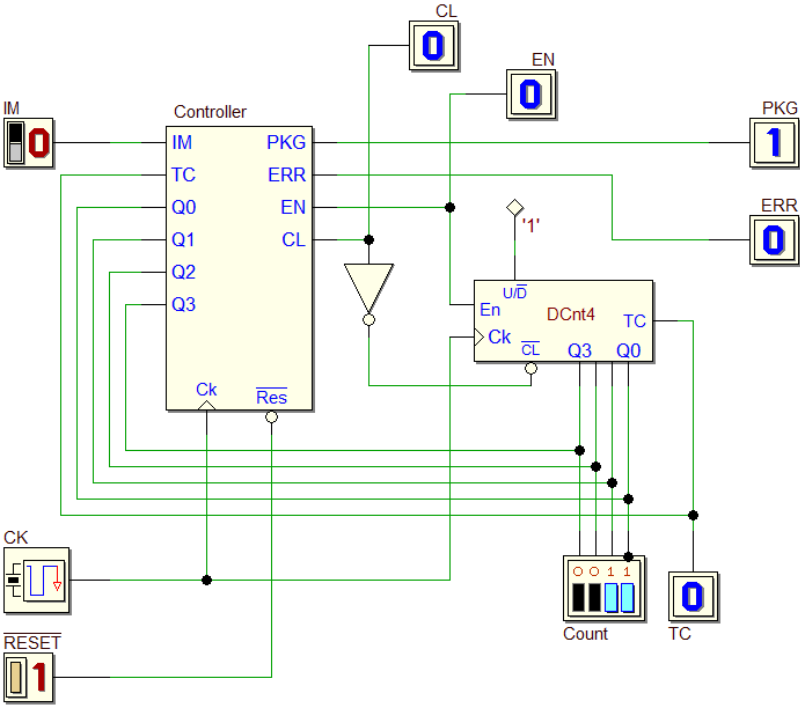
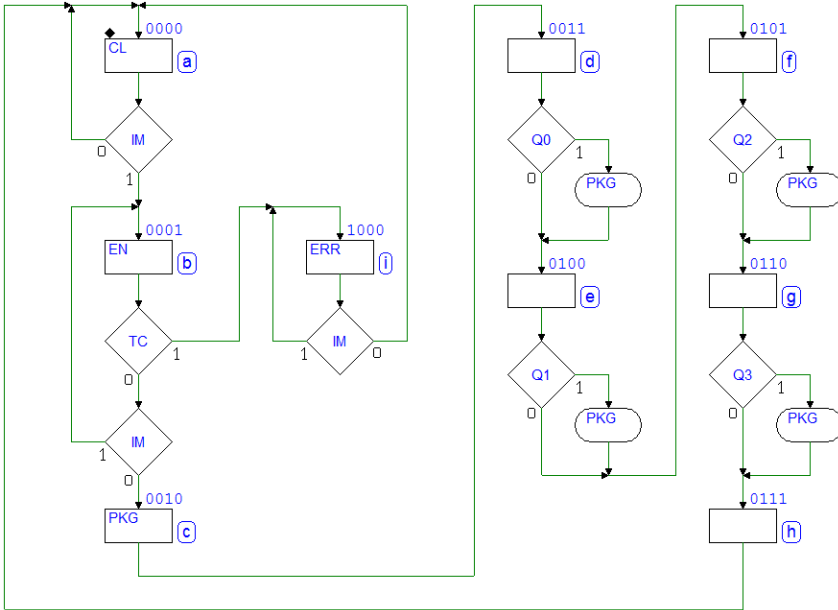
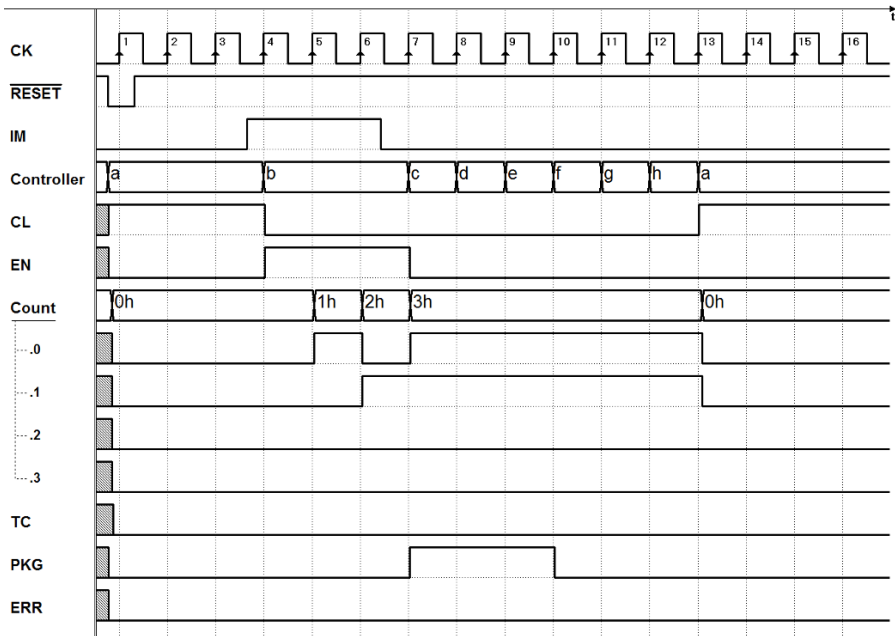


Diagramma degli stati del controllore:

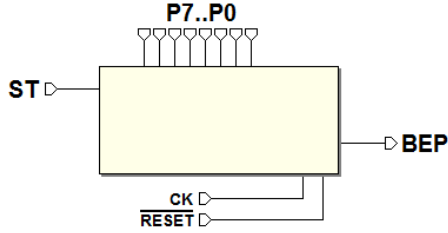


Simulazione temporale:



Soluzione esercizio 2

Ingressi e uscite del sistema:



Schema della rete (controllore + datapath):

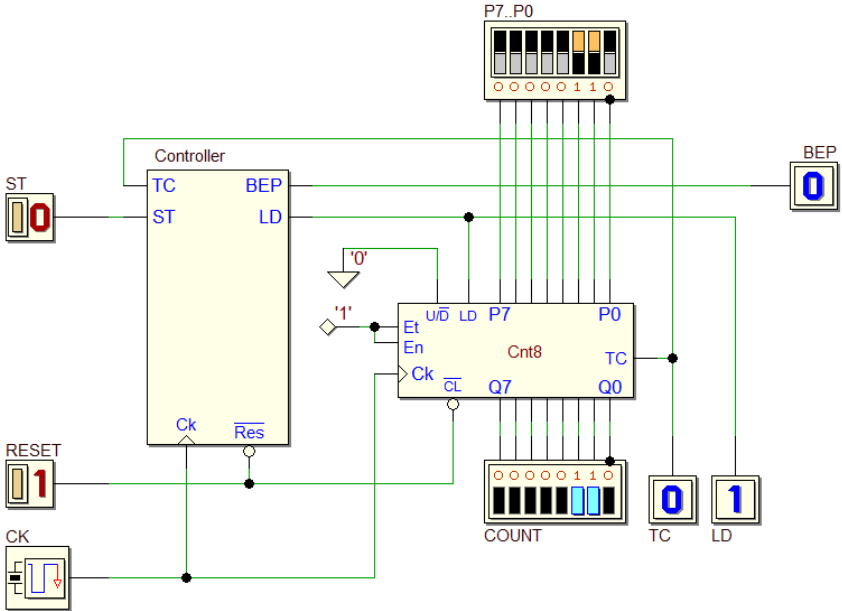
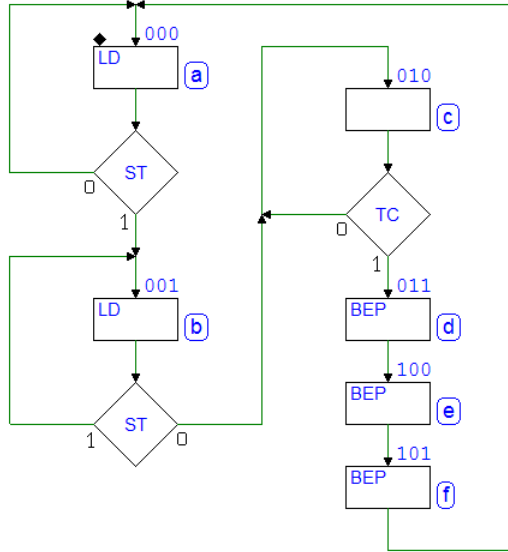
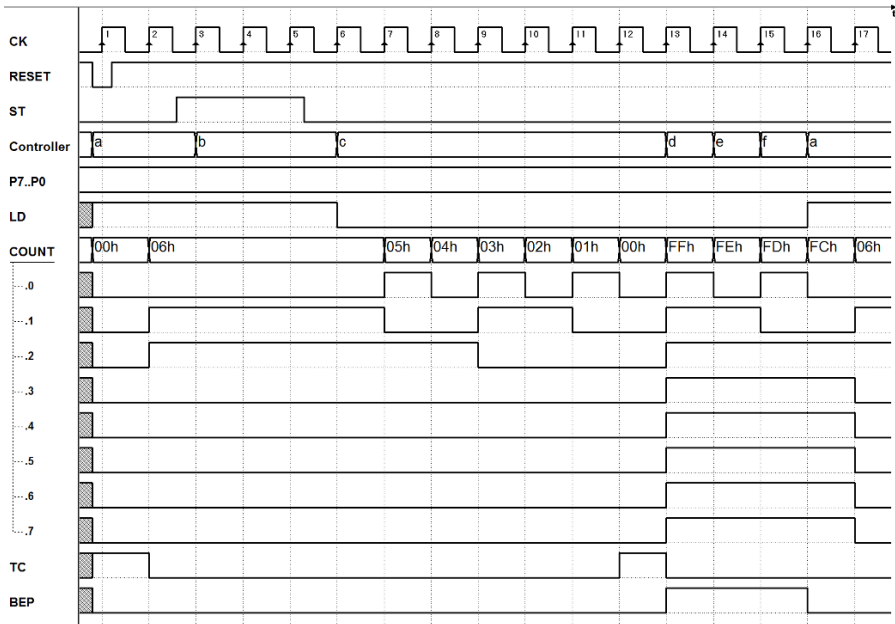


Diagramma degli stati del controllore:

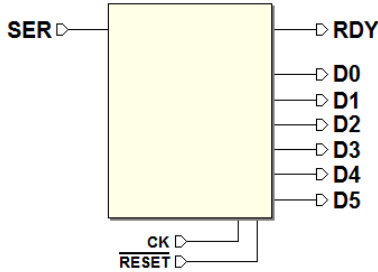


Simulazione temporale:

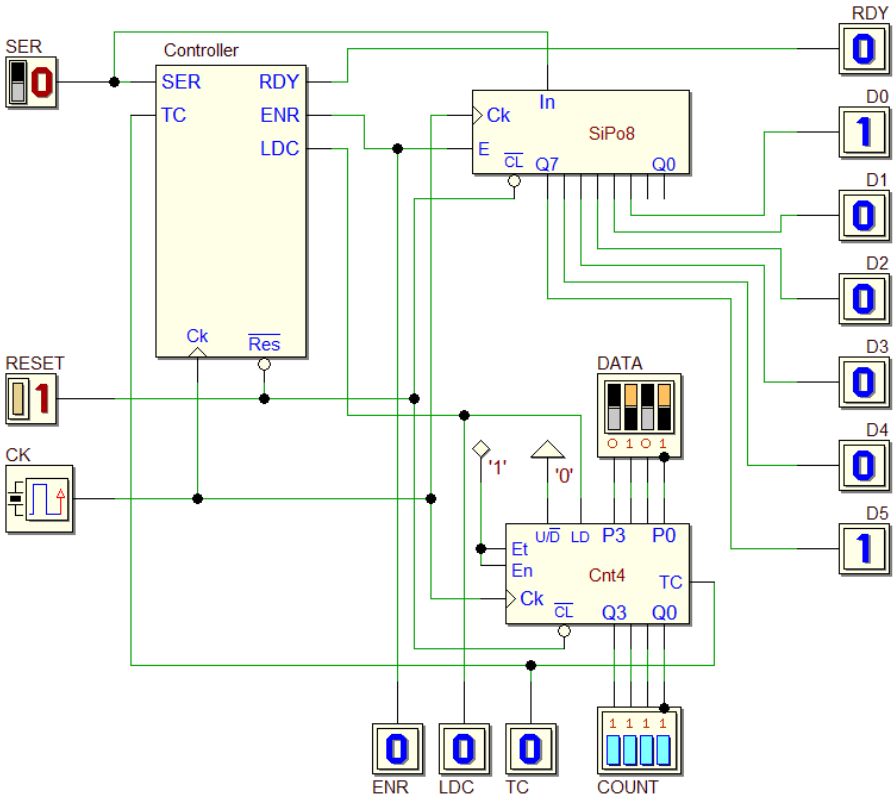


Soluzione esercizio 3

Ingressi e uscite del sistema:

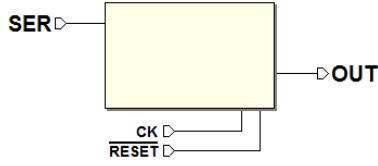


Schema della rete (controllore + datapath):



Soluzione esercizio 4

Ingressi e uscite del sistema:



Schema della rete (controllore + datapath):

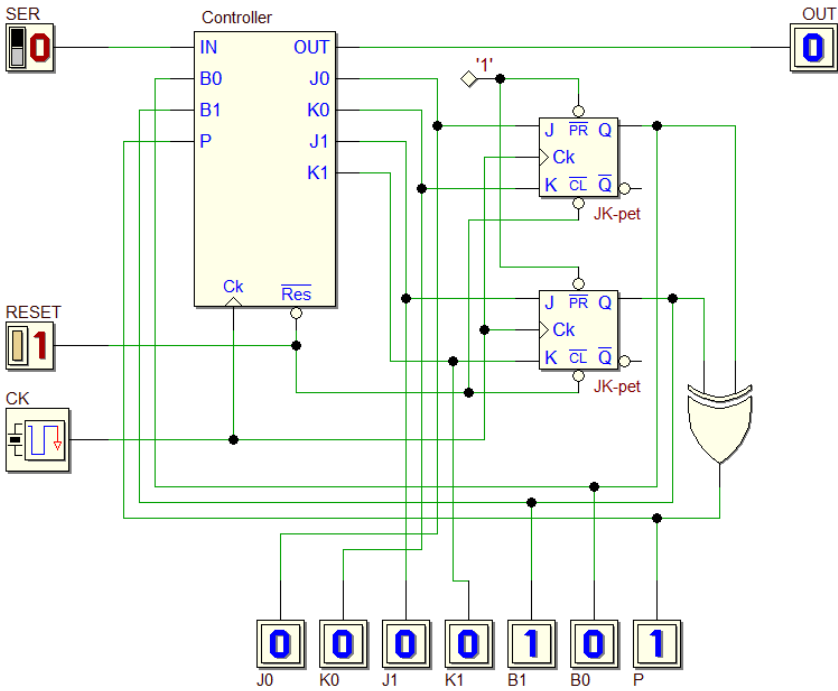
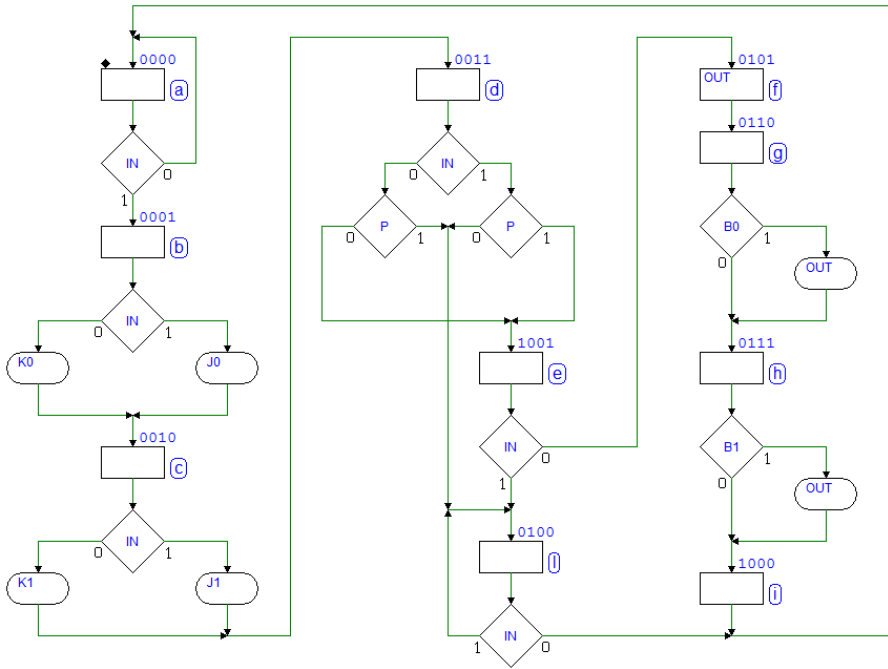
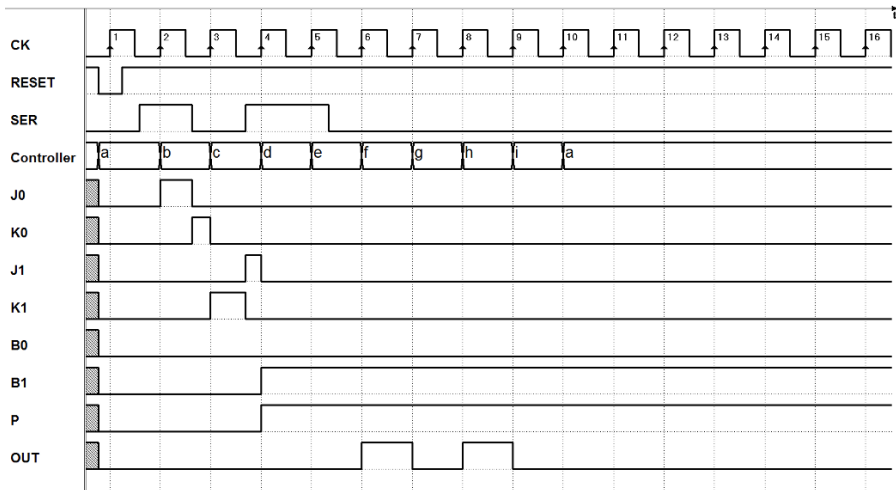


Diagramma degli stati del controllore:

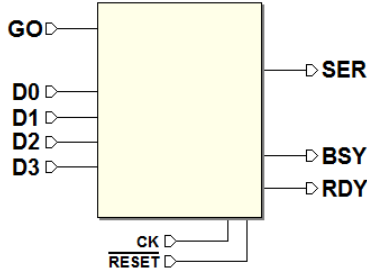


Simulazione temporale:



Soluzione esercizio 5

Ingressi e uscite del sistema:



Schema della rete (controllore + datapath):

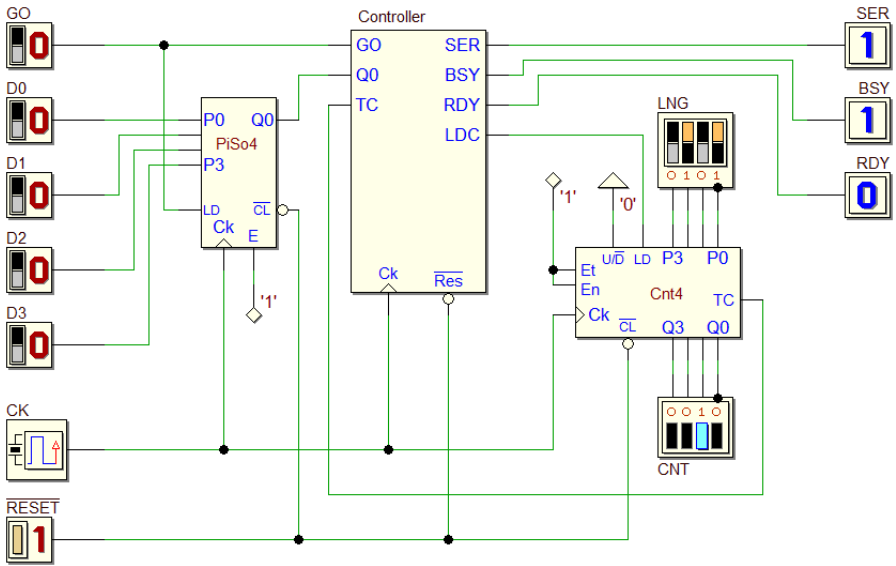
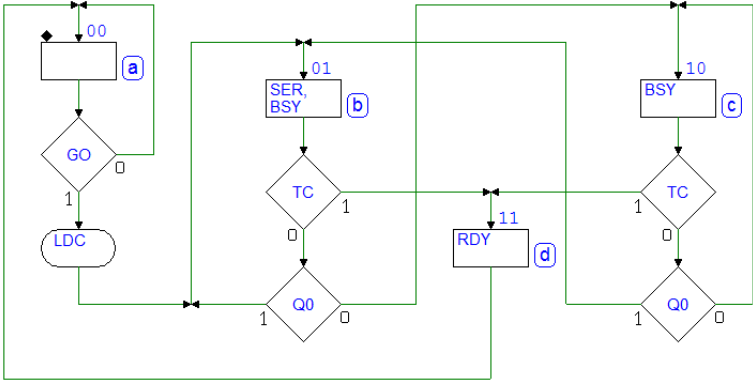
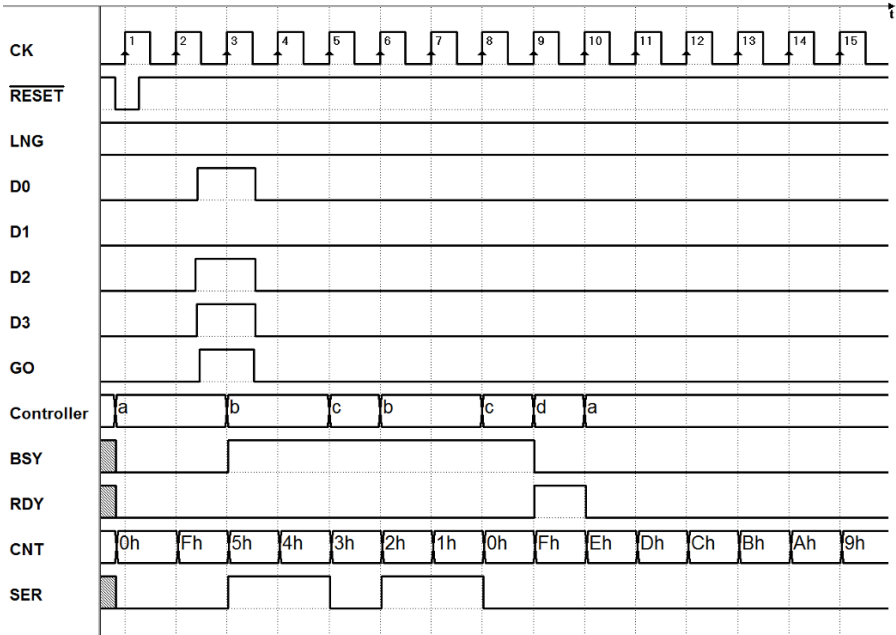


Diagramma degli stati del controllore:

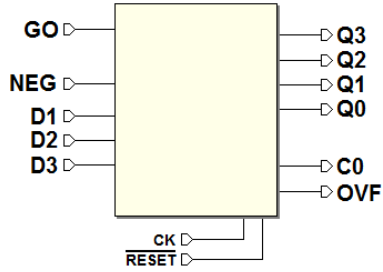


Simulazione temporale:

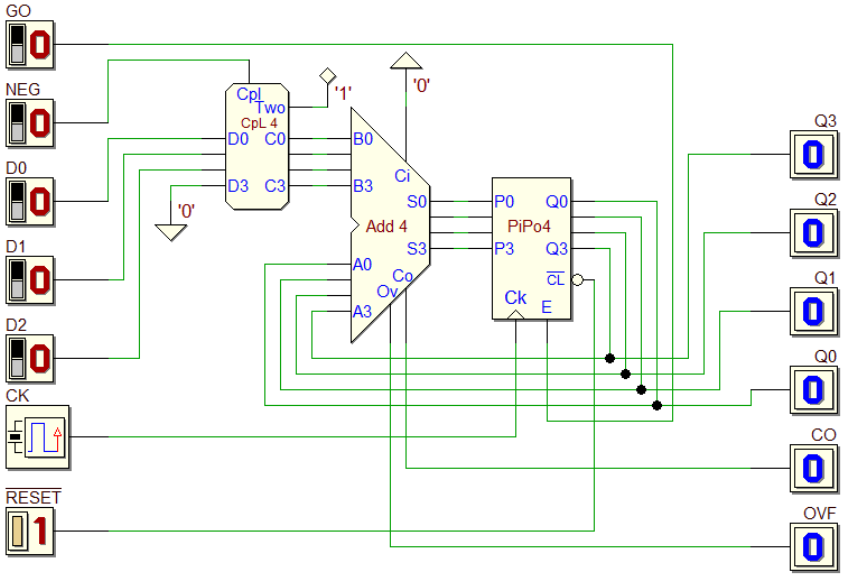


Soluzione esercizio 6

Ingressi e uscite del sistema:

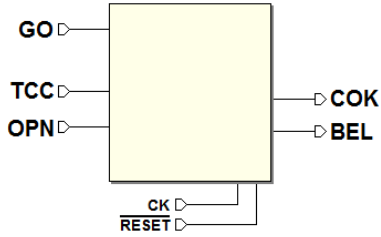


Schema della rete (solo datapath):



Soluzione esercizio 7

Ingressi e uscite del sistema:



Schema della rete (controllore + datapath):

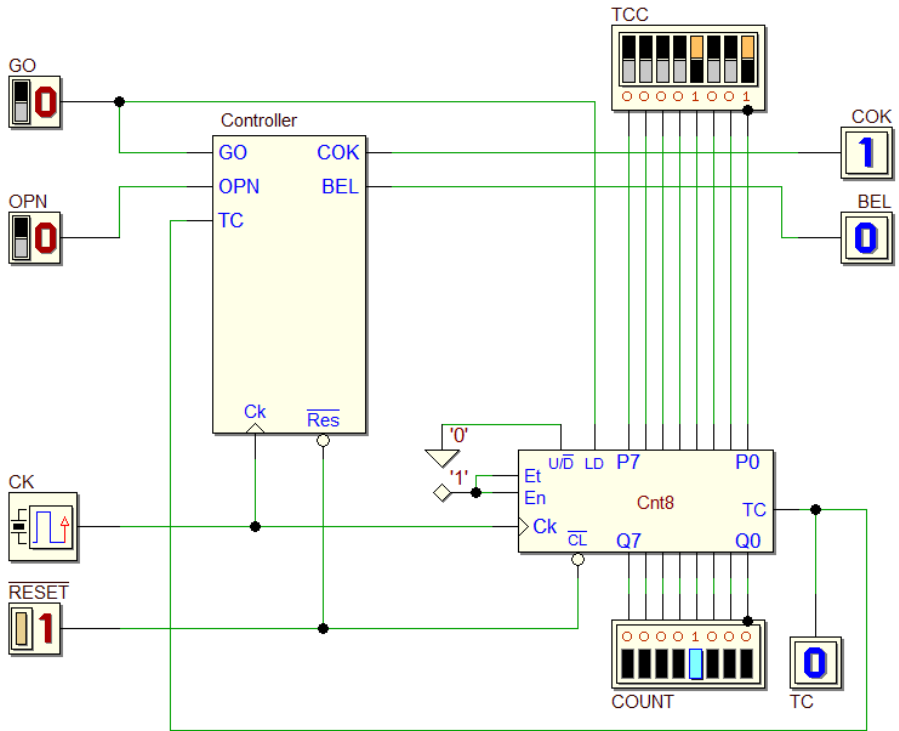
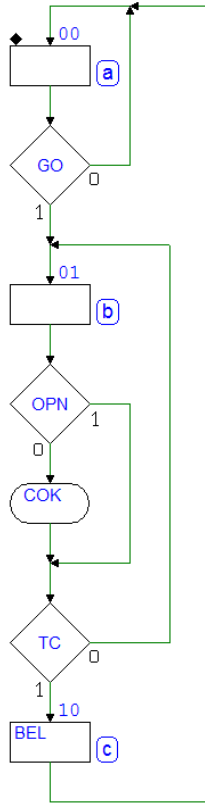


Diagramma degli stati del controllore:



Simulazione temporale:

