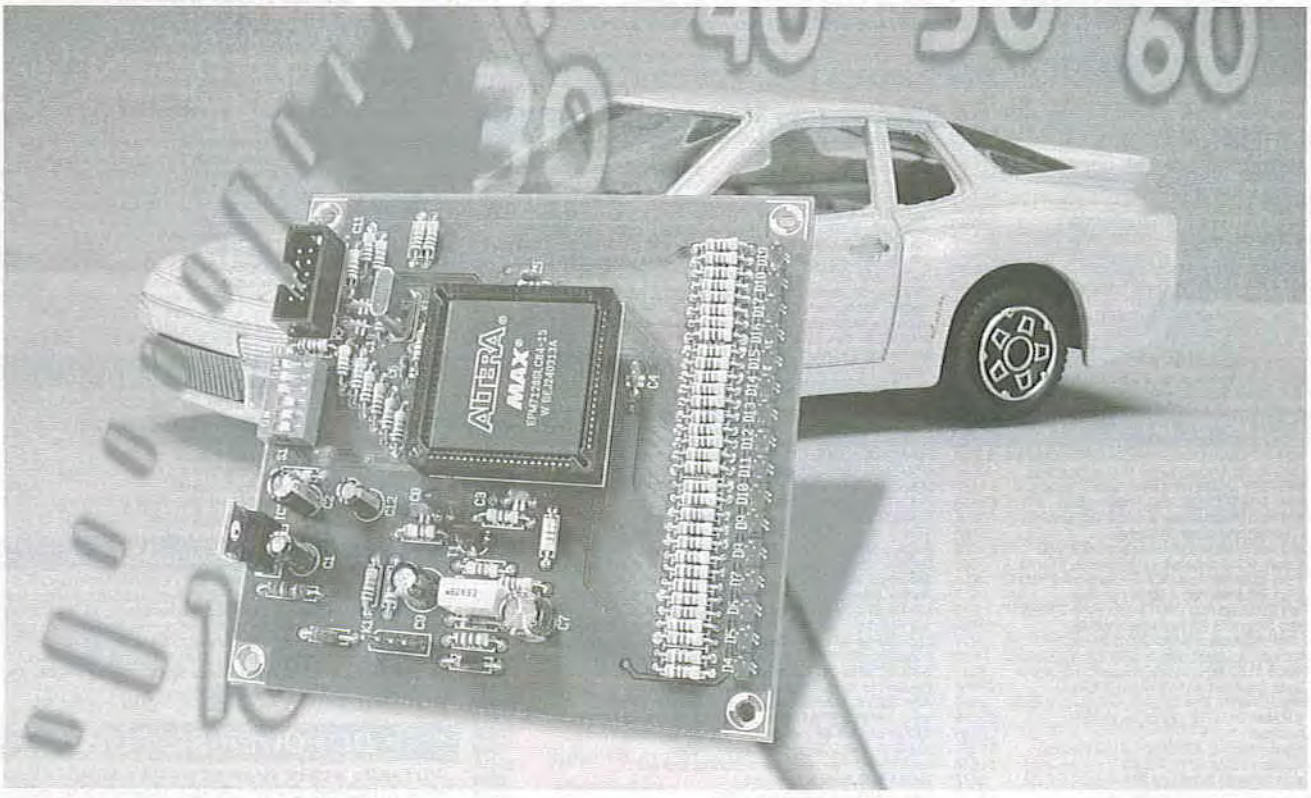
Hands-on CPLDs (2)

Part 2: Altera 7000S hardware

By A. Rosenkränzer

In this second article in the series we take a closer look at some of the hardware features of the 7000S-CPLD from Altera, this should help explain the process of programming the rev counter circuit into the CPLD.



After last month's look at the rev counter circuit we now delve a bit deeper into the internal structure of a CPLD and use the Verilog hardware description software to configure the logic in a CPLD chip. For more in-depth information the Altera web site has a rich source of support documentation.

The smallest unit in the CPLD is the macrocell — it consists of a flipflop together with some peripheral logic. The logic can be wired to 36 inputs. Both true and inverted versions of these 36 inputs are available internally. These signals can then be selectively AND gated to provide up to five product terms per macrocell. The resultant signals can now be connected to the inputs of OR or XOR gates and used as a data input to a flipflop. They can also be routed to other macrocells via a configurable expander path. The type of flipflop implemented in each macrocell can be defined as a D, T, JK or SR type. The flipflops can also be bypassed if a purely combinational logic function is required from the macrocell.

The flipflop can be clocked from either of two global clock signals or a product term from some on-board

combinational logic. Generally speaking a synchronous design (where all registers clock signals are referenced from a common clock) is preferred to an asynchronous design. It is more problematic (especially for complex circuits) if a clock pulse is generated asynchronously within the circuit. In principle it is possible to use both rising and falling edges of the clock to move data simultaneously but this should only be employed if there is no other solution because it reduces the maximum usable clock rate. It is better to gate enable signals together with the appropriate clock edge. Each flipflop has a SET and RESET input (these are effective immediately and are not gated by the clock signal). Both of these signals can be supplied from logic outputs or any RESET can be tied to the Global-clear-Signal and its input pin.

Altera-CPLDs have 16 macrocells in each LAB (Logic Array Block). Each LAB has 36 input signals connected to a PIA (Programmable Interconnect Array). This arrangement can put some limitations on logic operations e.g. it is not possible to compare two twenty-bit words in one LAB because it would require 40 inputs. In practice 36 signals are sufficient for the vast majority of applications. All outputs from each macrocell together with special input pins and all I/O pins are connected to the PIA.

The different designation numbers in the 7000S series indicate the amount of logic available on the chip. The 7032S is the smallest in the family and has just two LABs i.e., 32 macrocells. The two numbers following the 7 therefore indicate the number of macrocells in the chip. The package outline and pin-count defines if all or only a few macrocells can have an I/O pin.

The file containing chip programming information is transferred from the PC to the chip over the JTAG interface connector. The ByteBlaster adapter links the JTAG port with the parallel port of the PC. The newer ByteBlasterMV runs in a wider range of computer operating systems and can accommodate a wider range of operating voltages (the original could only handle 5 V). Earlier still was the BitBlaster adapter, this

Table I

```
/*
% ========================================================================== %
% |                                                                        | %
% | Function   : PAL for AR rev counter                                    | %
% |                                                                        | %
% |========================================================================| %
% | Chip Type  : 7128S PLCC84                                              | %
% | Manufacturer: Altera                                                   | %
% | Project    : Rev counter with freely programmable LEDs                 | %
% | PC Board   : Prototype                                                 | %
% | Reference  : Rev counter                                               | %
% | Language   : Verilog, Quartus 3.0                                      | %
% | Author     : AR                                                        | %
% | Company    : Myself                                                    | %
% |                                                                        | %
% |========================================================================| %
% | Version  Date      Modification/Reason                                 | %
% |------------------------------------------------------------------------| %
% | V 1.0    13.10.02  Initial Version                                     | %
% |------------------------------------------------------------------------| %
% | V 2.0    08.11.02  Clock at 455kHz, ceramic Resonator                  | %
% |------------------------------------------------------------------------| %
% | V 2.1.   10.11.02  Bar mode only, no individual LED                    | %
% |------------------------------------------------------------------------| %
% | V 3.0    28.11.02  7128, 4.915MHz xtal, thousands readout              | %
% |------------------------------------------------------------------------| %
% | V 3.1    28.12.02  External Reset                                      | %
% |------------------------------------------------------------------------| %
% | V 4.0    16.04.03  DZ_IN_B for Schmitt trigger                         | %
% |------------------------------------------------------------------------| %
% | V 4.1    25.05.03  Reduced brightness with lights on                   | %
% |------------------------------------------------------------------------| %
% | V 5.0    09.10.03  Converted to Verilog                                | %
% ========================================================================== %
*/

module drehzahl
        (
        CLK,
        RESET,
        ZYL,
        MODE,
        LICHT,
        MRES,
        C4_IN,
        C4_OUT,
        DZ_IN,
        DZ_IN_B,
        LED_R_OA,
        LED_R_OB
        );

        input           CLK;
        input   [2:0]   ZYL;
        input   [2:0]   MODE;
        input           LICHT;
        input           C4_IN;
        input           RESET;
        input           DZ_IN;

        output          MRES;
        reg             MRES;
        reg             Next_MRES;

        output          C4_OUT;
        reg             C4_OUT;

        output          DZ_IN_B;
        reg             DZ_IN_B;

        reg     [8:0]   M;
```

plugged into the computer serial interface. More recently a USB version has been released. It is however not too important which path the data takes to get to the JTAG interface.

Programming

Many different software packages from as many different manufacturers are available. Altera offer both MAX2PLUS and QUARTUS.

MAX2PLUS is a relatively old software package. The free web version only allows the circuit connectivity to be entered using circuit schematics or an AHDL text file. Verilog and VHDL input can only be used on the licensed version of the software. Altera have announced that this package will not be supported in the future.

QUARTUS offers the language Verilog and VHDL in its free web edition and it can be used for large FPGAs but it does require more processing power from your computer and also more memory. These requirements should not pose a problem for any modern-day home PC. Quartus will therefore be used for the development of this project. Version 3.0 of the software is currently available for download from Altera. Once installed it is necessary to accept the terms of a *License File* via email. You will need the number of your hard drive or network adapter. The reply should only take a few minutes.

QUARTUS allows the design information to be entered in several different formats. Designs can be organised hierarchically or in a mixed format. For this project we will use a text-based input file and keep the design simple by using a flat hierarchy (only one layer). The program language VERILOG will be used throughout the development. VERILOG is not so hardware-near as AHDL but it is simpler to use. AHDL would be a better choice for applications where you are trying to squeeze maximum performance from the chip in as small as possible package outline.

The design file (XXX.v) can in principle be produced and read by any text editor program. The editor supplied by QUARTUS however uses coloured text to differentiate parts of the file (reserved words are blue, comments green etc) this makes it much easier to spot syntax errors such as a missing END statements and also makes it simpler to read the comments.

The file header

Table 1 shows the first part of the .v-file for the rev counter project that we described last month. The comment field starts after the /* symbols and ends with the */ symbols. The file has a large file header written as commentary giving a short history of the file along with the version numbers. A good tip here is to ensure you always make a back up before any major changes are made to the file.

Following the header the next line begins with the key word *module* followed by the module name (*drehzahl* for 'rev counter'). The inputs and outputs are now defined. It is not strictly necessary to separate all the inputs from all the outputs, they can all be mixed up but it does help to organise the file a bit better if they are kept separate. A comment can be added to the end of each line to give a short description of the signal. Later on the pin numbers can be added in the comment field for information. The actual pin out information is produced in one of the other files.

```
        reg     [8:0]   Next_M;

        reg     [10:0]  V;
        reg     [10:0]  Next_V;

        reg             VRES;
        reg             Next_VRES;

        reg             DZ_IN_1D;
        reg             Next_DZ_IN_1D;

        reg             DZ_IN_2D;
        reg             Next_DZ_IN_2D;

        reg             TOR;
        reg             Next_TOR;

        reg             TOR_D;
        reg             Next_TOR_D;

        reg             TOR_NF;
        reg             Next_TOR_NF;

        reg     [15:0]  LED;
        reg     [15:0]  Next_LED;
        reg     [15:0]  LED_R;
        reg     [15:0]  Next_LED_R;

        reg     [15:0]  LED_Z;

        output  [15:0]  LED_R_OA;
        reg     [15:0]  LED_R_OA;

        output  [15:0]  LED_R_OB;
        reg     [15:0]  LED_R_OB;

        reg     [15:0]  MASK;

reg     [4:0] i;

/*=================================================================*/
```

Inputs and outputs

DZ_IN Ignition input from the vehicle coil. The signal will be filtered and level shifted to TTL signal thresholds.

CLK Global clock input driven from the crystal oscillator output C4_OUT.

C4_IN The crystal oscillator input.

MODE[2..0] The three signals from the three-way DIP switch used to select the different display modes. When there are several signals they can be grouped together under the same name they can then be

Table 2

```
/*=============================================================*/
/* FlipFlops */
always @ (posedge CLK or negedge RESET)
begin
        if (RESET == 0)
        begin
                M               <=      9'd0;
                MRES            <=      1'b0;
                V               <=      11'd0;
                VRES            <=      1'b0;
                DZ_IN_1D<=      1'b0;
                DZ_IN_2D<=      1'b0;
                TOR             <=      1'b0;
                TOR_D           <=      1'b0;
                TOR_NF          <=      1'b0;
                LED             <=      16'd0;
                LED_R           <=      16'd0;

        end
        else
        begin
                M               <=      Next_M;
                MRES            <=      Next_MRES;
                V               <=      Next_V;
                VRES            <=      Next_VRES;
                DZ_IN_1D<=      Next_DZ_IN_1D;
                DZ_IN_2D<=      Next_DZ_IN_2D;
                TOR             <=      Next_TOR;
                TOR_D           <=      Next_TOR_D;
                TOR_NF          <=      Next_TOR_NF;
                LED             <=      Next_LED;
                LED_R           <=      Next_LED_R;
        end
end
/*=============================================================*/
```

Table 3

```
/*=============================================================*/
/* Clock oscillator */
always @ (C4_IN)
begin
        C4_OUT = !C4_IN;
end
/*=============================================================*/
```

Table 4

```
/*=============================================================*/
always @ (M or ZYL)
begin
        if (    (M == 9'd478) && (ZYL == 3'd0)   ||
                (M == 9'd238) && (ZYL == 3'd1)   ||
                (M == 9'd158) && (ZYL == 3'd2)   ||
                (M == 9'd118) && (ZYL == 3'd3)   ||
                (M == 9'd78)  && (ZYL == 3'd4)   ||
                (M == 9'd58)  && (ZYL == 3'd5)   ||
                (M == 9'd38)  && (ZYL == 3'd6)   ||
                (M == 9'd28)  && (ZYL == 3'd7)   )
                        Next_MRES = 1'b1;
        else
                        Next_MRES = 1'b0;
end
/*=============================================================*/
```

referred to individually or as a group: MODE[1] is just one signal, MODE[1..0] is the two lowest bits and MODE[2..0] will be all three bits.

ZYL[2..0] These three inputs from the DIP switch allow the cylinder count of the engine to be selected. This allows the circuit to be used on different engines without the need to reprogram the CPLD.

LICHT reduces the brightness of the LEDs when the vehicle headlights are switched on.

RESET input for the Power-on-Reset network.

C4_OUT The crystal oscillator output.

MRES Debug output pin.

LED_R_OA [15 to 0] The first set of 16 outputs for the LEDs.

LED_R_OB [15 to 0] The second set of 16 outputs for the LEDs.
The output pins can only sink 12 mA so each LED has two outputs connected in parallel to increase the current.

DZ_IN_B The buffered input signal DZ_IN. The two external resistors produce a Schmitt trigger.

All outputs and internal signals are also defined as **reg** (register), but this does not necessarily mean that the signals will be produced by a flipflop.
All the flipflops used in the design are listed in **Table 2**. The signals in the *sensitivity list* (in the brackets after *always*) are the conditions that cause the output signals to change i.e. the positive edge of the clock or the negative edge of the RESET signal. All flipflops outputs are cleared to zero during reset. At the rising clock edge the signals will be updated to the values in the *NEXT_State*. The code for the *NEXT* state assignments is separated from the output logic assignments. This is not essential but is recommended by many Verilog design guides especially for modelling State machines.

The logic functions

The quartz oscillator is defined first; it consists of just a single inverter (**Table 3**).

The lines of equals' signs serve no other purpose than to visually separate each function block. One line of description is sufficient to describe the oscillator. The output signal C4_OUT is simply equal to the inverted input signal C4_IN. The exclamation mark indicates

signal inversion.
In the next description the clock frequency is divided down according to the settings of the 3 DIP switches Z0,Z1 and Z2. A four-stroke engine produces one ignition impulse per cylinder for every two revolutions of the crankshaft (assuming single spark ignition). A four cylinder engine therefore produces two impulses per revolution and an eight cylinder four impulses. To count the input pulses it would be possible to divide them down but this can produce display flicker so it is better to divide the clock frequency.

The statements in **Table 4** describe the timing conditions necessary to generate the reset signal MRES for the 9-bit counter M. Altogether there are eight possible counter values that can generate MRES. Looking at the expression in brackets on the left-hand side the counter output M is compared to a decimal value on the right side of the brackets. The value of this expression will only be 1 when the counter output equals the decimal value otherwise it will be 0. Then comes two ampersands indicating the AND operator and another set of brackets containing the description of three inputs from the 3-way ZYL (CYL.) DIP switches. The value of the switches is compared with three bits representing the decimal number in the range 0 to 7. Only when the statements in both brackets are true will a 1 be generated. Each line has two vertical lines indicating that it is OR'ed with the next line so it only needs the statement on one of the lines to be true before the reset pulse MRES is set to a 1 otherwise it will be 0.

Line 4 for example will be '1' when counter reaches 118 and the DIP switch has the value 3. At the next rising edge of CLK the counter increments to 119 and MRES will change to a '1'. At the next clock the counter will be reset to zero and this will make MRES return to '0'. The counter is now ready to start counting up again at the next clock edge. It therefore counts from 0 to 119 which is 120 clocks altogether.

The behaviour of counter M is defined by the values in **Table 5**. When MRES is High, the next value of M will be 0 otherwise it will have the value M + 1. On the next positive clock edge the counter value will either be incremented or reset to zero, depending on the state of MRES.

Table 6 indicates that the input signal from the ignition coil DZ_IN is simply buffered and output as DZ_IN_B. Two external resistors use the buffered output to build a Schmitt trigger for DZ_IN.
The ignition input signal will not be synchronised to the internal clock frequency. It is necessary to re-clock this signal so that it can be

Table 5

```
/*=================================================================*/
/* Counter M, Reset using MRES, else count up */
always @ (M or MRES)
begin
        if (MRES ==1'b1)
                Next_M = 9'd0;
        else
                Next_M = M + 1'b1 ;
end
/*=================================================================*/
```

Table 6

```
/*=================================================================*/
/* Input signal, counter gating, etc. (TOR = gate) */
always @ (DZ_IN)
begin
        DZ_IN_B = DZ_IN;
end
/*=================================================================*/

/*=================================================================*/
always @ (DZ_IN)
begin
        Next_DZ_IN_1D = DZ_IN;
end
/*=================================================================*/

/*=================================================================*/
always @ (DZ_IN_1D)
begin
        Next_DZ_IN_2D = DZ_IN_1D;
end
/*=================================================================*/

/*=================================================================*/
always @ (DZ_IN_1D or DZ_IN_2D or TOR)
begin
        if (DZ_IN_1D & !DZ_IN_2D)
                Next_TOR = !TOR;
        else
                Next_TOR = TOR;
end
/*=================================================================*/
/*=================================================================*/
always @ (TOR)
begin
        Next_TOR_D = TOR;
end
/*=================================================================*/

/*=================================================================*/
always @ (TOR or TOR_D)
begin
        Next_TOR_NF = !TOR & TOR_D;
end
/*=================================================================*/

/*=================================================================*/
always @ (TOR_NF)
begin
        Next_VRES = TOR_NF;
end
/*=================================================================*/
```

Table 7

```
/*=============================================================*/
/* Counter V */
always @ (V or VRES or MRES or TOR)
begin
        if (VRES == 1'b1)
                Next_V = 11'd0;
        else
        if (!VRES & MRES & TOR)
                Next_V = V + 1'b1;
        else
                Next_V = V ;
end
/*=============================================================*/
```

Table 8

```
/*=============================================================*/
/* LEDs */
/* Mode 0    1000 to 6000 rpm, res. 333  */
/* Mode 1     750 to 4500 rpm, res. 250  */
/* Mode 2    4125 to 6000 rpm, res. 125  */
/* Mode 3    2500 to 10000 rpm, res. 500  */

always @ (LED[15] or MODE or VRES)
begin
        if (    LED[15] && (V == 204) && (MODE == 0) ||
                LED[15] && (V == 272) && (MODE == 1) ||
                LED[15] && (V == 204) && (MODE == 2) ||
                LED[15] && (V == 122) && (MODE == 3) )
                Next_LED[15] = 1'b0;
        else
        if      (       VRES & !LED[15])
                        Next_LED[15] = 1'b1;
        else
                        Next_LED[15] = LED[15];
end
/*=============================================================*/
.
.
/*=============================================================*/
always @ (LED[0] or MODE or VRES)
begin
        if (    LED[0] && (V == 1228) && (MODE == 0) ||
                        LED[0] && (V == 1637) && (MODE == 1) ||
                        LED[0] && (V == 297) && (MODE == 2) ||
                        LED[0] && (V == 491) && (MODE == 3) )
                        Next_LED[0] = 1'b0;
        else
        if      (       VRES & !LED[0])
                        Next_LED[0] = 1'b1;
        else
                        Next_LED[0] = LED[0];
end
/*=============================================================*/
```

Table 9

```
/*=============================================================*/
/* Copying into output register */
always @ (LED_R or LED or TOR_NF)
begin
        if ( TOR_NF )
                Next_LED_R[15:0] = LED[15:0];
        else
                Next_LED_R[15:0] = LED_R[15:0];
end
/*=============================================================*/
```

used in the CPLD. The input signal is sampled by the flipflop DZ_IN_1D the first time and then DZ_IN_2D the second time. TOR toggles only if DZ_IN_1D is high and DZ_IN_2D is low which occurs at the rising edge of the input signal. The circuit operates like a digital differentiator. TOR therefore toggles between high and low for every rising edge of the ignition input signal. TOR_D is the TOR signal delayed by one clock period. TOR_NF detects the negative edge of TOR and is used to control the counter and registers.

Table 7 defines an 11-bit counter V[10..0]. A synchronous reset is generated by the VRES signal. The counter only increments when both MRES and TOR are high. The MRES pulse is one clock period wide and is used to reset the clock divider. M[] is one clock period wide. V[] does not count at every clock edge but only those gated by MRES. The frequency of MRES is defined by the 3 way DIPswitch ZYL[ ]. VRES is the signal TOR_NF delayed by one clock period.

The display

Table 8 indicates that each display LED has a flipflop assigned to it that is set to 1 by VRES. The DIP switch setting MODE [] controls the counter value at which the flipflop will be reset. The comments indicate displayed rev ranges and resolution (revs per LED). Thanks to the adjustable prescaler these settings are independent of the type of engine in use. To avoid repetition only LEDs 15 and 0 are shown. **Table 9** shows that at the negative going edge of TOR (TOR_NF goes high) the contents of flipflops LED [n:0] are transferred into the LED_R[n:0].

To sum up

DZ_IN is the impulse from the ignition coil and is sampled twice. The positive edge triggers the TOR toggle flipflop. During the high phase of TOR the counter V[] counts using MRES as the clock source. M[] is a selectable divider which is adjusted to accommodate different engine types. The FF LED[] is reset when counter V reaches a user–defined maximum count. The negative edge of TOR loads the data from LED[] to LED_R[]. One clock period later V[] and LED[] are set to 1. As soon as TOR goes high again the whole process begins again and repeats.

From this description it can be seen that the time period between every other ignition pulse is counted to work out the engine speed, the period between counting is used to latch data and reset the circuit. LED display data is stored in an intermediate latch so

that it simply gets overwritten by the latest value at the falling edge of TOR. An earlier version of this circuit did not store the data so the LEDs flickered.

The circuit described so far simply displays the engine speed on a row of LEDs but there are also a couple of features that we have not yet explored. Firstly the LEDs brightness is automatically reduced when the vehicle lights are switched on. This is achieved by the first two lines of the FOR loop in **Table 10**. When the lights are off the first line of the expression is true and LED_Z[i] is the same as LED_R[i]. When the vehicle lights are switched on the second line is now true and bit 3 of the M counter output is included in the expression so the LEDs are now switched by this square wave and their brightness is reduced.

Secondly, there are eight possible display modes each with a different range of engine speeds so without any markers it can be difficult to interpret the display. To make the display more readable each LED representing a thousand revs on the scale glows constantly. As the engine speed increases and reaches these markers they switch to full brightness. The masks for these marker LEDs is included in the CASE declaration where the row of 16 LEDs is represented by a line of ones and

Table 10

```
/*=================================================================*/
/* LEDs for thousands should light dimly, M0, M1 und M2 create Duty
Cycle */
/* M0 and M1 = 1/4, for lights off, M0, M1 and M2 = 1/8, for lights
on */
/* Mode 0    1000 to 6000 rpm, res. 333, LED 0,3,6,9,12,15 */
/* Mode 1     750 to 4500 rpm, res. 250, LED 1,5,9,13  */
/* Mode 2    4125 to 6000 rpm, res. 125, LED 7,15   */
/* Mode 3    2500 to 10000 rpm, res. 500, LED 1,3,5,7,9,11,13,15 */

always @ (LED_R or LICHT or M or MODE)
begin
        case (MODE)
                0:              MASK = 16'b1001001001001001;
                1:              MASK = 16'b0010001000100010;
                2:              MASK = 16'b1000000010000000;
                3:              MASK = 16'b1010101010101010;
                default: MASK = 16'b0000000000000000;
        endcase

  for (i=0;i<=15;i=i+1)
        begin
                LED_Z[i]        =  LED_R[i] & !LICHT
                                |  LED_R[i] &  LICHT & M[2]
                                |  MASK[i]  & !LICHT & M[0] & M[1]
                                |  MASK[i]  &  LICHT & M[0] & M[1]
& M[2];
        end
end
/*=================================================================*/
```

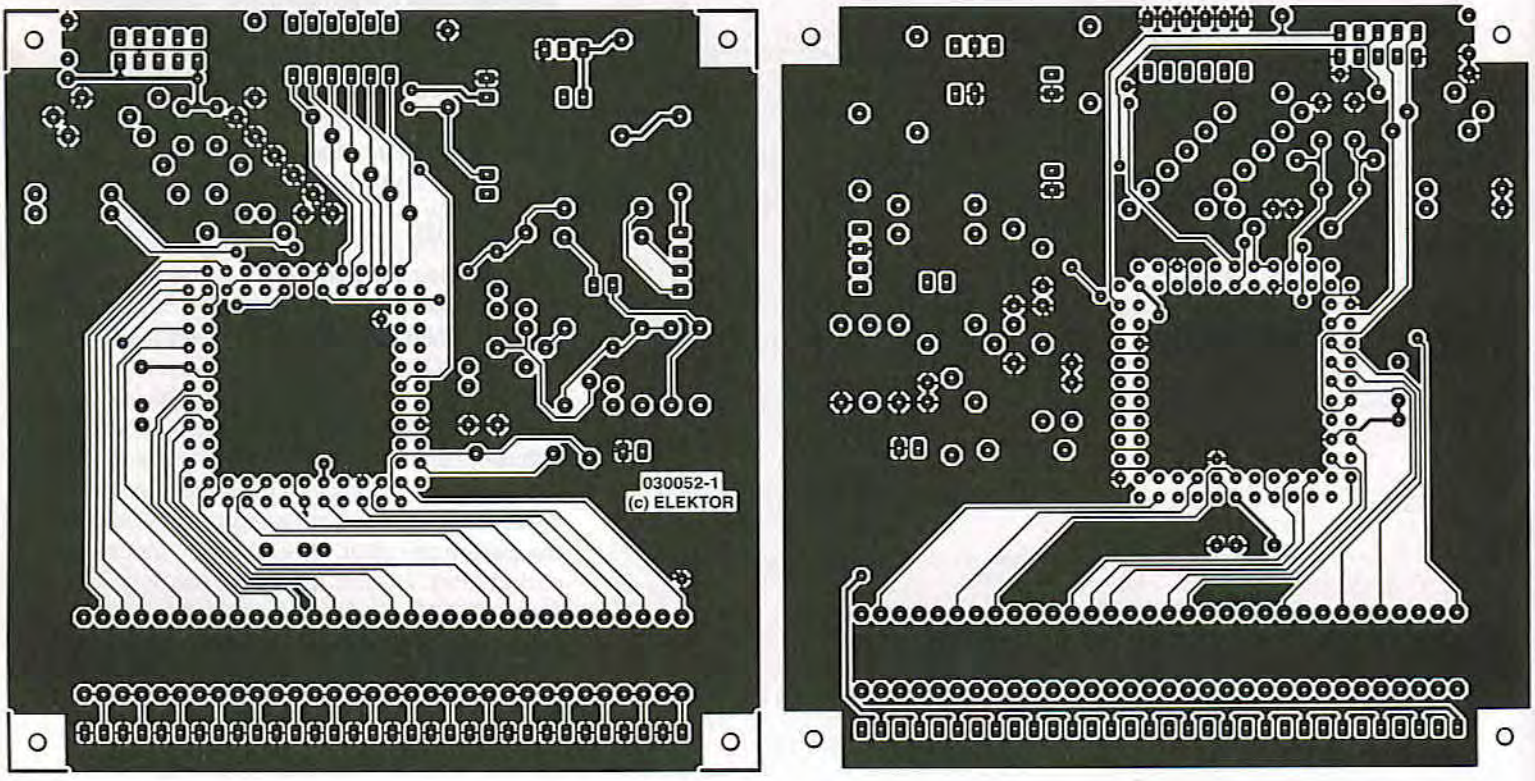


Figure 1. The double-sided PCB layout for the CPLD rev counter.

Table 11

```
/*=================================================================*/
always @(LED_Z)
begin
  for (i=0;i<=15;i=i+1)
  begin
    if (LED_Z[i])
      LED_R_OA[i] = 1'b0;
    else
      LED_R_OA[i] = 1'bz;
  end
end
/*=================================================================*/
```

Table 12

```
/*=================================================================*/
always @(LED_R or LICHT)
begin
  for (i=0;i<=15;i=i+1)
  begin
    if (LED_R[i] & !LICHT)
      LED_R_OB[i] = 1'b0;
    else
      LED_R_OB[i] = 1'bz;
  end
end
/*=================================================================*/
endmodule
```

zeroes, each 1 indicates a marker LED. The bit pattern from the three MODE inputs is used to select the correct mask. The mask is included in the last two lines of the FOR loop in **table 10** to control the markers according to the state of the vehicle lights (LICHT).

In the expression in **Table 11** the open-collector outputs LED_R_OA[i] are assigned to the output signals LED_Z[i]. Open-collector outputs can only sink current to ground; they cannot supply any output current to the load. Connecting two outputs in parallel effectively shares the current between the two outputs.

Finally, **Table 12** defines the function of the second row of outputs LED_R_OB[i]. Theses outputs are driven directly from LED_R[i], they do not output any thousand rpm markers. These outputs are also switched off completely when the vehicle lights are switched on. The key word *endmodule* is used to indicate the end of the design files.

If this short introduction to CPLD programming has whetted your appetite you will be pleased to know that we have more QUARTUS and Verilog workshops planned. Also in the pipeline is a fully-fledged CPLD evaluation board ideal for prototyping new designs.

(030052-2)