Appendix A
The Powers of 2

As we are dealing with binary values, it is useful to remember the values of the most significant powers of 2. Below are the smallest:

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

We see that $2^{10} = 1024$ is equal to approximately $10^3 = 1000$. Let the prefix K (meaning Kilo) indicate the value 1024 even though the *International Electrotechnical Commission* (IEC) standard establishes the prefix Kibi (from Kilo binary):

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

Similarly 2^{20} let M (Mega) indicate (Mebi), equal to about 10^6:

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

Let G (Giga) (10^9) indicate (Gibi), which is equal to 2^{30}:

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

Let T (Tera) indicate 2^{40} (Tebi), about 10^{12}.

G. Donzellini et al., *Introduction to Digital Systems Design*,
https://doi.org/10.1007/978-3-319-92804-3

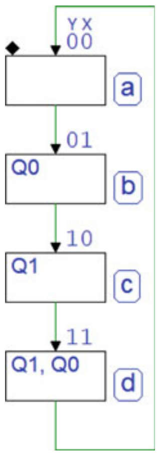
To be thorough, a table including all the prefixes follows.

IEC prefix		Representations			Customary prefix	
Name	Symbol	Base 2	Base 1024	Base 10	Name	Symbol
Kibi	Ki	2^{10}	1024^1	$= 1.024 \times 10^3$	Kilo	k or K
Mebi	Mi	2^{20}	1024^2	$\approx 1.049 \times 10^6$	Mega	M
Gibi	Gi	2^{30}	1024^3	$\approx 1.074 \times 10^9$	Giga	G
Tebi	Ti	2^{40}	1024^4	$\approx 1.100 \times 10^{12}$	Tera	T
Pebi	Pi	2^{50}	1024^5	$\approx 1.126 \times 10^{15}$	Peta	P
Exbi	Ei	2^{60}	1024^6	$\approx 1.153 \times 10^{18}$	Exa	E
Zebi	Zi	2^{70}	1024^7	$\approx 1.181 \times 10^{21}$	Zetta	Z
Yobi	Yi	2^{80}	1024^8	$\approx 1.209 \times 10^{24}$	Yotta	Y

Appendix B
State Diagrams

The *"State Diagram"* is one of several methods for describing the behavior and performing the synthesis of a FSM, alternative to the ASM method developed in the book. It is quite similar to the former and still used in textbooks and documentation materials.

As the ASM, it describes the machine state-to-state transitions and outputs. In the following, we show how to convert *ASM diagrams* into *State Diagrams*, using as examples state machines presented in the book as ASM.

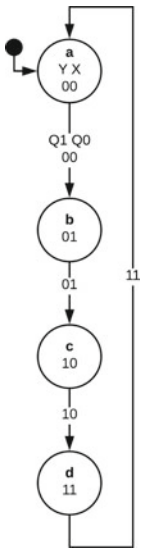


A first example of a *State Diagram* (right) is the module-4 counter described in Sect. 7.2.1 (left).

The circles, or *"bubbles"* represent the states, which are connected by *lines (arches)* with arrows that point to the direction of the transition.

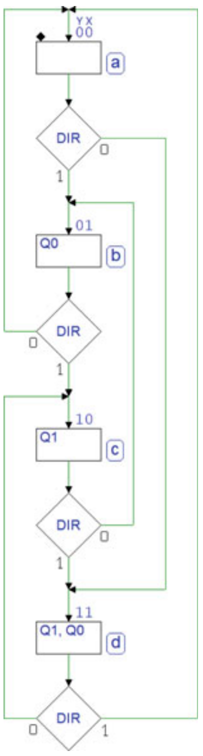
The states are identified by a letter (a, b, c, d); each circle contains also the corresponding binary code. The lines are labelled with the Q1 Q0 output values.

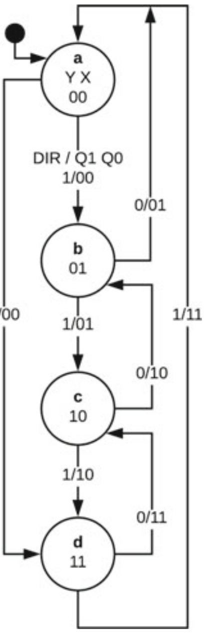
In other representations, in the case of a *Moore* machine, the outputs could also be written inside the state circles.



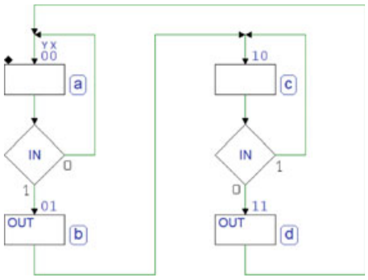
If there are inputs conditioning the state transitions, they are written by the lines, using digits separated by a slash. The first set of digits (before the slash) indicates the value of the inputs: with only one input there will be two lines exiting each state.

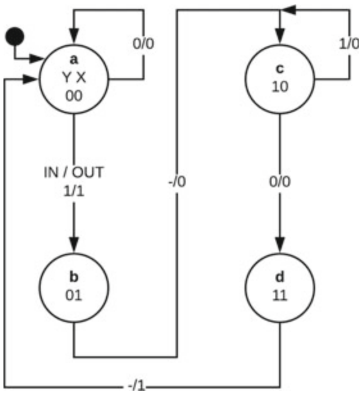
The figure below shows the bidirectional counter introduced in Sect. 7.2.2. There is now an input *DIR* that sets the count direction. If *DIR* = 1 the count is up, if *DIR* = 0 the count is down. The couple of digits after the slash shows the outputs, as in the previous case.



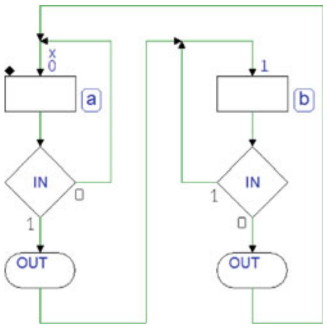


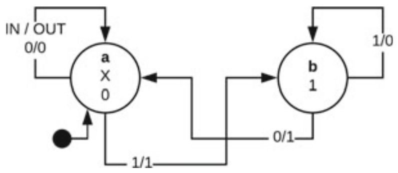
The edge detector (Sect. 7.2.2) is another example of a *Moore* machine. A line that starts and ends in the same state indicates a waiting loop.



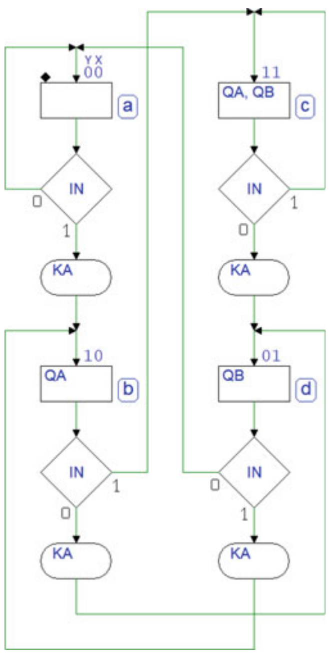


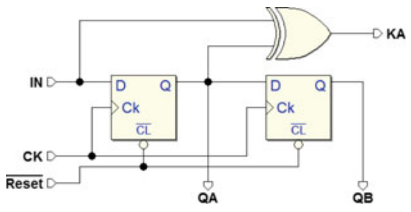
In a *Mealy* machine, such as the edge detector with conditioned outputs (Sect. 7.2.3), the outputs are function of both state and inputs. The *State Diagram* indicates, as before, the outputs in the lines after the slash, but in this case, outputs in the same state are different when conditioned by an input.

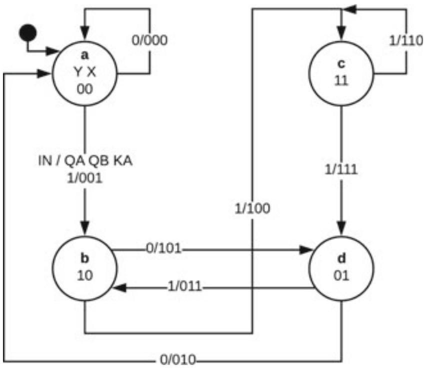




The figure below is another example of a *Mealy* machine. It represents the algorithm of the circuit seen in Sect. 7.3.9.





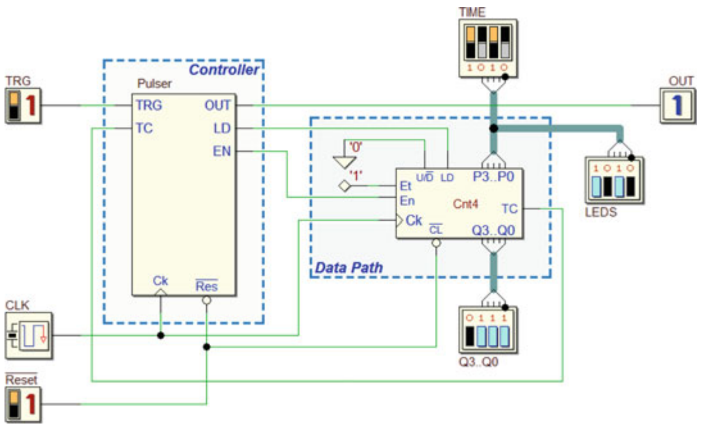


Appendix C
VHDL Code

In this pages, we list the VHDL codes described in Chap. 9. Listings are almost complete. The omitted parts can be obtained by exporting with *Deeds* the projects available in the digital contents pages (on *Deeds* Web site).

C.1 Code from the Pulse Generator Example

C.1.1 *Schematic*



C.1.2 *Top Entity*

```
--------------------------------------------------------------------------------
-- Deeds (Digital Electronics Education and Design Suite)
-- VHDL Code generated on (10/03/2018, 14:50:44)
--      by the Deeds (Digital Circuit Simulator)(Deeds-DcS)
--      Ver. 2.10.300 (Jan 19, 2018)
-- Copyright(c)2002-2018 University of Genoa, Italy
--      Web Site:  https://www.digitalelectronicsdeeds.com
--------------------------------------------------------------------------------
-- Code compiled for: "DE0-CV Board"
-- Chip FPGA: Intel/Altera Cyclone(r) V (5CEBA4F23C7)
-- Proprietary EDA Tool: Quartus(r) II (Ver = 12.1sp1)
--------------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.all;

ENTITY Pulser_DE0CV IS
  PORT(
    ------------------------------------------------------------------> Clocks:
    iCLOCK_50MHz: IN  std_logic;   --> PIN_V15 --> "iCLK": 2 Hz (Sw[09], LEDR[09]
                                   --> Key[01] for Slow Clock Mode)
    ------------------------------------------------------------------> Inputs:
    iTRG:         IN  std_logic;   --> PIN_M6,   PButton: Key[03] H (if pressed)
    inReset:      IN  std_logic;   --> PIN_P22,  PButton: Reset L (if pressed)
    iTIME_03:     IN  std_logic;   --> PIN_T12,  Switch: Sw[03]
    iTIME_02:     IN  std_logic;   --> PIN_T13,  Switch: Sw[02]
    iTIME_01:     IN  std_logic;   --> PIN_V13,  Switch: Sw[01]
    iTIME_00:     IN  std_logic;   --> PIN_U13,  Switch: Sw[00]
    -----------------------------------------------------------------> Outputs:
    oOUT:         OUT std_logic;   --> PIN_L2,   Red Led: LEDR[08]
    oLEDS_03:     OUT std_logic;   --> PIN_Y3,   Red Led: LEDR[03]
    oLEDS_02:     OUT std_logic;   --> PIN_W2,   Red Led: LEDR[02]
    oLEDS_01:     OUT std_logic;   --> PIN_AA1,  Red Led: LEDR[01]
    oLEDS_00:     OUT std_logic;   --> PIN_AA2,  Red Led: LEDR[00]
    oQ3_Q0_03:    OUT std_logic;   --> PIN_U1,   Red Led: LEDR[07]
    oQ3_Q0_02:    OUT std_logic;   --> PIN_U2,   Red Led: LEDR[06]
    oQ3_Q0_01:    OUT std_logic;   --> PIN_N1,   Red Led: LEDR[05]
    oQ3_Q0_00:    OUT std_logic;   --> PIN_N2,   Red Led: LEDR[04]
    ------------------------------------------------------------> Added Inputs:
    iSLOW_Sw09:   IN  std_logic;   --> PIN_AB12 --> "iCLK"  Sw[09]
    iPULSE_Key01: IN  std_logic;   --> PIN_W9   --> "iCLK"  Key[01]
    -----------------------------------------------------------> Added Outputs:
    oCLOCK_LedR09: OUT std_logic;  --> PIN_L1,   Red Led:  LEDR[09]
    ---------------------------------------------------------> Default Outputs:
    oPIN_U21:     OUT std_logic;   --> PIN_U21,  Seven Segm. Display: HEX 0 0 (a)
    oPIN_V21:     OUT std_logic;   --> PIN_V21,  Seven Segm. Display: HEX 0 1 (b)
    oPIN_W22:     OUT std_logic;   --> PIN_W22,  Seven Segm. Display: HEX 0 2 (c)
    oPIN_W21:     OUT std_logic;   --> PIN_W21,  Seven Segm. Display: HEX 0 3 (d)
    oPIN_Y22:     OUT std_logic;   --> PIN_Y22,  Seven Segm. Display: HEX 0 4 (e)
    oPIN_Y21:     OUT std_logic;   --> PIN_Y21,  Seven Segm. Display: HEX 0 5 (f)
    oPIN_AA22:    OUT std_logic;   --> PIN_AA22, Seven Segm. Display: HEX 0 6 (g)
    oPIN_AA20:    OUT std_logic;   --> PIN_AA20, Seven Segm. Display: HEX 1 0 (a)
    oPIN_AB20:    OUT std_logic;   --> PIN_AB20, Seven Segm. Display: HEX 1 1 (b)
    oPIN_AA19:    OUT std_logic;   --> PIN_AA19, Seven Segm. Display: HEX 1 2 (c)
    oPIN_AA18:    OUT std_logic;   --> PIN_AA18, Seven Segm. Display: HEX 1 3 (d)
    oPIN_AB18:    OUT std_logic;   --> PIN_AB18, Seven Segm. Display: HEX 1 4 (e)
    oPIN_AA17:    OUT std_logic;   --> PIN_AA17, Seven Segm. Display: HEX 1 5 (f)
    oPIN_U22:     OUT std_logic;   --> PIN_U22,  Seven Segm. Display: HEX 1 6 (g)
    oPIN_Y19:     OUT std_logic;   --> PIN_Y19,  Seven Segm. Display: HEX 2 0 (a)
    oPIN_AB17:    OUT std_logic;   --> PIN_AB17, Seven Segm. Display: HEX 2 1 (b)
    oPIN_AA10:    OUT std_logic;   --> PIN_AA10, Seven Segm. Display: HEX 2 2 (c)
    oPIN_Y14:     OUT std_logic;   --> PIN_Y14,  Seven Segm. Display: HEX 2 3 (d)
    oPIN_V14:     OUT std_logic;   --> PIN_V14,  Seven Segm. Display: HEX 2 4 (e)
    oPIN_AB22:    OUT std_logic;   --> PIN_AB22, Seven Segm. Display: HEX 2 5 (f)
    oPIN_AB21:    OUT std_logic;   --> PIN_AB21, Seven Segm. Display: HEX 2 6 (g)
    oPIN_Y16:     OUT std_logic;   --> PIN_Y16,  Seven Segm. Display: HEX 3 0 (a)
    oPIN_W16:     OUT std_logic;   --> PIN_W16,  Seven Segm. Display: HEX 3 1 (b)
    oPIN_Y17:     OUT std_logic;   --> PIN_Y17,  Seven Segm. Display: HEX 3 2 (c)
    oPIN_V16:     OUT std_logic;   --> PIN_V16,  Seven Segm. Display: HEX 3 3 (d)
    oPIN_U17:     OUT std_logic;   --> PIN_U17,  Seven Segm. Display: HEX 3 4 (e)
    oPIN_V18:     OUT std_logic;   --> PIN_V18,  Seven Segm. Display: HEX 3 5 (f)
    oPIN_V19:     OUT std_logic;   --> PIN_V19,  Seven Segm. Display: HEX 3 6 (g)
    oPIN_U20:     OUT std_logic;   --> PIN_U20,  Seven Segm. Display: HEX 4 0 (a)
    oPIN_Y20:     OUT std_logic;   --> PIN_Y20,  Seven Segm. Display: HEX 4 1 (b)
    oPIN_V20:     OUT std_logic;   --> PIN_V20,  Seven Segm. Display: HEX 4 2 (c)
    oPIN_U16:     OUT std_logic;   --> PIN_U16,  Seven Segm. Display: HEX 4 3 (d)
    oPIN_U15:     OUT std_logic;   --> PIN_U15,  Seven Segm. Display: HEX 4 4 (e)
    oPIN_Y15:     OUT std_logic;   --> PIN_Y15,  Seven Segm. Display: HEX 4 5 (f)
    oPIN_P9:      OUT std_logic;   --> PIN_P9,   Seven Segm. Display: HEX 4 6 (g)
    oPIN_N9:      OUT std_logic;   --> PIN_N9,   Seven Segm. Display: HEX 5 0 (a)
    oPIN_M8:      OUT std_logic;   --> PIN_M8,   Seven Segm. Display: HEX 5 1 (b)
    oPIN_T14:     OUT std_logic;   --> PIN_T14,  Seven Segm. Display: HEX 5 2 (c)
    oPIN_P14:     OUT std_logic;   --> PIN_P14,  Seven Segm. Display: HEX 5 3 (d)
    oPIN_C1:      OUT std_logic;   --> PIN_C1,   Seven Segm. Display: HEX 5 4 (e)
    oPIN_C2:      OUT std_logic;   --> PIN_C2,   Seven Segm. Display: HEX 5 5 (f)
    oPIN_W19:     OUT std_logic    --> PIN_W19,  Seven Segm. Display: HEX 5 6 (g)
    );
```

```
END Pulser_DE0CV;
ARCHITECTURE structural OF Pulser_DE0CV IS
  ------------------------------------------------------------------> Components:
  COMPONENT ClockScaler IS
    PORT( iMClk: IN  std_logic;   -- Master Clock
          iH4:   IN  std_logic;   -- iH4..iH0 = "high" frequency selection
          iH3:   IN  std_logic;
          iH2:   IN  std_logic;
          iH1:   IN  std_logic;
          iH0:   IN  std_logic;
          iL3:   IN  std_logic;   -- iL3..iL0 = "low" frequency selection
          iL2:   IN  std_logic;   --                  and Button Modes
          iL1:   IN  std_logic;
          iL0:   IN  std_logic;
          iSwch: IN  std_logic;   -- Switch
          iBut:  IN  std_logic;   -- Button for manual pulsed Clock
          oSClk: OUT std_logic;   -- Output Clock
          oLed:  OUT std_logic  -- Slow "Clock Pulse" Led
          );
  END COMPONENT;
  --
  COMPONENT Counter4b IS
    PORT( Ck : IN std_logic;
          nCL: IN std_logic;
          LD : IN std_logic;
          ENP: IN std_logic;
          ENT: IN std_logic;
          UD : IN std_logic;
          P3 : IN std_logic;
          P2 : IN std_logic;
          P1 : IN std_logic;
          P0 : IN std_logic;
          Q3 : OUT std_logic;
          Q2 : OUT std_logic;
          Q1 : OUT std_logic;
          Q0 : OUT std_logic;
          Tc : OUT std_logic );
  END COMPONENT;
  -----------------------------------------------------> Finite State Machine(s):
  COMPONENT Pulser_EN IS
    PORT( ---------------------------------->Clock & Reset:
          Ck:    IN std_logic;
          Reset: IN std_logic;
          ---------------------------------->Inputs:
          i_TRG: IN std_logic;
          i_TC:  IN std_logic;
          ---------------------------------->Outputs:
          o_OUT: OUT std_logic;
          o_LD:  OUT std_logic;
          o_EN:  OUT std_logic
          ------------------------------------------
          );
  END COMPONENT;
  ---------------------------------------------------------------------> Signals:
  SIGNAL S001: std_logic;
  SIGNAL S002: std_logic;
  SIGNAL S003: std_logic;
  SIGNAL S004: std_logic;
  SIGNAL S005: std_logic;
  SIGNAL S006: std_logic;
  SIGNAL S007: std_logic;
  SIGNAL S008: std_logic;
  SIGNAL S009: std_logic;
  SIGNAL S010: std_logic;
  SIGNAL S011: std_logic;
  SIGNAL S012: std_logic;
  SIGNAL S013: std_logic;
  SIGNAL S014: std_logic;
  SIGNAL S015: std_logic;
  SIGNAL S016: std_logic;
  SIGNAL S017: std_logic;
  ---------------------------------------------------------------> Added Signals:
  SIGNAL SSLOW_Sw09: std_logic;
  SIGNAL SPULSE_Key01: std_logic;
  SIGNAL iCLK: std_logic;
  SIGNAL SCLOCK_LedR09: std_logic;

BEGIN -- structural
  -----------------------------------------------------------------------> Input:
  S002 <= NOT iTRG;
  S005 <= iCLK;
  S007 <= inReset;
  S014 <= iTIME_00;
  S015 <= iTIME_01;
```

```vhdl
  S016 <= iTIME_02;
  S017 <= iTIME_03;
  ----------------------------------------------------------------------> Output:
  oOUT <= S008;
  oLEDS_00  <= S014;
  oLEDS_01  <= S015;
  oLEDS_02  <= S016;
  oLEDS_03  <= S017;
  oQ3_Q0_00  <= S010;
  oQ3_Q0_01  <= S011;
  oQ3_Q0_02  <= S012;
  oQ3_Q0_03  <= S013;
  -------------------------------------------------------------------> Constants:
  S004 <= '0';
  S001 <= '1';
  oPIN_U21   <= '1';
  oPIN_V21   <= '1';
  oPIN_W22   <= '1';
  oPIN_W21   <= '1';
  oPIN_Y22   <= '1';
  oPIN_Y21   <= '1';
  oPIN_AA22  <= '1';
  oPIN_AA20  <= '1';
  oPIN_AB20  <= '1';
  oPIN_AA19  <= '1';
  oPIN_AA18  <= '1';
  oPIN_AB18  <= '1';
  oPIN_AA17  <= '1';
  oPIN_U22   <= '1';
  oPIN_Y19   <= '1';
  oPIN_AB17  <= '1';
  oPIN_AA10  <= '1';
  oPIN_Y14   <= '1';
  oPIN_V14   <= '1';
  oPIN_AB22  <= '1';
  oPIN_AB21  <= '1';
  oPIN_Y16   <= '1';
  oPIN_W16   <= '1';
  oPIN_Y17   <= '1';
  oPIN_V16   <= '1';
  oPIN_U17   <= '1';
  oPIN_V18   <= '1';
  oPIN_V19   <= '1';
  oPIN_U20   <= '1';
  oPIN_Y20   <= '1';
  oPIN_V20   <= '1';
  oPIN_U16   <= '1';
  oPIN_U15   <= '1';
  oPIN_Y15   <= '1';
  oPIN_P9    <= '1';
  oPIN_N9    <= '1';
  oPIN_M8    <= '1';
  oPIN_T14   <= '1';
  oPIN_P14   <= '1';
  oPIN_C1    <= '1';
  oPIN_C2    <= '1';
  oPIN_W19   <= '1';
  -----------------------------------------------------------> Component Mapping:
  SSLOW_Sw09 <= iSLOW_Sw09;
  SPULSE_Key01 <= iPULSE_Key01;
  oCLOCK_LedR09 <= SCLOCK_LedR09;
  ClockScaler_iCLK: ClockScaler PORT MAP (
      iCLOCK_50MHz, '1', '0', '1', '0', '0', '1', '1', '1', '1',
      SSLOW_Sw09, SPULSE_Key01, iCLK, SCLOCK_LedR09 );

  C680: Counter4b PORT MAP ( S005, S007, S003, S006, S001, S004, S017, S016,
                             S015, S014, S013, S012, S011, S010, S009 );
  C704: Pulser_EN PORT MAP ( S005, S007, S002, S009, S008, S003, S006 );
END structural;
```

C.1.3 Components

```vhdl
-----------------------------------------------------------------------------------
-- Deeds (Digital Electronics Education and Design Suite)
-- VHDL Code generated on (10/03/2018, 14:50:44)
--      by the Deeds (Digital Circuit Simulator)(Deeds-DcS)
--      Ver. 2.10.300 (Jan 19, 2018)
-- Copyright(c)2002-2018 University of Genoa, Italy
--      Web Site:  https://www.digitalelectronicsdeeds.com
-----------------------------------------------------------------------------------
-- Code compiled for: "DE0-CV Board"
-- Chip FPGA: Intel/Altera Cyclone(r) V (5CEBA4F23C7)
-- Proprietary EDA Tool: Quartus(r) II (Ver = 12.1sp1)
---------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

ENTITY Counter4b IS
  PORT( Ck : IN std_logic;
        nCL: IN std_logic;
        LD : IN std_logic;
        ENP: IN std_logic;
        ENT: IN std_logic;
        UD : IN std_logic;
        P3 : IN std_logic;
        P2 : IN std_logic;
        P1 : IN std_logic;
        P0 : IN std_logic;
        Q3 : OUT std_logic;
        Q2 : OUT std_logic;
        Q1 : OUT std_logic;
        Q0 : OUT std_logic;
        Tc : OUT std_logic );
END Counter4b;

--------------------------------------------------------------------
ARCHITECTURE behavioral OF Counter4b IS
BEGIN
  Count4b: PROCESS( Ck, nCL, ENP, ENT, UD )
  variable aCnt: unsigned( 3 downto 0 );
  BEGIN
    if    (nCL = '0') then            aCnt := (others =>'0');
    elsif (nCL = '1') then
      if (Ck'event) AND (Ck='1') then
        if    (LD = '1') then         aCnt := (P3 & P2 & P1 & P0); -- Load
        elsif (LD = '0') then
          if  (ENP = '1') and (ENT = '1')then
            if    (UD = '1') then
              if (aCnt < "1111") then aCnt := aCnt + 1;
              else                    aCnt := (others =>'0');
              end if;
            elsif (UD = '0') then
              if (aCnt > "0000") then aCnt := aCnt - 1;
              else                    aCnt := (others =>'1');
              end if;
            else                      aCnt := (others =>'X'); -- (UD: Unknown)
            END IF;
          elsif not((ENP ='0')or
                    (ENT ='0') ) then aCnt := (others =>'X'); -- (ENP: Unknown)
          END IF;
        else                          aCnt := (others =>'X'); -- (LD: Unknown)
        END IF;
      END IF;
    else                              aCnt := (others =>'X'); -- (nCL: Unknown)
    END IF;
    --
    Tc <= ENT and (    (aCnt(3) and aCnt(2) and aCnt(1) and aCnt(0) and UD) or
                   (not(aCnt(3) or  aCnt(2) or  aCnt(1) or  aCnt(0) or  UD)) );
    --
    Q3 <= aCnt(3);
    Q2 <= aCnt(2);
    Q1 <= aCnt(1);
    Q0 <= aCnt(0);
  END PROCESS;
END behavioral;
--------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

-- Clock Scaler (Altera DE1, DE2 and DE2-115 version, master clock = 50 MHz)
ENTITY ClockScaler IS
  PORT( iMClk: IN  std_logic;   -- Master Clock
      iH4:   IN  std_logic;   -- iH4..iH0 = "high" fr. sel.
      iH3:   IN  std_logic;
      iH2:   IN  std_logic;
      iH1:   IN  std_logic;
      iH0:   IN  std_logic;
      iL3:   IN  std_logic;   -- iL3..iL0 = "low" freq. sel.
      iL2:   IN  std_logic;   --              and Button Modes
      iL1:   IN  std_logic;
      iL0:   IN  std_logic;
      iSwch: IN  std_logic;   -- Switch
      iBut:  IN  std_logic;   -- Button for manual pulsed Clock
      oSClk: OUT std_logic;   -- Output Clock
      oLed:  OUT std_logic  -- Slow "Clock Pulse" Led
      );
  END ClockScaler;

--------------------------------------------------------------------
ARCHITECTURE behavioral OF ClockScaler IS
BEGIN
   --   ...omissis... (see the complete code generated by Deeds)
END behavioral;
```

C.1.4 Finite State Machine

```
--------------------------------------------------------------------------------
-- Deeds (Digital Electronics Education and Design Suite)
-- VHDL Code generated on (10/03/2018, 14:50:44)
--       by the Deeds (Finite State Machine Simulator)(Deeds-FsM)
--       Ver. 2.10.300 (Jan 19, 2018)
-- Copyright(c)2002-2018 University of Genoa, Italy
--       Web Site:  https://www.digitalelectronicsdeeds.com
--------------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Pulser_EN IS
  PORT( ---------------------------------->Clock & Reset:
        Ck:    IN std_logic;
        Reset: IN std_logic;
        ---------------------------------->Inputs:
        i_TRG: IN std_logic;
        i_TC:  IN std_logic;
        ---------------------------------->Outputs:
        o_OUT: OUT std_logic;
        o_LD:  OUT std_logic;
        o_EN:  OUT std_logic );
END Pulser_EN;

ARCHITECTURE behavioral OF Pulser_EN IS        -- (Behavioral Description)
  TYPE states is ( state_rs,
                   state_wt,
                   state_ps,
                   dummy_11 );
  SIGNAL State,
         Next_State: states;
BEGIN
  -- Next State Combinational Logic ---------------------------------
  FSM: process( State, i_TRG, i_TC )
  begin
    CASE State IS
      when state_wt =>
                  if (i_TRG = '1') then
                    Next_State <= state_ps;
                  else
                    Next_State <= state_wt;
                  end if;
      when state_ps =>
                  if (i_TC = '1') then
                    Next_State <= state_rs;
                  else
                    Next_State <= state_ps;
                  end if;
      when state_rs =>
                  if (i_TRG = '1') then
                    Next_State <= state_rs;
                  else
                    Next_State <= state_wt;
                  end if;
      when OTHERS =>
                  Next_State <= state_rs;
    END case;
  end process;

  -- State Register --------------------------------------------------
  REG: process( Ck, Reset )
  begin
    if (Reset = '0') then
              State <= state_rs;
    elsif rising_edge(Ck) then
              State <= Next_State;
```

```vhdl
      end if;
  end process;

  -- Outputs Combinational Logic ----------------------------------
  OUTPUTS: process( State, i_TRG, i_TC )
  begin
    -- Set output defaults:
    o_OUT <= '0';
    o_LD <= '0';
    o_EN <= '0';

    -- Set output as function of current state and input:
    CASE State IS
      when state_wt =>
                  o_LD <= '1';
      when state_ps =>
                  o_OUT <= '1';
                  if (i_TC = '0') then
                    o_EN <= '1';
                  end if;
      when OTHERS =>
                  o_OUT <= '0';
                  o_LD <= '0';
                  o_EN <= '0';
    END case;
  end process;
END behavioral;
```

C.2 Other VHDL Examples

C.2.1 Decoder

```vhdl
----------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

ENTITY Decoder_2_4 IS
  PORT( A1: IN  std_logic;
        A0: IN  std_logic;
        EN: IN  std_logic;
        Y0: OUT std_logic;
        Y1: OUT std_logic;
        Y2: OUT std_logic;
        Y3: OUT std_logic );
END Decoder_2_4;

ARCHITECTURE behavioral OF Decoder_2_4 IS
  SIGNAL aNumber: std_logic_vector( 2 downto 0 );
BEGIN
  aNumber <= EN & A1 & A0;
  with aNumber select
    Y0 <= '0' when "000", '0' when "001",
          '0' when "010", '0' when "011",
          '1' when "100", '0' when "101",
          '0' when "110", '0' when "111", 'X' when others;
  with aNumber select
    Y1 <= '0' when "000", '0' when "001",
          '0' when "010", '0' when "011",
          '0' when "100", '1' when "101",
          '0' when "110", '0' when "111", 'X' when others;
  with aNumber select
    Y2 <= '0' when "000", '0' when "001",
          '0' when "010", '0' when "011",
          '0' when "100", '0' when "101",
          '1' when "110", '0' when "111", 'X' when others;
  with aNumber select
    Y3 <= '0' when "000", '0' when "001",
          '0' when "010", '0' when "011",
          '0' when "100", '0' when "101",
          '0' when "110", '1' when "111", 'X' when others;
END behavioral;
```

C.2.2 Multiplexer

```vhdl
----------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

ENTITY Multiplexer_4_1 IS
  PORT( I0: IN  std_logic;
        I1: IN  std_logic;
        I2: IN  std_logic;
        I3: IN  std_logic;
        S1: IN  std_logic;
        S0: IN  std_logic;
         Q: OUT std_logic );
END Multiplexer_4_1;

ARCHITECTURE behavioral OF Multiplexer_4_1 IS
BEGIN
  Q <= I0 when ((S1 = '0') and (S0 = '0')) else
       I1 when ((S1 = '0') and (S0 = '1')) else
       I2 when ((S1 = '1') and (S0 = '0')) else
       I3 when ((S1 = '1') and (S0 = '1')) else 'X';
END behavioral;
```

C.2.3 Demultiplexer

```vhdl
---------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
ENTITY Demultiplexer_1_4 IS
  PORT(  I: IN  std_logic;
        S1: IN  std_logic;
        S0: IN  std_logic;
        Q0: OUT std_logic;
        Q1: OUT std_logic;
        Q2: OUT std_logic;
        Q3: OUT std_logic );
END Demultiplexer_1_4;

ARCHITECTURE behavioral OF Demultiplexer_1_4 IS
  SIGNAL aNumber: std_logic_vector( 2 downto 0 );
BEGIN
  aNumber <= I & S1 & S0;
  with aNumber select
    Q0 <= '0' when "000", '0' when "001",
          '0' when "010", '0' when "011",
          '1' when "100", '0' when "101",
          '0' when "110", '0' when "111", 'X' when others;
  with aNumber select
    Q1 <= '0' when "000", '0' when "001",
          '0' when "010", '0' when "011",
          '0' when "100", '1' when "101",
          '0' when "110", '0' when "111", 'X' when others;
  with aNumber select
    Q2 <= '0' when "000", '0' when "001",
          '0' when "010", '0' when "011",
          '0' when "100", '0' when "101",
          '1' when "110", '0' when "111", 'X' when others;
  with aNumber select
    Q3 <= '0' when "000", '0' when "001",
          '0' when "010", '0' when "011",
          '0' when "100", '0' when "101",
          '0' when "110", '1' when "111", 'X' when others;
END behavioral;
```

C.2.4 Full Adder

```
----------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

ENTITY Adder_Full IS
  PORT( CIN: IN  std_logic;
        COUT:OUT std_logic;
        A:   IN  std_logic;
        B:   IN  std_logic;
        S:   OUT std_logic );
END Adder_Full;

ARCHITECTURE behavioral OF Adder_Full IS
  SIGNAL ABC: std_logic_vector( 2 downto 0 );
BEGIN
  ABC <= A & B & CIN;
  with ABC select
  S <= '0' when "000",
       '1' when "001",
       '1' when "010",
       '0' when "011",
       '1' when "100",
       '0' when "101",
       '0' when "110",
       '1' when "111",
       'X' when others;
  with ABC select
  COUT <= '0' when "000",
          '0' when "001",
          '0' when "010",
          '1' when "011",
          '0' when "100",
          '1' when "101",
          '1' when "110",
          '1' when "111",
          'X' when others;
END behavioral;
```

C.2.5 Comparator

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

ENTITY Compar_4 IS
  PORT( A3:  IN  std_logic;
        A2:  IN  std_logic;
        A1:  IN  std_logic;
        A0:  IN  std_logic;
        B3:  IN  std_logic;
        B2:  IN  std_logic;
        B1:  IN  std_logic;
        B0:  IN  std_logic;
        MIN: OUT std_logic;
        EQU: OUT std_logic;
        MAJ: OUT std_logic );
END Compar_4;

ARCHITECTURE behavioral OF Compar_4 IS
BEGIN
  Cmp4: PROCESS( A3, A2, A1, A0, B3, B2, B1, B0 )
  variable A: unsigned( 3 downto 0 );
  variable B: unsigned( 3 downto 0 );
  BEGIN
    A := (A3 & A2 & A1 & A0);
    B := (B3 & B2 & B1 & B0);
    --
    if    (A > B) then MIN <= '0'; EQU <= '0'; MAJ <= '1';
    elsif (A < B) then MIN <= '1'; EQU <= '0'; MAJ <= '0';
    elsif (A = B) then MIN <= '0'; EQU <= '1'; MAJ <= '0';
    else               MIN <= 'X'; EQU <= 'X'; MAJ <= 'X';
    END IF;
  END PROCESS;
END behavioral;
```

C.2.6 Flip-Flop D-PET

```
library ieee;
use ieee.std_logic_1164.all;

ENTITY DpetFF IS
  PORT(   D, Ck    : IN std_logic;
          nCL, nPR: IN std_logic;
          Q, nQ    : OUT std_logic );
END DpetFF;

ARCHITECTURE behavioral OF DpetFF IS
BEGIN
  Dff: PROCESS( Ck, nCL, nPR )
  BEGIN
    if    (nCL='0') and (nPR='0') then Q <= 'X'; nQ <= 'X';
    elsif (nCL='0') and (nPR='1') then Q <= '0'; nQ <= '1';
    elsif (nCL='1') and (nPR='0') then Q <= '1'; nQ <= '0';
    elsif (nCL='1') and (nPR='1') then
      if (Ck'event) AND (Ck='1') THEN -- Positive Edge
          Q <=  D;  nQ <= not D;
      END IF;
    else Q <= 'X'; nQ <= 'X';
    END IF;
  END PROCESS;
END behavioral;
```

C.2.7 Flip-Flop E-PET

```
library ieee;
use ieee.std_logic_1164.all;

ENTITY EpetFF IS
  PORT(   D, E, Ck: IN std_logic;
          nCL, nPR: IN std_logic;
          Q, nQ    : OUT std_logic );
END EpetFF;
ARCHITECTURE behavioral OF EpetFF IS
BEGIN
  Eff: PROCESS( Ck, nCL, nPR )
  BEGIN
    if    (nCL='0') and (nPR='0') then Q <= 'X'; nQ <= 'X';
    elsif (nCL='0') and (nPR='1') then Q <= '0'; nQ <= '1';
    elsif (nCL='1') and (nPR='0') then Q <= '1'; nQ <= '0';
    elsif (nCL='1') and (nPR='1') then
      if (Ck'event) AND (Ck='1') THEN -- Positive Edge
        if       (E = '1') then Q <=  D;  nQ <= not D;
        elsif not(E = '0') then Q <= 'X'; nQ <= 'X';
        END IF;
      END IF;
    else Q <= 'X'; nQ <= 'X';
    END IF;
  END PROCESS;
END behavioral;
```

C.2.8 Flip-Flop JK-PET

```
library ieee;
use ieee.std_logic_1164.all;

ENTITY JKpetFF IS
```

```
  PORT(  J, K, Ck: IN std_logic;
         nCL, nPR: IN std_logic;
         Q, nQ   : OUT std_logic );
END JKpetFF;

ARCHITECTURE behavioral OF JKpetFF IS
BEGIN
  JKff: PROCESS( Ck, nCL, nPR )
    variable  OutQ: STD_LOGIC;
  BEGIN
    if     (nCL='0') and (nPR='1') then OutQ := '0';
    elsif (nCL='1') and (nPR='0') then OutQ := '1';
    elsif (nCL='1') and (nPR='1') then
      if (Ck'event) AND (Ck='1') THEN
        -- Positive Edge
        if     (J = '0') AND (K = '1') THEN OutQ := '0';
        elsif (J = '1') AND (K = '0') THEN OutQ := '1';
        elsif (J = '1') AND (K = '1') THEN OutQ := not OutQ;
        elsif not((J='0')AND(K='0'))   THEN OutQ := 'X';
        END IF;
      END IF;
    else                                    OutQ := 'X';
    END IF;
    --
    Q   <= (     OutQ);
    nQ  <= (not OutQ);
    --
  END PROCESS;
END behavioral;
```

C.2.9 Parallel Register

```
library ieee;
use ieee.std_logic_1164.all;

ENTITY PiPoE4 IS
  PORT( Ck : IN std_logic;
        nCL: IN std_logic;
        E  : IN std_logic;
        P3 : IN std_logic;
        P2 : IN std_logic;
        P1 : IN std_logic;
        P0 : IN std_logic;
        Q3 : OUT std_logic;
        Q2 : OUT std_logic;
        Q1 : OUT std_logic;
        Q0 : OUT std_logic );
END PiPoE4;
ARCHITECTURE behavioral OF PiPoE4 IS
BEGIN
  RegPiPoE4: PROCESS( Ck, nCL )
    variable aReg: std_logic_vector( 3 downto 0 );
  BEGIN
    if     (nCL = '0') then     aReg := (others =>'0');
    elsif (nCL = '1') then
      if (Ck'event) AND (Ck='1') THEN -- Positive Edge
        if (E = '1') then
                aReg := (P3 & P2 & P1 & P0);
        elsif not(E = '0') then
                aReg := (others =>'X');
        END IF;
      END IF;
    else        aReg := (others =>'X');
    END IF;

    Q3  <= aReg(3);
    Q2  <= aReg(2);
    Q1  <= aReg(1);
    Q0  <= aReg(0);

  END PROCESS;
END behavioral;
```

C.2.10 Shift Register (S.I.P.O.)

```
library ieee;
use ieee.std_logic_1164.all;

ENTITY SiPoE4 IS
  PORT( I   : IN std_logic;
        Ck  : IN std_logic;
        nCL: IN std_logic;
        E   : IN std_logic;
        Q3  : OUT std_logic;
        Q2  : OUT std_logic;
        Q1  : OUT std_logic;
        Q0  : OUT std_logic );
END SiPoE4;

ARCHITECTURE behavioral OF SiPoE4 IS
BEGIN
  RegSiPoE4: PROCESS( Ck, nCL )
  variable aReg: std_logic_vector( 3 downto 0 );
  BEGIN
    if    (nCL = '0') then aReg := (others =>'0');
    elsif (nCL = '1') then
      if (Ck'event) AND (Ck='1') THEN -- Positive Edge
        if (E = '1') then
          aReg := (I & aReg(3) & aReg(2) & aReg(1));
        elsif not(E = '0') then
          aReg := (others =>'X');
        END IF;
      END IF;
    else aReg := (others =>'X');
    END IF;
    --
    Q3 <= aReg(3);
    Q2 <= aReg(2);
    Q1 <= aReg(1);
    Q0 <= aReg(0);
    --
  END PROCESS;
END behavioral;
```