```
============================
EA999 / SW-2
SEQUENCE OF EXAMPLES
============================

1)  COUNT_DOWN.VHD
    - warmup from week-1
    - counter with control inputs & parameters (via generic)
    - Q: which other value, would you also declare as generic?


2)  ALARM_LEVEL_DISPLAY_WO_DEFAULT.VHD
    - priority in combinational processes
    - Q: what would you improve/change in this code?


3)  ALARM_LEVEL_DISPLAY.VHD
    - default statements inside processes
      (possible with sequential statements)
    - identify most common assignments
    - Q: what happens with missing default
         (or incomplete assignment in comb-logic)


3)  HEX2SEVSEG_W_CONTROL.VHD
    - comb logic with nested IF/THEN & CASE/WHEN
    - Q: draw & fill out truth table
    - Q: how could change block to use only once NOT()


4)  SHIFTREG_P2S.VHD
    - example of shiftregister circuit
    - Comment: applications of shiftregisters
    - Exercise: pseudo-random-register (PRG)
    - Q: implement PRG with sequence length of (2^4)-1
    - Q: implement PRG with sequence length of (2^128)-1
    - Q: how would you initialise your PRG?


5)  SYNC_N_EDGEDETECTOR.VHD
    - why synchronise external inputs /or/ inputs different clock domains
    - avoid metastability
    - Q: Draw the RTL-diagram (RTL analysis) / timing diagram =>as homework


6)  FSM_EXAMPLE.VHD
    - example for enumerated datatype declaration
      and fsm implementation syntax with CASE/WHEN
    - Q: draw bubble diagram


7)  FSM_EXERCISE.VHD
    - example of 2 RTL structures in a single file
    - Q: draw bubble- and RTL-diagram
    - Q: understand functioning.
    For example, what is max interval between take_1 and take_2 to be
    able to achieve state count_down?
```

```vhdl
-----------------------------------------
-- Block code:  count_down.vhd
-- History:     12.Nov.2013 - 1st version (dqtm)
--                 <date> - <changes>  (<author>)
-- Function: down-counter, with start input and count output.
--          The input start should be a pulse which causes the
--          counter to load its max-value. When start is off,
--          the counter decrements by one every clock cycle till
--          count_o equals 0. Once the count_o reachs 0, the counter
--          freezes and wait till next start pulse.
--          Can be used as enable for other blocks where need to
--          count number of iterations.
-----------------------------------------


-- Library & Use Statements
-----------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;


-- Entity Declaration
-----------------------------------------
ENTITY count_down IS
GENERIC (width : positive := 4);
  PORT( clk,reset_n     : IN    std_logic;
        start_i         : IN    std_logic;
        count_o         : OUT   std_logic_vector(width-1 downto 0)
        );
END count_down;


-- Architecture Declaration
-----------------------------------------
ARCHITECTURE rtl OF count_down IS
-- Signals & Constants Declaration
-----------------------------------------
CONSTANT    max_val:            unsigned(width-1 downto 0):= to_unsigned(4,width);
-- convert integer value 4 to unsigned with 4bits
SIGNAL      count, next_count:  unsigned(width-1 downto 0);


-- Begin Architecture
-----------------------------------------
BEGIN


  -------------------------------------------------
  -- PROCESS FOR COMBINATORIAL LOGIC
  -------------------------------------------------
  comb_logic: PROCESS(start_i,count)
  BEGIN
    -- load
    IF (start_i = '1') THEN
        next_count <= max_val;

    -- decrement
    ELSIF (count > 0) THEN
        next_count <= count - 1 ;

    -- freezes
    ELSE
        next_count <= count;
    END IF;

  END PROCESS comb_logic;
```

```
71      ------------------------------------------------
72      -- PROCESS FOR REGISTERS
73      ------------------------------------------------
74      flip_flops : PROCESS(clk, reset_n)
75      BEGIN
76        IF reset_n = '0' THEN
77            count <= to_unsigned(0,width); -- convert integer value 0 to unsigned with
            4bits
78        ELSIF rising_edge(clk) THEN
79            count <= next_count ;
80        END IF;
81      END PROCESS flip_flops;
82
83
84      ------------------------------------------------
85      -- CONCURRENT ASSIGNMENTS
86      ------------------------------------------------
87      -- convert count from unsigned to std_logic (output data-type)
88      count_o <= std_logic_vector(count);
89
90
91     -- End Architecture
92    ---------------------------------------------
93    END rtl;
94
95
```

```vhdl
-----------------------------------------
-- Block code:  alarm_level_display_wo_default.vhd
-- History:     30.Sep.2011 - example for introduction to comb logic
--              05.Okt.2013 - also check example with default statements!! (dqtm)
-- Function: Decodes the output for a alarm level display.
--           Only comb logic. Example of logic with priority.
-----------------------------------------

-- Library & Use Statements
LIBRARY ieee;
use ieee.std_logic_1164.all;

-- Entity Declaration
ENTITY alarm_level_display_wo_default IS
  PORT(
        alarm_prio1     : IN    std_logic;
        alarm_prio2     : IN    std_logic;
        alarm_prio3     : IN    std_logic;
        display_red     : OUT   std_logic;
        display_orange  : OUT   std_logic;
        display_yellow  : OUT   std_logic;
        display_green   : OUT   std_logic
        );
END alarm_level_display_wo_default ;

-- Architecture Declaration
ARCHITECTURE rtl OF alarm_level_display_wo_default IS

-- Begin Architecture
BEGIN

  -----------------------------------------
  -- Process for combinational logic
  -----------------------------------------
  comb_alarm: PROCESS(alarm_prio1,alarm_prio2,alarm_prio3)
  BEGIN
    IF (alarm_prio1 = '1') THEN
        display_red     <= '1';
        display_orange  <= '0';
        display_yellow  <= '0';
        display_green   <= '0';

    ELSIF(alarm_prio2 = '1') THEN
        display_red     <= '0';
        display_orange  <= '1';
        display_yellow  <= '0';
        display_green   <= '0';

    ELSIF(alarm_prio3 = '1') THEN
        display_red     <= '0';
        display_orange  <= '0';
        display_yellow  <= '1';
        display_green   <= '0';

    ELSE
        display_red     <= '0';
        display_orange  <= '0';
        display_yellow  <= '0';
        display_green   <= '1';
    END IF;
  END PROCESS comb_alarm;

END rtl;
```

```vhdl
-----------------------------------------
-- Block code:  alarm_level_display.vhd
-- History:     30.Sep.2011 - example for introduction to comb logic
--              05.Okt.2013 - added default statements (dqtm)
-- Function: Decodes the output for a alarm level display.
--           Only comb logic. Example of logic with priority.
-----------------------------------------

-- Library & Use Statements
LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- Entity Declaration
ENTITY alarm_level_display IS
  PORT(
        alarm_prio1     : IN    std_logic;
        alarm_prio2     : IN    std_logic;
        alarm_prio3     : IN    std_logic;
        display_red     : OUT   std_logic;
        display_orange  : OUT   std_logic;
        display_yellow  : OUT   std_logic;
        display_green   : OUT   std_logic
        );
END alarm_level_display ;

-- Architecture Declaration
ARCHITECTURE rtl OF alarm_level_display IS

-- Begin Architecture
BEGIN
  -----------------------------------------
  -- Process for combinational logic
  -----------------------------------------
  -- OBS.: The implementation with Default Statements is only
  --       possible within a process (sequential statements)
  -----------------------------------------
  comb_alarm: PROCESS(alarm_prio1,alarm_prio2,alarm_prio3)
  BEGIN
    -- Default Statements
        display_red     <= '0';
        display_orange  <= '0';
        display_yellow  <= '0';
        display_green   <= '0';
    --Check inputs
    IF (alarm_prio1 = '1') THEN
        display_red     <= '1';

    ELSIF(alarm_prio2 = '1') THEN
        display_orange  <= '1';

    ELSIF(alarm_prio3 = '1') THEN
        display_yellow  <= '1';

    ELSE
        display_green   <= '1';
    END IF;
  END PROCESS comb_alarm;

END rtl;

-------------------------------------------------
-- Because there is only 1 statement after each then
-- you could also write the IF/ELSIF/THEN as
--
--  IF    (alarm_prio1 = '1') THEN  display_red     <= '1';
--  ELSIF (alarm_prio2 = '1') THEN  display_orange  <= '1';
--  ELSIF (alarm_prio3 = '1') THEN  display_yellow  <= '1';
--  ELSE                            display_green   <= '1';
--  END IF;
--
-------------------------------------------------
```

```vhdl
-----------------------------------------
-- Block code:  hex2sevseg_w_control.vhd
-- History:     24.Sep.2013 - 1st version for lab5 (dqtm)
--              24.Sep.2013 - add ctrl inputs,lab6 (dqtm)
-- Function: Hexa to seven-seg converter
--            plain functionality plus control inputs
--            implemented with comb logic inside process
-----------------------------------------

-- Library & Use Statements
LIBRARY ieee;
use ieee.std_logic_1164.all;

-- Entity Declaration
ENTITY hex2sevseg_w_control IS
  PORT(
        blank_n_i:          IN  std_logic;
        lamp_test_n_i:      IN  std_logic;
        ripple_blank_n_i:   IN  std_logic;
        hexa_i:             IN  std_logic_vector(3 downto 0);
        ripple_blank_n_o:   OUT std_logic;
        seg_o:              OUT std_logic_vector(6 downto 0)); -- Sequence is
        "gfedcba" (MSB is seg_g)
END hex2sevseg_w_control ;


-- Architecture Declaration
ARCHITECTURE rtl OF hex2sevseg_w_control IS

    -- Signals & Constants Declaration
    CONSTANT display_0      : std_logic_vector(6 downto 0):= "0111111";
    CONSTANT display_1      : std_logic_vector(6 downto 0):= "0000110";
    CONSTANT display_2      : std_logic_vector(6 downto 0):= "1011011";
    CONSTANT display_3      : std_logic_vector(6 downto 0):= "1001111";
    CONSTANT display_4      : std_logic_vector(6 downto 0):= "1100110";
    CONSTANT display_5      : std_logic_vector(6 downto 0):= "1101101";
    CONSTANT display_6      : std_logic_vector(6 downto 0):= "1111101";
    CONSTANT display_7      : std_logic_vector(6 downto 0):= "0000111";
    CONSTANT display_8      : std_logic_vector(6 downto 0):= "1111111";
    CONSTANT display_9      : std_logic_vector(6 downto 0):= "1101111";
    CONSTANT display_A      : std_logic_vector(6 downto 0):= "1110111";
    CONSTANT display_B      : std_logic_vector(6 downto 0):= "1111100";
    CONSTANT display_C      : std_logic_vector(6 downto 0):= "0111001";
    CONSTANT display_D      : std_logic_vector(6 downto 0):= "1011110";
    CONSTANT display_E      : std_logic_vector(6 downto 0):= "1111001";
    CONSTANT display_F      : std_logic_vector(6 downto 0):= "1110001";
    CONSTANT display_blank  : std_logic_vector(6 downto 0):= (others =>'0');


-- Begin Architecture
BEGIN

    -----------------------------------------
    -- Process for combinatorial logic
    -----------------------------------------
    sevseg_comb: PROCESS (ALL)
    BEGIN
    -- Default statement for ripple_blank output= inactive
    ripple_blank_n_o <= '1';

        -- Control Inputs from high to low priority:
        --           blank; lamp_test; ripple_blank
        --------------------------------
        IF (blank_n_i='0') THEN
            seg_o <= NOT(display_blank);

        ELSIF (lamp_test_n_i = '0') THEN
            seg_o <= NOT(display_8);

        ELSE
        -- Hexa input values 1-F not affected by ripple_blank
```

```
71          -- For hexa input x0, check status of ripple_blank ctrl
72              CASE hexa_i IS
73                  WHEN x"0" =>
74                      IF (ripple_blank_n_i = '0') THEN
75                          seg_o <= NOT(display_blank);
76                          ripple_blank_n_o <= '0';
77                      ELSE
78                          seg_o <= NOT(display_0);
79                      END IF;
80
81                  WHEN x"1" =>   seg_o <= NOT(display_1);
82                  WHEN x"2" =>   seg_o <= NOT(display_2);
83                  WHEN x"3" =>   seg_o <= NOT(display_3);
84                  WHEN x"4" =>   seg_o <= NOT(display_4);
85                  WHEN x"5" =>   seg_o <= NOT(display_5);
86                  WHEN x"6" =>   seg_o <= NOT(display_6);
87                  WHEN x"7" =>   seg_o <= NOT(display_7);
88                  WHEN x"8" =>   seg_o <= NOT(display_8);
89                  WHEN x"9" =>   seg_o <= NOT(display_9);
90                  WHEN x"A" =>   seg_o <= NOT(display_A);
91                  WHEN x"B" =>   seg_o <= NOT(display_B);
92                  WHEN x"C" =>   seg_o <= NOT(display_C);
93                  WHEN x"D" =>   seg_o <= NOT(display_D);
94                  WHEN x"E" =>   seg_o <= NOT(display_E);
95                  WHEN x"F" =>   seg_o <= NOT(display_F);
96                  WHEN OTHERS => seg_o <= NOT(display_blank);
97              END CASE;
98          END IF;
99      END PROCESS sevseg_comb;
100
101 -- End Architecture
102 END rtl;
103
104
```

```vhdl
-----------------------------------------
-- Block code:  shiftreg_p2s.vhd
-- History:     12.Nov.2013 - 1st version (dqtm)
--                 <date> - <changes>  (<author>)
-- Function: shift-register working as a parallel to serial converter.
--          The block has a load( or shift_n) control input, plus a parallel data input.
--          If load is high the parallel data is loaded, and if load is low the data is shifted towards the LSB.
--          During shift the MSB gets the value of '1'.
--          The serial output is the LSB of the shiftregister.
--          Can be used as P2S in a serial interface, where inactive value (or rest_value)  equals '1'.
-----------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY shiftreg_p2s IS
  PORT( clk,set_n       : IN    std_logic;          -- Attention, this block has a set_n input for initialisation!!
        load_i          : IN    std_logic;
        par_i           : IN    std_logic_vector(3 downto 0);
        ser_o           : OUT   std_logic
        );
END shiftreg_p2s;

ARCHITECTURE rtl OF shiftreg_p2s IS
-- Signals & Constants Declaration
-----------------------------------------
    SIGNAL      shiftreg, next_shiftreg:    std_logic_vector(4 downto 0);    -- add one FF for start_bit

BEGIN

  -------------------------------------------------
  -- PROCESS FOR COMBINATIONAL LOGIC
  -------------------------------------------------
  shift_comb: PROCESS(ALL)
  BEGIN
    IF (load_i = '1') THEN            -- load parallel data + add start_bit
        next_shiftreg <= par_i & '0'; -- LSB='0' is the start_bit

    ELSE                              -- shift; shift direction towards LSB
        next_shiftreg <= '1' & shiftreg(4 downto 1);
    END IF;

  END PROCESS shift_comb;

  -------------------------------------------------
  -- PROCESS FOR REGISTERS
  -------------------------------------------------
  shift_dffs : PROCESS(clk, set_n)
  BEGIN
    IF set_n = '0' THEN
```

```
50          shiftreg <= (others=>'1');
51        ELSIF rising_edge(clk) THEN
52          shiftreg <= next_shiftreg ;
53        END IF;
54      END PROCESS shift_dffs;
55
56      ---------------------------------------------------
57      -- CONCURRENT ASSIGNMENTS
58      ---------------------------------------------------
59      -- take LSB of shiftreg as serial output
60      ser_o <= shiftreg(0);
61
62  END rtl;
63
64
```

```vhdl
-----------------------------------------
-- Block code:  sync_n_edgeDetector.vhd
-- History:     15.Nov.2017 - 1st version (dqtm)
--              15.Jan.2018 - adapt reset value for usage in mini-project (dqtm)
--              01.Mar.2018 - rename in English (dqtm)
--                 <date> - <changes>  (<author>)
-- Function: edge detector with rise & fall outputs.
--           Declaring FFs as a shift-register.
-----------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY sync_n_edgeDetector IS
  PORT( data_in     : IN    std_logic;
        clock       : IN    std_logic;
        reset_n     : IN    std_logic;
        data_sync   : OUT   std_logic;
        rise        : OUT   std_logic;
        fall        : OUT   std_logic
        );
END sync_n_edgeDetector;


ARCHITECTURE rtl OF sync_n_edgeDetector IS
    -- Signals & Constants Declaration
    SIGNAL shiftreg, next_shiftreg: std_logic_vector(2 downto 0);

BEGIN
    -----------------------------------------
    -- Process for combinatorial logic
    -- OBs.: small logic, could be outside process,
    --       but doing inside for didactical purposes!
    -----------------------------------------
    comb_proc : PROCESS(data_in, shiftreg)
    BEGIN
        next_shiftreg <= data_in & shiftreg(2 downto 1) ;  -- shift direction towards LSB
    END PROCESS comb_proc;

    -----------------------------------------
    -- Process for registers (flip-flops)
    -----------------------------------------
    reg_proc : PROCESS(clock, reset_n)
    BEGIN
        IF reset_n = '0' THEN
            shiftreg <= (OTHERS => '1');
        ELSIF (rising_edge(clock)) THEN
            shiftreg <= next_shiftreg;
        END IF;
    END PROCESS reg_proc;

    -----------------------------------------
    -- Concurrent Assignments
    -- OBs.: no logic after the 1st-FF (shiftreg(2)) because it was added for sync purposes
    -----------------------------------------
    rise        <=     shiftreg(1)  AND NOT(shiftreg(0));
    fall        <= NOT(shiftreg(1)) AND     shiftreg(0);
    data_sync   <=     shiftreg(1) ;    -- take serial_in at same period as fall/rise pulse

END rtl;

```

```vhdl
-----------------------------------------
-- Block code:  fsm_example.vhd
-- History:     23.Nov.2017 - 1st version (dqtm)
--                  <date> - <changes>  (<author>)
-- Function: fsm_example using enumerated type declaration.
-----------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY fsm_example IS
    PORT (
        clk,reset :     IN std_logic;
        d, n :          IN std_logic;
        z :             OUT std_logic
    );
END fsm_example;


ARCHITECTURE rtl OF fsm_example IS
    TYPE t_state IS (s_e1, s_e2, s_e3);     -- declaration of new datatype
    SIGNAL s_state, s_nextstate :  t_state; -- 2 signals of the new datatype

BEGIN

  ---------------------------------------------------
  -- PROCESS FOR COMB-INPUT LOGIC
  ---------------------------------------------------
  fsm_drive : PROCESS (s_state, d, n)
  BEGIN
    -- Default Statement
    s_nextstate <= s_state;

    CASE s_state IS
        WHEN s_e1 =>
            IF (d = '1') THEN
                s_nextstate <= s_e3;
            ELSIF (n = '1') THEN
                s_nextstate <= s_e2;
            END IF;
        WHEN s_e2 =>
            IF (d = '1') THEN
                s_nextstate <= s_e1;
            ELSIF (n = '1') THEN
                s_nextstate <= s_e3;
            END IF;
        WHEN s_e3 =>
            IF (d = '1') THEN
                s_nextstate <= s_e2;
            ELSIF (n = '1') THEN
                s_nextstate <= s_e1;
            END IF;
        WHEN OTHERS =>
            s_nextstate <= s_e1;
    END CASE;
  END PROCESS fsm_drive;


  ---------------------------------------------------
  -- PROCESS FOR COMB-INPUT LOGIC
  ---------------------------------------------------
  fsm_output : PROCESS (s_state, d, n)
  BEGIN

    CASE s_state IS
        WHEN s_e2 | s_e3 => z <= '1';
        WHEN OTHERS =>      z <= '0';
    END CASE;
  END PROCESS fsm_output;


```

```
72      ------------------------------------------------
73      -- PROCESS FOR REGISTERS
74      ------------------------------------------------
75      PROCESS (clk, reset)
76      BEGIN
77        IF (reset = '1') THEN
78            s_state <= s_e1;
79        ELSIF rising_edge(clk) THEN
80            s_state <= s_nextstate;
81        END IF;
82      END PROCESS;
83
84      END rtl;
85
```

```vhdl
-----------------------------------------
-- Block code:  fsm_exercise.vhd
-- History:     21.Apr.2014 - 1st version (dqtm)
-- Function:    fsm exercise for mid-sem-exam
-----------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

 ENTITY fsm_exercise IS
    PORT (
        clock   : in std_logic;
        reset_n : in std_logic;
        take1   : in std_logic;
        take2   : in std_logic;
        acknow  : in std_logic;
        count_o : out std_logic_vector(3 downto 0)
        );
END fsm_exercise;

ARCHITECTURE rtl OF fsm_exercise IS
    -- Signals & Types Declaration
    TYPE fsm_state_typ IS (idle, check_interval, count_down, wait_ack);
    SIGNAL state, next_state    : fsm_state_typ ;
    SIGNAL count, next_count    : unsigned(3 downto 0);

BEGIN
    -----------------------------------------
    -- Process for combinational logic
    -----------------------------------------
    comb_logic: PROCESS(take1,take2,acknow,state,count)
    BEGIN
    -- default statements
    next_state <= state;
    next_count <= count;

    -- state transitions
    CASE state IS
        WHEN idle =>
            -- check take1
            IF take1='1' THEN
                next_state <= check_interval;
                next_count <= to_unsigned(7,4);
            END IF;

        WHEN check_interval =>
            -- check & decrement counter
            IF count>0 THEN
                next_count <= count-1;
                --check take2
                IF take2='1' THEN
                    next_state <= count_down;
                    next_count <= to_unsigned(15,4);
                END IF;
            ELSE  -- count=0
                next_state <= idle;
            END IF;

        WHEN count_down =>
            --check counter
            IF count>0 THEN
                next_count <= count-1;
            ELSE
                next_state <= wait_ack;
            END IF;

        WHEN wait_ack =>
            --check acknowledge input
            IF acknow='1' THEN
                next_state <= idle;
            END IF;
```

```
72
73          WHEN OTHERS =>
74              next_state <= idle;
75              next_count <= to_unsigned(0,4);
76          END CASE;
77      END PROCESS comb_logic;
78
79
80      -------------------------------------------
81      -- Process for registers (flip-flops)
82      -------------------------------------------
83      flip_flops : PROCESS(clock, reset_n)
84      BEGIN
85          IF reset_n = '0' THEN
86              state <= idle;
87              count <= to_unsigned(0,4);
88          ELSIF RISING_EDGE(clock) THEN
89              state <= next_state;
90              count <= next_count;
91          END IF;
92      END PROCESS flip_flops;
93
94      -------------------------------------------
95      -- Concurrent Assignements
96      -- e.g. Assign outputs from intermediate signals
97      -------------------------------------------
98      count_o <= std_logic_vector(count);
99
100 END rtl;
101
102
103
104
105
106
107
```