FPGA Programing Tutorial
Lab II
Design of a Digital Alarm Clock

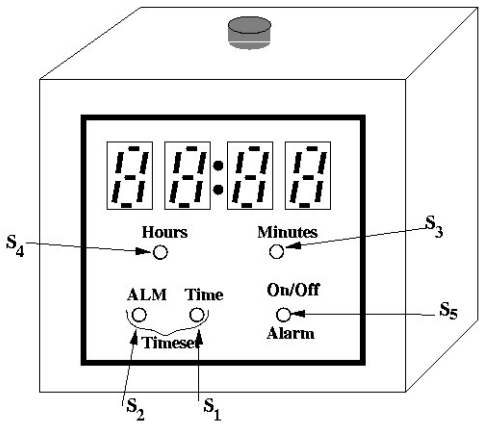

Figure 1: Alarm Clock

The goal of this lab is to develop a controller for the alarm clock shown in figure 1

Alarm Clock Specification

The clocks has a certain number of inputs (buttons) and outputs (display, and bell or LED), which are used for the interaction between the user and the clock.

Inputs

- Switch S_1, used to display normal time.
- Switch S_2, used to display alarm time.
- Switch S_3, used to increment the minutes.
- Switch S_4, used to increment the hours.
- Switch S_5, used to activate the alarm.

Outputs

- 4 Seven-segment-displays to show the normal time or the alarm time.
- 1 Bell that rings upon reaching the alarm time set. We can use a LED for the same purpose.

States

Our goal is to model the controller as a finite state machine that can move in three different states as shown in figure 2:

- **time**: This is the default state. In this state, seconds, minutes and hours are normally incremented.
  If alarm is activated and alarm time is equal current time, then the bell rings.
- **Set_time**: The controller moves into this state if the switch S_1 is pressed.
  If S_1 and S_3 are pressed simultaneously, then the minutes are incremented.
  If S_1 and S_4 are pressed simultaneously, then the hours are incremented.
  By releasing S_1, the controller returns in state **time**.
- **Set_Alarm**: The controller moves into this state if the switch S_2 is pressed.
  If S_2 and S_3 are pressed simultaneously, then the minutes are incremented.
  If S_2 und S_4 are pressed simultaneously, then the hours are incremented.
  By releasing S_2, the controller returns in state **time**.
- By pressing S_1 and S_2 at the same time, the controller returns to state **time**.

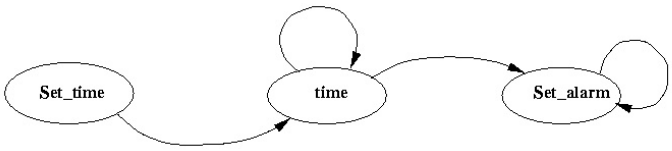

Figure 2: State diagram of the digital alarm clock

Preparation

Complete the state diagram of figure 2. Include the remaining arcs and label all arcs with the conditions which triggers the transition from one state to the next one using that arc. Label the states with the corresponding outputs, when the system is in those states.

Implementation the alarm clock

The implementation of the alarm clock will be done on the "Nexys 3"-Board, which features a Xilinx Spartan 6 FPGA, several switches and LEDs, 4 seven segment display, and 5 pus buttons.

To implement our alarm clock, the four seven-segment-displays, use four push buttons, (BTN1, BTN2, BTN3, BTN4) and 2 switches (SW1, SW7) of your choice. A ring bell can be connected to a general purpose pin on the board. If we don't have one, we can just use a LED for signalling an alarm. You may adopt the following mapping: $S_1 \rightarrow$ BTN1, $S_2 \rightarrow$ BTN2, $S_3 \rightarrow$ BTN3, $S_4 \rightarrow$ BTN4, $S_5 \rightarrow$ SW1, $S_6 \rightarrow$ SW7. Switch S_6 is used as reset.

Figure 3 shows the structure of our VHDL-Description. However you are free to develop your own structure and code.

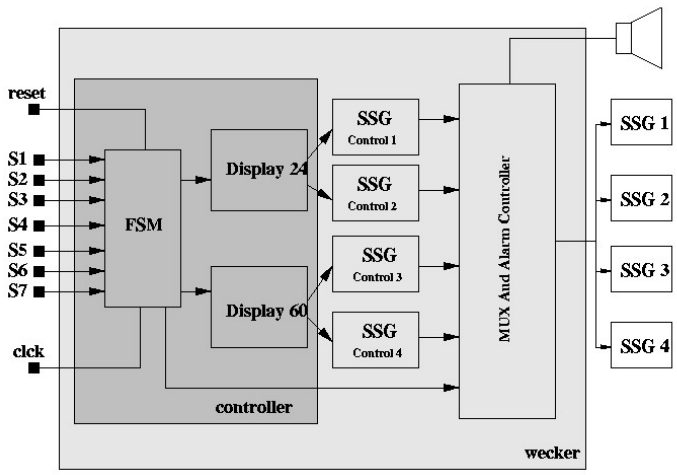

Figure 3: Structur of the VHDL-Program

The code developped in the previous lab can be reused for this lab. **Display24** display a number between 0 and 23 on two digits. **Display60** displays a number between 0 and 59 on two digits.

- Complete the following VHDL-Code for the FSM described earlier

```
-- DEFINITION of the FSM-states

  signal sectrigger : std_logic;          -- '1' each secind
                                          -- else 0
  signal DISPLAY10, DISPLAY11, DISPLAY20, DISPLAY21: integer range 0 to 9 ;
  signal DISPLAY30, DISPLAY31, DISPLAY40, DISPLAY41: integer range 0 to 9 ;
```

```
  signal hours, whours: integer range 0 to 23;
  signal secs, mins, wmins: integer range 0 to 59;

  type state_type is (ntime, set_time, set_alarm);
   signal sectrigger : std_logic;
  signal current_state: state_type;
begin

  SECTIMER: process(clk,reset)
    variable timer_msb_to_zero : std_logic_vector(timer'RANGE);
  begin
    if reset='1' then
      timer <= (others => '0');
    elsif clk'EVENT and clk='1' then
      timer_msb_to_zero := timer;
      -- set msb to zero, the msb will be our overflow bit
      timer_msb_to_zero(timer'LENGTH-1) := '0';
      timer <= timer_msb_to_zero + 1;
    end if;
  end process SECTIMER;
  sectrigger <= timer(timer'LENGTH-1);

  FSM: process (clk,reset,current_state,secs,mins,hours,wmins,whours, S)
    variable next_state: state_type;
  begin
    if reset = '1' then
      hours <= 0; mins <= 0;  secs <= 0;
      whours <= 0; wmins <= 0; alarm <= '0';
      current_state <= ntime;
    elsif (clk'event and clk='1') then
      if (not (current_state = set_time) and (sectrigger='1')) then
        -- Zhle Uhr hoch
        if secs = 59 then
          secs <= 0;
          if mins = 59 then
            mins <= 0;
            if hours = 23 then
              hours <= 0;
            else
              hours <= hours+1;
            end if;
          else
            mins <= mins+1;
          end if;
        else
          secs <= secs+1;
        end if;
      end if;
```

```
      case current_state is
        when ntime =>
          -- check if alarm is set

          -- set next state

          -- State Set_time

          -- set minutes and hours with S(3) or S(4)

          -- set next state

          -- state set_alarm

          -- set WMinute and WStunde with S(3) bzw. S(4)

          -- set next state
.......;

END PROCESS FSM;
```

- Complete the following VHDL-Code for the controller, under the assumption that the modules **Display24** and **Display60** are available.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity controller is port (
  S:                                        in STD_LOGIC_VECTOR (1 to 5);
  DISPLAY0, DISPLAY1, DISPLAY2, DISPLAY3: out integer range 0 to 9 ;
  alarm:                                    out STD_LOGIC;
  clk:                                      in  std_logic;
  reset:                                    in  std_logic );
end controller;

architecture Behavioral of controller is

  component Display60 is Port (
    number : in integer range 0 to 59;
    position1, position0: out integer range 0 to 9);
  end component;

  component Display24 is Port (
    number : in integer range 0 to 23;
    position1, position0: out integer range 0 to 9);
  end component;
```

```
  signal timer: std_logic_vector (13 downto 0);
  signal sectrigger : std_logic;
  signal DISPLAY10, DISPLAY11, DISPLAY20, DISPLAY21: integer range 0 to 9
  signal DISPLAY30, DISPLAY31, DISPLAY40, DISPLAY41: integer range 0 to 9

  signal hours, whours: integer range 0 to 23;
  signal secs, mins, wmins: integer range 0 to 59;

  type state_type is (ntime, set_time, set_alarm);

  signal current_state: state_type;
begin

 SECTIMER: process(clk,reset)
 begin
   -- A code of a second-trigger will be placed here
 end process SECTIMER;

 FSM: PROCESS (clk,reset,current_state,secs,mins,hours,wmins,whours, S)
   BEGIN
   -- Place your FSM-Code here
 END PROCESS FSM;

-- MINUTE-DISPLAY (TIME)
 MIN_CONVERT: Display60
   PORT MAP (number => ...., position1 => DISPLAY11,  position0 => DISPLAY

-- HOURS-DISPLAY (TIME)
 HR_CONVERT: Display24
   PORT MAP (number => ....., position1 => DISPLAY21,  position0 => DISPLA

--  MINUTE-DISPLAY (ALARM-TIME)
 WMIN_CONVERT: Display60
   PORT MAP (number => ...., position1 => DISPLAY31,  position0 => DISPLAY

-- HOURS-DISPLAY (ALARM-TIME)
 WHR_CONVERT: Display24
  PORT MAP (number => ..... , position1 => DISPLAY41,  position0 => DISPLA

  -- Describes how to set the DISPLAY-Variables
 Switch_Display: process (S, DISPLAY10, DISPLAY11, DISPLAY20, DISPLAY21, D
                   DISPLAY31, DISPLAY40, DISPLAY41 ) is
 begin
   if current_state /= set_alarm then
     DISPLAY0 <= DISPLAY10 ;
     DISPLAY1 <= DISPLAY11;
     DISPLAY2 <= DISPLAY20 ;
     DISPLAY3 <= DISPLAY21 ;
```

```
    else
      DISPLAY0 <= DISPLAY30;
      DISPLAY1 <= DISPLAY31;
      DISPLAY2 <= DISPLAY40;
      DISPLAY3 <= DISPLAY41;
    end if;
  end process display;
END Behavioral;
```

- Give the complete VHDL-code of the alarm clock.

Lab

Implement the alarm clock on the ”Nexys 3” Board.