Points	Grade

Philipp Lehninger 1327039
Benedikt Morgenbesser 1027440

Digital Integrated Circuits Lab (LDIS)

384.088, Summer Term 2019

Supervisors:
Christian Krieg, David Radakovits, Axel Jantsch

Task 1: Digital Thermometer

1 Problem statement

Your goal is to design a system that reads data from a temperature sensor, that performs simple digital signal processing on captured temperature data, and outputs the processed temperature value via a Seven-segment display

Figure 1: Block diagram of the system

You have to design three intellectual property (IP) cores:

1. Data sampling: This IP core controls and reads the output of the temperature sensor, and provides temperature data of a pre-defined sample rate at its output. The sample rate should be adjustable pre-synthesis. The temperature signal's resolution should be 16-bit.
2. The digital signal processor (DSP) core should implement a moving average algorithm with a pre-defined length of the gliding window, which should be adjustable during run-time.
3. The output stage takes the processed temperature signal and provides it to the seven segment display.

Design the system using either VHDL or Verilog. You can use third-party intellectual property (3PIP), but you have to fully understand what you are doing, and you have to cite the original source.

2 Team management

You can solve the problem on your own, or you can team up with others in order to share work and knowledge. If you team up, it is essential that you understand every aspect of the solution, rather that just the part that was implemented by you. In any case, use *git* to organize collaboration, even in case you work alone. We will assess your activity in the project based on your commits.

3 Proposed solution

Model the system in any hardware description language (HDL) of your choice. Design a testbench with reasonable test cases, and verify the design's functional correctness by simulating your design. Use any simulator you prefer, however, support is only provided for *ghdl*, *Verilator*, *Icarus Verilog*, and *xsim*. You can use free and open-source tools for modeling, simulation and synthesis. However, for technology mapping and bitstream generation (i.e., *implementing* the design), you will need *Vivado*. Implement the design on the Nexys 4 DDR board.

4 Implementation

To solve this task, we have implemented three components *sampling*, *DSP* and *output* and a top level module *thermometer*. Detailed information for the components is given in the following sections.

4.1 Sampling

The Board includes an ADT7420 temperature sensor. The interface between sensor and FPGA ist shown in figure 2. For the data aquisition of the temperature sensor we used the *TempSensorCtl* module from the Nexys Demo Version by Elod Gyorgy.[1] We just had to adapt the resolution lenght of the temperature vector to 16 bit. Therefore, we set the configuration register (address 0x03) for the temperature sensor to 0x80. The sample rate can be set presynthesis. A list of the allowed values and their corrensponding sample time is shown in table 1. The temperature is actually read continously and saved in the temperature value register. What we call sample rate is how often we read this data. The reading of the temperature register is shown in figure 3.

[1] https://github.com/Digilent/Nexys-Video-OLED/releases

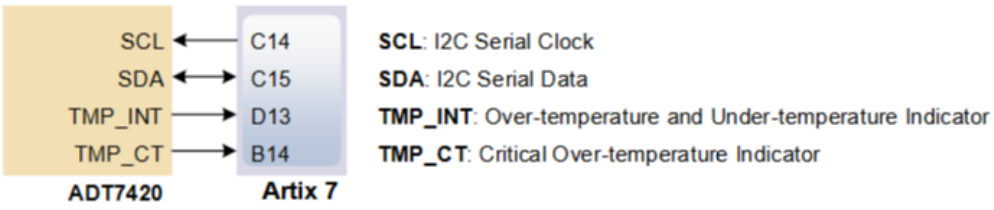

Figure 2: Temperature Sensor Interface.

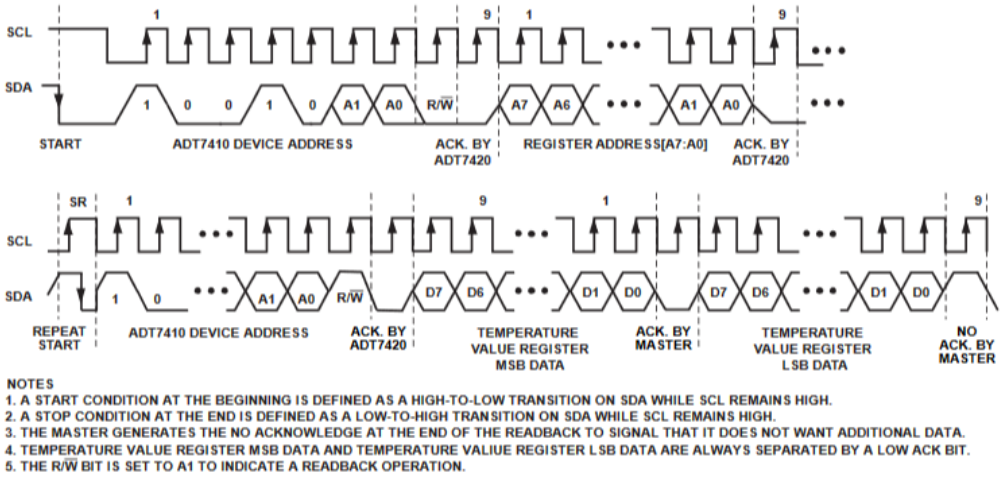

Figure 3: Reading in data

sampling rate value	corresponding sample time
1	0.25 sec
2	0.5 sec
4	1 sec
3	not valid

Table 1: Sampling time.

4.2 DSP

For the moving average block we have implemented three different window sizes, i.e. 1 x / 2 x / 4 x the sampling period. This can be changed during runtime using the up and down button on the board.
The code for the averaging algorithm is shown below. Each new temperature value is safed in temp_in_ex0. To keep the 2's complement addition consistent we add two more bits and copy the MSB into it. Theoretically for an addition of four values four additional bits would be needed. However, for practical reasons we have constraint the maximum temperature range to $\pm 99\,^{\circ}\mathrm{C}$ which allows us to add to values in 16 bit without an overflow. (16 bits correspond to a temperature range of about $\pm 255\,^{\circ}\mathrm{C}$). The division is simply done by shifting the summed value.
The window sizes of 1 x / 2 x / 4 x the sampling period might be a little short, but this could easly be extended to 2^n x sampling rates by adding 2^n registers an n bits.

```
average: process(Sample_Clk, Temp)

 variable temp_sum : std_logic_vector(17 downto 0) := (others => '0');     -- summarized
  temperature

begin

  if(rising_edge(Sample_Clk)) then

    temp_in_ex3 <= temp_in_ex2;    -- shift Temp register
```

```
      temp_in_ex2 <= temp_in_ex1;
      temp_in_ex1 <= temp_in_ex0;

      temp_in_ex0(15 downto 0) <= Temp; --Copy input in extended vector for bigger range
      temp_in_ex0(16) <= Temp(15);     --and duplicate sign bit to MSB
      temp_in_ex0(17) <= Temp(15);     --and duplicate sign bit to MSB

      case state is
      when 0 =>    Average_Temp <= temp_in_ex0(15 downto 0);

      when 1 =>    temp_sum := std_logic_vector(signed(temp_in_ex0) + signed(temp_in_ex1));

            for i in 0 to 15 loop
            Average_Temp(i) <= temp_sum(i+1); -- divide by factor 2
            end loop;

      when 2 => temp_sum := std_logic_vector(signed(temp_in_ex0) + signed(temp_in_ex1) + signed(
    temp_in_ex2) + signed(temp_in_ex3));

            for i in 0 to 15 loop
            Average_Temp(i) <= temp_sum(i+2); -- divide by factor 4
            end loop;

      when others => Average_Temp <= "0000000000000000";

      end case;
    end if;
  end process;
```

4.3 Output

The eight digits of the display have a common anode and individual cathodes to drive the seven segments. (See figure 4.) The digits are addressed periodically. An example for one cycle with four digits is shown in figure 5. To get a steady image a refresh period between 60 Hz and 1 kHz should be used. We set our display clock to 1 kHz.

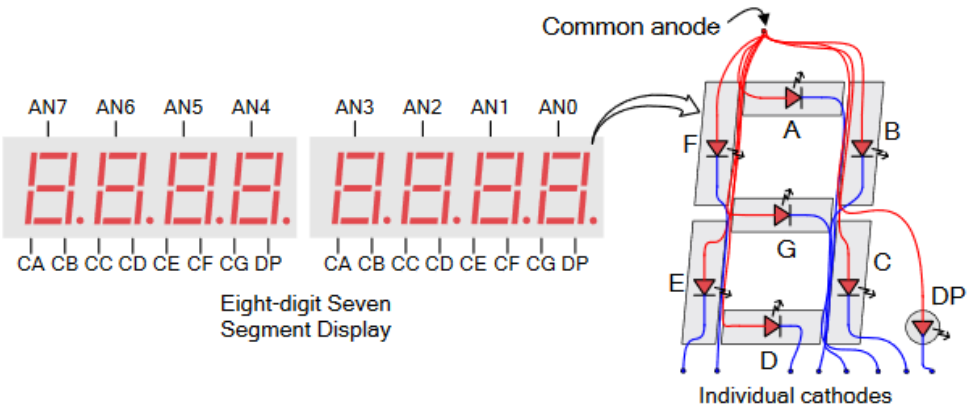

Figure 4: Driving of 7 segment digits

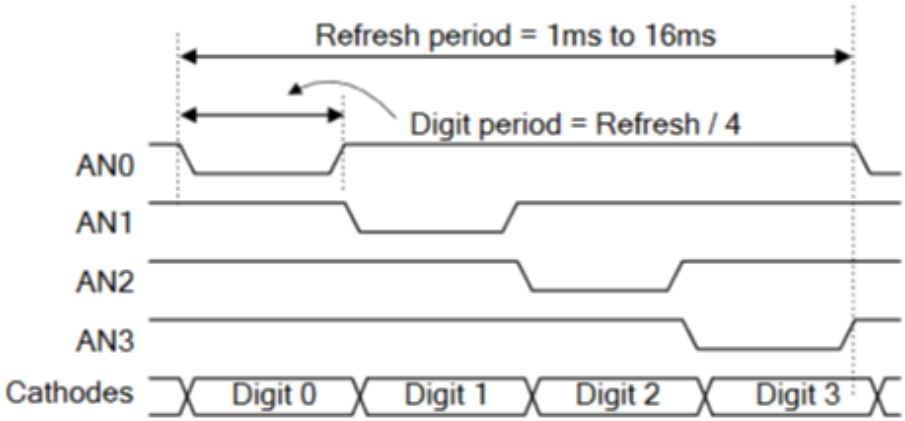

Figure 5: Scanning cycle for 4 digits

To display the temperature we convert the 2's complement std_logic_vector into an array, where each component represents one digit of the display. One LSB of the temperature sensor is equivalent to 0.0078 °C. The algorithm first sets a signbit. Then the 2's complement vector is converted to an integer and multiplied by 78. This value represents now the temperature · 1E4. To get the individual digits, the value is divided by the highest decimal power and saved as integer into the array. Substract this integer times the decimal power from the original value to get rid of the digit and repeat for the next smaller decimal power....

```
    parsing: process(sample_clk)
        variable y: integer := 0;
        variable z : integer;
    begin
        if sample_clk'event and sample_clk = '1' then
            if temp_averaged(15) = '1' then
                signbit <= '1';
            end if;

            y := to_integer(signed(temp_averaged)) * 78;
            y := abs(y);

            z := y/100000;
            seg_numbers(5) <= z;
            y := y - z * 100000;

            z := y/10000;
            seg_numbers(4) <= z;
            y := y - z * 10000;

            z := y/1000;
            seg_numbers(3) <= z;
            y := y - z * 1000;

            z := y/100;
            seg_numbers(2) <= z;
            y := y - z * 100;

            z := y/10;
            seg_numbers(1) <= z;
            y := y - z * 10;
            seg_numbers(0) <= y;
        end if;
    end process;
```

For the representation of the digits we have used the case statement we have already implemented during the instruction of this course, enhanced only for the decimal point. As already said the digits are addressed periodically. This is done by the variable counter. The decimal point is fixed between digit four and five.

```
output: process(display_clk)
        variable dot : std_logic;
    begin
```

```
        if display_clk'event and display_clk = '1' then

            if seg_counter = 4 then
                dot := '0';
            else
                dot := '1';
            end if;

            if seg_counter < 6 then
                case seg_numbers(seg_counter) is
                    ---------------- abcdefg + dot ------------------
                    when 0 => CATHODES <="0000001" & dot;  -- '0'
                    when 1 => CATHODES <="1001111" & dot;  -- '1'
                    when 2 => CATHODES <="0010010" & dot;  -- '2'
                    when 3 => CATHODES <="0000110" & dot;  -- '3'
                    when 4 => CATHODES <="1001100" & dot;  -- '4'
                    when 5 => CATHODES <="0100100" & dot;  -- '5'
                    when 6 => CATHODES <="0100000" & dot;  -- '6'
                    when 7 => CATHODES <="0001111" & dot;  -- '7'
                    when 8 => CATHODES <="0000000" & dot;  -- '8'
                    when 9 => CATHODES <="0000100" & dot;  -- '9'

                    --nothing is displayed when a number more than 9 is given as input.
                    when others=> CATHODES <= "11111111";
                end case;
            elsif seg_counter = 6 then
                if signbit = '1' then
                    CATHODES <= "11111101";
                else
                    CATHODES <= "11111111";
                end if;
            else
                CATHODES <= "11111111";
            end if;

            ANODES <= x"FF";
            ANODES(seg_counter) <= '0';

            seg_counter <= seg_counter + 1;
            if seg_counter = 7 then
                seg_counter <= 0;
            end if;
        end if;
    end process;
```

5 Verification Plan & Simulation

As for the *TempSensorCtl* block we have simply assumed, that it is working as intended, since we took it from the official demo.

For the algorithms for averaging and displaying the data we have created several test cases and observed the outcome of the simulation. Simulation results can be seen in figures 6 & 7.

A simulation of the clock divider is shown in figure 8.

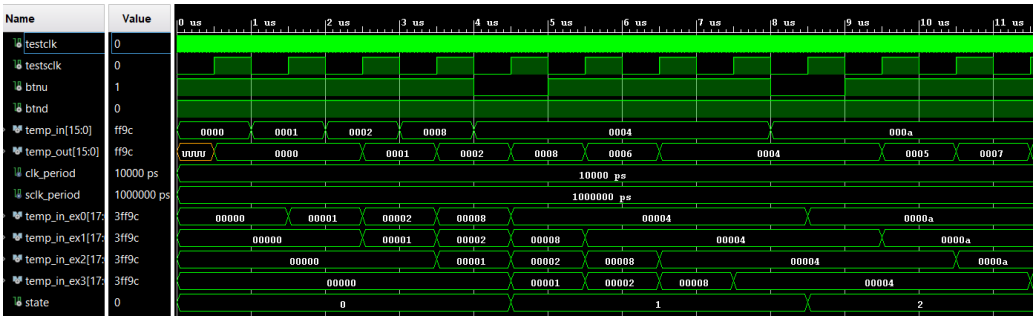

Figure 6: Moving average.

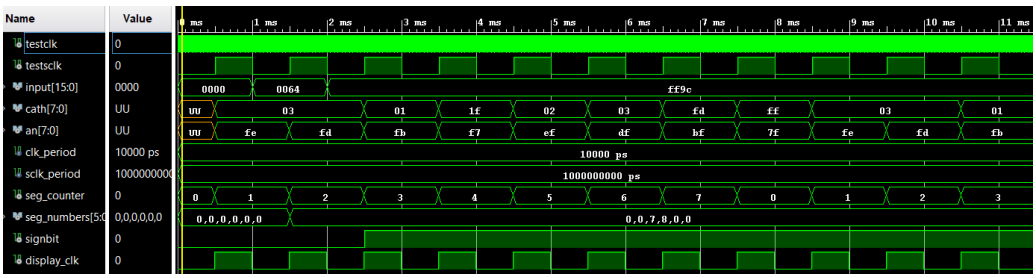

Figure 7: 7 segment display.

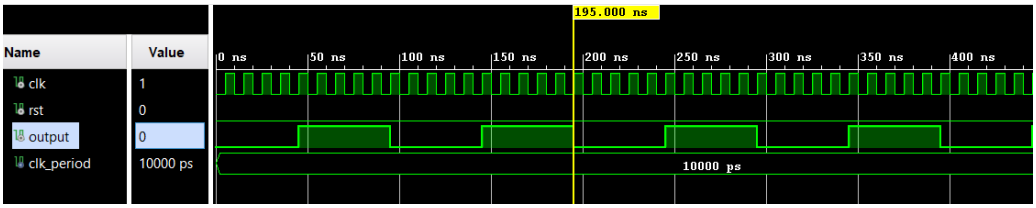

Figure 8: Clock divider.

6 Resource Consumption

Name	Slice LUTs (63400)	DSPs (240)	Bonded IOB (210)	BUFGCTRL (32)	Slice Registers (126800)	Slice (15850)	LUT as Logic (63400)
thermometer	3451	2	35	3	402	1046	3451
mavg (moving_average)	0	0	0	0		12	0
sampleclkgen (clkdivide)	8	0	0	0		13	8
seg7 (whole7segment)	3249	2	0	0		968	3249
dclkgen (clkdivide_param...	6	0	0	0		8	6
tmpsensor (TempSensorCtl)	151	0	0	0		50	151
Inst_TWICtl (TWICtl)	119	0	0	0		40	119

Figure 9: Resource consumption

The ressource consumption is shown in figure 9. The display block needs the most ressources due to the parsing of the temperature vector which needs a lot of computation. However, there are still more than enough ressources available.

7 Timing

The timing report is shown in figure 10. We did not set any timing constraints. Timing values are satisfying though.

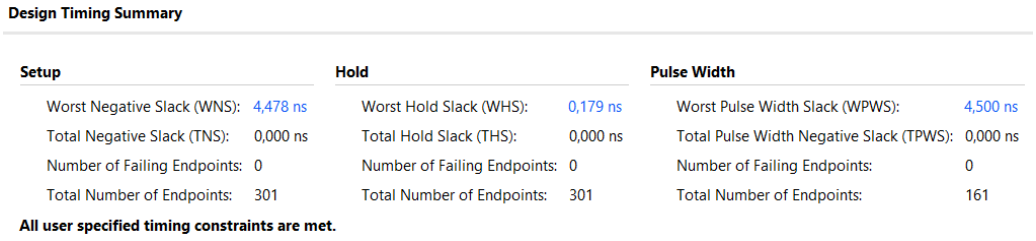

Figure 10: Timing report