UNIVERSITY OF TWENTE.

Faculty EEMCS

Practicum manual

Digital Hardware
(module Computer Systems)

E. Molenkamp

201400217
October 2019
Version 2019 - 1.0

Contents

Change log with version 1.0
-

Practicum manual Digital Hardware

1 Introduction

The goal of this practicum is to design and realize a digital circuit in a Field Programmable Gate Array (FPGA) with the aid of VHDL and modern design tools.

A VHDL tutorial is available that can help you with the VHDL tools that are used for this assignment. The system is divided into subsystems. A subsystem or composition of subsystems in this manual is formulated as a partial assignment.

You must keep your journal up-to-date.

The practicum content and the journal are assessed. Both must be sufficient. The assessment of the practicum content is mainly based on the technical quality of the design. An inadequate assessment cannot be improved.

Before you begin with the lab, it is necessary that VHDL is studied. Without knowledge of VHDL you cannot finish the lab successfully. A student assistant will give no assistance if it appears that you have not sufficiently prepared and / or the VHDL description does not comply with the synthesizable subset.

2 Planning and marking of assignments

A detailed schedule with recommended and hard deadlines is in chapter 8.

Important:
- Before the first lab session you should have installed the software and finished the VHDL tutorial until chapter 5 (see Canvas).
- Be sure that your journal is up-to-date. The student assistant will regularly mark your progress.

Assignment <nr>	**For some assignments it is explicitly stated that the result has to be shown to the student assistant before you continue. These assignments have a colored background identical to this example.**

3 FPGA board

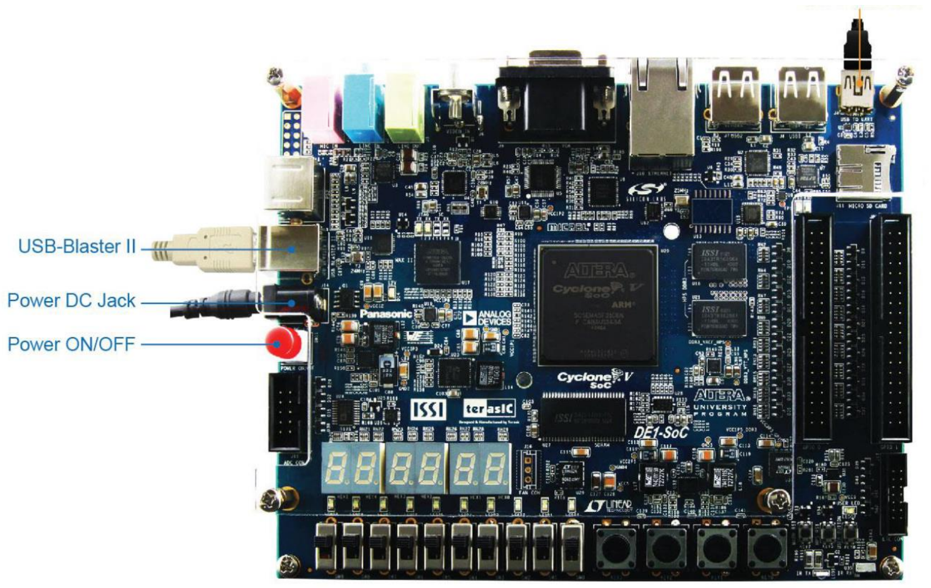


Figure 1: The DE1-SoC board.

For this practicum the DE1-SoC board is used (Figure 1). More information on this board at http://de1-soc.terasic.com

During the lab there are (at most) two boards available. The majority of the time you don’t need a board. Before you use the board you must have simulated your design thoroughly and no serious warnings are reported during synthesis. Therefore the board is only used a few minutes!

4 *VHDL tutorial*

You have installed the software and used the button design before the first lecture Digital Hardware! Only programming the DE1-SoC is postponed until the first lab session.
Because a limited number of DE1-SoC boards are available program the board this afternoon (you don't have to show it to the student assistant). It is important that you can program and test button on the DE1-SoC board.

VHDL tutorial (see Canvas).
Make the assignment chapter 2 until 4 (simulate, synthesize and perform a post simulation) of the *VHDL tutorial*.
Note: select a Cyclone IV device. You will not realize anything on the board, so any Cyclone IV device is fine (post simulation with timing is not supported for a Cyclone V device).

4.1 Flashing_light

An output of the FPGA is connected with a LED (light emitting diode). The LED should be on for about 1 or 2 seconds and off for about 1 or 2 seconds. The system has an input clk that is connected to a crystal with a clock frequency of 50 MHz. Furthermore there is a reset pin. The reset is active low.

The constraint file and a framework of the design are below:

```vhdl
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
ENTITY flashing_light IS
  PORT (clk, reset : IN std_logic;
        led : OUT std_logic);
END ENTITY flashing_light;

ARCHITECTURE bhv OF flashing_light IS

BEGIN
  PROCESS(clk,reset)

  BEGIN
    -- NO VHDL CODE HERE
    IF reset='0' THEN



    ELSIF falling_edge(clk) THEN



    END IF;
    -- NO VHDL CODE HERE
  END PROCESS;



END;
```

Framework flashing light

```
set_global_assignment -name TOP_LEVEL_ENTITY flashing_light
set_global_assignment -name DEVICE 5CSEMA5F31C6
set_location_assignment PIN_AF14 -to clk
set_location_assignment PIN_AA14 -to reset
set_location_assignment PIN_V16  -to led
```

flashing _light.qsf

Assignments:
1. Give a synthesizable description of the flashing light.
2. Simulate your design for 4 seconds (this is a simple test; you can perform the test with a few commands in the transcript window. No VHDL test bench is required).
3. When the simulation is correct synthesize your design and realize it on the DE1-SoC board.

Assignment 0	You must show the flashing light to the assistant (your journal must contain the VHDL description and a relevant part of the waveform).

5 *Global specification*

5.1.1 Introduction

In this lab the DE1-SoC board is used (Figure 5.1). The relevant components are marked.

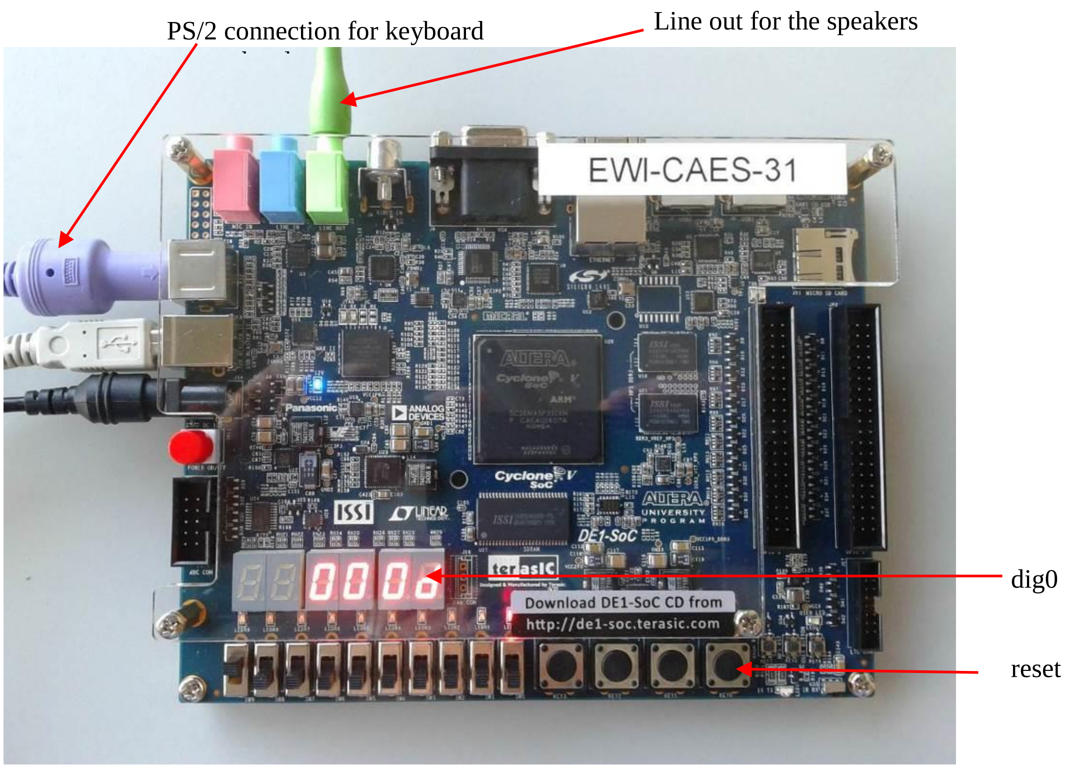


Figure 5.1: DE1-SoC board

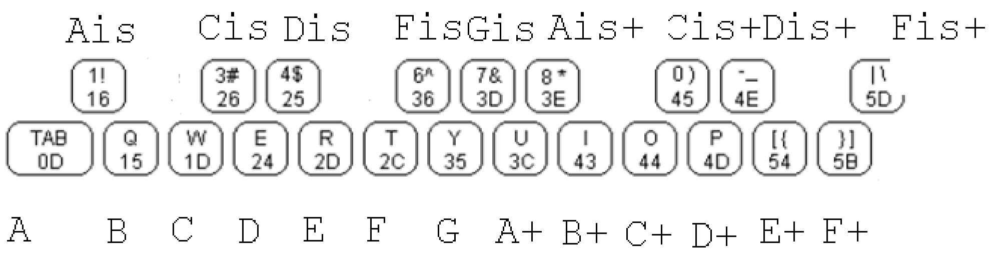


Figure 5.2 PS/2 keyboard with the keys used for the organ

A PS/2 keyboard is connected with the DE1-SoC board. With the keys shown in Figure 5.2, a (simple) melody can be played. The black keys of an organ are the keys on the top row; the white keys correspond to the bottom row.

An organist can press two or more keys simultaneously. This is not supported by our organ!

The background of the musical scale (see also Figure 5.2):

- The musical scale consists of 12 (full / half) tones A, Ais, B, C, Cis, D, Dis, E, F, Fis, G, Gis and again A+ etc..
- The tone A has the frequency 440 Hz.
- The frequency of tone A+, one octave higher, is 880 Hz.

- There is a constant C such that $tone_{i+1}=C\times tone_i$. Furthermore $tone_{i+12}=2\times tone_i$. Hence $C^{12}=2$

  $$C = 2^{\frac{1}{12}} \sim 1{,}05946$$

Every time key ‘A’ is pressed the octave is incremented with one until the highest octave is reached. Every time key 'Z' is pressed all tones are decreased one octave until the lowest octave is reached. If the highest octave is reached then pressing of key ‘A’ has no effect, similar for the lowest octave. In your organ it should be possible to shift over six octaves.

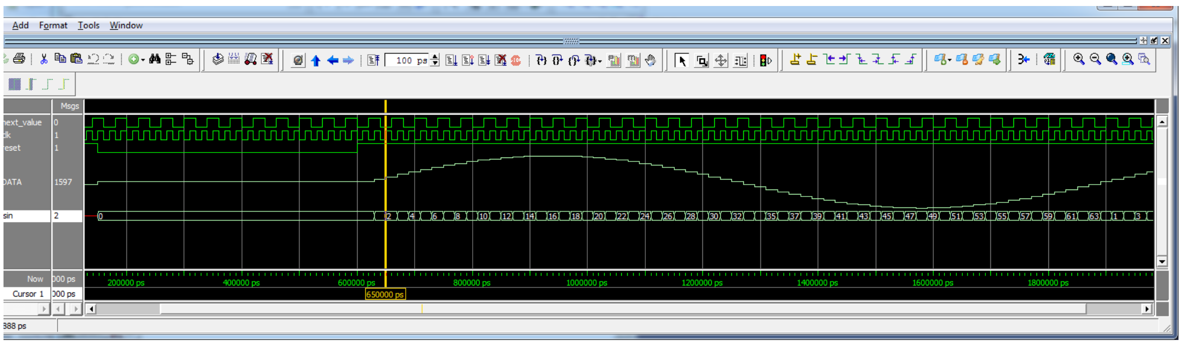


Figure 5.3 Input of the Filter between the AudioL of the FPGA and the speaker

The PS/2 connector is directly connected to the programmable device (an Altera Cyclone V). For the audio the WM8731 audio codec of Wolfson microelectronics is used. An audio interface, developed by Koushik Roy, is used as IP-block. The input (left and right) is a 16 bits signed value. An example of the sine wave you have to generate is shown in Figure 5.3.

5.1.2 Recognize and generate a tone

Undoubtedly there are many ways to realize an organ. In the context of this practicum many pointers are given such that a working organ can be realized in time. You must adhere to the division in subsystems as is presented in this document.

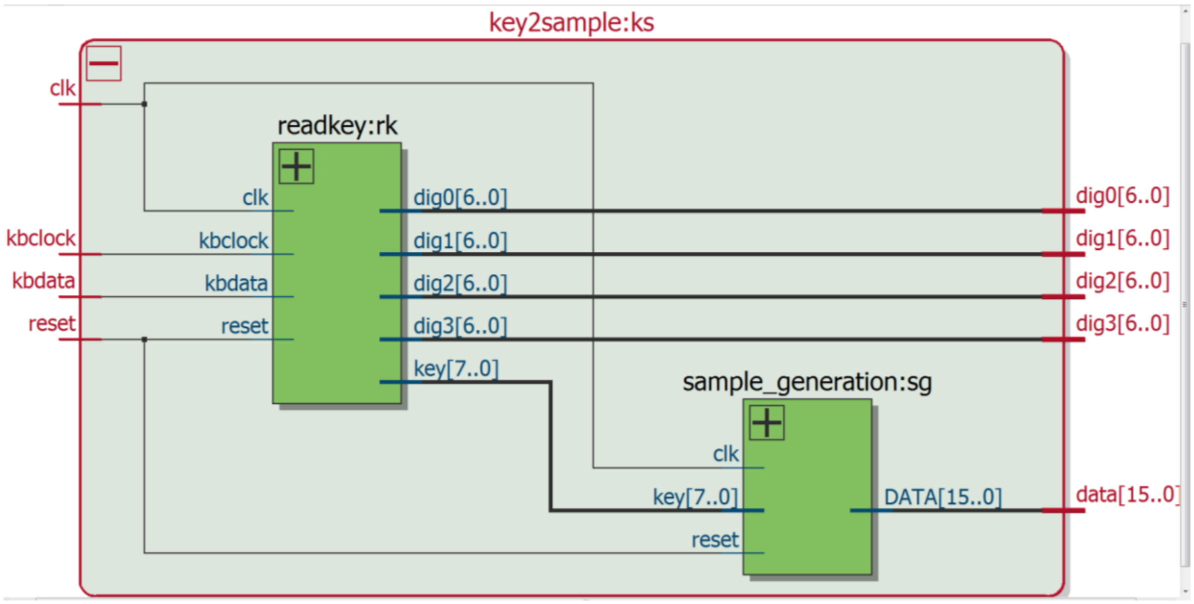


Figure 5.4 Division of the organ into two subsystems(output is connected with the audio interface)

The communication of the keyboard with programmable device is via the signals KBCLOCK and KBDATA. The keyboard generates a code that corresponds to the key that is pressed.

For example if key 'TAB' is pressed the hexadecimal code $0D_{16}$ is generated. When you hold down the key then this code is repeatedly generated. When key 'TAB' is released, two consecutive bytes are transmitted: code $F0_{16}$ followed by code $0D_{16}$ (the code of the key that is released).
This key code is converted in a sine waveform. Figure 5.4 gives the division of the design into two subsystems:

- READKEY
  If a key is pressed, and as long as this key is pressed, the corresponding key code (one byte) is a constantly present at the output of READKEY.
- SAMPLE GENERATION
  A key code not equal to 00_{16} at the input of the sample generation unit will generate a sine wave (the tone) on the output. The generated sine wave is input of the audio interface. When the input of the sample generation unit is 00_{16} the output is 000…0 (no sound).

In Figure 5.4, four additional outputs shown: DIG3 until DIG0 (on the board marked with HEX3 until HEX0). These outputs are connected to four 7-segment displays and are used for debugging (see next chapter).

5.1.3 Readkey

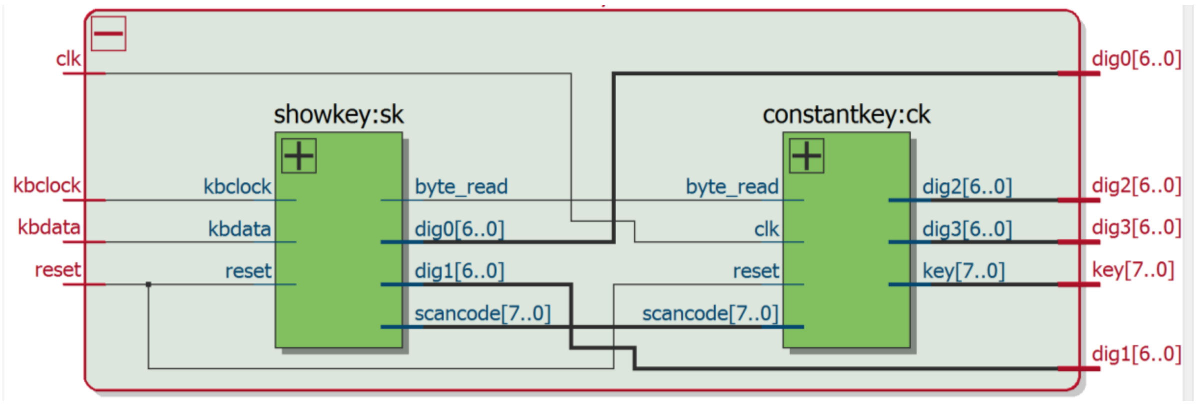


Figure 5.5 Readkey

First you have to study the communication protocol of a PS/2 keyboard: **read the pages 1 to 4 of the document keyboard.pdf**.
We are only interested in the communication from the keyboard to the PC and we restrict ourselves to the keys shown in Figure 5.2 and the keys ‘A’ (octave higher) and ‘Z’ (octave lower).

For all of these keys applies:

- When a key is pressed a byte is generated: < keycode >
- When a key is pressed continuously, this byte repeatedly transmitted: < keycode >,< keycode >..
- When the key is released, two consecutive bytes are sent: $F0_{16}$, < keycode of the released key >

Note: **In this assignment at most one key is pressed on the keyboard.** The protocol is significantly more complex when two or more keys are pressed simultaneously.

5.1.3.1 Showkey

The subsystem SHOWKEY gets its input from the keyboard (KBDATA and KBCLOCK). The synchronous logic of this subsystem operates on the generated clock from the keyboard: KBCLOCK. In addition, this subsystem and also the other subsystems have an asynchronous reset. When KEY0 is pressed the reset is applied. The relevant part of the schematic is given in Figure 5.6. Is the reset low or high active?

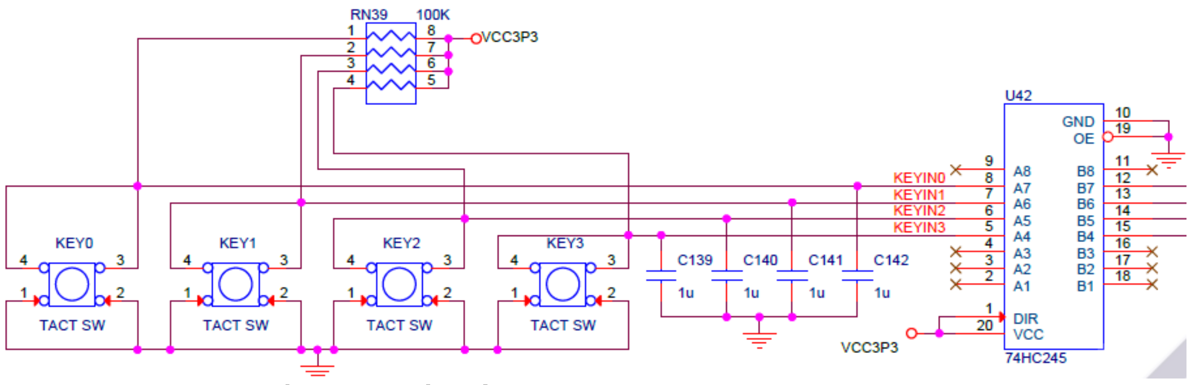


Figure 5.6 Reset circuit on the DE1-SoC board.

If a key of the keyboard is not pressed then the clock line, KBCLOCK, is continuously high. When a key is pressed from our limited set of keys 11 bits are sent:
start bit, keycode (8 bits, from LSB to MSB), odd parity bit, stop bit.

Note 1: In this assignment the parity bit is ignored.

Note 2: You should realize that the data, KBDATA, is stable on the falling edge of KBCLOCK (see also document keyboard.pdf).

The subsystem also generates the signal BYTE_READ. This output signal is '1' when the 11 transmitted bits are received. This signal is used by the subsystem CONSTANTKEY, more on that later. Every time the keyboard sends a byte (e.g. when holding or releasing the key) BYTE_READ is first time '0' and again ‘1’ after it has received the 11 bits.
For debugging, this system has the two 7-segment outputs: DIG0 and DIG1. On DIG0, in hexadecimal notation, the value of the four least significant bits of the received code is shown and DIG1 shows the hex value of the four most significant bits.

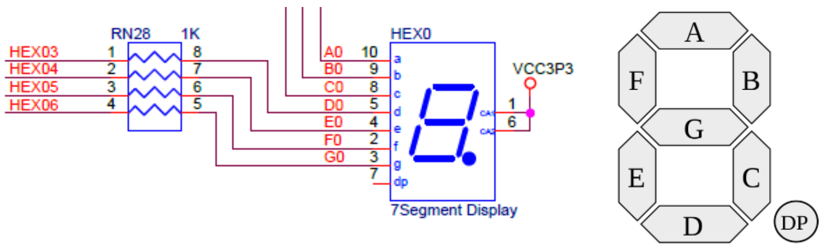


Figure 5.7 Part of schematic of the DE1-SoC board and on the right the usual coding of the LEDs.

5.1.3.2 Constantkey

The subsystem CONSTANTKEY operates on the clock of the DE1-SoC board (50 MHz). As previously described, the output signal BYTE_READ indicates that the SCANCODE is valid.

When a key is pressed, the corresponding scan code is as a constant value available at the output of CONSTANTKEY. There are many ways how this can be implemented. In this assignment the input SCANCODE is used only when in the previous clock cycle (of the 50 MHZ clock) the input BYTE_READ was ‘0’ and the current input value is ‘1’.
The outputs DIG3 and DIG2 show the constant values in hex of the 4 most significant bits and the least significant 4 bits are shown, respectively (used for debugging).

5.1.4 Sample generation

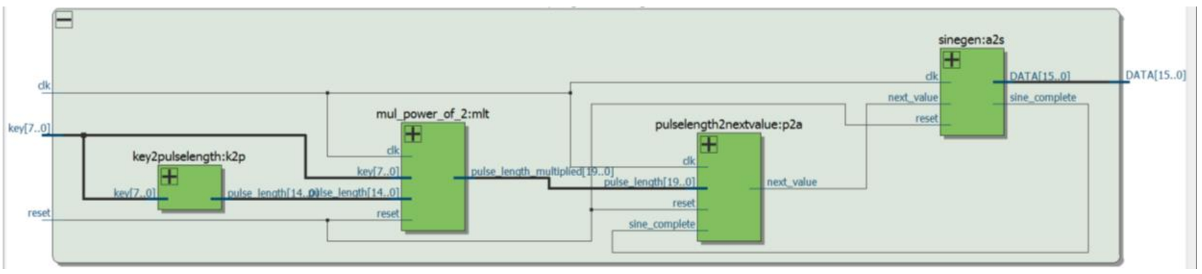


Figure 5.8 Sample generation

The subsystem SAMPLE_GENERATION (figure 5.8) has four subsystems. The subsystem SAMPLE_GENERATION gets its input from the subsystem READKEY (see also Figure 5.4). The behavior of the four subsystems is roughly as follows:

- SINEGEN
  The output of this component is a discrete sine wave (16 bits signed number). The component has a LUT which approximates the sine. When input NEXT_VALUE is '1' the next entry in the LUT is read.
- KEY2PULSELENGTH
  When a key of our organ is pressed a sine wave is generated. The frequency depends on the key pressed. The output PULSE_LENGTH is a constant value (value depends only on the key that is pressed) that is the number of clock cycles before the next entry from the LUT in SINEGEN is read.
- MUL_POWER_OF_2
  When key 'A' or 'Z' is pressed an octave change is realized (an octave higher or lower means that the frequency is doubled respectively halved). The output of this subsystem is PULSE_LENGTH_MULTIPLIED. Again a constant value for a given tone.
- PULSELENGTH2NEXTVALUE
  This subsystem generates the pulse for SINEGEN and depends on this depends on the input PULSE_LENGTH.

5.2 VHDL subset: guidelines that must be used!

In this course, as is often the case in industry, only combinational and synchronous logic is used. If you are deviating from the frameworks beneath **no support of the student assistant is to be expected**. Also make sure that the description is well documented and has a proper indentation.

5.2.1 Synchronous logic

For a synchronous system the following framework must be used:

```
process(reset,clk)
 <here your local declarations>
begin
  <no VHDL code here!!!!>
  if reset='0' then    -- for the DE1-SoC board it must be '0'!
    <reset the variables/signals assigned to in this process>
  elsif falling_edge(clk) then
    <here the behaviour (of course sequential code)>
  end if;
  <no VHDL code here!!!!>
end process;
```

It is sometimes very tempting to respond to a change of an input signal, e.g.

```
  if falling_edge(button) then
```

During simulation, this often seems to be fine but in the real hardware this results in an incorrect operation of the circuit. So only wait for a falling edge of a clock! Do not use a mixture of rising_edge and falling_edge.

5.2.2 Combinational logic

Combinational logic can be modeled with concurrent statements. Sometimes the behavior is modeled better with a sequential description, i.e. using a process statement.
When a process statement is used the following requirements must be taken into account (unfortunately it is not sufficient):

- The process must have a sensitivity list: process (list of signals).
- All signals read in the process has to be in the sensitivity list.
  ModelSim can check this requirement when the compile option "check_synthesis" is enabled (see chapter 6.1.1).
- A variable must be assigned a value before it is read.
- In this process statement don't use: 'event, rising_edge or falling_edge.

5.2.2.1 Combinational loop

If during synthesis the following message is reported:
Warning: Timing Analysis is analyzing one or more combinational loops as latches
then somewhere in your design a memory is modeled that is realized with latches. Correct this first, in this assignment latches are not allowed.

5.2.3 Difference between variables and signals

It cannot be stated often enough:

- Variables are <u>always</u> updated immediately. The assignment operator is :=
- Signals are <u>never</u> updated immediately. The assignment operator is <=
  An exception: for the initial value the assignment operator is :=

5.3 Journal

Make sure that your journal contains at least the following: VHDL descriptions (including VHDL test benches), relevant parts of a waveform (and an explanation!), schematics (RTL view) of the synthesis result and of course the problems and solutions.
A journal is not a report! A journal is a chronological order of activities. A journal has to be up-to-date.

> The student assistant can only control your progress on the basis of your journal.

5.4 Assignments

In this section you will find the assignments. A general description of the subsystems was given in the previous chapters.

5.4.1 Showkey

NOTE: This partial assignment can only be made after section 5.1.3 has been carefully studied. In addition the information on pages 1 to 4 of the document keyboard.pdf is known.

On page 4 of the document keyboard.pdf a time diagram of the PS/2 protocol of the keyboard is given. Depending on the key that is pressed, the corresponding "scan code" is generated (page 3 in document keyboard.pdf). The DATA and CLOCK are connected respectively to the inputs of KBDATA and KBCLOCK of the entity SHOWKEY.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY showkey IS
  PORT (
    reset     : IN std_logic;
    kbclock   : IN std_logic; -- low freq. clk (~ 20 kHz) from keyboard
    kbdata    : IN std_logic; -- serial data from the keyboard
    dig0, dig1: OUT std_logic_vector(6 DOWNTO 0);
    -- shows the key pressed on display in Hex dig1 (upper 4 bits) dig0 (lower 4 bits)
    scancode  : OUT std_logic_vector(7 DOWNTO 0);
    byte_read : OUT std_logic
    );
END showkey;
```

Assignment 1	The entity description of SHOWKEY is given above. Give an architectural description for SHOWKEY. (Use only the subset of VHDL that is supported by the synthesis tool.)

Note: The VHDL files in this document are on Canvas. Do not copy VHDL fragments from this document because the formatting is probably lost.

5.4.1.1 Testbench for Showkey

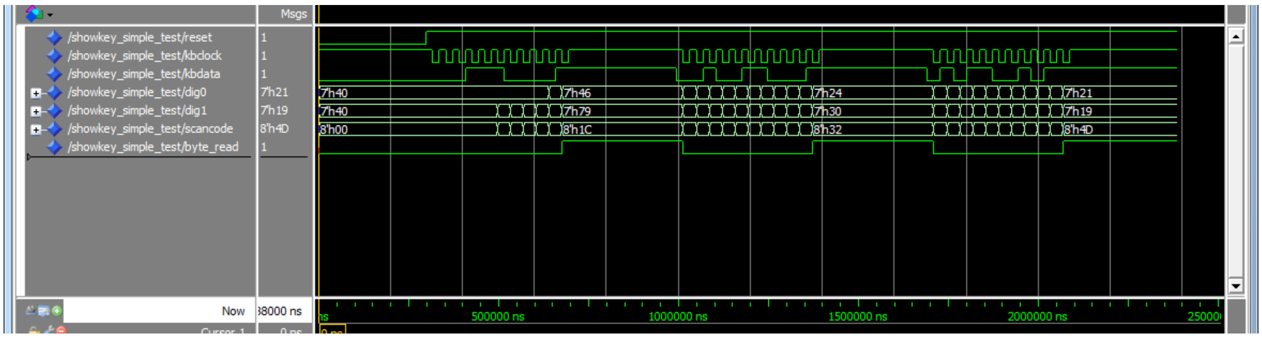
The subsystem SHOWKEY is realized with the DE1-SoC board. However, before the system is realized on this board it must be simulated first to detect design errors. To make this possible a (too) simple test setup is provided; see Figure 5.8 (file showkey_simple_test.vhd).

An explanation of this description:

- Function HEX2DISPLAY
  When a byte is sent, on the display the hexadecimal notation of the 4 most significant and 4 least significant bits of the byte is shown. This function handles the conversion.
  Tip: this function is also synthesizable!
- Procedure SEND_BYTE
  In the test environment data is sent from keyboard to SHOWKEY. The procedure SEND_BYTE composes the data to be sent to the PS/2 bus (start bit, byte, odd parity bit and stop bit) and runs the communication protocol.
- Instantiation of SHOWKEY. This is trivial.
- In a process three bytes are transmitted by the 'keyboard'. After completion of the SEND_BYTE the correct values should be on the output of SHOWKEY.
  With an ASSERT statement, the two outputs DIG0 and DIG1 are compared with the expected values. If an expected value differs from the output of SHOWKEY the message "expected .." is displayed.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY showkey_simple_test IS
END showkey_simple_test;

ARCHITECTURE test OF showkey_simple_test IS
  SIGNAL reset : std_logic := '0';
  SIGNAL kbclock : std_logic := '1';
  SIGNAL kbdata : std_logic := '0';
  SIGNAL dig0, dig1: std_logic_vector(6 DOWNTO 0);
  SIGNAL scancode : std_logic_vector(7 DOWNTO 0);
  SIGNAL byte_read : std_logic;

  FUNCTION hex2display (n:std_logic_vector(3 DOWNTO 0)) RETURN std_logic_vector IS
    VARIABLE res : std_logic_vector(6 DOWNTO 0);
  BEGIN
    CASE n IS          --               gfedcba; low active
            WHEN "0000" => RETURN NOT "0111111";
            WHEN "0001" => RETURN NOT "0000110";
            WHEN "0010" => RETURN NOT "1011011";
            WHEN "0011" => RETURN NOT "1001111";
            WHEN "0100" => RETURN NOT "1100110";
            WHEN "0101" => RETURN NOT "1101101";
            WHEN "0110" => RETURN NOT "1111101";
            WHEN "0111" => RETURN NOT "0000111";
            WHEN "1000" => RETURN NOT "1111111";
            WHEN "1001" => RETURN NOT "1101111";
            WHEN "1010" => RETURN NOT "1110111";
            WHEN "1011" => RETURN NOT "1111100";
            WHEN "1100" => RETURN NOT "0111001";
            WHEN "1101" => RETURN NOT "1011110";
            WHEN "1110" => RETURN NOT "1111001";
            WHEN OTHERS => RETURN NOT "1110001";
    END CASE;
  END hex2display;

  PROCEDURE send_byte (
      CONSTANT byte    : IN std_logic_vector(7 DOWNTO 0);
      SIGNAL kbclock   : OUT std_logic;
      SIGNAL kbdata    : OUT std_logic)
  IS
    VARIABLE odd_parity : std_logic;
    VARIABLE data     : std_logic_vector(10 DOWNTO 0);
    CONSTANT half_period_kbclock : time := 18 us; -- kbclock ~ 27 KHz
  BEGIN
    -- parity
    odd_parity:='1'; -- needed in the next loop to generate ODD parity
    FOR i IN 7 DOWNTO 0 LOOP
      odd_parity := odd_parity XOR byte(i);
    END LOOP;
    data := '1' & odd_parity & byte & '0';
    -- send data
    FOR i IN 0 TO 10 LOOP
          kbdata <= data(i);
          kbclock <= '1';
          WAIT FOR half_period_kbclock;
          kbclock <= '0';
      WAIT FOR half_period_kbclock;
    END LOOP;
    kbclock <= '1';
  END send_byte;

BEGIN
  sk:ENTITY work.showkey PORT MAP (
    reset     => reset,
    kbclock   => kbclock,
    kbdata    => kbdata,
    dig0      => dig0,
    dig1      => dig1,
    scancode  => scancode,
    byte_read => byte_read
  );

  PROCESS
  BEGIN
    reset <='0';
    WAIT FOR 300 us;
    reset <= '1';

    -- key A: 1C (hex) => 0001 1100 of hexadecimaal X"1C"
    send_byte(X"1C", kbclock, kbdata);
    ASSERT (dig1=hex2display(X"1") AND dig0=hex2display(X"C")) REPORT "expected: 1C (hex)" SEVERITY error;
    WAIT FOR 300 us;

    -- key B: 32 (hex) => 0011 0010
    send_byte(X"32", kbclock, kbdata);
    ASSERT (dig1=hex2display(X"3") AND dig0=hex2display(X"2")) REPORT "expected: 32 (hex)" SEVERITY error;
    WAIT FOR 300 us;

    -- key P: 4D (hex) => 0011 0010
    send_byte(X"4D", kbclock, kbdata);
    ASSERT (dig1=hex2display(X"4") AND dig0=hex2display(X"D")) REPORT "expected: 4D (hex)" SEVERITY error;
    WAIT FOR 300 us;
    REPORT "test finished" SEVERITY note;
    WAIT;
  END PROCESS;

END test;
```

Figure 5.8 Test environment for SHOWKEY



Msgs
/showkey_simple_test/reset 1
/showkey_simple_test/kbclock 1
/showkey_simple_test/kbdata 1
/showkey_simple_test/dig0 7'h21
/showkey_simple_test/dig1 7'h19
/showkey_simple_test/scancode 8'h4D
/showkey_simple_test/byte_read 1
7'h40
7'h46
7'h24
7'h21
7'h40
7'h79
7'h30
7'h19
8'h00
8'h1C
8'h32
8'h4D
Now
38000 ns
500000 ns
1000000 ns
1500000 ns
2000000 ns
25000

Figure 5.10 Result of a simulation run of the design of the lecturer

Figure 5.10 shows the simulation result of the design of the lecturer. It is important that DIG0 and DIG1 have the correct values when BYTE_READ is ‘1’. If BYTE_READ is 0 DIG0 and DIG1 can have any value (depends of the design).

Note: the radix of a signal in the wave window can be changed; right click on the name of signal in the wave window, and select radix.

Assignment 2 Test your design with the supplied test environment (figure 5.8)

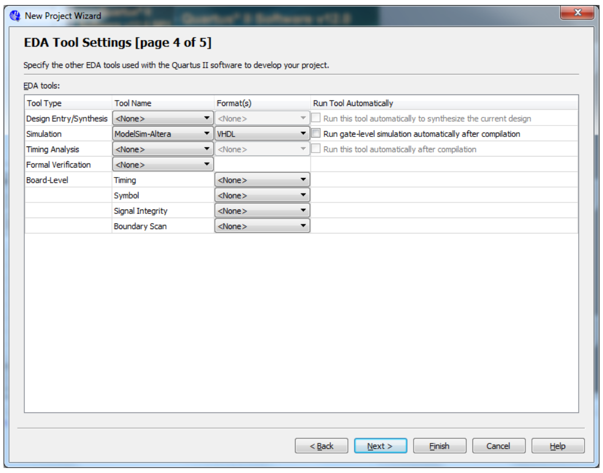
- Compile your SHOWKEY design. Make sure that you do not change the entity description.
- Compile the file with the test environment: showkey_simple_test.vhd
- Simulate SHOWKEY_SIMPLE_TEST. After the design is loaded in the simulator apply the following commands:
    - add wave *
    - run -all

5.4.2 Realization of Showkey

If you did not violate any of the synthesis rules then your design should be synthesizable.

5.4.2.1 Synthesis with met Quartus.

Before you start Quartus file showkey.qsf (with the pin mappings) should be in the project directory!
Synthesize your design with Quartus with the Cyclone V device 5CSEMA5F31C6N.



EDA Tool Settings [page 4 of 5]

Specify the other EDA tools used with the Quartus II software to develop your project.

EDA tools:

Tool Type	Tool Name	Format(s)	Run Tool Automatically
Design Entry/Synthesis	<None>	<None>	☐ Run this tool automatically to synthesize the current design
Simulation	ModelSim-Altera	VHDL	☐ Run gate-level simulation automatically after compilation
Timing Analysis	<None>	<None>	☐ Run this tool automatically after compilation
Formal Verification	<None>		
Board-Level	Timing	<None>	
	Symbol	<None>	
	Signal Integrity	<None>	
	Boundary Scan	<None>	

Figure 5.11: EDA Tool Setting

A post simulation is to be performed therefore you have to instruct Quartus to generate post simulation files.

Note: Post simulation with timing is not supported for Cyclone V devices (the standard delay output file, *.sdo, is not generated).

5.4.2.2 Performing a functional post simulation

As a result of synthesis a directory "..\simulation\modelsim" is created that contains a vhdl file (*.vho). This file can be used with the same test environment (file showkey_simple_test.vhd) to perform a functional post simulation.

Assignment 3	Perform a functional post simulation using the test environment (figure 5.8). You must show the correct operation to the student assistant (your journal has to be up-to-date!).

Note 1: A generated post simulation VHDL file is a model of the hardware that is realized. In the real hardware there is not an "integer" only a bundle of wires that has an integer representation. Therefore the types in the post simulation VHDL file are sometimes different, e.g.:
```
inp : IN integer range 0 to 3
```
is changed into
```
inp : IN std_logic_vector(1 downto 0)
```
In the latter situation, you must slightly modify the test environment.

5.4.2.3 Program the DE1-SoC board.

Connect your laptop with the DE1-SoC board and program the board with your design.

Note: If you did not used file showkey.qsf the result is not correct. When you compile your design again in Quartus be sure to remove all intermediate file and a place showkey.qsf in the project directory before you start Quartus.

Assignment 4	Program the FPGA Since we did not explicitly specify the state of unused pins of the device not used LEDS and SEGMENTS on the board may be on or off.

Assignment 5	You must show the correct operation of SHOWKEY to the student assistant (your journal has to be up-to-date!).

5.4.3 Constantkey

In section 5.1.3.2 the behavior of CONSTANTKEY is described.
The entity description is:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY constantkey IS
  PORT (
    reset      : IN std_logic;
    clk        : IN std_logic; -- 50MHz clock
    scancode   : IN std_logic_vector(7 DOWNTO 0);
    byte_read  : IN std_logic;
    dig2, dig3 : OUT std_logic_vector(6 DOWNTO 0);
               -- show key pressed on display dig2 and dig3 (resp high & low).
    key        : OUT std_logic_vector(7 DOWNTO 0)
    );
END constantkey;
```

System SHOWKEY is a synchronous system that operates on the clock generated by the keyboard (~20 kHz). System CONSTANTKEY is also a synchronous system, but operates on an independent clock of 50 MHz.
Input BYTE_READ is a control signal. If the current value of this input is ‘1’ and it was ‘0’ in the previous clock cycle then a new SCANCODE is available. This is an easy way to detect a new scancode (NEW_SCANCODE_DETECTED). However since the clocks of both systems are independent it is possible that BYTE_READ changes near the active edge of the 50 MHz clock. Hence, BYTE_READ is an asynchronous input of system CONSTANTKEY.

Assignment 6

In the architecture of CONSTANTKEY a PROCESS is used that generates the signal NEW_SCANCODE_DETECTED. This signal is exactly one clock period ‘1’ when the current value of synchronized BYTE_READ signal is ‘1’ and the previous value is ‘0’

```
-- detection of a new scancode; with BYTE_READ synchronization
PROCESS(clk,reset)
  ..
BEGIN
  IF reset='0'THEN
    ..
    new_scancode_detected<=false;
  ELSIF rising_edge(clk) THEN
    ..
    new_scancode_detected<= …
  END IF;
END PROCESS;
```

- Give a VHDL description of this PROCESS statement
- how would you realize this in hardware (give a schematic)?

In section 5.1.3 a detailed description of pressing and releasing a key is given.
In system CONSTANTKEY the output KEY should be constant if the same scancode is on the input SCANCODE (if the key on the keyboard is pressed a relatively long time).

Assignment 7	Draw a state machine with inputs SCANCODE and NEW_SCANCODE_DETECTED. The output is 00_{16} if no key is pressed or the scancode of the key that is pressed. Hint: read both notes beneath.

Note 1: The scancode has 8 bits, hence 2^8 different patterns. If you draw a state machine with the 256 possible patterns for scancode it will be huge. Realize that the protocol of pressing the key, and possibly keep pressing the key, until the release of the key is almost independent of the value of the scancode. The relevant information of the scancode is: the scancode is $F0_{16}$ (start of the release pattern) or a different value. The crucial part of the state machine can be modeled with a few states.

Note 2: When you release the button, two bytes are sent: $F0_{16}$ followed by a keycode. If the organ is used correctly, i.e. keys are not pressed simultaneously the keycode after $F0_{16}$ is the same as the key that was pressed the last time. If you press keys simultaneously, the keycode after $F0_{16}$ is not trivial. Of course if you accidently press keys at the same time it would be nice if your organ can handle that.

Assignment 8	Show the finite state machine diagram (no VHDL) to the student assistant (your journal has to be up-to-date!).

Assignment 9	Give a VHDL description of architecture CONSTANTKEY. Use only the VHDL synthesis subset.

Assignment 10	A simple VHDL test environment is available for CONSTANTKEY: file constant_simple_test.vhd. Test your design and add a relevant part of the waveform in your journal

Assignment 11	Synthesize CONSTANTKEY with Quartus. The design is <u>not yet</u> realized on the DE1-SoC board, but synthesis problems can be identified.

Note 1: The design has two unrelated clocks: the keyboard clock (*kbclock*) and the 50 MHz clock that is on the DE1-SoC (*clk*). In the next section it is explained how we can inform the synthesis tool on this.

5.4.4 Readkey

In section 5.1.3 the behavior of READKEY is described.
The entity description is:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY readkey IS
  PORT (
    clk        : IN std_logic; -- high freq. clock (~ 50 MHz)
    reset      : IN std_logic;
    kbdata     : IN STD_LOGIC; -- low freq. clk (~ 20 kHz) serial data from the keyboard
    kbclock    : IN STD_LOGIC; -- clock from the keyboard
    key        : OUT std_logic_vector(7 DOWNTO 0);
                 -- shown on 8 Leds
    dig0, dig1 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
                 -- show key pressed on display
    dig2, dig3 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
                 -- show key pressed on display; after processed by CONSTANTKEY
   );
END readkey;
```

Assignment 12	Give a VHDL description of architecture READKEY. Use only the VHDL synthesis subset.

Assignment 13	Verify READKEY with the provided test environment in file readkey_simple_test.vhd.

Note 1: The design has two unrelated clocks: the keyboard clock (*kbclock*) and the 50 MHz clock that is on the DE1-SoC (*clk*). In the file readkey.sdc the relationship between the two clocks is specified. If this file is in the project directory before you start Quartus the tool will use this information.
(Hence, readkey.sqf and readkey.sdc file should be in the project directory before you start Quartus)

Assignment 14	Synthesize READKEY with Quartus and realize it on the DE1-SoC board (don't forget the provided constraint files: **readkey.qsf** and **readkey.sdc** before you start Quartus ☺).

Assignment 15	You must show the correct operation of READKEY to the student assistant (your journal has to be up-to-date!).

5.4.5 Key2pulselength

In section 5.1.4 the behavior of KEY2PULSELENGTH is described.
The entity description is:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
ENTITY key2pulselength IS
  GENERIC (max_length : integer := 20000);
  PORT (key     : IN std_logic_vector(7 DOWNTO 0);
        pulse_length : OUT INTEGER RANGE 0 TO max_length
       );
END key2pulselength;
```

In this partial assignment the mapping of the relevant keys of the organ to the frequency of the sine is determined. You may assume that the component SINEGEN generates a discrete sine in 64 steps (clock cycles).

If an invalid key is pressed the output has integer value 0.
(Remember: when the output of KEY2PULSELENGTH key is 0 no tone is generated.)

In figure 5.8 (section 5.1.4) it is shown that input KEY is also connected to system MUL_POWER_OF_2. In MUL_POWER_OF_2 input KEY is used to increase or decrease an octave. In subsystem KEY2PULSELENGTH the output PULSE_LENGTH is integer 0 when key 'A' or 'Z' is pressed.

Assignment 16	Give a VHDL description of architecture KEY2PULSELENGTH. Use only the VHDL synthesis subset. Hint: use constants like CONSTANT key_Tab : std_logic_vector := X"0D";

Assignment 17	Verify KEY2PULSELENGTH. Since this is a very simple design (I hope!) a real VHDL test environment is not required. You may use a script file with the ModelSim simulation commands.[1] Add your script file and relevant part of the waveform in your journal.

Assignment 18	Synthesize KEY2PULSELENGTH with Quartus. The design is not yet realized on the DE1-SoC board, but synthesis problems can be identified.

[1] The contents in the transcript windows are save to a file with the command “write transcript <name_of_file>”. You can edit this file. To execute this file type in the transcript window “do <name_of_file>.

5.4.6 Multiply with a power of 2

In section 5.1.4 the behavior of MUL_POWER_OF_2 is described.
When input KEY has the scancode of character 'A' or 'Z' the output PULSE_LENGTH_MULTIPLIED should be doubled respectively halved.

The entity description is:

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY mul_power_of_2 IS
  GENERIC (max_length : integer := 20000);
  PORT ( clk                     : IN std_logic;
         reset                   : IN std_logic;
         key                     : IN std_logic_vector(7 DOWNTO 0);
         pulse_length            : IN integer RANGE 0 TO max_length;
         pulse_length_multiplied : OUT integer RANGE 0 TO max_length*32
              );
END mul_power_of_2;
```

Note: When a key is pressed it is detected a number of contiguous clock cycles in this subsystem but it should of course increment/decrement the octave only once.

Assignment 19	Give a VHDL description of architecture MUL_POWER_OF_2. Use only the VHDL synthesis subset.

Assignment 20	Verify MUL_POWER_OF_2. Since this is a very simple design a VHDL test environment is not required. You may use a script file with the ModelSim simulation commands Add your script file and relevant part of the waveform in your journal

Assignment 21	Synthesize MUL_POWER_OF_2 with Quartus. The design is not yet realized on the DE1-SoC board, but synthesis problems can be identified.

5.4.7 Pulselength2nextvalue

In section 5.1.4 the behavior of PULSELENGTH2NEXTVALUE is described.
The entity description is:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY pulselength2nextvalue IS
  GENERIC (max_length : integer := 20000);
  PORT  (clk           : IN std_logic;
         reset         : IN std_logic;
         sine_complete : IN std_logic;
         pulse_length  : IN INTEGER RANGE 0 TO max_length;
         next_value    : OUT std_logic
  );
END pulselength2nextvalue;
```

Every PULSE_LENGTH clock cycles the output NEXT_VALUE is ‘1’ for 1 clock cycle.
When input PULSE_LENGTH has integer value is 0 NEXT_VALUE is ‘0’.

When PULSE_LENGTH is changed the current sine cycle is completed before the new value of PULSE_LENGTH is used. The input SINE_COMPLETE is ‘1’ when a sine cycle is complete.

Assignment 22	Give a VHDL description of architecture PULSELENGTH2NEXTVALUE. Use only the VHDL synthesis subset.

Assignment 23	Verify PULSELENGTH2NEXTVALUE . Since this is a very simple design a VHDL test environment is not required. You may use a script file with the ModelSim simulation commands Add your script file and relevant part of the waveform in your journal

Assignment 24	Synthesize PULSELENGTH2NEXTVALUE with Quartus. The design is not yet realized on the DE1-SoC board, but synthesis problems can be identified.

5.4.8 Sinegen

In section 5.1.4 the behavior of SINEGEN is described.
The entity description is:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;
ENTITY sinegen IS
  PORT(
    next_value    : in std_logic;
    clk           : in std_logic;
    reset         : in std_logic;
    sine_complete : OUT std_logic;
    DATA          : OUT std_logic_vector(15 downto 0)
  );
END sinegen;
```

The sine is approximated with a discrete wave using 64 time steps. For the approximation a look-up table is used, however not all 64 entries are stored in this look-up table. Due to the properties of the sine about a quarter of the values are stored in a LUT; not stored values are derived from the stored values in the LUT. The amplitude of the sine is approximately 8000 (decimal) and 16 bits signed representation is used.
When a complete cycle is finished the SINE_COMPLETE output is '1' else it is '0'. Why is the value of SINE_COMPLETE '1' when an asynchronous reset is applied?

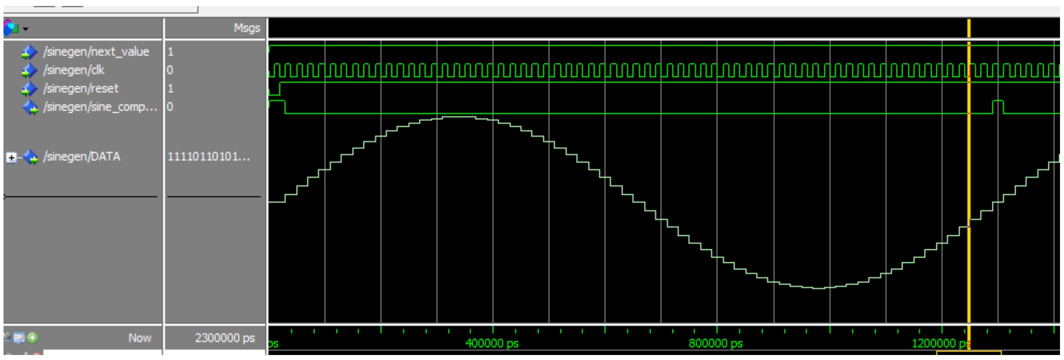


Figure 5.12: The discrete sine wave generated by SINEGEN

Assignment 25	Determine the content of the Look-up table (in your journal) and show it to the student assistant.

Assignment 26	Give a VHDL description of the architecture SINEGEN. Use only the VHDL synthesis subset.

Assignment 27	Verify SINEGEN. You may use a script file with the ModelSim simulation commands Add your script file (or VHDL test environment) and relevant part of the waveform in your journal.

Assignment 28	Synthesize SINEGEN with Quartus. The design is not yet realized on the DE1-SoC board, but synthesis problems can be identified.

5.4.9 Sample generation

In section 5.1.4 the behavior of SAMPLE_GENERATION is described.
The entity description is:

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY sample_generation IS
  PORT (
    clk   : IN std_logic;
    reset : IN std_logic;
    key   : IN std_logic_vector (7 DOWNTO 0);
    DATA  : OUT std_logic_vector(15 downto 0)
        );
END ENTITY sample_generation;
```

Assignment 29	Give a VHDL description of the architecture SAMPLE_GENERATION. Use only the VHDL synthesis subset.

Assignment 30	Verify SAMPLE_GENERATION. You may use a script file with the ModelSim simulation commands Add your script file (or VHDL test environment) and relevant part of the waveform in your journal.

Assignment 31	Synthesize SAMPLE_GENERATION with Quartus. The design is <u>not yet</u> realized on the DE1-SoC board, but synthesis problems can be identified.

5.4.10 Key2sample

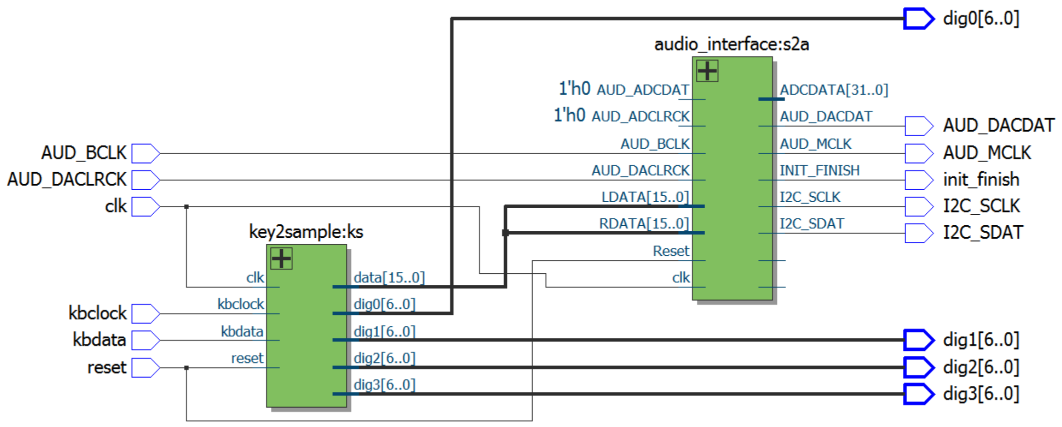


Figure 5.13: Organ

In figure 5.13 a structural description of the organ is shown. The VHDL file of this structural description is available. Also the VHDL file of AUDIO_INTERFACE is available. The entity description of KEY2SAMPLE is:

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY key2sample IS
  PORT (
    clk          : IN std_logic;
    reset        : IN std_logic;
    kbdata       : IN STD_LOGIC; -- low freq. clk (~ 20 kHz) serial data from the keyboard
    kbclock      : IN STD_LOGIC; -- clock from the keyboard
               -- debug outputs
    dig0, dig1   : OUT STD_LOGIC_VECTOR(6 DOWNTO 0); -- show key pressed on display
    dig2, dig3   : OUT STD_LOGIC_VECTOR(6 DOWNTO 0); -- show key pressed on display

    -- audio codec in and outputs
    data        : OUT std_logic_vector(15 downto 0)
);
END ENTITY key2sample;
```

The KEY2SAMPLE design contains the components READKEY and SAMPLE_GENERATION (see figure 5.4).

Assignment 32 Create a VHDL structural description of KEY2SAMPLE.
Use only the VHDL synthesis subset.

Assignment 33 Verify KEY2SAMPLE with a VHDL test environment. A VHDL test environment is not provided (you may also use a script file).
Keep the test simple since a simulation run takes a long time!
The test environment or script file and a relevant part of the waveform must be in your journal.

Assignment 34 Synthesize KEY2SAMPLE with Quartus. The design is not yet realized on the DE1-SoC board, but synthesis problems can be identified.

5.4.11 Organ

Until now the design has two unrelated clocks: the keyboard clock (kbclock) and the 50 MHz clock that is on the DE1-SoC (clk). An additional clock, derived from the 50 MHz clock, is required for the audio interface for the WM8731 audio codec (chapter 5.4.11). In the file organ.sdc the relationship between the three clocks is specified.

Assignment 35	Synthesize ORGAN with Quartus and realize it on the DE1-SoC board (don't forget the constraint files **organ.qsf** and **organ.sdc**).

Assignment 36	You must show the correct operation of your organ to the student assistant (your journal has to be up-to-date!).

6 Known tool issues

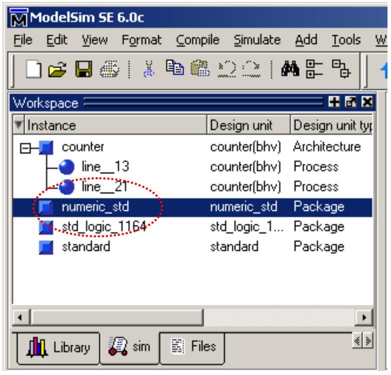
6.1 ModelSim

6.1.1 You did not restrict to the synthesis subset of VHDL

A common error is that not all signals are in the sensitivity list of a process that models combinational logic. ModelSim can check this constraint.
Go to "Compiler Options" and check then enable option "Check for synthesis".

6.1.2 No signals in the waveform?

If after the command "add wave *", the signals are not shown in the wave window, it is likely that in the left-hand window not the design (in this example, counter) is selected but a package.



Instance	Design unit	Design unit typ
counter	counter(bhv)	Architecture
line__13	counter(bhv)	Process
line__21	counter(bhv)	Process
numeric_std	numeric_std	Package
std_logic_1164	std_logic_1...	Package
standard	standard	Package

6.2 Quartus

During the synthesis many files are generated. These files are also in the project directory. Also the constraint file (*.qsf) is in this directory and changed by Quartus. If you have selected the wrong device than that device is added in the constraint file and in the next synthesis run a wrong technology is used (or a mismatch is reported). Keep a copy of the constraint file elsewhere and copy it to the project directory before you start Quartus again.

7 Planning of the lab sessions in 2019

Assignment		Recommended Deadline & **Hard deadline**
	Install software on your laptop (Quartus and ModelSim) and program the device. See Canvas.	Monday morning 30 September; 10.45-12.30
	In DH self-study hours before fist lab session : VHDL tutorial; chapters 2 until 4	
0	Blinking light + program FPGA	3 Oct.
1	Showkey (VHDL)	
2	Test	
3	Post simulation showkey	
4	Program FPGA	
5	Show result to student assistant	10 Oct.
6	New scancode detected (VHDL; only PROCESS statement)	
7	State machine diagram (no VHDL)	
8	Show state machine to student assistant	15 Oct.
9	Constant key (VHDL)	
10	Simulation	
11	Synthesis	
12	Readkey (VHDL)	
13	Verify	
14	Synthesis and program FPGA	
15	Show result to student assistant	17 Oct.
16	Key2Pulselength (VHDL)	
17	Verify	
18	Synthesis	
19	Multiply with power 2 (VHDL)	
20	Verify	
21	Synthesis	
22	Pulselength2nextvalue (VHDL)	
23	Verify	
24	Synthesis	
25	Sinegen Show content LUT to student assistant	18 Oct.
26	VHDL description	
27	Verify	
28	Synthesis	
29	Sample generation (VHDL)	
30	Verify	
31	Synthesis	22 Oct.
32	Key2sample Structural description	
33	Verify	
34	Synthesis	
35	Organ Synthesis	
36	Program FPGA and show correct operation to assistant (journal is up-to-date)	23 Oct. **Hard deadline 24 Oct. (8.45-10.30).**