UNIVERSITY OF TWENTE.

Faculty EEMCS

Practicum manual

Fundamentals of Digital Logic

E. Molenkamp

Vakcode: 192130014
March 2013
Version 1.0

Contents

Practicum manual Fundamentals of Digital Logic

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. For the intermediate assignments the system is already divided into subsystems. A subsystem or composition of subsystems in this manual is formulated as a partial assignment. In the final assignment the intermediate assignment is extended with storage facility.

During the tasks related to the intermediate and final assignment you must keep your journal up-to-date.

The practicum content and the journal are assessed. Both should 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 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*

There are 8 lab sessions of 4 hours scheduled for the lab. Overall, the following schedule will be used. A detailed schedule with DEADLINES is in chapter 8.

Important:

- Before the first lab session you should have installed the software (see Blackboard; course information → software) and a VHDL tutorial (see Blackboard; Course materials→Practicum→software and VHDL tutorial before first lab session)
- The deadline for the intermediate assignment is lab session 7
- For a successful completion of the final project you should have finished successfully the intermediate assignment in lab session 5.
- 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 like this example.**

3 FPGA development board

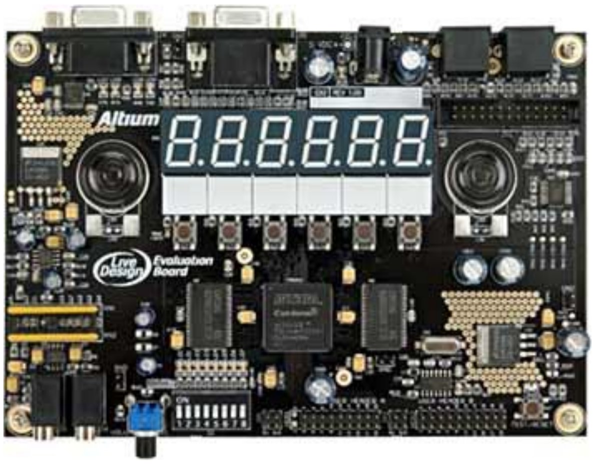


Figure 1: The LiveDesign Evaluation Kit with an Altera device.

For this practicum the LiveDesign Evaluation Kit is used (Figure 1). More information on this board can be found at www.cs.utwente.nl/~molenkam/midP

There are a limited number of kits available. The majority of the time you don't need this kit. Before you use it you must simulate your design properly and no serious warnings are reported during synthesis. Therefore the evaluation kit is only used a few minutes!

4 *VHDL tutorial*

Read and make the assignments in the VHDL tutorial chapters 1 t/m 6. Programming the device (chapter 6) is covered by assignment 0 in this manual.

An output of the FPFA 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 (active low) pin.

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 blinking_light IS
  PORT (clk, reset : IN std_logic;
        led : OUT std_logic);
END ENTITY blinking_light;

ARCHITECTURE bhv OF blinking_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 blinking light

```
set_location_assignment PIN_J16 -to clk
set_location_assignment PIN_U14 -to reset
set_location_assignment PIN_R1  -to led
```

blinking_light.qsf

Assignments:
1. Give a synthesizable description of the blinking light.
2. Simulate your design for 4 seconds.
3. If the simulation is correct then synthesize your design and realize it on the LiveDesign Evaluation Kit.

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

5 *Global specification*

5.1.1 Introduction

In this lab the LiveDesign Evalaution Kit is used (Figure 5.1). The relevant components are marked.

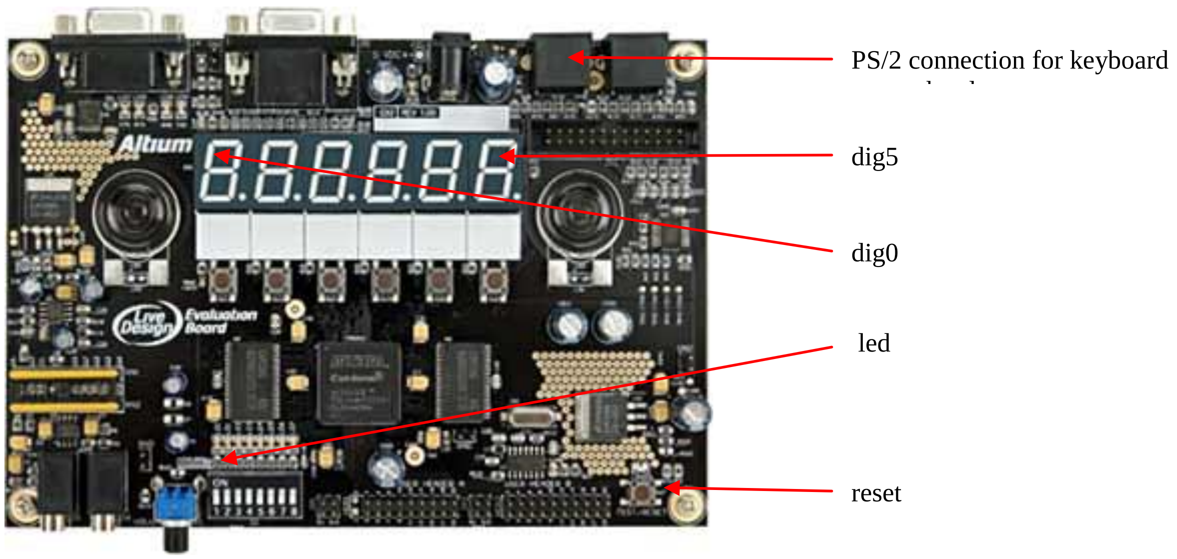


Figure 5.1:LiveDesign Evaluation Kit

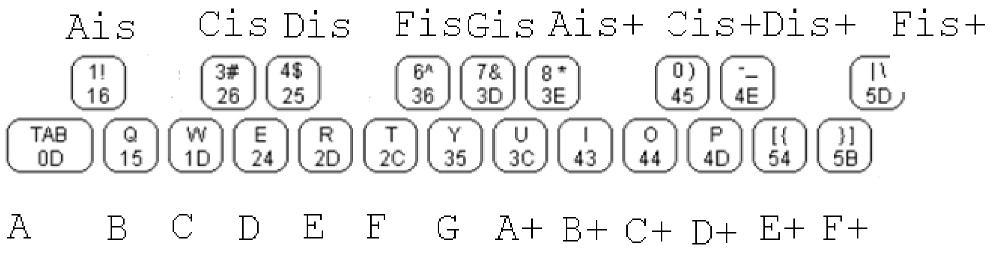


Figure 5.2 PS/2 keyboard with the keys used for the organ

A PS/2 keyboard is connected with the LiveDesign Evaluation Kit. With the keys shown in Figure 5.2, a (simple) melody can be played. The black notes 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! As compensation, your musical qualities are not part of the assessment.

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 twice that of tone A: 880 Hz.
- There is a constant C such that $\text{tone}_{i+1}=C\times\text{tone}_i$. Furthermore $\text{tone}_{i+12}=2\times\text{tone}_i$.
  Hence $C^{12}=2$

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

Every time key 'A' is pressed the octave is incremented bij 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.

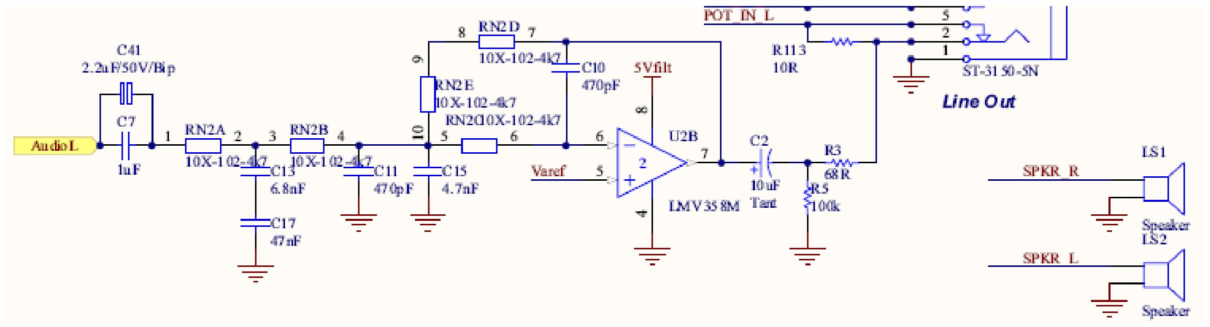


Figure 5.3 Filter between the AudioL of the FPGA and the speaker

The PS/2 connector is directly connected to the programmable device (an Altera FPGA). The two output pins (stereo) for audio are not directly connected to the speakers. Figure 5.3 shows the analog circuit that is on the board. This is an interesting circuit, but not for this course.

For this assignment, the following is sufficient:

> A tone with a frequency of 100 Hz has a time period time of 10 ms. If the output AudioL has a periodic signal of 5ms high (logic '1') and 5 ms low (logical '0'), the tone of 100 Hz generated.

The filter ensures that the generated square wave at the output AudioL (and also Audio R) of the programmable device is converted into an approximately sine wave with the same period of time.

5.1.2 Recognize and generate a tone

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

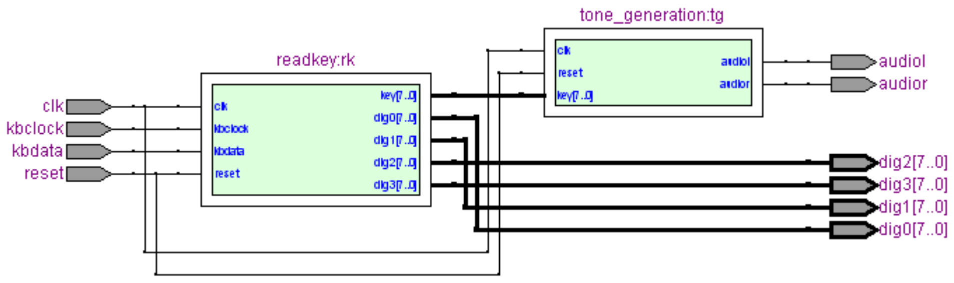


Figure 5.4 Division of the organ into two subsystems

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 is generated. When you hold down the key then this code is repetitively generated. When key 'TAB' is released, two consecutive bytes are transmitted; namely code F0 followed by code 0D (the code of the key that is released).
This key code is converted into a tone. 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.

- TONE GENERATION
  A key code not equal to 00_{16} at the input of the tone generation unit will generate a square wave (~ tone) on the two audio outputs pins (AUDIOL and AUDIOR are the same; mono). When the input of the tone generation unit is 00_{16} both audio outputs are constant '0' (no sound).

In Figure 5.4, there are 4 additional outputs shown: DIG0 t/m DIG3. These outputs are connected to four 7-segment displays and are used for debugging.
The outputs DIG0 and DIG1 show, in hexadecimal notation, the last generated code from the keyboard. This is used for 'verification' of unit SHOWKEY. The outputs DIG2 and DIG3 show the constant key code, in hexadecimal notation, while it shows 00_{16} if no key is pressed.

5.1.3 Readkey

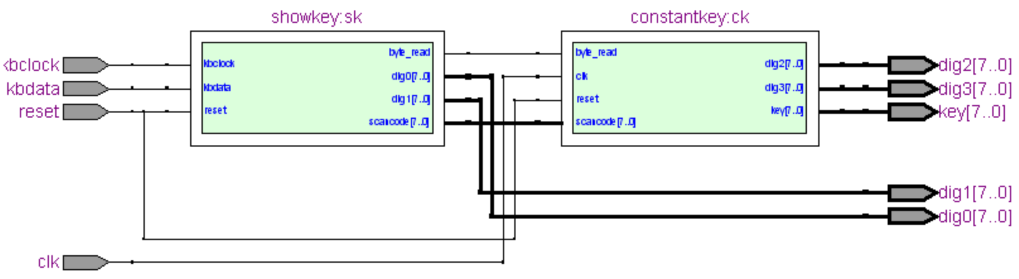


Figure 5.5 Readkey

First you have to read 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 the key is pressed a byte is generated: < keycode >
- When the key is pressed continuously, this byte repetitively 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). This synchronous logic of the system operates on the generated clock from the keyboard: KBCLOCK. In addition, this subsystem and also the other subsystems have a reset. The relevant part of the diagram for the reset is given in Figure 5.6. From this it can be seen that the reset is active when a '0' is applied on the input (the switch is pushed).

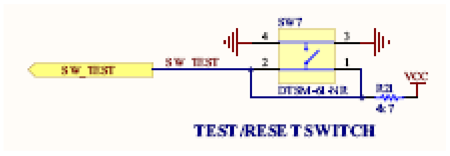


Figure 5.6 Reset circuit on the LiveDesign Evaluation Kit.

If the key 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, 8 bit keycode (the keycode 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 generated 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 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 most significant bits of the last generated code is shown and DIG1 shows the hex value of the four lowest significant bits.

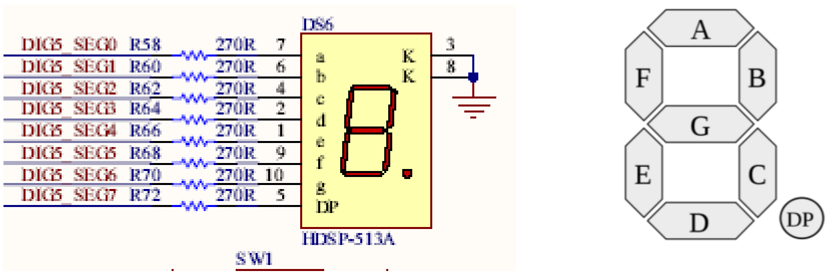


Figure 5.7 Part of schematic of the LiveDesign Evaluation Kit and on the right the usual coding of the LEDs.

5.1.3.2 Constantkey

The subsystem CONSTANTKEY operates on the clock of the LivedDesign Evaluation Kit (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 constantly 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 during the previous clock period (of the 50 MHZ clock) the input BYTE_READ was ‘0’ and the current input value is ‘1’. The reason for this is explained later.
The outputs DIG2 and DIG3 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 Tone generation

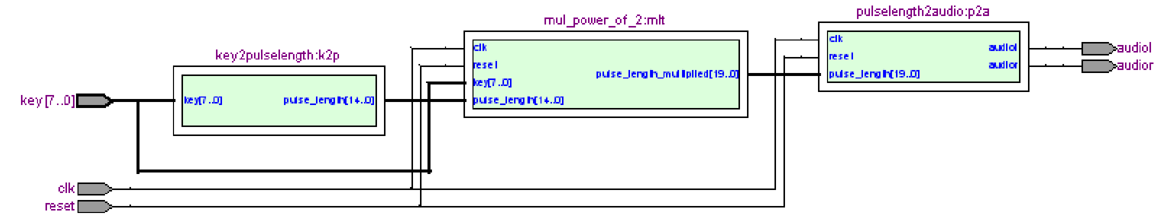


Figure 5.8 Tone generation

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

- KEY2PULSELENGTH
  When a relevant key of our organ is pressed a square wave is generated on both audio outputs with the same clock period as the tone corresponding to the pressed key. The output PULSE_LENGTH has a constant value that indicates the number of clock periods that the audio output signals must be high.
- 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.

- PULSELENGTH2AUDIO
  This subsystem generates a square wave at the two audio output pins with a period time of 2 × PULSE_LENGTH_MULTIPLIED × clock period of CLK. The duty cycle of the square wave is 50%.

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 in the realized hardware. 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 neatly aligned (indented).

5.2.1 Synchronous logic

For a synchronous system only use the following framework is used:

```
process(reset,clk)
 <here your local declarations>
begin
  <no VHDL code here!!!!>
  if reset='0' then    -- for the LiveDesign Evaluation Kit 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! In a synchronous system 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 this 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 7.1.1).
- A variable must be assigned 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 effect 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 always updated immediately. The assignment operator is :=
- Signals are never 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 determine your progress on the basis of the journal.

5.4 Intermediate assignment

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

5.4.1 Partial assignment 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(7 DOWNTO 0);
    -- shows the key pressed on display in Hex dig0 (upper 4 bits) dig1 (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 Blackboard. So do not copy VHDL fragments from this document because the formatting is lost.

5.4.1.1 Testbench for Showkey

The subsystem SHOWKEY is realized with the LiveDesign Evaluation Kit. However, before the system is realized it must be simulated first to detect design errors. To make this possible a (too) simple test setup is provided to you; see Figure 5.9 (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 synthetisable!
- 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(7 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
  BEGIN
    CASE n IS  --       hgfedcba
         WHEN "0000" => RETURN "00111111";
         WHEN "0001" => RETURN "00000110";
         WHEN "0010" => RETURN "01011011";
         WHEN "0011" => RETURN "01001111";
         WHEN "0100" => RETURN "01100110";
         WHEN "0101" => RETURN "01101101";
         WHEN "0110" => RETURN "01111101";
         WHEN "0111" => RETURN "00000111";
         WHEN "1000" => RETURN "01111111";
         WHEN "1001" => RETURN "01101111";
         WHEN "1010" => RETURN "01110111";
         WHEN "1011" => RETURN "01111100";
         WHEN "1100" => RETURN "00111001";
         WHEN "1101" => RETURN "01011110";
         WHEN "1110" => RETURN "01111001";
         WHEN OTHERS => RETURN "01110001";
    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
```

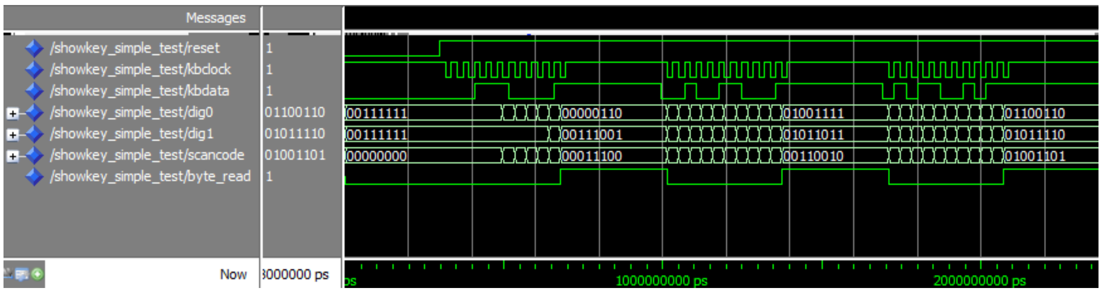
```
    reset <='0';
    WAIT FOR 300 us;
    reset <= '1';

    -- key A: 1C (hex) => 0001 1100 of hexadecimaal X"1C"
    send_byte(X"1C", kbclock, kbdata);
    ASSERT (dig0=hex2display(X"1") AND dig1=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 (dig0=hex2display(X"3") AND dig1=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 (dig0=hex2display(X"4") AND dig1=hex2display(X"D")) REPORT "expected: 4D (hex)"
           SEVERITY error;
    WAIT FOR 300 us;
    WAIT;
  END PROCESS;

END test;
```
Figure 5.8 Test environment for SHOWKEY


Messages
/showkey_simple_test/reset 1
/showkey_simple_test/kbclock 1
/showkey_simple_test/kbdata 1
/showkey_simple_test/dig0 01100110
/showkey_simple_test/dig1 01011110
/showkey_simple_test/scancode 01001101
/showkey_simple_test/byte_read 1
00111111
00000110
01001111
01100110
00111111
00111001
01011011
01011110
00000000
00011100
00110010
01001101
Now 3000000 ps
ps
1000000000 ps
2000000000 ps
Figure 5.10 Result of a simulation run of the design of the lecturer

Assignment 2

Test your design with the supplied test environment (figure 5.9)
- Compile your SHOWKEY design. Make sure that you do not change the entity description.
- Next 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

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.

5.4.2 Partial assignment: realization of Showkey

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

5.4.2.1 Synthesis with met Quartus II.

Synthesize your design with Quartus II.

The technology used:
- device family: Altera➔Cyclone
- device: EP1C12F324C8

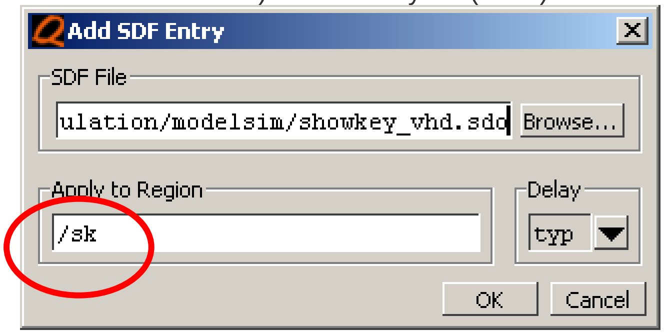
Since a post simulation is to be performed you have to instruct Quartus II to generate post simulation files (step 4 in the New Project wizard; assuming that the wizard is not changed in a newer version of Quartus II).

Note: Do not forget to include constraint file (SHOWKEY.QSF) in the project directory. The constraint file is changed by Quartus II. Hence, keep a copy of your original constraint file elsewhere.

5.4.2.2 Performing a post simulation

As a result of synthesis a directory "..\simulation\modelsim" is created that contains a vhdl file (*.vho) and the associated delay file (*.sdo).
These files can be used with the same test environment (file showkey_simple_test.vhd) to perform a post simulation (see VHDL tutorial).

Note: Since SHOWKEY is not the top level you must add region SK (the label of the component instantiation) for the delay file (*.sdo).



SDF File: ulation/modelsim/showkey_vhd.sdo
Apply to Region: /sk
Delay: typ

Assignment 3	Perform a post simulation using the test environment (figure 5.8)

Note 1: The generated post simulation VHDL file is a model of the hardware that is realized. In the real hardware there is not an "integer" only group of wires that have an integer representation. Therefore some synthesis tools the type in the entity declaration:
`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.

Note 2: Post simulation has to be performed when an ASIC is made ("sign-Off simulation"). With post simulation, among others, the timing is checked (a spike on the clock line, data on the data input is not stable at the active edge of the clock of a flip-flop, etc.). It is a (financial) disaster if this is detected after the ASIC is manufactured.
When an FPGA is used, the post simulation step is sometimes skipped, because you can quickly program the FPGA and then determine if it works. But does it really operate correctly? You tested the FPGA with an ambient temperature of 21 ^{0}C, but at the customer's location it is 18 ^{0}C. Does it still work at that temperature?

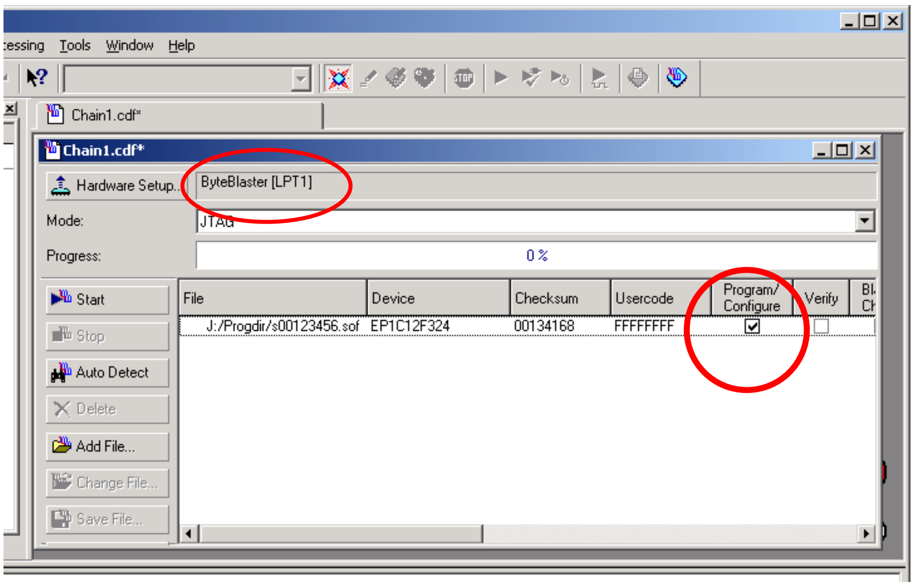
5.4.2.3 Transport of the programming file to the PC's with the Livedesign Evaluation Kit.

Copy the generated programming file showkey.sof to an USB memory stick.

5.4.2.4 Programming the FPGA.

The setup with programming PC and LiveDesign Evaluation Kit is ready for use, i.e.
- Quartus II is already running and the programmer software window is available
- The LiveDesign Kit is connected to the PC and to the PS/2 keyboard.

Note: If the programmer software is not yet running:
- start Quartus II, if necessary
- Tools→Programmer



Hardware Setup... ByteBlaster [LPT1]

Mode: JTAG

Progress: 0 %

File	Device	Checksum	Usercode	Program/ Configure	Verify
J:/Progdir/s00123456.sof	EP1C12F324	00134168	FFFFFFFF	✓	

Figure 5.11:Settings for programming the LiveDesign Evaluation Kit

Assignment 4	Program the FPGA - Ensure that the hardware setup is "byte blaster (LPT1)" or "byteblasterMV (LPT1)": - If "no hardware" is displayed then click on "Hardware Setup" - In the new window double-click ByteblasterMV - Close this window. Now the correct hardware should be shown - Add file → select the programming your programming file on your USB device. - Check that "Program / Configure" is enabled (see Figure 5.11). - Click on “start” and the FPGA is programmed.

Note: When programming the FPGA, you might get an JTAG error message. This can be solved by the disconnecting the power of the LiveDesign Evaluation Kit and connecting it again.

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

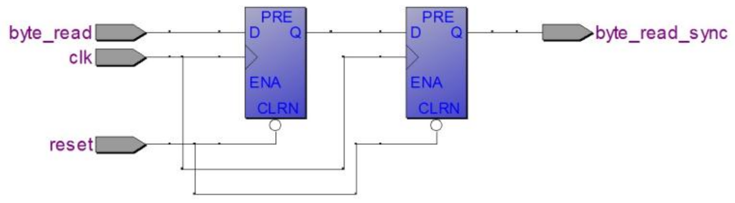
5.4.3 Partial assignment Constantkey

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(7 DOWNTO 0);
                  -- show key pressed on display dig2 and dig3 (resp high & low).
    key         : OUT std_logic_vector(7 DOWNTO 0)
    );
```

```
END constantkey;
```

System READKEY 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. However since the clocks of both systems are independent it is possible that BYTE_READ changes at the active edge of the 50 MHz clock. BYTE_READ is a so called asynchronous input of system CONSTANTKEY. The details are discussed in the lecture, but in the system CONSTANTKEY you must synchronize input BYTE_READ with two flipflops in series. The line BYTE_READ_SYNC is to be used as control line in CONSTANTKEY instead of BYTE_READ!



Assignment 6	A signal NEW_SCANCODE_DETECTED is used in CONSTANTKEY (see later). This signal is exactly one clock period ‘1’ when the current value of BYTE_READ_SYNC is ‘1’ and the previous value is ‘0’. - Give a VHDL description with inputs BYTE_READ, CLK and RESET and output NEW_SCANCODE_DETECTED. - simulate the design (you may apply the input patterns via the transcript windows of ModelSim). - synthesize the design (technology is not important) and study the RTL view.

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). Remember that a new scancode is detected when NEW_SCANCODE_DETECTED is ‘1’.

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 used 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 (the lecturer needed only three states).
In the state where the key pressed is detected the scancode must be stored. Hence, only an 8 bit register is needed.

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. no keys are pressed simultaneously the keycode after $F0_{16}$ is the same as the key that was pressed the last time. If you press 2 or more keys simultaneously, the keycode after F0_{16} is not trivial. Of course if you accidently press two or more buttons it would be nice if your organ can handle that. That is easy: accept any scancode after F0_{16}.

Assignment 8	Show the state machine 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	Verify CONSTANTKEY with the provided test environment in file constant_key_simple_test.vhd

Assignment 11	Synthesize CONSTANTKEY with Quartus II. The design is not yet realized on the LiveDesign Evaluation Kit, but synthesis problems can be identified.

5.4.4 Partial assignment 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);
                -- I/O check via 7-segment displays
    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 constant key
   );
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.

Assignment 14	Synthesize READKEY with Quartus II and realize it on the LiveDesign Evaluation Kit (don’t forget the constraint file ☺).

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 Partial assignment 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 pulse duration is determined. For each relevant key (see figure 5.2) a constant value has to be produced at the output of the system. Now is the time to bring the theory of the tone scale into practice!
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 Add your script file and relevant part of the waveform in your journal.

Assignment 18	Synthesize KEY2PULSELENGTH with Quartus II. The design is not yet realized on the LiveDesign Evaluation Kit, but synthesis problems can be identified.

5.4.6 Partial assignment 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;
```

Assignment 19	Give a VHDL description of architecture MUL_POWER_OF_2. Use only the VHDL synthesis subset. Hint: read both notes beneath.

Note 1: When a key is pressed a constant value of the corresponding keycode is at input KEY. Be sure that you handle this correctly.

Assignment 20	Verify the composition of KEY2PULSELENGTH and MUL_POWER_OF_2 with a VHDL test environment. The VHDL test environment is not provided.

Assignment 21	Synthesize MUL_POWER_OF_2 with Quartus II. The design is not yet realized on the LiveDesign Evaluation Kit, but synthesis problems can be identified.

5.4.7 Partial assignment pulselength2audio

In section 5.1.4 the behavior of PULSELENGTH2AUDIO is described.
The entity description is:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
ENTITY pulselength2audio IS
  GENERIC (max_length : integer := 20000);
  PORT  (clk    : IN std_logic;
         reset  : IN std_logic;
         pulse_length : IN INTEGER RANGE 0 TO max_length;
         audiol : OUT std_logic;
         audior : OUT std_logic
  );
END pulselength2audio;
```

Input PULSE_LENGTH is an integer value that represents the number of clock cycles that the audio outputs must be ‘0’. The generated square wave should have a duty cycle of 50%.
When the value on the PULSE_LENGTH is changed the new sequence high and low sequence will begin after the previous cycle is completely finished (see figure 5.12).
When input PULSE_LENGTH has integer value 0 both audio outputs are constant low (‘0).

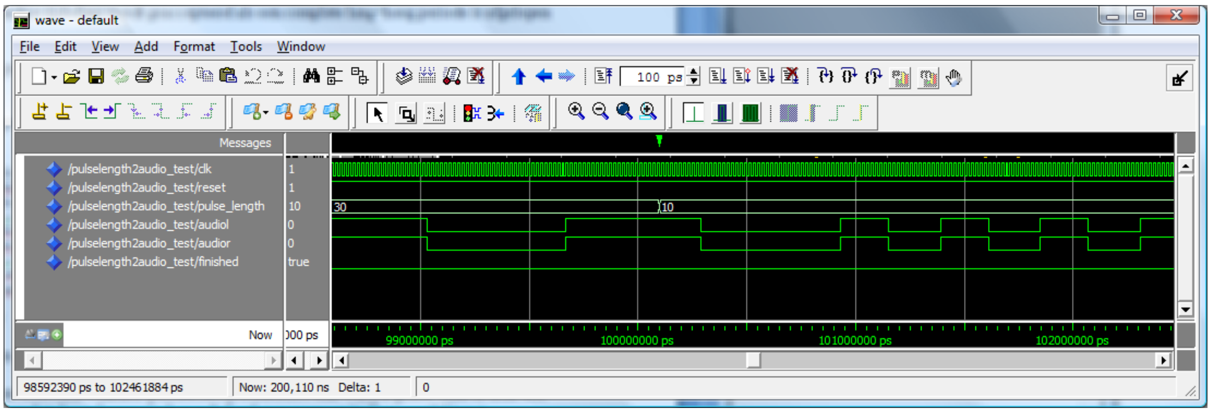

wave - default
File Edit View Add Format Tools Window
Messages
/pulselength2audio_test/clk 1
/pulselength2audio_test/reset 1
/pulselength2audio_test/pulse_length 10
/pulselength2audio_test/audiol 0
/pulselength2audio_test/audior 0
/pulselength2audio_test/finished true
30
10
Now
99000000 ps
100000000 ps
101000000 ps
102000000 ps
98592390 ps to 102461884 ps
Now: 200,110 ns Delta: 1

Figure 5.12: simulatierun of `pulselength2audio`

Assignment 22	Give a VHDL description of architecture PULSELENGTH2AUDIO. Use only the VHDL synthesis subset.

Assignment 23	Verify PULSELENGTH2AUDIO with a VHDL test environment. The VHDL test environment is not provided. The test environment an relevant part(s) of the waveform must be in your journal.

Assignment 24	Synthesize PULSELENGTH2AUDIO with Quartus II. The design is <u>not yet</u> realized on the LiveDesign Evaluation Kit, but synthesis problems can be identified.

5.4.8 Partial assignment tone generation

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

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY tone_generation IS
  PORT (clk    : IN std_logic;
        reset  : IN std_logic;
        key    : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        audiol : OUT std_logic;
        audior : OUT std_logic);
END ENTITY tone_generation;
```

Assignment 25	Give a VHDL description of the architecture TONE_GENERATION. Use only the VHDL synthesis subset.

Assignment 26	Verify TONE_GENERATION with a VHDL test environment. The VHDL test environment is not provided. Keep the test simple since a simulation run takes a long time.

Assignment 27	Synthesize TONE_GENERATION with Quartus II. The design is not yet realized on the LiveDesign Evaluation Kit, but synthesis problems can be identified.

5.4.9 Partial assignment organ

The organ has two subsystem: READKEY and TONE_GENERATION. See section 5.1.2.
The entity description is:

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY organ 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
        audiol : OUT std_logic;
        audior : OUT std_logic;
               -- debug outputs
        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
);
END ENTITY organ;
```

Assignment 28	Give a VHDL description of architecture ORGAN. Use only the VHDL synthesis subset.

Assignment 29	Synthesize ORGAN with Quartus II and realize it on the LiveDesign Evaluation Kit (don’t forget the constraint file ☺).

Assignment 30	You must show the correct operation of your organ to the student assistant (your journal has to be up-to-date!).

6 *Final assignment: organ with recording capability*

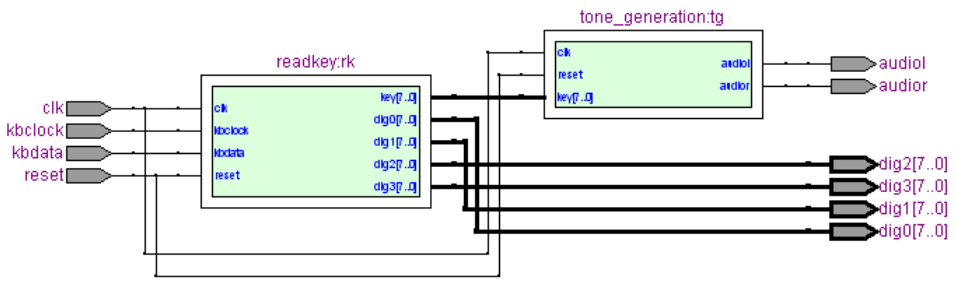


Figure 5.13: Division of organ with two subsystems

Until now all design decisions were taken for you! In this final assignment you have to extend the organ with a recording facility.

You must fulfill the following design constraints:

1. The subsystems READKEY and TONE_GENERATION may not be changed.
2. The recording / playback unit has three buttons (use the buttons on the live design board):
    a. Stop; terminate the recording or playback.
    b. Rec; record what is currently being played (which should also be heard during recording).
    c. Play; Playback.
3. The SRAM on the LiveDesign Evaluation Kit stores what is recorded.
4. The organist, who will use your design, can play, and record, many hours continuously.

Additional constraints/comment:

- Your specification of the record/play system should be in the journal including a motivation for the sample frequency of the output KEY of system READKEY.
- During recording you can increase/decrease an octave. This change is stored in the tone generation unit. If you playback what is recorded the tone generation unit can have another octave then it has at the start of the recording. You don't need to solve this problem.
- Verify the record/play system including a VHDL simulation model of the SRAM with a VHDL test environment.
    - o The datasheet of the SRAM is on BlackBoard including simplified VHDL simulation model.
    - o A simulation can be very long because the clock frequency is 50 MHz and a key press takes about 1 sec. (Perhaps the correct operation can also be shown if the sample frequency is scaled.)
- Realize your design with the live design board. The entity ORGAN_WITH_RECORDER.VHD and constraint file organ_with_recorder.qsf are on Blackboard.
    - o If the organ does not work properly then you are in trouble. Since the record/play system seems to work properly perhaps a little mistake is made in the architectural description of the organ. If this is not the case you can make a simple VHDL test environment for the whole system (organ and simulation model of SRAM). Probably the test environment for READKEY can inspire you (readkey_simple_test.vhd).

Assignment 31	Discuss your design with the student assistant BEFORE writing VHDL!

Assignment 32	Show your organ with recording facility to the student assistant (your journal has to be up-to-date!).

7 *Known tool issues*

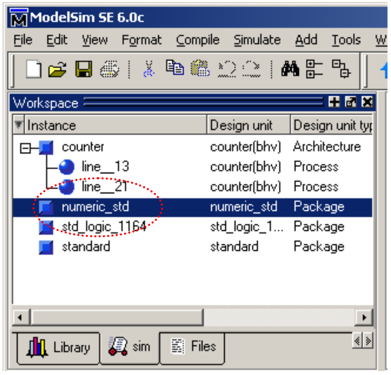
7.1 ModelSim

7.1.1 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".

7.1.2 No signals are 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

7.2 Quartus II

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 II. 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 an mismatch is reported). Keep a copy of the constraint file elsewhere and copy it to the project directory before a new synthesis run is performed.

7.3 Programmer

7.3.1 Byteblaster

Perhaps the first time you should instruct the programmer using the "byteblasterMV (LPT1)" with JTAG interface. The default in the programmer (Tools => Programmer "No Hardware"), click on "Hardware Setup" and select ByteblasterMV.
When programming the FPGA, you might get an JTAG error message. This can be solved by the disconnecting and connecting the power again to the LiveDesign Evaluation Kit.

8 *Planning of the lab sessions (2 possible routes)*

Assignment		Intermediate assignment recommended deadline in Lab Session	Intermediate assignment + Final assignment recommended deadline in Lab Session
	Before first lab session: **- software is installed on your laptop** **- the VHDL tutorial is completed**		
0	Blinking light + program FPGA	1	1
1	Showkey (VHDL)		
2	Test		
3	Post simulation showkey		
4	Program FPGA		
5	Show result to student assistant	2	1
6	New scancode detected (VHDL)		
7	State machine (graphical/no VHDL)		
8	Show result to student assistant		
9	Constant key (VHDL)		
10	Simulation		
11	Synthesis	3	2
12	Readkey (VHDL)		
13	Test		
14	Synthesis and program FPGA		
15	Show result to student assistant		3
16	Key2Pulselength (VHDL)		
17	Test		
18	Synthesis		
19	Multiply with power 2 (VHDL)		
20	Test		
21	Synthesis	4	
22	Pulselength2 audio (VHDL)		
23	Test		
24	Synthesis		4
25	Tone generation (VHDL)		
26	Test		
27	Synthesis	5	
28	Organ (VHDL)		
29	Synthesis organ + program FPGA		
30	Show result to student assistant	6 **Hard deadline** is 7	5 **Hard deadline** for a successful completion of the final assignment
31	Final assignment +design	not applicable	6
32	Final assignment + program FPGA	not applicable	8