

## Arithmetic Logic Unit (ALU)

### FIR Filter

---

v.1.0.0

#### **Abstract**

The work for this lab is divided into two separate and independent parts. Each part has its own preparation, coding and design flow. The first part is a simple arithmetic logic unit (ALU); but to be able to provide the inputs and perform the required operations, a finite state machine (FSM) is also needed. The whole design should finally be synthesized and downloaded into the FPGA. 7-segment displays on the FPGA board should be utilized to show the results of the ALU. Therefore, an architecture in VHDL to drive the 7-segment displays is required. The second part of this lab is to design a finite impulse response (FIR) filter in VHDL. Furthermore, a pipelined version of the same filter should also be designed. There is no need to physically implement them into the FPGA, but a synthesis report is required to compare the limits of the both designs. In order to pass the lab, preparations and fulfilling both parts (ALU and FIR) are mandatory.

## **Lab Preparation**

- Go through the entire manual and try to understand the required functionality and given tasks. Make sure that you have understood what is expected from the lab before you start coding. If the functionality or any task sounds unclear consult the lab assistants during the lab session.
- Prepare yourself as much as possible before the lab session.
- Do tasks 1 and 2 of the first part (ALU). Also, task 1 of the second part (FIR) should be done before the lab session.
- Go through the files that have already been provided in the lab directory. For each part of this lab, try to figure out how much coding you need to develop to fulfill the tasks.

## Part I

# Arithmetic Logic Unit - ALU

## Introduction

The purpose of the first part of this lab is to design a simple arithmetic logic unit (ALU). You do this using VHDL hardware description language. Afterwards, you need to download the synthesized code to the FPGA board. This is done in order to test and verify the functionality of the design. The design flow is based on VHDL Modelsim simulator, XILINX ISE synthesis tools and XILINX Spartan-3 FPGA board for hardware implementation.

The ALU responds to the user's input commands and performs one of the selected operation; addition and subtraction. Furthermore, the result needs to be displayed on the 7-segment display module available on the board. In more detail, every input of the ALU is an 8-bit digit which can be set by an arbitrary selection of the switches on the FPGA board. The position of every switch represents a binary 0 or 1; therefore, the decimal range of each of the ALU inputs is an un-signed integer between 0 and 255. The required functionality of the ALU unit is as follows.

## Requirements

- After reset, the value shown on the 7-segment module is simply the decimal representation of the 8-bit binary value set by the switches on the board.
- When the user has set the first operand (A), a push button on the board (BTN0) can be used as "enter" to save this value in a register.
- While the first ALU operand has been saved, the second operand (B) can be set by re-arranging the positions of the switches and should be saved in a separate register when the "enter" push button is pressed for the second time.
- As soon as the ALU has both its inputs, it can perform the two required operations. The first operation could be "A+B" as soon as the input B has decided, then by pressing the button for the third time "A-B" can be performed.
- The ALU output should toggle between "A+B" and "A-B" when the button is pressed several times.
- The output of the ALU must be continuously shown on the 7-segment display.

Looking at the required functionality it is obvious that a possible implementation could involve a controller, i.e., a state machine, a datapath containing an ALU, a register block to keep the ALU input values, a binary to binary-coded-decimal converter (Bi/BCD) and a driver for the 7-segment display. An example of a possible implementation is shown in Figure 1.

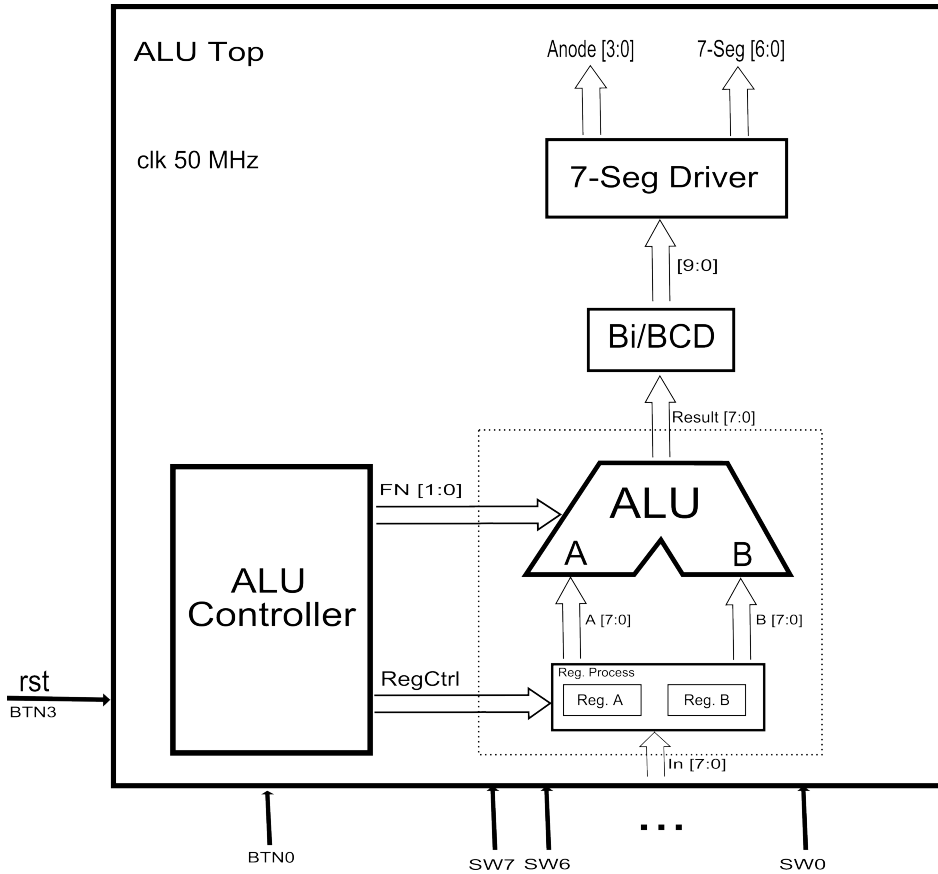


Figure 1: ALU top structure

**Task 1** Propose a FSM that based on the input signal (Enter), provides the control signals to the ALU in order to achieve the required functionality shown in Figure 3. On a piece of paper draw a detailed finite state machine (FSM) for your solution. Determine what control signals must be applied to the ALU in each state. The FSM can be either of type Moore or Mealy.

**Task 2** Specify how you can use the controller you have designed previously in Task 1 to provide the proper control signals for the register update process according to the required functionality (Figure 3). On your state diagram clarify what control signals are needed based on the state of the controller.

## 1 ALU Block

The ALU block of this lab is a simple combinatory logic circuit that performs the following functions according to the command from the controller.

The ALU performs addition and subtraction on a pair of 8-bit binary inputs. Therefore, each input has a range between 0 and 255 in decimal representation. The result of the ALU is an 8-bit value. It also includes the signals "sign" and "overflow" that indicate whether the result is negative or an overflow has occurred.

**Task 3** Write VHDL code for the ALU block to perform the required functions as given in 1. For this purpose, you should use the file "alu.vhd" in the lab directory and

FN[1:0]	operation
00	ALU output is A (Pass A)
01	ALU output is B (Pass B)
10	ALU output is A + B
11	ALU output is A - B

Table 1: **ALU block** - functions and the corresponding input control signals

complete it with your code.

When you have written the functional VHDL code for ALU block you need to make sure that your ALU works as expected. In order to verify the correct functionality of your ALU entity a testbench has been already provided in the lab directory.

**Task 4** Make a new project in Modelsim and add “alu.vhd” and “tb\_alu.vhd” into your project. Compile the files and run a simulation. To add the waveforms and perform the simulation for a period of time, a set of scripts are provided in the file “alu\_sim.do” in the lab directory. When the wave window is open in Modelsim, write the following command in the command line. That will add the required signals and will run the simulation for 300 ns. You should then see the waveforms as shown in Figure 2.

```
do alu_sim.do
```

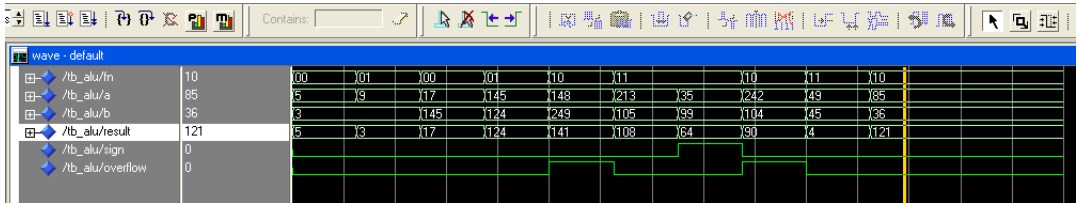


Figure 2: Waveforms after simulation of ALU in Modelsim

## 2 ALU Controller

According to the needed functionality, a controller, i.e. a finite state machine is required. It manages the behavior of the ALU while the enter button is being pressed by the user. The ALU behavior, according to the enter signal sequence is shown in Figure 3.

When you finally download your design into the FPGA, the input of the controller (the “enter” signal) comes from the BTN0 push button available on the board. The outputs are the control signals to the ALU and register block as shown in Figure 1.

**Task 5** Design the register transfer level implementation of the controller in VHDL. Don’t integrate it into your design unless you know that it works as expected.

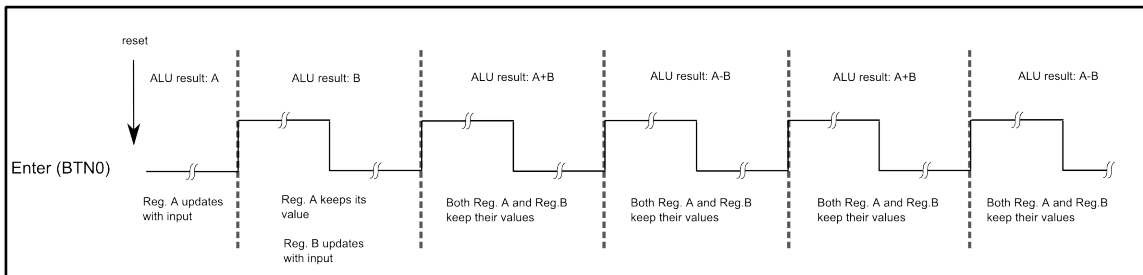


Figure 3: functionality of ALU based on the input commands

### 3 Register Process

Finally, when the whole design is downloaded into the FPGA, the input for operands A and B are provided with the 8 switches on the board (SW0 - SW7). Thus, only one register can be updated at any time by the binary value set with the switches. You can use the controller you have designed and tested previously to control when each register should be updated, depending on the state of the controller. An RTL structure describing the needed sequential and combinatorial processes is shown in Figure 4.

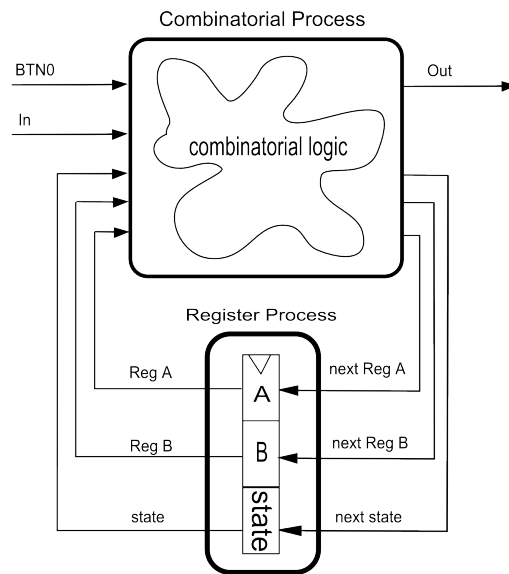


Figure 4: The RTL structure describing the processes needed

**Task 6** Design an entity to provide suitable inputs for the register process in this lab based on the expected functionality described in Figure 3. For your design, consider the following input ports; RegA , RegB, In, RegCtrl and the following output ports; next\_RegA, next\_RegB.

## 4 Binary to BCD Converter

BCD or binary-coded-decimal is a method of using four binary digits to represent the decimal digits 0 to 9. For instance, "249" with binary representation "11111001" can be represented by "010 0100 1001" in BCD format. Converting a binary value to BCD is very common, in order to represent each digit separately on a 7-segment display (also see the attachment).

**Task 7** Write a binary to BCD entity VHDL code and a proper testbench to verify its functionality. Compile the converter and simulate it using Modelsim.

## 5 7-Segment Driver

The FPGA board in the lab provides four common-anode type 7-segment displays (AN3-An0). The segments of each display are called a, b, up to g as shown in Figure 5. In order to reduce the number of connections on the board the cathode pins of the different displays are connected together to form seven common terminals between displays for every segment a, b up to g. To illuminate a segment of a particular display, e.g. segment e of the third display, one needs to apply a "0" to Anode3 (in order to select the third display) and a "0" (GND or Low) to cathode node e (to light segment e).

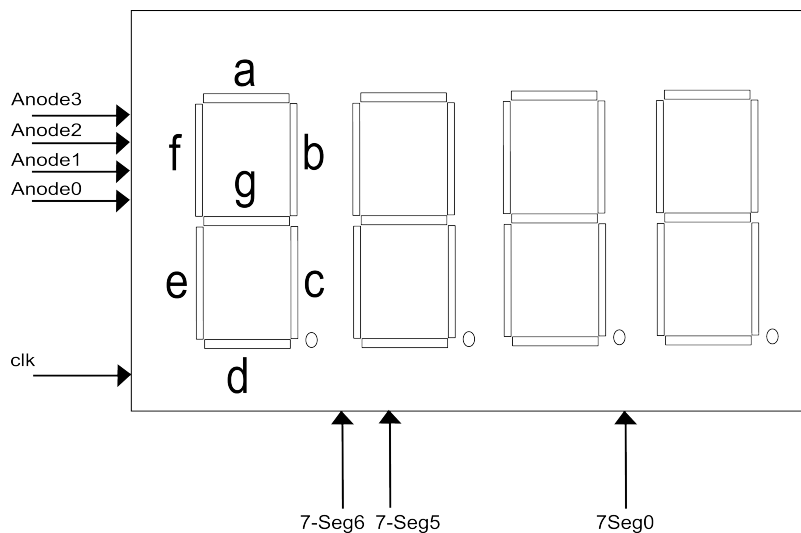


Figure 5: 7-segment display

Due to a shared bus between the segments and to reduce the power consumption, 7-segment displays are not normally enabled continuously. They are rather sequentially enabled and disabled with a frequency such that the human eye will have the illusion that the display is continuous. Thus, if two or more digits are to be displayed, time multiplexing with a reasonable cycling interval between the digits of the 7-segment display must be employed.

**Task 8** This is the last major block needed to complete your system. Develop the VHDL code for the four digit 7-segment display. This involves: (1) Considering time multiplexed selection of 7-segment modules. (2) Illuminating the required segments

of each module based on the digit to be displayed. Use the left most module to show”-” when the result is negative or ”F” when an overflow has occurred.

## 6 Synthesizing the whole ALU structure

Now that you have written and tested different parts of the whole ALU structure, you are ready to implement a prototype circuit. This is often done using an FPGA technology. To accomplish this, we use Xilinx Spartan-3 FPGA boards available in the lab. To finalize your design before the synthesizing and download to FPGA, you need to fulfill two remaining tasks: (1) You need to integrate all the codes you have developed so far into a project. (2) Mapping the ports of the top structure of your design to the physical ports of the FPGA.

**Task 9** Initiate a project in Xilinx ISE environment and add the required VHDL files you have developed so far into that project. To connect the separate components of your design, add a new VHDL source file into the project and develop the VHDL code for the ALU top structure. Use the interface model shown in Figure 6 for the top structure.

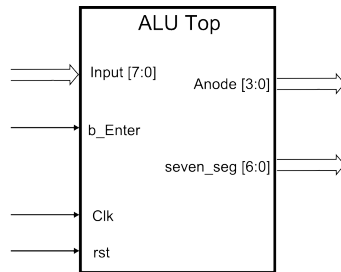


Figure 6: ALU\_top block

**Task 10** In the lab directory, the configuration definition for the physical implementation has been provided in the file “ALU\_top.ucf”. Add the file to your project in order to map the ALU\_top ports to the physical ports on the board.

**Task 11** Synthesize and download the whole design in FPGA using Xilinx ISE tool. Verify that your design works as expected.

## Part II

# Finite Impulse Response Filter

## Introduction

In this part, you will design a 16-tap FIR filter in VHDL and will simulate it in Modelsim. The filter should be implemented once in direct form and then pipelining should be applied. The structure of a direct form FIR filter is given in Figure 7.

## 7 The FIR Filter

The FIR filter is a digital filter with a finite duration impulse response. These types of filters have applications in digital signal processing and the communication systems, where a linear phase is required in the pass band. The FIR filter algorithm is actually a convolution of the input signal with a set of pre-defined coefficients. Mathematically, the FIR filtering process can be described as

$$y[k] = \sum_{m=0}^{n-1} h[m]x[k-m], \quad (1)$$

where  $n$  is the number of taps. The data flow diagram of a digital FIR filter consists of a series of multipliers, adders and delay elements to store the  $w$ -bit input words as they transverse the filter. A shift register, a multiplier and an adder are used for each tap in the filter. For any multiplier in a certain tap, one input is the delayed input words and the other input is programmed with the value of the associated coefficient.

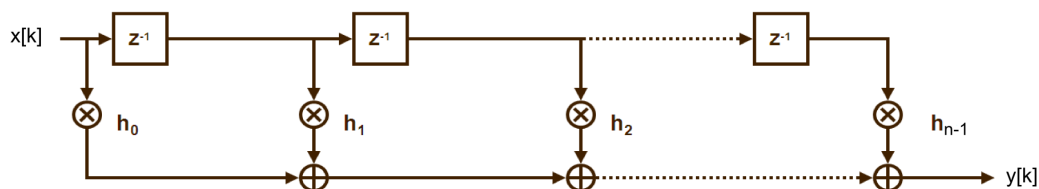


Figure 7: Structure of a  $n$ -tap FIR filter in direct form

## Assignments

**Task 1** Inputs of the 16-tap filter are represented by 8 bits, also 7 bits are assigned for each of the filter coefficients. What are the required word lengths for the adders and multipliers in every tap to achieve a full precision processed output? Consider the direct form structure for the filter as in Figure 7?

**Task 2** Implement a direct form FIR filter in VHDL using the coefficients given bellow

$$H = [-1, 0, 1, -3, -9, -2, 30, 63, 63, 30, -2, -9, -3, 1, 0, -1]$$

Assign 7 bits for every filter coefficients and assume an 8-bit input sequence. The impulse and step responses of such a filter are shown in Figure 8.

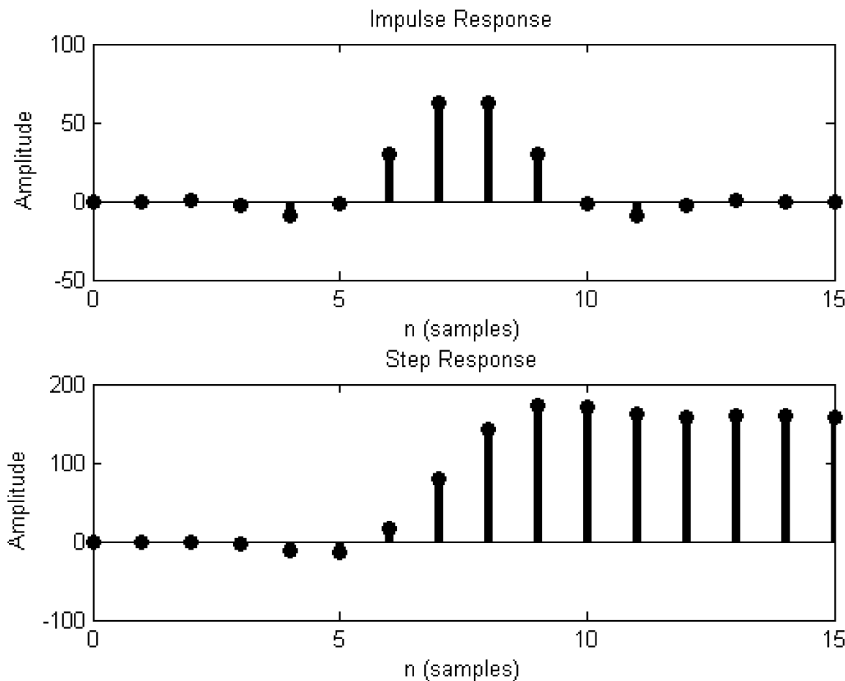


Figure 8: FIR filter impulse and step responses

The filter output must be in full precision, so you must consider the proper word length for the different taps of the filter. In order to verify the correct functionality of the filter, an input vector is provided in a text file that can be read by a testbench. Use the provided testbench "tb\_fir.vhd" and your design for "fir.vhd" to write the output sequence of the filter in a file. Choose the name "VHDL\_output.dat" for the output file.

**Task 3** Initiate Matlab and run the m-file provided in the lab directory to compare your results with the expected output.

**Task 4** Now implement the filter in a coarse grained pipelined structure in VHDL. Add a single pipeline stage between taps 7 and 8 of the filter as shown in Figure 9. Verify its correct functionality using Modelsim likewise as the previous task.

Now start Xilinx ISE and initiate a new project. Add the codes you have developed so far into the project. Also add "fir\_top.vhd" into the project. The top structure includes your fir filter as a component, also two flip-flops at the input and output of the filter.

**Task 5** Synthesize both structures of the filter separately using the Xilinx ISE. What are the maximum possible clock frequencies for each structure? What do you gain and what do you lose from applying a pipeline stage?

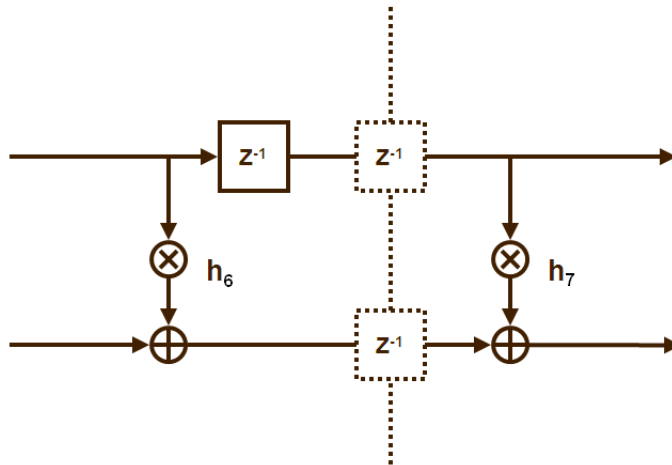


Figure 9: A pipeline stage between 7th and 8th taps of the filter

## Summary of Lab Assignments

### Part I - ALU

- Task 1** Propose a FSM that based on the input signal (Enter), provides the control signals to the ALU in order to achieve the required functionality shown in Figure 3. On a piece of paper draw a detailed finite state machine (FSM) for your solution. Determine what control signals must be applied to the ALU in each state. The FSM can be either of type Moore or Mealy.
- Task 2** Specify how you can use the controller you have designed previously in Task 1 to provide the proper control signals for the register update process (again as shown in Figure 3). On your state diagram clarify what control signals are needed based on the state of the controller.
- Task 3** Write VHDL code for the ALU block to perform the required functions as given in 1. For this purpose, you should use the file “alu.vhd” in the lab directory and complete it with your code.
- Task 4** Make a new project in Modelsim and add “alu.vhd” and “tb\_alu.vhd” into your project. Compile the files and run a simulation. To add the waveforms and perform the simulation for a period of time, a set of scripts are provided in the file “alu\_sim.do” in the lab directory. When the wave window is open in Modelsim, write the following command in the command line. That will add the required signals and will run the simulation for 300 ns. You should then see the waveforms as shown in Figure 2.
- Task 5** Design the register transfer level implementation of the controller in VHDL. Don't integrate it into your design unless you know that it works as expected.
- Task 6** Design an entity to provide suitable inputs for the register process in this lab based on the expected functionality described in Figure 3. For your design, consider the

following input ports; RegA , RegB, In, RegCtrl and the following output ports; next\_RegA, next\_RegB.

**Task 7** Write a binary to BCD entity VHDL code and a proper testbench to verify its functionality. Compile the converter and simulate it using Modelsim.

**Task 8** This is the last major block needed to complete your system. Develop the VHDL code for the four digit 7-segment display. This involves: (1) Considering time multiplexed selection of 7-segment modules. (2) Illuminating the required segments of each module based on the digit to be displayed. Use the left most module to show "-" when the result is negative or "F" when an overflow has occurred.

**Task 9** Initiate a project in Xilinx ISE environment and add the required VHDL files you have developed so far into that project. To connect the separate components of your design, add a new VHDL source file into the project and develop the VHDL code for the ALU top structure. Use the interface model shown in Figure 6 for the top structure.

**Task 10** In the lab directory, the configuration definition for the physical implementation has been provided in the file "ALU\_top.ucf". Add the file to your project in order to map the ALU\_top ports to the physical ports on the board.

**Task 11** Synthesize and download the whole design in FPGA using Xilinx ISE tool. Verify that your design works as expected.

## Part II - FIR

**Task 1** Inputs of the 16-tap filter are represented by 8 bits, also 7 bits are assigned for each of the filter coefficients. What are the required word lengths for the adders and multipliers in every tap to achieve a full precision processed output? Consider the direct form structure for the filter as in Figure 7?

**Task 2** Implement a direct form FIR filter in VHDL using the coefficients given below

$$H = [-1, 0, 1, -3, -9, -2, 30, 63, 63, 30, -2, -9, -3, 1, 0, -1]$$

Assign 7 bits for every filter coefficients and assume an 8-bit input sequence. The impulse and step responses of such a filter are shown in Figure 8.

**Task 3** Initiate Matlab and run the m-file provided in the lab directory to compare your results with the expected output.

**Task 4** Now implement the filter in a coarse grained pipelined structure in VHDL. Add a single pipeline stage between taps 7 and 8 of the filter as shown in Figure 9. Verify its correct functionality using Modelsim likewise as the previous task.

**Task 5** Synthesize both structures of the filter separately using the Xilinx ISE. What are the maximum possible clock frequencies for each structure? What do you gain and what do you lose from applying a pipeline stage?

## Attachment

Some useful information about implementing Bi/BCD in hardware could be found by following the links given below.

[http://www.engr.udayton.edu/faculty/jloomis/ece314/notes/devices/binary\\_to\\_BCD/bin\\_to\\_BCD.html](http://www.engr.udayton.edu/faculty/jloomis/ece314/notes/devices/binary_to_BCD/bin_to_BCD.html)

or file **“D7.3 CombCkt\_Verilog-2.ppt”** in:

<http://www.secs.oakland.edu/~polis/Lectures/>