Assignment 3: Introduction to FPGA Application Development

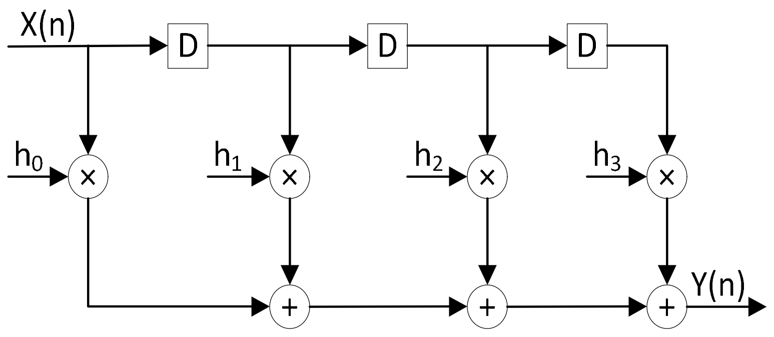
University of Massachusetts, Amherst
Department of Electrical and Computer Engineering
ECE 636 Reconfigurable Computing

Table of Contents

1. Functional Specifications

In this assignment, you will use the Verilog language to describe a 4-tap Finite Impulse-Response (FIR) filter. You will synthesize your design and test it using the Altera DE1 Development board. Additionally, you will learn how to write efficient self-checking test benches to simulate and verify the hardware design before creating an FPGA bitstream used to program the FPGA. The FIR filter is a widely-used digital filter used in digital signal processing applications. An FIR filter can operate in a host of different modes by carefully choosing the weight of each h_i coefficient. The following block diagram illustrates the *direct* form of the 4-tap FIR filter.

Figure 1: Direct form of the 4-tap FIR filter



$$y(n) = \sum_{i=0}^{N} h_i \times X_{n-i}$$

A filter *tap* includes a single multiplication and summing operation. Figure 1 shows a 4-tap filter used for signal processing. Note that the tap on the left does not require addition. The blocks labeled D are synchronous registers that separate the taps. This circuit is part of many audio and video operations. You will evaluate this circuit structure in FPGA hardware to complete this assignment.

To complete the assignment, you will write the Verilog code needed for a single tap of the filter. Then, you will instantiate the module four times using a separate Verilog file (top-level) to obtain the requested structure of the 4-tap FIR filter. A modular design offers several benefits for testing and implementation. By adopting this coding style, you will be able to simulate and debug your design efficiently. This design style also allows a designer to adjust the number of the synthesized FIR taps according to application needs. Your Verilog design should be parameterized to allow the user to change the number of taps by configuring a Verilog parameter in the top-level file. In the fir_filter_arm/doc/adder/ directory there is an example of a parameterized module. This module realizes a parameterized adder, whose size (number of full adders) can be configured by a Verilog parameter in a top-level design file. Reference [1] (included in your project folder) has a number of additional Verilog design examples, and recommended HDL coding styles and practices.

The following table below shows the I/O port list of the FIR module. You should implement the interface of your FIR module to match the given specifications.

Table 1: I/O Port List for FIR module

Port Name	Width	Direction	Function
clk	1	Input	Input clock to the FIR module.
rst	1	Input	Active high asynchronous reset for the FIR module.
load	1	Input	Active high signal used to indicate the parallel loading of the coefficients into the FIR module.
valid_in	1	Input	Asserted by the FIR wrapper for one clock cycle to indicate that the signal_in [] input port contains valid data.
valid_out	1	Output	Asserted by the FIR module for one clock cycle to indicate that the signal_out [] output port contains valid data. Note that the valid_out signal controls the write_enable port of a FIFO and hence you should assert to one this signal only for one clock cycle and only when the FIR module outputs a valid result.
coeff_in []	$N \times 16$	Input	Coefficient data bus used to input the coefficients into the FIR module. The width of the bus depends on the order of the filter.
signal_in []	16	Input	16-bit input data bus used to input the values of the incoming signal into the FIR module. For simplicity, assume integer arithmetic with signed 2's complement values.
signal_out []	16	Output	16-bit output data bus used to output the values of the filtered signal from the FIR module. For simplicity, ignore any kind of arithmetic overflow. This means that every bus in the internal structure of the FIR module should have 16-bit width.

2. Debugging the Design

Testing and debugging a complex circuit can be a difficult task. A systematic approach can ease this task and speed up the design cycle. To complete this assignment, you will need to write Verilog files (test benches) to simulate and verify the correctness of your FIR design. A test bench supplies stimulus signals to the tested design and evaluates the resulting outputs in an automated manner. The automated evaluation of the results is an important aspect of the testing process since even a simple circuit can generate a large number of outcomes. Note that testing is a repetitive process in the design flow and whenever the designer modifies the circuit a new testing cycle begins.

For this assignment, you have been provided with an FIR test bench (*fir_filter_arm/bench/fir_tb.v*). We have already implemented the functionality of the FIR test bench, but you still need to instantiate the top-level of the FIR module and make the appropriate port connections. Along with *fir_tb.v*, you have been provided the three text files that the FIR test bench uses to obtain the coefficients (*fir_filter_arm/tests/coeff.txt*) of the filter, the values of the input signals (*fir_filter_arm/tests/signal_in.txt*) and the expected values of the filtered signals (*fir_filter_arm/tests/expected_output.txt*).

The FIR test bench is responsible for the following functions: (1) it uses the function $readmemh() to read the three text files noted above and stores the data in three vectors; (2) it initializes and feeds the top-level of the FIR module, using a while-loop statement, with the coefficients of the filter and the values of the input signals; (3) at the same time it compares each output of the filter with the expected value and prints out the result of the comparison along with an index to the computer console display window using the function $display(); (4) after computation finishes, the test bench prints out a PASS/FAIL message to the console to inform the user about the result of the simulation. At this point, the console will contain the information needed to debug the design. You can use the simulation results in conjunction with the waveform viewer to view all the transitions of the simulated signals. Keep in mind that the FIR test bench is not synthesized or implemented in the FPGA at this point.

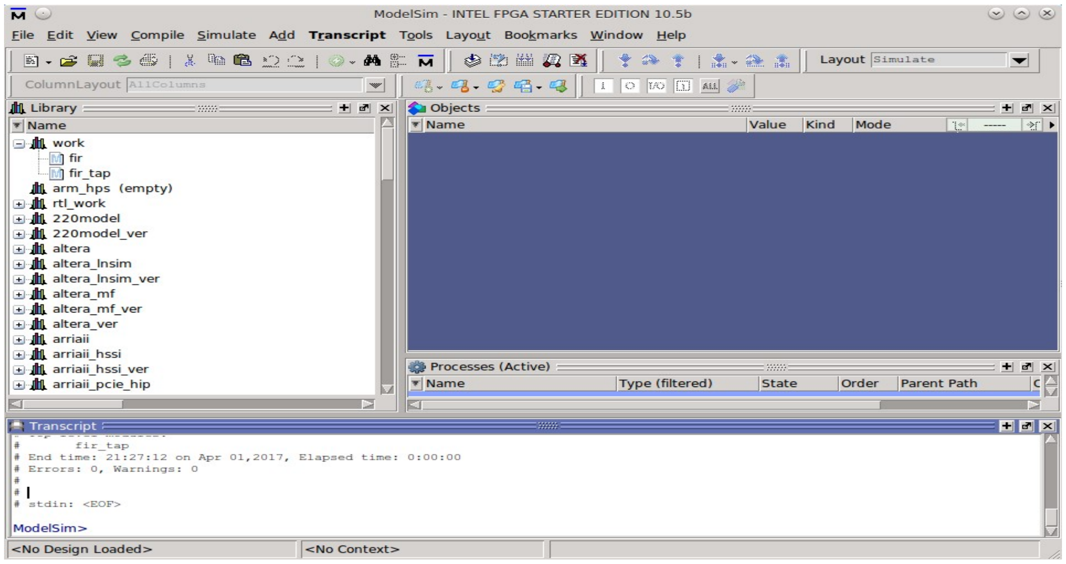
You have been provided with a zip file (*fir_filter_arm.zip*) which includes all the supplementary files needed to simulate your design. Extract the lab files into the *fir_filter_arm* directory, open **Quartus Prime Lite Edition 16.1** and follow the directions below to simulate the design:

0. On the home screen, select “Open Project” from the File menu and locate the Quartus Prime project file *fir_filter_arm.qpf*, you can find this file in the following path: /fir_filter_arm/syn/fir_filter_arm_5csema5/.
1. The Quartus Prime project contains several implementation files but in this step, you need only to simulate and test the FIR module. First, add to the project the Verilog files you have written to describe the FIR filter. The module name for the top level of your design should be *fir*. In Quartus Prime, click on the “Project” menu and select the option “Add/Remove Files in Project”. Click on the “…” button and navigate to the directory where you keep your Verilog files. Select the files, click on the “Open” button and then select “Add” – “Apply” – “OK”. You can check that you have successfully added your files to the project with the File menu in the Project Navigator.

2. **Ensure that the top-level file of your design is the top-level entity of the Quartus project.** To do so, select “Files” from the Project Navigator and then right click on the top-level file of your design to select the “Set as Top-Level Entity” option.
3. Before invoking the simulator (ModelSim – Intel FPGA Start Edition) the first time, you need to synthesize your design. To perform this action, open the submenu “Start” from the Processing menu and select the “Analysis and Synthesis” option. Assuming no errors in the RTL code, compilation will finish and you will be able to start the simulator.
4. To start the simulator, open the submenu “Run Simulation Tool” from the Tools menu and click on the “RTL Simulation” option. After starting ModelSim you should see a display similar to the one in Figure 2.
5. The working library of the simulator (*work*) should contain all the modules of your design and the FIR test bench file *fir_tb.v*. To add the FIR test bench, click on the “Compile” option from the Compile menu. A window will pop up prompting you to select a file to compile. Navigate to the location of the FIR test bench (/fir_filter_arm/*bench/*fir_tb.v.) select the file and click on the “Compile” button to compile it and then close the window. Keep in mind that every time you modify a Verilog file, you need to recompile the design following the same steps.
6. Assuming that you have successfully compiled the implementation files and the FIR test bench you can start the simulation. To do so, right click on the fir_tb.v file and select the “Simulate” option (see Figure 3).
7. To use the Waveform viewer first select all the signals you want to examine from the Object window and then right click and choose the “Add Wave” option.
8. Finally, run the simulation by typing the following command into the Transcript pane of the simulator: **run 8 us**

If your design works correctly you should see a display similar to the one in Figure 4. For the purposes of this assignment, the above steps are sufficient to test and debug your design. However, it is important to familiarize yourself with the simulator and learn how to use it efficiently.

Figure 2: ModelSim – Intel FPGA Start Edition – Home screen

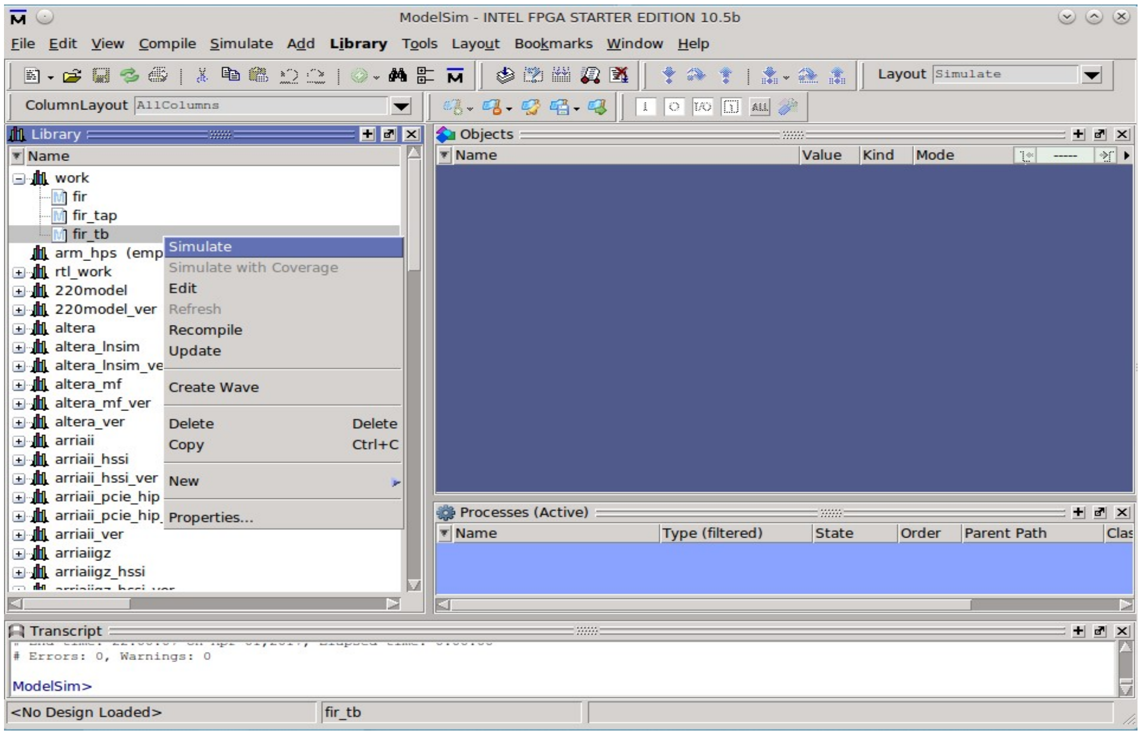


```
#        fir_tap
# End time: 21:27:12 on Apr 01,2017, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
#
# |
# stdin: <EOF>

ModelSim>
```

<No Design Loaded>   <No Context>

Figure 3: Start the simulation of the FIR module

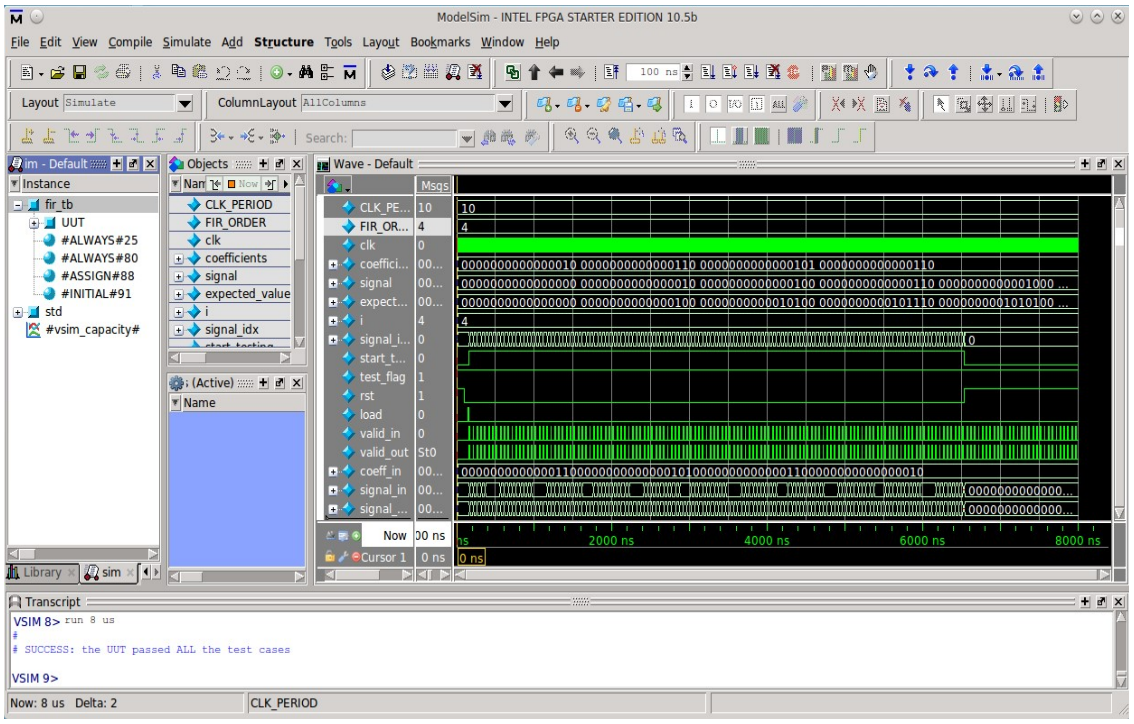


```
# Errors: 0, Warnings: 0

ModelSim>
```

```
<No Design Loaded>    fir_tb
```

Figure 4: Running the simulation of the FIR module



```
VSIM 8> run 8 us
#
# SUCCESS: the UUT passed ALL the test cases

VSIM 9>
```

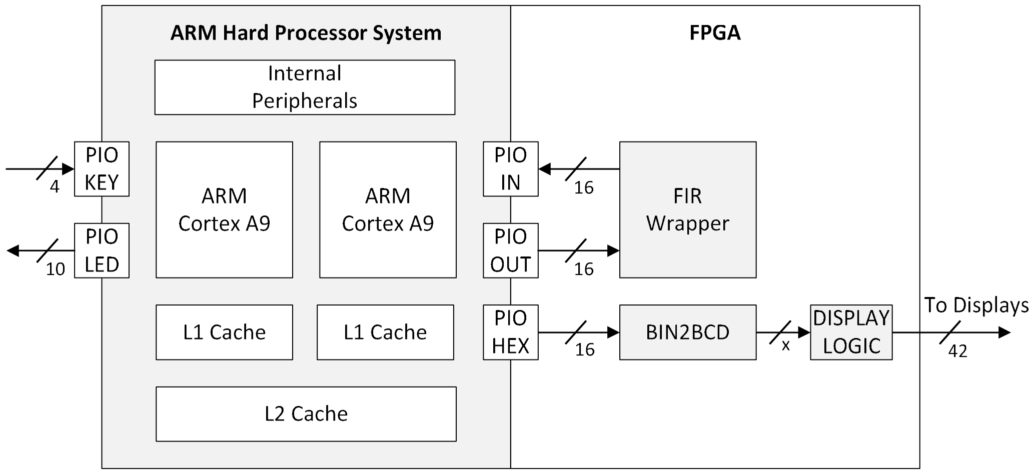
```
Now: 8 us  Delta: 2    CLK_PERIOD
```

3. Testing the Circuit

To test your design on the DE1 board you will use the ARM Hard Processor System (HPS) which is implemented as a hardcore device on the DE1 development board. The ARM-HPS consists of a dual-core ARM Cortex-A9 processor, on-chip memories and peripherals. We would like to repeat the testing procedure performed via simulation using the FPGA/processor board hardware. The ARM-HPS will be responsible for providing the FIR module with the values of the input signals and gathering the outcomes which are examined to determine if they match expected values. You have been provided with the source code of a program (*fir_filter_arm/sw/test_fir_filter.c*), written in C, which will be executed by the ARM-HPS. Figure 5 illustrates the top-level architecture of the system you will implement to test the FIR module on the board.

The **FIR** module will be implemented on the FPGA fabric using standard components such as LUTs, Flip-flops etc. The logic needed for the communication of your module with the ARM-HPS is included in the module **FIR WRAPPER**. The module has been provided to you (*fir_fliter_arm/rtl/fir_filter_wrapper.v*) but you still need to instantiate your FIR module inside the Verilog file.

Figure 5: High-level block diagram of the FIR testing circuit



The FIR WRAPPER is responsible for the following functions: (1) it monitors the parallel output port (PIO OUT) of the ARM-HPS to gather the values of the FIR module input signals *signal_in.* Whenever the wrapper receives a new incoming value it asserts the *valid_in* signal of the FIR module and places the data on the input port (*signal_in*) of the FIR module. The data remains on the input port until a new value arrives but the *valid_in* signal is only asserted for one clock cycle; (2) it instantiates a FIFO to hold FIR outputs. The write *enable* signal of the FIFO is triggered whenever the FIR module outputs a new value. The FIFO is sized to accommodate all produced results by the FIR module. You do need to be sure that your FIR module only places valid results on the out port of the FIR module (*signal_out*); (3) after input value processing and the generation of outputs that are stored in the FIFO, the FIR WRAPPER module uses a counter to read the FIFO and place the stored results on the input port (PIO IN) of the ARM-HPS. A software function reads these results of the FIR module and compares them to expected values using *test_fir_filter*.

After finishing the comparison, the software function turns LED0 on the board on to signal the end of the testing process. If all retrieved results match with expected values it also turns on LED1. By observing these LEDs, you will be able to test whether the FIR module works properly on the FPGA or if there are issues need to be addressed.

The six Hex displays on the board can be used to check the produced results. Pressing KEY1 will trigger the operation of *test_fir_filter.c* which will place all results generated by the FIR module (one every second) on the output port (PIO HEX) of the ARM-HPS. Note that the data placed on this port are in binary format. Each Hex display on the board is controlled separately and, hence, you cannot directly connect the PIO HEX port to the displays. For this reason, you will design, using Verilog, a **BIN2BCD** converter so that the binary values that the ARM-HPS outputs on the PIO HEX signals can be converted to BCD representation. Additionally, as shown in Figure 5, you will connect the PIO HEX output port of the ARM-HPS to the data input port of the BIN2BCD converter. Finally, you need to design the **DISPLAY LOGIC** module. This module turns on each LED of the Hex display on the board according to the BCD digits provided by the BIN2BCD unit. Please consult Section 3.6.2: 7-segment Displays of the DE1 development board manual [2] before implementing this unit.

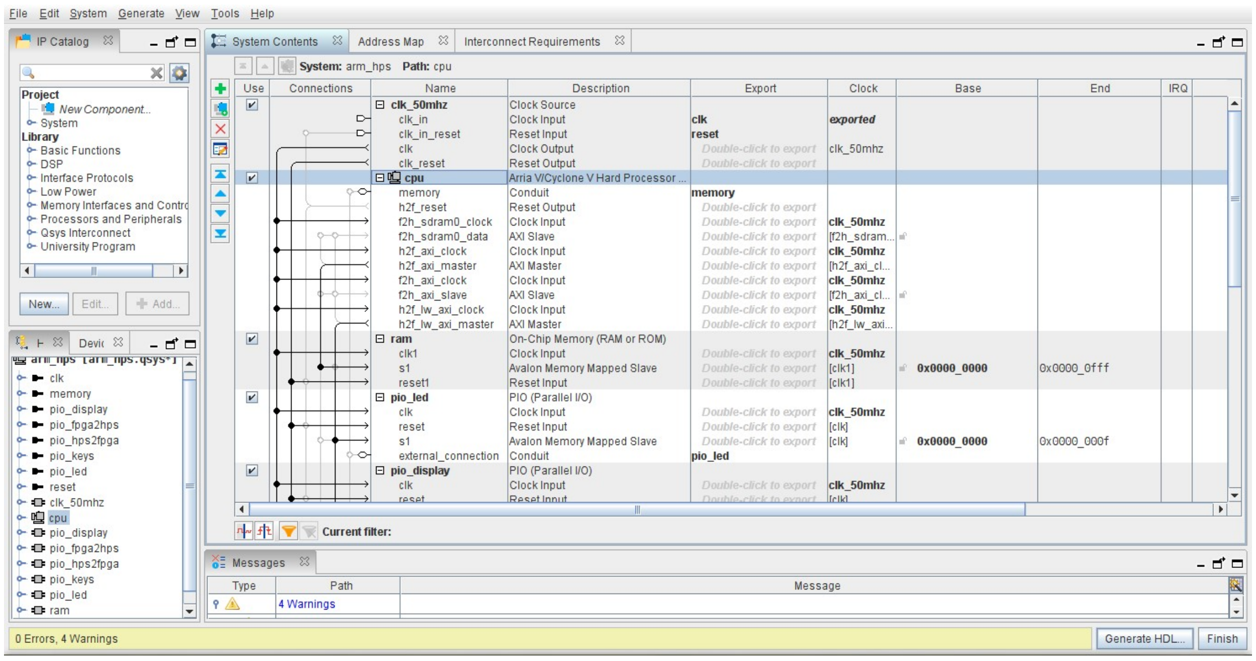
4. Building the Project

You have been provided with a zip file (*fir_filter_arm.zip*) which includes all the files needed to build the Quartus project. Open Quartus Prime Lite Edition 16.1 and follow the directions below to build the project:

0. On the home screen, select “Open Project” from the File menu and locate the Quartus project file *fir_filter_arm.qpf*. You can find this file in the following path: /fir_filter_arm/syn/fir_filter_arm_5csema5/.
1. First, you will need to regenerate the QSys system which contains the ARM-HPS IP block. To do this, in Quartus Prime select “QSys” from the Tools menu and wait for QSys to launch. A window will pop up prompting you to select a .qsys file to open. Select the *arm_hps.qsys* which is located in the directory /fir_filter_arm_5csema5/.
2. Click “Generate HDL” at the bottom of the window to open the Generation window. In this window you can specify synthesis and simulation parameters as well as the path of the output directory. Keep the default settings and click on the “Generate” button at the bottom. A window will pop up to inform you that your QSys project has been saved successfully. Select “Close” to close the window and wait for the system to generate the files needed. Once finished, click again on “Close” to close the generation window and press the “Finish” button at the bottom to close the QSys project. A window will pop up to inform you that you can add the generated .qip file to your project. Ignore the message by clicking on the “Ok” button since the .qip file has already been added to your project.
3. Next, ensure that you have added all the Verilog files that you have written to describe the FIR filter, the BIN2BCD converter and the DISPLAY LOGIC module in the project by checking the File menu in the Project Navigator. If there are missing files, follow Step 1 of Section 2 to add them to the Quartus project.

4. **Make sure that now the *fir_filter_arm_top.v* is the top-level entity of the project.** To do so, follow Step 2 of Section 2.
5. The final step in project building is to compile the design to generate the .sof bitstream file that will program functionality into the FPGA. Select "Start Compilation" from the Processing menu. Assuming no errors in the RTL code, compilation will finish and you will be able to program the FPGA with your design. It is safe to ignore the post compilation warnings. If error messages are shown during compilation, fix the errors in the Verilog code and restart the compilation.

Figure 6: ARM-HPS, peripherals and connections in QSys system



System: arm_hps Path: cpu

Use	Connections	Name	Description	Export	Clock	Base	End	IRQ
✓		⊟ clk_50mhz	Clock Source					
		clk_in	Clock Input	clk	exported			
		clk_in_reset	Reset Input	reset				
		clk	Clock Output	Double-click to export	clk_50mhz			
		clk_reset	Reset Output	Double-click to export				
✓		⊟ cpu	Arria V/Cyclone V Hard Processor ...					
		memory	Conduit	memory				
		h2f_reset	Reset Output	Double-click to export				
		f2h_sdram0_clock	Clock Input	Double-click to export	clk_50mhz			
		f2h_sdram0_data	AXI Slave	Double-click to export	[f2h_sdram...			
		h2f_axi_clock	Clock Input	Double-click to export	clk_50mhz			
		h2f_axi_master	AXI Master	Double-click to export	[h2f_axi_cl...			
		f2h_axi_clock	Clock Input	Double-click to export	clk_50mhz			
		f2h_axi_slave	AXI Slave	Double-click to export	[f2h_axi_cl...			
		h2f_lw_axi_clock	Clock Input	Double-click to export	clk_50mhz			
		h2f_lw_axi_master	AXI Master	Double-click to export	[h2f_lw_axi...			
✓		⊟ ram	On-Chip Memory (RAM or ROM)					
		clk1	Clock Input	Double-click to export	clk_50mhz			
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk1]	0x0000_0000	0x0000_0fff	
		reset1	Reset Input	Double-click to export	[clk1]			
✓		⊟ pio_led	PIO (Parallel I/O)					
		clk	Clock Input	Double-click to export	clk_50mhz			
		reset	Reset Input	Double-click to export	[clk]			
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_0000	0x0000_000f	
		external_connection	Conduit	pio_led				
✓		⊟ pio_display	PIO (Parallel I/O)					
		clk	Clock Input	Double-click to export	clk_50mhz			
		reset	Reset Input	Double-click to export	[clk]			

Messages

Type	Path	Message
⚠	4 Warnings	

0 Errors, 4 Warnings

5. Running the Test Program

To compile, load and run the test program in the *test_fir_filter.c* source file, use the **Altera Monitor Program**. The Monitor Program is a software tool that can compile/assemble an ARM/NIOS II software application. It runs on a Host PC and can be used to download the application into the processor on the FPGA device. The Monitor Program then runs or debugs the executable by communicating with the processor system. In [3] you can find a detailed tutorial of how to use the Monitor Program.

The Monitor Program is available as a part of Altera's University Program Design Suite (UPDS) and you can download it from the Download Center on Altera's website at https://www.altera.com/support/training/university/materials-software.html#Monitor-Program. Follow the installation instructions in Section 2: Installing the Monitor Program of the tutorial [3] to download the appropriate set up file and install the Monitor Program on your computer.

Once the installation is finished, start the Monitor Program software and click "Yes" if a window pops up asking you to enable the new debugging features of the latest release. You have been provided with the source code of the testing program (*test_fir_filter.c*) and the corresponding Monitor Program project file (*fir_filter_arm/sw/test_fir_filter.amp*). Execute the following steps below to compile and run the testing program on the ARM-HPS system after completing the steps in Section 4:

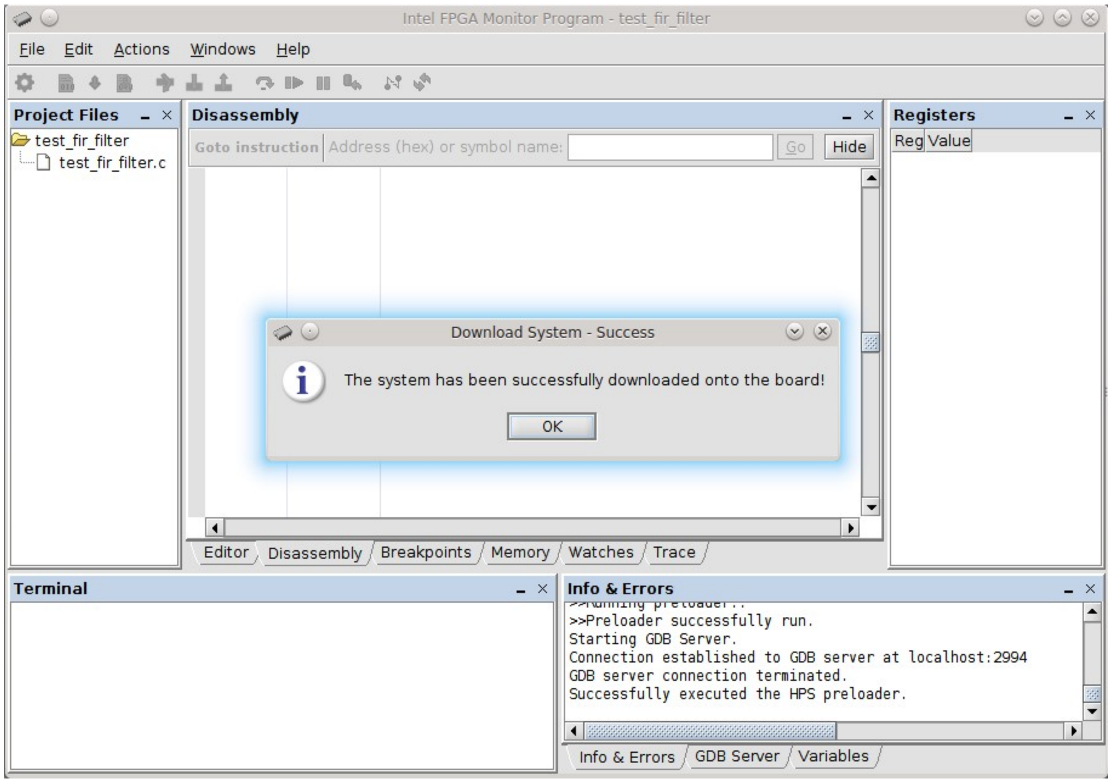
1. On the home screen, select "Open Project" from the File menu and locate the Monitor Program project file *test_fir_filter.amp*. You can find this file in the following path: /fir_filter_arm/sw/. A window will pop up asking you if you want to download onto the board the system associated (*fir_filter_arm.sof*) with this project. Click the "Yes" button once the board is powered up and connected to your computer. Once the FPGA is successfully programmed, a display similar to the one in Figure 7 should appear.
2. Compile the testing program (*test_fir_filter.c*) and then load the generated SREC file onto the board. To do so, select the option "Compile & Load" from the Actions menu.

After successfully loading the program, the processor will halt at the first instruction in the testing program. At this point, you should see a display similar to

Figure 8 where the first instruction is highlighted with yellow shading. Run the testing program by selecting the option "Continue" from the Actions menu or by clicking on the green Play button on the toolbar.

As mentioned in Section 3, the test program will send input signal values to the FIR module and then gather output data. It will then compare the outcomes of the FIR module with expected values. Upon finishing the comparison phase, the test program illuminates LED0 on the board and if all returned results match expected values, LED1 will be illuminated as well.

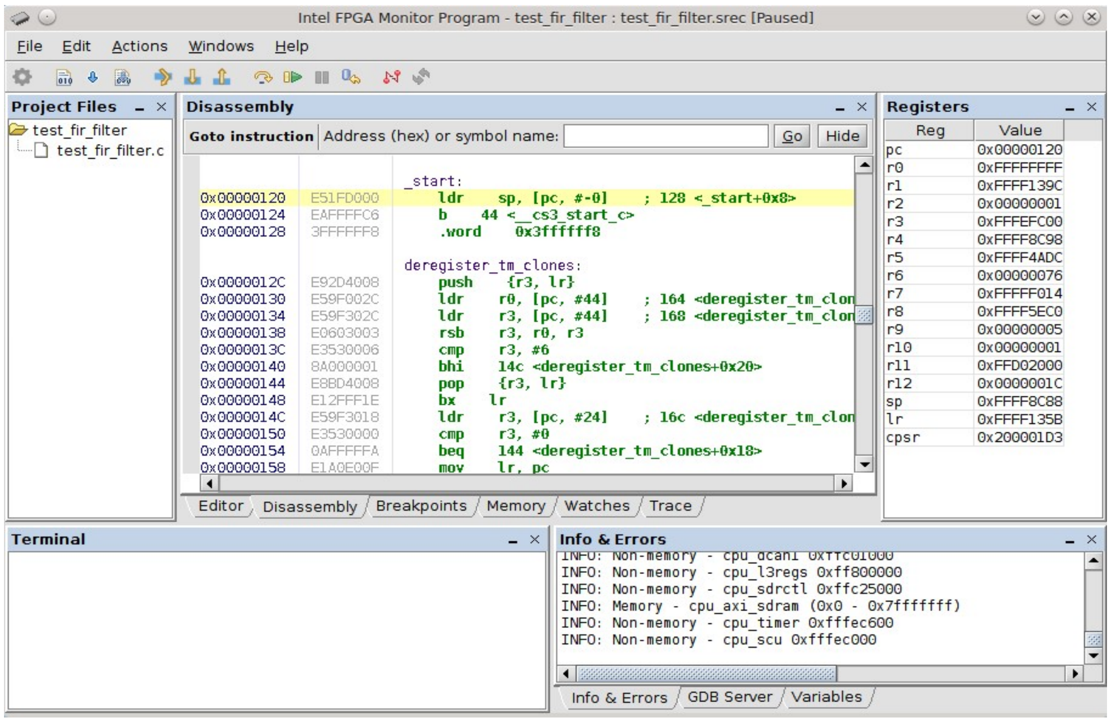
Figure 7: Downloading the testing program onto the FPGA board



The system has been successfully downloaded onto the board!

```
>>Preloader successfully run.
Starting GDB Server.
Connection established to GDB server at localhost:2994
GDB server connection terminated.
Successfully executed the HPS preloader.
```

Figure 8: Completing the loading of the testing program onto the ARM-HPS



```
                           _start:
0x00000120   E51FD000          ldr     sp, [pc, #-0]      ; 128 <_start+0x8>
0x00000124   EAFFFFC6          b    44 <__cs3_start_c>
0x00000128   3FFFFFF8          .word    0x3ffffff8

                           deregister_tm_clones:
0x0000012C   E92D4008          push    {r3, lr}
0x00000130   E59F002C          ldr     r0, [pc, #44]      ; 164 <deregister_tm_clon
0x00000134   E59F302C          ldr     r3, [pc, #44]      ; 168 <deregister_tm_clon
0x00000138   E0603003          rsb     r3, r0, r3
0x0000013C   E3530006          cmp     r3, #6
0x00000140   8A000001          bhi     14c <deregister_tm_clones+0x20>
0x00000144   E8BD4008          pop     {r3, lr}
0x00000148   E12FFF1E          bx    lr
0x0000014C   E59F3018          ldr     r3, [pc, #24]      ; 16c <deregister_tm_clon
0x00000150   E3530000          cmp     r3, #0
0x00000154   0AFFFFFA          beq     144 <deregister_tm_clones+0x18>
0x00000158   E1A0E00F          mov     lr, pc
```

Reg	Value
pc	0x00000120
r0	0xFFFFFFFF
r1	0xFFFF139C
r2	0x00000001
r3	0xFFFEFC00
r4	0xFFFF8C98
r5	0xFFFF4ADC
r6	0x00000076
r7	0xFFFFF014
r8	0xFFFF5EC0
r9	0x00000005
r10	0x00000001
r11	0xFFD02000
r12	0x0000001C
sp	0xFFFF8C88
lr	0xFFFF135B
cpsr	0x200001D3

```
INFO: Non-memory - cpu_dcan1 0xffc01000
INFO: Non-memory - cpu_l3regs 0xff800000
INFO: Non-memory - cpu_sdrctl 0xffc25000
INFO: Memory - cpu_axi_sdram (0x0 - 0x7fffffff)
INFO: Non-memory - cpu_timer 0xfffec600
INFO: Non-memory - cpu_scu 0xfffec000
```

6. Hints and Tips

- Inside the provided Verilog files there are sections labeled "**FILL THIS OUT**". Locate these sections and follow the provided instructions and tips to instantiate the three modules that you need to implement (FIR, BIN2BCD converter and the DISPLAY LOGIC unit).
- Signals from the on-board push buttons should perform the following operations in the FPGA:
  - KEY0: press KEY0 to reset FIR_WRAPPER if needed.
  - KEY1: when the testing procedure is complete, you can tap KEY1 and the testing program will start printing the values of the filtered signal (one every second) produced by the FIR module on the HEX displays. You can stop the printing of the results by holding down KEY1 for two seconds.
  - KEY2: press KEY2 to repeat the test. **Make sure before repeating the test to reset the FIR_WRAPPER by pressing KEY0.** If you forget to reset FIR_WRAPPER, you can reset the system by reloading the bitstream into the FPGA.
- If you modify the provided files make sure that the constant parameter SIGNAL_SIZE (defined in the *test_fir_filter.c* file) and the Verilog parameter SIGNAL_SIZE_LOG (defined in the *fir_filter_wrapper.v* file) have the same value. Note that the latter is expressed in powers of two.
- Carefully study the Peripherals Connected section of the DE1 development board manual [2] before trying to implement the DISPLAY_LOGIC module.
- When implementing the DISPLAY_LOGIC unit, consider that the values of the filtered signal can be negative. Since the width of the data buses is 16-bit and only six digits are needed to display a result, the middle LED of the left most display can be used to indicate the sign of the number being displayed.

- Please note that the range of the output values of the FIR module should be all the integers from **-32,766 to 32,767** (-(2^{N-1})+2 to 2^{N-1}-1). The two most negative values (-32,768 and -32,767) are reserved by the framework and used in the communication of the FIR WRAPPER with the HPS. If the FIR module generates any of these two values, the FIR WRAPPER will send the value -32,766 to the HPS instead.
- Ensure that you have checked the polarities of the on-board I/O components (buttons, switches, and LEDs) before trying to use them in your system.

7. Deliverables

To complete the assignment, you need to submit a zip file that will includes the modified Verilog source files and test benches you have written to describe and test the FIR, the BIN2BCD converter and the DISPLAY LOGIC module. Note that your code should be well commented and tested, and fully comply with the specifications of the provided framework, since the latter is the primary means for checking your solution on the board.

In addition, you need to deliver a technical report of approximately five to ten pages which should include the following sections:

- Introduction
- Design of FIR: provide the RTL schematic of your top-level design generated by Quartus Prime, and a brief discussion on the design approaches you have followed to implement the FIR.
- Testing: briefly discuss any test bench you have written, and provide screenshots with waveforms from simulations you have run to verify the correctness of your design.
- Results: study the synthesis report generated by the Quartus Prime and explain the details of the implementation results (Flip-flops, LUTs etc.).
- Conclusion and possible issues you have encountered.

8. Supplementary Files

The following table summarizes the provided files and describes their purpose.

Table 2: File types and directory structure of the project

Directory	File	Description
fir_filter_arm/		
sw/	test_fir_filter.c tets_fir_filter.amp	This directory contains the source code of the testing program and the Monitor Program project file

tests/	coeff.txt, signal_in.txt expected_results.txt	This directory contains the text files that the FIR test bench reads to obtain the coefficients of the filter, the values of the input signal and the expected results
bench/	fir_tb.v	This directory contains the Verilog file of the FIR test bench
rtl/	fir_filter_arm_top.v fir_filter_wrapper.v fifo.v	This directory contains the Verilog files for the top-level module, the FIR wrapper and a FIFO unit used by the wrapper
syn/fir_filter_arm_5csema5/	osc_50_pll/ fir_filter_arm.qpf fir_filter_arm.qsf arm_hps.qsys	This directory contains the Quartus Prime project and settings file as well as the project file of the QSys system
doc/	HDL_style_qts_qii51007.pdf DE1-SoC User Manual.pdf Intel_FPGA_Monitor_Program_ARM.pdf adder/{adder.v, full_adder.v, adder_tb.v}	This directory contains the PDF files of the manuals referred to in this document and the design example that shows how to design parameterized modules

9. References

[1] Altera Corporation (2014). Recommended HDL Coding Styles. Retrieved from: http://people.ece.cornell.edu/land/courses/ece5760/DE1_SOC/HDL_style_qts_qii51007.pdf

[2] Terasic (2014). DE1-SoC User Manual. Retrieved from: ftp://ftp.altera.com/up/pub/Altera_Material/Boards/DE1-SoC/DE1_SoC_User_Manual.pdf

[3] Intel Corporation (2016). Intel FPGA Monitor Program Tutorial for ARM. Retrieved from: ftp://ftp.altera.com/up/pub/Intel_Material/16.1/Tutorials/Intel_FPGA_Monitor_Program_ARM.pdf