2IN35 – VLSI Programming – Lab Work

Assignment 2: Fast FIR filter

Hrishikesh Salunkhe, h.l.salunkhe@tue.nl,
Alok Lele, a.lele@tue.nl

May 13, 2015

Contents

1 Introduction

In this assignment you will:

- Compile an FIR filter hardware design to an FPGA representation.
- Design a high speed pipelined FIR filter as a component of an embedded system, compile that system, and run it on the Spartan 6 ATLYS board.

For this assignment you need the following files from the website:

- Two embedded system designs in which your filter will be placed: `L2_audio_single_rate_offline.rar` and `L2_audio_single_rate_realtime.rar`
- The FIR filter implementation that will be used: `filter.rar`
- The raw audio file: `input.bin`

2 Place & Route

The 32-stage direct-form FIR filter similar to assignment 1 will be used as a baseline for the rest of this assignment. So create an ISE project with FIR specific verilog files that are downloaded from the website, specific to assignment 2 (For now, do not add `firwrapper.v` to the project.).

Note that in this assignment, `fir.v` module reads filter coefficients from the external system through the input port 'h_in'. This is another way of reading coefficients. When implementing your filter, all the i/o channels of the filter are mapped to FPGA pins represented by "bonded IOBs". In this case, the synthesis of the filter design will fail since it exceeds the number of bonded IOBs on the FPGA. This is because a lot of pins are required for the coefficients. The module `firwrapper.v` contains a shift-register for the coefficients, trading IOBs for regular flip-flops. Add this new module using *Project* and then clicking *Add Copy of Source....* (Note that select it for only implementation.). Now, the tool should automatically infer `firwrapper` as Top Module, and you will be able to synthesize your design.

The synthesize process compiles the Verilog source code in your project into a high level circuit representation, called a net list. A net list only contains information about the circuit elements and their connectivity, not how the circuit will actually be laid out on the FPGA. This results in timing estimates that are too optimistic, because it is not always possible to place elements that are near each other in the net-list close to each other on the FPGA.

To improve the timing estimates, a lower level representation of the design is necessary, where individual elements have been mapped to FPGA elements (e.g. LUTs and MULTs) and where interconnections have been mapped to the FPGA interconnection fabric. In Xilinx ISE, this is a 3-step process, consisting of *Translate*, *Map* and *Place & Route*.

Before running these processes, you have to enable the generation of an extended timing report after place and route, which is disabled by default. This is done as follows: select `firwrapper` (the top level module) in the sources list, then go down the process tree to *Implement Design*

→ *Place & Route* → *Generate Post Place & Route Static Timing* and select *Properties...* from its context menu. In the property dialog, first set *Property display level* to "advanced", then set the *Report Type* to "Verbose Report" and mark the option *Perform Advanced Analysis*.

Now execute the *Place & Route* process for `firwrapper`. This will automatically run all processes on which it depends (synthesize, translate, and map) that haven't been executed yet, or for which the output is outdated. The design summary should now have been extended with reports for every executed process plus a static timing report. In the (Post-PAR) static timing report you can find 3 timing figures, which correspond to 3 of the 4 timing figures reported by the synthesis process:

- Minimum period: the largest delay among all combinational paths (paths with no registers on them) starting at a registers and ending at a register.
- Minimum input required time before clock: the largest delay among all combinational paths starting at an input channel and ending at a register.
- Maximum output delay after clock: the largest delay between an input and a register of your design.

A derivation of these values can be found in this report as well. Note that the maximum combinational path delay, which was 1 of the 4 numbers generated by the synthesis process, is absent in this report.

When you actually implement your filter on the FPGA later on in this assignment, your filter will not be the only component taking up resources on the FPGA. This means that the timing statistics for the actual implementation can be worse than what you found when you placed & routed your design in isolation. Note that the maximum sample frequency of the design according to any report can be calculated by inverting the maximum of three types of delays explained earlier:

$$f_{max} = \frac{1}{\max\left(\begin{array}{c}\textit{Minimum period, Minimum input required time before clock,}\\ \textit{Maximum output delay after clock}\end{array}\right)} \tag{1}$$

The result of the place and route process can be viewed graphically (and modified if necessary) by using the *View/Edit Routed Design (FPGA Editor)* located under *Place & Route*. This process will start a separate application that displays the physical lay-out of the FPGA and where each elements of the design has been placed. If you enable *Routes* on the toolbar, the interconnections between the elements will also be displayed.

Questions

1. What is the maximum sample frequency of the design according to the synthesis report?
2. What is the maximum sample frequency of the design according to the static timing report? Compare this with your answer to the previous question.

3 Designing a faster FIR filter

Design a direct-form FIR filter with 32 taps, that can run at a clock frequency and sample frequency of $100Mhz$. The filter module you design must be named "fir" and have the same parameters and i/o channels as the original FIR filter design of this assignment.

When implementing your filter, all the i/o channels of the filter are mapped to FPGA pins. Once your filter has been integrated with the provided basic system, the i/o channels of your filter will connect to other components on the FPGA, instead of connecting to the FPGA pins directly. Therefore, `firwrapper` will not be integrated into the basic system and hence, do not make any changes in `firwrapper`.

Use the static timing report to see if your design is able to meet the required timing specifications. And make sure you test your filter in the simulator for functional correctness. Once your design is functionally correct and meets the timing specifications, you can proceed to the next section.

Although your design probably does not exceed the capabilities of the FPGA, the available resources on the FPGA are shown in table 1. The first column of the table lists the types of resources available on Spartan 6 ATLYS FPGAs, the second column lists the number of resources available on the Spartan 6 ATLYS FPGA, the third column lists the number of resources occupied by the basic system, and the fourth column lists what remains for your filter implementation. The basic system is used, among other things, to communicate to the board via its serial port and its audio channels. Note that we only listed the resource types which are of interest for this assignment.

Resource	**Available**	**Utilized**	**Remaining**
Flip Flops	54,576	6,080	48,489
Slice LUTs	27,288	7,444	19,844
DSP48A1(MULT18x18s)	58	3	55
BRAMs	116	22	94

Table 1: Resources used by the bare audio processing system

Furthermore the clock signals can be shared between the basic system and your filter, since they will be running at the same frequency. The basic system runs only at 100Mhz, so the filter will run 35Mhz slower than its specification requires, which gives a comfortable buffer when it is integrated into the system.

Questions

3. What did you do to obtain the required sample frequency?
4. How many of the available resources are utilized by your design?
5. Can you account for all utilized resources?
6. What is the maximum sample frequency of your design according to the synthesis report after modification?

4 Integrating your design into the base system

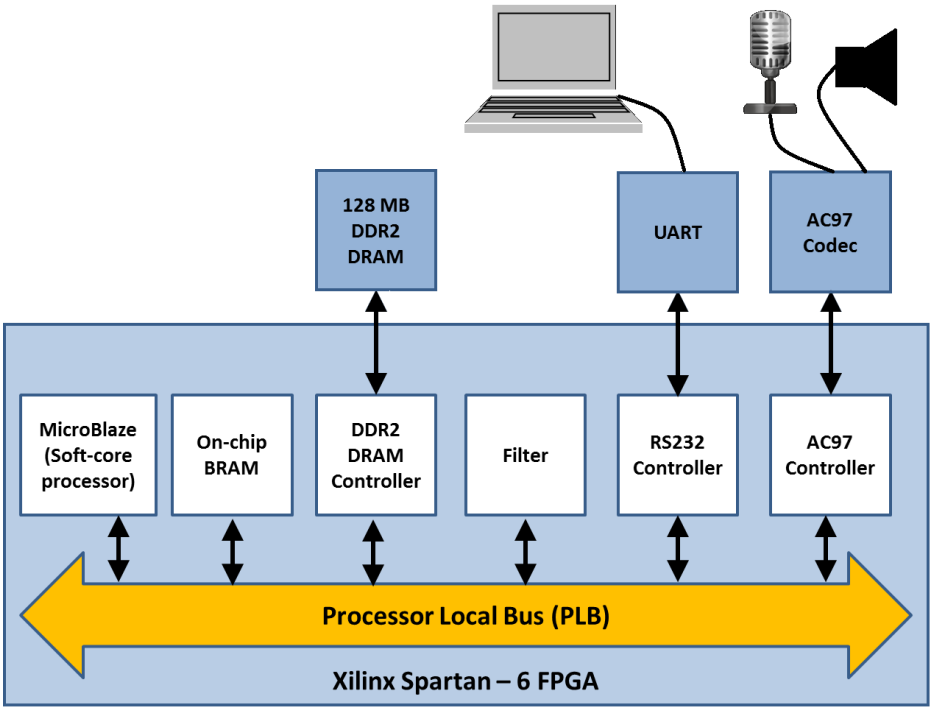

Figure 1: The architecture of the audio processing system

Before being able to download your design to the board, it has to be integrated into the base system that is provided on the website (the file `L2_audio_single_rate_realtime.rar`). The architecture of the base system is shown in figure 1. To compile the base system, the Xilinx Embedded Development Kit (EDK) is required. Because there is no WebPACK version of the EDK, the full version has been installed on the ngrid machines. There are 16 different ngrid machines where you can access EDK from (`ngrid1.win.tue.nl` to `ngrid16.win.tue.nl`). To access any ngrid machine, you need an SSH client.

To integrate your design into the base system, perform the following steps:

1. Create a working directory (`mkdir <dirname>`) in your home directory on an ngrid machine, and go into it (`cd <dirname>`).
2. Transfer the base system `L2_audio_single_rate_realtime.rar` to this newly created directory using an SSH file transfer program.
3. Unpack the zip file (`unzip L2_audio_single_rate_realtime.rar`).
4. The file `pcores/plb_filt_v1_00_a/hdl/verilog/fir.v` contains a FIR filter stub. Replace this file by your implementation.
5. Add all auxiliary files that are required for synthesis (e.g. `firstage.v`) to the same directory as `fir.v`. Simulation specific files, such as `firtest.v`, and `firwrapper.v` can be omitted.

6. The EDK system needs to be made aware of all the files you have just added to it. This requires adding lines to 1 file:

   (a) `pcores/plb_filt_v1_00_a/data/plb_filter_v2_1_0.pao`:
   find the line `lib plb_filt_v1_00_a filter verilog` and insert a similar lines before that one for each of the files you added in the previous step: `lib plb_filt_v1_00_a <filename> verilog` where you replace the `<filename>` with the file name of the extra verilog file, without its path and ".v" extension.

   Editors like `nano`, `vi` or `emacs` can be used to edit the files directly, or you can transfer those 2 files to your own system, edit them, and then transfer them back to the ngrid machine.

5 Compiling the system

The system is now ready to be compiled using the Xilinx EDK. To compile the base system, execute the compile.sh script i.e. $sh compile.sh. The standard output and standard error outputs are placed in your home directory in the files `audio_single_rate.o<jobid>` and `audio_single_rate.e<jobid>` respectively. If the tool encounters any error then it will log the error details into these files. These files can be viewed while the job is running by a command such as:

```
tail -f audio_single_rate.o123456
```

6 Programming the FPGA

If the implementation finishes successfully the following file is created in the (audio) base system root directory:

- `download.bit`: contains a binary representation of the system that can be loaded into the FPGA. It not only contains the hardware, but also the software that will run on the MicroBlaze (this software is stored in some of the FPGA's block RAMs).

Transfer the file to your computer. To program the FPGA with this `download.bit`, you can use the Adept software. It can be downloaded from http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,66,828&Prod=ADEPT2 website.

Connect the FPGA board to your laptop by using 2 USB to micro-USB cables and power it up. This adds a virtual serial port to your computer. Open your serial communication program and connect to the serial port using a speed of 230400 baud, 8 data bits, no parity, 1 stop bit and no flow control. Now start the Adept and it shall detect the board and you can select the board in the *Connect* dropdown box. Once this is done, you can browse and select the `download.bit` file and press the button *Program*. If everything goes well, it will show the output saying *Programming successful*, the FPGA is now running the provided system containing your filter. If everything goes well, you should get the main menu of the basic system, consisting of the options to go to the data transfer menu or the filter menu on the serial GUI terminal.

Note: If you would exit the menu the basic system will stop and you would need to reset/program your FPGA board to get back into the system.

7 Testing your design on the FPGA

In the data transfer menu, you can download data to and upload data from the DDR memory on the board. The DDR memory is mapped into the system's address range from 0x0000000 to 0x0fffffff (128 MB).

Now, *Download input data* option in the *Data transfer* menu to download `input.bin` using X modem *send* protocol. Be sure to use the "xmodem" protocol to transfer files.

Check the filter coefficients (using filter menu), which should be identical to the ones used in the simulator, and then execute the filter. Demonstrate the filtered output on the board.

Now repeat the above integration procedure for the second EDK *L2_audio_single_rate_offline.rar* and execute the filter on the FPGA. Demonstrate the filtered output stored on the board using the *Audio playback* menu. Then go back to the *Data transfer* menu. Upload the filtered data back onto your own system using option *Upload filtered audio data*. Compare this data to the output file from simulation (using Audacity). Play an audio file on your system and listen to the effect of your filter on it.

8 Adding constraints (Optional)

The *Place & Route* process optimizes the implementation by minimizing the delay between two registers or input to register or register to output in the design. However, it is possible to give some direction to this optimization process using constraints. To open the constraint editor, select `fir`, and run the process *User Constraints* → *Create Timing Constraints*. Press *Yes* if ISE asks you whether you want to create a constraints file and add it to the project.

Figure 2 describes the ISE User constraint editor. On the left-side pane, designated by the Blue rectangle, in the constraint editor, you can see different constraint types. Among these types, Timing Constraints section is important which is shown by the Green rectangle. Unfold the Timing Constraints label. You can see four different types under the Timing Constraints. To specify a value for the clock period, double click on the *Clock Domains*, and change the default value of "20ns" to any desired value such as "10ns" in the textbox which is labelled with *Time*, under the *Specify time* radio button. After pressing OK, the cell present on the right-side in the constraint editor, demarcated by the Red rectangle, should contain "10ns 50 % HIGH" under the columns named *Period, Duty Cycle* and *Edge* respectively. To set a constraint on the input delay, double click on the *Inputs* and press *Next*. Here you can see that the ISE has already filled the input delay constraints (Rising edge constraints) from the Clock Domains constraints for you. Now press *Finish*. Do the same for the *Outputs*, which constraints the output delay. All relevant timing constraints are now set, so save the constraints and close the constraint editor.

Now execute the *Place & Route* process again. It will now try to implement the design such that all the constraints are met. If impossible constraints have been specified (e.g. the timing constraints are lower than the synthesis estimate) it will report an error. However,

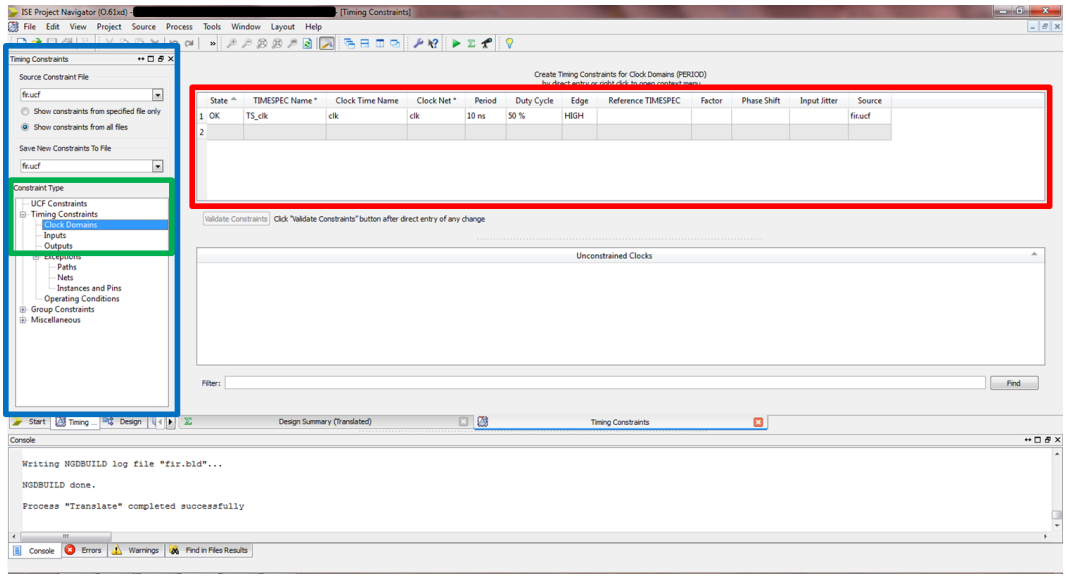

Figure 2: Xilinx ISE User Constraint Editor

if constraints have been specified that are theoretically possible but cannot be attained by *Place & Route*, it will deliver a functionally correct implementation that can run at a lower speed, but it will not report an error! The *Place & Route* report will show you which of the constraints were met and which ones were not.

Questions

7. What is the maximum sample frequency of the design according to the static timing report after adding the constraints? Compare this with your answer to the question 2?

8. Find out what the maximum sample frequency of the design is, by tightening the constraints in 0.5ns steps. Note that the time it takes to run the *Place & Route* process increases as the constraints are tightened?

9. Is it possible to get a higher sample frequency than indicated by the synthesis report?