ASIC Design Laboratory
Lab 7 : Math Algorithms in Hardware
(Convolution & FIR Filter Design)
Lab Manual

Fall 2020

The purpose of this lab exercises is to help you become familiar with implementing non-trivial multi-step math algorithms using 'datapath' style hardware where the algorithm is executed over multiple cycles.

It is important to note, that given the fundamentally parallel and interactive nature of hardware designs, debugging designs described with HDL code requires a method that strictly identifies and leverages guaranteed cause-effect relationships with in the design's description. Other lazy or speculative debugging methods will generally result in vast amounts of wasted time, effort, and frustration and can easily increase debugging times by a factor of 10x.

In this lab, you will perform the following tasks:

- Synthesize, test, and verify the functionality of the mapped version of a Moore model state machine of a Finite Impulse Response (FIR) filter.
- Use Verilog file IO to feed a grayscale image through your finite impulse response filter.

1 Lab Setup

In a UNIX terminal window, issue the following commands, to setup your Lab 7 workspace:

mkdir –p ~/ece337/Lab7
cd ~/ece337/Lab7
dirset
setup7

The setup7 command is an alias to a script file that will check your Lab 7 directory structure and give you file needed for starting the lab. If you have trouble with this step please ask for assistance from your TA.

IMPORTANT: Make sure to add this new workspace into your 337 Repository, like you did in Lab1.

This way, you will always have the original copy in storage.

2 Lab Work Overview

2.1 Required Preparation

To prepare for implementing your FIR Filter design you must complete the following:

- State Transition diagram for a Moore model FSM based controller for the provided datapath module that will perform a 2-point High-Pass FIR Filter
- Psuedocode description of a the operation of a 4-point High-Pass FIR Filter, in similar style as the 2-point version provided in Section 3.2
- State Transition diagram for a Moore model FSM based controller for the provided datapath module that will perform a 4-point High-Pass FIR Filter
- Create a complete RTL diagram for the controller block described in Section 5.2
- Create a complete schematic diagram for the magnitude block described in Section 5.5

You must have these diagrams submitted as either PDF file(s) or image files using common standards (JPEG, PNG, or BMP) via "submit Lab7prep" in order to earn points for them.

It is highly recommended that you complete all of these diagrams prior to starting to write any design code, as this should save you tremendous amounts of time debugging/rewriting code later on.

2.2 Required Verification

2.3 Expectations Regarding Lab 7

In Lab 7 you will be working on part of the design of a datapath based Finite Impulse Response (FIR) filter. In this lab, you have been given access to a library that contains complied and verified modules for implementing the datapath you will be using within the filter design, which are described in Section 6. As these modules are contained in a library they must not be included in the various makefile variables, otherwise the makefile will error out trying to find local copies of the files. You will have to use the simulation rules from the makefile in order for library they are contained in to be linked against when starting the simulation. You also have been given a test bench template. You will need to design and test the blocks described in Section 5, which includes the top-level block. You are not and will not being specifically instructed on how to design these blocks, only their intended behavior/function. You are only told the expected architecture for the design, which is comprised of the inputs to each block, the outputs from each block and what function the block is to perform. It is up to you to come up with a working solution for your blocks and then integrate the building blocks to form the Receiver block.

2.4 Grading Policy

A fully functioning Receiver Block is defined as a design that passes all the tests that are contained within the automated grading test bench. The code for this test bench will not be provided to you nor will you be told what each test case in the test bench is checking. More than eighty-five percent (>85%) of your grade for this lab will be determined from the mapped version of your design implementation. The automated grading system will run the grading test bench on both the source and the mapped version of your design, but only the mapped version results will be used for grading. Thus, in order to run the automatic grading script, your Receiver Block design must have an error-free run through Design Compiler®. Your final grade will be determined by the most recent total grade you have obtained in a mapped test run and not a combination of different test runs. You will be allowed a maximum of 3 passes through the Lab 7 grading script.

The grading script is not there to for you to use to test your design, it is there to grade your design. Much like you are expected to check your own work prior to turning in homework in other classes, you are expected to do your own testing of your design prior to submission for grading. In the real world, you will likely be designing something because you need to create the first instance of it and will not have a reference model to verify with and will have to perform all testing through a well-designed test bench. Because of this, in this lab you will be given a starter test bench to guide you in the creation of well-designed test benches. In regards to the design, you should ensure that you are naming the blocks the names that are specified in this lab. In addition, the interface signals for the top-level receiver block must be identical to those listed in Section 4 of this lab. Failure to name the interface signals correctly will result in the automated grading test bench failing and that corresponding run will count as 1 of your 3 possible runs. You will need to score 50% (30/60) or higher on your most recent mapped version test in order to satisfy the outcome for this lab.

2.5 Submission Commands

submit Lab7prep Submits the contents of your “docs” folder for the preparation phase and will be due prior to the main design submission

submit Lab7 Submits your design for automated grading

submit Lab7post Submits the contents of your "docs" folder for the post-lab work phase and will be due at the same time as the automated design grading

submit Lab7re Submits your design for automated grading for early remediation purposes and will only be activated after the regular deadline has passed for all lab sections

submit Lab7r Submits your design for automated grading for regular remediation purposes and will only be activated after the early remediation deadline has passed for all lab sections

2.6 Additional Comments

- You are required update the variables in the makefile provided to you by dirset so that it can compile and synthesize your design. The grading script will parse your makefile for the values assigned to the variables and will use those to compile and synthesize everything during the testing. If your variables are not up to date for your design it will result in the grading system not being able to compile and synthesize your design or sections of it, which will effectively waste one of your submission attempts.
- You must set the clock period constraint in the ‘CLOCK_PERIOD’ variable in your makefile, for the design to be guaranteed to run at 100 MHz.
  NOTE: Given the complexity of the design, if you only use a constraint of 10 ns for the clk constraint the synthesizer will stop working once it has “met” it even if it results in a critical path of exactly 10 ns. We have observed that the delays in QuestaSim® tend to be slightly longer than those reported by Design Compiler® in your mapped reports folders. To adjust for this, use time constraints smaller than 10 ns, such as 9ns or less, to force it to optimize further and give your design the cushion it needs to handle these longer simulation delays.
- You will be allowed a maximum of 3 passes through the Lab 7 grading test bench (i.e. 3 chances to run ‘submit Lab7’. Only the last submission will count! So use git! If a previous submission is better than your most recent submission, restore it from git and submit again.

3 Convolution and Finite Impulse Response (FIR) Filter Design

FIR Filter designs are based on convolution, where the sample stream to filter is convolved with a 'function' of defined constants which control the filter effects in the frequency domain, subject to the input or sampling rate to the filter. Figure 1 illustrates the convolution for a 4-point FIR filter.

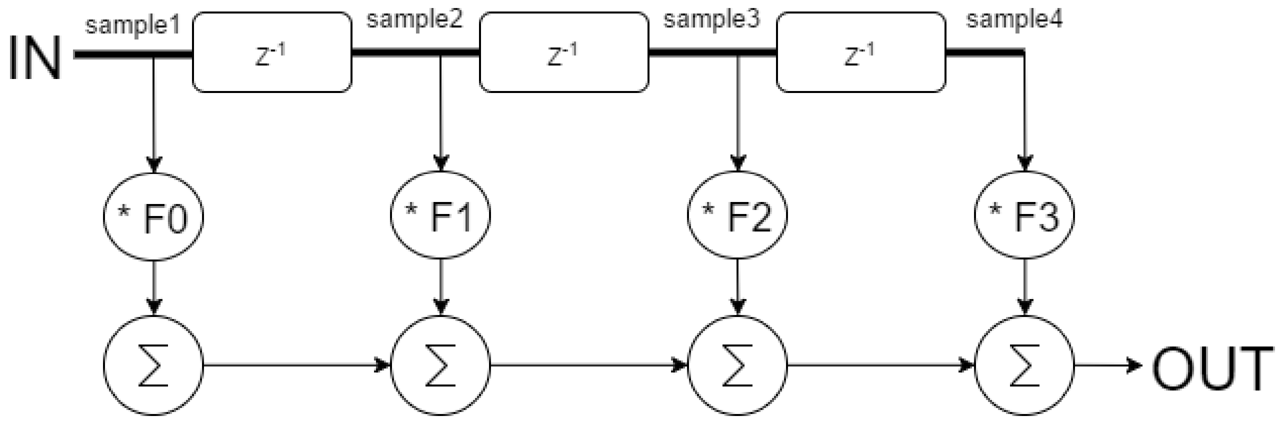

Figure 1: Finite Impulse Response Filter Theory of Operation

NOTE: Z^{-1} is equivalent to the circuit design storing a value for one clock cycle. Thus 'sample2' is 'sample1' from 1 cycle before.

3.1 FIR Coefficients for a High-Pass Filter

To implement a high-pass filter using a multiple of two coefficents, the coefficients must be symmetric in magnitude but opposite in sign. As the number of coefficients increase, the 'center' ones have higher magnitudes and with regard to sign, the new ones are successively inserted in the 'center' with an inverted sign of their respective predecessor. A 2-point (two coefficients) high-pass filter would have a large negative for F0 and a large postive for F1. A 4-point (four coefficients) high-pass filter would have a small negative for F0, large positive for F1, large negative for F2, and a small positive integer for F3.

3.2 Example Operation of a 2-Point Datapath-based High-Pass FIR Filter

To start things out, let us consider a simpler FIR filter design that only uses two samples instead of four. There should be an initial state that the FSM in the controller module goes to out of reset that handles loading in the four FIR coefficients and storing them in two registers that will not be overwritten for the duration of filter operation, reg[4] and reg[5] here. The pseudo code in Source Code 1 describes the operation of the whole design for a two-sample sliding window averaging filter using the same architecture as detailed in Section 4, excluding the initial coefficient loading phase discussed in Section 5.2.

IMPORTANT: In Source Code 1, the result of the 'large negative coefficient' multiplication is subtracted because all coefficients are actually loaded in as unsigned values and the sign is enforced via either adding or subtracting their respective product.

```
idle:   if (data_ready=0) goto idle;    // wait until data_ready=1
store:  if (data_ready=0) goto eidle;
        reg[3] = data;                  // Store data in a register
        err = 0;                        // reset error
zero:   reg[0] = 0;                     // zero out accumulator
sort1:  reg[1] = reg[2];                // Reorder registers
sort2:  reg[2] = reg[3];                // Reorder registers
mul1:   reg[6] = reg[1] * reg[4];       // sample2 * F1
add:    reg[0] = reg[0] + reg[6];       // add Large pos. coefficient
        if (V) goto eidle;              // On overflow, err condition
mul2:   reg[6] = reg[2] * reg[5];       // sample1 * F0
sub:    reg[0] = reg[0] - reg[6];       // sub Large neg. coefficient
        if (V) goto eidle;              // On overflow, err condition
        else goto idle;
eidle:  err = 1;
        if (data_ready=1) goto store;   // wait until data_ready=1
        if (data_ready=0) goto eidle;
```

Source Code 1: Psuedo-Assembly Code for the execution of a 2-Point High-Pass using a Datapath

4 Design Architecture

You will be designing an ASIC that implements a very small finite impulse response (FIR) filter using a provided datapath module. It will read in a series of data samples and output the filtered data, as well as keeping track of the number of samples processed. Each output sample will be computed as the weighted sum of the last four input samples. This design implements a configurable four sample finite impulse response filter, and is an example of a digital filter. The filter's design allows the FIR coefficients to be adjustable during operation. Such a design would be useful for smoothing an A/D conversion to reduce noise, for example. The design architecture diagram of the system is given to you in Figure 2. Although the hardware described in this lab is not the most efficient way to solve this simple problem, it is very extensible and the function of the system could be easily changed to do other tasks with very little redesign.

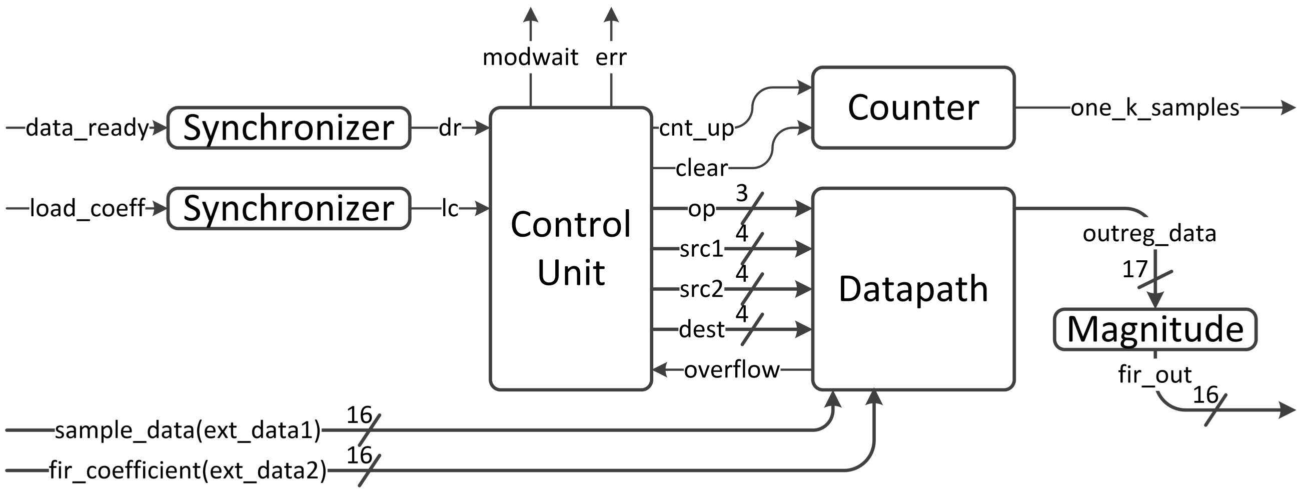

Figure 2: Finite Impulse Response Filter Architecture

NOTE: Clock and Reset signals are assumed to be sent to the appropriate blocks and are not shown.

4.1 List of the Top-Level Ports and Purpose

Clk This is the system clock port. It should be connected to a 100 MHz clock.

N_Rst This is the active-low asynchronous system reset signal.

Sample Data This is the port used for streaming in 16-bit unsigned samples.

FIR Coefficient This is the port used for sequencing in the magnitudes of the filters four coefficients.

Load Coeff This is the active-high load enable/value ready signal for the coefficient data.

Data Ready This is the active-high load enable/value ready signal for the sample data.

One K Smaples This is a status output that indicates when a multiple of 1000 samples have been processes with the current set of coefficients.

Modwait This is the active-high busy/wait signal for the design and indicates when the design is actively processing a sample and thus all other outputs are to be ignored.

FIR Out This is the unsigned sample that results from the FIR Filter operation on the past four samples.

ERR This is the active-high error flag for the design and should be asserted as soon any error happens during processing and must be maintained until either a new sample or new set of coefficients is supplied.

4.2 List of the Functional Units/Blocks and Purpose

Synchronizer The two synchronizers are used to synchronized the otherwise asynchronous data and coefficient 'ready' signals and must be 'reset-to-low' synchronizers (your 'sync_low' modules), since the 'ready' signals are active-high.

Control Unit The FSM based controller responsible for the enforcing/implementing the sequence of necessary steps for using the provided datapath to execute the overall algorithm.

Datapath The provided computational datapath (further described in Section 6) that both holds the various samples, coefficients, and calculation values in an internal Register File and performs all math and manipulation of the Register File.

Counter The counter that tracks the number of samples processed (regardless of math errors) and reports each time a multiple of 1000 samples have been processed.

5 Specifications for the Blocks You Must Design and Implement

5.1 FIR Filter

5.1.1 Block Description

This is the top level module which will connect all the individual components. An external source puts an unsigned 16-bit word on the data input, and asserts data_ready to indicate to the design that the data is valid. (Note that data and data_ready are asynchronous signals.) At this point the design will store the contents of data in a register file. Then it will multiply the oldest sample by FIR coefficient F3, through to multiplying the newest sample by FIR coefficient F0, as seen in Figure 1. Then it will sum those intermediate sample products and store the result in the accumulator (one of the registers in the datapath). Sample products that had 'negative' coefficients must be subtracted from the accumulator instead of added to it. The samples will need to be moved to make way for the next cycle, discarding the last data point, e.g., sample4 is discarded, sample3 becomes sample4, and so on. While multiplying sampleX by its relevant FX coefficient, do not overwrite the original sampleX with the result as it will still be needed for the next calculation where it will become sample(X+1). While processing occurs, the modwait signal is asserted to indicate that the system is not yet ready to process a new sample. If an overflow occurs during the filtering process, err is asserted to indicate an error and remains asserted until the beginning of the next filtering operation. The data on fir_out is not valid during this time. Figure 3 is a timing diagram showing this operation, although the 1,000 sample signal is not shown as a 1,000 samples being processed would not have legible waveforms when confined to tolerable physical size.

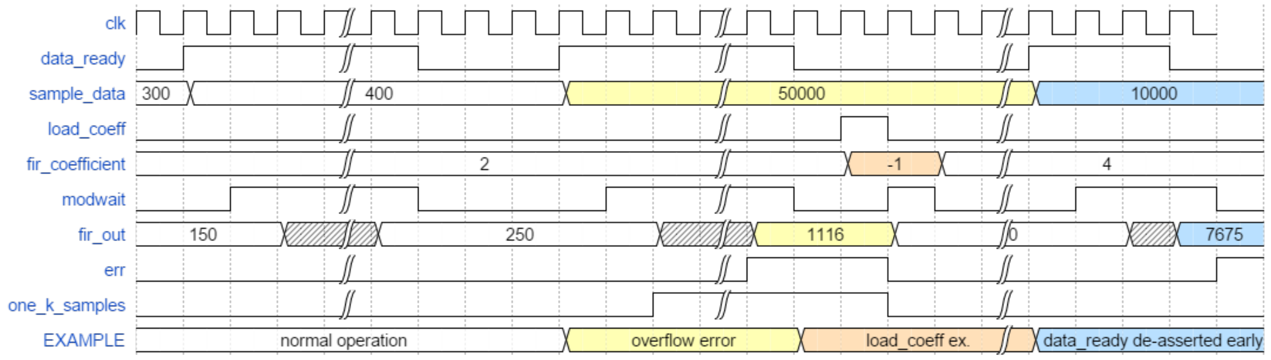

Figure 3: Example FIR Operation Waveform

5.1.2 Design Usage Constraints

- The total time to process one sample must not exceed 25 clock cycles (250 ns). Within 25 cycles of data_ready assertion, the outputs must be valid and the modwait signal deserted.
- The modwait signal must be glitch free.
- The modwait signal must always be high while processing a sample. It cannot be '0' while any of the outputs of the design are not valid/finalized.

5.1.3 Module Specifications

Required Module Name: fir_filter

Required Filename: fir_filter.sv

Required Ports:

Port name	Direction	Description
clk	input	The system clock. (Maximum Operating Frequency: 100 MHz)
n_reset	input	This is an asynchronous, active-low system reset. When this line is asserted (logic '0'), all registers/flip-flops in the device must reset to their initial value.
sample_data[15:0]	input	The 16-bit unsigned data input
fir_coefficient[15:0]	input	16-bit fixed-point coefficient [0.0, 1.0] to load
load_coeff	input	Active high signal that indicates when FIR coefficients must be loaded
data_ready	input	Active high signal that indicates when a sample is ready to be processed
one_k_samples	output	Active high signal indicating that 1,000 samples have been processed since the last assertion/coefficient load
modwait	output	Active high signal indicating that the design is busy
fir_out[15:0]	output	The 16-bit unsigned FIR filter of the last four samples
err	output	Active high signal indicating that an error occurred during the processing of the most recent sample

5.2 Controller

5.2.1 Block Description

The control unit is the brain of your system. It has to regulate and control the operation sequence and input signals to the other components in the system so your system can operate as specified. A more in-depth description of the unit's operation is as follows:

After receiving the dr signal, the new data must be stored in the register file, and the modwait signal must be raised. Lowering the modwait signal indicates that the design is ready to receive another data sample and done processing the previous sample. After receiving dr, set the modwait signal high, then the data is reorganized and the oldest data thrown out, the data counter must be incremented, and any error signals must be de-asserted.

Then the controller will manipulate the datapath to perform the FIR filtering algorithm presented in Figure 4 and explained in section Section 5.2.1. If at any point during the summation or multiplication an overflow occurs, stop processing, assert the error signal, err, and continue to the next step. (Cutting the addition short like this provides a small power savings.) If the dr signal goes low before the data is loaded the controller must stop processing it, assert the error signal, err, and not increment the counter. See Figure 4 for an example of correct behavior when dr is de-asserted before modwait is de-asserted.

Once everything is done, modwait signal must be de-asserted to notify the external device that your system is done processing the data. Keep in mind that you need to store the final addition result in the output register in the register file, so that it will be displayed via design output when the modwait signal is de-asserted. You may assume that the external device will know that unless the modwait signal is low, the data in the output line might be invalid. Also you may assume that the external device will attempt to send the design a new sample once the modwait signal transitions from high to low. This is why having a stable (i.e. no glitches) modwait signal is crucial. Section 6 describes the datapath in detail.

NOTE: The easiest way to ensure that modwait is stable is to have the output value come directly from a flip-flop. This is referred to as 'registering the output'.

Data Processing Operation The data_ready input must be asserted until after the modwait output is asserted as shown in the first "Valid 3 cycle DR" portion of Figure 4. However due to the delay of the synchronizers between the data_ready input and the dr internal signal a 2-cycle pulse of data_ready will work as well. However, a 1-cycle pulse of data_ready must result in immediate processing termination and error reporting as shown in the "Invalid 1-cycle DR" section of Figure 4.

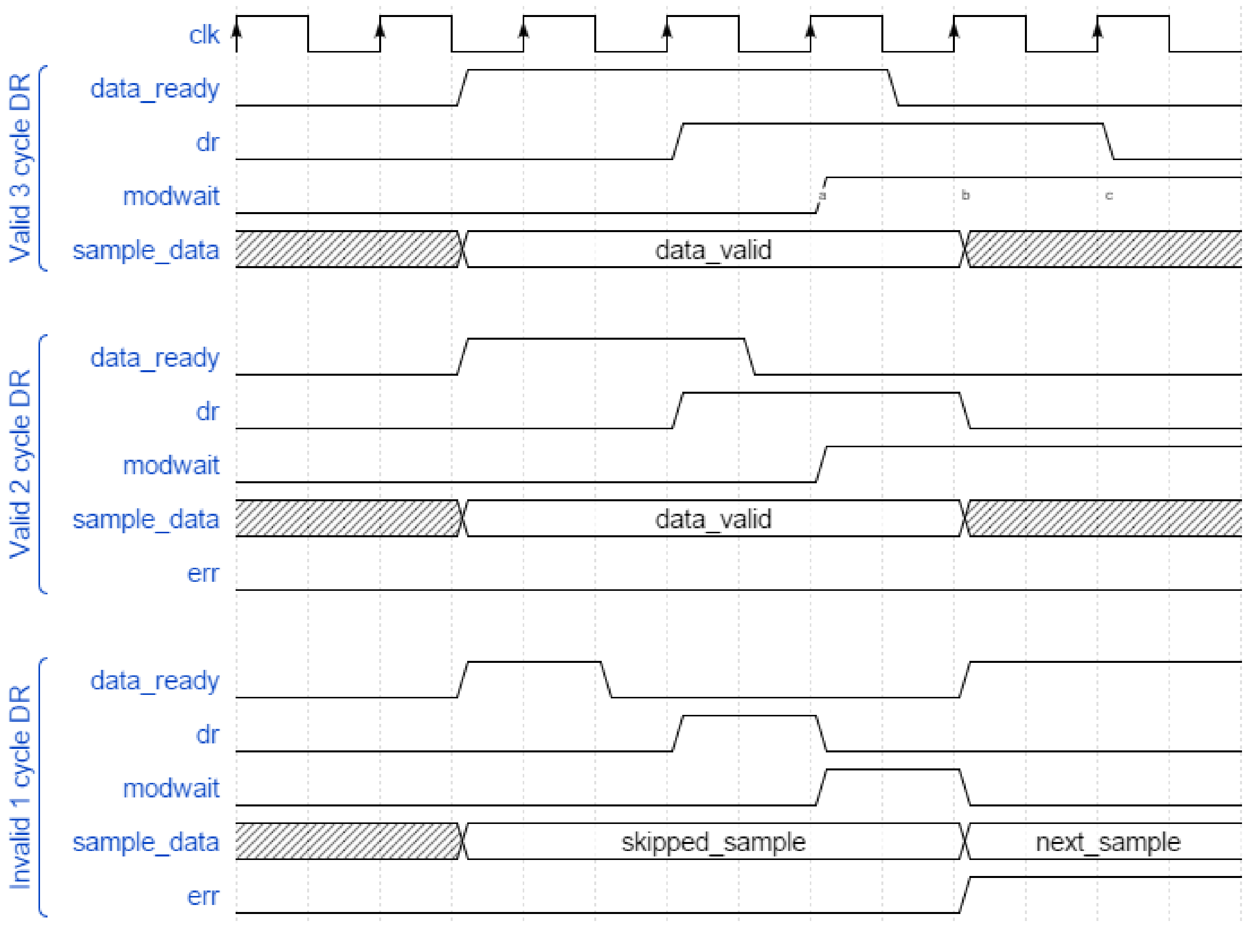

Figure 4: Data Processing Ignition Timing Diagram

FIR Coefficient Loading Figure 5 illustrates the allowed timing sequences for loading coefficients into the design. When in the IDLE state and the load_coeff signal is asserted, the control unit must handle loading the new four coefficient values into four registers of your choosing and clear the counter. Lower the modwait signal after storing each coefficient to indicate the control module is ready to receive the next coefficient. Four coefficients will be used in this design. F0 will be sent first and F3 will be sent fourth. The control module only needs to allow coefficient loading or data sample processing sequences to start while in IDLE. Once either of the sequences has started it must run to completion before a new sequence (either coefficient loading or sample processing) is allowed to start. FIR filters operate on the current sample and several relative-to-time older data samples, 1 current and 3 older data samples for this filter. It is important to note that you have to perform this algorithm in multiple cycles since you have a datapath that can only perform 1 operation per cycle. Loading of the coefficient is allowed to take up to 5 cycles (5 cycle long active modwait) per coefficient but realistically it can simply be done in 1 cycle per coefficient.

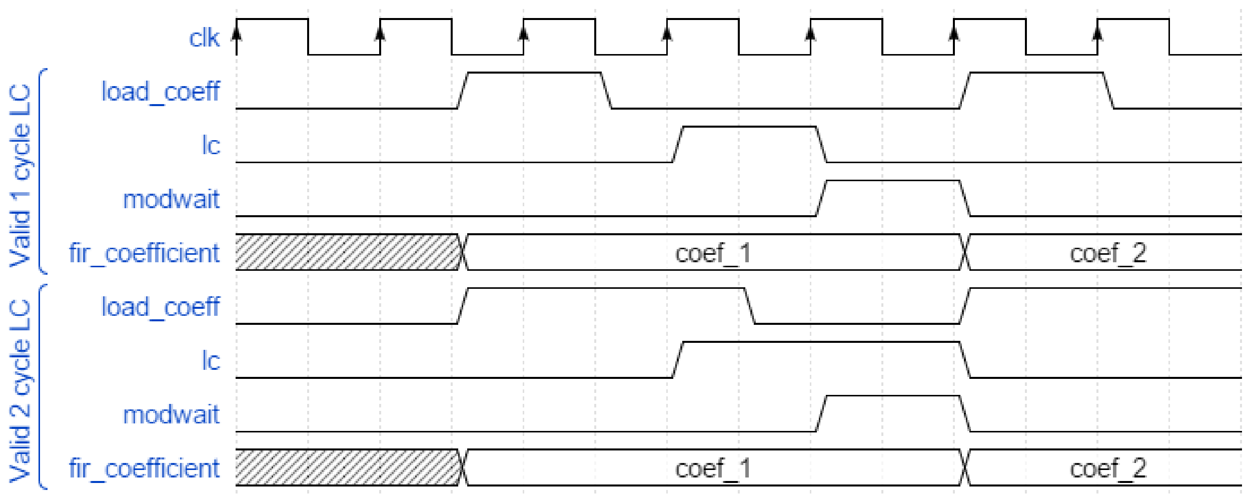

Figure 5: Allowed Load Coefficient Timing Sequences

FIR Coefficients The FIR coefficients will be a small negative for F0, large positive for F1, large negative for F2, and a small positive integer for F3. The coefficients will be sent to the FIR filter module as 16-bit unsigned fixed-point decimals. The MUL op in the datapath is not a general-purpose multiplier. The MUL op does not support signed numbers and performs fixed point math expecting the coefficient in reg[src2] and the data sample in reg[src1]. Thus, it is necessary to make the product F0*sample1 and F2*sample3 negative by subtracting them from the accumulator, instead of adding them (SUB instead of ADD). It is ok for the reg[0] to have a negative value during or after the FIR algorithm is done.

5.2.2 Module Specifications

Required Module Name: controller

Required Filename: controller.sv

Required Ports:

Port name	Direction	Description
clk	input	The system clock. (Maximum Operating Frequency: 100 MHz)
n_rst	input	This is an asynchronous, active-low system reset. When this line is asserted (logic ‘0’), all registers/flip-flops in the device must reset to their initial value.
dr	input	This is the synchronized form of the data ready signal that signifies that the next sample data is ready.
lc	input	This is the synchronized form of the load coeff signal that signifies that the next coefficient is ready load.
overflow	input	Indicates that an overflow occurred in the Datapath.
cnt_up	output	This signal is the count enable for the counter. This signal must only be pulsed for 1 clock cycle (i.e. asserted for only 1 clock cycle and then cleared).
clear	output	This signal is used to clear the counter to zero, when lc is asserted.
modwait	output	This signal is to tell the external device connected to your design that the system is still processing the new data and the external device must wait. This signal must be stable (i.e. no glitches or edges unless during transition). To achieve stability, think about the state machine style or logic devices that you could use.
op[2:0]	output	Op-code for the Datapath. See description of Datapath.
src1[3:0]	output	Register number of the first source operand for Datapath. See description of Datapath.
src2[3:0]	output	Register number of the second source operand for Datapath. See description of Datapath.
dest[3:0]	output	Register number for where to store the result of the Datapath operation. See description of Datapath.
err	output	Error flag. Asserted when an err is detected, and de-asserted when the next data sample is read in.

5.3 Synchronizer

Since the requirements are identical to those for the sync_low design you created and tested in lab 5, this should simply be a copy of that design, which is why that design file was copied into your Lab7 folder by the setup7 script.

5.4 Counter Unit

5.4.1 Block Description

The counter unit simply counts how many samples have been processed and asserts its output signal high after 1000 samples have been processed for the current set of coefficients since the last assertion or power on. It output must be held high and stable until the next sample is being processed, and must be cleared before the processing of that sample has finished. The cnt_up signal tells the counter a sample has been processed. This signal is supplied to the counter by the controller. Also, the counter must be cleared upon a change of coefficients.

NOTE: Since the requirements are really just a special case of those for the flexible counter design you created and test in lab 4, this should simply be a wrapper file that uses that design, which is why that design file was copied into your Lab7 folder by the setup script.

5.4.2 Module Specifications

Required Module Name: counter

Required Filename: counter.sv

Required Ports:

Port name	Direction	Description
clk	input	The system clock. (Maximum Operating Frequency: 100 MHz)
n_rst	input	This is an asynchronous, active-low system reset. When this line is asserted (logic '0'), all registers/flip-flops in the device must reset to their initial value.
cnt_up	input	This signal is the count enable for the counter. This signal should only be pulsed for 1 clock cycle (i.e. asserted for only 1 clock cycle and then cleared).
clear	input	This signal is a synchronous reset, set the counter to zero
one_k_samples	output	Indicates that 1,000 samples have been processed since the assertion/coefficient load. Must be an active-high signal and stable while modwait is low.

5.5 Magnitude

If the FIR algorithm produces negative output from the datapath, this block will output the positive magnitude on fir_out instead. Otherwise, this block will pass the datapath's out_reg value on to fir_out. This is desired as this block works with negative coefficients and the samples are unsigned, thus the output should be unsigned as well. The negative output will be in 2's compliment format and will have to be converted to the positive form of 2's compliment and will have to be converted to the positive form of 2's compliment, should the outreg_data of the datapath have a negative number on it.

Required Module Name: magnitude

Required Filename: magnitude.sv

Required Ports:

Port name	Direction	Description
in[16:0]	input	17-bit signed input.
out[15:0]	output	16-bit unsigned magnitude of the input.

6 Specifications for Provided Blocks

This section discusses the datapath block provided to you for this lab via the Labs_IP library. You can use objects from this library for both simulation and synthesis, but you do not have access to the source code. However, RTL and schematic level block diagrams have been provided for reference.

6.1 Block Description

A datapath is a term for the computational logic in a microprocessor. This datapath contains an ALU for arithmetic operations and a register file for storing data. There are 16 registers available for you to use, though you will not need them all for this design. Register 0 is the output register, so any values assigned to this register will appear immediately on the output. You will use this block to implement the arithmetic functions of your averaging filter. Signals "src1", "src2", and "dest" are the numbers or addresses of the registers that you would like to use as either sources (signals "src1" & "src2") of data for the operation or the destination (signal "dest") of its result.

Figure 6 depicts the overall architecture of the provided datapath. And Figure 7 depicts the architecture of the register file contained inside the datapath.

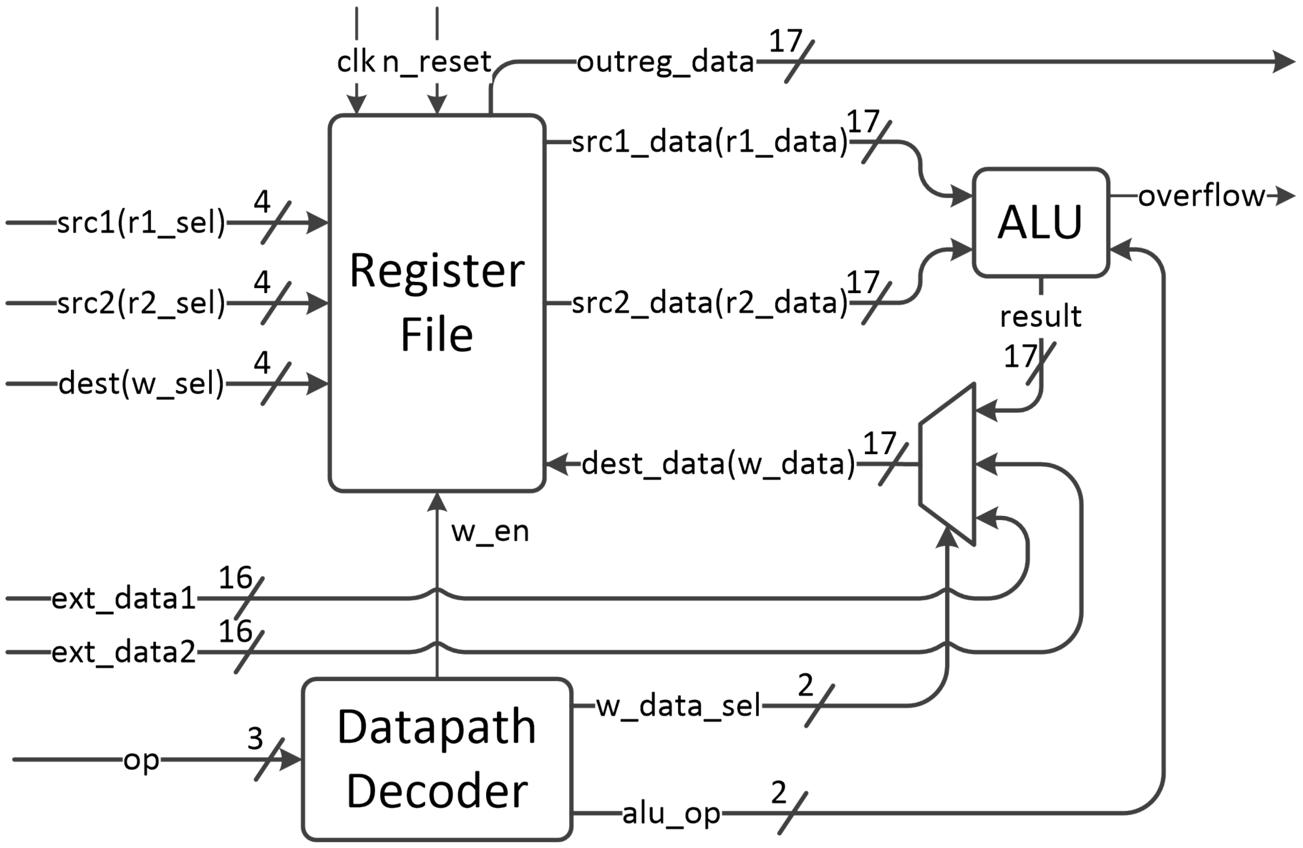

Figure 6: Datapath Architecture Diagram

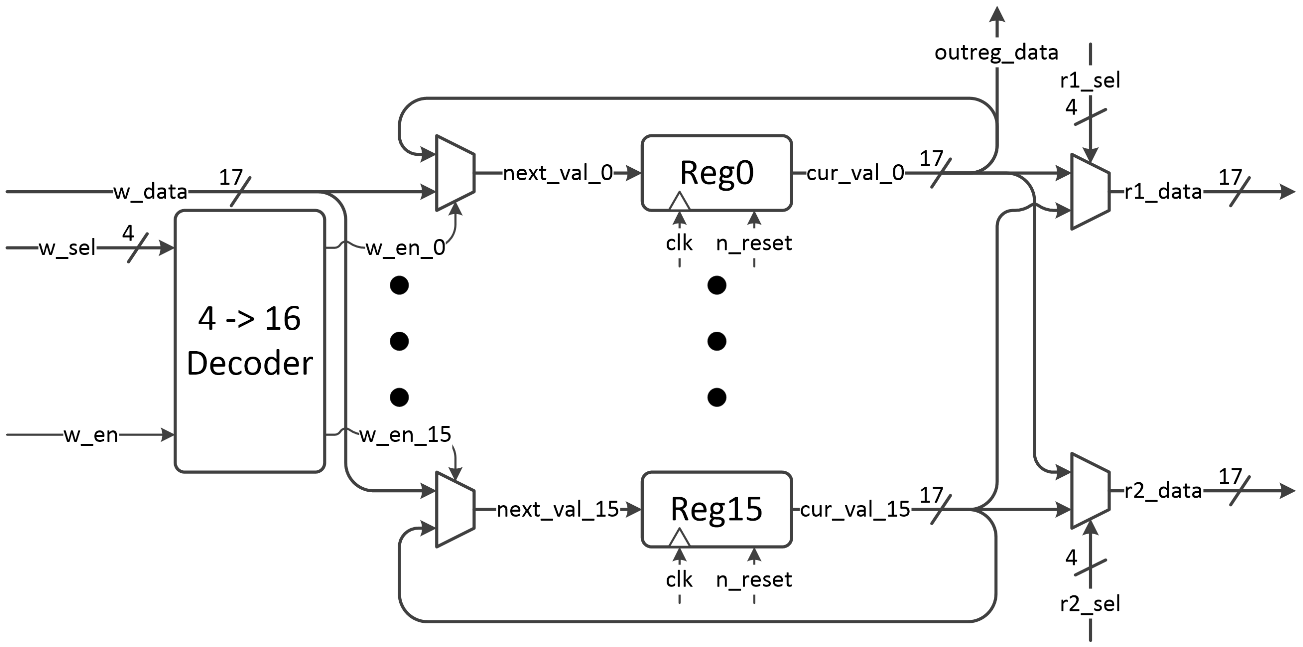

Figure 7: Register File Architecture Diagram

6.2 Module Specifications

Required Module Name: datapath

Required Filename: datapath.sv

Required Ports:

Port name	Direction	Description
clk	input	The system clock. (Maximum Operating Frequency: 100 MHz)
n_reset	input	This is an asynchronous, active-low system reset. When this line is asserted (logic ‘0’), all registers/flip-flops in the device must reset to their initial value.
op[2:0]	input	The operation to perform: 000 – NOP 001 – COPY: Copy from register ’src1’ to register ’dest’ 010 – LOAD1: Store ’ext_data1’ in register ’dest’. 011 – LOAD2: Store ’ext_data2’ in register ’dest’. 100 – ADD: register ’dest’ = register ’src1’ + register ’src2’ 101 – SUB: register ’dest’ = register ’src1’ – register ’src2’ 110 – MUL: register ’dest’ = register ’src1’ * register ’src2’
src1[3:0]	input	Source Operand #1 (register address/number)
src21[3:0]	input	Source Operand #2 (register address/number)
dest[3:0]	input	Destination Operand #3 (register address/number)
ext_data1[15:0]	input	Data to be loaded into a register (see op).
ext_data2[15:0]	input	Data to be loaded into a register (see op).
outreg_data[16:0]	output	Value stored in the output register (register 0).
overflow	output	Set if the current operation produces an overflow. (Generated asynchronously and valid when the op is ADD, SUB, or MUL.)

7 Post-Lab Work

7.1 Design Questions

Place answers to the following questions in a file called ‘Lab7.txt’ inside your ‘docs’ directory.

Question: What is the minimum amount of time that data_ready must remain asserted to ensure correct operation? What is the minimum amount of time, in clock cycles, that data must remain valid after data_ready is asserted in order to ensure correct operation? (You may assume that all setup and hold times, as well as any propagation delays, are negligible.)

7.2 Verilog File IO Demo

Part of what the "setup7 " script did was copy over a customized makefile, customized test bench file, 24-bit bitmap image, and a waves formatting ’.do’ script. These files were copied over in order to support this quick demo on how to use the file IO syntax in Verilog in test bench designs. For this, the provided test bench uses Verilog file IO syntax to open the bitmap image file, extract the file header information, feed the image data through the filter design you completed in this lab, and then store the filtered image data and appropriate file header in a new bitmap file. The test bench is heavily commented and you are expected to look through it and try to use it to help you understand how to utilize the file IO syntax in Verilog to work with files for testing purposes. Upon analyzing the test bench file (tb_fir_filter_image.sv) you should notice that the file IO syntax in Verilog is very similar to file IO in C. At this point in the course you are not expected to fully understand how to use Verilog File IO, however it is often very useful during testing project designs later in this course so it is in your best interest to look over the code provided in the test bench, conduct online searches about Verilog File IO and try to create some simple test benches with that use file IO to feed values to your design from this lab, as well as ask the course staff questions you have about using Verilog File IO. To run the file IO demo, run the following command in your Lab7 directory:

make sim_image

After running the simulation has finished open both the "test_1.bmp" and "filtered_1.bmp" files in image viewers and compare them. Place answers to the following questions in a file called ’Lab5.txt’ inside your ‘docs’ directory.

Question: How are the image files different? Does this make sense given the filter design built in the lab? Why or why not?

Question: What is the general syntax for each of the file IO functions used in the provided test bench (tb_fir_filter_image.sv)?

Question: What are the different format specifiers available for use in file functions like $fscanf(. . .)?

Once you have answered all of the post-lab questions and have finished the team report, run the following command on terminal window to submit your post-lab:

submit Lab7post

8 Closing Remarks

- Turn in your check-off sheet at the beginning of Lab Lab 8 .