This is the write-up for ECE574, Test #2 – Filter Processor Design.

***Test 2: PART 1***

In order to successfully implement a processor which does FIR filtering, the following sets of instructions need to be performed. They are written in a generic pseudo-code fashion using the SAYEH addressing instruction format. This system will ultimately cater to an FIR filter of size **Filt\_Length**.

**mil** <initial samples> **-- instantiate original memory locations**

**lda** *<degree of filter>* **-- read in Filt\_Length**

**…**

**start:**

**lda** *<c0 >* **-- read in filter coefficients into 1 side of each register**

**add** <next memory index> **-- increment memory index for next filter read operation**

**… -- perform for ALL coefficients [Filt\_Length + 1]**

**lda** <next input data sample> **-- load next data sample from memory**

**mul** <coeff = coeff \* data sample> **-- multiply coefficient by data sample, and store in register**

**… -- perform for ALL data samples [Filt\_Length + 1]**

**add** <reg = reg + next reg> **-- accumulate by adding all multiplied data samples**

**… -- perform [Filt\_Length] times, with final result in last register**

**mil** <write address>

**mih** <write addess> **-- declare the address where the output sample will be written**

**sta** <final register> **-- store the final filter result at specified address in memory**

**jmp start;**  **-- repeat process for next train of [Filt\_Length + 1] samples**

These sets of instructions, when looped to run across all sets of *Filt\_Length + 1* sample trains across the collected sample space, will produce the correct number of filter outputs for the system.

This concludes the analysis for Test 2, Part 1.

***Test 2: PART 1***

In order to implement an Nth order FIR filter, a modified version of the SAYEH (Simple Architecture, Yet Enough Hardware) processor is used. Here are highlights of the primary changes:

* Simplification of ALU to only utilize ADD and MULTIPLY operations.
* Elimination of WindowPointer and need for Shadowing check.
* Instruction will be simply decoded within the Register of the SAYEH processor.
* Eliminated unnecessary signals and busses, assuming that instructions will correctly and sequentially apply filtering operations to all sample trains.

A picture of the modified SAYEH DataPath is seen below:



This concludes the analysis for Test 2, Part 2.

***Test 1: PART 3***

The Verilog code for this section can be found in the Pt 3 folder of the ZIP file:

*<Verilog Files>*

This concludes the analysis for Test 2, Part 3.

***Test 1: PART 4***

The following is a list of all ***control signals*** that will be used in this to create an FIR filter implementation using the SAYEH Processor, as depicted in the DataPath System Diagram. A brief description is also listed:

*Address\_on\_Databus*

* Tri-state buffer signal, dictating whether a signal will be placed on the Databus or not.
* This is used to load a new instruction into the InstructionRegister.

*ALUout\_on\_Databus*

* Tri-state buffer signal, dictating whether a signal will be placed on the Databus or not.
* This is used to either load a new value back into a Register, or to write information back to the Memory block when an output is done computing.

*AaddB, AmulB*

* Used to control the two ALU functions to be used in the FIR filter.

*ResetPC, PCplusI, PCplus1, R0plusI, R0plus0*

* Used to control the different address decoding and instructions based on SAYEH format.

*IRload*

* Asserted when a new instruction is to be loaded into the instruction register (after Address Logic places the correct, decoded message on the Databus).

*ReadMem, WriteMem, MemDataReady*

* All control signals that indicate whether or not we are reading or writing into the memory.

This concludes the analysis for Test 2, Part 4.