For:reese
Printed on:Mon, Mar 1, 1999 08:27:03

Document:Lab4
Last saved on:Tue, Aug 4, 1998 16:25:29

EE 4743 Lab #4 — Six–Tap FIR filter

A six–tap Finite Impulse Response Digital Filter is of the form:

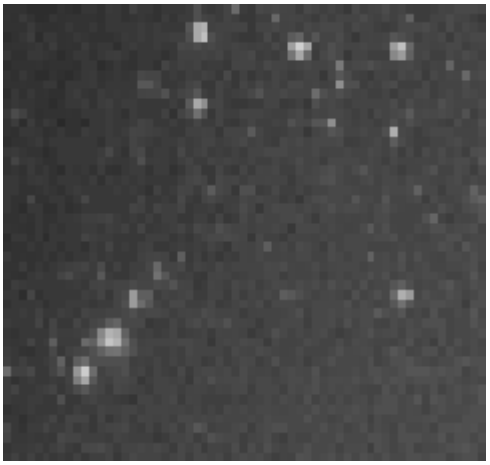
$$y = x^* a0 + x@1 * a1 + x@2 * a2 + x@3 * a3 + x@4 * a4 + x@5 * a5$$

where a0 – a5 are the filter coefficients. 'x' is the current input sample, 'x@1' is the input sample from the previous sample period, x@2 is the input sample from two sample periods back, and so on. One high pass filter has coefficients of:

$$y = | (0.25x + 0.25x@1 + 0.25x@4 + 0.25x@5) – (0.5x@2 + 0.5x@3) |$$

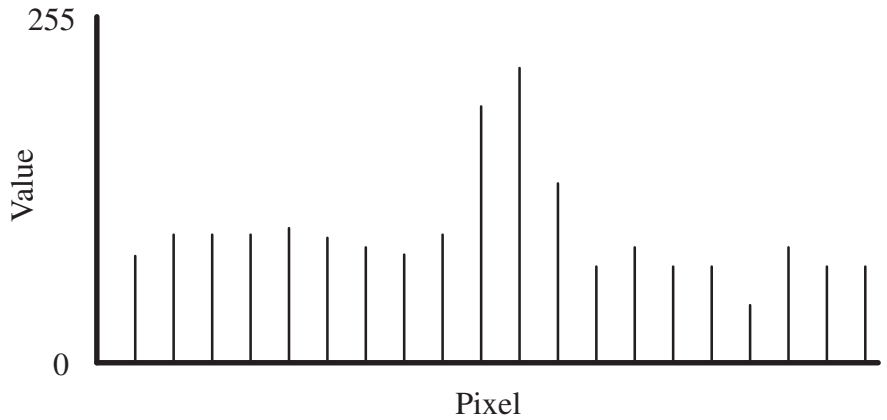
The filter subtracts the average of two pixels from the average of the four neighboring pixels. Where would a filter like this be used? One example is to find 2–pixel–wide bright spots in a digital image. Consider a digitized image.

Image

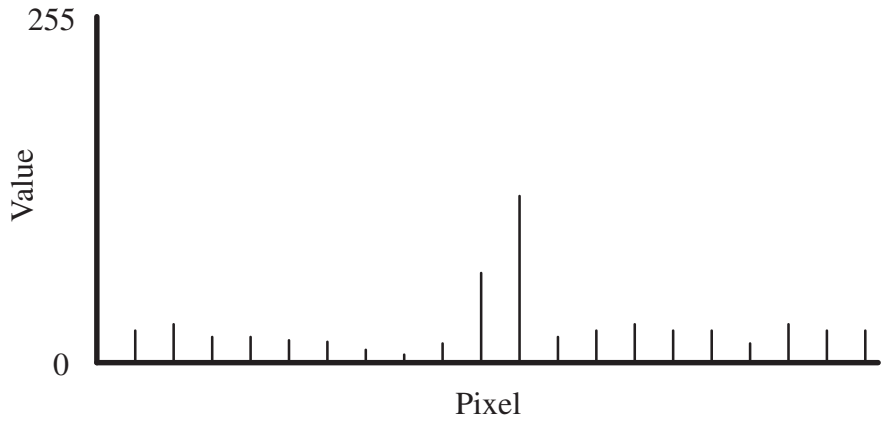


Zoomed in to area around some stars

If we ''zoom in'' enough, we can see the individual pixels of the image. A dark pixel corresponds to a low number, and a bright pixel to a high number. If we graphed the pixel values across a row of the image, it might look like this. The ''notch'' in the graph corresponds to the star against a dark background. The high areas are the background haze. (Notice that we use unsigned numbers, and therefore unsigned arithmetic, in image processing.)



After applying the high pass filter, which subtracts the average of two pixels from the average of the pixels around it, the filtered version would look like this. Notice that now two–pixel–wide bright spots are highlighted, but not diffuse blobs of light (like background haze).



The filtered image looks like this. Notice how stars are highlighted, and the background haze is mostly gone.

Filtered Image

Now, back to the filter...

$$y = | (0.25x + 0.25x@1 + 0.25x@4 + 0.25x@5) - (0.5x@2 + 0.5x@3) |$$

To save area, we will expand the constant multiplications into adds/sub/shifts; this produces:

$$y = | (x + x@1 + x@4 + x@5)>>2 - (x@2 + x@3)>>1 |$$

The minimum number of datapath components needed to implement this design is 1 adder, 1 "absolute value subtractor", and 1 generic shift unit. A possible schedule for implementing this calculation is:

CC 1:	input sample value, advance sample pipeline, clear SUM1 and SUM2;
CC 2:	SUM1 = SUM1 + x
CC 3:	SUM1 = SUM1 + x@1
CC 4:	SUM1 = SUM1 + x@4
CC 5:	SUM1 = SUM1 + x@5
CC 6:	SUM2 = SUM2 + x@2
CC 7:	SUM2 = SUM2 + x@3
CC 8:	SUM1 = SUM1 >> 2
CC 9:	SUM2 = SUM2 >> 1
CC 10:	y = | SUM2 – SUM1 |, assert DATA_RDY

By using 1 shifter unit, 2 adder/subtractors, and 8 registers, we can decrease the number of clock cycles via the following schedule:

CC 1:	input sample value, advance pipeline, T1=x@4+x@5, T2=x@2+x@3;
CC 2:	T3 = x + x@1, T2 = T2 >> 1;
CC 3:	T1 = T1 + T3;
CC 4:	T3 = T1 >> 2;
CC 5:	y = | T3 – T2 |, assert DATA_RDY

Figure 1 shows a datapath for the five clock cycle schedule. (It is not as bad as it looks!) Figure 2 shows an ASM chart for the needed control. Implement the datapath and control; use the command file in ~russ/EE4743/Lab4 to test your design and compare against the lab4 ".GOLDEN" file. You must use VHDL to specify the control for the datapath; you can use any state encoding method you wish.

Interface:

Inputs:	x[7:0]	sample input
	START	input sample ready
	CLK	system clock
	RESET	global reset (only used once to zero the pipeline)
Outputs:	DATA_RDY	output data ready
	y[7:0]	data output

Use the Xilinx X4000 library. **A schedule of when things are due is below.**

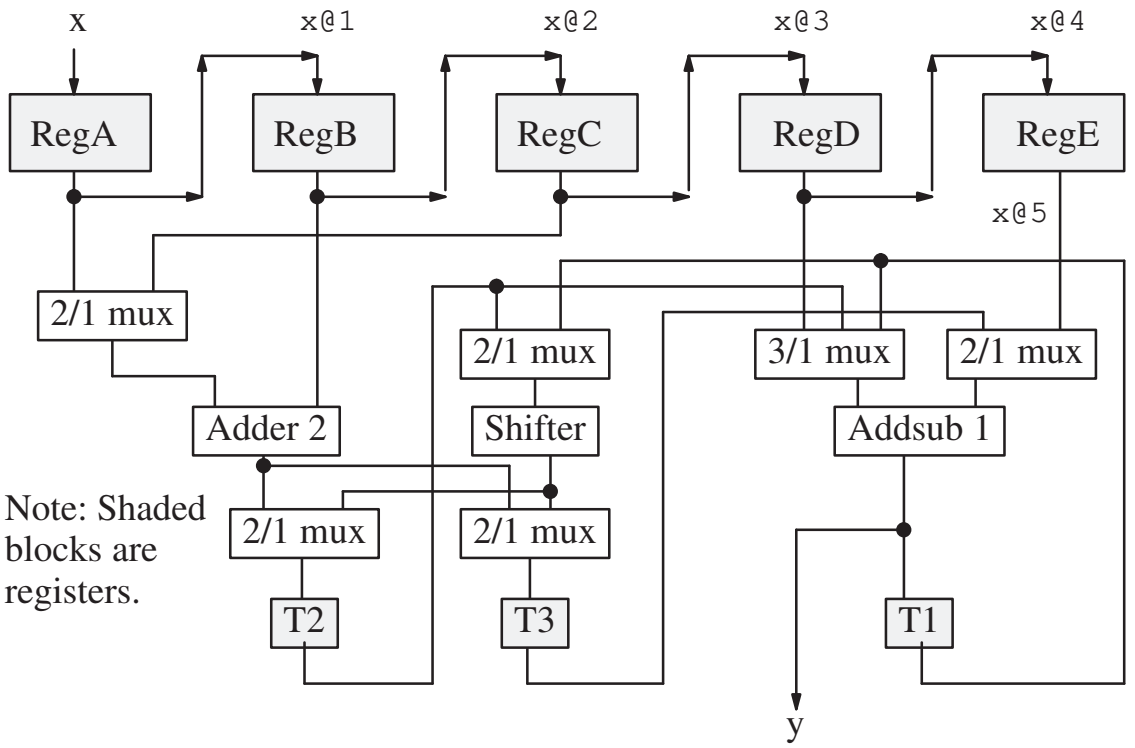


Figure 1: Filter Datapath

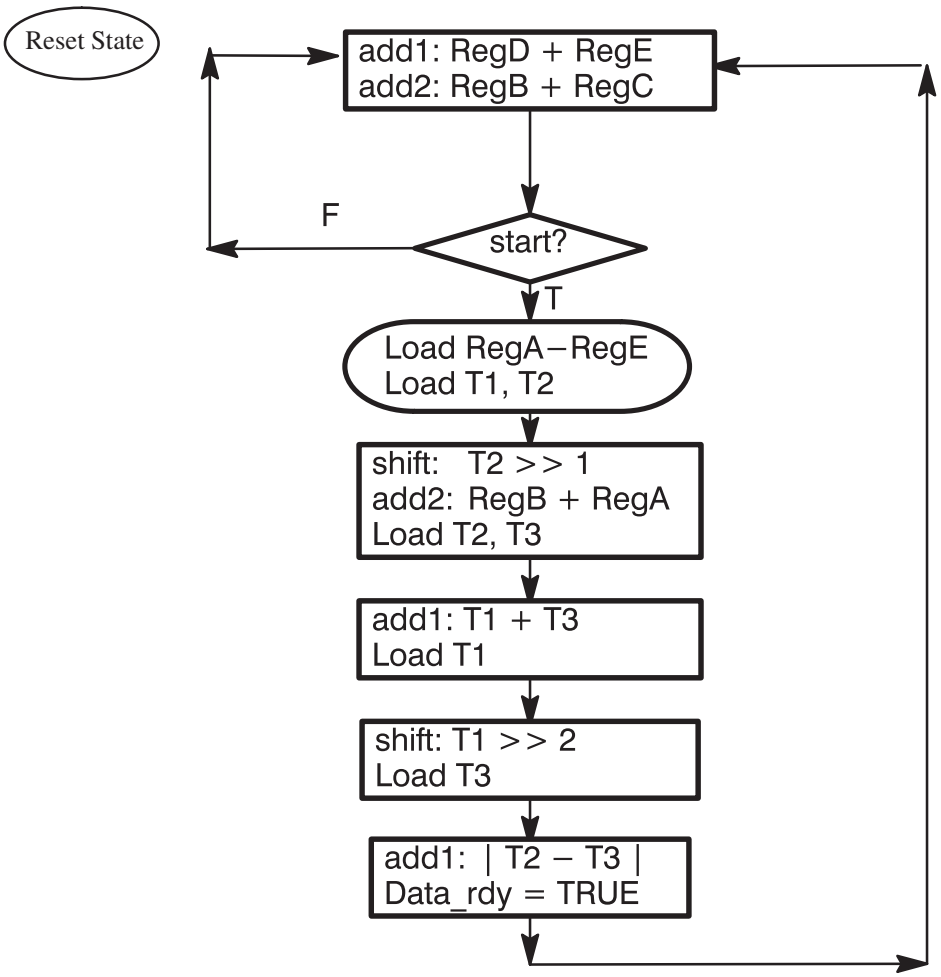


Figure 2: Filter ASM Chart

Part 1: Datapath
Implement the datapath section drawn above. Document carefully which control signals are used and what values are needed on each signal to produce the desired result.

Part 2: Control
Implement the controller for the datapath. Use the simulation files mentioned above to check your results.

A few extra notes...

1. Since the numbers are UNSIGNED, do you want to use arithmetic right shifts or logical right shifts? Consider the number 192, which is C0 in hex. C0 >> 1 = 192 / 2 = 96 = 60 hex.
1100 0000 >> 1 = 0110 0000 in binary.
Is this an arithmetic shift or a logical shift?

2. It may be helpful to see some of your internal signals when you try to debug your lab. The easiest way to do this is to add it to the command file and therefore the waveform display.
Step 1: Click on the desired net in the schematic display. Right down the name that appears in the lower left corner, including all the gibberish. A typical name might be ”$1I184/$1I72/$1N473”.
Step 2: Add a line at the top of your file like this... ”vector dout $1I184/$1I72/$1N473”.
Step 3: Add “dout” (or whatever you just called it) to the end of your wave list.

3. On clock cycle 1 (“Reset State”), all input values advance to the next register. For example, x@1 is in register A and advances to register B. Where is x@2 at the BEGINNING of clock cycle 1? Therefore, which “mux select” should you use for the mux that selects the input to the shifters?

4. The reset state contains both unconditional and conditional outputs. Why are some conditional and some unconditional? What is the difference between the two? Hint: Think about the timing differences.