CpE318 Test 2 – Take-home portion (50 Points)

Take home is to be done on an individual basis. Please do not discuss this test with anyone besides your professor and possibly the TAs in any way until after the due date.

Application (not required to solve problem)

Generating high-quality real-time ultrasound images requires an extraordinary number of calculations be done in a very short period of time. The time it takes for a sound wave to travel from a transducer to a point in the image must be calculated for every transducer and for every pixel in the image. This time is called the time-of-flight (TOF). As illustrated in Figure 1, if the speed of sound in the medium is c and the horizontal and vertical offsets of the pixel and transducer are given by X and Z, then the straight-line distance from the transducer to the pixel is given by

$$\text{distance} = \text{SQRT}(X^2+Z^2)$$

and

$$\text{TOF} = \text{distance}/c = \text{SQRT}(X^2+Z^2)/c.$$

Calculating a square-root of a sum-of-squares, however, takes many clock cycles. As a result, one cannot generate a high-quality real-time image if this calculation is performed for every pixel in the image[1]. An approximation is required. One method is to approximate TOF over the region of interest (over the "image") using a polynomial, and then calculate the polynomial using its difference form. Difference equations are discussed below.

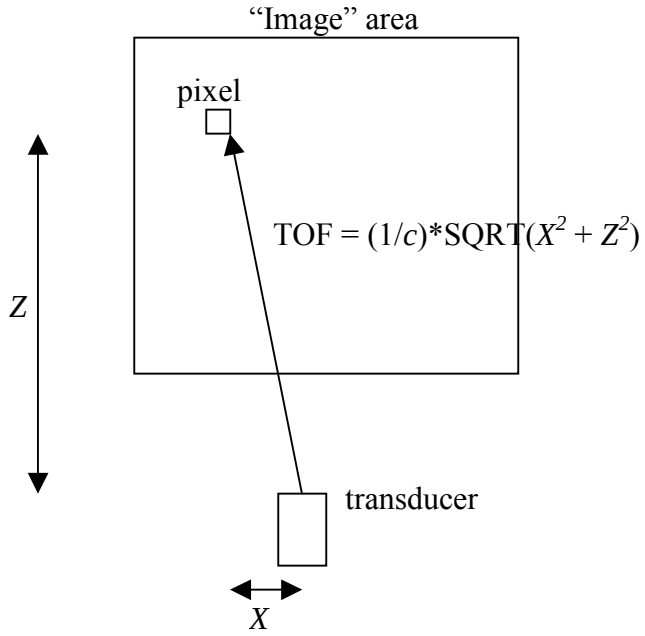

Figure 1: Calculation of time-of-flight from an ultrasound transducer to a pixel in the "image" space (region of interest).

[1] An image must be generated every 1/30 of a second. Calculating a square root of a sum of squares may take 30 or more clock cycles. A single 2D image may require more than 2.7e+08 calculations of time-of-flight - many more if generating 3D images.

Difference equation

A polynomial is an equation in the form:

$$f(x) = \sum a_i x^i$$

where the summation is performed from i=1 to n, and n is the degree of the polynomial. For example, $f(x) = a_0 + a_1x + a_2x^2$ is a second degree polynomial.

Say we know the value of f(x) and would like to calculate the value of f(x+δ). We can do so using the difference equation. The first difference in f(x) is given by:

$$d_1(x) = f(x+\delta) - f(x) = \delta a_1 + a_2\delta(2x+\delta)$$

The second difference is given by:

$$d_2(x) = d_1(x+\delta) - d_1(x) = 2\delta^2 a_2$$

The third difference is given by:

$$d_3(x) = d_2(x+\delta) - d_2(x) = 0.$$

The nth difference will always equal 0 for an nth degree polynomial.

If we know f(x), then we can find f(x+δ) as:

$$f(x+\delta) = f(x) + d_1(x)$$

We can then find f(x+2δ) as:

$$f(x+2\delta) = f(x+\delta) + d_1(x+\delta)$$

where $d_1(x+\delta)$ can be calculated from its value at x as:

$$d_1(x+\delta) = d_1(x) + d_2(x)$$

Using this idea, we can calculate a succession of values of f(x) at x, x+δ, x+2δ, and so on, where each new value is calculated for the last value of f(x) and its differences. For example, if we wanted to know f(x) at (x+δ, x+2δ,… x+nδ) and knew f(x), $d_1(x)$, $d_2(x)$, then the value of f, d_1, and d_2 at each location would be:

	x+δ	x+2δ	x+3δ	…	x+nδ
f	$f(x) + d_1(x)$	$f(x+\delta) + d_1(x+\delta)$	$f(x+2\delta) + d_1(x+2\delta)$	…	$f(x+n\delta) + d_1(x+n\delta)$
d_1	$d_1(x) + d_2(x)$	$d_1(x+\delta) + d_2(x+\delta)$	$d_1(x+2\delta) + d_2(x+2\delta)$	…	$d_1(x+n\delta) + d_2(x+n\delta)$
d_2	$d_2(x) + d_3(x) = d_2(x)$	$d_2(x+\delta) + d_3(x+\delta) = d_2(x)$	$d_2(x+2\delta) + d_3(x+2\delta) = d_2(x)$	…	$d_2(x+n\delta) + d_3(x+n\delta) = d_2(x)$
d_3	0	0	0	…	0

That is, for each new value of x, we can simply add differences together to get the new result. The same form applies to much higher-order polynomial equations. If done in hardware with appropriate pipelining, each new value of f(x) can be calculated in the time of a single addition.

Requirements

Write a *behavioral* model which will generate values of a 2nd degree polynomial, f(x), in the time of a single addition using the difference equation. The interface for your model is:

```
entity poly_calc is
      port(load: in bit; data:in bit_vector(15 downto 0);
```

```
            clk: in bit; result:out bit_vector(19 downto 0));
end entity poly_calc;
```

The values of f(x), $d_1(x)$, and $d_2(x)$, are loaded into the model, poly_calc, one at a time. They are loaded by setting load high, placing the appropriate value on the data bus, and toggling the clock. On the first low-to-high transition of the clock, f(x) is loaded. On the second clock cycle, $d_1(x)$ is loaded (load is still high), and on the third clock cycle, $d_2(x)$ is loaded. Load must be high the entire time. If load goes low, then you will have to start the process over again. When load is low, a new result is generated every time the clock transitions from low to high (i.e. after the first clock, result = $f(x+\delta)$, after the second, result = $f(x+2\delta)$, and so on). f(x), $d_1(x)$, and $d_2(x)$ are 16-bit binary fixed-point numbers ranging from –8.0 inclusive to +8 exclusive. Signal result is a 20-bit binary fixed-point number ranging from –255 to +255. You are free to make internal registers as large as you like.

Create a *behavioral* model of this polynomial calculator. It should have the same general structure as the intended hardware, but unlike a structural model can use signals of type real inside the model instead of bit-vectors. Comment your code well, so I can tell what you did. Use assert and report statements to give feedback to the user when something goes wrong. Use procedures or functions whenever appropriate to save the amount of code you're writing or to make your code more readable and testable. Write a **VHDL** testbench which thoroughly tests your code. Your testbench should automatically test results (i.e. a human isn't needed to stimulate the model or determine that results are correct). For the testbench, you might test your model's ability to calculate $e^{-x} \approx 1 - x + x^2/2$ for values of x=(0,0.1,0.2,0.3,…). For this approximation, f(0)=1, $d_1(0)$=-0.095, $d_2(0)$=0.01. Feel free to test other polynomials too (I will). To promote testing, create a second entity which contains your polynomial calculator but will accept floating point numbers:

```
entity poly_calc_test is
      port(load: in bit; data:in real;
      clk: in bit; result:out real);
end entity poly_calc;
```

I will be running my testbench *primarily* on this model.

FYI: Error tends to build up in this type of calculator as it is run. The answer after 100 clock cycles will not be as accurate as after 10 cycles if difference coefficients cannot be represented precisely by the hardware (given # of bits).

Turn in

Hardcopy:

a) A (structural) drawing of your model
b) A brief explanation of how your model works (and how it should be tested, if necessary).
c) A copy of your VHDL code (model and testbench – testbench must be VHDL)
d) A copy of your simulation results. Explain why you feel your simulation shows your model works.

Electronically (as an attachment sent with MS Outlook) turn in a project directory including:

e) The code for your model, poly_calc
f) The code for your test model, poly_calc_test
g) The code for your testbench
h) Any special instructions necessary to use your model (in a README.txt file).

Please put your identifier in any code-file you turn in, so I can easily tell whose it is. Your real name must not occur anywhere within these files. Please also name your tar.zip file with your identifier. For example, if your code is in a directory called "takehome" and your identifier is "superman", then tar and zip your file using the Unix command:

tar -cf - takehome | gzip -9 > superman.tar.gz

Please CC yourself when you send this email and double check that the file you sent is OK because I will not accept a re-submission after the due date, even if the file is corrupt. To untar/unzip use: "cat superman.tar.gz | gunzip | tar –xf –".

Extra credit (10%)

Extend your model to the 2D case, where $f(x,z) = \sum\sum a_{ij}x^i z^j$ is a second degree polynomial in both x and z. Test your model for the case δ=0.12 and the polynomial f(x,z) approximates the function $TOF(x,z) = 1/1500000\ SQRT((x-15)^2 + (z+50)^2)$. Turn in all of the above for this case.