CMPE 491/691 – Homework/Project #4

Fall 2014

Work individually, but I recommend working with someone in the class nearby so you can help each other when you get stuck, with consideration to the **course collaboration policy (please read it in the course website)**. Please send me email if something isn't clear and I will update the assignment. Changes are logged at the bottom of this page.

Before getting started, you should go through the verilog notes located under Course Readings on the course home page.

A paper copy of everything and electronic copies of all your code and testing files (all in one zipped file) are due at the beginning of class on the due date.

Notes:

- [15% of points] Clearly state whether your design is fully functional, and state the failing sections if any exist.
- Make sure your design and code are easily readable and understandable (clear and well commented).
    - Up to 5% extra credit will be given for especially thorough, well-documented, or insightful solutions.
- *** Where three '*'s appear in the homework, perform the required test(s) and turn in a printout of either:
    1. a table printed by your verilog testbench module listing all inputs and corresponding outputs,
    2. an Isim waveform plot which clearly shows (labeled and highlighted) corresponding inputs and outputs, or
    3. a section of testing code which **clearly** compares the designed circuit and a simple reference circuit, and two short cut & paste sections of text from your simulation (one for pass, and one for fail where you purposely make a slight change to your reference code to make it fail) that look something like this:
    `Error: input=0101, out_module=11110000, out_ref=11110001`

    In all three options, each test case must be marked whether the output is correct or not.

- Keep "hardware" modules separate from testing code. Instantiate a copy of your processing module(s) in your testing module (the highest level module) and drive the inputs and check the outputs from there.

This problem consists of the design, implementation, and synthesis of a small-area complex fixed-point 16-point Decimation-In-Time (DIT) Fast Fourier Transform processor. The main datapath supports 16-bit fixed point for both real and imaginary components. Register all inputs

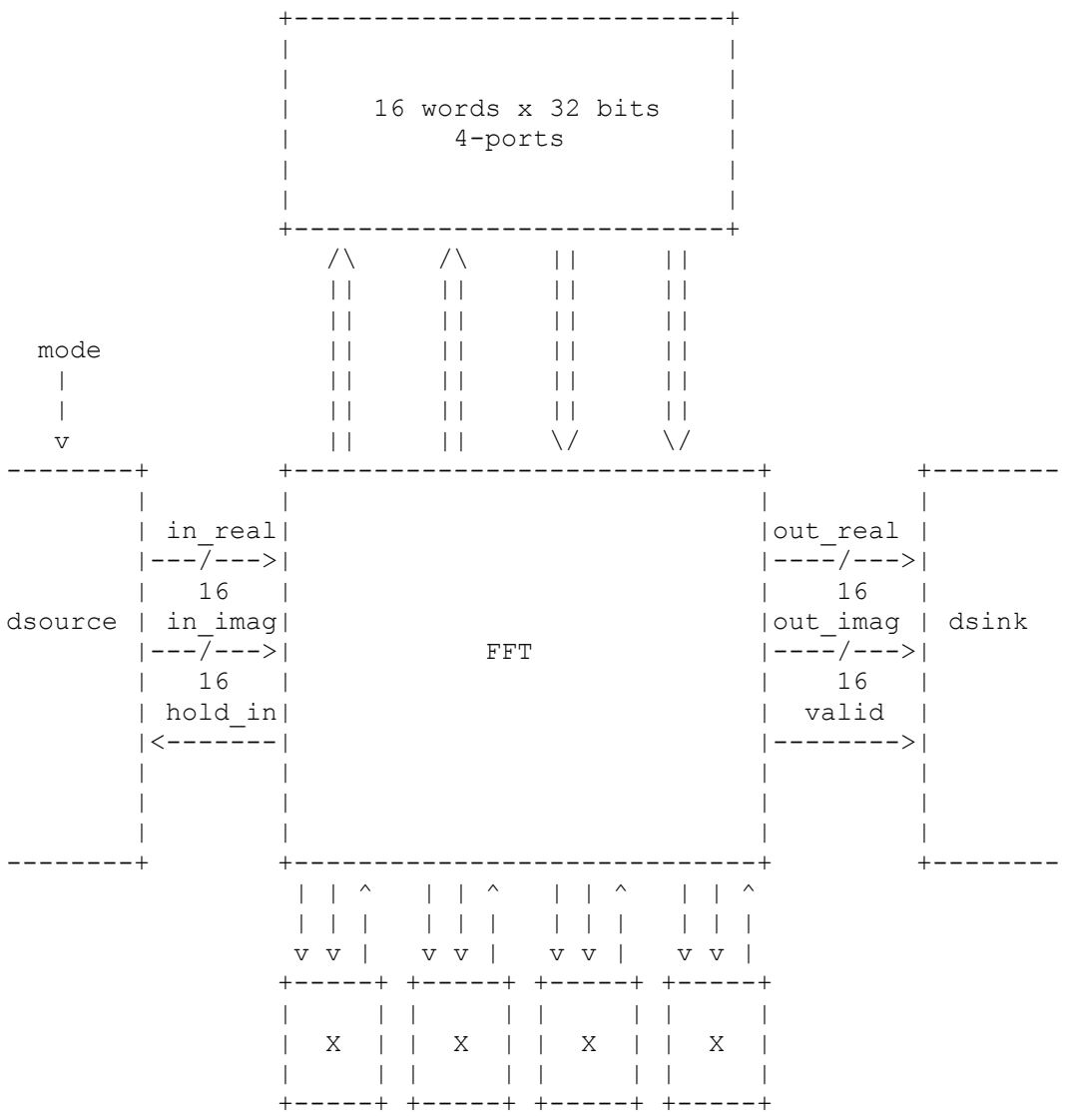
before using them and register all outputs before they exit the FFT block. Use a single 250 MHz (4 ns) clock for all circuits.

Design the processor with a single radix-2 butterfly. Pipeline it so that it runs at 250 MHz--you should not need many pipe stages to meet timing. Note that the multipliers contain two pipeline registers and the memory contains one. Generate W_N coefficients with hardware lookup tables whose verilog is produced by a matlab program. Perform the complex multiplication using four real multipliers.

Perform signal scaling by 1) dividing inputs by 2, and 2) scaling in butterflies by 1/2 so that outputs never overflow.

The key block signals are (others may be added as needed within reason):

- Fixed point number format. Each number (real or imag) is a 16-bit fixed point 2's complement number for both real and imaginary components. In some cases, it is convenient to consider numbers as 32-bit complex words.
- `reset`
  Synchronous reset, asserted high.
- `hold_in`
  Used to tell the upstream block when to advance data. If `hold_in` = 0 on the rising clock edge, `in_real` and `in_imag` will be advanced to the next sample. Otherwise, the sample will not change.
- `in_real[15:0], in_imag[15:0]`
  Data inputs, real and imag components.
- Four addresses used for memory
- Two data write busses used for memory
- Two data read busses used for memory
- `wren`
  "write enable". When high on a positive clock edge, data on `dataw*` is written to address `addrw*`. If low, data from location at address `addrr*` is placed on `datar*`.
- `out_real[15:0], out_imag[15:0]`
  Data outputs, real and imag components.
- `valid`
  Used to tell the downstream block when to grab valid output data. If `valid` = 1 on the rising clock edge, `out_real` and `out_imag` will be logged as valid FFT output data.
- `stall`
  Used to stall the FFT processor's operation so that its output can be efficiently merged with other blocks.



Verilog modules

Several verilog modules have been written to give you a running start. They are written with various sizes and word widths--so you must modify them. They are also not debugged and NOT in final form. Use them or don't use them. You can modify the modules as long as you do not change the latency through the combinational part of the block. For example, the multiplier logic must be allowed a full clock cycle to perform the multiply. The inputs to the multiplier and the outputs from the multiplier must be registered in your fft module.

dsource.v Generates various blocks of data for the FFT. Several different types of input sets are available selectable by a `mode` input port. Writes input data to the matlab-readable file datain.m

dsink.v Writes output data to the matlab-readable file dataout.m

mem_32x40.v A 32-word by 40-bit memory with 2 read ports and 2 write ports.

mult_20x20_carsav_noreg.v A 20-bit by 20-bit signed multiplier with carry-save outputs. Do not use other multiplier modules. Use all mult output bits before rounding.

fft.v Shell of FFT including a state machine and counter framework, and input hold control.

topfft.vt Top level test module, clock generator and a few other things.

Synthesis

Synthesize the design for:

- `clk_period = 4000`

Design goals

The overall goal of the project is to design a system with low area.

In order of importance, you should:

1. Design components and get verilog working
2. Get synthesis working
3. Meet clock cycle time without timing violations (negative slack)
4. Minimize area (meaningless without meeting timing)
5. Maximize accuracy compared to matlab fft()

Designing, Testing, and Grading [200 pts + 0-75 pts]

Hardware design

Design and write verilog for the FFT block. **Full credit cannot be given unless the FFT can compute the four cases in part (c). Caution: don't take documentation lightly; large numbers of points will be deducted for incomplete or unclear descriptions.**

a) [25 pts] Design and draw a block diagram of the FFT processor including datapath and control. Include pipeline stages and word widths in bits. There must be enough detail so that the exact *functional* operation of the block can be determined by someone with a reasonable knowledge of simple blocks and your diagram and explanation.

b) [25 pts] Draw a pipelined block diagram of the processor including memory, datapath, control, and I/O. This diagram is a simplified block diagram with the additional characteristic that all blocks and signals are aligned into the pipeline stages in which they execute.

c) [25 pts] Show that your FFT correctly calculates an FFT for the following cases by loading verilog results (already done for you in dsource.v and dsink.v) and comparing them in matlab with difff and turning in: 1) the `difff()` plot, and 2) the text output of `difff()`, such as s below.

```
>> difff(fft(in), out)
max(data0-data1) =  -0.000100 +0.000000i =  -1e-04 +0i
min(data0-data1) =  -0.000100 +0.000000i =  -0.0001 +0i
max(data0/data1) = -Inf -Infi
min(data0/data1) = +Inf +Infi
approx separated mean(data0/data1) = +0.984368+ +1.000000i
Energy_data0 = 174784.000000
Energy_diff  = 0.000021
Energy_diff/Energy_data0 = 0.000000 = -94.363217dB
>>
```

i) [25 pts] Positive real impulse at DC (mode=00 in dsource.v).
ii) [25 pts] Positive real impulse at freq=1 (mode=02 in dsource.v).
iii) [25 pts] Rect input (mode=06 in dsource.v).
iv) [25 pts] Random input (mode=07 in dsource.v).

d) [25 pts] Accuracy points for smallest error for random signal input (part c.iv) compared to matlab fft, in comparison to other working designs in the class. Write the Energy_diff/Energy_data0 value in dB from difff.m in your report.

The error of the butterfly's output should be on the order of an LSB or two. The best you can do is a little more than 1/2 LSB (partly due to W_N rounding). Three LSBs of error for just the butterfly is a lot. Debug your error with just the butterfly. Remember the signal grows through the FFT so errors across stages do not directly add. Getting the rounding exactly correct is surprisingly quite difficult. Getting it close is not hard. The matlab function `difff()` will give you a huge help in reducing error.

e) [10 pts] State how many clock cycles your FFT requires to complete the transform from the last sample in to the first sample out. Also give a simple breakdown of how those cycles are used.

f) [0-25 pts] Latency points for shorter time from last sample in to first sample out; in relation to other designs in the class. Must be fully functional, with working and tested matlab-accurate plots.

Synthesis

e) [40 pts] Synthesize the FFT block without errors or serious warnings. Turn in paper copies of the following. Print in a way that is clear and easy to understand but conserves paper (multiple files per page, 8 or 9 point font, multiple columns).

1. `Slice/summary report`
2. `Any warning/error report`
3. `Power report`
4. `Timing report`; first (longest) path only
5. source verilog files
6. test verilog files
7. source matlab files

f) [0-25 pts] Area points for smaller FFT area in relation to other designs in the class. Must be fully functional, with working and tested matlab-accurate plots and clean synthesis (no errors or serious warnings).

Possibly-helpful suggestions

- Work on a block at a time; for example, get things working in this order:
  - First generate a single butterfly output--the imaginary part of the X butterfly output would be a good one to start with.
  - Next get the whole butterfly processor working such that it matches matlab butterfly equations with adequate precision and random inputs.
  - Next pipeline and synthesize the datapath so it meets timing.
  - Now for the control: write the FFT entirely in matlab using the same equations you will use in verilog. The matlab functions bitget0.m and bitset0.m can come in very handy.
  - Now write verilog to generate all addresses and W_N ROM addresses; pipeline it, and you're almost done.
  - Plan for lots of debugging time!
- As a rule, register all inputs into your top level to give those signals a full clock cycle to work inside your block before they have to be registered again. This does not apply to the multiplier and memory interfaces.

FPGA test using UART

You need to implement the FFT on virtex 5 FPGA and test the design using UART. You can show the FFT functionality using your own input sequence or . The results of the computation must closely match with your simulation results.

What to return

Fully tested design with testbench and hardware demonstration.

- Verified design in simulation
    - First test UART with a testbench separately. Then test UART and FFT module with a complete testbench.

- Verified design in Hardware and demonstration

Use matlab and serial interface to send data from PC to FPGA ML505 board and receive data and show the image using matlab.

- Complete report with detail block diagrams and plots for your results.

For undergraduates

Undergraduates will design an 8-point FFT and all other requirements for implementation and return will stay the same.