Design of a Simple Pipeline (RTL Coding)

Objective

To understand and appreciate the improved readability provided by RTL coding (Register Transfer Language style of coding) compared to structural coding at block-level design.
To understand where to use Blocking assignments and where to use Non-Blocking assignments in RTL coding.
To pay attention to the fact that you need to produce an intermediate variable before you attempt to consume it.
To pay attention to the fact that combinational paths in a pipeline may go across stages.
To note that an intermediate signal (in the NSL combinational logic) generated in a clocked procedural block should not be accessed outside that procedural block either for viewing in waveform or for actual logic coding.

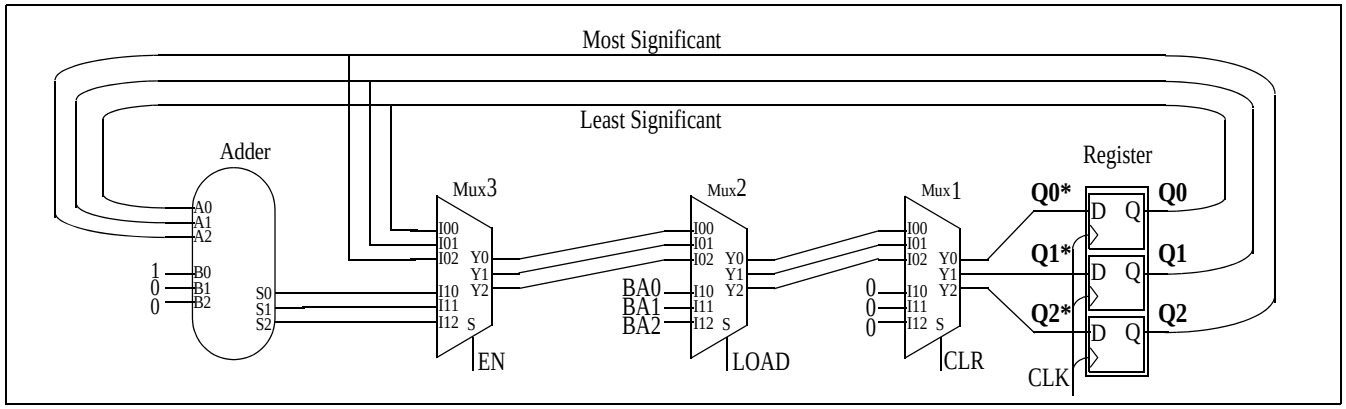
Introduction

This lab, Lab #7 Part #3 Subparts #3 and #4, is a continuation of the earlier Lab #7 Part #3 Subparts #1 and #2. Subparts #1 and #2 utilized the structural coding technique. Commonly structural coding is used at higher-level of design integration and not at block-level. In industry, a major design is first divided into 5 to 20 majors blocks and each block is assigned to an engineer for HDL coding. Individual blocks are usually coded in RTL. After proving the design of individual blocks, they are instantiated in the TOP level design file in the structural coding style. You will experience this in EE560.

There are some important issues in RTL coding which can easily be missed unless they are taught/demonstrated carefully. This lab intends to cover these issues.

RTL Coding

In EE201L, we used the coding of a simple synchronous counter with clear, load and enable controls to illustrate the difference between RTL coding style and structural coding style so far as readability and maintainability are concerned.



RTL Coding

```
always @(posedge clk)
    if (clr)
       Q <= 3'b000;
    else if (load)
       Q <= BA;
    else if (en)
       Q <= Q + 1;
```

Structural Coding

```
assign next_Q_after_en_mux   = en   ? (Q + 1) : Q;
assign next_Q_after_load_mux = load ? BA: next_Q_after_en_mux;
assign next_Q = clr ? 3'b000 : next_Q_after_load_mux;

always @(posedge clk)  Q <= next_Q;
```

The RTL coding and its intent can easily be understood even in the absence of the above schematic diagram, where as in the case of the structural coding, one has to mentally form the schematic diagram in order to figure out what the code is meant to do! And it is not easy to draw a schematic diagram in many cases. For example in a priority encoder, which combinationally selects the minimum of given 8 numbers, it is hard to draw a schematic diagram by hand. Basically, you do not want to design the circuit by hand. With RTL behavioral coding, you let the synthesis tool to synthesize an appropriate circuit for you.

© Copyright 2011 Gandhi Puvvada

The block diagram from Subpart #1 is reproduced on the page 4 of 7 for your immediate reference.

Now let us consider the coding of **EX2 stage together with the EX2/WB stage register**.

Structural coding (from ee457_lab7_P3.v)

```
assign FORW2 = EX2_XMEX1 & (EX2_ADD4 | EX2_MOV) & WB_WRITE;

assign EX2_ADDER_IN = FORW2 ? WB_RD : EX2_XD;

assign EX2_ADDER_OUT = EX2_ADDER_IN + (+4); // Add 4

assign SKIP2 = ~(EX2_ADD1 | EX2_ADD4);

assign EX2_XD_OUT = SKIP2 ? EX2_ADDER_IN : EX2_ADDER_OUT;

assign EX2_WRITE = EX2_MOV | EX2_SUB3 | EX2_ADD4 | EX2_ADD1;

pipe_reg2 EX2_WB(a lengthy instantiation port mapping which is not quite readable);
```

RTL coding (from ee457_lab7_P3_RTL_Coding_Style.v)

```
FORW2 = EX2_XMEX1 & (EX2_ADD4 | EX2_MOV) & WB_WRITE;

if (FORW2)
   EX2_ADDER_IN = WB_RD;
else
   EX2_ADDER_IN = EX2_XD;

EX2_ADDER_OUT = EX2_ADDER_IN + (+4); // Add 4

SKIP2 = ~(EX2_ADD1 | EX2_ADD4);

if (SKIP2)
   EX2_XD_OUT = EX2_ADDER_IN;
else
   EX2_XD_OUT = EX2_ADDER_OUT;
```

FORW2 is generated *before* utilizing

SKIP2 is generated *before* utilizing

Blocking assignments in the stage combinational logic

All variables in RHS expressions are either primary inputs or register outputs or intermediate variables. If they are intermediate variables, they should be produced, before they are referred (read).

```
// EX2_WB stage register
WB_RD <= EX2_XD_OUT;
WB_RA <= EX2_RA;
WB_WRITE <= EX2_MOV | EX2_SUB3 | EX2_ADD4 | EX2_ADD1;
```

Non-Blocking assignments in the stage register

General plan for coding a clocked (synchronous) system in RTL

Very rarely a combinational logic is either stand-alone or without a downstream register. In such a case, we write a combinational procedural block with the inputs to the combinational logic listed in the event list of the "always" block. In other cases, where a combinational logic has a downstream register, the combinational logic is treated as a NSL (Next State Logic) to the SM (state memory = stage register) and is described in a clocked always block. In either case, intermediate signals are assigned using blocking assignment so that they can be generated and immediately be consumed.

Since all registers are assigned using non-blocking assignments, the order of coding these non-blocking assignments is unimportant except for readability. You can potentially consider dragging all stage registers to the right-side of the diagram (or to the bottom of the clocked block) and pushing the stage combinational logic to the left side of the block diagram (to the upstream of the coding for those registers). The combinational logic can be arranged in "levels" of logic so that you code the 1st level followed by the next, and

© Copyright 2011 Gandhi Puvvada

so on. The above is actually not desirable as the code becomes highly unreadable. So we tend to code combinational logic of a pipeline stage (for example EX1 stage logic) followed by the stage register (EX1/EX2 stage register), and so on. Going along the above suggestion, we may be coding in a clocked always block, (a) EX1 stage combinational logic (using blocking assignments), (b) EX1/EX2 stage register (using non-blocking assignments), (c) EX2 stage combinational logic (using blocking assignments), (d) EX2/WB stage register (using non-blocking assignments), and so on. Here, we made an important mistake. The X2_Mux output is used not only in EX2 stage but also in EX1 stage. Forwarding help into EX1 comes from the EX2 stage X2_Mux output. It means, we consumed EX2_ADDER_IN in EX1 priority mux before EX2_ADDER_IN is produced (generated) in EX2. Such mistakes are difficult to debug as they are not syntax errors. So what is the recommendation so far as the order of coding? Code the FORW2 and the X2_mux (of EX2) (or the entire EX2) before coding the EX1 logic. Generally, since a later stage helps an earlier stage in a pipeline, you may choose to code later stages first and earlier stages next.

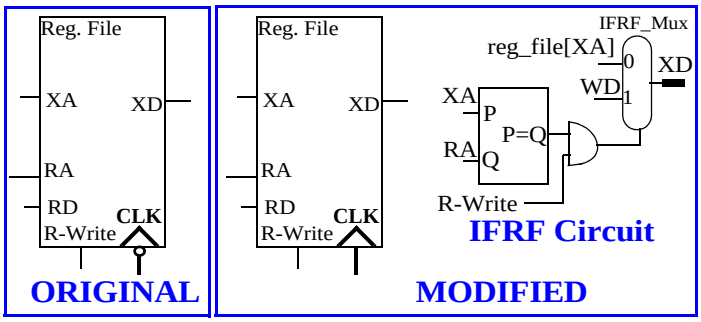
The Subpart #3 assignment

The completed RTL code, `ee457_lab7_P3_RTL_Coding_Style.v`, coding the design of Subpart #1 (shown in the block diagram on the next page) is provided to you, fully commented explaining the RTL coding issues. Please go through it completely. Simulate the design using the provided testbench and wave.do files.

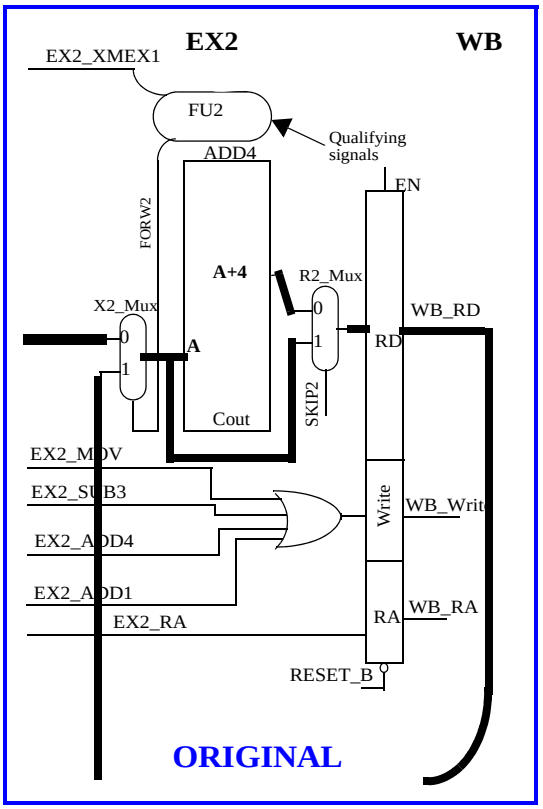
Now modify the RTL code (call it `ee457_lab7_P3_RTL_Coding_Style_R2_Mux_in_WB.v`) to support the following two changes to the design:

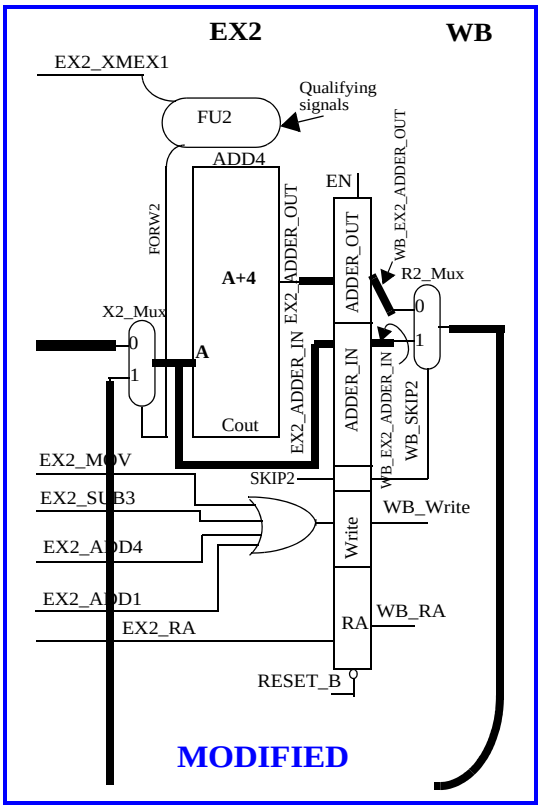
1. Here the **register file** is changed from being **negative**-edge sensitive to being **positive**-edge sensitive. Hence, you need to code the internal forwarding mechanism in the register file.

2. The **R2-Mux** is **moved** from the **EX2** stage to the **WB** stage. So mux inputs and the select line are to be carried to the WB stage.

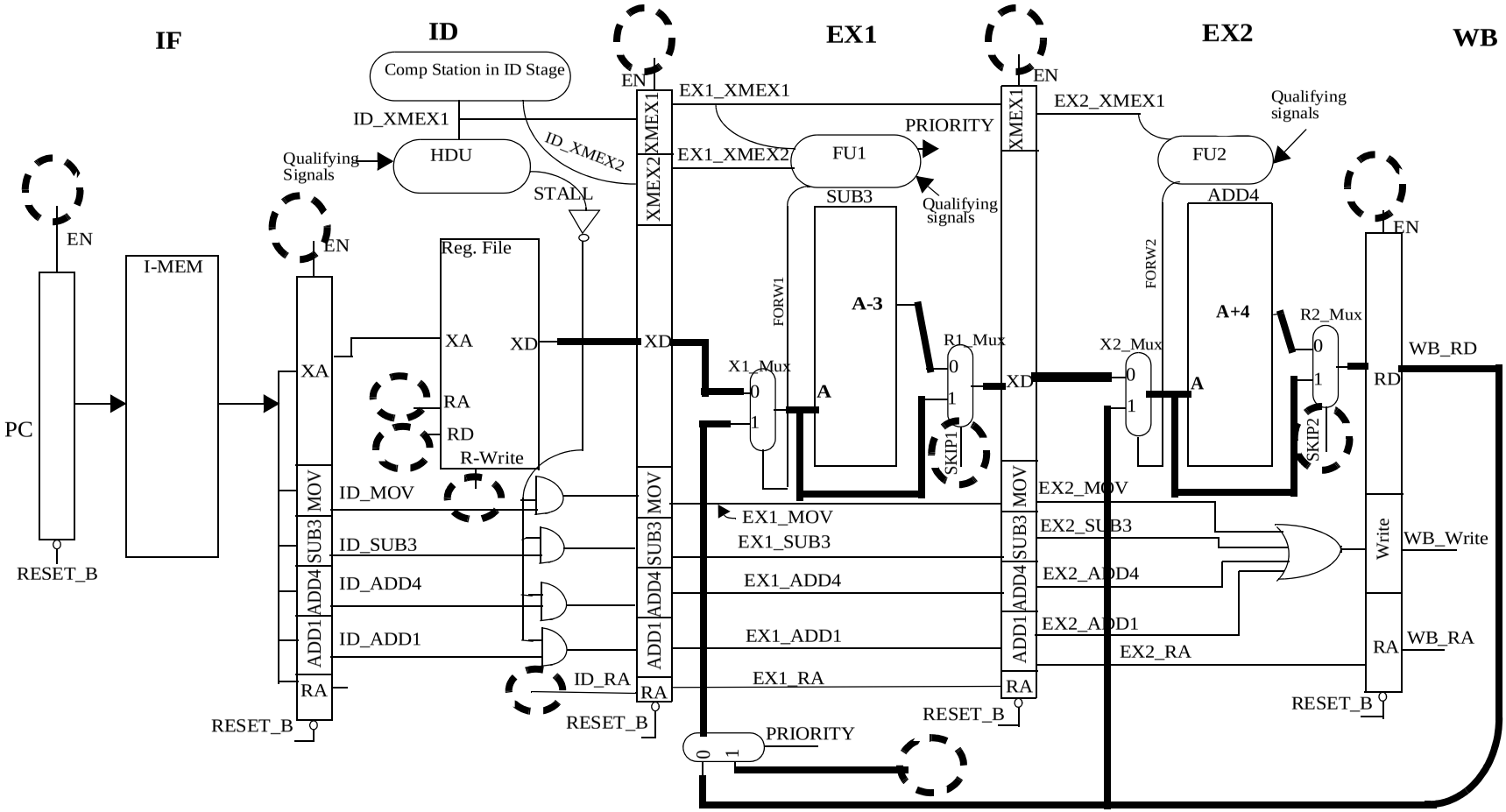


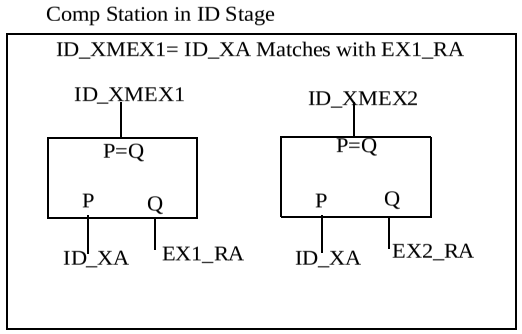
The block diagram for this modified version is shown on next to next page. Read the notes on the block diagram. R2_Mux movement is shown below. Simulate your completed modified design with the given testbench and wave.do files.





© Copyright 2011 Gandhi Puvvada

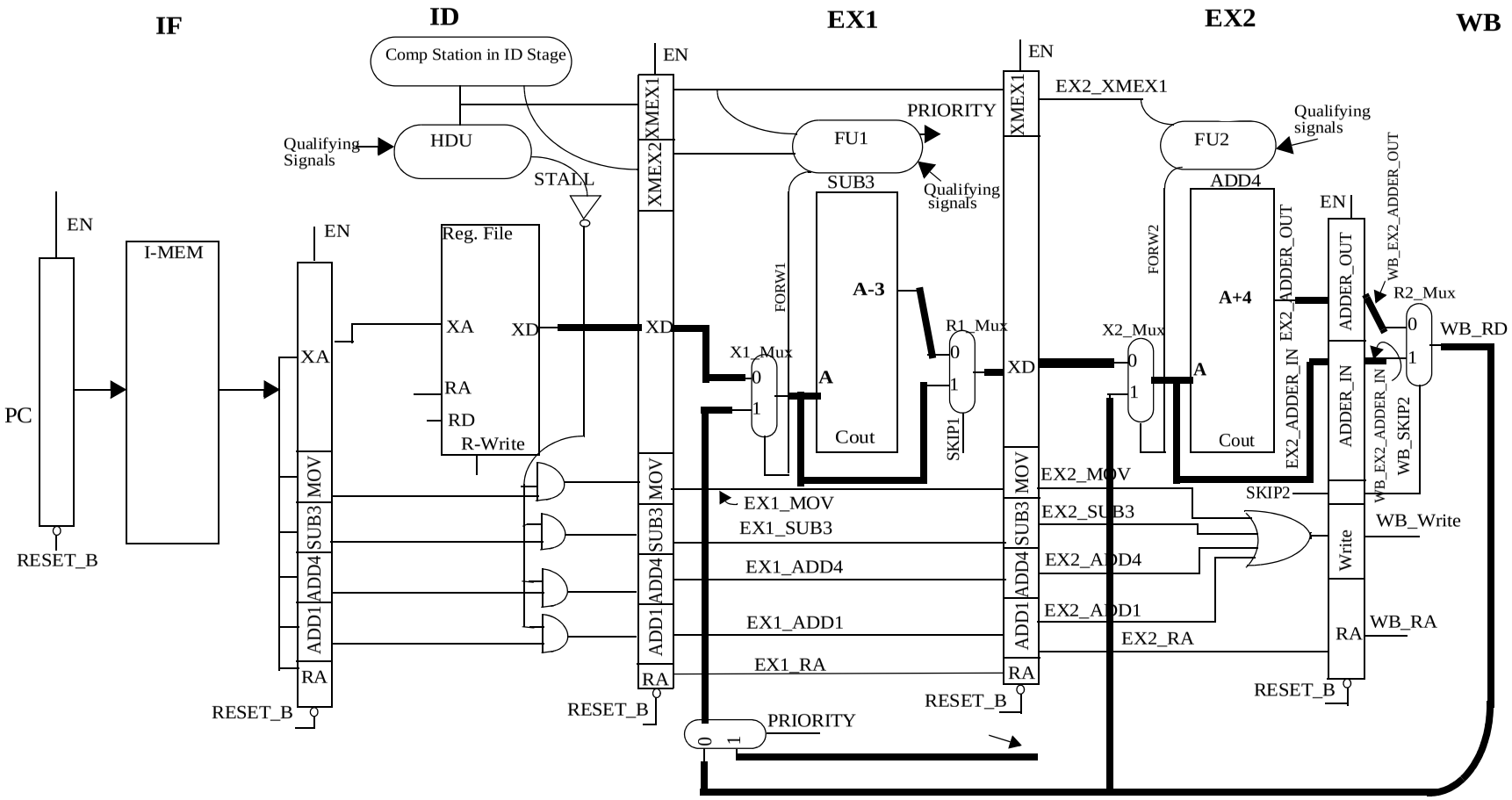


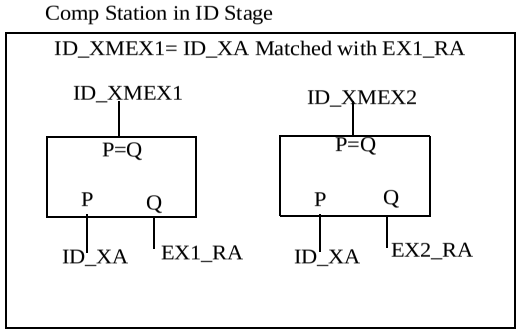


LAB 7 Part 3 Block Diagram for subpart 1 (for your reference) **Fig. 1**

revised 7/18/2010

© Copyright 2011 Gandhi Puvvada





1. You **do not need to complete** this block diagram or submit this.
2. This shows that the R2_Mux has been moved to the WB stage for your RTL exercise.
   Notice the signals, SKIP2, EX2_ADDER_IN, EX2_ADDER_OUT in EX2 stage and
   the new signals, **WB_SKIP2, WB_EX2_ADDER_IN, WB_EX2_ADDER_OUT** in WB stage.
3. Clock signals are not shown on any of the components for simplicity.
4. In earlier parts of this lab, the Register File was designed to write on the negative edge of the clock.
   This makes the internal forwarding in the register file happen automatically without any comparison unit or a multiplexer.
   For the RTL exercise, the Register File is considered as writing on the positive edge of the clock.
   Hence students have to code the needed logic for the internal forwarding.

LAB 7 Part 3 Block Diagram modified for Subpart 3 RTL Exercise

Fig. 1 RTL Exercise

revised 11/22/2010

© Copyright 2011 Gandhi Puvvada

The Subpart #4 assignment

Here, you are asked to code the EX1-EX2 merged design of the Subpart #2 in RTL coding style.

What you have to do

1. There are four subparts to this Lab#7 Part3. The fours subparts are:

Subpart	File Name Prefix
Lab #7 Part #3 Subpart #1	ee457_lab7_P3
Lab #7 Part #3 Subpart #2	ee457_lab7_P3_with_EX1_EX2_merged
Lab #7 Part #3 Subpart #3	ee457_lab7_P3_RTL_Coding_Style
Lab #7 Part #3 Subpart #4	ee457_lab7_P3_RTL_Coding_Style_with_EX1_EX2_merged

2. There was a separate handout on subparts #1 and #2 and you finished those subparts earlier. Here, you are doing Subparts #3 and #4.

3. Let us start working on **Lab #7 Part #3 Subpart#3 (ee457_lab7_P3_RTL_Coding_Style)**.

4. Import ee457_lab7_P3_RTL_Coding_Style.zip. and transfer the verilog and `.do` files (for RTL coding of Subpart #1) (`ee457_lab7_P3_RTL_Coding_Style.v, ee457_lab7_P3_RTL_Coding_Style_tb.v, ee457_lab7_P3_RTL_Coding_Style_wave.do for Sub part #1`) into `C:\ModelSim_projects\ee457_lab7_P3_RTL_Coding_Style`. Go through the completed design file, `ee457_lab7_P3_RTL_Coding_Style.v` . Setup the Modelsim project `ee457_lab7_P3_RTL_Coding_Style` with the project directory `C:\ModelSim_projects\ee457_lab7_P3_RTL_Coding_Style`. Compile the verilog files and start simulation. Setup the waveform using the given wave.do file. Display the register file contents. Simulate for 1 ns (run 1ns) and display the initial register file contents. You can use the examine command at the VSIM> prompt.

```
VSIM> examine -radix hex UUT/reg_file
# {0001 0002 0004 0008 0010 0020 0040 0080 0100 0200 0400 0800 1000 2000 fff8 ffff}
```

Now simulate for additional 1199 ns (run 1199ns) and check the register file contents again.

```
VSIM> examine -radix hex UUT/reg_file
# {0008 0009 000c 0025 000a 000d 0043 0080 0102 0043 0043 0044 0044 0041 fff8 ffff}
```

Read the `TimeSpace.txt` file in `Notepad++`

5. Read and understand completely the RTL code for Subpart #1 (the file: `ee457_lab7_P3_RTL_Coding_Style.v`). Read the comments fully.

6. Import `ee457_lab7_P3_RTL_Coding_Style_R2_Mux_in_WB.zip` . unzip and place the only one file provide through this .zip file, `ee457_lab7_P3_RTL_Coding_Style_R2_Mux_in_WB_wave.do`, in a new project directory `C:\ModelSim_projects\ee457_lab7_P3_RTL_Coding_Style_R2_Mux_in_WB.` Copy the `ee457_lab7_P3_RTL_Coding_Style_tb.v` from the earlier project directory to this project directory. Make a copy of the earlier RTL code file, `ee457_lab7_P3_RTL_Coding_Style.v`, into this directory with a new name, `ee457_lab7_P3_RTL_Coding_Style_R2_Mux_in_WB.v`. Take time to modify this carefully and thoughtfully. Create a modelsim project, `ee457_lab7_P3_RTL_Coding_Style_R2_Mux_in_WB`. Like in the previous step, simulate and verify the final contents of the register file. Copy the reg_file contents (after running for 1ns and again after running for further 1199ns) into a file. Copy the responses in the transcript window into `"RF_Content_Lab7_P3_RTL_Coding_Style_R2_Mux_in_WB.txt"` for online submission. If you design worked correctly, this file should have contents as shown below.

```
# {0001 0002 0004 0008 0010 0020 0040 0080 0100 0200 0400 0800 1000 2000 fff8 ffff}
# {0008 0009 000c 0025 000a 000d 0043 0080 0102 0043 0043 0044 0044 0041 fff8 ffff}
```

7. Let us start working on **Lab #7 Part #3 Subpart#4 (ee457_lab7_P3_RTL_Coding_Style_with_EX1_EX2_merged)**.

8. Download `ee457_lab7_P3_RTL_Coding_Style_with_EX1_EX2_merged.zip` and transfer the testbench verilog file and the wave.do file provided through this .zip file into `C:\ModelSim_projects\ee457_lab7_P3_RTL_Coding_Style_with_EX1_EX2_merged`.

© Copyright 2011 Gandhi Puvvada

9. Make a copy of the given completed RTL code for Subpart #1, `ee457_lab7_P3_RTL_Coding_Style.v`, call it ee457_lab7_P3_RTL_Coding_Style_with_EX1_EX2_merged.v, carry out the needed modifications and simulate with the given testbench and wave.do files. Use the signal names for the merged design as used in the wave.do file. some of the signal names are listed here:
1-bit control signals: `EX12_MOV, EX12_SUB3, EX12_ADD4, EX12_ADD1, EX12_XMEX12`
4-bit result register ID: `EX12_RA`
16-bit data: `EX12_XD, EX12_SUB3_IN, EX12_SUB3_OUT, EX12_ADD4_IN, EX12_ADD4_OUT, EX12_XD_OUT`

10. Create a modelsim project with the name `ee457_lab7_P3_RTL_Coding_Style_with_EX1_EX2_merged`, add the verilog files, compile them, start simulation of the module `ee457_lab7_P3_tb`. Use `ee457_lab7_P3_RTL_Coding_Style_with_EX1_EX2_merged_wave.do` to set up the waveform, run for 1ns and run further for 1199ns. Examine the register file contents, the waveform, and the **`TimeSpace.txt`**. Copy the register file contents into `"RF_Content_Lab7_P3_RTL_Coding_Style_with_EX1_EX2_merged.txt"` for submission.

11. General Guidelines

12. Start early and seek help early if needed.
13. Finally submit online (through your unix account) (submission commands specified separately) one set of files for a team of two students:
14. You need to use the file names exactly as stated and follow the submission procedure exactly as specified. We use unix script files to automate grading.
15. **Non-working lab submission**: In simulation, it will be evident if your lab isn't working. We discourage you from submitting a non-working lab. If you want to submit a non-working lab, each member of your team needs to send an email to all lab graders (with a copy to all TAs) stating in the subject line, "EE457 Non-working lab submission request" and obtain an approval from one of them. Submitting a non-working lab or partial lab without such approval is interpreted as an **intention to cheat**. Sorry to say all this, but this makes sure that the system works well.

What you have to turn-in

On-line (one submission for a team of 2 students)

Please turn in the following (two separate submissions):

```
1. submit -user ee457lab -tag puvvada_lab7_p3_rtl
   ee457_lab7_P3_RTL_Coding_Style_R2_Mux_in_WB.v
   RF_Content_Lab7_P3_RTL_Coding_Style_R2_Mux_in_WB.txt TimeSpace.txt names.txt
2. submit -user ee457lab -tag puvvada_lab7_p3_rtl_merged
   ee457_lab7_P3_RTL_Coding_Style_with_EX1_EX2_merged.v
   RF_Content_Lab7_P3_RTL_Coding_Style_with_EX1_EX2_merged.txt TimeSpace.txt names.txt
```

Paper submission (No paper submission for these two subparts)

© Copyright 2011 Gandhi Puvvada