Group #:

G

Final Project

Part 2

ENSC 350 2020

Last Name:

SID:

Last Name:

SID:

Last Name:

SID:

Last Name:

SID:

Final Project – Part 2

Table of Contents:

Table of Contents:

Final Project – Part 2

Part 1:

Main Objective:

1) To design a Logic Unit and Arithmetic Unit.
2) To perform Functional Verification on both units.
3) To Synthesise both units for a Cyclone IV FPGA.
4) To perform Timing Verification on both units.

Sub-Objective 1:

1) To setup an organised process for Synthesis & Verification

Sub-Objective 2:

1) To design a Logic Unit for **RV64I** operatyions.
2) To perform Functional Verification on the Entity.
3) To Synthesise the Entity for a Cyclone IV FPGA.
4) To perform Timing Verification on the Entity.

Sub-Objective 3:

1) To design an Arithmetic Unit for **RV64I** operatyions.
2) To perform Functional Verification on the Entity.
3) To Synthesise the Entity for a Cyclone IV FPGA.
4) To perform Timing Verification on the Entity.

Part 2:

Main Objective:

1) To design a complete RISC-V Execution Unit, compatible with **RV64I**
2) To perform Functional Verification on all Entities.
3) To Synthesise all Entities for a Cyclone IV FPGA.
4) To perform Timing Verification on all Entities.

Sub-Objective 1:

1) To modify the Arithmetic Unit to include an output from the Adder.
2) To perform Functional Verification on the Entity.
3) To Synthesise the Entity for a Cyclone IV FPGA.
4) To perform Timing Verification on the Entity.

Sub-Objective 2:

1) To design three Barrel Shifters, SLL64, SRL64 & SRA64.
2) To perform Functional Verification on all Entities.
3) To Synthesise all Entities for a Cyclone IV FPGA.
4) To perform Timing Verification on all Entities.

Sub-Objective 3:

1) To design the Shift Unit that performs both 64-bit & 32-bit operations.
2) To perform Functional Verification on the Entity.
3) To Synthesise the Entity for a Cyclone IV FPGA.
4) To perform Timing Verification on the Entity.

Sub-Objective 3:

1) To design the Execution Unit that compatible with **RV64I** operations.
2) To perform Functional Verification on the Entity.
3) To Synthesise the Entity for a Cyclone IV FPGA.
4) To perform Timing Verification on the Entity.

Constraints: Part 1

The Logic Unit:

The Logic Unit should:

- be compatible with all **RV64I** logic instructions,
- include a control inputs LogicFN according to the following table,
- select the input operand, **B**, when LogicFN = "00".
- should only use `std_logic` or `std_logic_vector` for its interface signals.
  (to be compatible with the types created by Quartus' netlist)

```
Entity LogicUnit is
Generic ( N : natural := 64 );
Port ( A, B : in std_logic_vector( N-1 downto 0 );
       Y : out std_logic_vector( N-1 downto 0 );
       LogicFN : in std_logic_vector( 1 downto 0 ) );
End Entity LogicUnit;
```

LogicFn		operation
0	0	Pass B
0	1	A xor B
1	0	A or B
1	1	A and B

The Arithmetic Unit:

The Arithmetic Unit should:

- include an extra output directly from the Adder.
- perform 32-bit additions/subtractions when ExtWord = '1' compatible with **RV32I**,
- form the status signals Cout, Ovfl, Zero, AltB & AltBu directly from the Adder output.
- ensure that status signals are correct for **SLT** and **Branch** instructions.
- include a control signal NotA that complements input operand **A**,
- should only use `std_logic` or `std_logic_vector` for its interface signals.
  (to be compatible with the types created by Quartus' netlist)

```
Entity ArithUnit is
Generic ( N : natural := 64 );
Port ( A, B : in std_logic_vector( N-1 downto 0 );
       AddY, Y : out std_logic_vector( N-1 downto 0 );
-- Control signals
       NotA, AddnSub, ExtWord : in std_logic := '0';
-- Status signals
       Cout, Ovfl, Zero, AltB, AltBu : out std_logic );
End Entity ArithUnit;
```

Constraints: Part 2

The 64-bit Barrel Shifters:

The barrel shifters should

- Operate on 64-bit data, using 4-channel MUXes.
- perform the operations
  Shift Left Logical, (SLL64), Shift Right Logical, (SRL64) & Shift Right Arithmetic, (SRA64).

```
Entity SLL64 is
Generic ( N : natural := 64 );
Port (  X : in std_logic_vector( N-1 downto 0 );
            Y : out std_logic_vector( N-1 downto 0 );
      ShiftCount : in unsigned( integer(ceil(log2(real(N))))-1 downto 0 ) );
End Entity SLL64;
```

```
Entity SRL64 is
Generic ( N : natural := 64 );
Port (  X : in std_logic_vector( N-1 downto 0 );
            Y : out std_logic_vector( N-1 downto 0 );
      ShiftCount : in unsigned( integer(ceil(log2(real(N))))-1 downto 0 ) );
End Entity SRL64;
```

```
Entity SRA64 is
Generic ( N : natural := 64 );
Port (  X : in std_logic_vector( N-1 downto 0 );
            Y : out std_logic_vector( N-1 downto 0 );
      ShiftCount : in unsigned( integer(ceil(log2(real(N))))-1 downto 0 ) );
End Entity SRA64;
```

Final Project – Part 2

The Shift Unit:

The Shift Unit should:

- include a control inputs ShiftFN according to the following table,
- include a control input, ExtWord, that sign-extends 32-bit results to 64-bit,
- include an extra input, **C**, that is selected for ShiftFN = "00"
- perform 32-bit operations using the 64-bit barrel shifters,
- should ignore unnecessary bits when determining the shift count, and
- should only use `std_logic` or `std_logic_vector` for its interface signals.
  (to be compatible with the types created by Quartus' netlist)

```
Entity ShiftUnit is
Generic ( N : natural := 64 );
Port ( A, B, C : in std_logic_vector( N-1 downto 0 );
           Y : out std_logic_vector( N-1 downto 0 );
           ShiftFN : in std_logic_vector( 1 downto 0 );
           ExtWord : in std_logic );
End Entity ShiftUnit;
```

ShiftFN		operation
0	0	arith
0	1	sll
1	0	srl
1	1	sra

The Execution Unit:

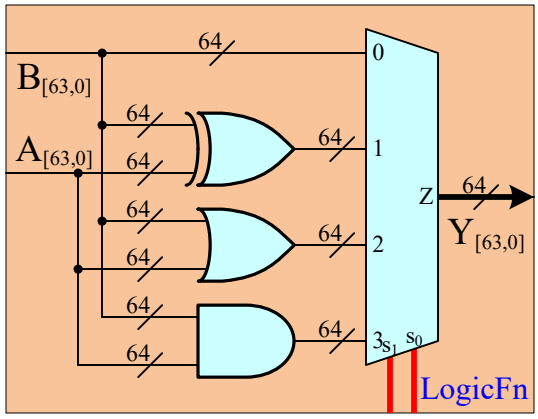
The Execution Unit should

- include control inputs ExUFunc := {NotA, FuncClass, LogicFN, ShiftFN, AddnSub, ExtWord}
- include FuncClass, encoded according to the following table,
- have three status outputs, ExUStatus := {Zero, AltB, AltBu},
- be compatible with all **RV64I** instructions, and
- should only use `std_logic` or `std_logic_vector` for its interface signals.
  (to be compatible with the types created by Quartus' netlist)

```
Entity ExecUnit is
Generic ( N : natural := 64 );
Port ( A, B : in std_logic_vector( N-1 downto 0 );
       NotA : in std_logic := '0';
       FuncClass : in std_logic_vector( 1 downto 0 );
       LogicFN : in std_logic_vector( 1 downto 0 );
       ShiftFN : in std_logic_vector( 1 downto 0 );
       AddnSub, ExtWord : in std_logic := '0';
       Y : out std_logic_vector( N-1 downto 0 );
       Zero, AltB, AltBu : out std_logic
       );
End Entity ExecUnit;
```

FuncClass		operation
0	0	sltu
0	1	slt
1	0	shift/arith
1	1	logic

Logic Unit – Circuit Diagram:

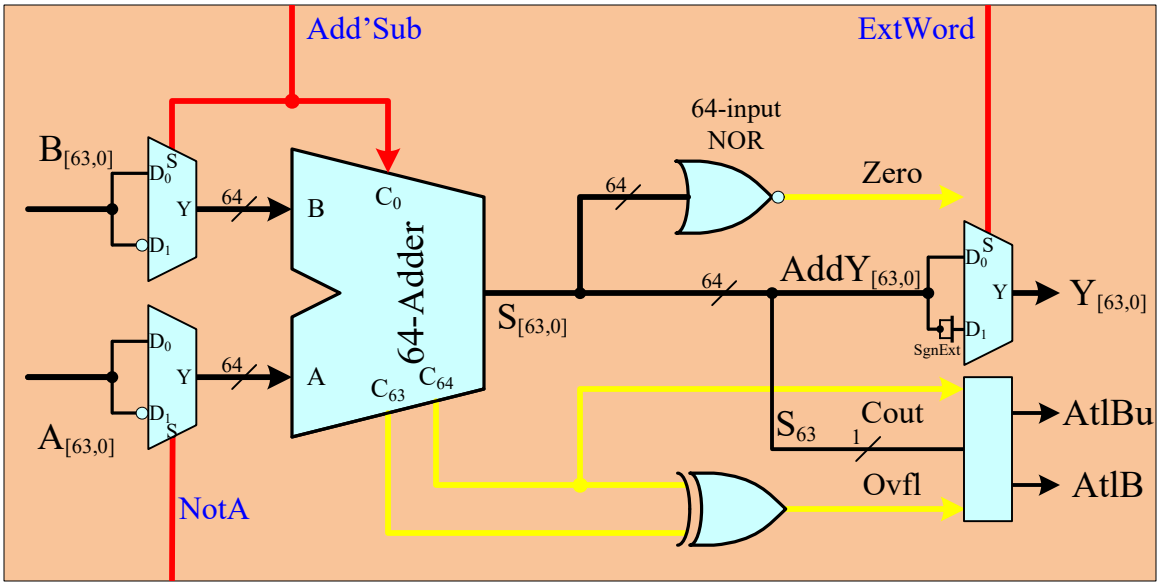


LogicFn		operation
0	0	Pass B
0	1	A xor B
1	0	A or B
1	1	A and B

The Logic Unit is extremely primitive. Build and Verify this circuit first as it will aid you in becoming familiar with the testing procedures.

```
Entity LogicUnit is
Generic ( N : natural := 64 );
Port ( A, B : in std_logic_vector( N-1 downto 0 );
       Y : out std_logic_vector( N-1 downto 0 );
       LogicFN : in std_logic_vector( 1 downto 0 ) );
End Entity LogicUnit;
```

Arithmetic Unit – Circuit Diagram:



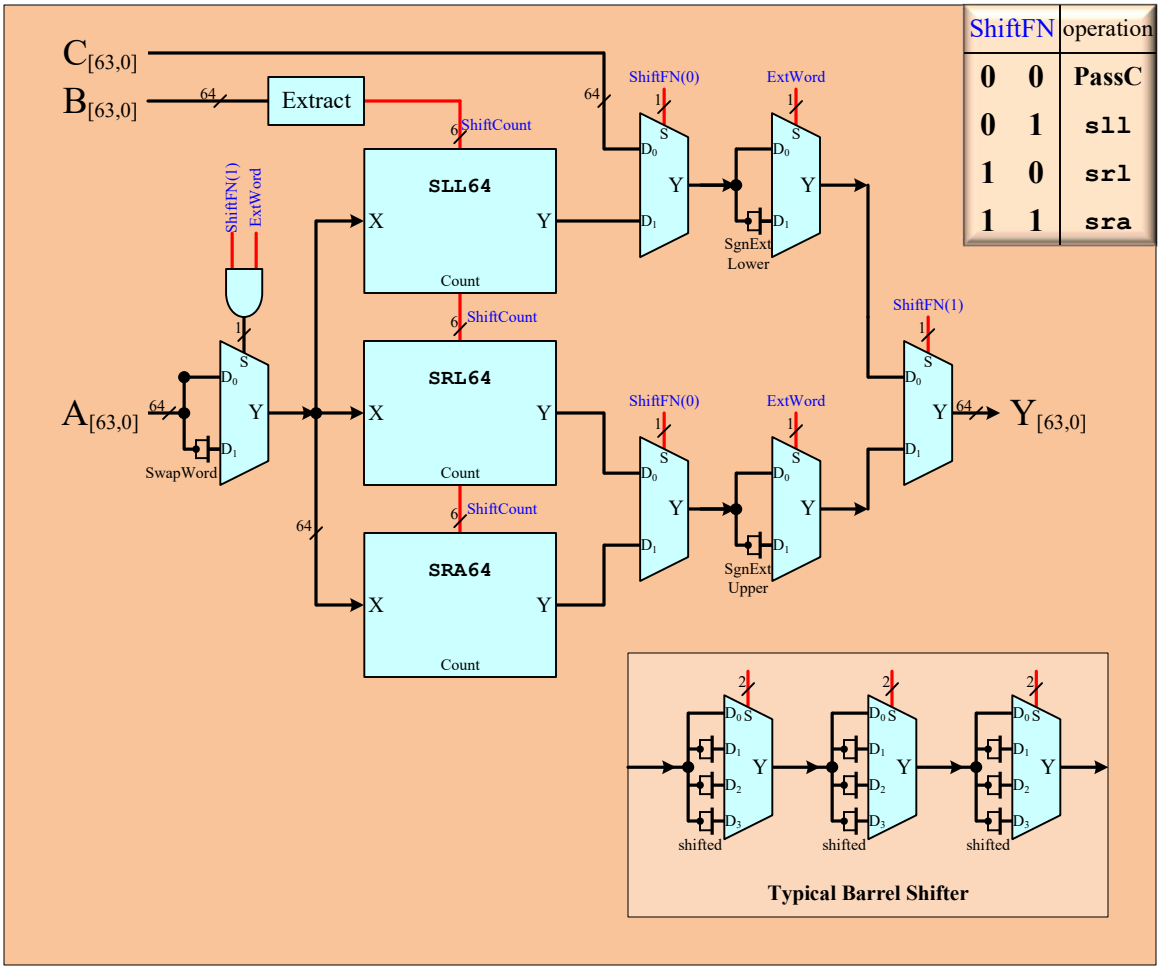
The status signals AltB and AltBu are only used for SLT and BR instructions, both of which involve 64-bit subtractions. As such it doesn't matter whether the status signals are formed before or after the sign-extension MUX. Furthermore, the sign-extension MUX is actually not in this part of the circuit and must eventually be removed.

Provided both AddY and Y outputs from this entity.

```
Entity ArithUnit is
Generic ( N : natural := 64 );
Port ( A, B : in std_logic_vector( N-1 downto 0 );
          AddY, Y : out std_logic_vector( N-1 downto 0 );
-- Control signals
        NotA, AddnSub, ExtWord : in std_logic := '0';
-- Status signals
        Cout, Ovfl, Zero, AltB, AltBu : out std_logic );
End Entity ArithUnit;


Entity Adder is
Generic ( N : natural := 64 );
Port ( A, B : in std_logic_vector( N-1 downto 0 );
       Y : out std_logic_vector( N-1 downto 0 );
-- Control signals
        Cin : in std_logic;
-- Status signals
        Cout, Ovfl : out std_logic );
End Entity Adder;
```

Shift Unit – Circuit Diagram:

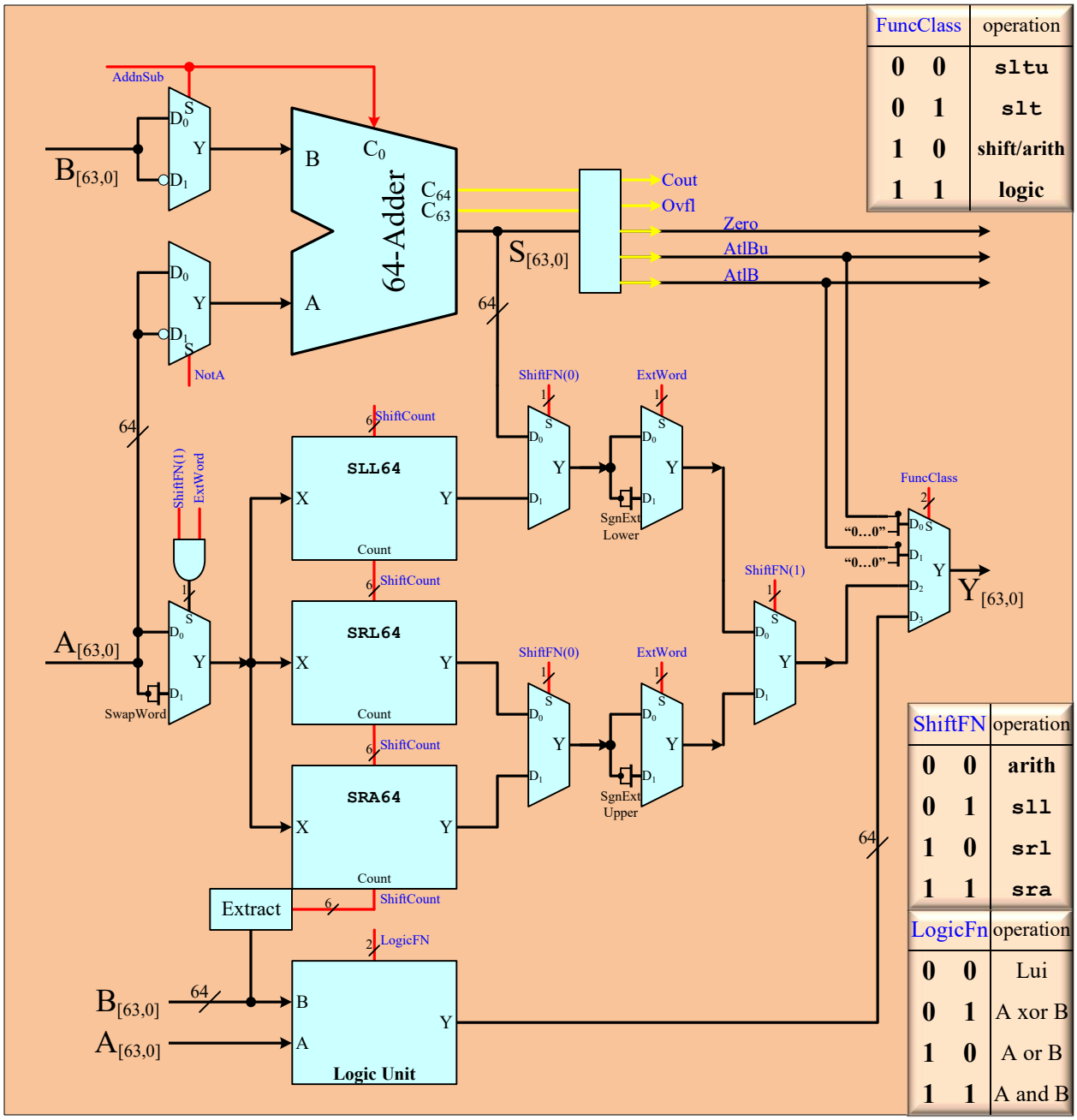


The shift count is contained in the lower bits of operand B. The element "extract" must correctly form the Shift Count according to the RISC-V ISA specification. (refer to pages 36 & 37)
The 64-bit Barrel Shifters contain only 4-channel MUXs, to perform their respective shift operations.

Warning: The VHDL "**sra**" operator is obsolete and most likely will not work correctly.
You may wish to learn the ieee functions, RESIZE(), SHIFT_LEFT() & SHIFT_RIGHT()

```
Entity ShiftUnit is
Generic ( N : natural := 64 );
Port ( A, B, C : in std_logic_vector( N-1 downto 0 );
           Y : out std_logic_vector( N-1 downto 0 );
           ShiftFN : in std_logic_vector( 1 downto 0 );
           ExtWord : in std_logic );
End Entity ShiftUnit;
```

Execution Unit – Circuit Diagram:



FuncClass		operation
0	0	sltu
0	1	slt
1	0	shift/arith
1	1	logic

ShiftFN		operation
0	0	arith
0	1	sll
1	0	srl
1	1	sra

LogicFn		operation
0	0	Lui
0	1	A xor B
1	0	A or B
1	1	A and B

```
Entity ExecUnit is
Generic ( N : natural := 64 );
Port ( A, B : in std_logic_vector( N-1 downto 0 );
       NotA : in std_logic := '0';
       FuncClass, LogicFN, ShiftFN : in std_logic_vector( 1 downto 0 );
       AddnSub, ExtWord : in std_logic := '0';
       Y : out std_logic_vector( N-1 downto 0 );
       Zero, AltB, AltBu : out std_logic   );
End Entity ExecUnit;
```

Procedure: (Part 1)

Final Project (part 1) Setup:

Prepare your filing structure. (§Filing Structure)
Copy the files that I provide into the correct folders. (§Filing Structure)
Prepare your activity logging files. (§Activity Logging Procedure)
Prepare your Final Project Report (part 1) document. (§Documentation)
Prepare ModelSim project. (§Setting up ModelSim)
Prepare a Quartus Project. (§Setting up Quartus)

The Logic Unit:

Design the Logic Unit as shown on page 7.
~~Write a Testbench~~. (§Creating a Testbench)
~~Create Test Vectors~~. (§Creating Test Vectors)
Run a Functional Simulation. (§Logic Unit - Functional Simulation)
Document the results of the Functional Simulation. (§Logic Unit – Saving Functional Waves)
Synthesise the Logic Unit and construct post-fit-Netlist and Timing files. (§Logic Unit – Synthesis)
Run Timing Simulation. (§Logic Unit - Timing Simulation)
Document the results of the Timing Simulation. (§Logic Unit – Saving Timing Waves)

The Arithmetic Unit:

Design the Arithmetic Unit as shown on page 8,
~~Write a Testbench~~. (§Creating a Testbench)
~~Create Test Vectors~~. (§Creating Test Vectors)
Run a Functional Simulation.
Document the results of the Functional Simulation.
Synthesise the Logic Unit and construct post-fit-Netlist and Timing files.
Run Timing Simulation.
Document the results of the Timing Simulation.

Complete the Final Project Report.
Prepare an archived package of all documents and submit to Canvas.

Procedure: (Part 2)

Final Project (part 2) Setup:

Copy the files that I provide into the correct folders. (§Filing Structure)
Prepare your activity logging files. (§Activity Logging Procedure)
Prepare your Final Project Report (part 2) document. (§Documentation)

The Arithmetic Unit:

(if not already done) Modify the Arithmetic Unit to include an extra output directly from the Adder.
Verify the function using a testbench that includes this modification.
Synthesise the Arithmetic Unit and construct post-fit-Netlist and Timing files.
Run Timing Simulation.

The 64-bit Barrel Shifters:

Design three Barrel Shifters, according to the constraints given on page 5.
~~Write a Testbench~~. (§Creating a Testbench)
~~Create Test Vectors~~. (§Creating Test Vectors)
Design the Shift Unit as shown on page 9.
Run a Functional Simulation.
Synthesise the Shift Unit and construct post-fit-Netlist and Timing files.
Run Timing Simulation.
Fully document the results of the previous three steps.

The Execution Unit:

Design the Execution Unit, as shown on page 10. (using the previously designed entities)
~~Write a Testbench~~. (§Creating a Testbench)
~~Create Test Vectors~~. (§Creating Test Vectors)
Run a Functional Simulation.
Synthesise the Shift Unit and construct post-fit-Netlist and Timing files.
Run Timing Simulation.
Fully document the results of the previous three steps.

Complete the Final Project Report.
Prepare an archived package of all documents and submit to Canvas.

Filing Structure:

Create a tree of folders as shown. *Use the names that I specify to ensure that the scripts run smoothly*.

Use the convention that files associated with
the Logic Unit have the string "LogicUnit" or "**LU**" within their filename.
the Arithmetic Unit have the string "ArithUnit" or "**AU**" within their filename.
the Shift Unit have the string "ShiftUnit" or "**SU**" within their filename.
the Execution Unit have the string "ExecUnit" or "**XU**" within their filename.
the Barrel Shifters are distinguished by the strings "SLL64", "SRL64" & "SRA64"

Testbenches have the same name as the entity that they test with the prefix "**TB**".

Avoid directory names in the path that contain "spaces", use underscores instead.

ExU: The top-level folder to be used as Quartus' project folder.

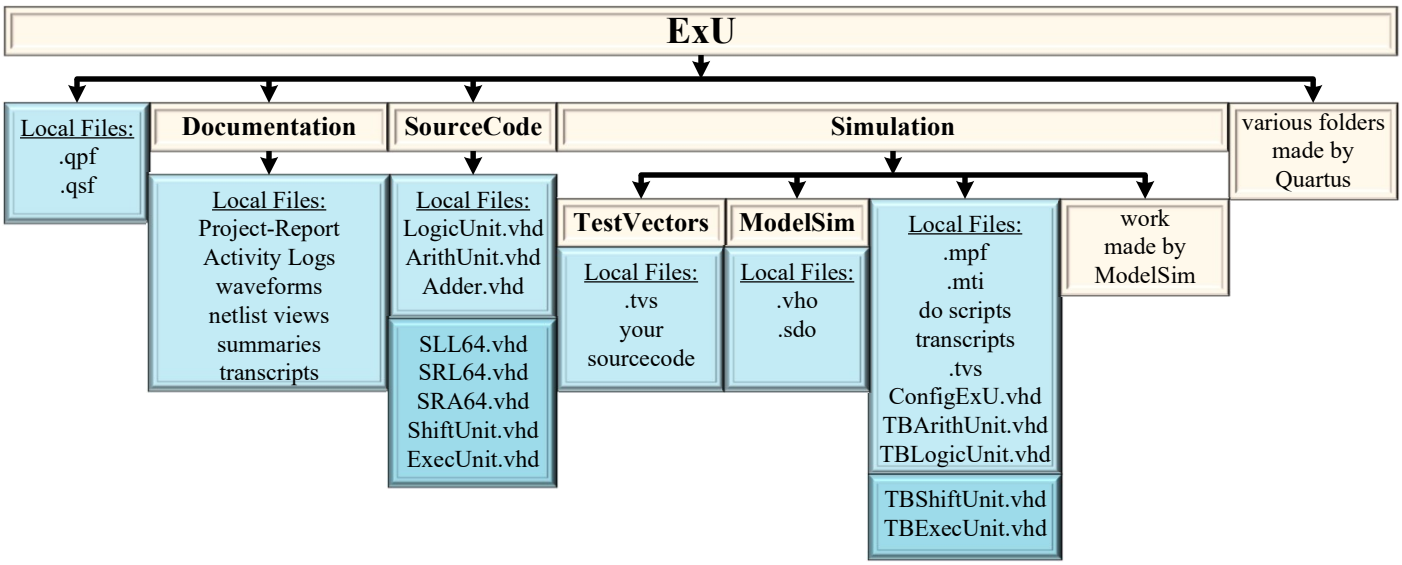
Documentation: Used to collect files needed for project documentation.

SourceCode: Used to store the synthesisable VHDL design files.

Simulation: Used as the ModelSim project folder.

ModelSim: Used by Quartus to store post-fit netlists and timing files.

TestVectors: Used by you to create Test Vectors using a programming language of your choice.



Files Provided:

Unzip the archive, "FP2.x-350-1201.zip"

FP-log-Gxx-xxxx-350-1201.xlsx → Documentation
ConfigExU.vhd → Simulation

Testbenches:
TBLogicUnit.vhd, TBArithUnit.vhd → Simulation
TBShiftUnit.vhd, TBExecUnit.vhd → Simulation

Test Vectors:
LogicUnit00.tvs, ArithUnit00.tvs, ArithUnit01.tvs → Simulation
SLL64Unit00.tvs, SRL64Unit00.tvs, SRA64Unit01.tvs → Simulation
SLL32Unit00.tvs, SRL32Unit00.tvs, SRA32Unit01.tvs → Simulation
ExecUnit00.tvs → Simulation

ModelSim Scripts:
FunctionalLogicUnit.do, TimingLogicUnit.do → Simulation
FunctionalArithUnit.do, TimingArithUnit.do → Simulation
FunctionalShiftUnit.do, TimingShiftUnit.do → Simulation
FunctionalExecUnit.do, TimingExecUnit.do → Simulation

WaveLogicUnit.do, WaveArithUnit.do → Simulation
WaveShiftUnit.do, WaveExecUnit.do → Simulation

Activity Logging Procedure:

Each group member must maintain a log of the work that they contribute to the project.
Each member should make a copy of the file "FP-log-Gxx-xxxx-350-1201.xlsx"
Rename the file replacing the 'x's
in "Gxx" with your group numbers digits and
"xxxx" is replaced with the last four digits of your student ID number.

Write your group number, Name and Student ID in the top header.

The workbook contains three worksheets,

- Enter your activity entries in the correct worksheet.
- There is one worksheet for each part of the final project.

You should create a log entries for all the time that you spend contributing to this project.
Enter dates and time in a format that excel recognises, otherwise excel will convert to a string.
Enter dates by typing, [datenum]<space>[a few characters for the month]
Enter times by typing, [hour]:[minute]<space>{a or p}

Whenever you begin a session, make note of the start time on a scrap piece of paper.
When working for extended periods, create a log entry approximately every two (or so) hours.
Whenever you take a break, make note of the end time, then create a log entry.

- The descriptions should be short.
- Break down your activities into small tasks.
- Do not combine multiple tasks into a single log entry. Instead make multiple log entries.
- Even a small task may have multiple entries because you take breaks.
- When a single task has many entries, write "DONE" by the final entries description.

You should enter values for your last-4-digits in each entry. Do this every time, as I will be merging and sorting the entries for all group members.

If you become ill or are unable to continue contributing to the project, make a final brief entry noting this event. (A group member may make this entry on your behalf)

Documentation:

Full Documentation:

Your project documentation will contain many files.
These files are to be zipped into a single archive called "FP2-Gxx-350-1201.zip"

The archive contents:

An Activity log excel workbook for each member of the group.

Final project report (for part 2)
VHDL sourcecode for all (eight) design entities.
Four + fourteen = eighteen transcript files,
A summary file – created by merging the eight summary files.
Four – post-fit netlists (.vho)
Four – standard delay format files (.sdo)

WaveLogicUnit.do and/or WaveArithUnit.do. (only if you modified them)
WaveShiftUnit.do and/or WaveExecUnit.do. (only if you modified them)

The Project Report:

You are to submit a Final Project Report, named "FP2-Report-Gxx-350-1201.pdf".
Include the correct cover page – provided at the beginning of this document.

This report is cumulative. You should start with your report from part 1 and make all corrections and changes specified in the feedback provided by your teaching assistant. The work that you conduct for part 2 should now logically fit into this report.

DO NOT add listings of your source code to the report.
Your report should be well organised. You choose how to divide the report into sensible sections.

You may wish to copy/paste specific statements from your VHDL sourcecode into the document.
If you use notepad++, you can copy the selected text to the clipboard with syntax highlighting.
Use the menu, Plugins->NppExport->Copy all formats to clipboard. You can also change the colours of the syntax highlighting using the menu, settings->style configurator. Then select the language VHDL (on the left) and choose your colours. I chose mine to best match the default colour of the ModelSim Editor.

Documenting Design Entities in the project report:

For each Design Entity, you should provide comprehensive information describing:

- A reference to the name of the attached VHDL source file.
- A paragraph describing the functional behaviour of the entity.
- The VHDL interface – descriptions of the purpose of all port signals.
- A circuit diagram for the entity.
  (all labels must identically match your VHDL sourcecode.)

Documenting the Testbenches in the project report:

For Part 2, I have written the testbenches on your behalf.

You do not need to include documentation of the testbenches for part 2.

~~Documentation of a testbench should include~~

~~a diagram of the processes and signals within the architecture,~~
~~a description of the flow for the procedure STIM: (maybe including a flowchart)~~
~~and a description of the choice of Test Vectors.~~

Documenting Simulation Runs:

Simulation runs are to be with documented with the following content.

Transcript files:

The scripts that I provided each produce a transcript file.

- FuncLogicUnitTranscript.txt & TimeLogicUnitTranscript.txt
- FuncArithUnitTranscript.txt & TimeArithUnitTranscript.txt
- FuncShiftUnitTranscript.txt & TimeShiftUnitTranscript.txt
- FuncExecUnitTranscript.txt & TimeExecUnitTranscript.txt

The Shift Unit should be tested with all six sets of test vectors.

Rename the transcript file after each simulation run.

Substitute the 5 characters "Shift" with the appropriate 5 characters describing the tested shift operation. { SLL64, SRL64, SRA64, SLL32, SRL32, SRA32 }

All eighteen transcript files should be included in your submitted archive.

Documenting Simulation Waves:

You should also make screen clippings of the waveforms.

You should capture a set of waveforms from each of the four simulation runs.

I have setup signals to display in the files waveLogicUnit.do and waveArith.do

These files are read by the four simulation run scripts.

- You may wish to add internal signals from your entity.
- You may wish to make change the colours of the waveforms.
- You decide how best to effectively convey useful information.
- When the wave window is active, type <ctrl-s> and save the changes, overwriting my file.
- If you change the waveforms, then include the modified .do file in your submitted archive.

The simulation waveforms should be sensibly included in your project report, with titles and annotations. Use two images per landscape page as shown in the example on the next page.

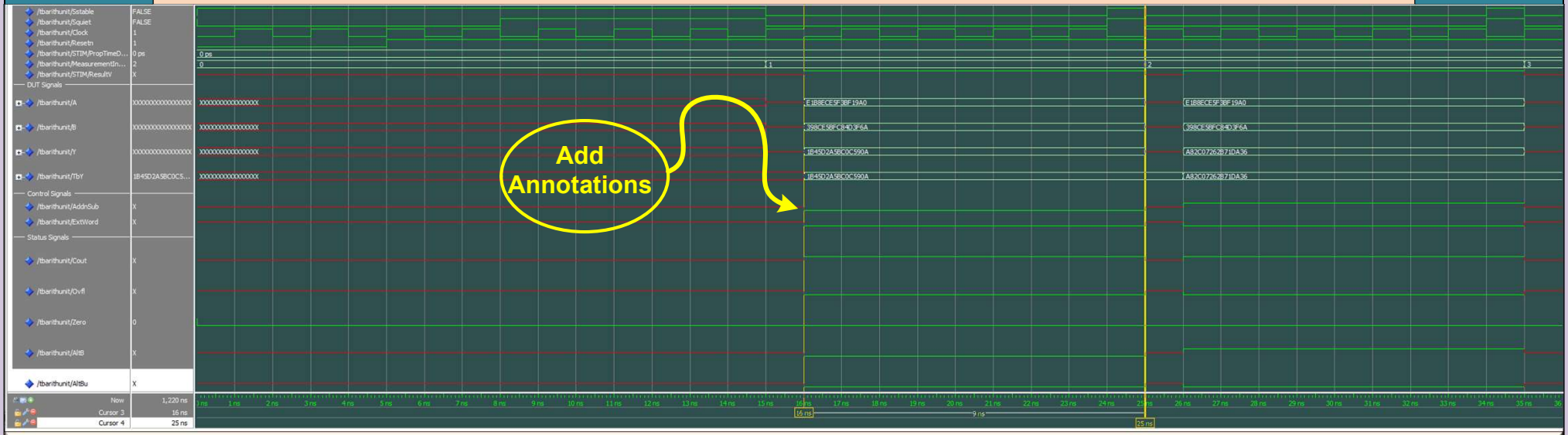
Each set should be accompanied with a discussion that explains why this simulation run verifies the functional behaviour and timing of the tested entity. The highlights on the images should be chosen to enhance this discussion.

To make the good images,

- Undock the wave window so that is occupies a large amount of the screen.
- Resize the window. Adjust the bottom so that the time-axis is close to the bottommost waveform.
- Type “r” - to select the zoom range specified in the procedure.
- Add measurement cursors as specified in the procedure.
- Clip the image - <window+shift+S>, then
- Paste the image to a file in the Documentation folder. <ctrl+shift+alt+V>

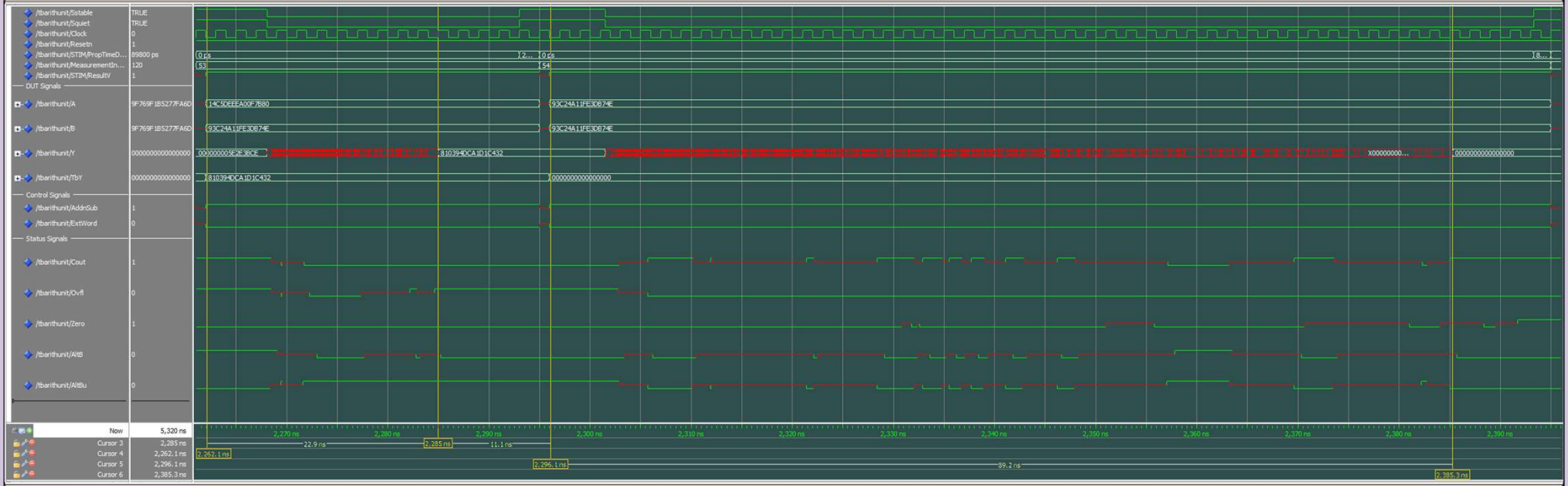
Always check that the text for the signal groups is readable.

The following page shows an example of some captures.



Functional Simulation - ArithUnit:

add some description of which measurements are being observed. Also the placement of the cursors. Add annotations to the wave.



Timing Simulation - ArithUnit:

Add a brief description. Measurement 53, t_{pd} = 22.9 ns, Measurement 54, t_{pd} = 89.2 ns

Documenting Synthesis and Fitting:
Using Quartus you will synthesise and fit your design entities. Quartus will produce summary reports which you should be included in your submitted archive.

Summary file: "FP2-Summary-Gxx-350-1201.txt"

Merge the eight summary files into a single file.

Insert title lines to ensure that the individual summary reports are easily distinguishable.

LogicUnit.map.summary, ArithUnit.map.summary,
ShiftUnit.map.summary, ExecUnit.map.summary,
LogicUnit.fit.summary, ArithUnit.fit.summary,
ShiftUnit.fit.summary, ExecUnit.fit.summary,

Screen Clipping from Netlist viewers:

You will need to collect screen clipping from the RTL netlist viewer and the Post-fitting netlist viewer.

These images should be included in the project report. (Refer to the example)

- The final Execution Unit contains all three sub-entities. As such, it is only necessary to collect images from this circuit. Document the sub-entities as well as the full circuit.

You decide how many is reasonable. Remember too many is BAD, not enough is also BAD.

You will be judged on your ability to sensible choose these images. You will score zero if it is clear that you are not making any effort to graphically convey useful information about the underlying circuit.

Refer to the examples on the next few pages.

Documenting VHDL Sourcecode for the Design Entities:
The VHDL sourcecode files are included separately from the project report. The project report simply references the code by filename.

Your VHDL sources files must contain useful comments. When commenting your code, imagine that you were reading the text in the distant future. Consider the comments as notes to yourself so that you could efficiently continue working on the project.

Documenting post-fit netlists and timing information:
Include in your submitted archive, the post-fit netlist files,

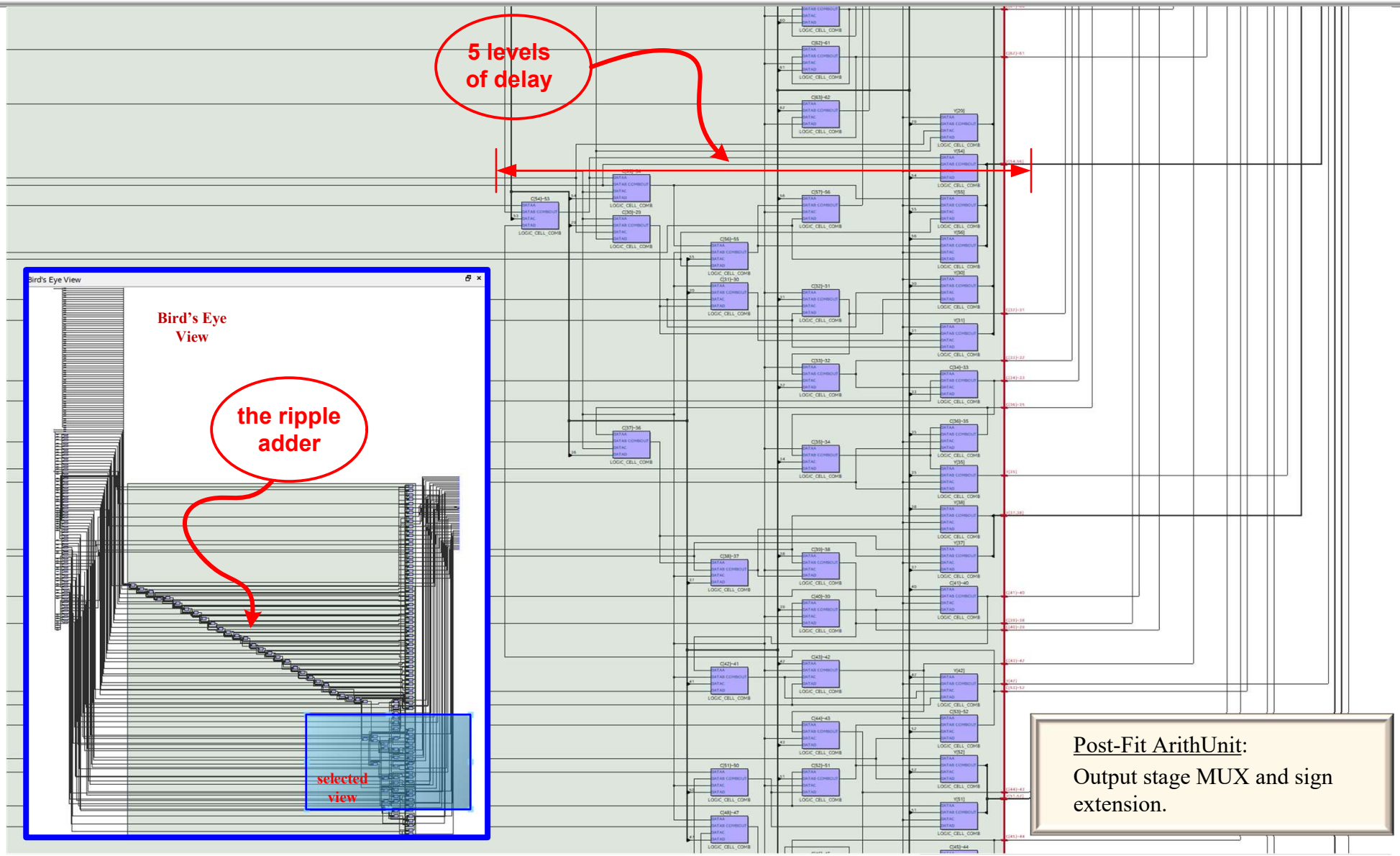
"LogicUnit.vho", "ArithUnit.vho", "ShiftUnit.vho", and "ExecUnit.vho",

and the standard delay format files,

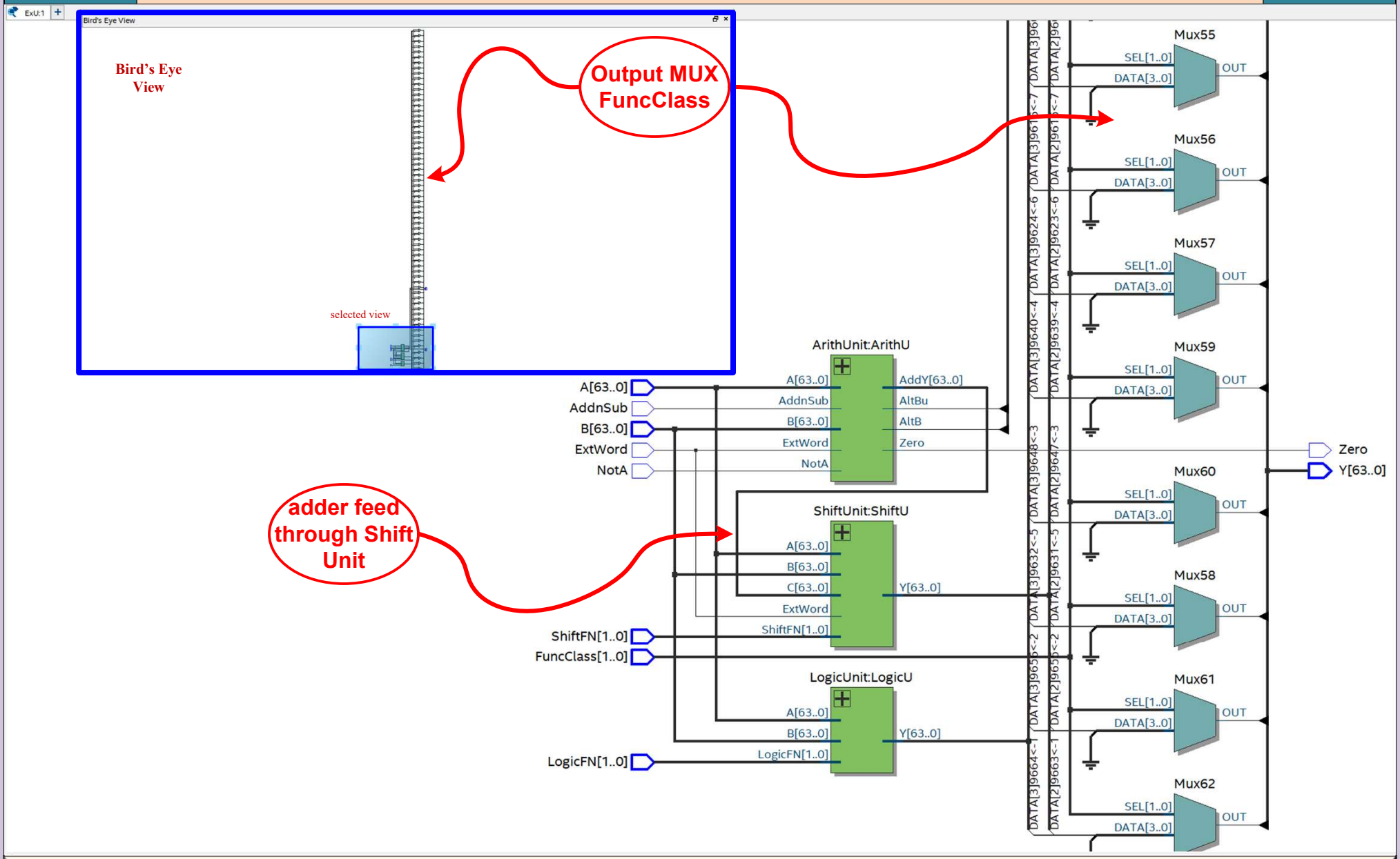
"LogicUnit.sdo", "ArithUnit.sdo", "ShiftUnit.sdo", and "ExecUnit.sdo".

Final Project – Part 2

Each screen clipping from a netlist viewer should occupy a full page so that details are not lost. (Here is an example)
Choose the page to be either landscape or portrait, whichever best matches the aspect ratio of your image.
Each image should have a title and a brief description. The image should contain both the zoomed view and the window for the bird's eye view.
Position and size the bird's eye window so that it doesn't obscure too much of the zoomed circuit. Choose the zoomed region wisely.



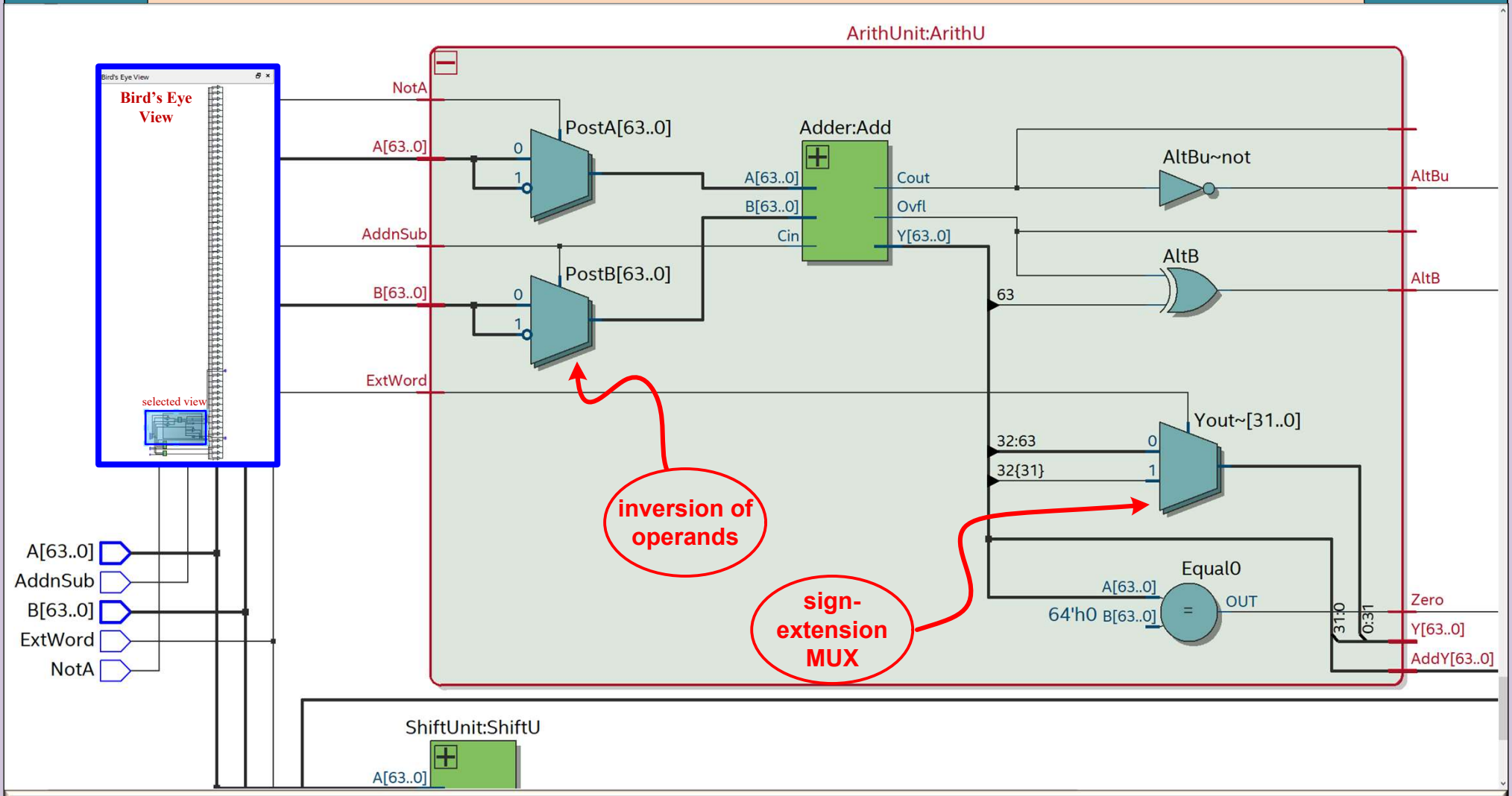
Final Project – Part 2



Post Synthesis RTL Circuit – Execution Unit:

The Execution Unit Top-Level View, showing the three main sub-circuits and the output MUX controlled by FuncClass.

Final Project – Part 2



Post-Synthesis RTL View – Execution Unit:

The internal view of the Arithmetic Circuit showing

- the formation of comparison signals (A<B) from output Carry and Overflow, the formation of the Zero flag and also the two input stage MUXes for selectively inverting the operands.
- The outputs Cout, Ovfl and Y[63..0] are left unconnected.

Setting up ModelSim:

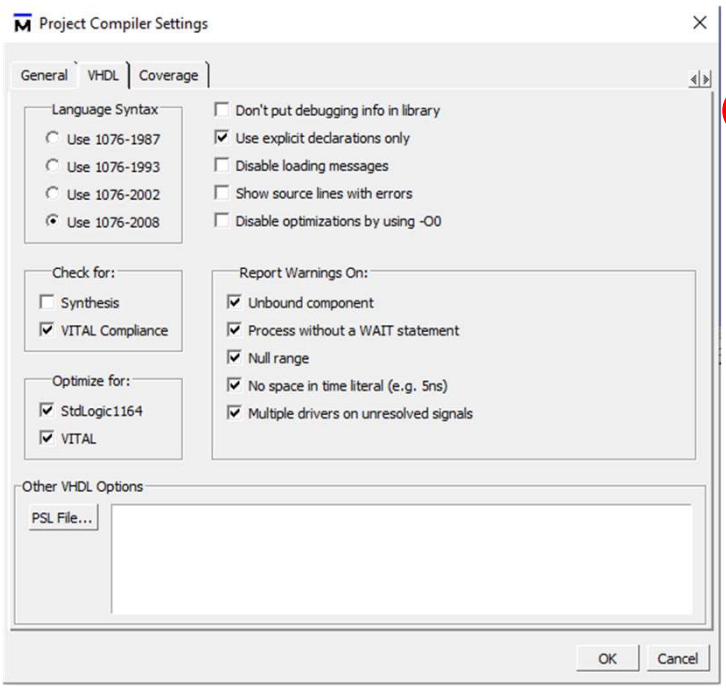
~~Create an Environment variable set to the project folder \ExU\.~~
~~Edit the four scripts so that the file reference use the local path on your computer. I couldn't make the environment variables work so you'll have to do this manually for now~~.

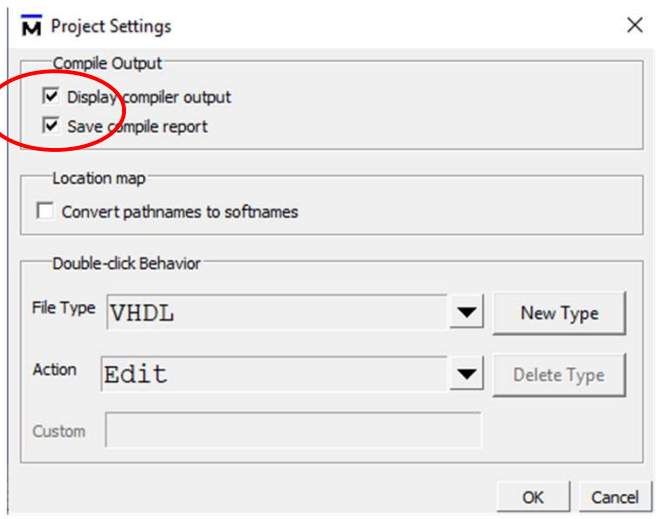
Start ModelSim
To ensure that you can start ModelSim by double-clicking on a ModelSim Project file (.mpf) use the menu help->register filetypes. (you will need to restart the OS)

The testbenches use File I/O which requires VHDL 2008 compatibility. You can check a source file's setting by <right-click>->properties to bring up the dialog.
Make sure that VHDL-2008 is selected.
This may not be necessary because the scripts explicitly request 2008 compilation.

Make sure that the project generates full compiler reports.
With project pane active,
select menu, project->project settings.
Make sure that both compiler flags, are ON

You can create custom keyboard short cuts using the menu window->keyboard shortcuts.
Save the project called ExU.mpf in the folder /Simulation/ .
Quit ModelSim and restart your OS.
From now on always start ModelSim using the project icon, "ExU.mpf".



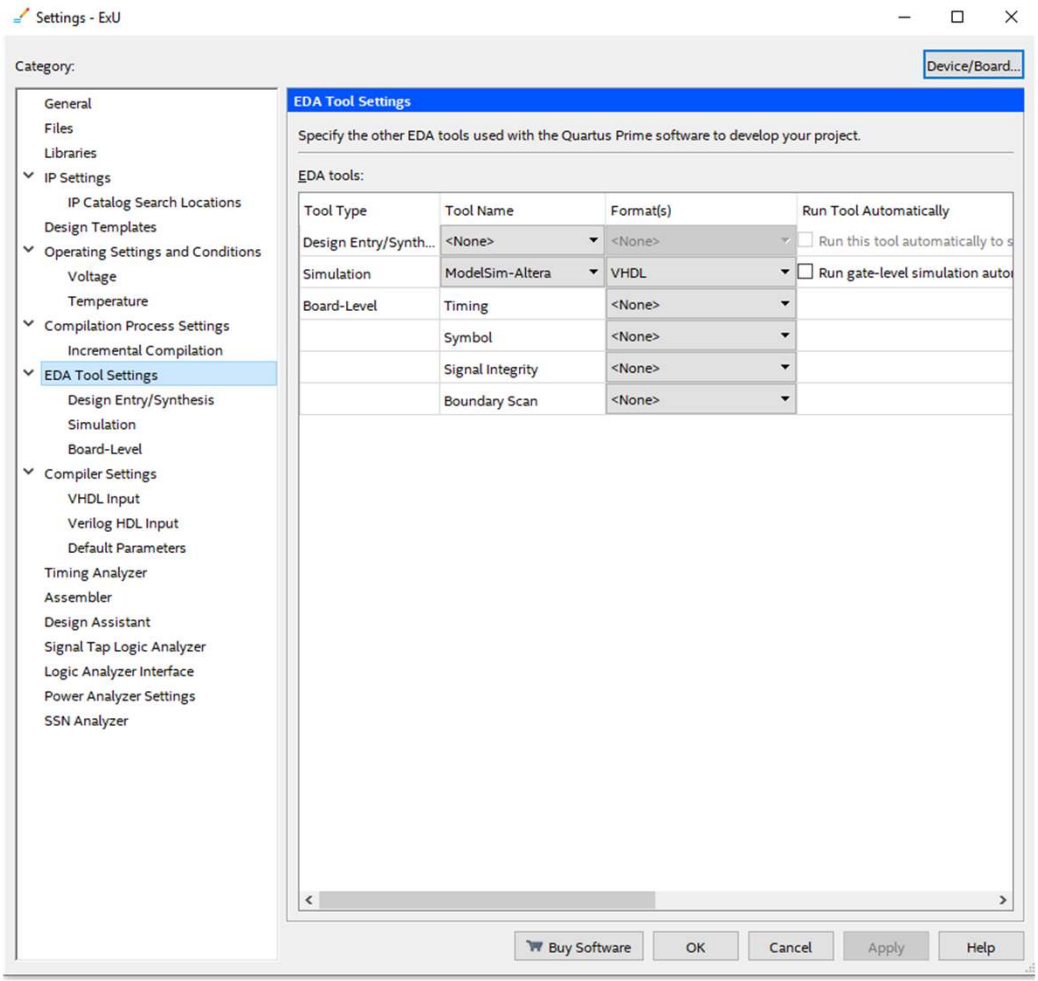


Setting up Quartus:

Use a single Quartus project for all activities. Use the folder /ExU/ as the Quartus project folder.

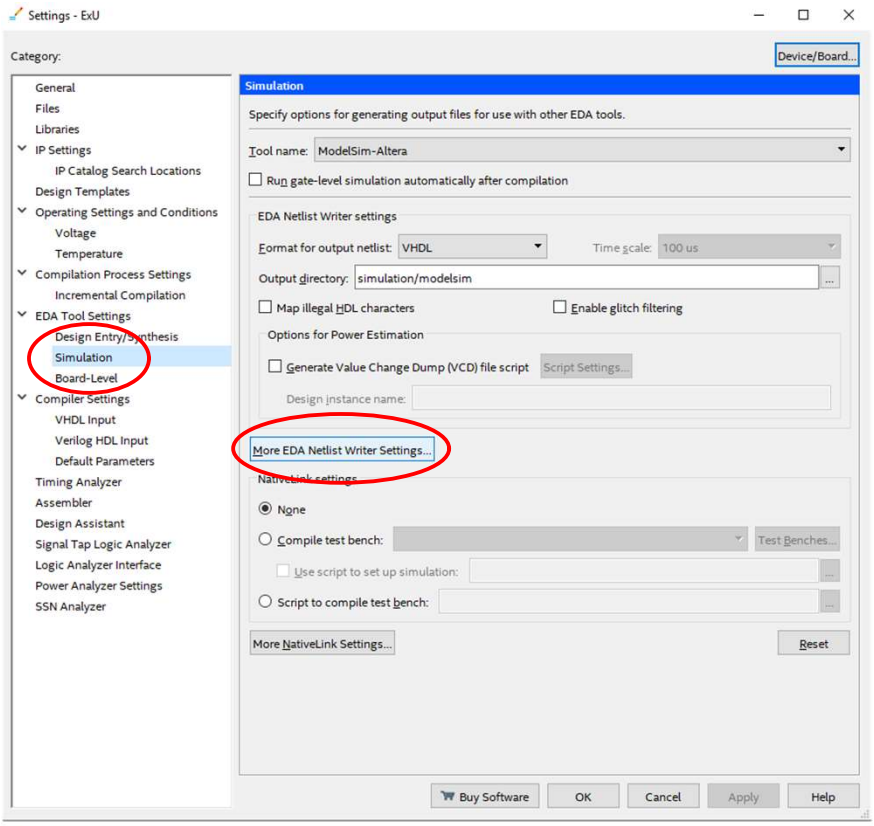
Start Quartus.
Make a new project using a Cyclone IV FPGA. – EP4CE115F29C7

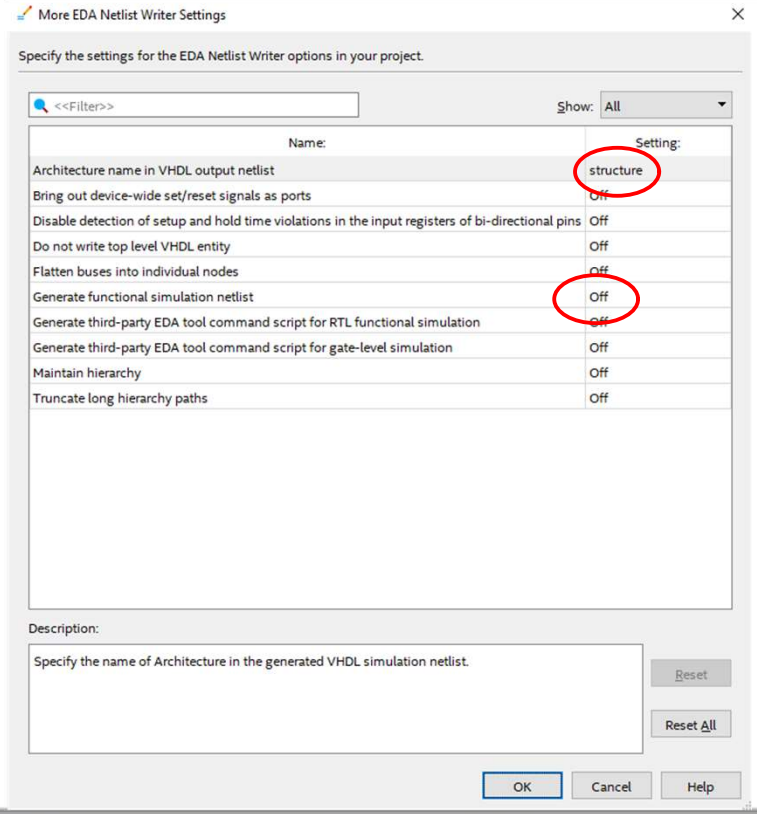
Under the EDA tool settings, choose ModelSim-Altera, format is VHDL
do not enable – "run gate-level
You can also activate this dialog by double-clicking the EDA Netlist writer – Edit settings.



Select EDA tool Settings – Simulation

Click – More EDA Netlist Writer Settings – turn **OFF** "Generate functional simulation netlist"





Close Quartus.

From now on always start Quartus by double-clicking on the project icon "ExU.qpf"

Create a Testbench:

I have created two testbenches on your behalf.
"TBLogicUnit.vhd" - used for both Functional Verification and Timing Simulation.
"TBArithUnit.vhd" - used for both Functional Verification and Timing Simulation.

- The AltB and AltBu are not tested unless the operation is a 64-bit subtraction.

"TBShiftUnit.vhd" - used for both Functional Verification and Timing Simulation.

- Use the same testbench for each of the 6 sets of test vectors, just edit the filename.

"TBExecUnit.vhd" - used for both Functional Verification and Timing Simulation.

"ConfigExU.vhd" - contains eight VHDL configuration declarations.
Once compiled, the VHDL configurations will appear in ModelSim's work library. When starting a simulation, we select a configuration as the top-level, not the testbench.
You don't need to worry about this subtlety as the configurations are automatically selected by the scripts.

The testbenches use file i/o functions that became standard in VHDL-2008.
Study the testbenches as you will be creating your own testbenches one day.
If a measurement fails, you will see a blue message in the ModelSim command window.
A message "Simulation Complete" once the last measurement has run.

Avoid modifying these testbenches.
The only item that may need changing is the constant *PostStimTime*, at the top. This constant is used to wait for a stable result at the output.
When an output result is stable, the propagation delay is measured and the loop continues to the next measurement. You design may have weird periods of stable but incorrect intermediate garbage. It this is the case you can simply extend the *PostStimTime*.

Creating Test Vectors:

The testbenches read input stimuli from rows of a text file. These rows are called Test Vectors.
Using File I/O is quite useful because you can use any programming language to create the test vectors. It is very common to use an interpreted (not compiled) language such as Matlab or Python.

The order of the characters in the row is very important. The data elements must match the statements in the testbench.

- ArithUnit00.tvs – calculates status signals AFTER the MUX.
- ArithUnit01.tvs – calculates status signals BEFORE the MUX.

Logic Unit - Functional Simulation:

I assuming that you have created a VHDL design for the Logic Unit and added the file to the ModelSim project.
The Entity declaration is given
(§Documentation – Documenting Design Entities in the project report)

Add the Testbench and Configuration to the project.

Make sure that the Architecture is called "rtl"
A configuration declaration will bind entities/architectures to components. I have assumed that the architectures used for functional simulation are called "rtl" while the architectures used for timing simulation are called "structure" (as specified by Quartus)

Run the script by typing "do FunctionalLogicUnit.do" in the ModelSim command window.
The script "FunctionalLogicUnit.do"
compiles "LogicUnit.vhd", "TbLogicUnit.vhd" & "ConfigExU.vhd",
loads the simulator using the configuration, FuncLUSim,
sets up wave window using "waveLogicUnit.do",
run the simulation for typically, 6500 ns,
results are written to "FuncLogicUnitTranscript.txt"

Check that no measurement errors have been asserted and that the window only displays "Simulation Complete".
Also check that you can see the last measurement, **#772** at the end of the run.
If you have measurements errors, you will need to debug.

Logic Unit – Saving Functional Waves:

Using the procedure described in §Documentation – Documenting Simulation Waves:
Make images
1) from t = 0 to 70 ns (after measurement #8)
2) from beginning of measurement #458 to end of measurement #464
3) from beginning of measurement #770 till 5 ns after the start of measurement #772.

Save these images using the filenames "waveFLU1.png", "waveFLU2.png" & "waveFLU3.png"

Logic Unit – Synthesis:

Start Quartus using the project file icon.
Add your design entity, "LogicUnit.vhd" to the project. Set this entity as the top-level entity.
Run a full synthesis, <ctrl-L>
Resolve any issue that make you unhappy. You will need to correct your VHDL sourcecode if your design is not synthesisable. Once these changes are made you must return to ModelSim and repeat the procedures for functional simulation.

Once the your design is both functional and synthesisable, run the full synthesis. <ctrl-L>

Verify that the folder ExU/Simulation/ModelSim/ now contains "ExU.vho" and "ExU_vhd.sdo".
The .vho files are VHDL models for your design after place-and-route on a Cyclone IV FPGA. The .sdo files contain the timing information for the logic elements and wires that were used by your design.
Rename these files, "LogicUnit.vho" and "LogicUnit.sdo".

Rename the summary files, "ExU.map.summary" and "ExU.fit.summary" to "LogicUnit.map.summary" and "LogicUnit.fit.summary".
Move these files to the /Documentation/ folder.
(§Documentation – Documenting Synthesis & Fitting)

Every time we use Quartus to create post-fit models we will need to rename the files as Quartus will always use the project name for these files.

Using the RTL netlist viewer capture images of the RTL synthesised circuit.
Using the post-fitting Netlist viewer capture images of circuit using FPGA elements.
(§Documentation – Documenting Synthesis & Fitting)

Using any text editor, open the netlist and verify that Quartus named the architecture "structure".
Close the netlist file and close Quartus.

Logic Unit - Timing Simulation:

Start ModelSim.
Add the file "LogicUnit.vho" to the project.
Run the script "TimingLogicUnit.do"

Run the script by typing "do TimingLogicUnit.do" in the ModelSim command window.
The script "TimingLogicUnit.do"

compiles "LogicUnit.vho", "TbLogicUnit.vhd" & "ConfigExU.vhd",
loads the simulator using the configuration, TimeLUSim,
sets up wave window using "waveLogicUnit.do",
run the simulation for typically, 15600 ns,
results are written to "TimeLogicUnitTranscript.txt"

By observing the blue text in the command window, you should see the message, "Back Annotation Successfully Completed" when the simulator loads and when it is restarted.
This means that the .sdo file was successfully read and the timing information was mapped to the VHDL generic within the timing model.

Check that no measurement errors have been asserted and that the window only displays "Simulation Complete".
Also check that you can see the last measurement, #772 at the end of the run.
If you have measurements errors, you will need to debug.

The simulation is now running with timing delays that were accurately measured by the manufacturer of the Cyclone IV FPGA.

Logic Unit – Saving Timing Waves:

Using the procedure described in §Documentation – Documenting Simulation Waves:
Make images

1) from t = 0 to a few ns into the beginning of measurement #2
   Place two cursors that measure the propagation delay of measurements #1.
2) from beginning of measurement #400 to end of measurement #416
   Delete all cursors.
3) from beginning of measurement #772 till 5 ns after the end of measurement #772.
   Place two cursors that measure the propagation delay of measurements #772.

Save these images using the filenames "waveTLU1.png", "waveTLU2.png" & "waveTLU3.png"

Arithmetic Unit - Functional Simulation:

Design the Arithmetic Unit. (Lecture 7 – handout page 6.)
Design a simple ripple adder in a separate file called "Adder.vhd". We can change the design later if we find that we need a faster adder. Instantiate the adder within the ArithUnit entity.
The Entity declarations are given
(§Documentation – Documenting Design Entities in the project report)
Make sure that the Architectures are called "rtl"
Add both entities to the ModelSim project.

Add the Testbench "TBArithUnit.vhd" to the ModelSim project.

Run the script by typing "do FunctionalArithUnit.do" in the ModelSim command window.
The script "FunctionalArithUnit.do"
compiles "Adder.vhd", "ArithUnit.vhd", "TbArithUnit.vhd" & "ConfigExU.vhd",
loads the simulator using the configuration, FuncAUSim,
sets up wave window using "waveArithUnit.do",
run the simulation for typically, 1220 ns,
results are written to "FuncArithUnitTranscript.txt"

Check that no measurement errors have been asserted and that the window only displays "Simulation Complete".
Also check that you can see the last measurement, **#120** at the end of the run.
If you have measurements errors, you will need to debug.

Arithmetic Unit – Saving Functional Waves:

Using the procedure described in §Documentation – Documenting Simulation Waves:
Make images
1) from t = 0 to 76 ns (after measurement #6)
2) from beginning of measurement #61 to end of measurement #66
3) from beginning of measurement #118 till 5 ns after the start of measurement #120.

Save these images using the filenames "waveFAU1.png", "waveFAU2.png" & "waveFAU3.png"

Arithmetic Unit – Synthesis:

Start Quartus using the project file icon.
Add your design entities, "ArithUnit.vhd" and "Adder.vhd" to the project.
Set ArithUnit as the top-level entity.
Run a full synthesis, <ctrl-L>

Resolve any issue that make you unhappy. You will need to correct your VHDL sourcecode if your design is not synthesisable. Once these changes are made you must return to ModelSim and repeat the procedures for functional simulation.

Once the your design is both functional and synthesisable, run the full synthesis. <ctrl-L>

Verify that the folder ExU/Simulation/ModelSim/ now contains "ExU.vho" and "ExU_vhd.sdo".
Rename these files, "ArithUnit.vho" and "ArithUnit.sdo".

Rename the summary files, "ExU.map.summary" and "ExU.fit.summary" to "ArithUnit.map.summary" and "ArithUnit.fit.summary".
Move these files to the /Documentation/ folder.
(§Documentation – Documenting Synthesis & Fitting)

Using the RTL netlist viewer capture images of the RTL synthesised circuit.
Using the post-fitting Netlist viewer capture images of circuit using FPGA elements.
(§Documentation – Documenting Synthesis & Fitting)

Using any text editor, open the netlist and verify that Quartus named the architecture "structure".
Close the netlist file and close Quartus.

Arithmetic Unit - Timing Simulation:

Start ModelSim.
Add the file "ArithUnit.vho" to the project.
Run the script "TimingArithUnit.do"

Run the script by typing "do TimingArithUnit.do" in the ModelSim command window.
The script "TimingArithUnit.do"

compiles "ArithUnit.vho", "TbArithUnit.vhd" & "ConfigExU.vhd",
load simulator using the configuration, TimeAUSim,
sets up wave window using "waveArithUnit.do",
run the simulation for typically, 5740 ns,
results are written to "TimeArithUnitTranscript.txt"

By observing the blue text in the command window, you should see the message, "Back Annotation Successfully Completed" when the simulator loads and when it is restarted.
This means that the .sdo file was successfully read and the timing information was mapped to the VHDL generic within the timing model.

Check that no measurement errors have been asserted and that the window only displays "Simulation Complete".
Also check that you can see the last measurement, **#120** at the end of the run.
If you have measurements errors, you will need to debug.

The simulation is now running with timing delays that were accurately measured by the manufacturer of the Cyclone IV FPGA.

Arithmetic Unit – Saving Timing Waves:

Using the procedure described in §Documentation – Documenting Simulation Waves:
Make images

1) from t = 0 to a few ns into the beginning of measurement #2
   Place two cursors that measure the propagation delay of measurements #1.
2) from beginning of measurement #61 to end of measurement #66
   Place two cursors that measure the propagation delay of measurements #65.
   Place two cursors that measure the propagation delay of measurements #66.
3) from beginning of measurement #120 till 5 ns after the end of measurement #120.
   Place two cursors that measure the propagation delay of measurements #120.

Save these images using the filenames "waveTAU1.png", "waveTAU2.png" & "waveTAU3.png"

Final Project – Part 2

Testing & Documenting: Part 2

The Shift Unit:

Functional Verification.

- Use the scripts "FunctionalShiftUnit.do" and "waveShiftUnit.do".
- Run and document **SIX** tests, one for each set of test vectors.

Synthesis – using the prefix "ShiftUnit"

- Rename the netlist and .SDO file,
- Rename the summary files.

Timing Verification.

- Use the scripts "TimingShiftUnit.do" and "waveShiftUnit.do".
- Run and document **SIX** tests, one for each set of test vectors.

The Execution Unit:

Functional Verification.

- Use the scripts "FunctionalExecUnit.do" and "waveExecUnit.do".
- Run and document **ONE** test, using "ExecUnit00.tvs".

Synthesis – using the prefix "ExecUnit"

- Rename the netlist and .SDO file,
- Rename the summary files.

Timing Verification.

- Use the scripts "TimingExecUnit.do" and "waveExecUnit.do".
- Run and document **ONE** test, using "ExecUnit00.tvs".

For each run, capture two images

one image showing the results of two intermediate measurements, and

one image showing the results of the last three measurements.

Save your images using the naming conventions previously described.