EC605: Computer Engineering Fundamentals Fall 2024

Lab 5: Single Cycle RISC-V Datapath

Goals

- Gain experience with digital design and Verilog.
- Gain understanding of computer organization via hands-on implementation of a RISC-V datapath.
- Gain knowledge on building complex, multi-module structures by mapping out parts in block diagrams.

Overview

In this lab you will design a single-cycle RISC-V-like datapath, which supports a variation on a subset of the RV32I Base Integer instructions. The structure of the datapath should follow the one described in the textbook and lectures.

You must completely follow the definition in this handout for the design.

Deliverables

We will follow the regular lab policies for lab 5: work in pairs, and have only one partner submit the required deliverables. The deliverables of this lab will be split into three milestones:

First Milestone Deliverables (Task 1 pt. 1)

On Blackboard, submit completed modules and working testbenches for the following portions of the datapath from task 1. Submit both Verilog files of the modules and testbenches:

- Instruction memory – store specified program and read specified instructions
- Register file – read and write operations
- Data memory – read and write operations
- ALU – all operations and zero flag

Second Milestone Deliverables

Using all the modules from task 1, draw a complete block diagram of the single cycle datapath with the necessary wires. Make sure to label your wires with names and the number of bits used that allow quick recognition of their roles. Submit the diagram as a PDF.

Final Deliverables (Task 1 pt. 2 + Task 2)

- Submit your Verilog code and testbenches for all datapath components from task 1 and the integrated datapath from task 2. Also submit a pdf with a waveform and a description of the tests done for each component and the top module, and a description of the design choices that you made. In your pdf, also include your block diagram from milestone 2.
- Sign-up to demo your datapath to a TA. Come prepared to answer questions on your design.

The simplified RISC-V instruction format is as follows:

R-type:

func7	rs2	rs1	func3	rd	opcode
31 25	24 20	19 15	14 12	11 7	6 0

I-type:

Imm[11:0]	rs1	func3	rd	opcode
31 20	19 15	14 12	11 7	6 0

S-type:

imm[11:5]	rs2	rs1	func3	imm[4:0]	opcode
31 25	24 20	19 15	14 12	11 7	6 0

B-type:

imm[12]	imm[10:5]	rs2	rs1	func3	imm[4:1]	imm[11]	opcode
31	30 25	24 20	19 15	14 12	11 8	7	6 0

U-type:

imm[31:12]	rd	opcode
31 12	11 7	6 0

J-type:

imm[20]	imm[10:1]	imm[11]	imm[19:12]	rd	opcode
31	30 21	20	19 12	11 7	6 0

Following is the instructions which will be used in this lab (Note: Opcode, Funct3 and Funct7 are not the same as Green Card):

Instruction	OPCODE	FUNCT3	FUNCT7
AND	7'b1110011	3'b000	7'b0000000
ADD	7'b1110011	3'b001	7'b0000000
OR	7'b1110011	3'b010	7'b0000000
SUB	7'b1110011	3'b001	7'b0100000
XOR	7'b1110011	3'b100	7'b0000000
SRA	7'b1110011	3'b101	7'b0000000
SLL	7'b1110011	3'b110	7'b0000000
SLT	7'b1110011	3'b111	7'b0000000
BEQ	7'b1101011	3'b000	-
BLT	7'b1101011	3'b001	-
SW	7'b1100011	3'b010	-
LW	7'b1000011	3'b010	-
LUI	7'b0110000	-	-
ADDI	7'b0011111	3'b000	-
ORI	7'b0011111	3'b001	-
XORI	7'b0011111	3'b010	-

Task 1: Datapath Components

Part 1: Basic modules

First we introduce the fundamental components of the datapath. These are the core elements that directly manipulate the data during the execution of a program. Implement and/or test these modules and submit them for milestone 1.

1. **Instruction Memory (IMem.v)**
   Modify the provided sample instructions in the initial block to include a complete test assembly program. Note that the instruction memory is read-only while implementing. Simulate the instruction memory module and make sure that it operates correctly (for milestone 1, all you need to do is ensure that you can store an initial set of instructions and output from the correct input address, not run a full program). Instructions listed above should all be included.

   Note: You may create a large 32-bit immediate by using lui+ori.

2. **Register File (Register_File.v)**
   Simulate the register file module and make sure that it operates correctly. Note that the first register (x0) has been set as zero and cannot be modified. Confirm that you can read from and write to chosen input register addresses.

3. **ALU (no provided .v file)**
   Implement and test a 32-bit ALU, supporting the following operations:

Opcode	Operation
0000	AND
0001	OR
0010	XOR
0011	Arithmetic shift right
0100	Logical shift left
0101	Subtract
0110	Add
0111	Set less than
rest	Any operation that you may need for implementation

   The ALU also has a Zero output flag. The input should be the 4 bit ALU OPCODE, A and B and the outputs are the 32 bit ALU output and zero flag. **Hint:** you should be able to build on your implementation from lab 2, but only need to include the zero flag.

4. **Data Memory (no provided .v file)**
   Implement and test a data memory module. This module should use the same structure to read and write data as the register file. Refer to the lecture slides to determine the necessary inputs and outputs for the data memory. A good size would be twice the size of the register file. **Hint:** Declaring an array of arrays in Verilog is done like so:

reg [3:0] **array** [15:0] *//array of size 16 containing 4-bit vectors*

Part 2: Additional modules

In addition to the primary modules for part 1, we will need to implement several control modules to create a fully functioning datapath. You do not need to prepare these modules for milestone 1. You should understand them and include them in your diagram for milestone 2. You will need to submit these .v files and any additional testbenches for the final submission, and should incorporate them into the top module for your datapath.

5. **Program Counter (no provided .v file)**
   We need a method to read the stored instructions from the instruction memory, so that the program can execute. Design a program counter that outputs the next instruction address on each clock edge. You should make a separate module file for the PC, and connect it to the instruction memory. When the PC is reset, it should output the address of the first instruction

   Note: In the standard datapath, the PC counter is incremented by 4. For this lab, increment the PC by 1 instead of 4.

6. **Control logic (no provided .v file)**
   Implement and test a control unit, which combines the functionality of the central control unit and the ALU control unit. The control unit should output all the necessary control signals, the register addresses to read/write, and the immediate signal.

7. **Branch control logic (no provided .v file)**
   Implement and test a branch control unit which determines if a branch should be taken and correctly outputs the next instruction address to the program counter. You will have to integrate this module with the program counter to correctly determine the next instruction address to read.

Task 2: Integrated Datapath (Datapath.v file)

1. In a new project, implement and test a working datapath that instantiates and connects all the datapath components. First draw your proposed design for milestone 2 with all connecting wires labelled with a name and their corresponding number of bits, and make sure you understand how all the modules fit together before you design the top module in Verilog. Refer to the datapath design in the lecture notes. Also note that the order in which modules are presented in task 1 does not reflect the order in which they are connected in the datapath.

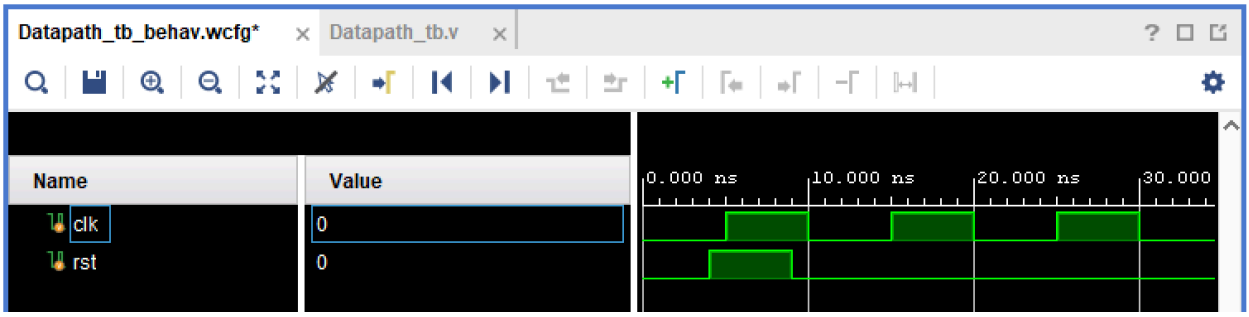
2. The datapath only takes clk and rst as inputs and runs the instructions stored in instruction memory. Instantiate all the modules from task 1 and connect them appropriately using internal wires. A template is provided in the file, including a place to instantiate each module with the necessary wires declared. Make any changes to the datapath as needed for your version of the design and your modules from task 1.

3. Make a testbench for the datapath and verify its functionality. Add internal signals to the waveform as necessary to determine if the datapath executes the program as intended. See below for advice on using a testbench with a large number of internal signals.

4. To demo the testbench, you should be able to explain the assembly program you wrote and demonstrate on the waveform that the contents of the register file and data memory are updated correctly. Additionally you should add internal signals from various parts of the datapath and explain how the instruction is processed at different stages of its execution.

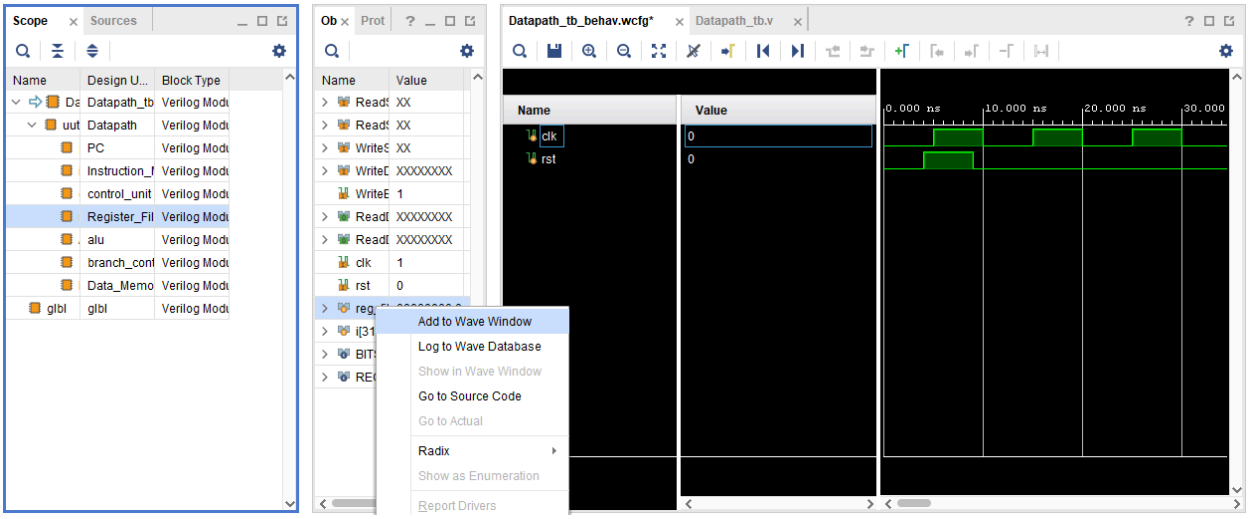
Appendix: Using the Testbench with Internal Signals

Because the datapath has only clk and rst as inputs, you will have to add internal signals to understand whether the datapath works as intended and debug it as necessary. Due to the size of this module, this can be quite complicated, and it is important to stay organized.

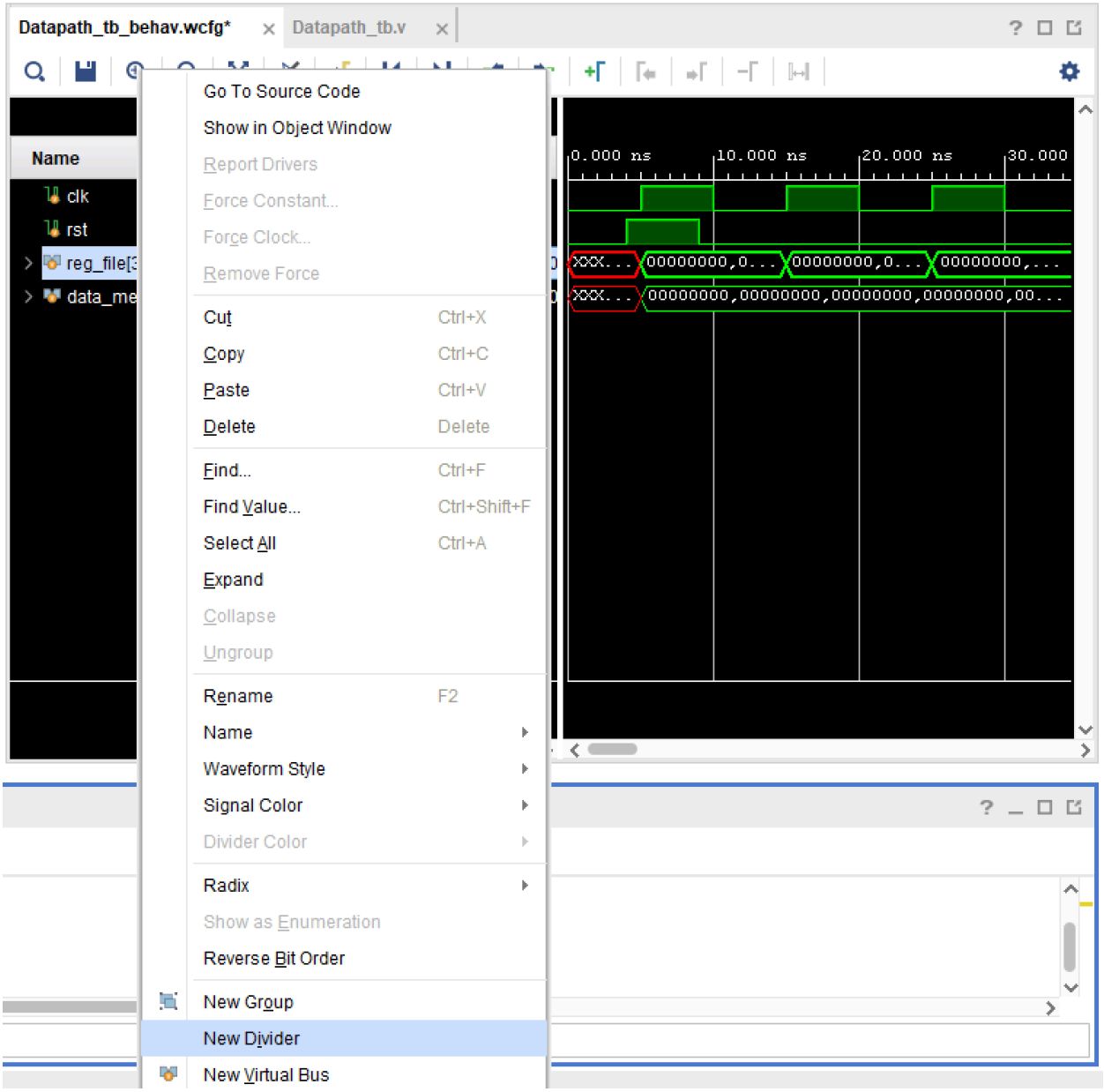


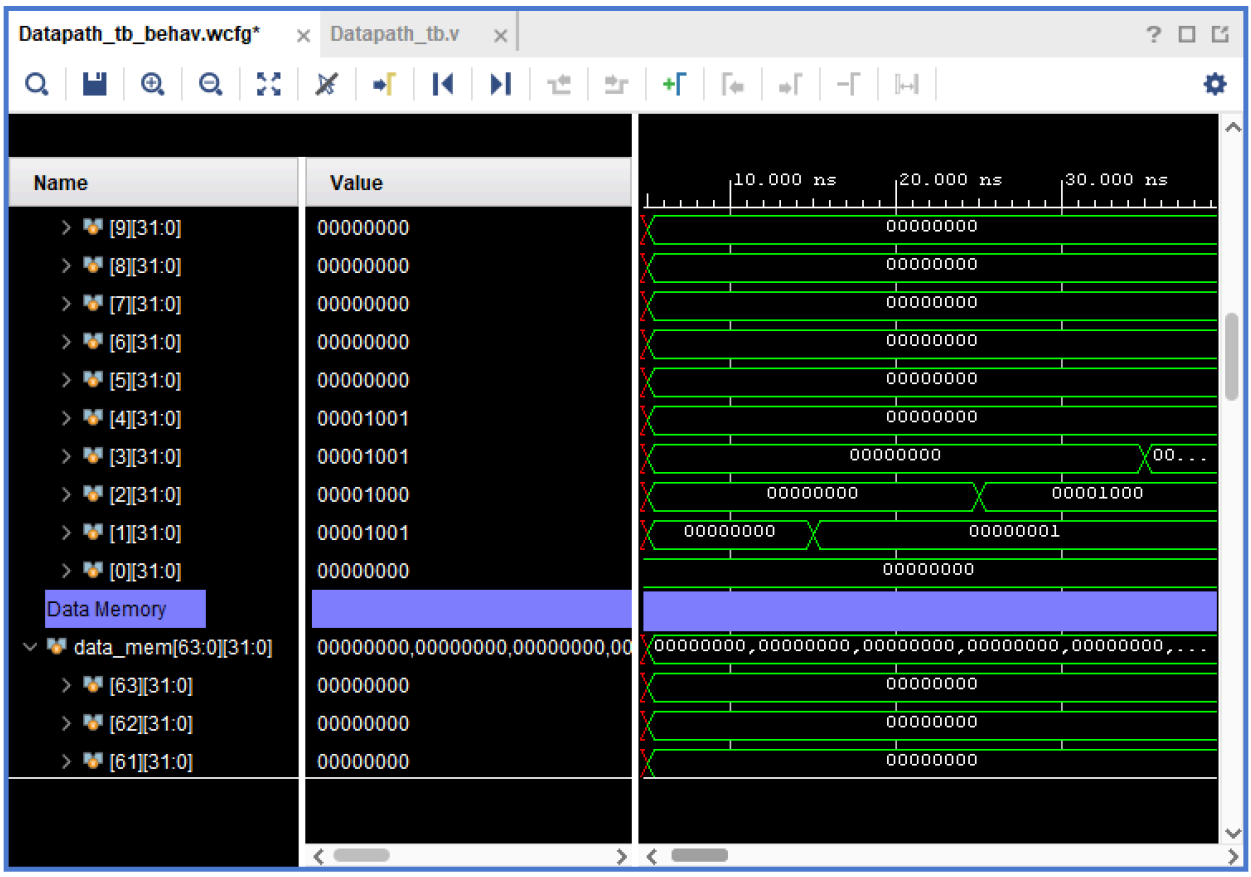
The first step to check if the datapath is working is to add the register file and data memory to the waveform. If the contents of the registers and memory are updated as expected for the assembly program you write, your design is most likely working.

To add internal signals to the waveform, expand the list of modules instantiated by the testbench under the “Scope” tab to the left of the waveform. Select the register file module, and under the “Objects” tab, right click the register file signal and select “Add to Wave Window.” Similarly, add the data memory to the waveform.



To keep the waveform organized, use dividers to group the signals you add to the waveform according to the module they're taken from. To add a divider to the waveform, right click the signal below which you want to add a divider and select "New Divider." You can label the divider to organize signals by module.




Datapath_tb_behav.wcfg*
Datapath_tb.v
Name
Value
10.000 ns
20.000 ns
30.000 ns
[9][31:0]
00000000
[8][31:0]
00000000
[7][31:0]
00000000
[6][31:0]
00000000
[5][31:0]
00000000
[4][31:0]
00001001
[3][31:0]
00001001
[2][31:0]
00001000
[1][31:0]
00001001
[0][31:0]
00000000
Data Memory
data_mem[63:0][31:0]
00000000,00000000,00000000,00
[63][31:0]
00000000
[62][31:0]
00000000
[61][31:0]
00000000

If the register file and data memory are not functioning correctly, add signals from one module at a time, and try to isolate the point where the datapath is failing, or look back at your code.

When the datapath is working, you should be able to add all the signals to the waveform and follow the path of an instruction as it moves through the datapath.

Note that as you add new signals and dividers to the waveform, you can save these formatting changes. If you need to continue working on the waveform at a later time, all the formatting will be reloaded automatically next time you run the simulation.