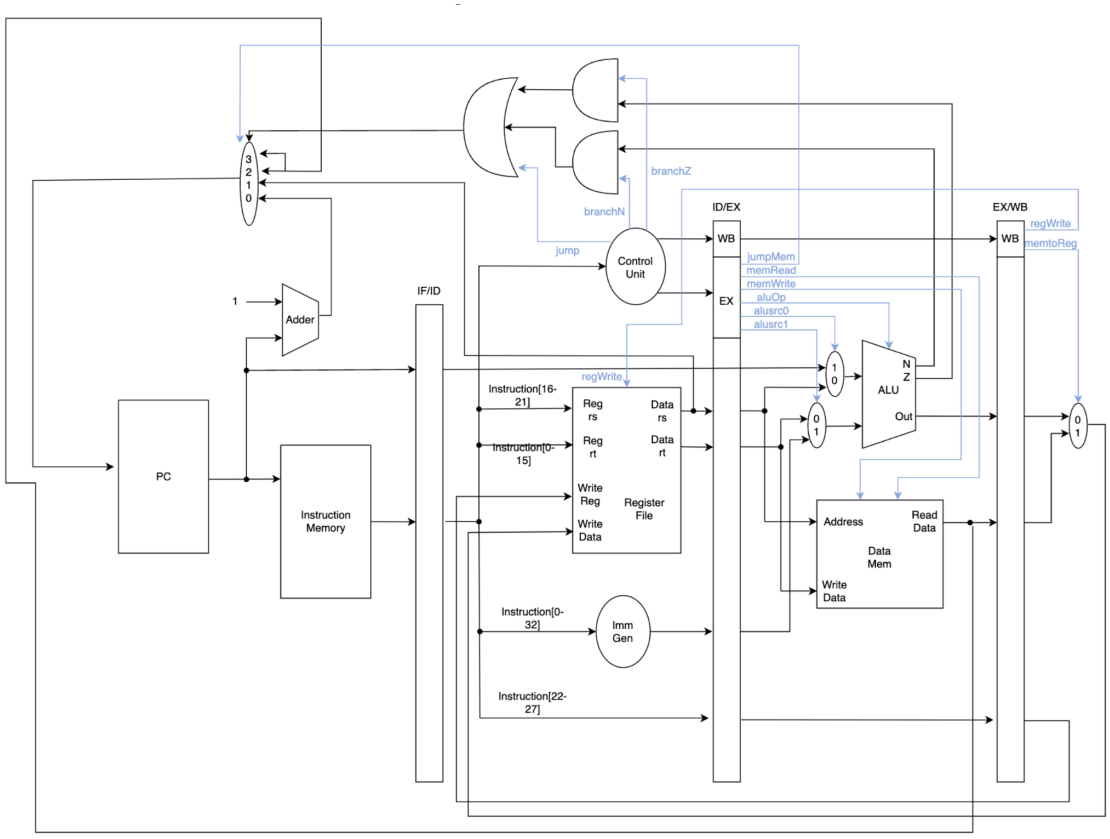
Eason Liu
Jason Fong
June 9, 2023

Final Project Report

1. Abstract

The Coen 122 final project task was to construct and design a structural model of a pipelined CPU with 13 instructions using Verilog. Given SCU ISA(SCU Instruction set architecture) we were supposed to design a working 32-bit pipeline CPU. We achieved this by breaking down the lab into individual parts and combining modules together to test the project with a test-bench. We were also able to learn how to construct these units in Verilog throughout each lab to gain a better understanding of how each piece connects with each other to perform more complex instructions.As a result, we were able to construct a working Pipeline CPU.

2. Detailed Description of CPU - Include final datapath and control truth table



The CPU updates through the positive edge and negative of the clock which works on the PC module and the address is also kept track of which feeds into the Instruction Memory. The IF/ID buffer will hold the IM and the original PC address in case of a jump instruction. Following the IF/ID buffer, data is fed into the register, immediate generator, control unit, and next buffer. The Register unit will take in the addresses of registers from the instructions and spit out data into the

ID/EX buffer. The control also feeds data into the ID/EX buffer as well. The immediate generator also transmutes the given data and feeds into the ID/EX buffer. Next, the ID/EX buffer feeds various control signals into different units and muxes. In this stage, we have data memory and ALU. The ALU will feed back and decide if we have to jump through a segment of OR and AND gates, and mux control. The data memory will feed its data into both the mux and next buffer. Lastly, the EX/WB buffer has two control signals and feeds two signals, one into a mux that goes into write data; the other into the write reg.

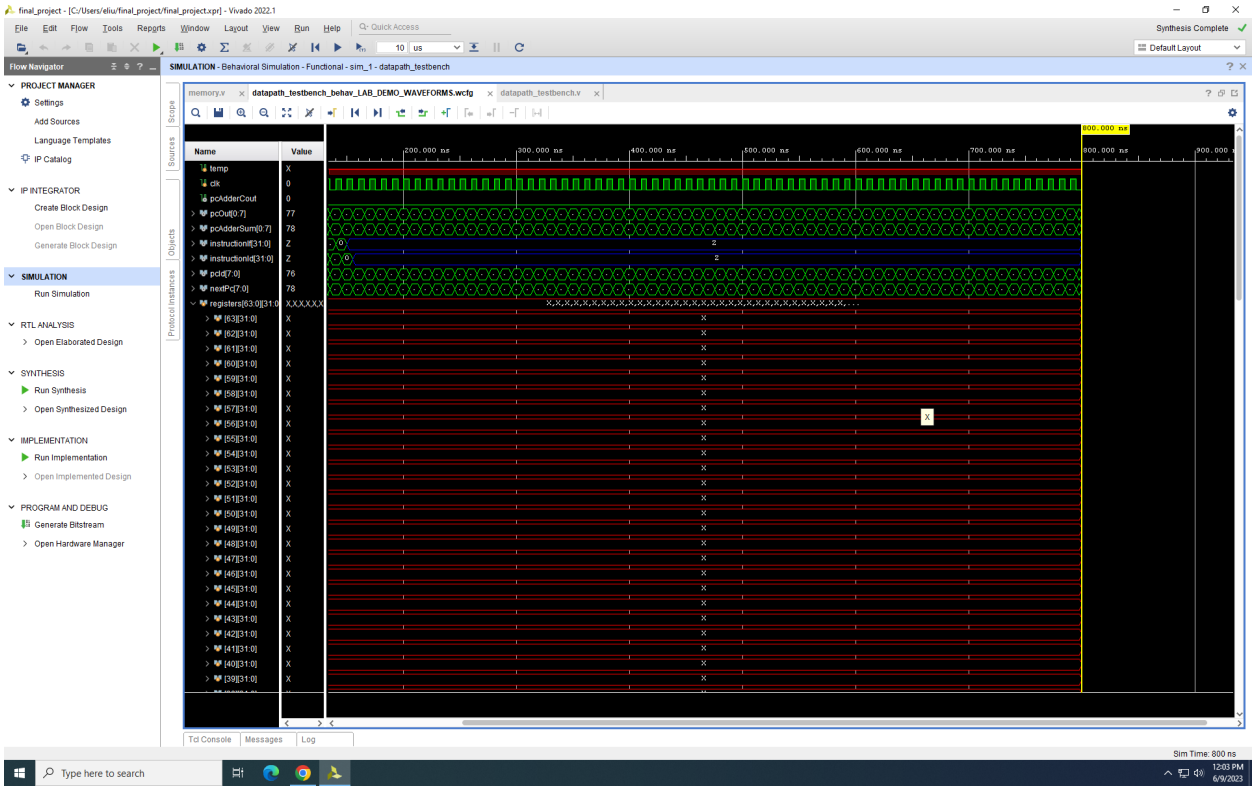
Truth Table

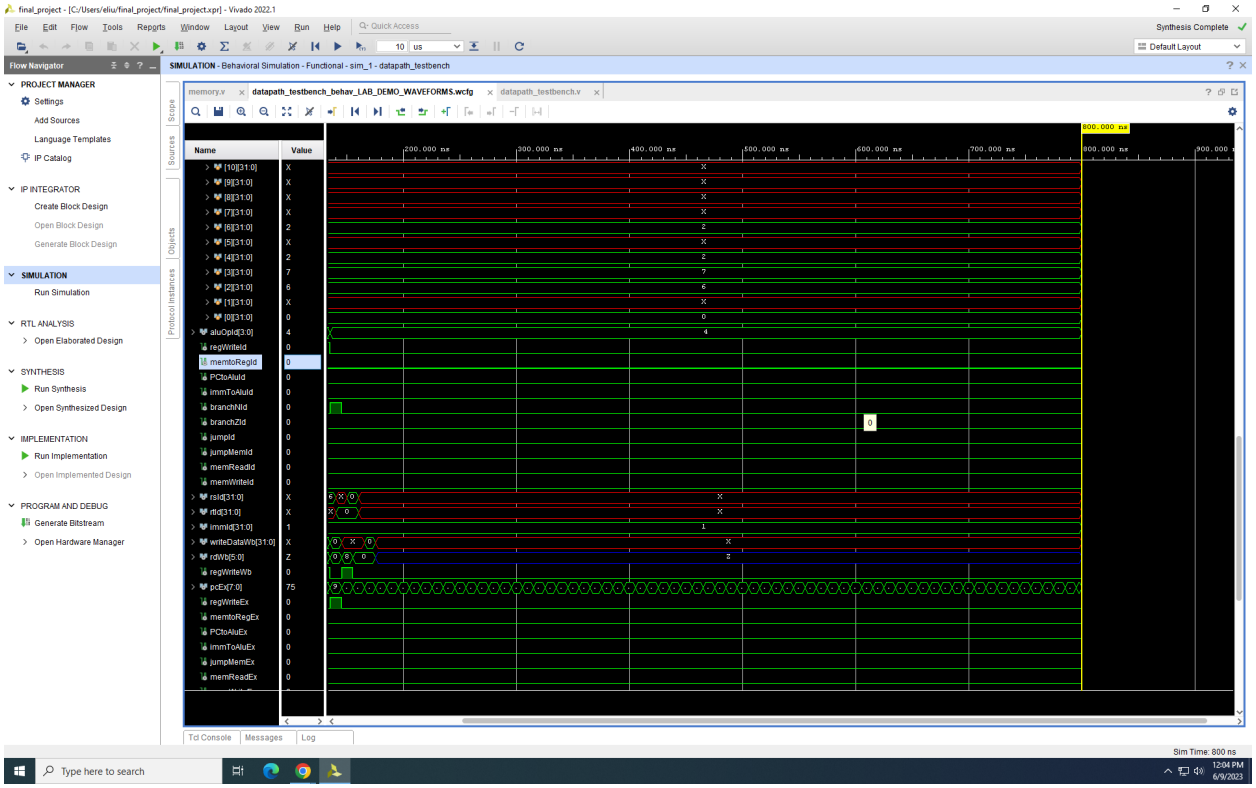
Instruction	Opcode	RegWrite	memtoReg	pctoALU	immtoALU	branchN	branchZ	jump	jumpMem	memRead	memWrite	aluOp
NOP	0000	0	0	0		0	0	0	0	0	0	0100
Store	0011	0	0	0	0	0	0	0	0	0	1	0100
Add	0100	1	0	0	0	0	0	0	0	0	0	0000
Increment	0101	1	0	0	1	0	0	0	0	0	0	0001
Negate	0110	1	0	0	0	0	0	0	0	0	0	0010
Subtract	0111	1	0	0	0	0	0	0	0	0	0	0011
Jump	1000	0	0	0	0	0	0	1	0	0	0	0100
Branch if Z	1001	0	0	0	0	0	1	0	0	0	0	0100
JumpMem	1010	0	0	0	0	0	0	0	1	1	0	0100
Branch if N	1011	0	0	0	0	1	0	0	0	0	0	0100
Load	1110	1	1	0	0	0	0	0	0	1	0	0100
Save PC	1	0	1	1	0	0	0	0	0	0	0	0001

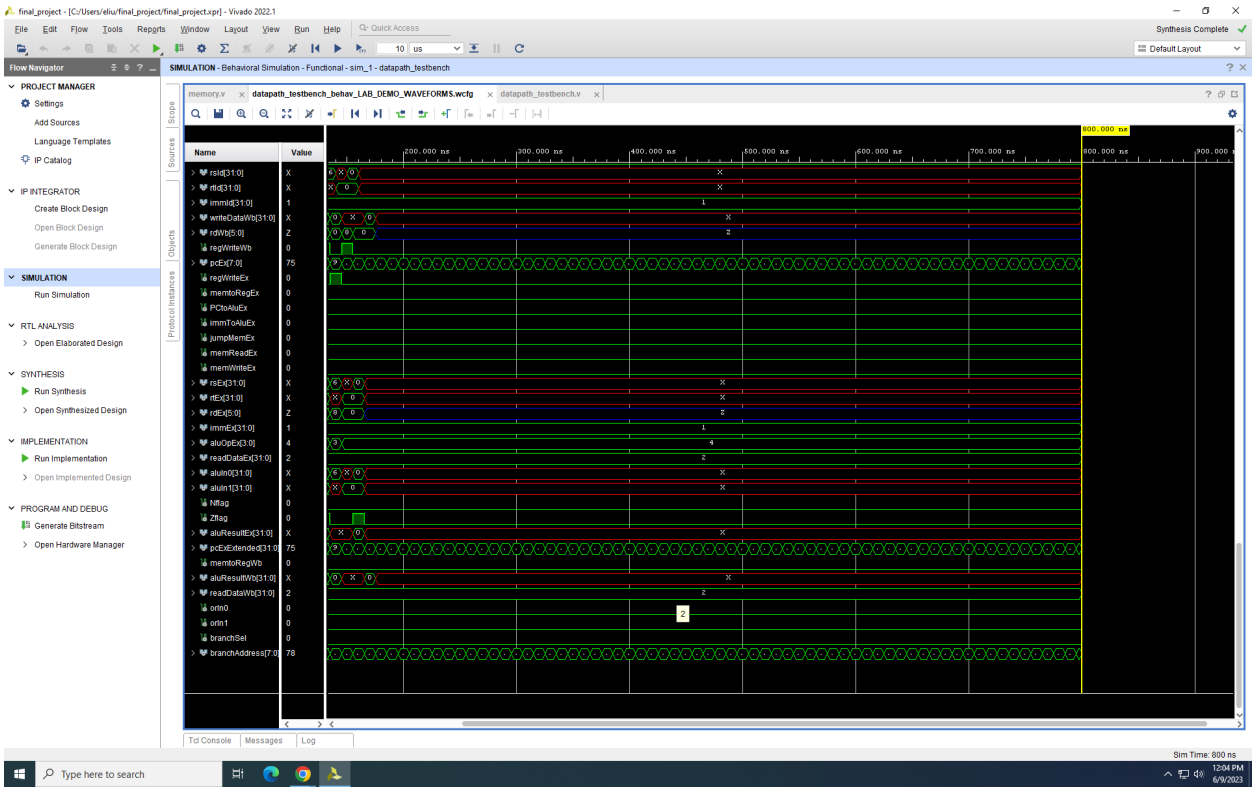
3. Simulation Verification

Below are a few waveforms and descriptions for the operation of at least two different instructions and a waveform and description for the sum calculation

assign instructions[1] = 32'b01010000010000100000000000000011;
//x1, x2, 3 increment
assign instructions[3] = 32'b01100000100010110000000000000000;
//negate //rd: 2, rs: 11



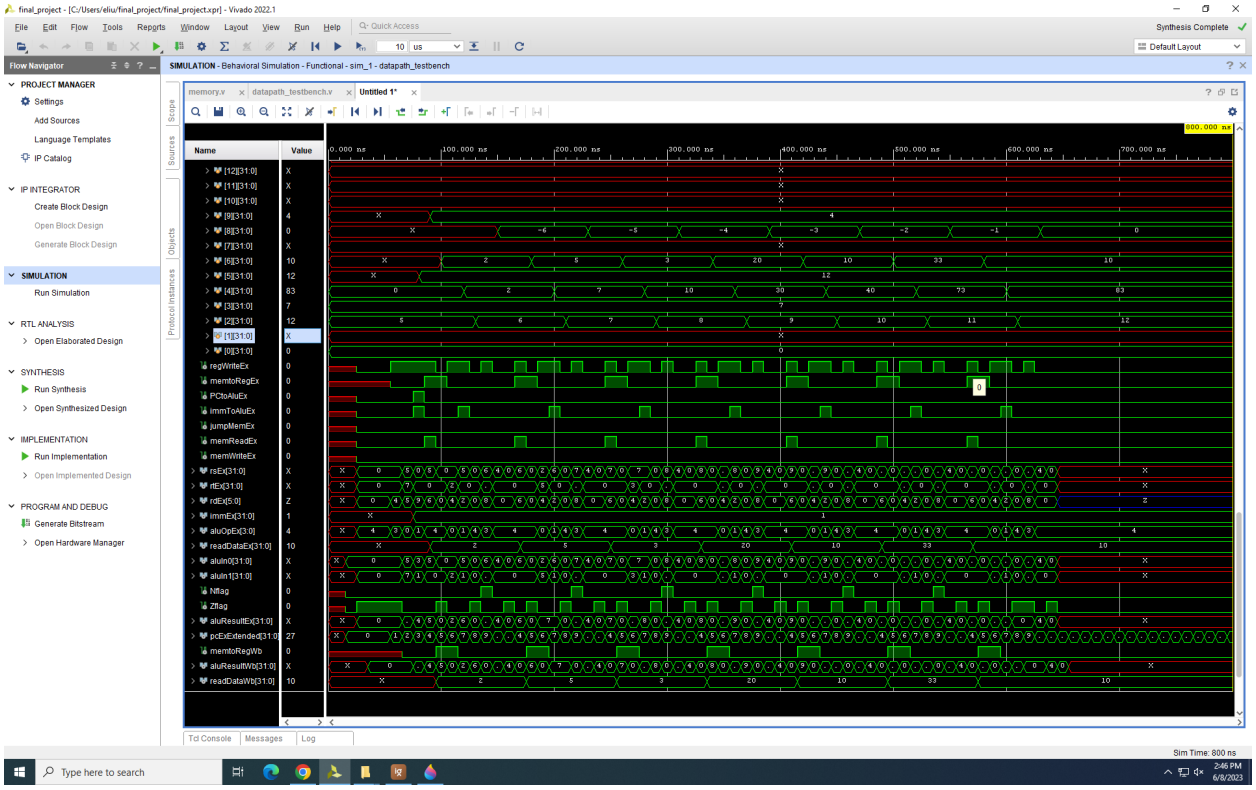

final_project - [C:/Users/eliu/final_project/final_project.xpr] - Vivado 2022.1
File Edit Flow Tools Reports Window Layout View Run Help
Quick Access
Synthesis Complete
Default Layout
Flow Navigator
PROJECT MANAGER
Settings
Add Sources
Language Templates
IP Catalog
IP INTEGRATOR
Create Block Design
Open Block Design
Generate Block Design
SIMULATION
Run Simulation
RTL ANALYSIS
Open Elaborated Design
SYNTHESIS
Run Synthesis
Open Synthesized Design
IMPLEMENTATION
Run Implementation
Open Implemented Design
PROGRAM AND DEBUG
Generate Bitstream
Open Hardware Manager
SIMULATION - Behavioral Simulation - Functional - sim_1 - datapath_testbench
memory.v
datapath_testbench_behav_LAB_DEMO_WAVEFORMS.wcfg
datapath_testbench.v
Name
Value
Tcl Console
Messages
Log
Sim Time: 800 ns
Type here to search
12:04 PM
6/9/2023

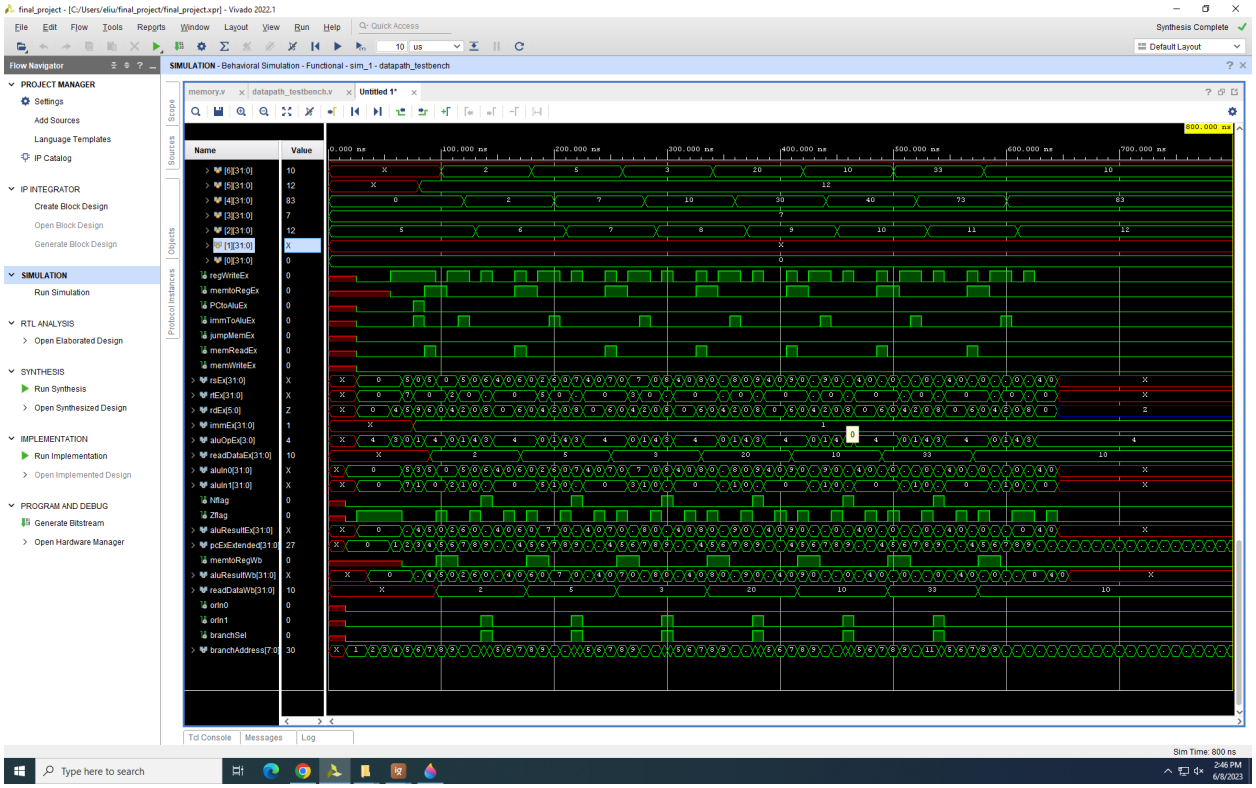

final_project - [C:/Users/eliu/final_project/final_project.xpr] - Vivado 2022.1
File Edit Flow Tools Reports Window Layout View Run Help
Quick Access
Synthesis Complete
Default Layout
Flow Navigator
PROJECT MANAGER
Settings
Add Sources
Language Templates
IP Catalog
IP INTEGRATOR
Create Block Design
Open Block Design
Generate Block Design
SIMULATION
Run Simulation
RTL ANALYSIS
Open Elaborated Design
SYNTHESIS
Run Synthesis
Open Synthesized Design
IMPLEMENTATION
Run Implementation
Open Implemented Design
PROGRAM AND DEBUG
Generate Bitstream
Open Hardware Manager
SIMULATION - Behavioral Simulation - Functional - sim_1 - datapath_testbench
memory.v
datapath_testbench_behav_LAB_DEMO_WAVEFORMS.wcfg
datapath_testbench.v
Name
Value
Tcl Console
Messages
Log
Sim Time: 800 ns
Type here to search
12:04 PM
6/9/2023

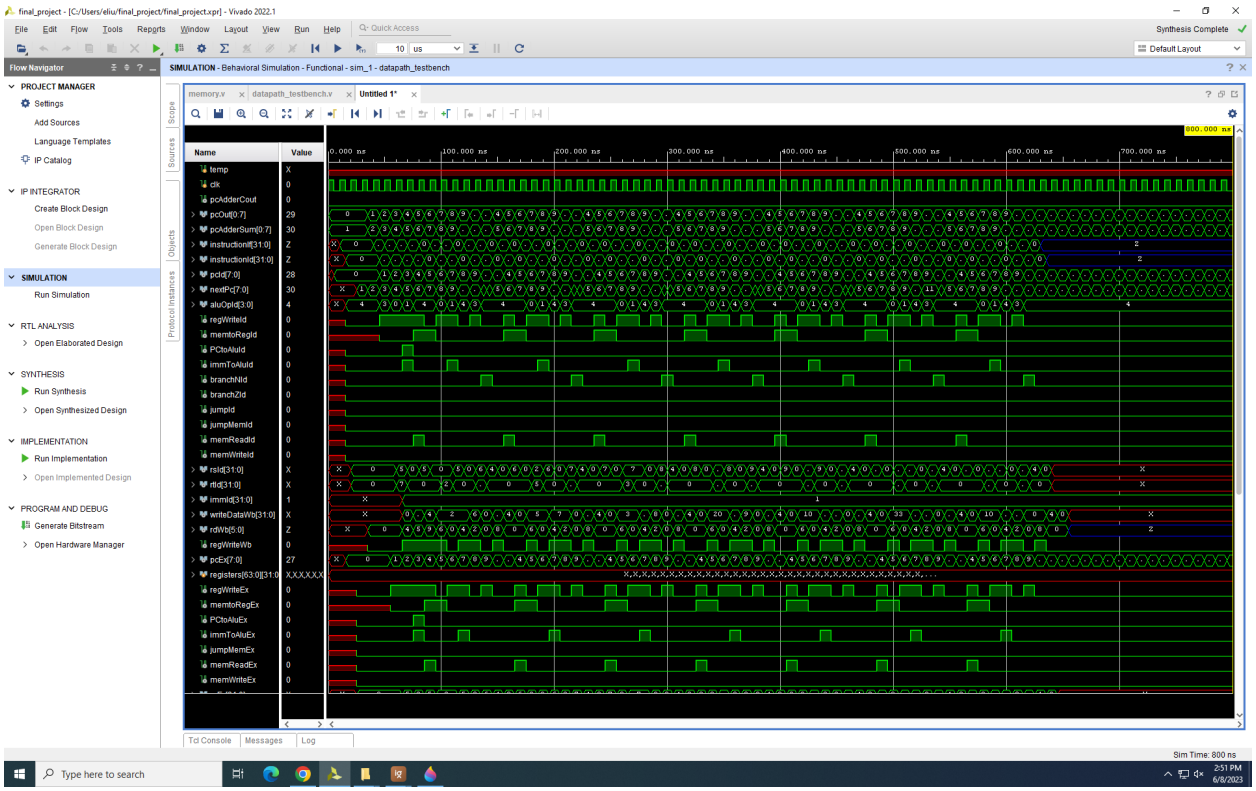
Sum // sum, rd, rs, rt

Adds values from memory[x rs] up to memory[xrs+xrt-1], stores result in x rd

Sum Waveforms







4. Assembly Code for sum calculation

```
assign instructions[0] = 32'b0000_00000000000000000000000000; // nop
assign instructions[1] = 32'b0111_00010000010000010000000000; // sub x4, x4, x4
assign instructions[2] = 32'b0100_00010100001000001100000000; // add x5, x2, x3
assign instructions[3] = 32'b1111_00100100000000000000000001; // svpc x9, 1
assign instructions[4] = 32'b1110_000110_000010_000000_0000000000; // ld x6, x2
assign instructions[5] = 32'b0000_00000000000000000000000000; // nop
assign instructions[6] = 32'b0100_000100_000100_000110_0000000000; // add x4, x4, x6
assign instructions[7] = 32'b0101_00001000001000000000000001; // inc x2, x2, 1
assign instructions[8] = 32'b0000_00000000000000000000000000; // nop
assign instructions[9] = 32'b0111_00100000010000101000000000; // sub x8, x2. x5
assign instructions[10] = 32'b1011_000000_001001_0000000000000000; // brn x9
assign instructions[11] = 32'b0000_00000000000000000000000000; // nop
```

Assumptions:

- X0 : 0
- X2 : base of A
- X3 : n -1 for n>= 1
- X4 : planned to be the Summation
- X5: address of a_n
- X9: address of 1st element in loop; branch address

Instructions:
SUB x4, x4, x4 // x4 = 0
ADD x5, x2, x3 // x5 = x2 + (n - 1)
SVPC X9, 1 // adds 1 to the current PC so it goes to the next instruction
LD x6, x2 // x6 = mem[x2]
ADD X4, X4, X6 // x4 += x6
INC X2, X2, 1 // x2 += 1
SUB X8, X2, X5 // comparing i to the address of A[n-1]
BRN X9 // branch to address stored in x9, continues next iteration if negative
STOP

5. **Estimate of Running Time for sum calculation**

Using the component delays defined in the lab handout, we formed a rough estimate for the number of cycles and the running time required for the execution of the sum assembly code

5*12 = 60 + 4 NOPs = 64 cycles * 10 ns/cycle
Total time: 640 ns