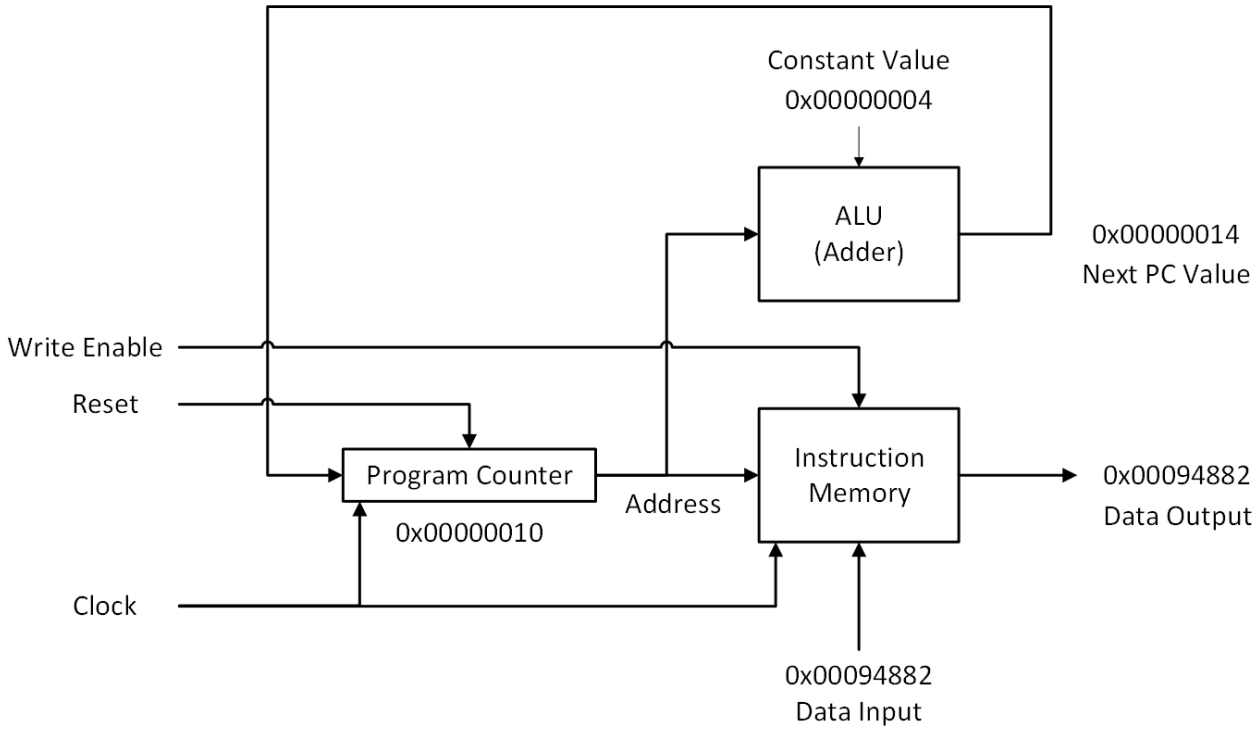
Elaboration on the Design Choices for Phase 1

by Matthew Collins and Lewis Bates



The VHDL implementation for Phase 1 of our MIPS processor was developed to meet the design constraints of the Phase 1 Design document. Our design is comprised of 6 main VHDL files: register_N.vhd, PC.vhd, ALU.vhd, Phase_1.vhd (top-level module), Phase_1_tb.vhd (testbench), and memory_1.vhd.

We began by analyzing the block diagram provided in the project description. The instruction fetch unit requires three key components: a 32-bit Program Counter (PC), an 8-bit addressable Instruction Memory with 32-bit output, and a combinational Add block (ALU) to increment the PC by 4.

From here, we followed this structure in order to complete the development process:

Component Design:

- **PC**: We implemented the PC as a synchronous 32-bit register using a generic register_N.vhd module. The generic parameter (N = 32) allowed flexibility, though we fixed it at 32 bits to match the MIPS architecture. The synchronous design ensured that the PC updates only on the rising edge of the clock, aligning with the requirements for a synchronous block.

- **Instruction Memory**: Per the design constraints, we avoided custom Megafunctions for the PC and ALU but used Quartus's pre-built memory IP (memory_1) for the Instruction Memory. We configured it as a single-port RAM with an 8-bit address (256 words) and 32-bit data width.

- **ALU**: The ALU was designed as a simple 32-bit combinational adder. Initially, we hardcoded the constant input of 4 inside the ALU, but after reviewing the design constraints, we modified it to accept a second 32-bit input (B), which we set to 4 in the top-level module.

Top-Level Integration:

- In Phase_1.vhd, we used a structural architecture to port-map the PC, Instruction Memory, and ALU. The PC output feeds the ALU and the 8 least significant bits (LSBs) connect to the memory address input. The ALU output loops back to the PC input, forming the fetch cycle. We added output ports (PC_out, instr_mem_out, alu_out, and alu_const_out) to expose internal signals for simulation. This allows for the testbench to monitor the PC input/output, memory address/output, ALU output, and constant input.

Testing the Instruction Fetch with a Testbench

In order to test the operation of the instruction fetch portion of the project, we were to write five MIPS instructions using the MIPS operations: add, sub, srl, lw, and sw. The fives instructions chosen are performing some manipulation on an index in an array. These instructions can be seen below. The base address of the array is stored in register $t0. The first instruction loads the fifth index of the array and stores it into register $t1. The next instruction increments the value of the index by the value stored in register $t2. The next instruction decrements the value of the index by the value stored in register $t3. The fourth instruction performs a two-bit right shift on the value of the index, effectively dividing it by four. After the manipulation has been done, the last instruction stores the value back into the array in memory.

```
lw      $t1,    5($t0)
add     $t1,    $t1,    $t2
sub     $t1,    $t1,    $t3
srl     $t1,    $t1,    2
sw      $t1,    5($t0)
```

These instructions need to be converted to machine code. The first and last instructions would be I-format instructions and the middle three would be R-format instructions. The table below shows the instructions in their machine code format, but still divided into the different sections of the instruction.

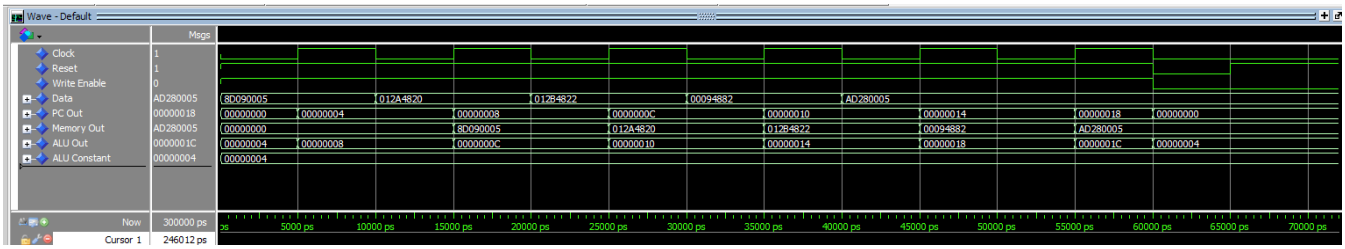
Operation	Op-Code	RS	RT	RD	Shamt	Func
				Address / Immediate		
lw	100011	01000	01001	0000000000000101		
add	000000	01001	01010	01001	00000	100000
sub	000000	01001	01011	01001	00000	100010
srl	000000	00000	01001	01001	00010	000010
sw	101011	01001	01000	0000000000000101		

For simpler representation, the instructions can be shown in hexadecimal as follows: (lw) 0x8D090005, (add) 0x012A4820, (sub) 0x012B4822, (srl) 0x00094882, and (sw) 0xAD280005. In the testbench used to test the instruction fetch, each instruction is saved as a variable with binary value of the machine code that corresponds to the instruction. The wren (write enable) signal and reset signal are set to prepare for writing. The data input signal is set with the value of the first instruction and the clock signal is toggled. The data signal is set with the next instruction and the process repeats for all five instructions. The clock is toggled an additional time because there seems to be a one clock cycle delay for the output of the memory.

The wren (write enable) bit is turned off and the reset signal is toggled to prepare for reading. The clock signal is given five rising edges, and an additional one to account for the delay, to display the values stored in the memory. The values of the inputs, outputs, and intermediate values through the duration of the testbench can be seen in the waveforms.

Waveform Analysis

The waveform in the figures below, generated using ModelSim, simulates the MIPS instruction fetch unit from Phase_1.vhd. It includes all required signals: clock, reset, wren, data, PC_out, instr_mem_out, alu_out, and alu_const_out. The simulation covers writing to and reading from the memory, allowing us to verify the implementation.

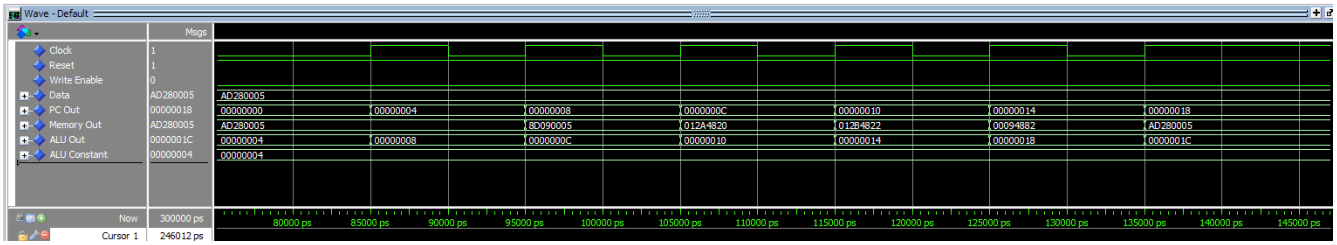


Writing to the Memory (0 ps to 60000 ps)

The testbench writes five MIPS instructions to memory addresses 0 to 4 with wren set to '1':

- Address 0: lw (0x8D090005) at 5000 ps
- Address 1: add (0x012A4820) at 15000 ps
- Address 2: sub (0x012B4822) at 25000 ps
- Address 3: srl (0x00094882) at 35000 ps
- Address 4: sw (0xAD280005) at 45000 ps

The PC_out(7 downto 0) provides the address, incrementing by 4 each cycle (0x00, 0x04, 0x08, 0x0C, 0x10). At 60000 ps, reset goes low to reset the PC, and wren is set to '0'.



Reading from the Memory (60000 ps to 110000 ps)

After reset, the PC starts at 0x00000000, and instructions are read with wren = '0':

- At 85000 ps, address 0 (0x00): Expected 0x8D090005 (lw), got 0x8D090005.
- At 95000 ps, address 1 (0x04): Expected 0x012A4820 (add), got 0x012A4820.
- At 105000 ps, address 2 (0x08): Expected 0x012B4822 (sub), got 0x012B4822.
- At 115000 ps, address 3 (0x0C): Expected 0x00094882 (srl), got 0x00094882.
- At 125000 ps, address 4 (0x10): Expected 0xAD280005 (sw), got 0xAD280005.

Successful Implementation

The waveform confirms a successful implementation. The instructions written to addresses 0 to 4 are correctly read back at the expected times (one clock cycle after the address is presented). The

instr_mem_out matches the written data at each address, and the PC increments by 4 each cycle, driven by the ALU with a constant input of 0x00000004, as shown by alu_const_out.

Concluding Remarks

Our implementation of the MIPS Processor successfully meets the project requirements outlined in the Phase 1 design requirements. The structural design, comprising the PC, Instruction Memory, and ALU components, adheres to the specified constraints, including the use of a synchronous 32-bit PC, an 8-bit addressable Instruction Memory, and a combinational ALU with a configurable constant input. The simulation waveform, as shown in the provided screenshots, confirms the correct writing and reading of five MIPS instructions (lw, add, sub, srl, sw) to and from memory addresses 0 to 4 at the expected time instances (5000 ps to 105000 ps), with the PC incrementing appropriately by 4 each cycle. The consistent match between written and read data, validated by the testbench, demonstrates a robust and functional design. Notably, the Instruction Memory's one-clock-cycle delay—where the output (instr_mem_out) updates one clock cycle after the address (PC_out[7:0]) changes—was observed and accounted for in the testbench by adding an extra clock cycle after writing and displaying the addresses. This delay, inherent to the synchronous memory, was evident in the waveform (e.g., address 0x00 at 65000 ps resulted in the correct output at 75000 ps) and did not impact the design's functionality, as the instructions were read correctly at the adjusted times. The exposure of internal signals for simulation further supports verification and ensures that the project's simulation meets the design requirements.

Overall, the implementation serves as our starting point for future phases of the project.

Appendix

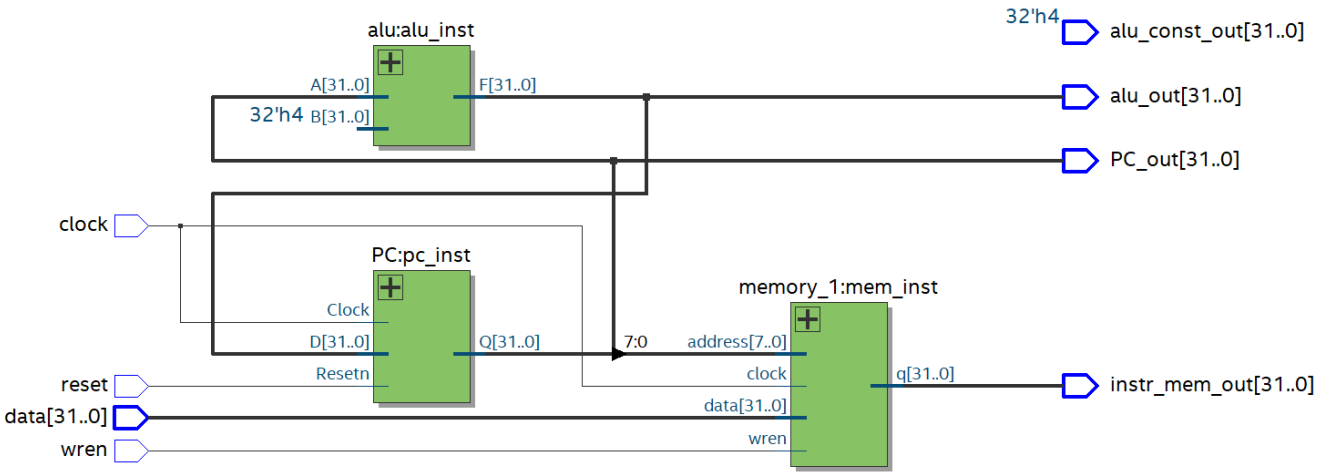
Compilation Flow Summary and Device Selection.

Flow Summary

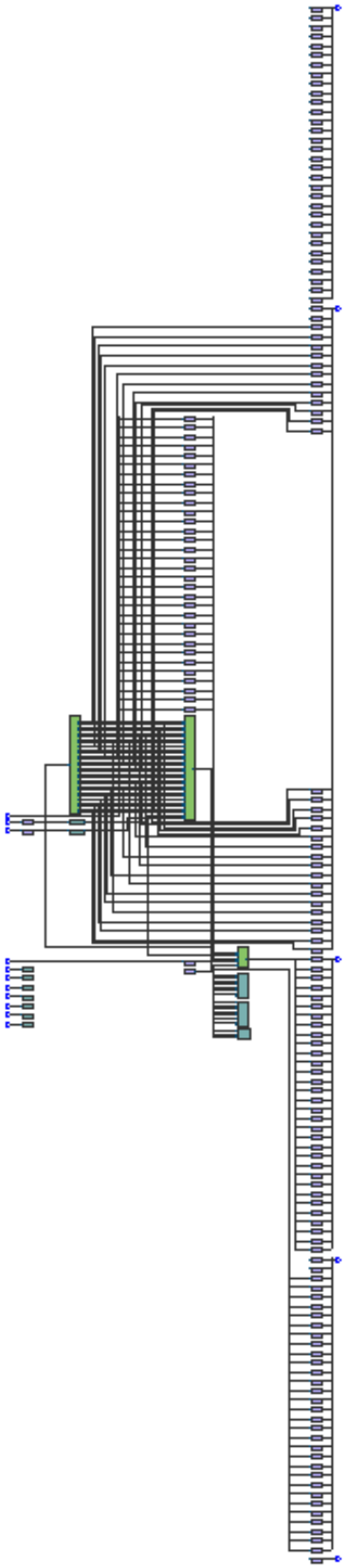
<<Filter>>

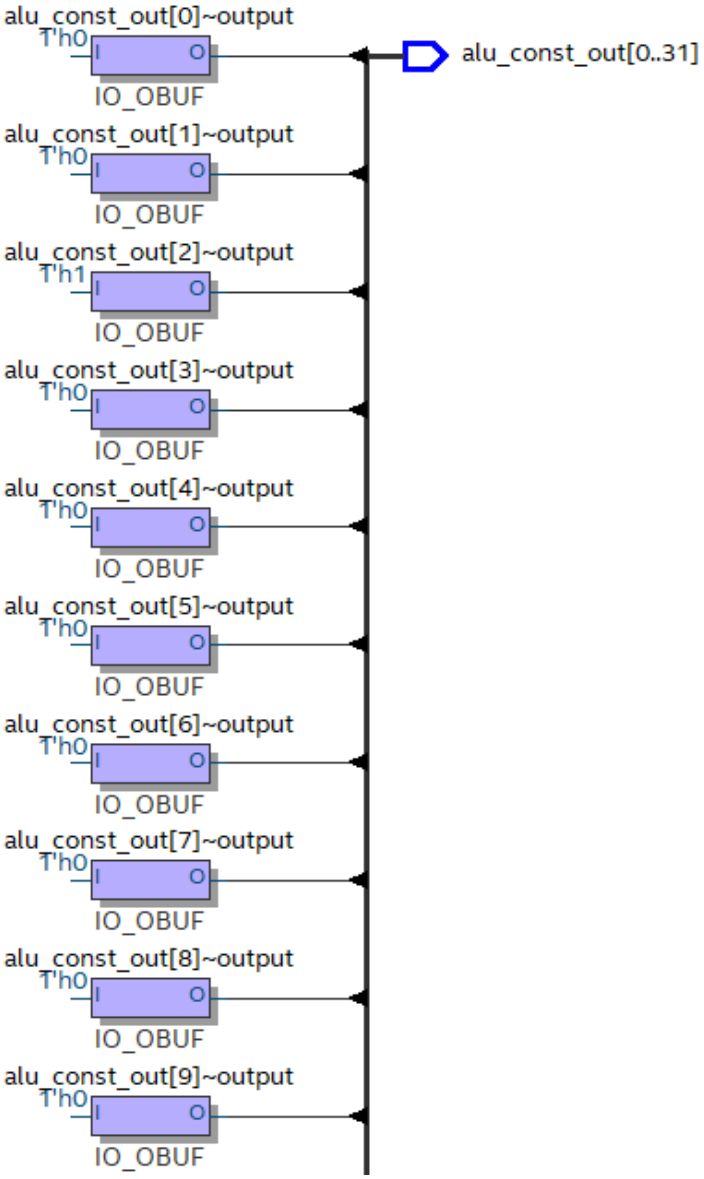
Flow Status	Successful - Fri Mar 14 15:20:51 2025
Quartus Prime Version	23.1std.1 Build 993 05/14/2024 SC Lite Edition
Revision Name	Phase_1
Top-level Entity Name	Phase_1
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	33 / 49,760 (< 1 %)
Total registers	30
Total pins	163 / 360 (45 %)
Total virtual pins	0
Total memory bits	8,192 / 1,677,312 (< 1 %)
Embedded Multiplier 9-bit elements	0 / 288 (0 %)
Total PLLs	0 / 4 (0 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 2 (0 %)

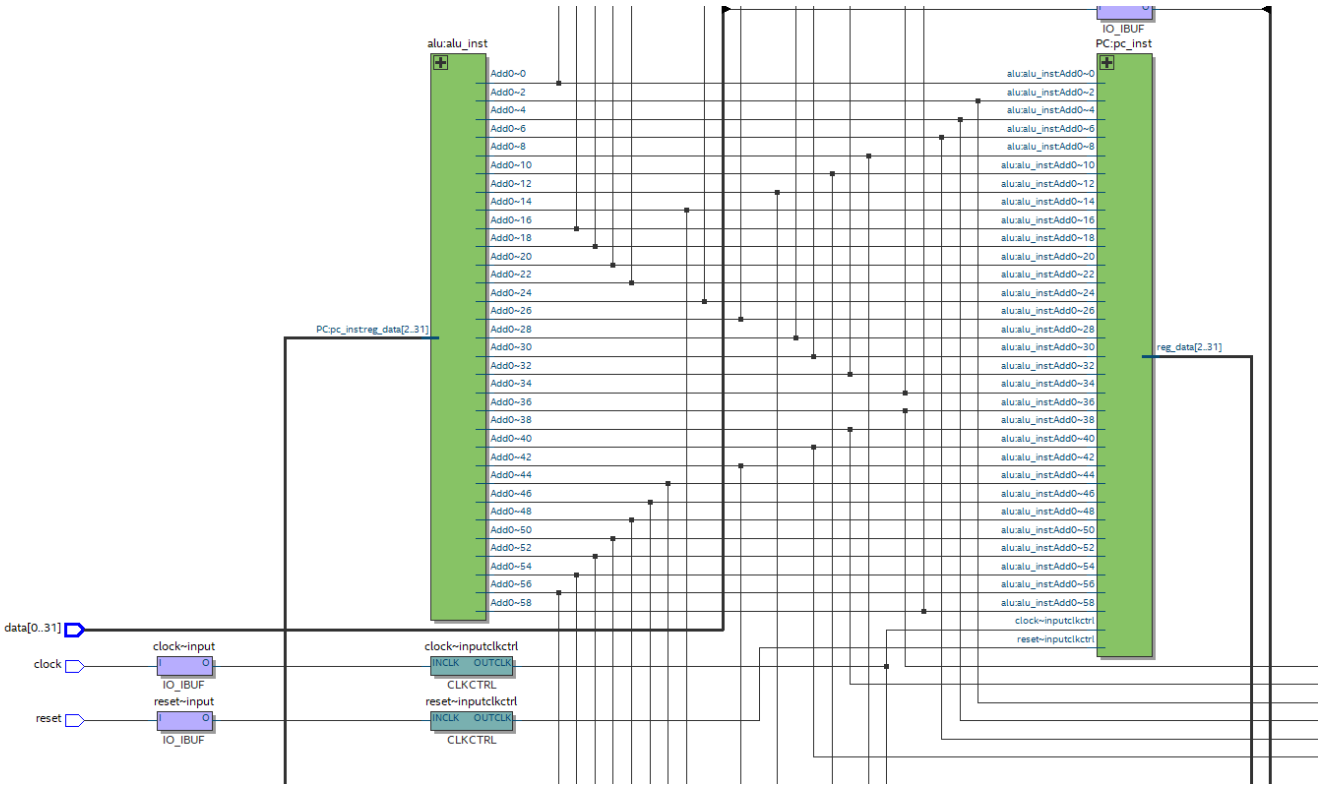
RTL View

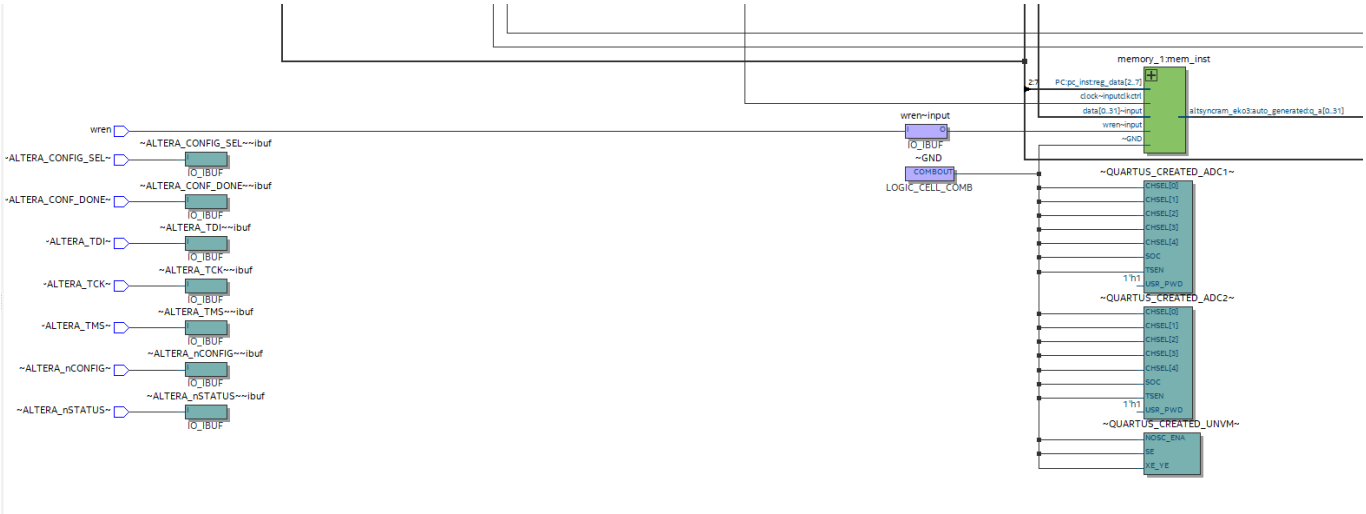


Technology Map




alu_const_out[0]~output
1'h0
I
O
IO_OBUF
alu_const_out[1]~output
1'h0
IO_OBUF
alu_const_out[2]~output
1'h1
IO_OBUF
alu_const_out[3]~output
1'h0
IO_OBUF
alu_const_out[4]~output
1'h0
IO_OBUF
alu_const_out[5]~output
1'h0
IO_OBUF
alu_const_out[6]~output
1'h0
IO_OBUF
alu_const_out[7]~output
1'h0
IO_OBUF
alu_const_out[8]~output
1'h0
IO_OBUF
alu_const_out[9]~output
1'h0
IO_OBUF
alu_const_out[0..31]


alu:alu_inst
Add0~0
Add0~2
Add0~4
Add0~6
Add0~8
Add0~10
Add0~12
Add0~14
Add0~16
Add0~18
Add0~20
Add0~22
Add0~24
Add0~26
Add0~28
Add0~30
Add0~32
Add0~34
Add0~36
Add0~38
Add0~40
Add0~42
Add0~44
Add0~46
Add0~48
Add0~50
Add0~52
Add0~54
Add0~56
Add0~58
PC:pc_inst|reg_data[2..31]
IO_IBUF
PC:pc_inst
alu:alu_inst|Add0~0
alu:alu_inst|Add0~2
alu:alu_inst|Add0~4
alu:alu_inst|Add0~6
alu:alu_inst|Add0~8
alu:alu_inst|Add0~10
alu:alu_inst|Add0~12
alu:alu_inst|Add0~14
alu:alu_inst|Add0~16
alu:alu_inst|Add0~18
alu:alu_inst|Add0~20
alu:alu_inst|Add0~22
alu:alu_inst|Add0~24
alu:alu_inst|Add0~26
alu:alu_inst|Add0~28
alu:alu_inst|Add0~30
alu:alu_inst|Add0~32
alu:alu_inst|Add0~34
alu:alu_inst|Add0~36
alu:alu_inst|Add0~38
alu:alu_inst|Add0~40
alu:alu_inst|Add0~42
alu:alu_inst|Add0~44
alu:alu_inst|Add0~46
alu:alu_inst|Add0~48
alu:alu_inst|Add0~50
alu:alu_inst|Add0~52
alu:alu_inst|Add0~54
alu:alu_inst|Add0~56
alu:alu_inst|Add0~58
clock~inputclkctrl
reset~inputclkctrl
reg_data[2..31]
data[0..31]
clock
clock~input
IO_IBUF
clock~inputclkctrl
CLKCTRL
reset
reset~input
IO_IBUF
reset~inputclkctrl
CLKCTRL


wren
~ALTERA_CONFIG_SEL~~ibuf
^ALTERA_CONFIG_SEL~
IO_IBUF
~ALTERA_CONF_DONE~~ibuf
^ALTERA_CONF_DONE~
IO_IBUF
~ALTERA_TDI~~ibuf
^ALTERA_TDI~
IO_IBUF
~ALTERA_TCK~~ibuf
^ALTERA_TCK~
IO_IBUF
~ALTERA_TMS~~ibuf
^ALTERA_TMS~
IO_IBUF
~ALTERA_nCONFIG~~ibuf
~ALTERA_nCONFIG~
IO_IBUF
~ALTERA_nSTATUS~~ibuf
~ALTERA_nSTATUS~
IO_IBUF
wren~input
IO_IBUF
~GND
COMBOUT
LOGIC_CELL_COMB
memory_1:mem_inst
2:7
PC:pc_inst|reg_data[2..7]
clock~inputclkctrl
data[0..31]~input
wren~input
~GND
altsyncram_eko3:auto_generated|q_a[0..31]
~QUARTUS_CREATED_ADC1~
CHSEL[0]
CHSEL[1]
CHSEL[2]
CHSEL[3]
CHSEL[4]
SOC
TSEN
1'h1
USR_PWD
~QUARTUS_CREATED_ADC2~
CHSEL[0]
CHSEL[1]
CHSEL[2]
CHSEL[3]
CHSEL[4]
SOC
TSEN
1'h1
USR_PWD
~QUARTUS_CREATED_UNVM~
NOSC_ENA
XE
XE_YE