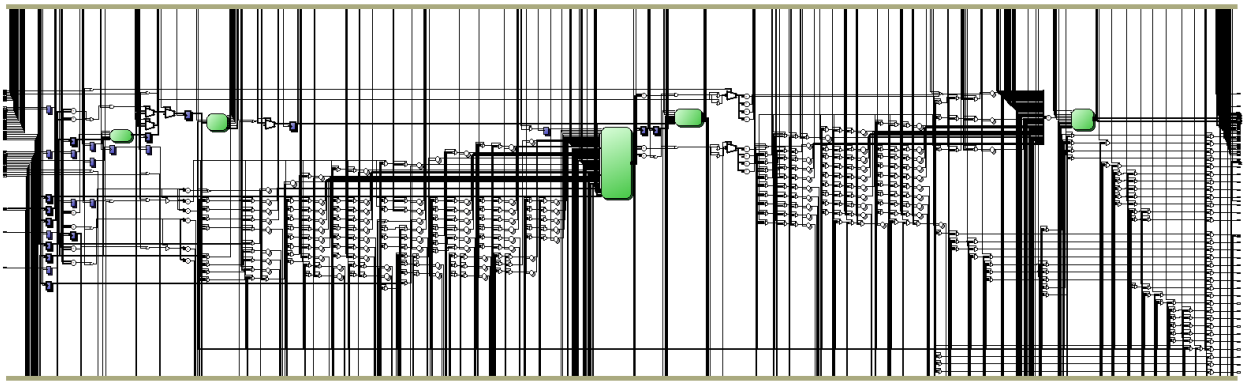
Lab 5

Design

The following block diagrams represent the system's design as shown in Quartus RTL viewer:

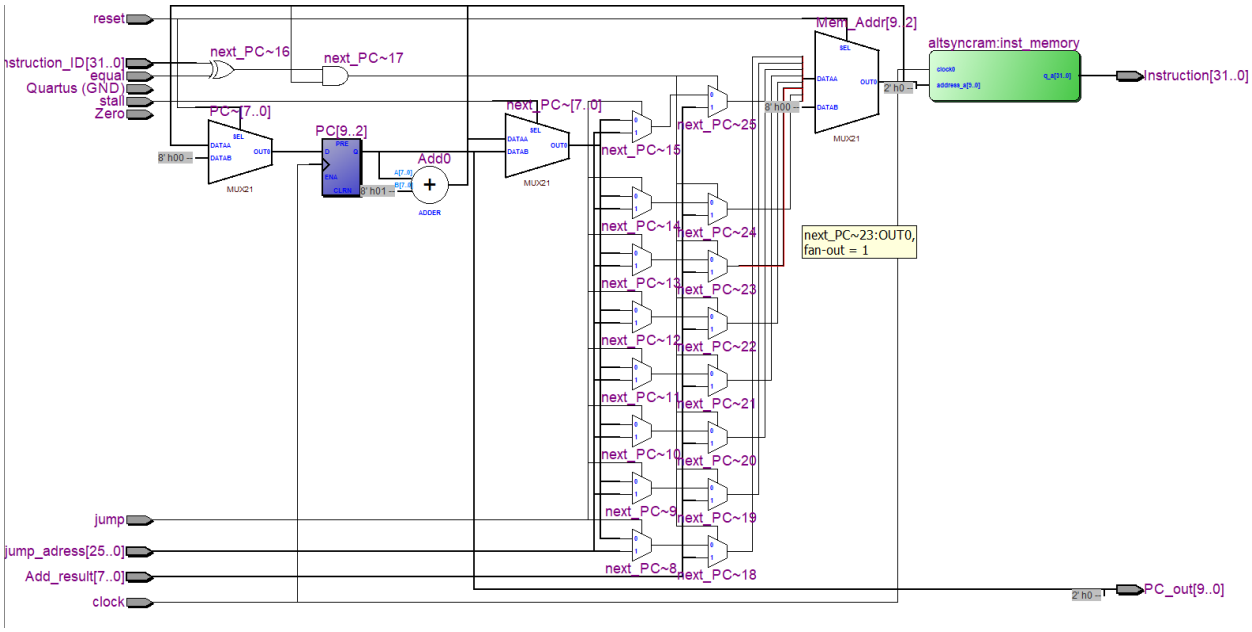


DESCRIPTION:

This is an overview of the design, which operates as a fully functional MIPS processor, being fed a memory component with data and a MIPS Assembly program, and syncs fetching the program, decoding, executing and reading/ writing to memory if needed.

Instruction FETCH block:

RTL diagram:

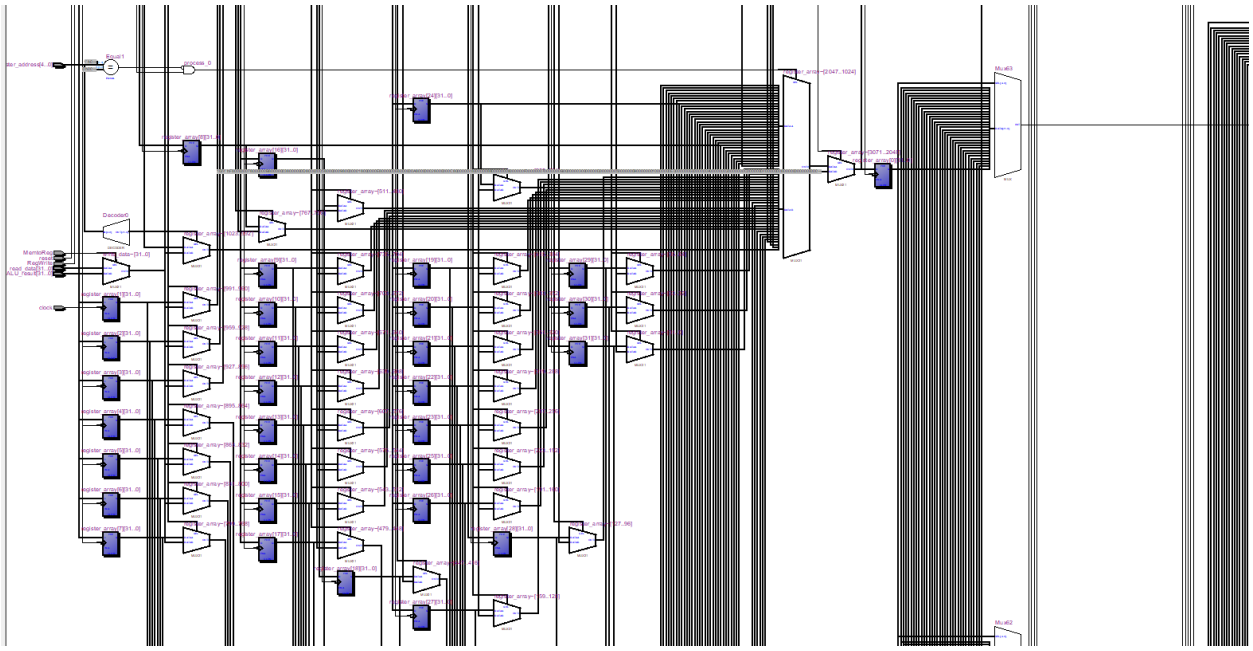


DESCRIPTION:

The fetch instruction block is responsible for the fetch instruction phase of each operation. It performs it and passes on the relevant signals to the next stage of the pipeline, the Decode stage through the IF/ID registers

Instruction Decode Block:

RTL diagram:

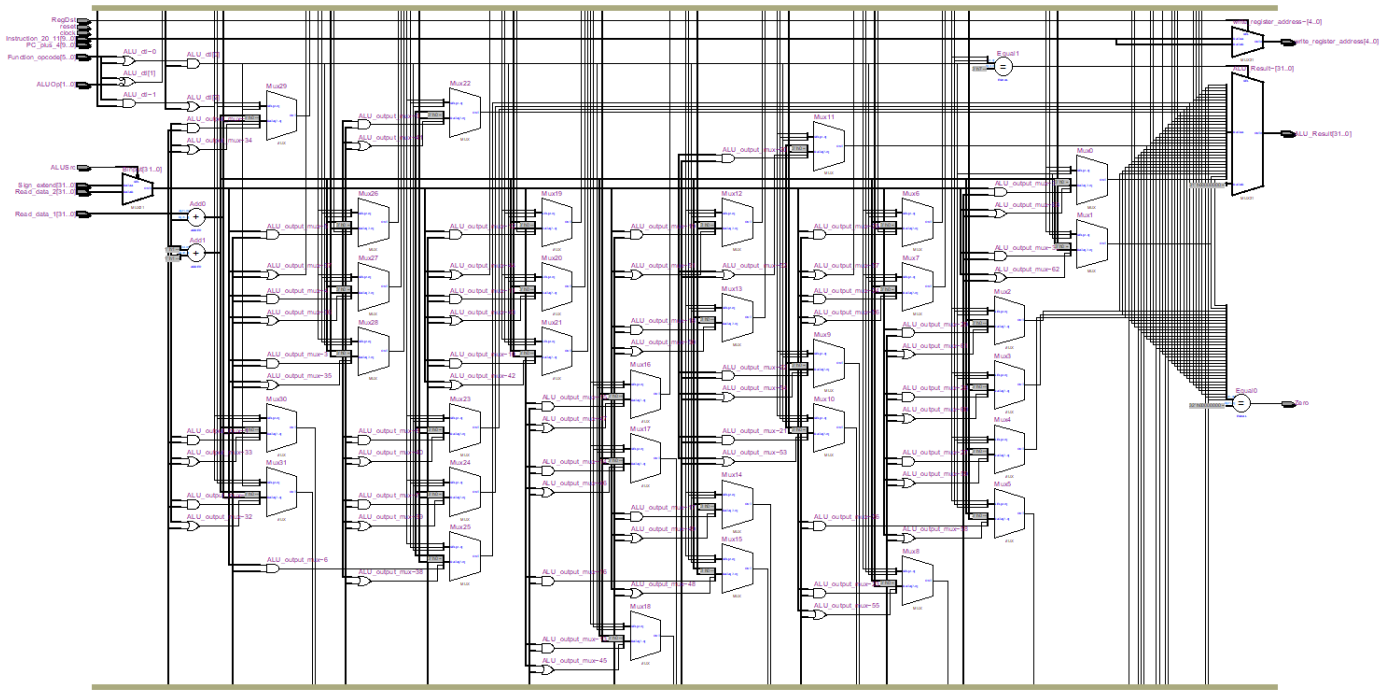


DESCRIPTION:

The Instruction Decode block is responsible for the decode phase of each operation. It performs it and passes on the relevant signals to the next stage of the pipeline, the execute stage through the ID/EX registers

Execute Block:

RTL diagram:

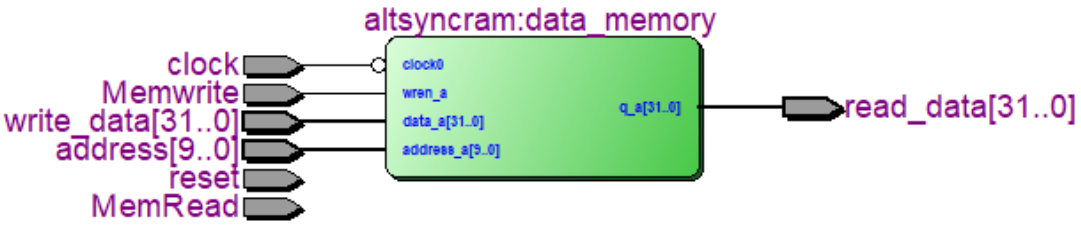


DESCRIPTION:

The Execute block is responsible for the execute phase of each operation. It performs it and passes on the relevant signals to the next stage of the pipeline, the memory stage, through the EX/MEM registers

Memory Stage Block:

RTL diagram:

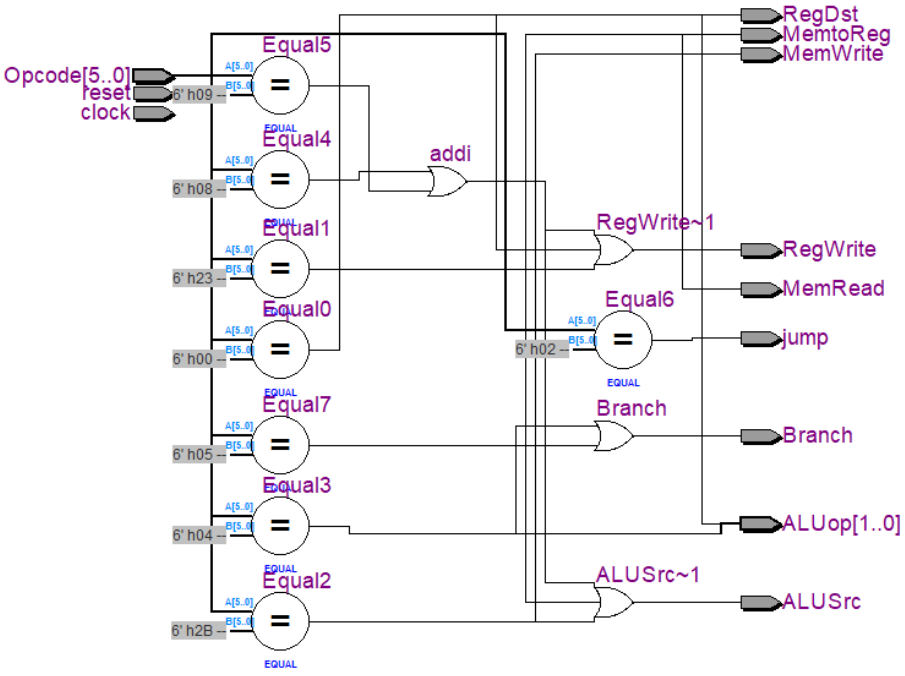


DESCRIPTION:

The DMem Unit is responsible for the memory access (read/write) phase of each operation. It performs it and passes on the relevant signals to the next stage of the pipeline, the write back stage, which, in our design, happens back in the Instruction Decode block.

Control Unit:

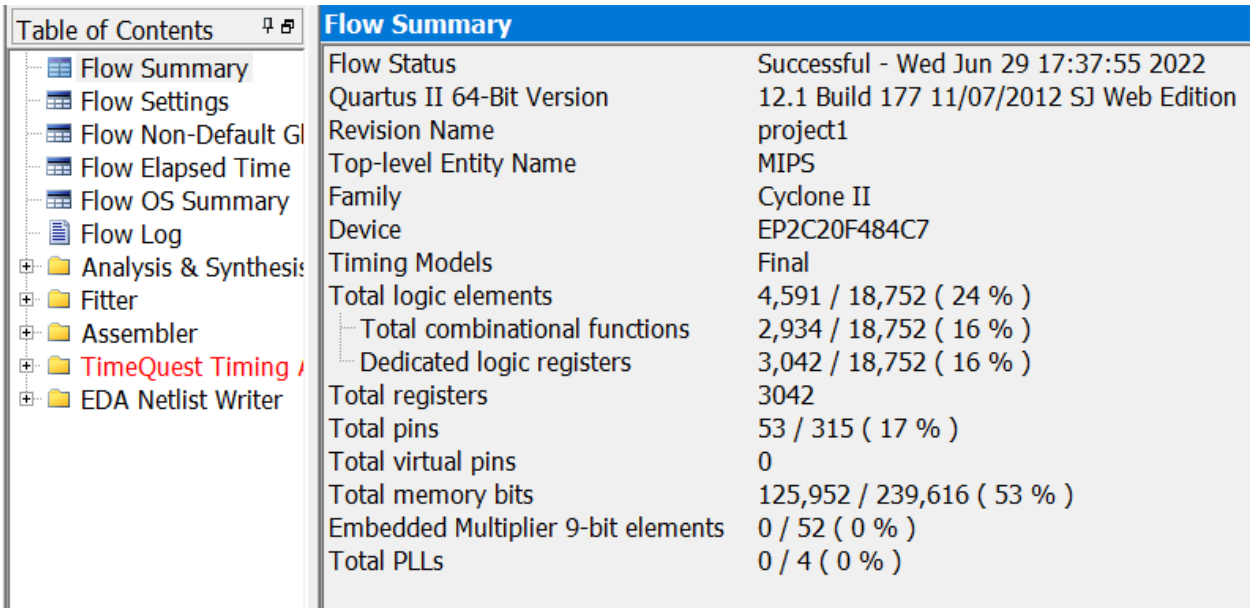
RTL diagram:



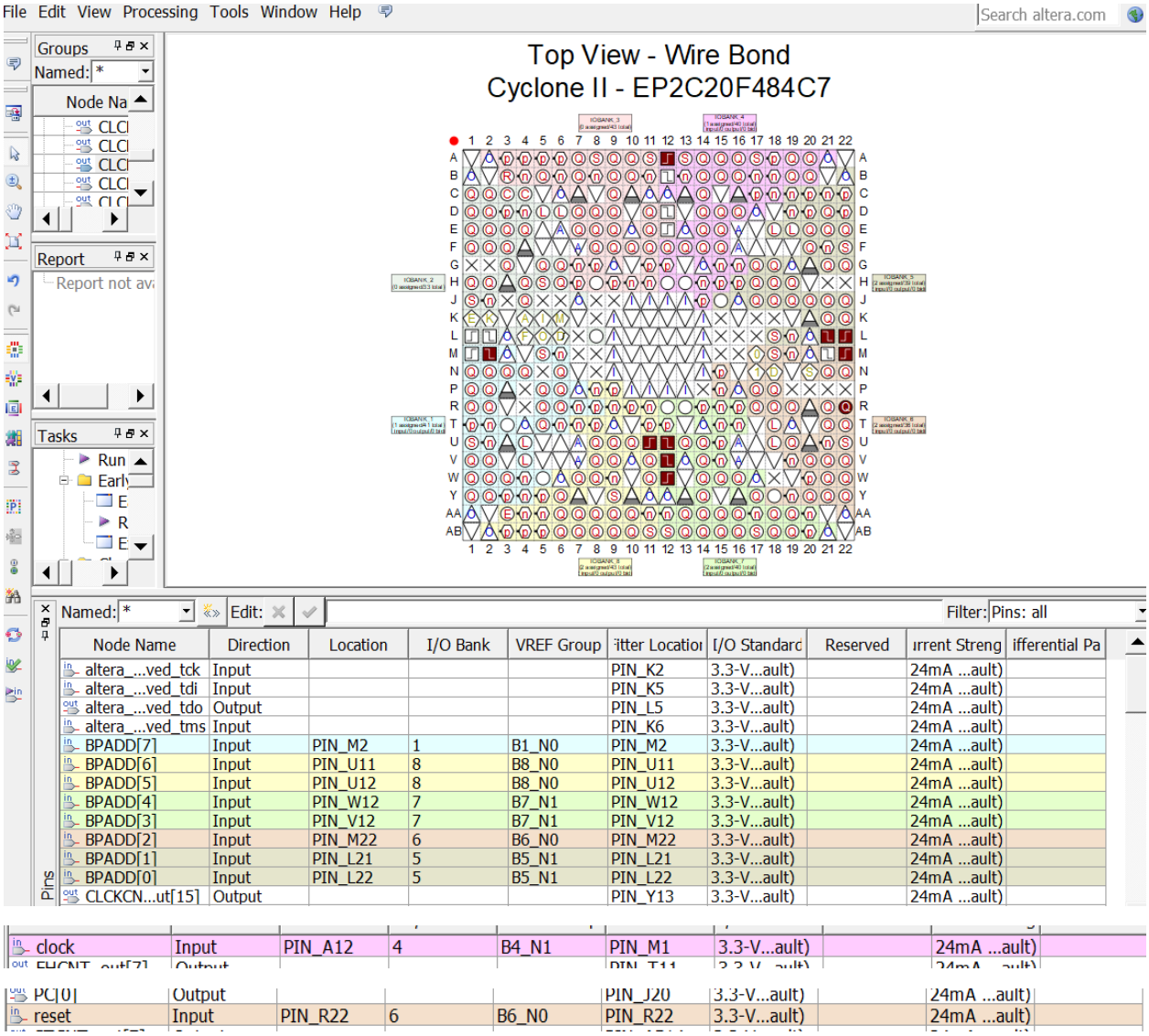
DESCRIPTION:

The Execute block is responsible for the execute phase of each operation. It performs it and passes on the relevant signals to the next stage of the pipeline, the memory stage, through the EX/MEM registers

Logic Usage



Pin planner

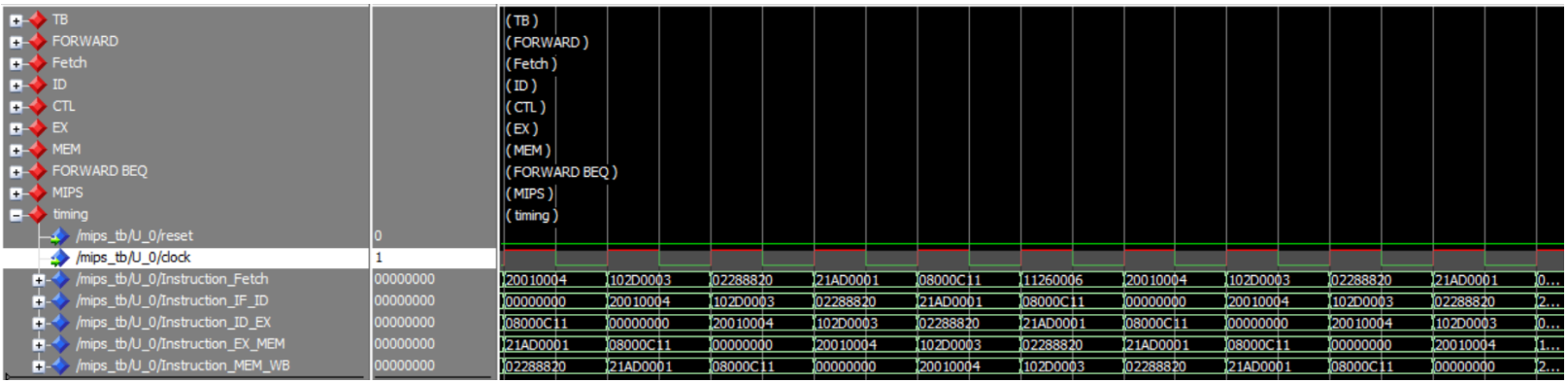


Pins were assigned to the relevant I/O functions. The above diagram show the layout on the device.

Signal waveform:

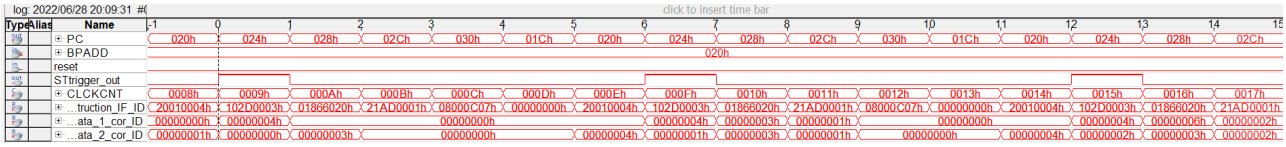
This is the waveform representation as given by the MODELSIM simulation:

This example is the addition of $x = 1001$ and $y = 1000$
The result is the expected one: $ALUout = 10001$



QUARTUS Signal Tap waveform:

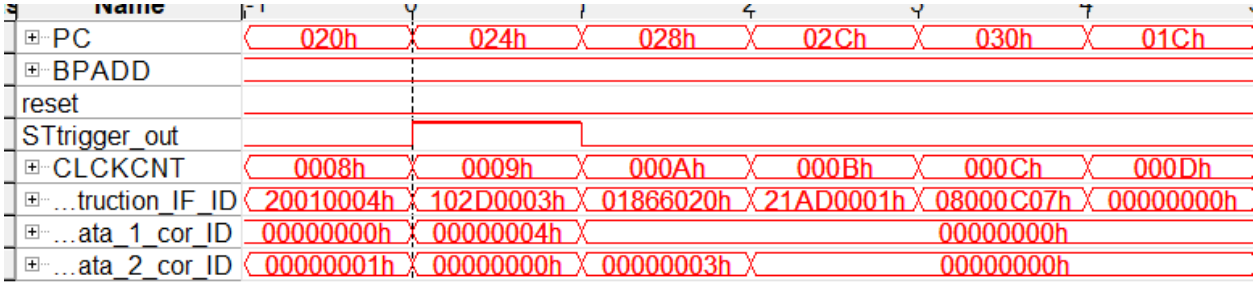
This is the waveform representation as given by the QUARTUS Signaltap:



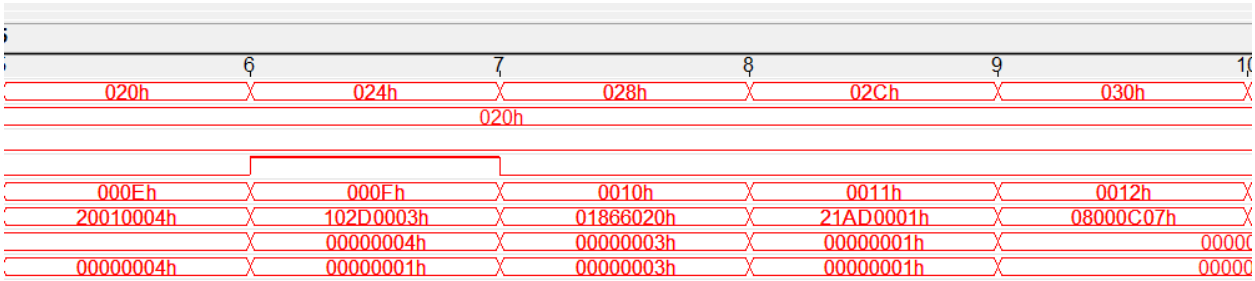
Here we perform the following assembly code, as part of the transpose program:

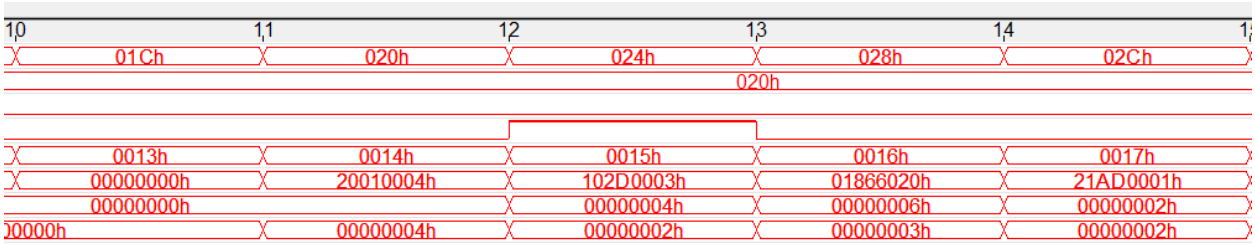
0x00000018	addiu $13,$0,0x0000..	0x240d0000	19: li $t5, 0
0x0000001c	addi $1,$0,0x00000004	0x20010004	22: beq $t5, 4, main
0x00000020	beq $1,$13,0x00000003	0x102d0003	
0x00000024	add $12,$12,$6	0x01866020	23: add $t4, $t4, $a2
0x00000028	addi $13,$13,0x0000..	0x21ad0001	24: addi $t5, $t5,1
0x0000002c	j 0x0000001c	0x08000007	25: j make 4M
0x00000030	add $16,$4,$0	0x00808020	29: add $s0, $a0, $zero

The result is the expected one: we get in a loop which can be easily seen in the enlarged screenshots below:



ssasdasdasdasdad





Timing

We analyzed the timing quest from the compilation report and got the following results:

Fmax analysis:

The following table shows the maximal clock frequency:

	Fmax	Restricted Fmax	Clock Name	Note
1	24.09 MHz	24.09 MHz	clock	

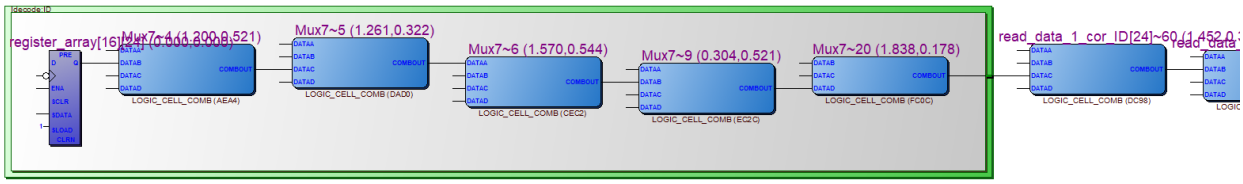
Critical path:

Critical path analysis:

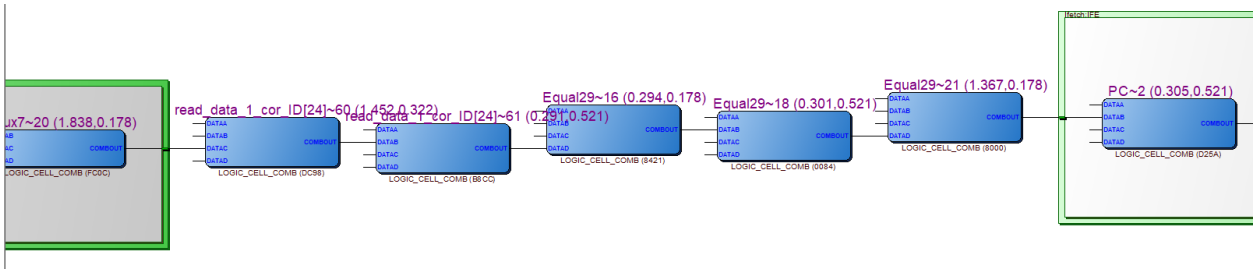
The following table shows the systems longest combinatorial (critical) path, as shown in time quest analysis:

Now, surprisingly, we can derive that the bottleneck is a forwarding path. It is the one responsible for a branch operation, it checks ALU for a zero flag to determine whether to jump or not.

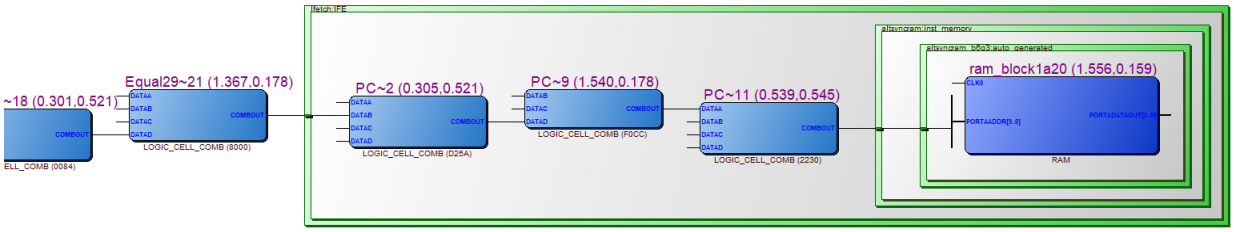
it is quite unexpected - as we would think that the execute block would require the most intense computation time. In real life, most of the time we would consider memory access as the longest path.



Here we see the part that passes through the decode stage, starting at the register array



This part forwards the information back to the fetch unit, to inform it of branch operation parameters.



This part goes through the fetch phase to control the program counter, and determine whether to branch to the branch address.

Optimizations:

Given the above insights, the ultimate step is to optimize the code in order to further the performance. The following was done accordingly:

- the clock period was changed to the optimal frequency:

$$t_{clk} = \frac{1}{f_{max}} = \frac{1}{24.9MHz} = 4.016 \times 10^{-8}$$

$$t_{clk}[ns] = \frac{1}{f_{max}} = \frac{1}{24.9MHz} = 40.16 \times 10^{-9}[sec]$$

$$= 40.16\,[ns] \sim 41ns$$

-unnecessary registers were removed

-unnecessary outputs were converted to signals

-unnecessary clocks were removed

-latches were replaced

In addition, we added an analysis of critical path and frequency limiting operations for each component:

The critical path inside the adder block can be seen in the green box below