# Part a

## Explanation

if(dec\_rs1 == exe\_rd && dec\_rs1\_renb == 1'b1 && exe\_rd\_wenb == 1'b1)

dec\_stall = 1'b1;

dec\_load\_use = exe\_load;

dec\_csr\_use = exe\_csr;

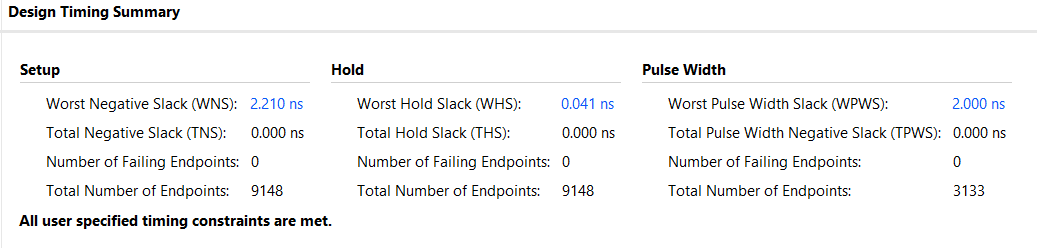
In my design I check each DEC operand’s address and see if they are the same as any of the EXE, MEM or WRB operands’ addresses. If they match, and the write enable bit of that stage is 1, then the system will stall.

## Evaluation

Cycle count = 1396  
CPI = 1.9  
Instruction Count = 620

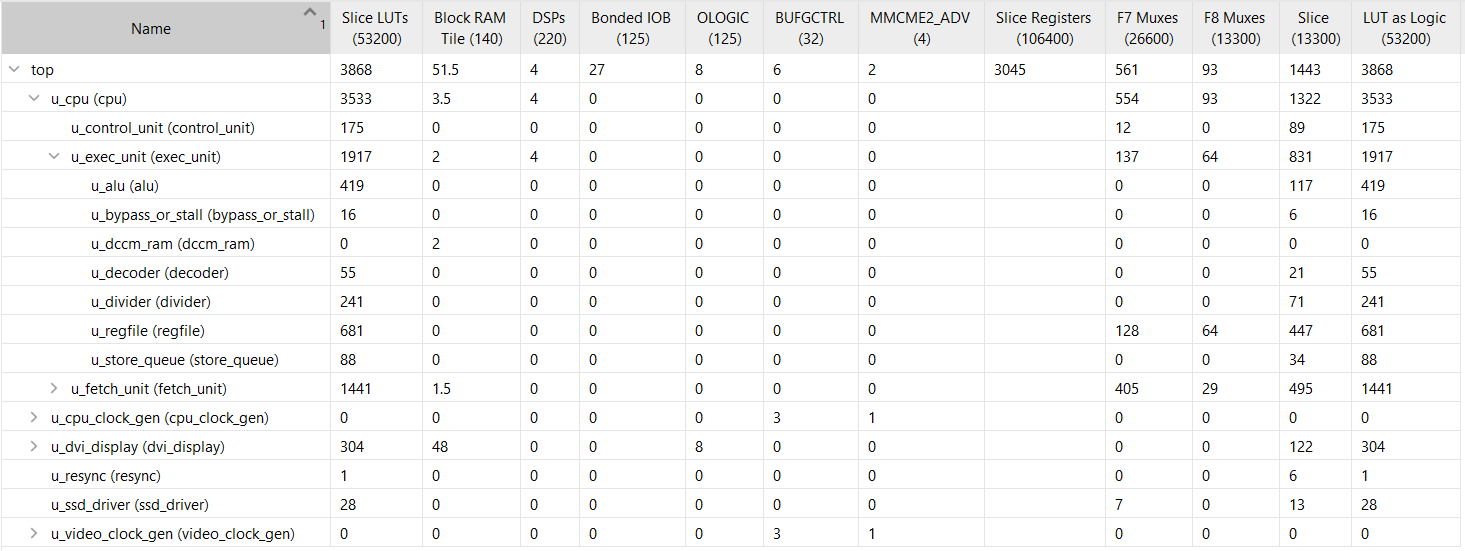
Critical Path:

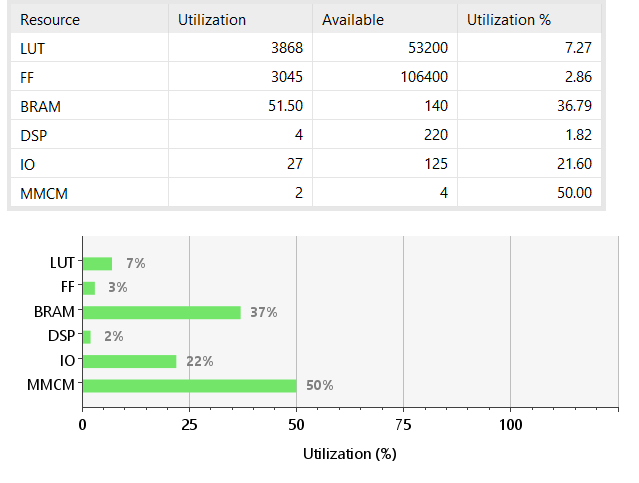
The Critical path is caused when branch prediction is carried out





Time for Mandlebrot = 14.2 seconds





# Part b

## Explanation

depend1 = 1'b0;  
depend2 = 1'b0;

I added these two registers to my design to allow me to tell which DEC operand has a dependency further down the pipeline.

if(dec\_rs1 == exe\_rd && dec\_rs1\_renb == 1'b1 && exe\_rd\_wenb == 1'b1 && depend1 == 1'b0)

begin

// Set the load and csr registers

dec\_load\_use = exe\_load;

dec\_csr\_use = exe\_csr;

depend1 = 1'b1;

// If the operation is load or csr then stall

if(exe\_load == 1'b1 || exe\_csr == 1'b1)

begin

dec\_stall = 1'b1;

end

else

dec\_rs1\_data = exe\_result;

end

At the EXE stage I only stall if there is a load or csr operation. Otherwise the exe\_result value is passed back to the relevant DEC operand.

if(dec\_rs2 == mem\_rd && dec\_rs2\_renb == 1'b1 && mem\_rd\_wenb == 1'b1 && depend2 == 1'b0)

begin

dec\_rs2\_data = mem\_result;

depend2 = 1'b1;

if(dec\_rs2 == exe\_rd && dec\_stall == 1'b1)

// If the dec operand has a dependency in the EXE stage but also later down the pipeline then there is no need to stall

dec\_stall = 1'b0;

end

At the MEM and WRB stages there will never be a stall. If the DEC operand had a dependency at the EXE stage but also has one at the MEM or WRB stage, then the stall is stopped.

## Evaluation

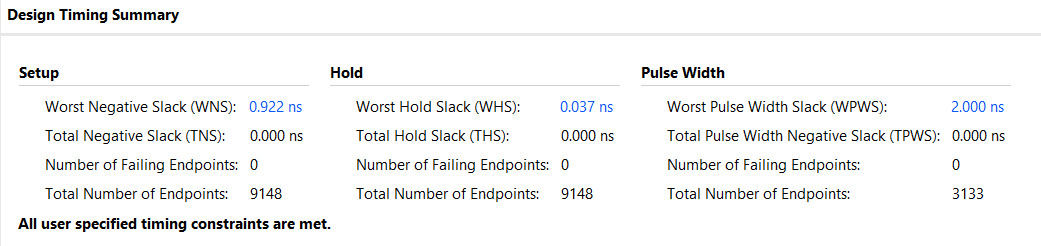
Cycle Count = 874  
CPI = 1.1  
Instruction Count = 620

**1. Why has the CPI not reduced all the way to 1.0 in this optimized run?**  
The CPI is not 1.0 as there are still stalls for CSR and load operations.

**2. Tabulate any remaining stalls that occur in this test**I counted a total of 40 load\_use stalls, no csr stalls and two pipeline flushes.

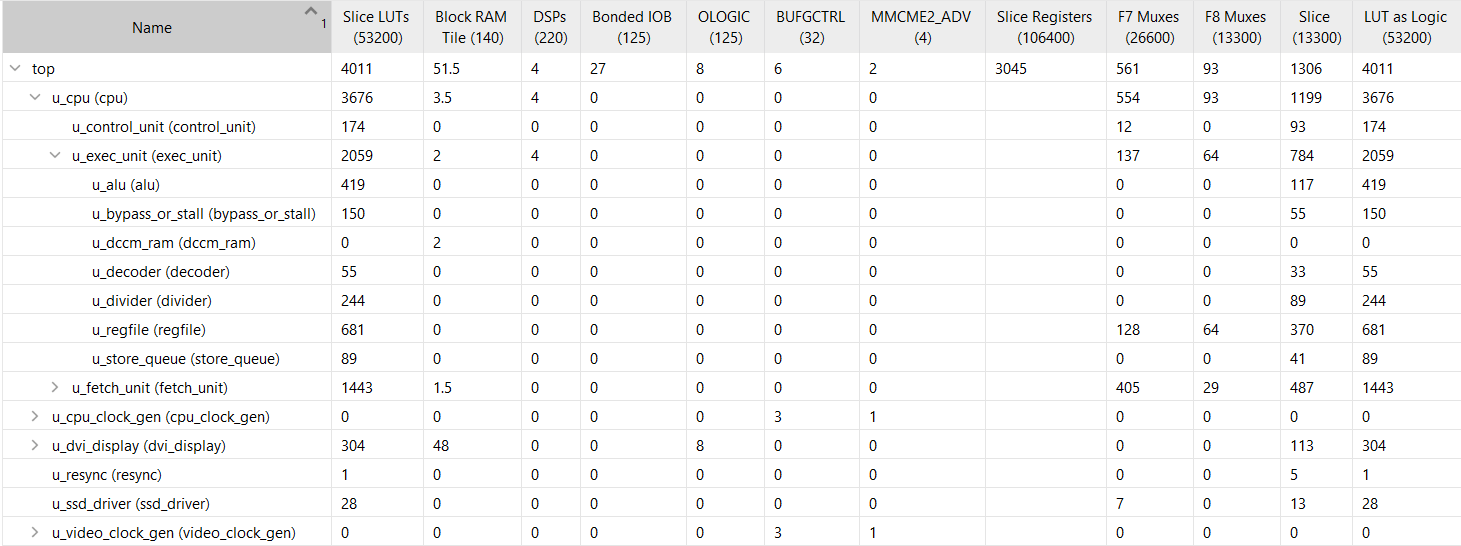
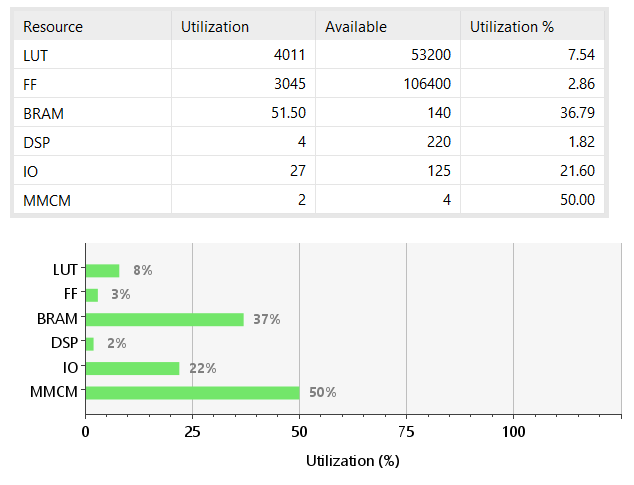
**3. If there are pipeline flushes, explain why they are there and estimate their performance impact for this test**Pipeline flushes occur when there is a branch, causing the next instruction to be incorrect. They would have a relatively big impact, as instructions were being carried out that need to be wiped and, after the flush, the EXE, MEM and WRB stage will not be doing anything.

Critical Path:  
The critical path in this optimised design is caused by a multiply operation.





Time for Mandlebrot = 8.5 seconds

# Part c

## Explanation

// If a multiply operation is carried out, send a signal to the stall\_or\_bypass to stall

case (exe\_alu\_opc\_r)

M32\_OPC\_MUL:

begin

m32\_result = m64\_result[31:0];

exe\_mul\_r = 1'b1;

end

M32\_OPC\_MULH,

M32\_OPC\_MULHSU,

M32\_OPC\_MULHU:

begin

m32\_result = m64\_result[63:32];

exe\_mul\_r = 1'b1;

end

M32\_OPC\_DIV,

M32\_OPC\_DIVU,

M32\_OPC\_REM,

M32\_OPC\_REMU: m32\_result = exe\_div\_result;

default: m32\_result = 32'hXXXX\_XXXX;

endcase

To optimise my design, I decided to prevent multiply results being passed back to bypass, as it was causing significant slowdown of the pipeline. When a multiply operation was carried out, I send a stall signal back to bypass\_or\_stall. This increased my CPI as it introduced additional stalls. Due to this optimisation, multiply is no longer the operation that causes the critical path and it is now branch prediction again.

if(dec\_rs1 == exe\_rd && dec\_rs1\_renb == 1'b1 && exe\_rd\_wenb == 1'b1 && depend1 == 1'b0)

begin

// Set the load, csr and mul registers

dec\_load\_use = exe\_load;

dec\_csr\_use = exe\_csr;

dec\_mul\_use = exe\_mul;

depend1 = 1'b1;

// If the operation is load, csr or multiply then stall

if(exe\_load == 1'b1 || exe\_csr == 1'b1 || exe\_mul == 1'b1)

begin

dec\_stall = 1'b1;

end

else

dec\_rs1\_data = exe\_result;

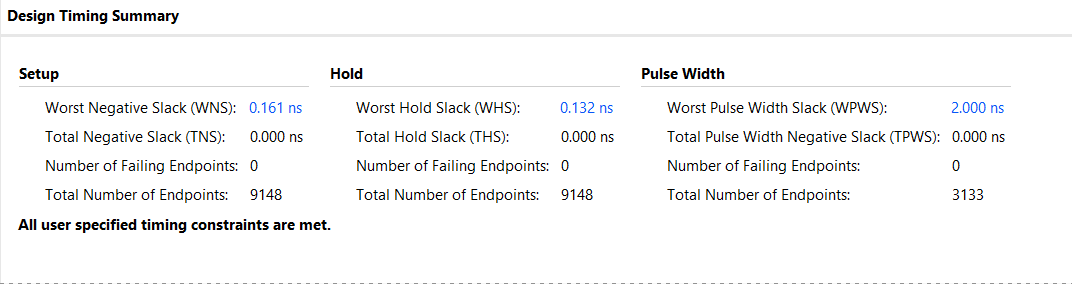
end

I added a condition to stall in the bypass\_or\_stall module if a multiply operation is being carried out.

## Evaluation

Cycle Count = 946  
CPI = 1.2  
Instruction Count = 620

Timing report after adjusting Fmax:





Final Fmax = 82MHz  
Time for Mandlebrot = 6.5 seconds

