MOTIVATION

Our main motivation is understanding the procedure and challenges of designing a pipelined processor. We first developed a 16-bit RISC microprocessor utilizing a simplified version of the MIPS architecture. We employed Harvard memory architecture. We then added 4 pipeline stages and pipeline registers to the design to improve upon the performance of the processor. We subsequently designed a data forwarding unit to resolve data dependencies.

SPECIFICATIONS

This section provides a concise overview of the processor's specifications key features, outlines the pins, presents a high-level diagram illustrating the external interface of the chip, and details the formats of the instruction words. The instruction set architecture consists of 16 instructions, complemented by the utilization of 16-bit external address lines for instruction and data memory. To summarize, we have:

- 16 instructions in the ISA
- 16 general purpose registers
- Instruction completes in 4 clock cycles.

Pins of the processor are as described,

Interface	Direction	Description
clk	Input	Clock signal
rst_n	Input	Reset signal active low.
imem_addr[15:0]	Output	16-bit address lines to instruction memory.
imem_rdata[15:0]	Input	16-bit instruction read line from instruction memory.
dmem_addr[15:0]	Output	16-bit address lines to data memory.
dmem_rdata[15:0]	Input	16-bit read data line from data memory.
dmem_wdata[15:0]	Output	16-bit write data line to data memory.
dmem_wr	Output	Data memory data write signal, active high
holt	Output	Signal line to show processor in holt state

Instruction Set Architecture (ISA)

The ISA of this processor consists of 16 instructions with a 4-bit fixed size operation code. The instruction words are 16-bits long. The following chart describes the all the instructions used in this project.

Operation	Opcode	Destination Register	Source Register	Target Register
	bits			
	15 14 13 12	*11 10 9 8*	*7 6 5 4*	*3 2 1 0*
ADD	0000	rd	rs1	rs2
SUB	0001	rd	rs1	rs2
AND	0010	rd	rs1	rs2
OR	0011	rd	rs1	rs2
XOR	0100	rd	rs1	rs2
NOT	0101	rd	rs1	0000
SLA	0110	rd	rs1	0000
SRA	0111	rd	rs1	0000
BLZ	1000	rs1	Offset	
JMP	1001	0000	Offset	
JAL	1010	rd	Offset	
RET	1011	0000	rs1	0000
LI	1100	rd	Immediate	
LW	1101	rd	rs1	0000
SW	1110	0000	rs1	rs2
HLT	1111	0000	0000	0000

This Processor features five instruction classes:

Arithmetic (Two's Complement) ALU operation

$$\boldsymbol{ADD}: rd = rs1 + rs2$$

Operands A and B stored in register locations rs1 and r2 are added and written to the destination register specified by rd.

$$\boldsymbol{SUB}: rd = rs1 - rs2$$

Operand B (rs2) is subtracted from Operand A (rs1) and written to rd.

Logical ALU operation

$$\boldsymbol{AND}: rd = rs1 \,\&\, rs2$$

Operand A (rs1) is bitwise anded with Operand B (rs2) and written into rd.

$$\boldsymbol{OR}: rd \ = \ rs1 \mid rs2$$

Operand A (rs1) is bitwise ored with Operand B (rs2) and written into rd.

$$\boldsymbol{XOR}: rd \ = \ rs1 \ \hat{} \ rs2$$

Operand A (rs1) is bitwise Xored with Operand B (rs2) and written into rd.

$$\boldsymbol{NOT}: rd \ = \ \sim rs1$$

Operand A (rs1) is bitwise inverted and written into rd.

$$\boldsymbol{SLA}: rd \ = \ rs1 \ << \ 1$$

Operand A (rs1) is arithmetically shifted to the left by one bit and written into rd.

$$\boldsymbol{SRA}: rd \ = \ rs1 \ >> \ 1$$

Operand A (rs1) is arithmetically shifted to the right by one bit and written into rd. The MSB (sign bit) will be preserved for this operation.

Memory operations

$$\boldsymbol{LI}: rd \ = \ 8 - bit \ Sign \ extended \ Immediate$$

The 8-bit immediate in the Instruction word is sign-extended to 16-bits and written into the register specified by rd.

$$\boldsymbol{LW}: rd \ = \ Mem[rs1]$$

The memory word specified by the address in register rs1 is loaded into register rd.

$$\boldsymbol{SW}: Mem[rs1] = rs2$$

The data in register rs2 is stored into the memory location specified by rs1.

Conditional Branch operations

$$\boldsymbol{BLZ}: PC \ = \ PC \ + \ 1 \ + \ Offset \ \boldsymbol{if} \ rs1 < 0$$

If register rs1 is less than zero, then the current Program Count (PC + 1) is offset to PC + 1 + Offset.

Program Count Jump operations

$$\boldsymbol{JMP}: PC \ = \ PC \ + \ 1 \ + \ Offset$$

Unconditional jump instruction will offset the program count to PC + 1 + Offset.

$$\boldsymbol{JAL}: rd \ = \ PC \ + \ 1 \ and \ PC \ = \ PC \ + \ 1 \ + \ Offset$$

Jump and Link instruction would write current Program Count in register rd and offset the program count to PC + 1 + Offset.

$$\boldsymbol{RET}: PC = rs1$$

Jump Return instruction will set the Program Count to the one previously stored in JAL.

The processor also has holt operation given by

$$\boldsymbol{HLT}: stop\ all\ proccessor\ operation$$

The processor stages are,

- Fetch Instruction:
  - Retrieve Instruction Word from Instruction Memory
  - Increment Program Count
- Decode Instruction:
  - Load Operands into Latches from Register File
- Execute Instruction
  - Perform ALU Operation based instruction word
  - Move data memory word for Load Word operation
  - Write Data into Memory from Register File for Store Word operation
- Write Back
  - Write ALU/Immediate/MEM data into Register File
  - Write new Program Count for Jump Operation or if Branch taken

IMPLEMENTATION

Micro-architecture provides a comprehensive perspective of a computing system, revealing essential elements like registers, buses, and pivotal functional units such as Arithmetic Logic Units (ALUs) and counters. The core subsystems constituting a processor include the Central Processing Unit (CPU), main memory, and input/output interfaces. The dynamic interplay between the data path and the control unit is instrumental in executing the primary processing tasks. The data path and the control unit collaborate closely, forming a symbiotic relationship to carry out the intricate operations required for efficient processing.

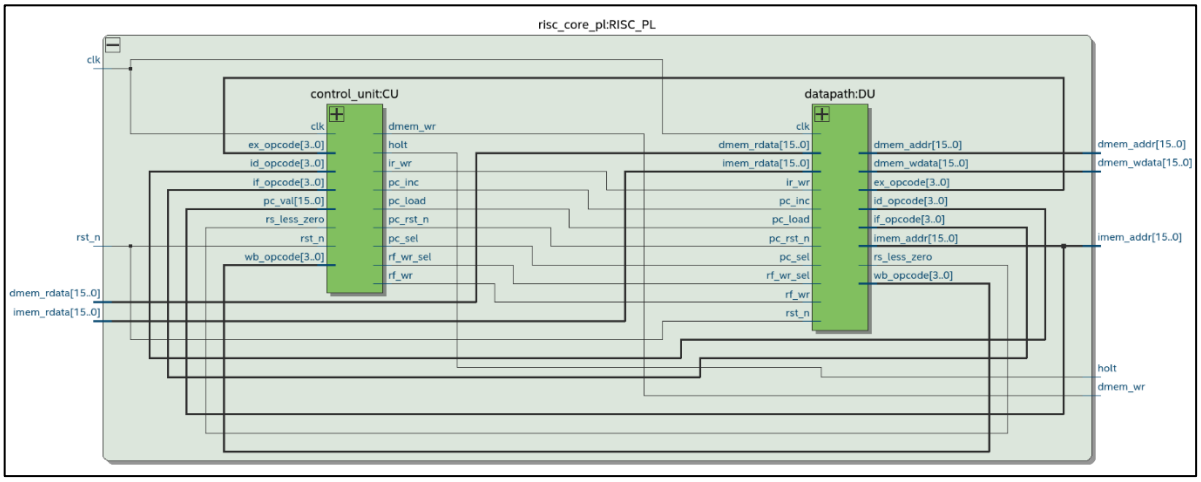

Figure 1: Microarchitecture of the Pipelined RISC processor

Control Unit

The control unit assumes a pivotal role in this coordination, receiving signals from the data path and, in turn, transmitting control signals back to the data path. These signals intricately govern the flow of data within the CPU and orchestrate the interaction between the CPU and both the main memory and input/output peripherals. Essentially, the control unit acts as the conductor, orchestrating the harmonious exchange of information and guiding the intricate dance of data throughout the computing system.

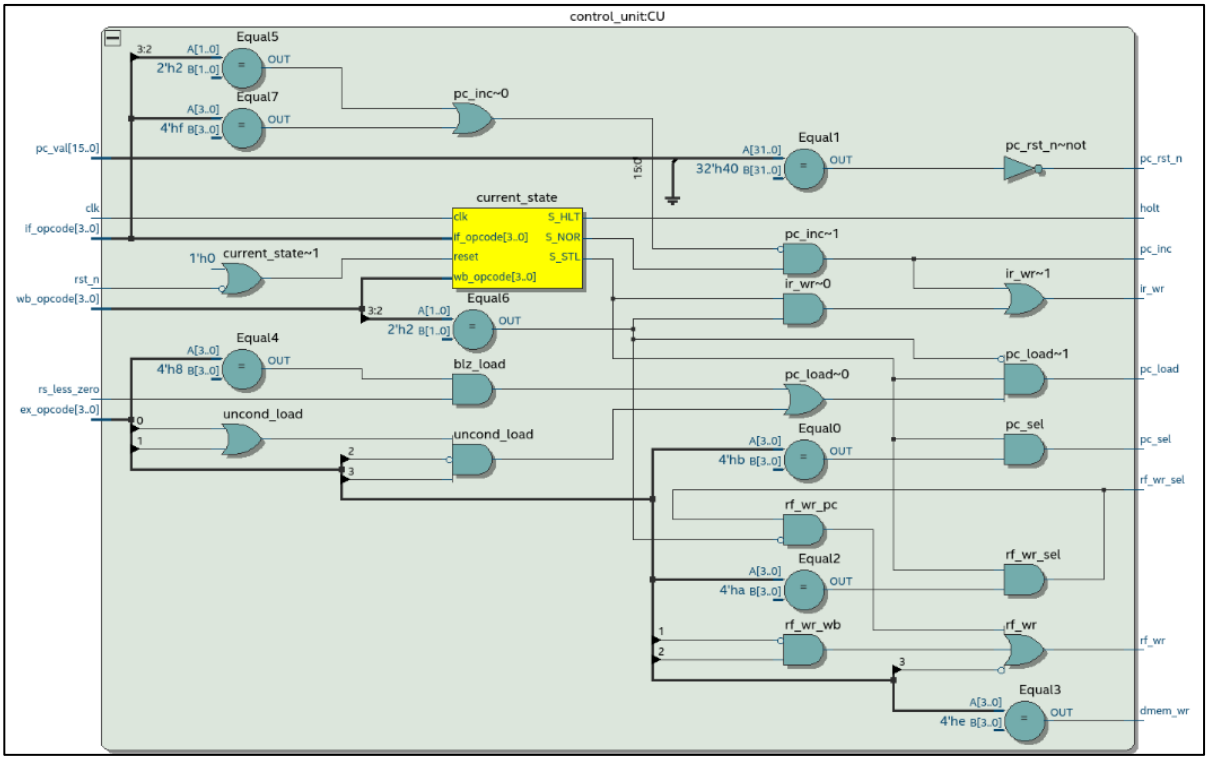

Figure 2: Control Unit of the Pipelined processor

As we shown the control unit receives opcodes of all the stages and generates appropriate signals as needed. The Finite State Machine (FSM) diagram of the design is,

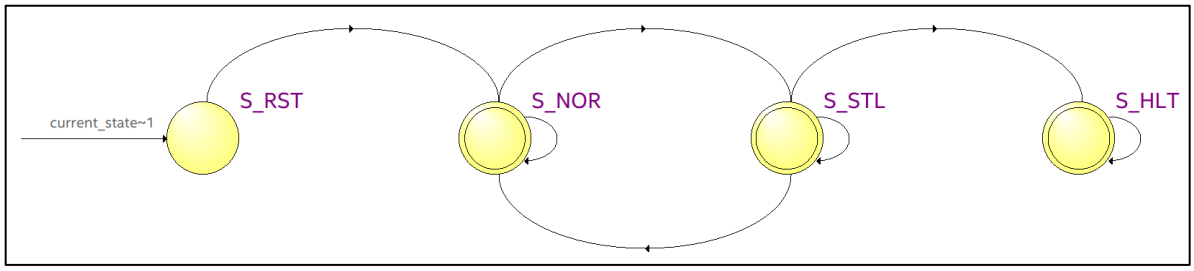

Figure 3: Finite State Machine (FSM) diagram of the pipelined processor

Upon reset, the processor goes to the reset state, and 1 cycle later enters the Normal operation state. When the processor encounters a jump/branch operation or holt operation it enters Stall state. The processor remains in Stall state until the pipeline finishes execution of the previous on-going instructions, the branch condition is resolved, and the new instruction target is calculated. If the stalling instruction is holt operation, then the processor enters holt state and remains there, else it returns to normal operation state.

Data Path Unit

The data path unit contains all the functional units and registers of the pipelined processor. It consists of Instruction Fetch Unit (IFU), IF -ID Pipeline Register (FDPR), Instruction Decode Unit (IDU), ID-EX Pipeline Register (DEPR), Execution Unit (EXU), EX-WB Pipeline Register (EWPR), Register File (RF), and Data Forwarding Unit (DFU).

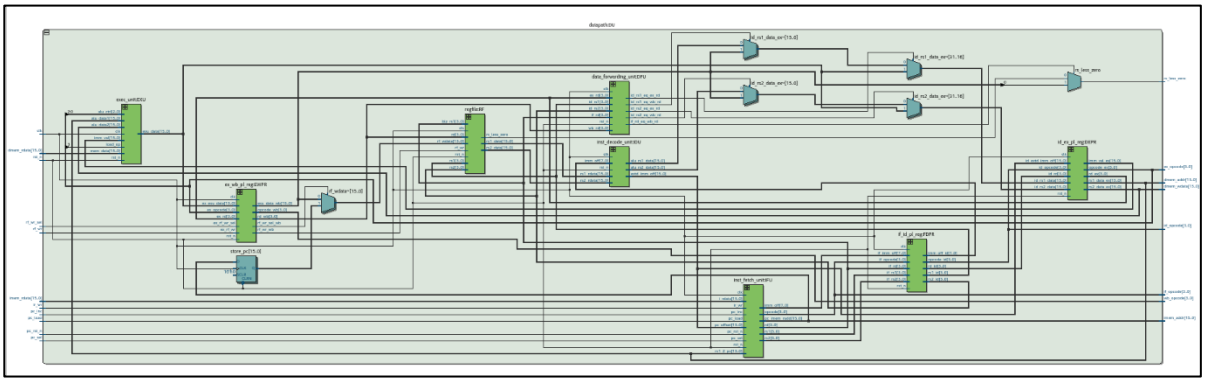

Figure 4: Data Path of the pipelined processor

The IFU houses the Instruction Register (IF) and the Program counter (PC) along with required logics for updating PC value.

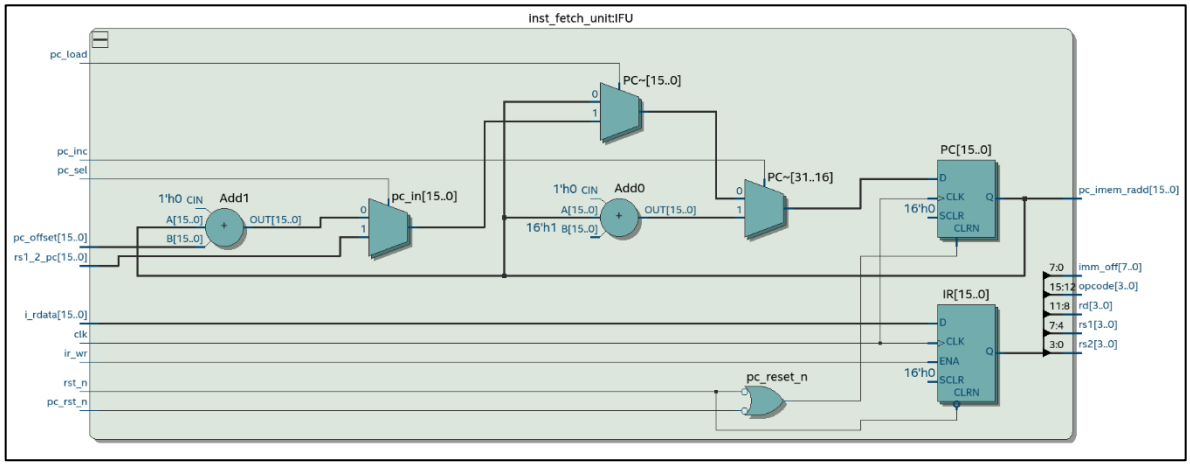

Figure 5: Instruction Fetch Unit showing IF and PC

The pipeline register FDPR between IFU & IDU holds the required values for the decode unit as we instruction will be coming into the IF register in the next cycle. IDU retrieves values of from the register file (RF), and sign extends the immediate/offset value provided with the instruction. DEPR holds those values to be used by execution unit (EXU).

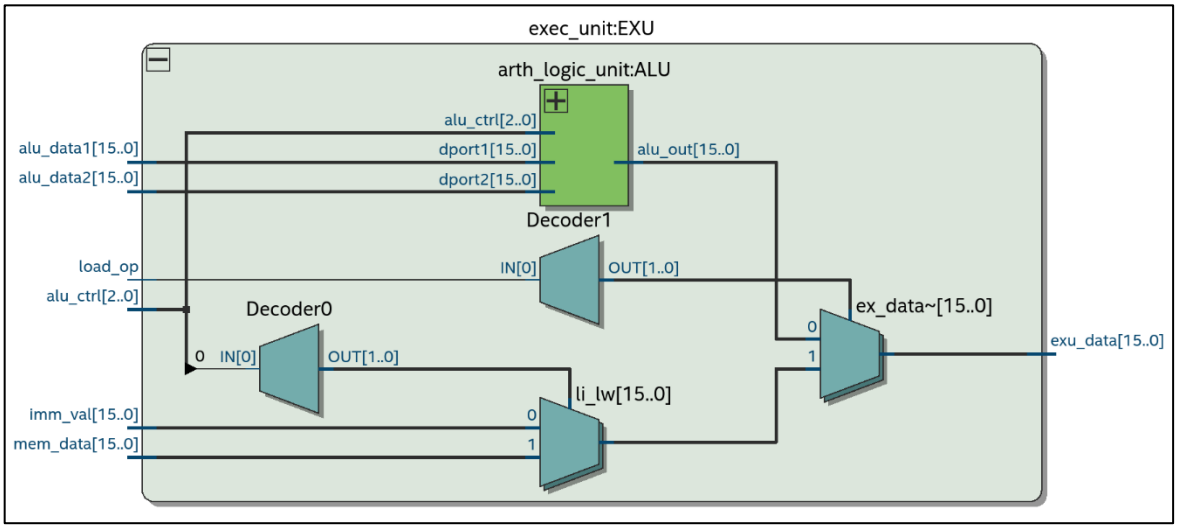

Figure 6: Execution Unit

Output of the EXU is selected between immediate value, memory data or ALU result depending on opcode. EWPR retains the values for writeback stage to be written in the register file.

The data forward unit resolves the data dependencies between instructions that are currently in different stages of the pipeline. If generates select signals for the multiplexers that select data to be hold by DEPG as rs1 and rs2.

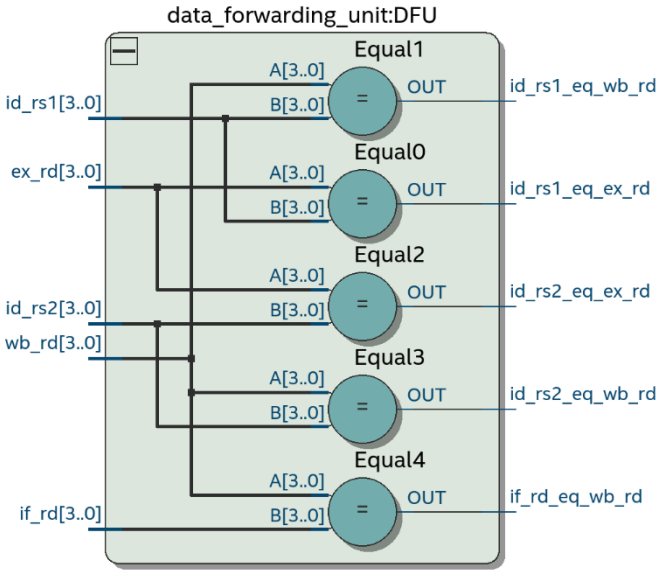

Figure 7: Data Forwarding Unit

SIMULATION

We used ModelSim to simulate our processor. To be able to simulate the processor core we added two memory units to build the SoC.

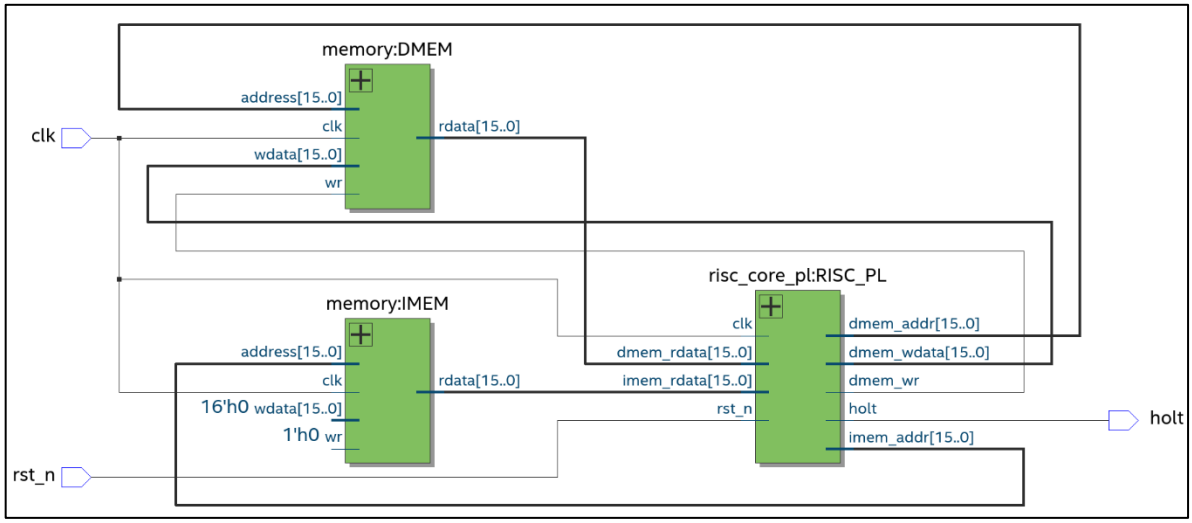

Figure 8: Pipelined RISC SoC

The assembly code used for the simulation is given,

Instruction memory address	Assembly	Machine Code
0x00	LI X0, 31	0xC01F
0x01	LI X1, 1	0xC101
0x02	JAL X15, +29	0xAF1D

0x03	LI X4, 4	0xC404
0x04	LI X5, 5	0xC505
0x05	LI X6, 6	0xC606
0x06	LI X7, 7	0xC707
0x07	JMP +51	0x9033
:	: : : :	:
0x10	LI X14, 0	0xCE00
0x11	SW X14, X8	0xE0E8
0x12	ADD X14, X14, X1	0x0EE1
0x13	SW X14, X9	0xE0E9
0x14	ADD X14, X14, X1	0x0EE1
0x15	SW X14, X10	0xE0EA
0x16	ADD X14, X14, X1	0x0EE1
0x17	SW X14, X11	0xE0EB
0x18	SUB X12, X2, X2	0x1C22
0x19	BLZ X12, +37	0x8C25
0x1a	XOR X8, X2, X3	0x4823
0x1b	NOT X9, X2	0x5920
0x1c	SLA X10, X3	0x6A30
0x1d	SRA X11, X2	0x7B20
0x1e	RET X15	0xB0F0
:	: : : :	:
0x20	LW X2, X0	0xD200
0x21	SUB X0, X0, X1	0x1001
0x22	LW X3, X0	0xD300
0x23	ADD X8, X2, X3	0x0823
0x24	SUB X9, X2, X3	0x1923
0x25	AND X10, X2, X3	0x2A23
0x26	OR X11, X2, X3	0x3B23
0x27	SUB X13, X3, X2	0x1D32
0x28	BLZ X13, -25	0x8DE7
:	: : : :	:
0x3B	SW X4, X8	0xE048
0x3C	SW X5, X9	0xE059
0x3D	SW X6, X10	0xE06A
0x3E	SW X7, X11	0xE07B
0x3F	HLT	0xF000

We simulated both the non-pipeline and pipelined design to see the improvement of performance.

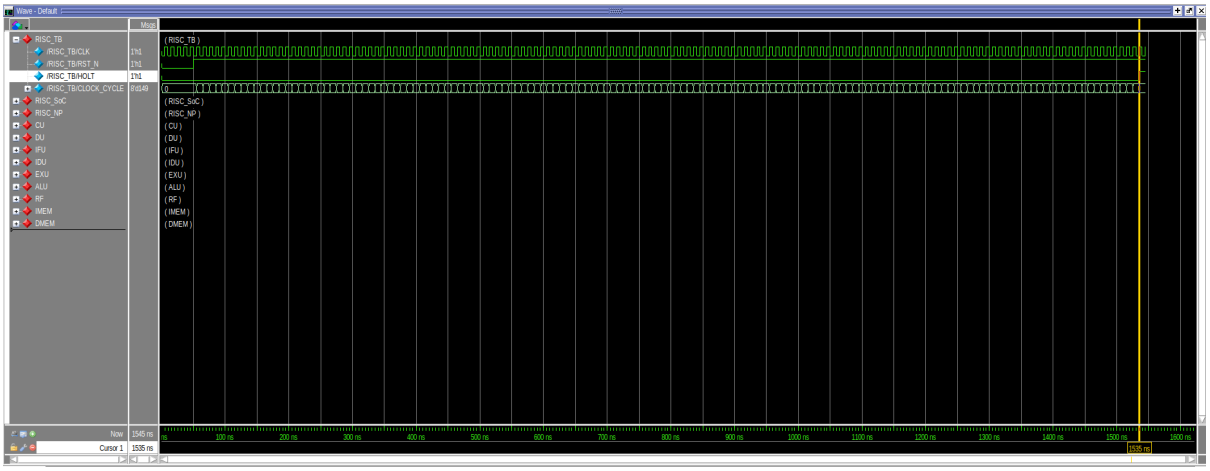

Figure 9: Simulation of the non-pipeline version of the processor.

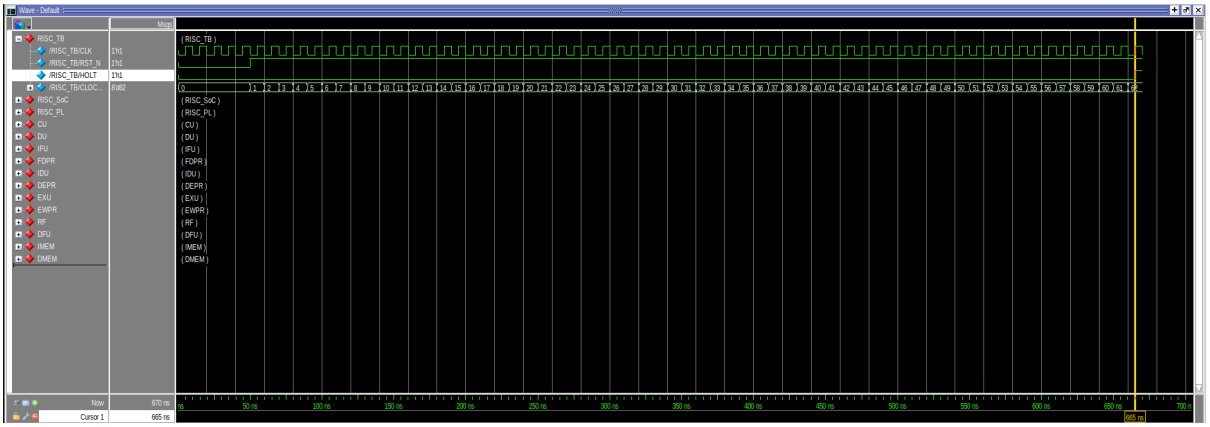

Figure 10: Simulation of pipelined processor.

RESULTS AND DISCUSSION

After simulating both the processors at $100MHz$ we found that for the same assembly code, the pipelined processor takes $665ns$ whereas the non-pipelined processor takes $1535ns$ to finish simulation. So, we are getting a performance improvement,

$$\frac{1535ns}{665ns} \cong 2.31$$

Comparison

We added a clock cycle counter in the test bench which increments every clock cycle after reset signal goes high and counts until holt signal goes high which also stops simulation.

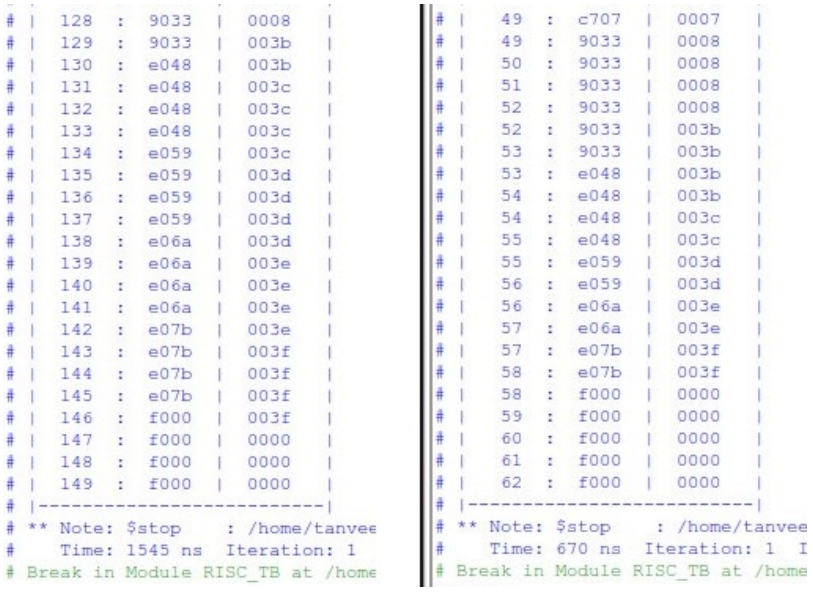

Figure 11: Clock cycle comparison of non-pipelined (left) vs pipelined (right).

We can see that the non-pipelined version takes 149 clock cycles and pipelined takes 62 clock cycles giving improvement of,

$$\frac{149}{62} \cong 2.4$$

We also implemented the designed processors in Xilinx FPGA chip using Vivado to find the estimated power, resource utilization, and maximum achievable frequency of the two version.

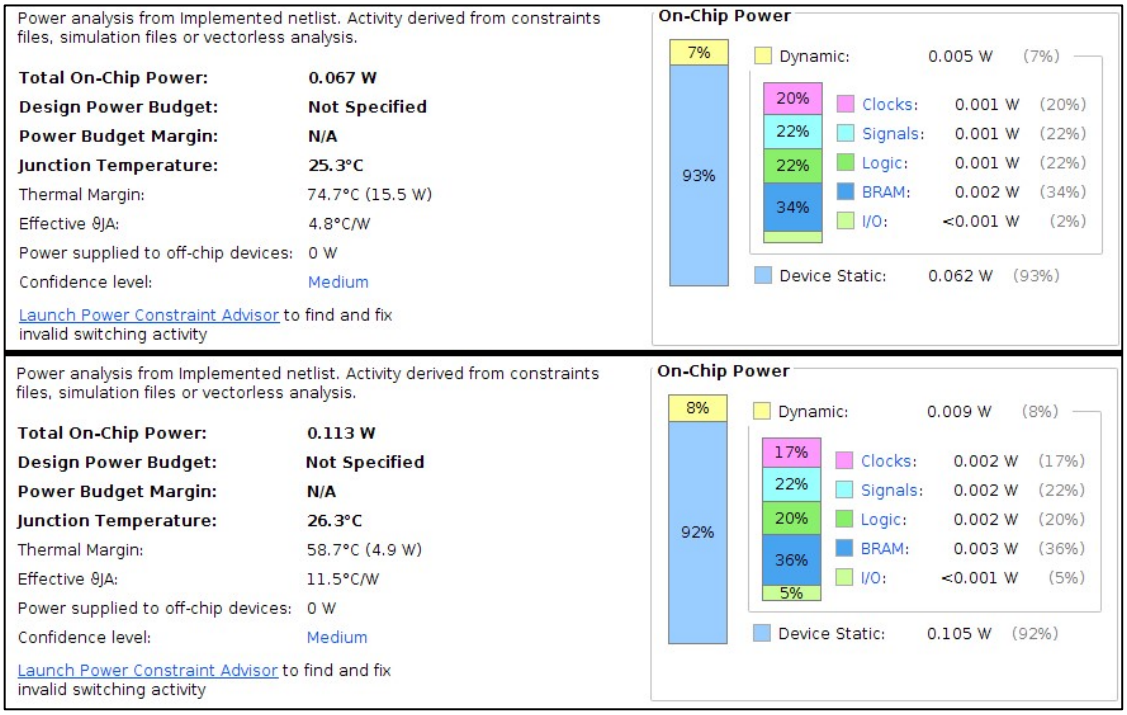

Figure 12: Power estimation comparison. (Top – Non-Pipeline , Bottom – Pipelined)

We can see that the non-pipeline version takes $0.067W$ total power of which $0.005W$ is dynamic power. Pipeline processor takes in $0.113W$ of total power of which $0.009W$ is dynamic power.

Name	Slice LUTs (32600)	Slice Registers (65200)	Slice (8150)	LUT as Logic (32600)	LUT as Memory (9600)	Block RAM Tile (75)	DSPs (120)
N risc_soc_synth	161	94	49	129	32	94	1
DMEM (blk_mem_gen_0)	2	0	1	2	0	0.5	0
IMEM (blk_mem_gen_0_HD2)	0	0	0	0	0	0.5	0
RISC_NP (risc_core_np)	159	94	49	127	32	0	0
CU (control_unit)	0	6	4	0	0	0	0
DU (datapath)	159	88	49	127	32	0	0
EXU (exec_unit)	0	16	16	0	0	0	0
ALU (arth_logic_unit)	0	0	4	0	0	0	0
IDU (inst_decode_unit)	50	40	32	50	0	0	0
SEU (signextender)	7	8	7	7	0	0	0
IFU (inst_fetch_unit)	78	32	33	78	0	0	0
RF (regfile)	32	0	8	0	32	0	0

Name	Slice LUTs (53200)	Slice Registers (106400)	Slice (13300)	LUT as Logic (53200)	LUT as Memory (17400)	Block RAM Tile (140)	DSPs (220)
N risc_soc_synth	231	140	73	203	28	140	1
DMEM (blk_mem_gen_0)	2	0	1	2	0	0.5	0
IMEM (blk_mem_gen_0_HD2)	0	0	0	0	0	0.5	0
RISC_PL (risc_core_pl)	229	140	73	201	28	0	0
CU (control_unit)	1	2	2	1	0	0	0
DU (datapath)	228	138	73	200	28	0	0
DEPR (id_ex_pl_reg)	101	48	38	101	0	0	0
EWPR (ex_wb_pl_reg)	21	26	28	21	0	0	0
EXU (exec_unit)	0	0	4	0	0	0	0
ALU (arth_logic_unit)	0	0	4	0	0	0	0
FDPR (if_id_pl_reg)	39	16	33	39	0	0	0
IFU (inst_fetch_unit)	39	32	22	39	0	0	0
RF (regfile)	28	0	7	0	28	0	0

Figure 13: FPGA resource utilization comparison. (Top – Non-Pipeline , Bottom – Pipelined)

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	4.990 ns	Worst Hold Slack (WHS):	0.105 ns	Worst Pulse Width Slack (WPWS):	3.750 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	573	Total Number of Endpoints:	573	Total Number of Endpoints:	163

All user specified timing constraints are met.

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	4.609 ns	Worst Hold Slack (WHS):	0.056 ns	Worst Pulse Width Slack (WPWS):	3.750 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	550	Total Number of Endpoints:	550	Total Number of Endpoints:	201

All user specified timing constraints are met.

Figure 14: Timing Summary at 100MHz clock. (Top – Non-Pipeline , Bottom – Pipelined)

The picture above shows the worst negative slack of the designs when timing constraint was $10.0ns$ or $100MHz$.

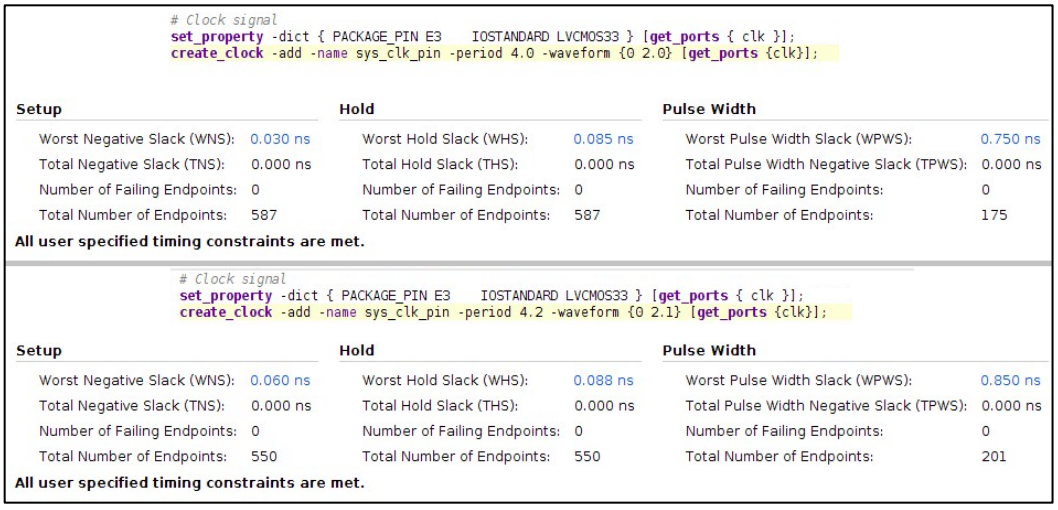

Figure 15: Maximum achieved frequency and timing comparison. (Top – Non-Pipeline , Bottom – Pipelined)

As seen, non-pipeline version achieved a maximum frequency of $250MHz$, and pipelined processor could go up to $238MHz$.

We also observe increase in power and resource utilization as we increase frequency.

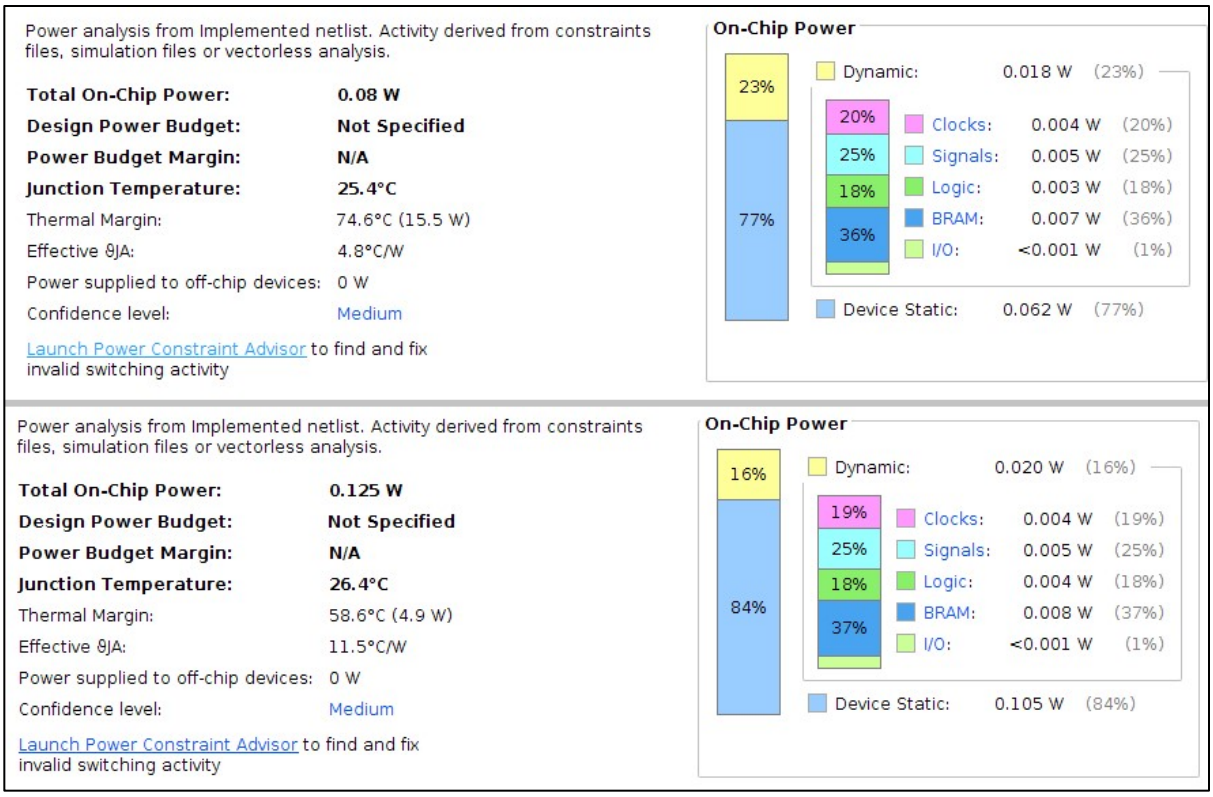

Figure 16: Increased power usage comparison. (Top – Non-Pipeline , Bottom – Pipelined)

The pipelined design increased power from $0.113W$ to $0.125W$ (dynamic $0.009W$ to $0.020W$), and non-pipelined design from 0.067W to 0.080W (dynamic $0.005W$ to $0.018W$).

CONCLUSIONS AND FUTURE DIRECTIONS

The project helped us understand different aspect and challenges of designing a pipelined processor. The non-pipelined design was comparatively straightforward as there was of overlap between instructions. In the pipelined design, the most complicated component was to changes needed in the control unit so that different instructions can be executed it the same with in different stages. Overlap of resource requirement of these instruction at the same time necessitated additional hardware to be added to be design. The project also expanded our understanding of the relationship between speed, power, and area in digital design.

As our pipelined processor does not have a branch predictor, we had to use stall cycles to make delay execution until the branch is resolved and program counter is updated with new instruction address. This has a significant negative effect on the performance of the processor. In the future a branch predictor should be added to improve upon the current design.

REFERENCES

[1] J. L. Hennessy, D. A. Patterson, *Computer Architecture: A Quantitative Approach*
[2] J. L. Hennessy, D. A. Patterson, *Computer Organization and Design: The Hardware/Software Interface - RISC-V Edition*
[3] R. K. S. Parihar, S. Reddy, *A Report on Design Of 16 – Bit RISC Processor*