**Implementation of KLHY21 RISC Processor**

**Suryanshu Basa ECE 2110040027 - KLHY21**

**KLHY21 is an 8-bit RISC processor designed to perform limited set of data transfer, arithmetic, logical and IO interface instructions. The processor is based on Harvard architecture. Each instruction is executed in single cycle with Pipelined architecture. The instruction word length is 16 bits and supports 26 instructions. It has 8 general purpose registers to store 8 bytes of data. The processor has on chip Data and Program Memory and also has a limited Stack memory. The scope of this project is to design and implement the processor using Xilinx FPGA.**

**RISC**

A **R**educed **I**nstruction **S**et **C**omputer is a type of microprocessor architecture that utilizes a small, highly-optimized set of instructions rather than the highly-specialized set of instructions typically found in other architectures.

**Characteristic of RISC**

* Simpler instruction, hence simple instruction decoding.
* Instruction comes undersize of one word.
* Instruction takes a single/multiple clock cycle to get executed.
* More general-purpose registers.
* Simple Addressing Modes.
* Fewer Data types.
* A pipeline can be achieved.

**Specification:**

* + 8-bit Data and Address bus.
  + 256x16 bit on Chip Program Memory.
  + 256x8 bit on Chip Data Memory.
  + 16-bit Instruction format. Supports up to 32 instructions including data transfer, arithmetic-logic functions and branch instructions.
  + 8 x 8-bit internal registers for data storage.
  + 8-bit ALU to support all arithmetic and logical operations.
  + Support for conditional and unconditional branch instructions.
  + Serial port interface to transfer serial data bit stream to external world.
  + PPI interface to transfer 8-bit parallel data to external world.
  + Timer circuit to generate programmable clock pulses.
  + Pipelined architecture, executes each instruction in single clock cycle.
  + 16x8 Stack memory with push/pop support.

**Architecture Diagram:**

**Prog memory**

**Prog Counter**

**Instr**

**Register**

**ALU**

**Control**

**Unit & Signal generator**

**Register Bank**

**Serial Interface**

**Data Memory**

Data Bus

Data Bus

DataData Bus

**Data Bus**

**Address Bus**

**Control Bus**

**Timer**

**PPI**

**Stack**

**Mem**

**Design Hierarchy:**

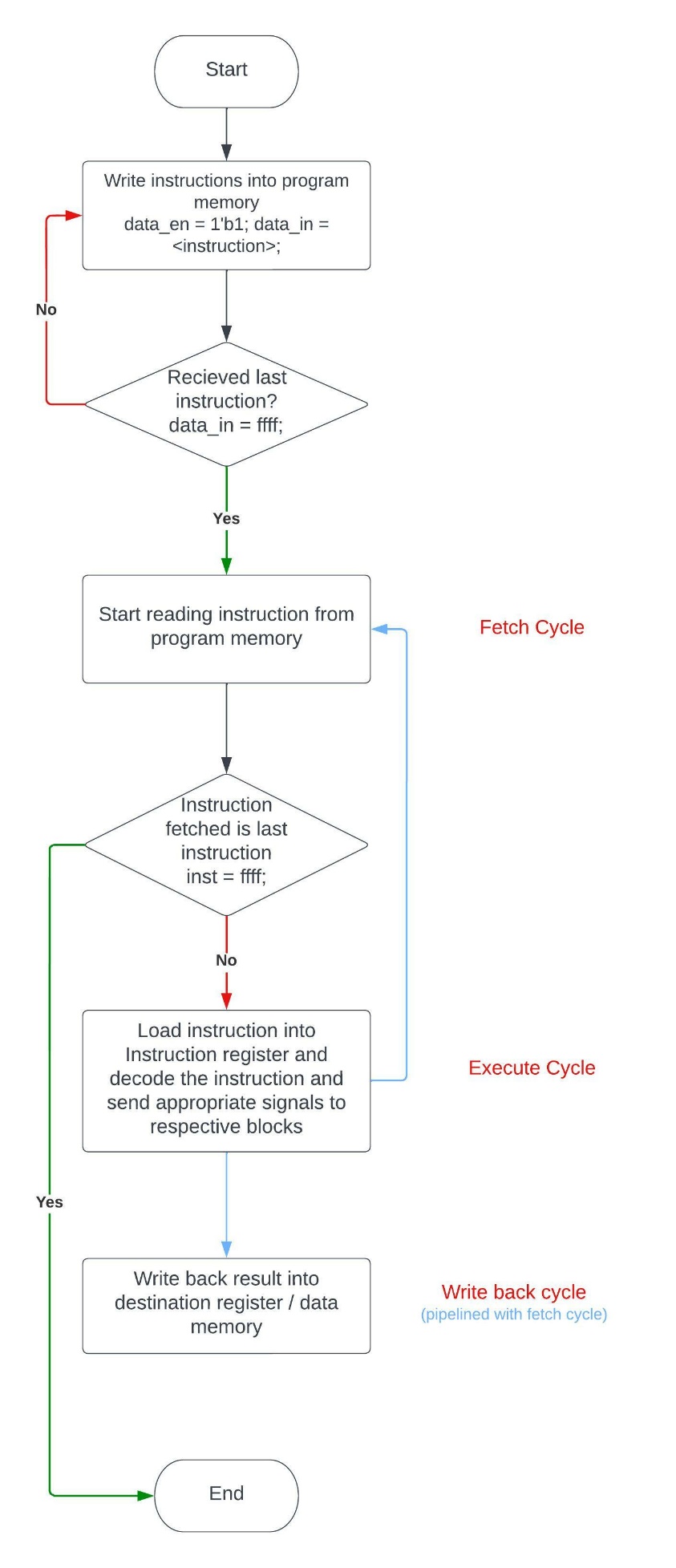
1. Control Unit
2. Program Counter
3. Instruction Register
4. Program Memory
5. Data Memory
6. ALU
7. Register Bank
8. Program Peripheral Interface
9. Serial Interface
10. Timer
11. Stack Memory
12. Signal Generator
13. Data Mux

**Addressing modes:**

Following addressing modes are used for framing the instructions.

1. Immediate Addressing mode.
2. Register Addressing mode.
3. Direct Addressing mode.

**Execution Flowchart:**



**Timing Diagram of Pipelining:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Cycle 1** | **Cycle 2** | **Cycle 3** | **Cycle4** | **Cycle5** | **Cycle6** |
| **Instruction 1** | **IF** | **IE** | **WB** |  |  |  |
| **Instruction 2** |  | **IF** | **IE** | **WB** |  |  |
| **Instruction 3** |  |  | **IF** | **IE** | **WB** |  |
| **Instruction 4** |  |  |  | **IF** | **IE** | **WB** |

**Instructions:**

Total 26 instructions have been supported covering Data transfer, arithmetic and Logic, ppi, serial interface, programmable timer, stack memory and branch instructions.

**Instruction opcodes**

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Instruction** | **Opcode** |
| 1 | MOV – Data transfer within Registers | 00001 |
| 2 | MOVI – Move Immediate data into Register | 00010 |
| 3 | DMW – Data Memory Write | 00011 |
| 4 | DMR – Data Memory Read | 00100 |
| 5 | PPW – Program Peripheral Write | 00101 |
| 6 | PUSH | 00110 |
| 7 | POP | 00111 |
| 8 | ADD – Arithmetic Add | 01000 |
| 9 | SUB – Arithmetic Subtraction | 01001 |
| 10 | MUL – Arithmetic Multiply | 01010 |
| 11 | DIV – Arithmetic Division | 01011 |
| 12 | AND – Logical AND | 01100 |
| 13 | OR – Logical OR | 01101 |
| 14 | NOT – Logical NOT | 01110 |
| 15 | XOR – Logical XOR | 01111 |
| 16 | LSH – Arithmetic Left Shift | 10000 |
| 17 | RSH – Arithmetic Right Shift | 10001 |
| 18 | INC – Arithmetic Increment | 10010 |
| 19 | DEC – Arithmetic Decrement | 10011 |
| 20 | TIMEROUT – Timer Pulse out | 10100 |
| 21 | SERIALOUT – Serial Data Out | 10101 |
| 22 | JMP – Unconditional Branch | 11001 |
| 23 | JMPC – Conditional Branch when Carry Flag is set | 11010 |
| 24 | JMPZ – Conditional Branch when Zero Flat is set | 11011 |
| 25 | JMPP – Conditional Branch when Parity Flat is set | 11100 |
| 26 | WAIT – introduce wait cycles (no operation) | 11101 |

**Note:** JMP\* instructions must be followed by a WAIT instruction to allow the program counter to load with the branching address.

**Instruction Format:**

**MOV <Destination Reg> < Source Reg>**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Opcode (5 bits) | Unused (2 bits) | Destination Register (3 bits) | Unused (3 bits) | Source Register (3 bits) |

**MOVI <register> #8-bit data**

|  |  |  |
| --- | --- | --- |
| Opcode (5 bits) | Destination Register (3 bits) | Immediate Data (8 bits) |

**DMW/DMR <register> #8-bit address**

|  |  |  |
| --- | --- | --- |
| Opcode (5 bits) | Destination Register (3 bits) | Data Memory Address (8 bits) |

**ADD/SUB/MUL/DIV/AND/OR/XOR <Destination Reg> <Source Register>**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Opcode (5 bits) | Unused (2 bits) | Destination Register (3 bits) | Source 2 Register (3 bits | Source 1 Register (3 bits) |

**Push/Pop/PPW/INC/DEC <Source register>**

|  |  |  |
| --- | --- | --- |
| Opcode (5 bits) | Unused (8 bits) | Source Register (3 bits) |

**JMP/JMPC/JMPZ/JMPP #8-bit program memory address**

|  |  |  |
| --- | --- | --- |
| Opcode (5 bits) | Unused (3 bits) | program Memory Address (8 bits) |

**Instruction examples.**

1. MOV BH, AL
2. MOVI BH #8bit Data
3. DMW BH, #8-bit Data Memory Address
4. DMR BH, #8-bit Data Memory Address
5. PPW BH
6. Push BH
7. Pop BH
8. ADD BH, AH, AL
9. SUB BH, AH, AL
10. MUL BH, AH, AL
11. DIV BH, AH, AL
12. AND BH, AH, AL
13. OR BH, AH, AL
14. NOT AH, AL
15. XOR BH, AH, AL
16. LSH AL, AH
17. RSH AL, AH
18. INC BH
19. DEC CL
20. TIMEROUT AL
21. SERIALOUT AL
22. JMP <#8-bit Prog Memory Address>
23. JMPC <#8-bit Prog Memory Address>
24. JMPZ <#8-bit Prog Memory Address>
25. JMPP <#8-bit Prog Memory Address>
26. WAIT

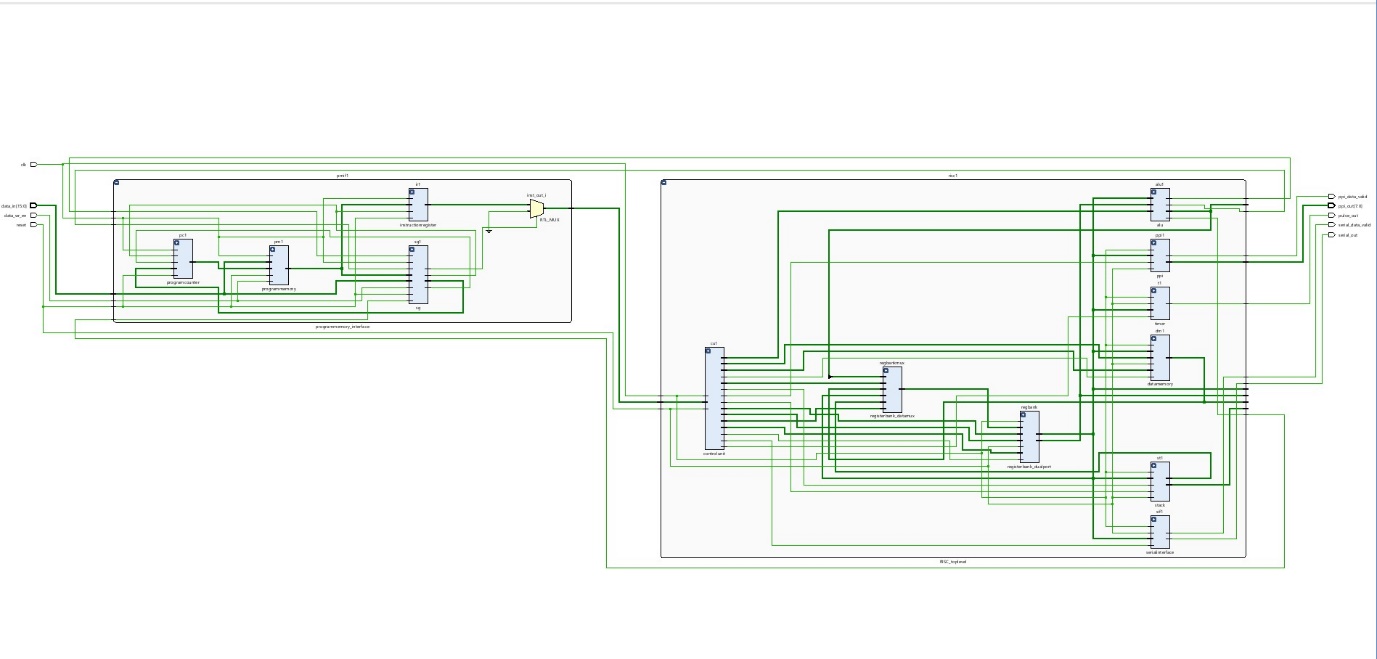
**Tool:**

Xilinx VIVADO is used for this project from RTL to Implementation.

**RTL:**

Verilog HDL is used for coding the behaviour of the Design.

**RTL Implementation schematic:**



**Verification Environment:**

Test bench is used to apply different instructions to test the functionality of the design and obtained the output from the design. The output from the Design is verified manually.

**Test Bench**

**Stimulus**

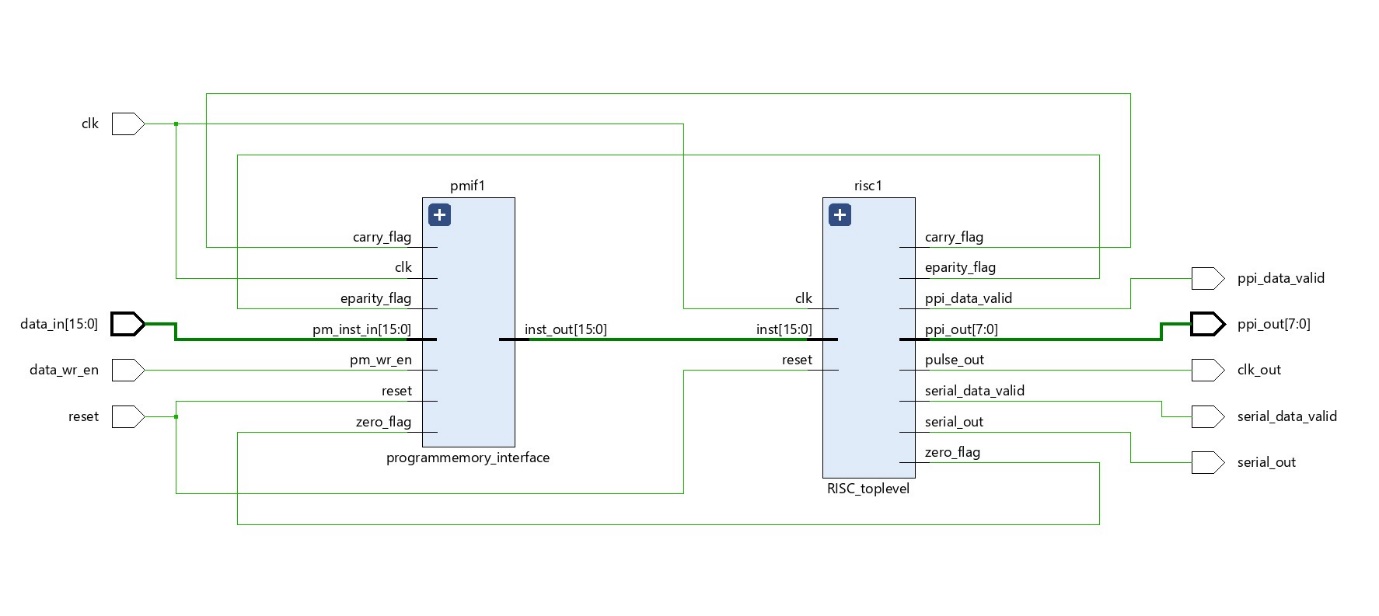
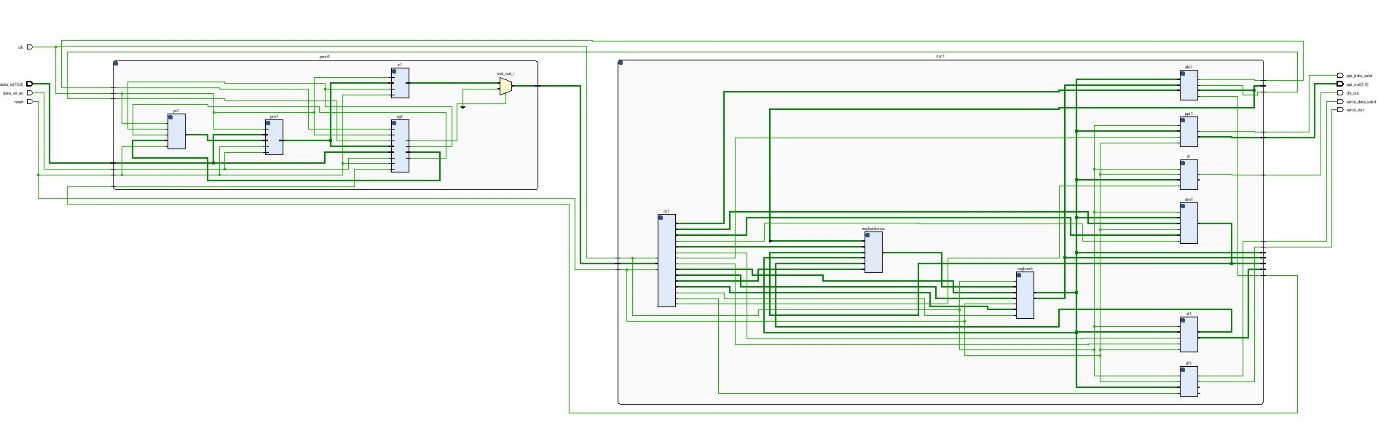
**RISC\_TOP**

**KLHY21 Design Units :**

**KLHY21\_top :**

KLHY21 is the top level design. This will have 2 submodules RISC\_TOP and programmemory\_interfae.

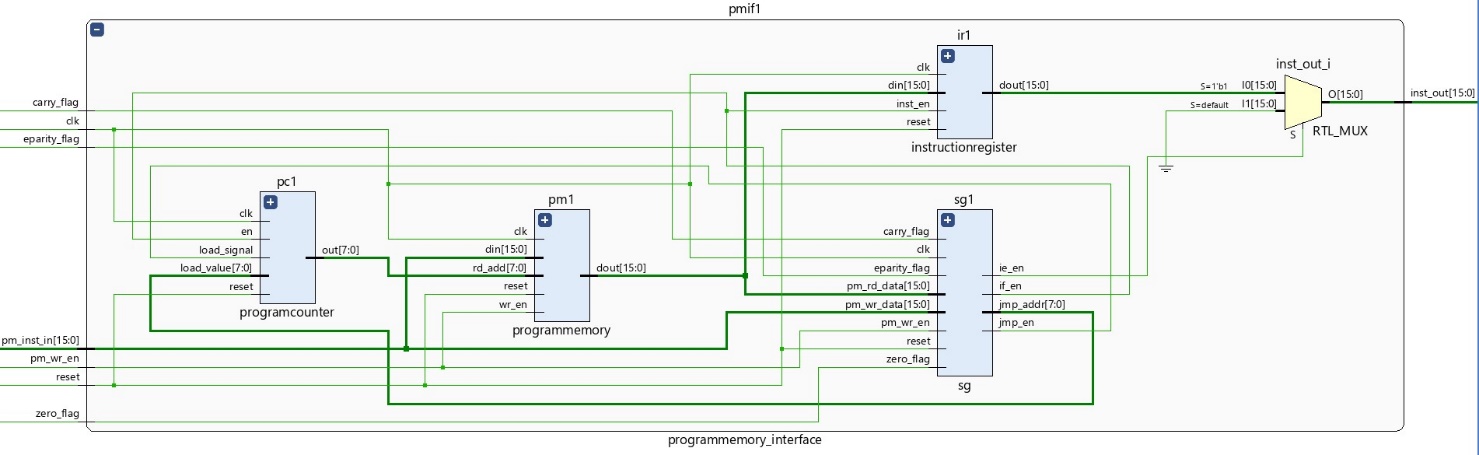
1. Program\_memoryinterface is for fetcing/storing the instruction and also responsible for generating the control signals for these operations.
2. RISC\_TOP contrains all the sub modules which are responsible for decoding and executing the fetched Insturciotions.

**Program memory interface:**

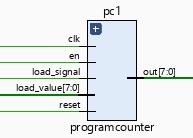
Program Memory interface have the following sub-blocks.

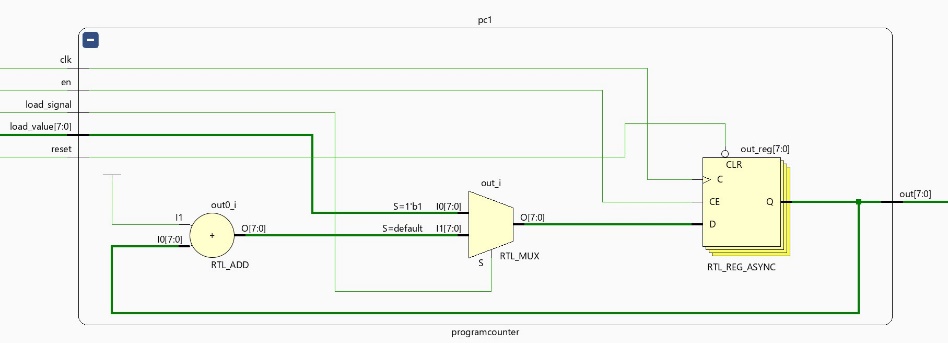
1. Program counter
2. Program Memory
3. Signal Generator
4. Instruction Register



**Program Counter:**

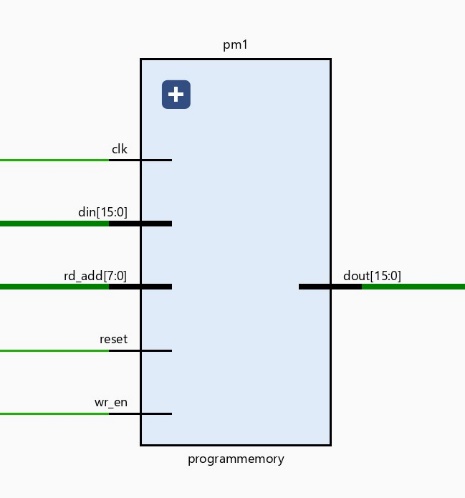
Program counter will supply the address of the program memory from which the instruction is fetched. Once every instruction is fetched, the address will be incremented by 1. When there is Jump instruction, the program counter will be loaded with the Jump address supplied by the instruction.





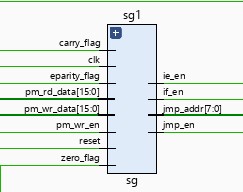
**Program Memory:**

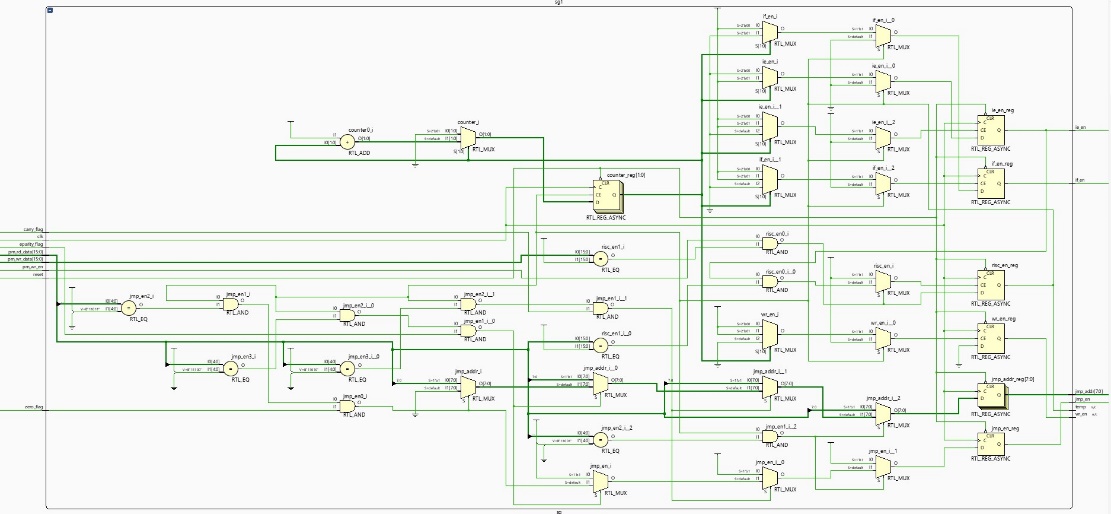
Program memory is 256x16 in size and is used to store the instructions.



**Signal Generator:**

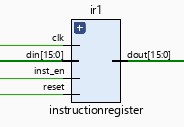
Signal generator is a mini control unit, generates the control signals for the program memory, program counter and instruction register.

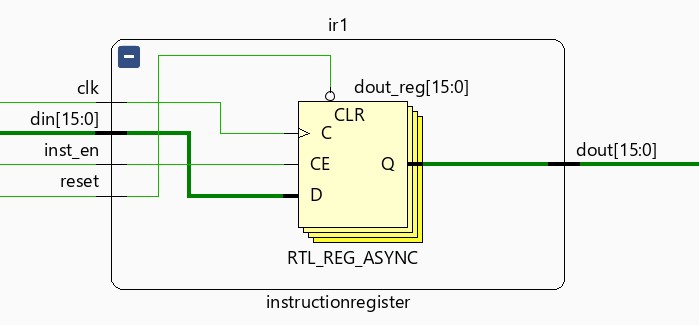




**Instruction Register:**

Instruction register is used for holding the instruction which is fetched from the program memory.



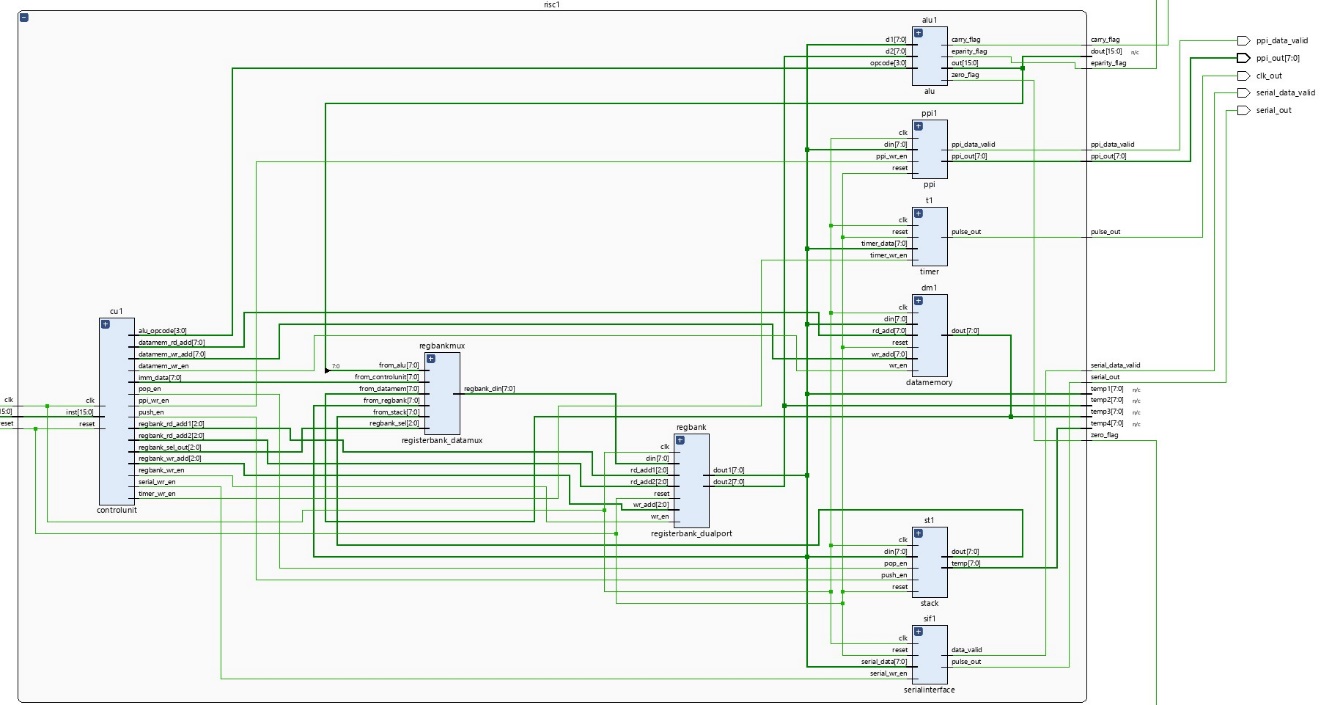


**Risc Toplevel:**

RISC top-level will have all the blocks which are directly controlled by the Instruction Control unit and are responsible for the Instruction execution.

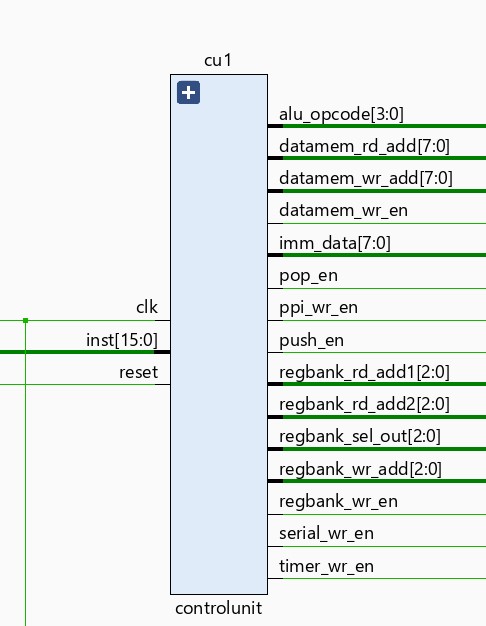
RISC Top have the following sub-blocks.

1. Control Unit
2. Reg Bank
3. Data Memory
4. Serial Interface
5. Timer
6. PPI
7. ALU
8. Stack



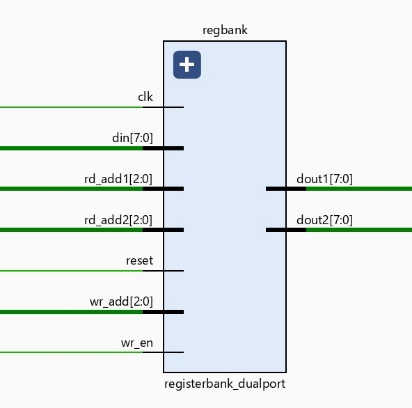
**Control Unit:**

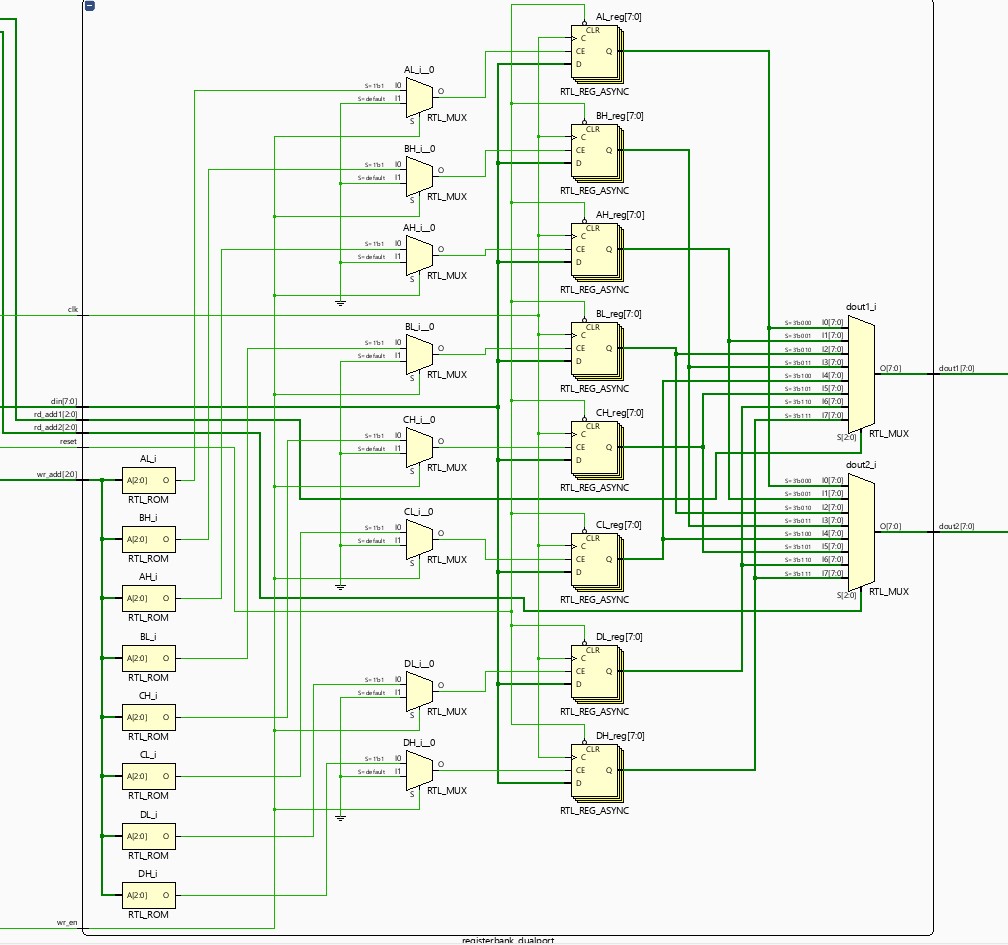
Control unit is the heart of the design. This unit will generate all the necessary control signals for all the blocks to execute the instruction.



**Register Bank:**

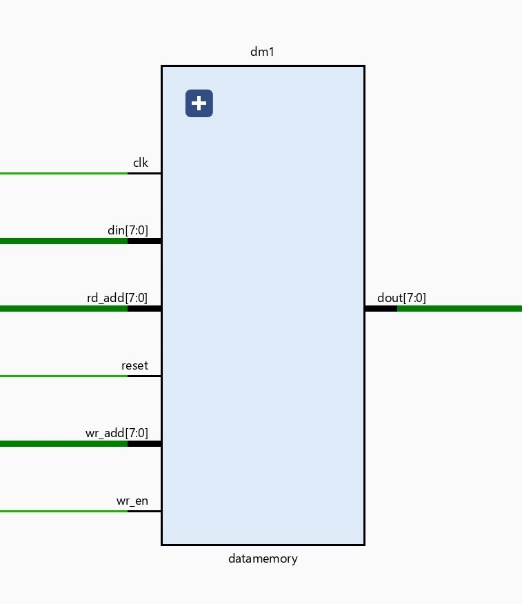
Register bank is group of 8 registers to store the internal data resulting from the execution of the Instructions.



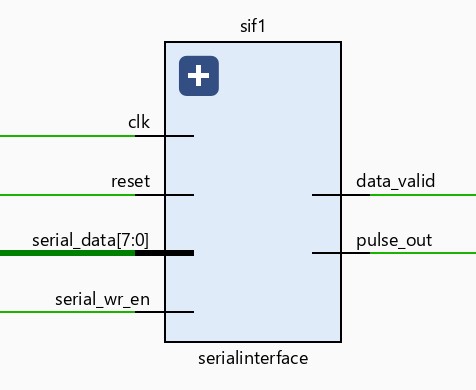


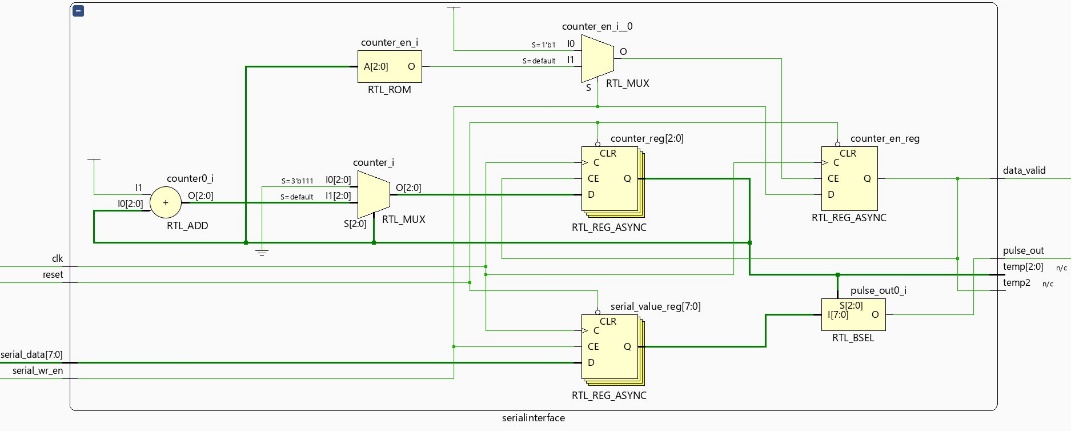
**Data Memory:**

Data memory is 256x8 in size and is used to store the internal data resulting from the Instruction execution.



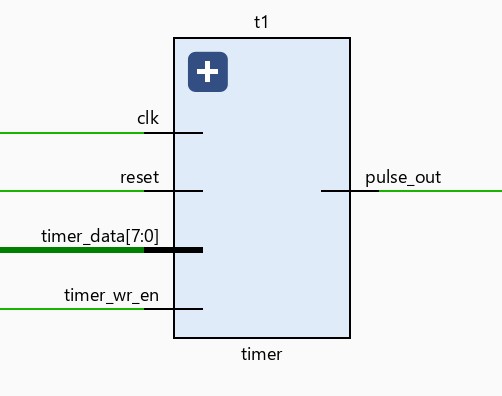
**Serial Interface:**

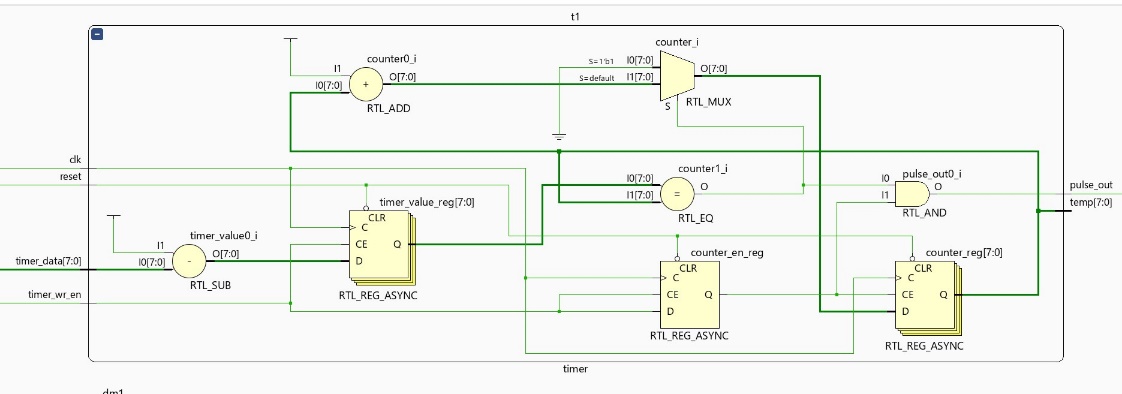
Serial interface is used for transferring the data out of the processor in a serial data format (1 bit at a time). The data is placed on the output along with the data\_valid signal.



**Timer:**

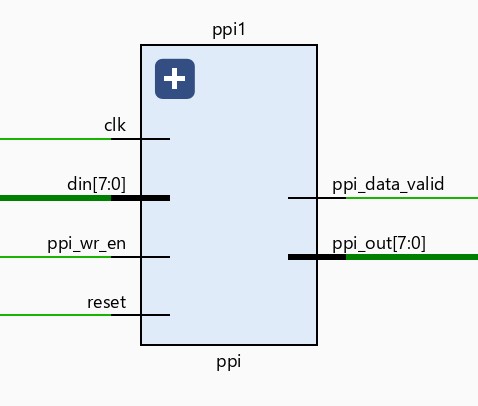
Timer circuit is used as a clock pulse generator for the external devices. It can be programmed to generate the clock pulse signal based on the programable count value. The generated clock pulse output will be of one clock duration.

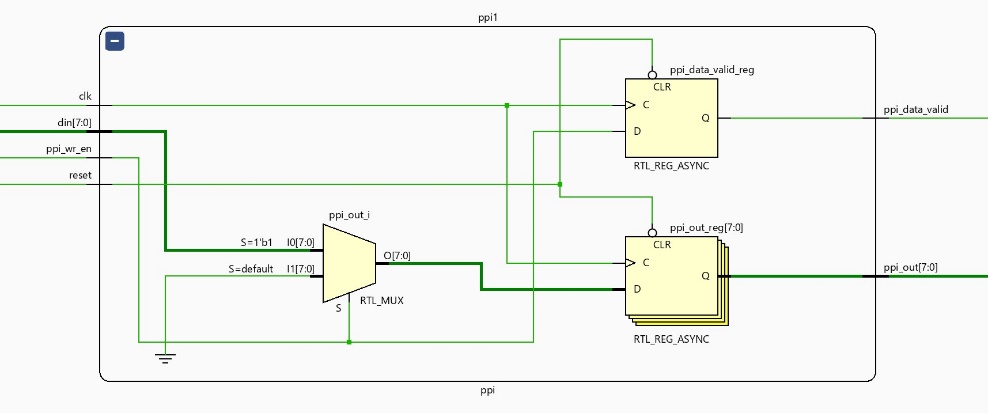




**PPI (Programmable Peripheral Interface):**

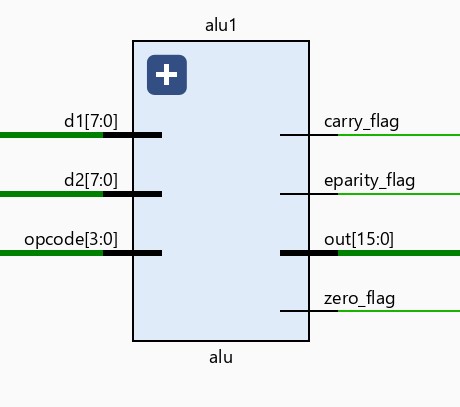
PPI is used to transfer parallel data to the external devices from the processor. The data is placed on the output along with the data\_valid signal.

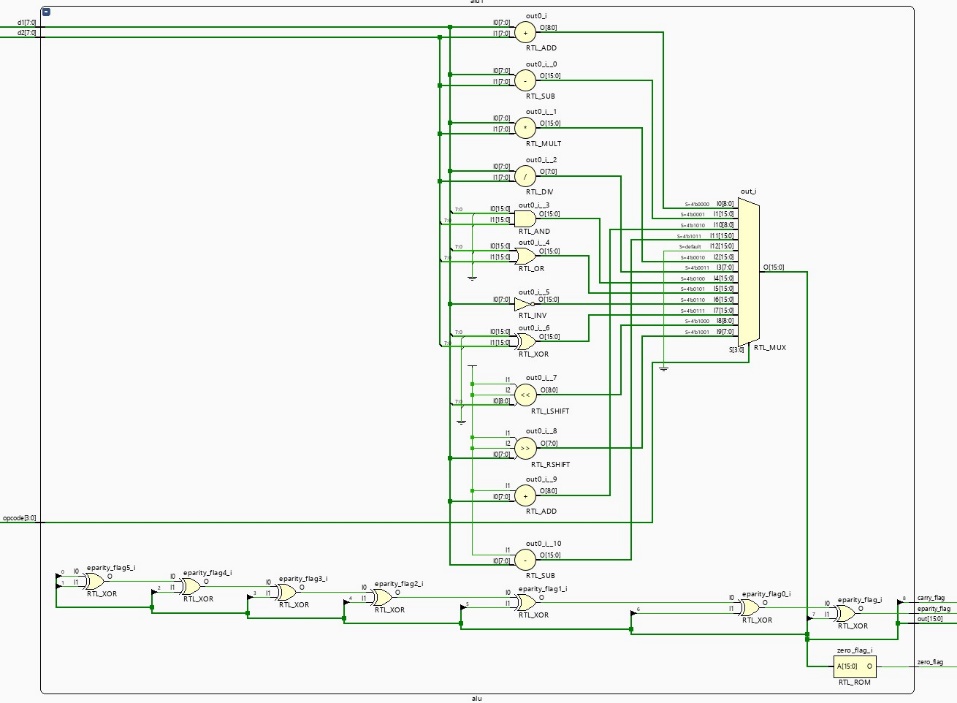




**ALU:**

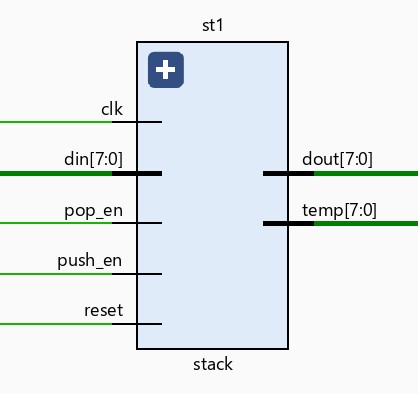
ALU is Arithmetic and Logic Unit used for executing all the Arithmetic and Logic instructions.





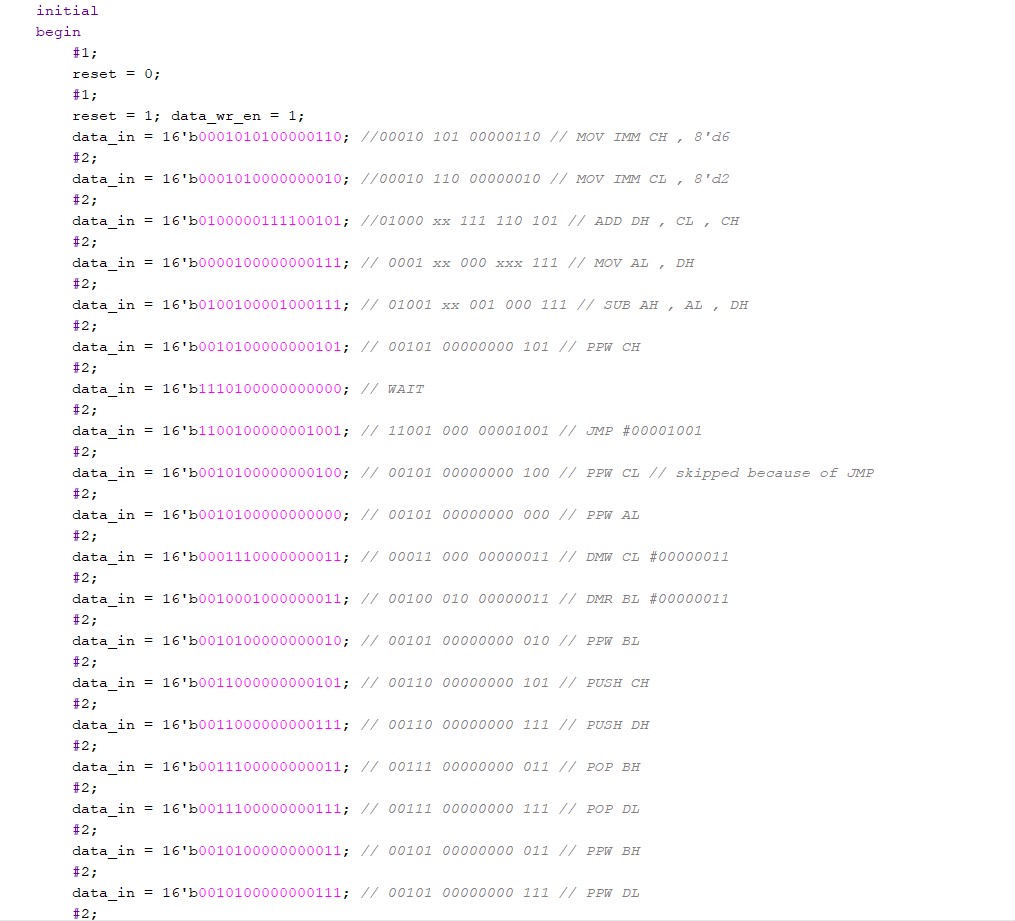
**Stack:**

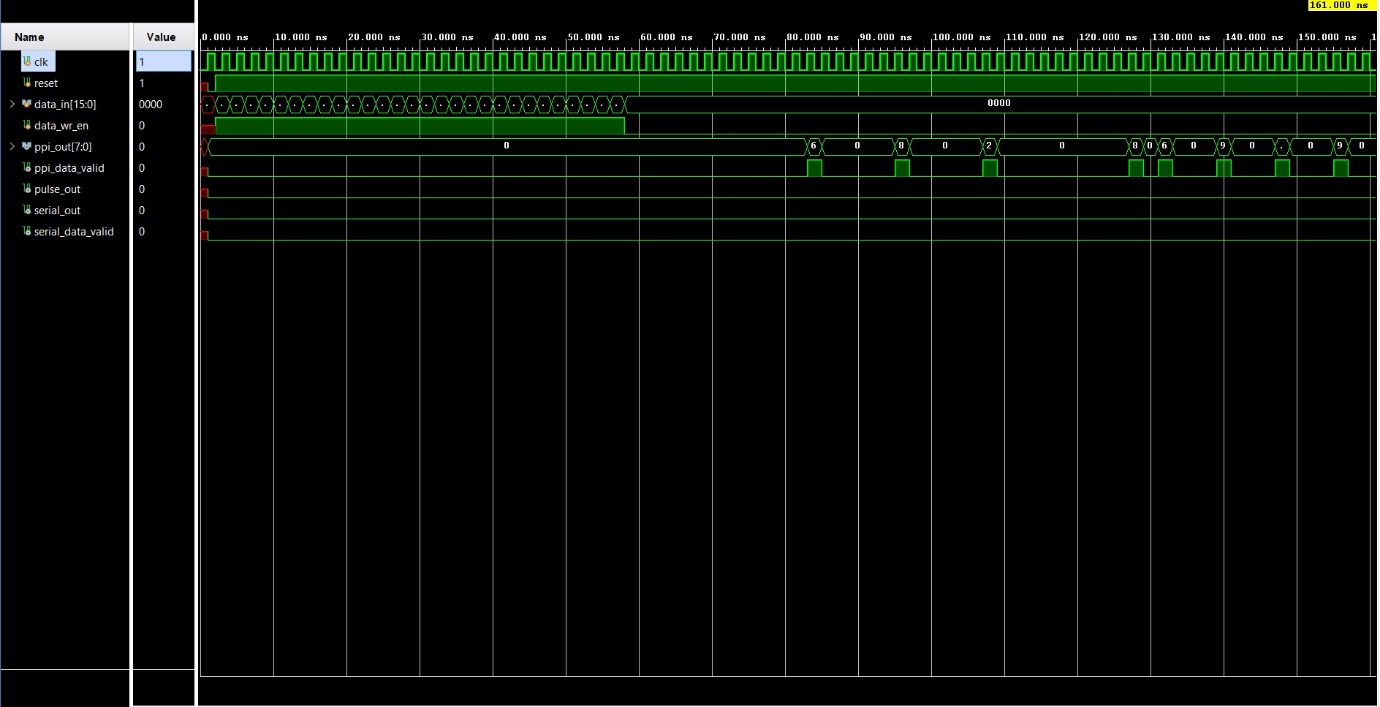
Stack memory is used for storing internal data which works on the Last-In-First-Out principle. This will have 16x8 memory space.



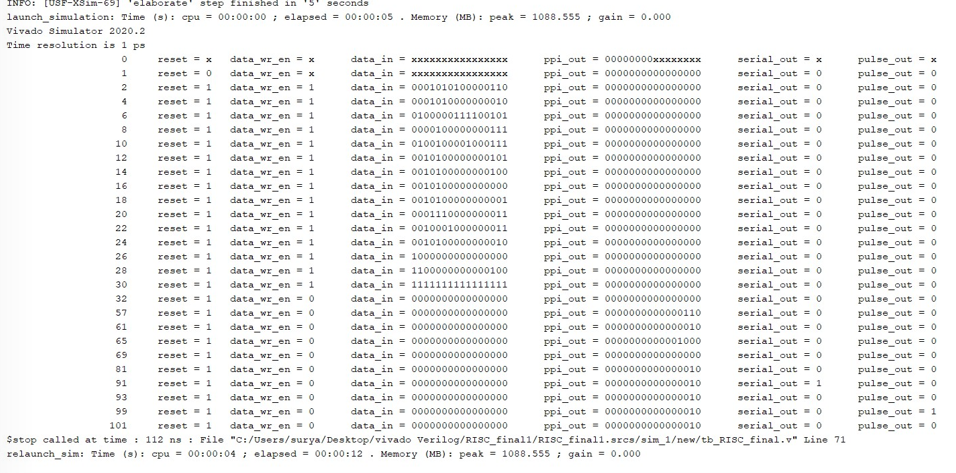
**Simulation results and Waveforms:**

**Stimulus applied from Test Bench.**



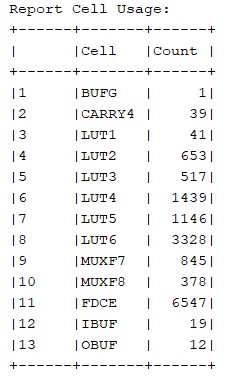
**Simulation output Waveform**

**Data output from simulation**

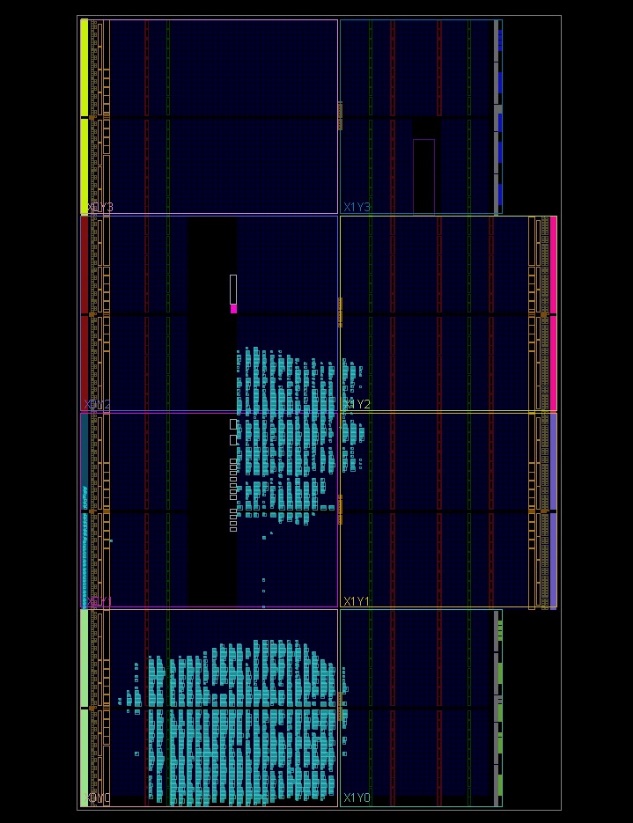


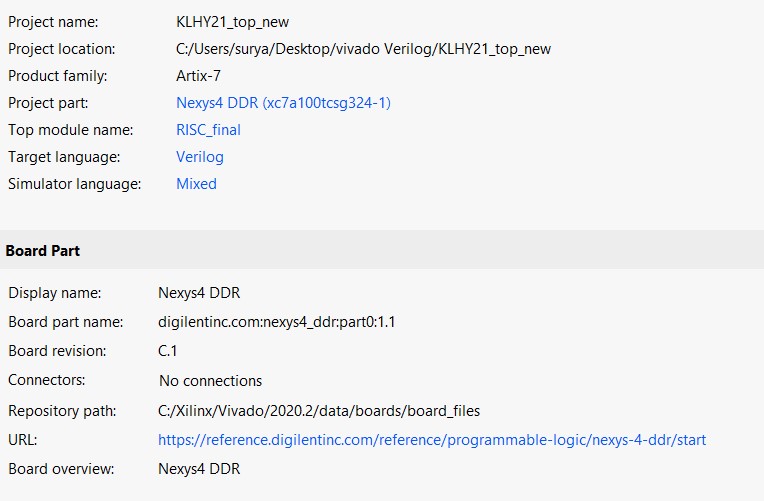
**Synthesis Results:**

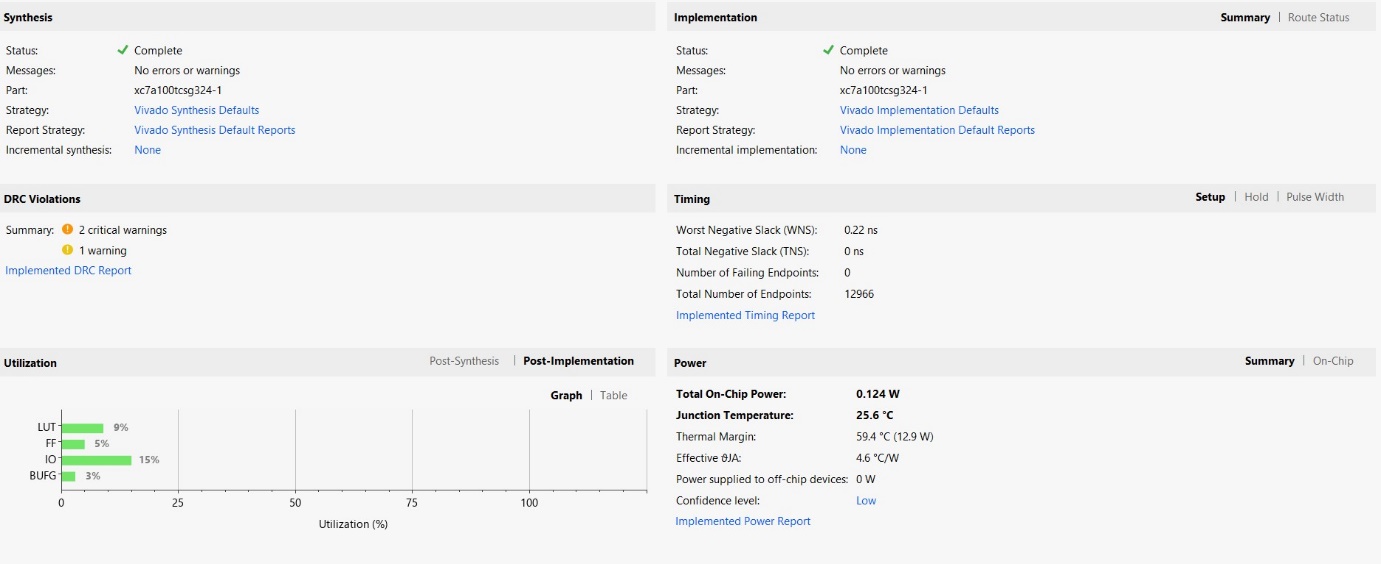
Total Gate Count after Synthesis is **14.965 K** Gates.



**Implementation Results:**



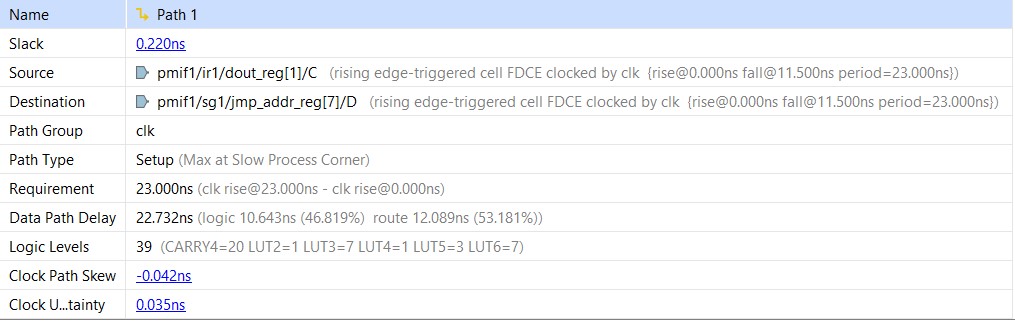




**Timing Summary:**

* Max Clock Freq of operation: **43.47 MHz**

Worst Path:



**Observation and Conclusion:**

Current implementation has hardware support for the MUL/DIV arithmetic operations with which we are getting the output in single cycle. The drawback with this implementation is it is adding additional cells and timing is becoming critical as the path through the MULT/DIV logic is taking longer delays and becoming critical paths. This implementation is limiting us to **43.47Mhz** frequency of operation.

We also have support for the MULT/DIV operation with instructions (MULT= repeated addition and DIV=repeated subtraction). This support does not need the MULT/DIV hardware and can be removed. When we removed the MULT/DIV logic, the cell count decreased and the critical path length also decreased. With this implementation, the processor can run at **71.2 Mhz**.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Area | clock freq achieved | Critical path Logic Levels |
| Mult/Div Hardware support | 14.96 K | 43.47 Mhz | 39 |
| Mult/Div software support (no hardware) | 14.79 K | 71.2 Mhz | 3 |