EE705

INTEGRATION PROJECT PART B

TEAM PANCHJANYA

TEAM SILICON TITANS

Here we need to integrate SOC, AXI4Lite to AXI4 Converter, Interrupt Control, RISCV Core and RISCV Top to build a RISCV Processor.

We have integrated all relevant modules to make RISCV Top and SOC separately and have tested them with corresponding test cases.

RISCV TOP MODULE

Verilog Code for Fetch Module is shown as under:

riscv_top.v

C:/VLSI Design Lab/final_top_integ/final_top_integ/final_top_integ.srcs/sources_1/imports/integmod_codes/riscv_top.v

```
module riscv_top
(
    // Inputs
     input           clk_i
    ,input           rst_i
    ,input           axi_i_awready_i
    ,input           axi_i_wready_i
    ,input           axi_i_bvalid_i
    ,input  [  1:0]  axi_i_bresp_i
    ,input  [  3:0]  axi_i_bid_i
    ,input           axi_i_arready_i
    ,input           axi_i_rvalid_i
    ,input  [ 31:0]  axi_i_rdata_i
    ,input  [  1:0]  axi_i_rresp_i
    ,input  [  3:0]  axi_i_rid_i
    ,input           axi_i_rlast_i
    ,input           axi_d_awready_i
    ,input           axi_d_wready_i
    ,input           axi_d_bvalid_i
    ,input  [  1:0]  axi_d_bresp_i
    ,input  [  3:0]  axi_d_bid_i
    ,input           axi_d_arready_i
    ,input           axi_d_rvalid_i
    ,input  [ 31:0]  axi_d_rdata_i
    ,input  [  1:0]  axi_d_rresp_i
    ,input  [  3:0]  axi_d_rid_i
    ,input           axi_d_rlast_i
    ,input           intr_i
    ,input  [ 31:0]  reset_vector_i

```

```
31
32      // Outputs
33      ,output          axi_i_awvalid_o
34      ,output [ 31:0]  axi_i_awaddr_o
35      ,output [  3:0]  axi_i_awid_o
36      ,output [  7:0]  axi_i_awlen_o
37      ,output [  1:0]  axi_i_awburst_o
38      ,output          axi_i_wvalid_o
39      ,output [ 31:0]  axi_i_wdata_o
40      ,output [  3:0]  axi_i_wstrb_o
41      ,output          axi_i_wlast_o
42      ,output          axi_i_bready_o
43      ,output          axi_i_arvalid_o
44      ,output [ 31:0]  axi_i_araddr_o
45      ,output [  3:0]  axi_i_arid_o
46      ,output [  7:0]  axi_i_arlen_o
47      ,output [  1:0]  axi_i_arburst_o
48      ,output          axi_i_rready_o
49      ,output          axi_d_awvalid_o
50      ,output [ 31:0]  axi_d_awaddr_o
51      ,output [  3:0]  axi_d_awid_o
52      ,output [  7:0]  axi_d_awlen_o
53      ,output [  1:0]  axi_d_awburst_o
54      ,output          axi_d_wvalid_o
55      ,output [ 31:0]  axi_d_wdata_o
56      ,output [  3:0]  axi_d_wstrb_o
57      ,output          axi_d_wlast_o
58      ,output          axi_d_bready_o
59      ,output          axi_d_arvalid_o
60      ,output [ 31:0]  axi_d_araddr_o
61      ,output [  3:0]  axi_d_arid_o
62      ,output [  7:0]  axi_d_arlen_o
63      ,output [  1:0]  axi_d_arburst_o
64      ,output          axi_d_rready_o
65  );
66
```

```
67  wire           dport_bridge_accept_w;
68  wire           dport_bridge_cacheable_w;
69  wire  [ 31:0]  icache_inst_w;
70  wire  [ 10:0]  dport_bridge_resp_tag_w;
71  wire           dport_bridge_invalidate_w;
72  wire  [ 31:0]  dport_bridge_addr_w;
73  wire  [ 31:0]  icache_pc_w;
74  wire           icache_rd_w;
75  wire           dport_bridge_error_w;
76  wire           icache_flush_w;
77  wire  [ 10:0]  dport_bridge_req_tag_w;
78  wire  [ 31:0]  dport_bridge_data_wr_w;
79  wire           dport_bridge_rd_w;
80  wire           icache_accept_w;
81  wire           icache_valid_w;
82  wire           icache_error_w;
83  wire           dport_bridge_ack_w;
84  wire  [  3:0]  dport_bridge_wr_w;
85  wire  [ 31:0]  dport_bridge_data_rd_w;
86  wire           dport_bridge_flush_w;
87  wire           icache_invalidate_w;
```

```
dport_bridge u_dport_bridge
(
    // Inputs
     .clk_i(clk_i)
    ,.rst_i(rst_i)
    ,.mem_addr_i(dport_bridge_addr_w)
    ,.mem_data_wr_i(dport_bridge_data_wr_w)
    ,.mem_rd_i(dport_bridge_rd_w)
    ,.mem_wr_i(dport_bridge_wr_w)
    ,.mem_cacheable_i(dport_bridge_cacheable_w)
    ,.mem_req_tag_i(dport_bridge_req_tag_w)
    ,.mem_invalidate_i(dport_bridge_invalidate_w)
    ,.mem_flush_i(dport_bridge_flush_w)
    ,.axi_awready_i(axi_d_awready_i)
    ,.axi_wready_i(axi_d_wready_i)
    ,.axi_bvalid_i(axi_d_bvalid_i)
    ,.axi_bresp_i(axi_d_bresp_i)
    ,.axi_bid_i(axi_d_bid_i)
    ,.axi_arready_i(axi_d_arready_i)
    ,.axi_rvalid_i(axi_d_rvalid_i)
    ,.axi_rdata_i(axi_d_rdata_i)
    ,.axi_rresp_i(axi_d_rresp_i)
    ,.axi_rid_i(axi_d_rid_i)
    ,.axi_rlast_i(axi_d_rlast_i)

```

```
    // Outputs
    ,.mem_data_rd_o(dport_bridge_data_rd_w)
    ,.mem_accept_o(dport_bridge_accept_w)
    ,.mem_ack_o(dport_bridge_ack_w)
    ,.mem_error_o(dport_bridge_error_w)
    ,.mem_resp_tag_o(dport_bridge_resp_tag_w)
    ,.axi_awvalid_o(axi_d_awvalid_o)
    ,.axi_awaddr_o(axi_d_awaddr_o)
    ,.axi_awid_o(axi_d_awid_o)
    ,.axi_awlen_o(axi_d_awlen_o)
    ,.axi_awburst_o(axi_d_awburst_o)
    ,.axi_wvalid_o(axi_d_wvalid_o)
    ,.axi_wdata_o(axi_d_wdata_o)
    ,.axi_wstrb_o(axi_d_wstrb_o)
    ,.axi_wlast_o(axi_d_wlast_o)
    ,.axi_bready_o(axi_d_bready_o)
    ,.axi_arvalid_o(axi_d_arvalid_o)
    ,.axi_araddr_o(axi_d_araddr_o)
    ,.axi_arid_o(axi_d_arid_o)
    ,.axi_arlen_o(axi_d_arlen_o)
    ,.axi_arburst_o(axi_d_arburst_o)
    ,.axi_rready_o(axi_d_rready_o)
);
riscv_core u_core
(
    // Inputs
     .clk_i(clk_i)
    ,.rst_i(rst_i)
    ,.mem_d_data_rd_i(dport_bridge_data_rd_w)
    ,.mem_d_accept_i(dport_bridge_accept_w)
    ,.mem_d_ack_i(dport_bridge_ack_w)
    ,.mem_d_error_i(dport_bridge_error_w)
    ,.mem_d_resp_tag_i(dport_bridge_resp_tag_w)
    ,.mem_i_accept_i(icache_accept_w)
    ,.mem_i_valid_i(icache_valid_w)
    ,.mem_i_error_i(icache_error_w)
    ,.mem_i_inst_i(icache_inst_w)
    ,.intr_i(intr_i)
    ,.reset_vector_i(reset_vector_i)
    ,.cpu_id_i(32'b0)
```

```
    // Outputs
    ,.mem_d_addr_o(dport_bridge_addr_w)
    ,.mem_d_data_wr_o(dport_bridge_data_wr_w)
    ,.mem_d_rd_o(dport_bridge_rd_w)
    ,.mem_d_wr_o(dport_bridge_wr_w)
    ,.mem_d_cacheable_o(dport_bridge_cacheable_w)
    ,.mem_d_req_tag_o(dport_bridge_req_tag_w)
    ,.mem_d_invalidate_o(dport_bridge_invalidate_w)
    ,.mem_d_flush_o(dport_bridge_flush_w)
    ,.mem_i_rd_o(icache_rd_w)
    ,.mem_i_flush_o(icache_flush_w)
    ,.mem_i_invalidate_o(icache_invalidate_w)
    ,.mem_i_pc_o(icache_pc_w)
);


icache u_icache
(
    // Inputs
     .clk_i(clk_i)
    ,.rst_i(rst_i)
    ,.req_rd_i(icache_rd_w)
    ,.req_flush_i(icache_flush_w)
    ,.req_invalidate_i(icache_invalidate_w)
    ,.req_pc_i(icache_pc_w)
    ,.axi_awready_i(axi_i_awready_i)
    ,.axi_wready_i(axi_i_wready_i)
    ,.axi_bvalid_i(axi_i_bvalid_i)
    ,.axi_bresp_i(axi_i_bresp_i)
    ,.axi_bid_i(axi_i_bid_i)
    ,.axi_arready_i(axi_i_arready_i)
    ,.axi_rvalid_i(axi_i_rvalid_i)
    ,.axi_rdata_i(axi_i_rdata_i)
    ,.axi_rresp_i(axi_i_rresp_i)
    ,.axi_rid_i(axi_i_rid_i)
    ,.axi_rlast_i(axi_i_rlast_i)

```

```
    // Outputs
    ,.req_accept_o(icache_accept_w)
    ,.req_valid_o(icache_valid_w)
    ,.req_error_o(icache_error_w)
    ,.req_inst_o(icache_inst_w)
    ,.axi_awvalid_o(axi_i_awvalid_o)
    ,.axi_awaddr_o(axi_i_awaddr_o)
    ,.axi_awid_o(axi_i_awid_o)
    ,.axi_awlen_o(axi_i_awlen_o)
    ,.axi_awburst_o(axi_i_awburst_o)
    ,.axi_wvalid_o(axi_i_wvalid_o)
    ,.axi_wdata_o(axi_i_wdata_o)
    ,.axi_wstrb_o(axi_i_wstrb_o)
    ,.axi_wlast_o(axi_i_wlast_o)
    ,.axi_bready_o(axi_i_bready_o)
    ,.axi_arvalid_o(axi_i_arvalid_o)
    ,.axi_araddr_o(axi_i_araddr_o)
    ,.axi_arid_o(axi_i_arid_o)
    ,.axi_arlen_o(axi_i_arlen_o)
    ,.axi_arburst_o(axi_i_arburst_o)
    ,.axi_rready_o(axi_i_rready_o)
);



endmodule
```

A brief summary of riscv_top module is as under:

RISCV_Top module integrates the following main components:

1. RISC-V Core
2. Instruction Cache
3. Data port bridge for memory-mapped AXI access

➢ riscv_core

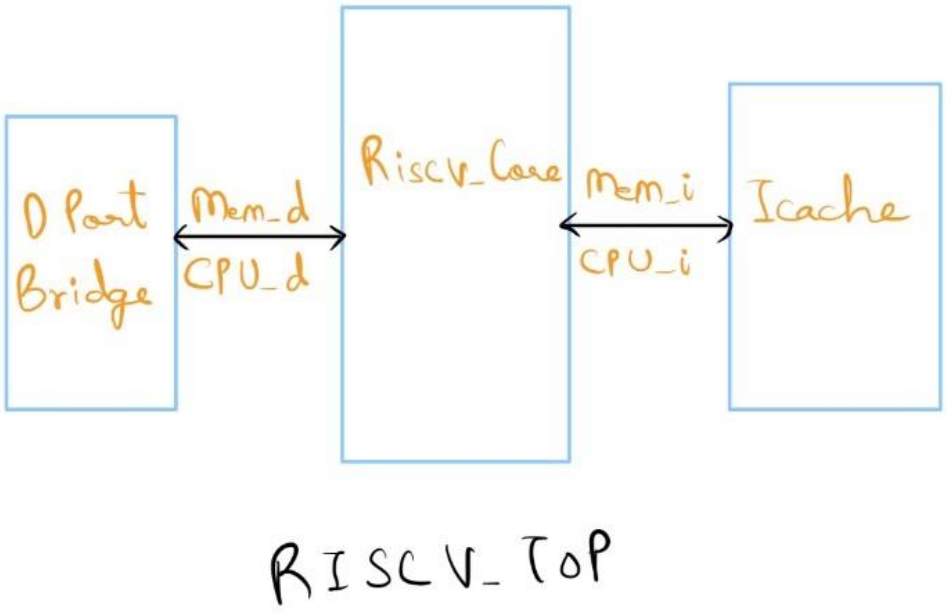
- It issues memory read and write requests for **data** via mem_d signals and read requests for **instructions** via mem_i signals.
- It handles interrupts and boots from the provided reset vector.
- The core module communicates with dport_bridge for data memory access and icache for instruction fetch.
- Modules integrated in riscv_core and their functions are as follows:

1. **riscv_exec**: Execution unit handles arithmetic, logical operations, and branch calculations.
2. **riscv_lsu**: Load-Store Unit manages memory operations and reports memory-related faults.
3. **riscv_csr**: Control and Status Register unit handles system instructions, interrupts, and exceptions.
4. **riscv_decode**: Instruction decoder unit interprets fetched instructions and controls dataflow between other units.
5. **riscv_fetch**: Instruction fetch unit retrieves instructions from memory and manages the program counter.

➢ icache

- It is situated between the riscv_core and the instruction AXI master interface, the AXI signals are connected to the AXI signals of AXI BRAM Controller which is a predefined IP in Xilinx Vivado to fetch instructions from Block Memory Generator.
- It accepts Program Counter and Fetch Request from the core.
- It outputs fetched instruction data to the core.
- It also handles cache flush and invalidate requests if the instruction is not fetched in one cycle.
- AXI transactions are generated to fetch instructions from Block Memory.

- **dport_bridge**
- It is acting as an interface between the core's data access requests and the AXI4 master interface.
- It handles read and write commands from the core and performs AXI transactions accordingly.
- It outputs responses, data, and error flags back to the core depending upon the data required.

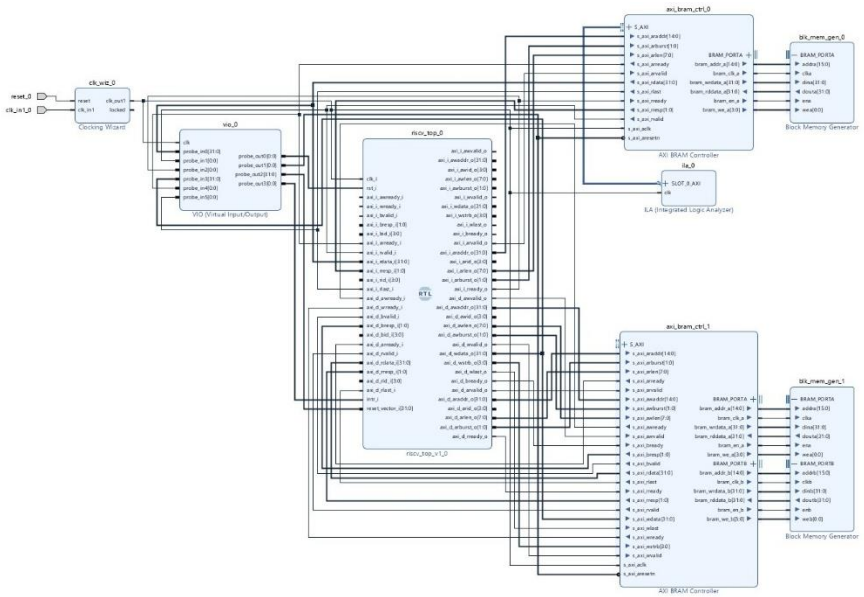


BLOCK DIAGRAM

In order to test the integrated top module, we have generated a Block Design instantiating the Integrated Module, Block Memory Generator, AXI BRAM Controller, VIO and Clocking Wizard.

Active High Reset of Top Module, Active Low Reset of AXI BRAM Controller, Interrupt Input and CPU ID are connected as outputs to VIO, while AXI Read signals of Icache are connected as inputs to it.

The Block Diagram of the implementation is shown as under:



COEFFICIENT FILE

In order to ensure that the integrated top module is getting instructions from Block Memory Generator in Icache which then passes them to Fetch block, which in turn passes them to Decode block which decodes the nature of operation and finally passes it to one of Execute, LSU Reg or CSR accordingly, we have instantiated the Block Memory with a Coefficient File which with annotations is shown as under:

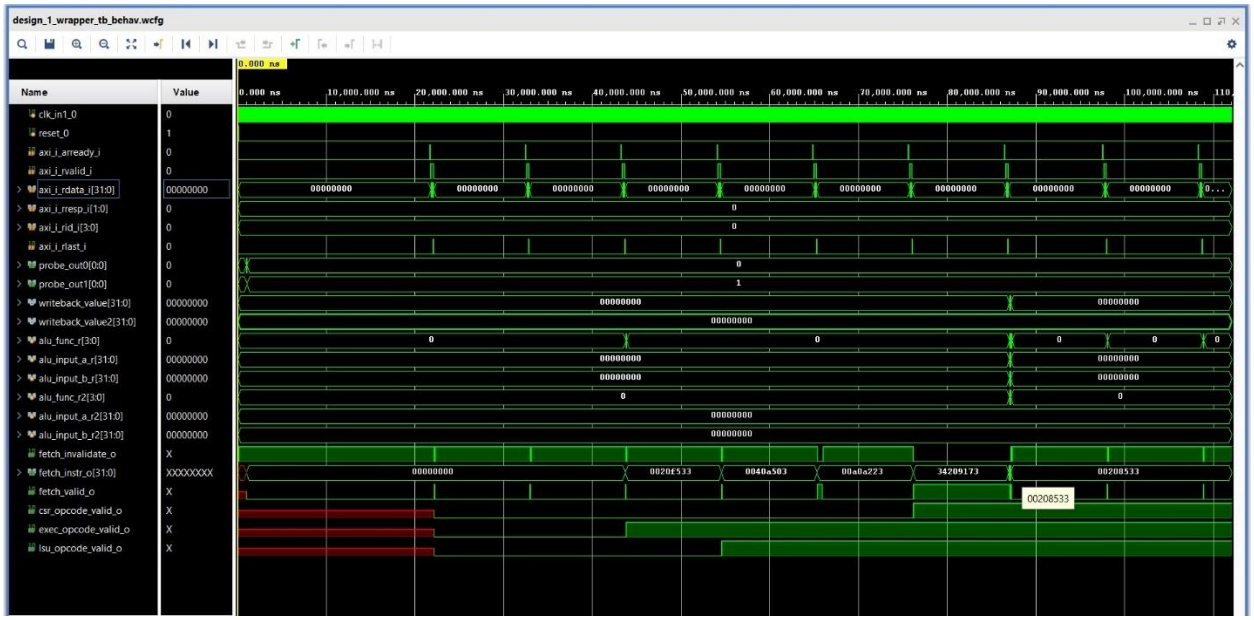
C:/Users/naman/Downloads/integmod/inst.coe

```
memory_initialization_radix=16;
memory_initialization_vector=

00000000,  // Initial Instruction saved as 0
0020F533,  // Instruction for Logical AND Operation of Content in Registers R1, R2
0040A503,  // Instruction for Loading R1 with content of memory location R4+10
00A0A223,  // Instruction for Storing content of R10 into memory address denoted by R1+4
34209173,  // Instruction for CSR Write
00208263,  // Instruction for Branch if contents of R1 and R2 are equal to an offset of 4
00208533;  //Instruction for Arithmetic Addition of Content in Registers R1, R2
```

BEHAVIORAL SIMULATION WAVEFORM

Based on the test cases written to test module's operation, the waveform of the Behavioral Simulation result is obtained as under:



POST SYNTHESIS TIMING SIMULATION WAVEFORM

Based on the test cases written to test module's operation, the waveform of the Post Synthesis Timing Simulation result is obtained as under:

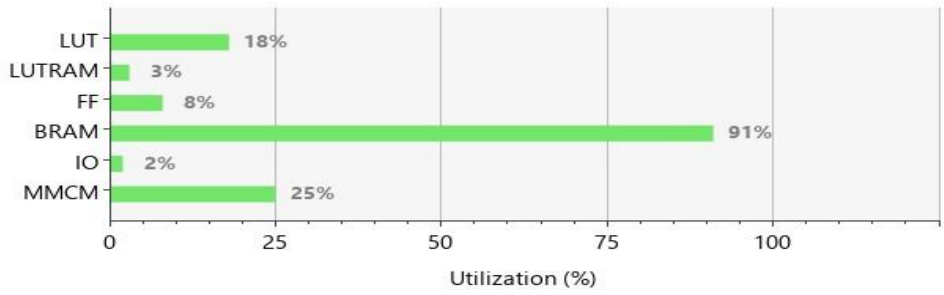
POST IMPLEMENTATION TIMING SIMULATION WAVEFORM

Based on the test cases written to test module's operation, the waveform of the Post Implementation Timing Simulation result is obtained as under:

RESOURCE UTILIZATION REPORT

After synthesizing and implementing the Verilog code, Resource Utilization Report is obtained as shown in the screenshot attached:

Resource	Utilization	Available	Utilization %
LUT	9715	53200	18.26
LUTRAM	475	17400	2.73
FF	8052	106400	7.57
BRAM	127	140	90.71
IO	2	125	1.60
MMCM	1	4	25.00



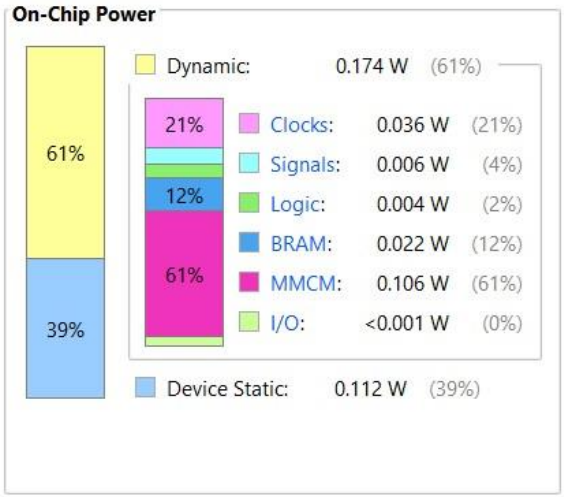
POWER UTILIZATION REPORT

After synthesizing and implementing the Verilog code, Power utilization report is obtained as shown in the screenshot attached:

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.286 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**28.3°C**
Thermal Margin:	56.7°C (4.7 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity



TIMING SUMMARY

After synthesizing and implementing the Verilog code, Design Timing Summary is obtained as shown in the screenshot attached:

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.328 ns	Worst Hold Slack (WHS):	0.015 ns	Worst Pulse Width Slack (WPWS):	3.000 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:	18928	Total Number of Endpoints:	18912	Total Number of Endpoints:	9021

All user specified timing constraints are met.

OBSERVATIONS

- The design uses a certain number of LUTs associated with RAM as inputs are fetched through them.
- The design uses **Virtual Input/Output (VIO)**, which is reducing the utilization of IO pins to just 2.
- Worst case negative slack for Setup and Hold conditions is positive which indicates all timing constraints are met satisfactorily.
- MMCM is Mixed Mode Clock Manager which is implemented through **Clocking Wizard** to synchronize the functioning of all modules like Integrated Top Module, AXI BRAM Controller and VIO.

SOC MODULE

The code of SOC module is over 700 lines, hence it is skipped from report and attached in the zip folder.

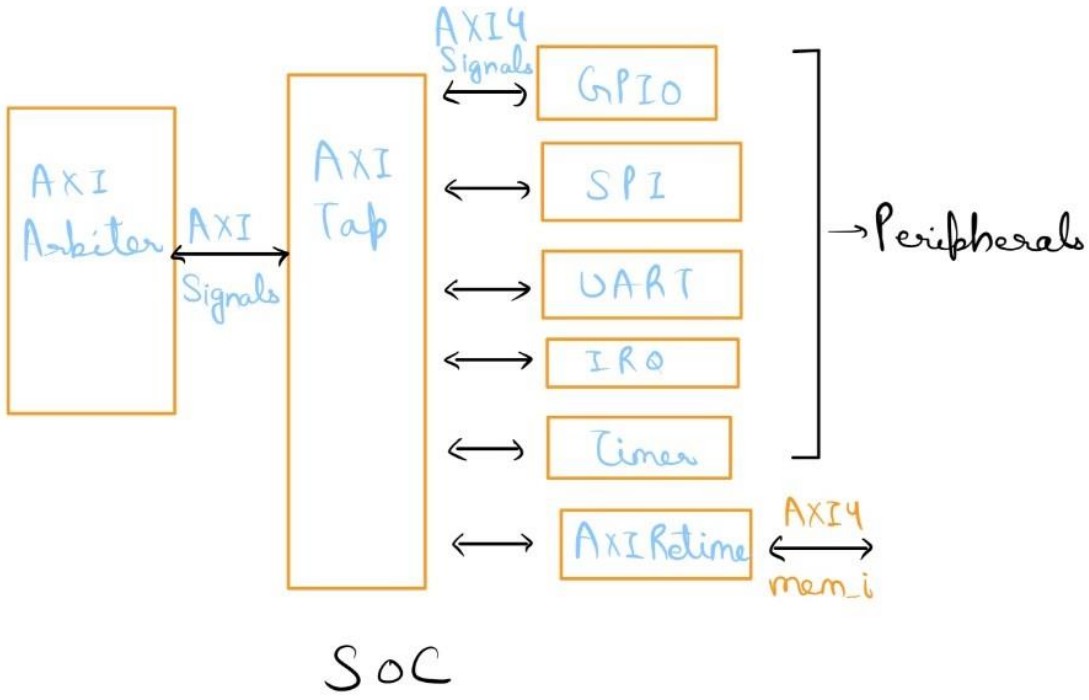
A brief summary of SOC Module is as under:

- SoC integrates several peripheral components interconnected through an AXI bus architecture. It includes:

1. An interrupt controller for managing system interrupts
2. A UART (Universal Asynchronous Receiver/Transmitter) for serial communication
3. A Timer module for time-based operations
4. A SPI (Serial Peripheral Interface) controller for interfacing with external SPI devices
5. A GPIO (General Purpose Input/Output) controller for digital I/O operations
6. AXI bus infrastructure components including:
    - An AXI arbiter to manage multiple bus masters
    - An AXI tap for routing transactions to peripherals
    - An AXI retiming module for synchronization

- The module has extensive connectivity with multiple input and output ports for the AXI bus interfaces, as well as dedicated signal lines for the peripheral interfaces (SPI, UART, GPIO).

- The design supports multiple bus masters including a CPU with separate instruction and data interfaces.



TESTBENCH OF SOC MODULE

The declarations in testbench of SOC Module have been skipped and only the test cases have been shown in the screenshots attached below:

```
246
247     // AXI Write Task
248     task axi_write;
249         input [31:0] addr;
250         input [31:0] data;
251         input [1:0]  master_sel; // 0: inport, 1: cpu_i, 2: cpu_d
252         begin
253             case (master_sel)
254                 0: begin // inport
255                     @(posedge clk_i);
256                     inport_awvalid_i = 1;
257                     inport_awaddr_i = addr;
258                     inport_awid_i = 4'b0000;
259                     inport_awlen_i = 8'h00;
260                     inport_awburst_i = 2'b01;
261                     inport_wvalid_i = 1;
262                     inport_wdata_i = data;
263                     inport_wstrb_i = 4'b1111;
264                     inport_wlast_i = 1;
265                     @(posedge clk_i);
266                     wait(inport_awready_o && inport_wready_o);
267                     @(posedge clk_i);
268                     inport_awvalid_i = 0;
269                     inport_wvalid_i = 0;
270                     inport_wlast_i = 0;
271                     inport_bready_i = 1;
272                     wait(inport_bvalid_o);
273                     @(posedge clk_i);
274                     inport_bready_i = 0;
275                 end
```

```
                1: begin // cpu_i
                    @(posedge clk_i);
                    cpu_i_awvalid_i = 1;
                    cpu_i_awaddr_i = addr;
                    cpu_i_awid_i = 4'b1000;
                    cpu_i_awlen_i = 8'h00;
                    cpu_i_awburst_i = 2'b01;
                    cpu_i_wvalid_i = 1;
                    cpu_i_wdata_i = data;
                    cpu_i_wstrb_i = 4'b1111;
                    cpu_i_wlast_i = 1;
                    @(posedge clk_i);
                    wait(cpu_i_awready_o && cpu_i_wready_o);
                    @(posedge clk_i);
                    cpu_i_awvalid_i = 0;
                    cpu_i_wvalid_i = 0;
                    cpu_i_wlast_i = 0;
                    cpu_i_bready_i = 1;
                    wait(cpu_i_bvalid_o);
                    @(posedge clk_i);
                    cpu_i_bready_i = 0;
                end
```

```
                2: begin // cpu_d
                    @(posedge clk_i);
                    cpu_d_awvalid_i = 1;
                    cpu_d_awaddr_i = addr;
                    cpu_d_awid_i = 4'b0100;
                    cpu_d_awlen_i = 8'h00;
                    cpu_d_awburst_i = 2'b01;
                    cpu_d_wvalid_i = 1;
                    cpu_d_wdata_i = data;
                    cpu_d_wstrb_i = 4'b1111;
                    cpu_d_wlast_i = 1;
                    @(posedge clk_i);
                    wait(cpu_d_awready_o && cpu_d_wready_o);
                    @(posedge clk_i);
                    cpu_d_awvalid_i = 0;
                    cpu_d_wvalid_i = 0;
                    cpu_d_wlast_i = 0;
                    cpu_d_bready_i = 1;
                    wait(cpu_d_bvalid_o);
                    @(posedge clk_i);
                    cpu_d_bready_i = 0;
                end
            endcase
        end
    endtask

```

```
    task send_uart_byte(input [7:0] data);
        integer i;
        begin
            #8000 uart_txd_i = 0; // Start bit
            for (i = 0; i < 8; i = i + 1) begin
                #8000 uart_txd_i = data[i];
            end
            #8000 uart_txd_i = 1; // Stop bit
        end
    endtask
```

```
335         // AXI Read Task
336         task axi_read;
337             input [31:0] addr;
338             input [1:0]  master_sel; // 0: inport, 1: cpu_i, 2: cpu_d
339             begin
340                 case (master_sel)
341                     0: begin // inport
342                         @(posedge clk_i);
343                         inport_arvalid_i = 1;
344                         inport_araddr_i = addr;
345                         inport_arid_i = 4'b0000;
346                         inport_arlen_i = 8'h00;
347                         inport_arburst_i = 2'b01;
348                         @(posedge clk_i);
349                         wait(inport_arready_o);
350                         @(posedge clk_i);
351                         inport_arvalid_i = 0;
352                         inport_rready_i = 1;
353                         wait(inport_rvalid_o);
354                         @(posedge clk_i);
355                         inport_rready_i = 0;
356                     end
357                     1: begin // cpu_i
358                         @(posedge clk_i);
359                         cpu_i_arvalid_i = 1;
360                         cpu_i_araddr_i = addr;
361                         cpu_i_arid_i = 4'b1010;
362                         cpu_i_arlen_i = 8'h00;
363                         cpu_i_arburst_i = 2'b01;
364                         @(posedge clk_i);
365                         wait(cpu_i_arready_o);
366                         @(posedge clk_i);
367                         cpu_i_arvalid_i = 0;
368                         cpu_i_rready_i = 1;
369                         wait(cpu_i_rvalid_o);
370                         @(posedge clk_i);
371                         cpu_i_rready_i = 0;
```

```
373                 2: begin // cpu_d
374                     @(posedge clk_i);
375                     cpu_d_arvalid_i = 1;
376                     cpu_d_araddr_i = addr;
377                     cpu_d_arid_i = 4'b0110;
378                     cpu_d_arlen_i = 8'h00;
379                     cpu_d_arburst_i = 2'b01;
380                     @(posedge clk_i);
381                     wait(cpu_d_arready_o);
382                     @(posedge clk_i);
383                     cpu_d_arvalid_i = 0;
384                     cpu_d_rready_i = 1;
385                     wait(cpu_d_rvalid_o);
386                     @(posedge clk_i);
387                     cpu_d_rready_i = 0;
388                 end
389             endcase
390         end
391     endtask
```

```
395
396         //----------GPIO Testcases-------------------------
397         gpio_input_i = 32'h900000FF;
398
399         // Read from GPIO
400         #20;
401         $display("reached 1");
402         axi_read(32'h94000004, 1);
403         #20;
404         // Write to GPIO
405         $display("reached 2");
406         axi_write(32'h94000008, 32'h0000AAAA, 1);
407         #20;
408         // Read back from GPIO
409         $display("reached 3");
410         axi_read(32'h94000008, 1);
411         #20;
412         // Write to GPIO
413         $display("reached 4");
414         axi_write(32'h94000018, 32'h000A0A0A, 2);
415         #20;
416         // Read back from GPIO
417         $display("reached 5");
418         axi_read(32'h94000018, 2);
419         #20;
420         // Write to GPIO
421         $display("reached 6");
422         axi_write(32'h94000000, 32'h0000AAAA, 1);
423         #20;
424         // Write to GPIO
425         $display("reached 7");
426         axi_write(32'h94000008, 32'h0000AAAA, 0);
427         $display("reached 8");
428         $display("Finished the GPIO Testcases");
429         #20;
430
```

```
430
431         //-------------SPI Test Cases----------------------------
432         #20;
433         $display("reached 9");
434         axi_write(32'h930000C0, 32'h000000C6, 1);
435         #20;
436         $display("reached 10");
437         axi_write(32'h93000068, 32'h000000AA, 0);
438         #1800;
439         $display("reached 11");
440         axi_read(32'h9300006C, 1);
441         $display("reached 12");
442         $display("SPI TestCases Completed");
443         #100;
444
445         //-----------------Timer Test Cases-------------------------
446         $display("reached 13");
447         axi_write(32'h9100000C, 32'hFFFFFFF0, 0); // Timer0 Count
448         #20;
449
450         $display("reached 14");
451         axi_write(32'h91000008, 32'h00000004, 2); // Timer0 Enable
452         #20;
453
454         $display("reached 15");
455         axi_read(32'h91000008, 1); // Timer1 Count
456         #20;
457
458         $display("reached 16");
459         axi_write(32'h91000014, 32'h00000004, 0); // Timer1 Enable
460         #100; // Wait for timers to run
461
462         $display("reached 17");
463         axi_read(32'h9100000C, 0);
464         #20;
465
466         $display("reached 18");
467         axi_read(32'h91000008, 0);
468         #20;
469
470         $display("reached 19");
```

```
445         //-----------------Timer Test Cases-------------------------
446         $display("reached 13");
447         axi_write(32'h9100000C, 32'hFFFFFFF0, 0); // Timer0 Count
448         #20;
449
450         $display("reached 14");
451         axi_write(32'h91000008, 32'h00000004, 2); // Timer0 Enable
452         #20;
453
454         $display("reached 15");
455         axi_read(32'h91000008, 1); // Timer1 Count
456         #20;
457
458         $display("reached 16");
459         axi_write(32'h91000014, 32'h00000004, 0); // Timer1 Enable
460         #100; // Wait for timers to run
461
462         $display("reached 17");
463         axi_read(32'h9100000C, 0);
464         #20;
465
466         $display("reached 18");
467         axi_read(32'h91000008, 0);
468         #20;
469
470         $display("reached 19");
471         axi_read(32'h91000010, 0);
472         #20;
473
474         $display("reached 20");
475         axi_read(32'h91000014, 0);
476         #20;
477
478         $display("reached 21");
479         axi_read(32'h91000018, 1);
480         #20;
481
482         $display("reached 22");
483         axi_read(32'h9100001C, 2);
484         $display("Finished the Timer Testcases");
485
```

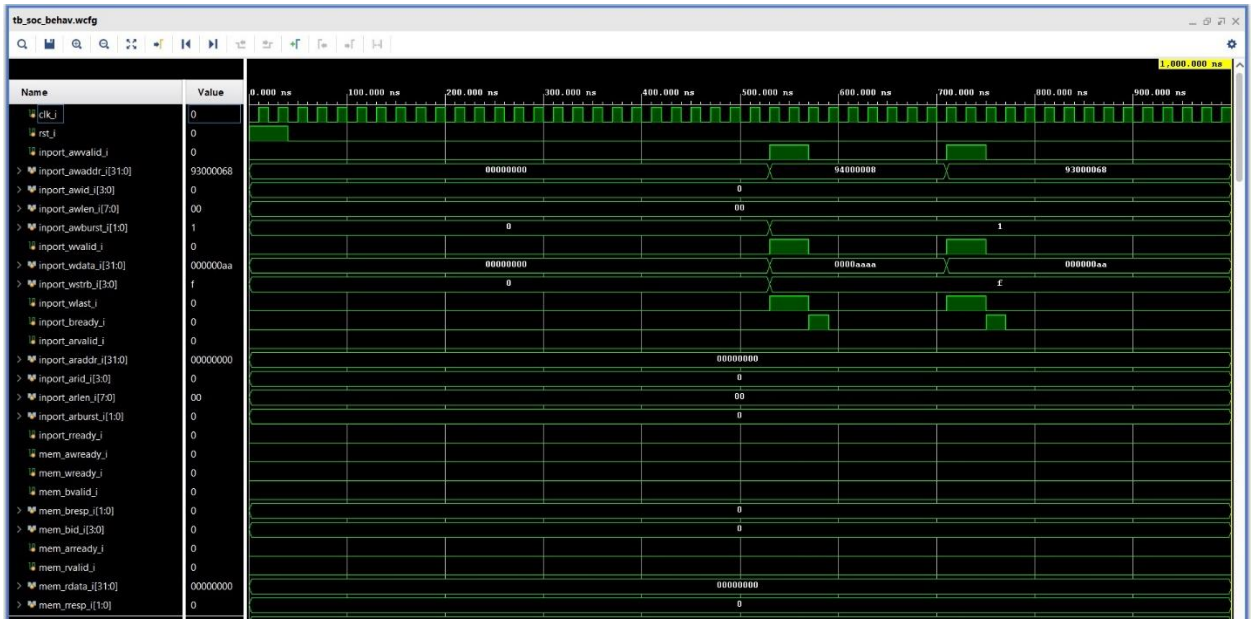
```
485
486         //-----UART Testcases---------------------------------------------
487         // 1. Enable UART Interrupt
488         $display("UART: Enabling Interrupt");
489         // Send UART byte (0xB5)
490         send_uart_byte(8'hB5); // inport
491         #40;
492
493         // 2. Write Data to TX Register
494         $display("UART: Writing 0xA5 to TX");
495         axi_write(32'h92000004, 32'h00000055, 1); // Transmit 'A' via inport
496         #40;
497
498         // 3. Read Status Register (Check TXEMPTY)
499         $display("UART: Reading Status (TXEMPTY)");
500         axi_write(32'h9200000C, 32'h00000016, 0); // inport
501         #40;
502
503         axi_read(32'h92000008, 0);
504         #40 axi_read(32'h92000000, 0);
505
506         // 4. Simulate UART Data Reception
507        $display("UART: Simulating RX Data (0x58)");
508         uart_rx_transmit(8'h58); // Transmit 'X' to uart_txd_i
509         #100;
510
511         // 5. Read RX Register
512         $display("UART: Reading RX Data");
513         axi_read(ULITE_RX, 0); // inport
514         #14000;
515
516         $display("Testbench completed");
517         $finish;
518     end
519
520 endmodule
```

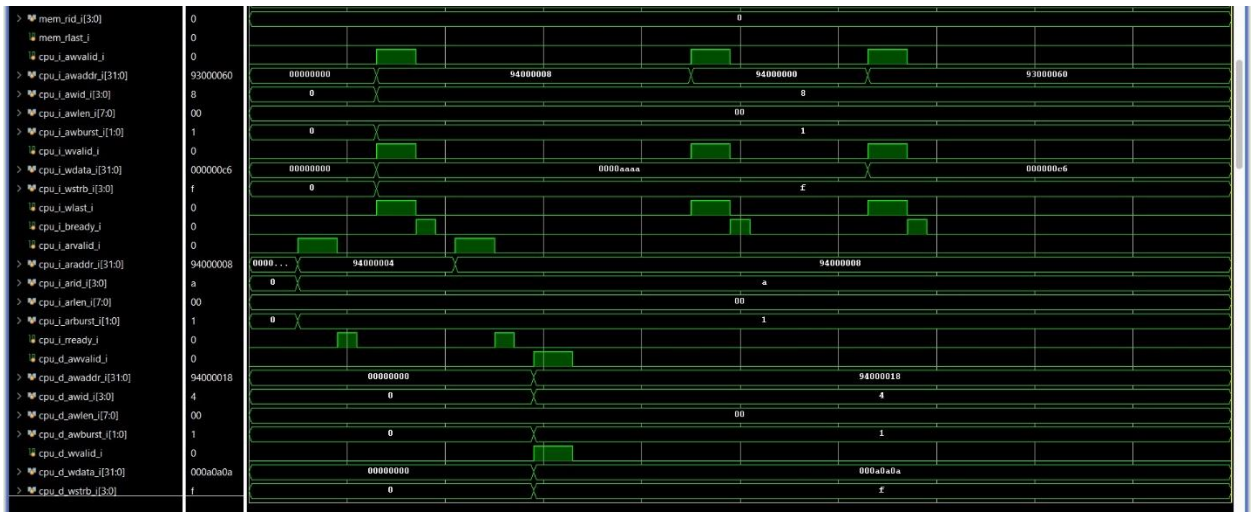
Implementation has not been done due to large number of ports in SOC Module, but we have checked it by the behavioral and post synthesis timing simulation waveform based on the test cases written.

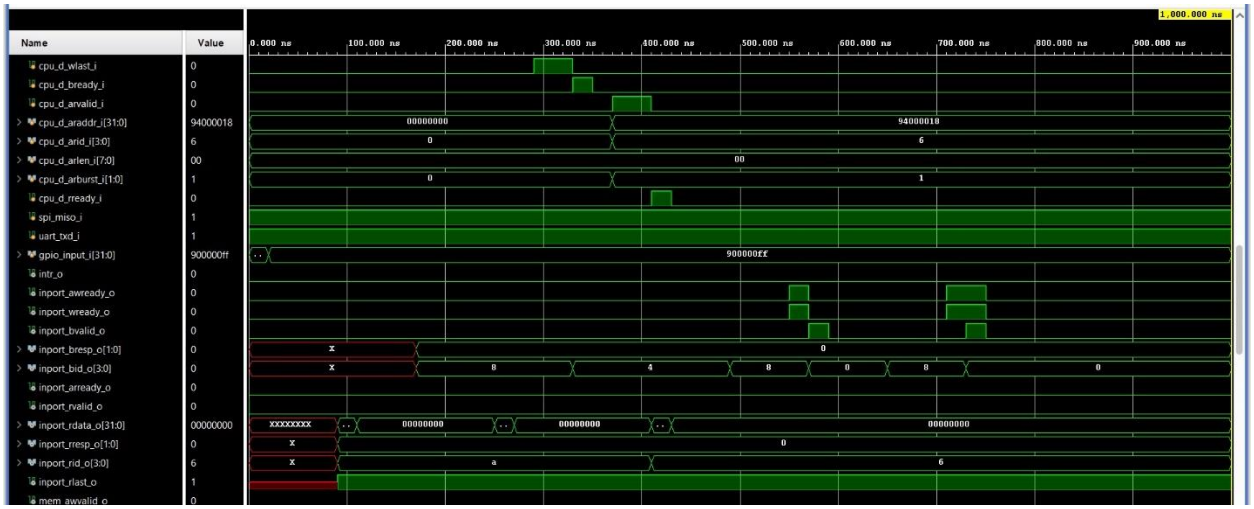
Simulation Waveform has large number of signals hence Behavioral Simulation waveform is skipped in the report, and only Post Synthesis Timing simulation waveform is shown below.

POST SYNTHESIS TIMING SIMULATION WAVEFORM

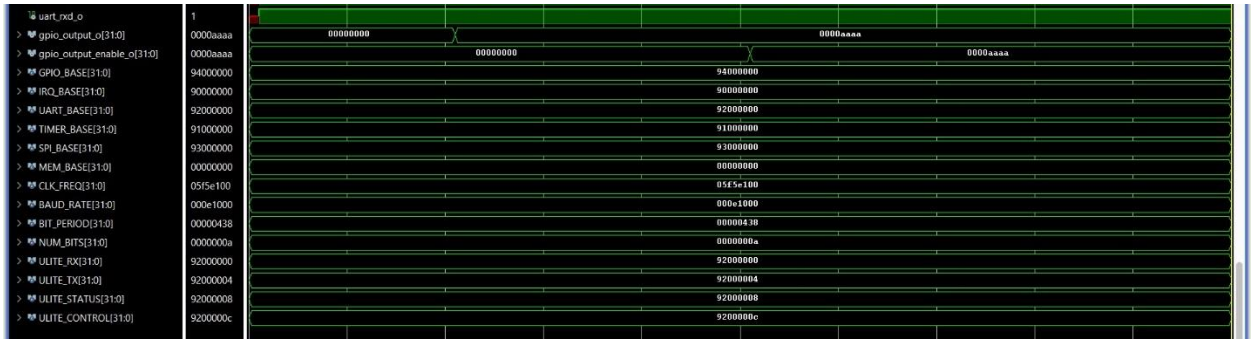
Based on the test cases written to test module's operation, the waveform of the Post Synthesis Timing Simulation result is obtained as under:






Name
Value
0.000 ns
100.000 ns
200.000 ns
300.000 ns
400.000 ns
500.000 ns
600.000 ns
700.000 ns
800.000 ns
900.000 ns
1,000.000 ns
cpu_d_wlast_i
cpu_d_bready_i
cpu_d_arvalid_i
cpu_d_araddr_i[31:0]
cpu_d_arid_i[3:0]
cpu_d_arlen_i[7:0]
cpu_d_arburst_i[1:0]
cpu_d_rready_i
spi_miso_i
uart_txd_i
gpio_input_i[31:0]
intr_o
inport_awready_o
inport_wready_o
inport_bvalid_o
inport_bresp_o[1:0]
inport_bid_o[3:0]
inport_arready_o
inport_rvalid_o
inport_rdata_o[31:0]
inport_rresp_o[1:0]
inport_rid_o[3:0]
inport_rlast_o
mem_awvalid_o


Name
Value
mem_rready_o
cpu_i_awready_o
cpu_i_wready_o
cpu_i_bvalid_o
cpu_i_bresp_o[1:0]
cpu_i_bid_o[3:0]
cpu_i_arready_o
cpu_i_rvalid_o
cpu_i_rdata_o[31:0]
cpu_i_rresp_o[1:0]
cpu_i_rid_o[3:0]
cpu_i_rlast_o
cpu_d_awready_o
cpu_d_wready_o
cpu_d_bvalid_o
cpu_d_bresp_o[1:0]
cpu_d_bid_o[3:0]
cpu_d_arready_o
cpu_d_rvalid_o
cpu_d_rdata_o[31:0]
cpu_d_rresp_o[1:0]
cpu_d_rid_o[3:0]
cpu_d_rlast_o
spi_clk_o
spi_mosi_o
spi_cs_o


uart_rxd_o
gpio_output_o[31:0]
gpio_output_enable_o[31:0]
GPIO_BASE[31:0]
IRQ_BASE[31:0]
UART_BASE[31:0]
TIMER_BASE[31:0]
SPI_BASE[31:0]
MEM_BASE[31:0]
CLK_FREQ[31:0]
BAUD_RATE[31:0]
BIT_PERIOD[31:0]
NUM_BITS[31:0]
ULITE_RX[31:0]
ULITE_TX[31:0]
ULITE_STATUS[31:0]
ULITE_CONTROL[31:0]

RISCV SOC MODULE

The Verilog code of RISCV SOC module has only 3 modules instantiated in it along with some intermediate signal declarations, hence it is skipped from report and attached in the zip folder.

A brief summary of SOC Module is as under:

- This module is a System-on-Chip (SoC) design built around a RISC-V processor core. It integrates the following key components:

1. RISC-V CPU Core
2. AXI4-Lite to AXI4 protocol converter
3. SoC peripherals and memory interface management

- The module features extensive connectivity through AXI4 bus interfaces, with separate instruction and data buses for the CPU. It includes peripherals like GPIO, SPI, UART.

- It uses a bus architecture to connect these components, with many wire declarations for the interconnections between the core, converter and peripherals. The architecture allows external memory connection through AXI4 memory interfaces and supports interrupt handling.

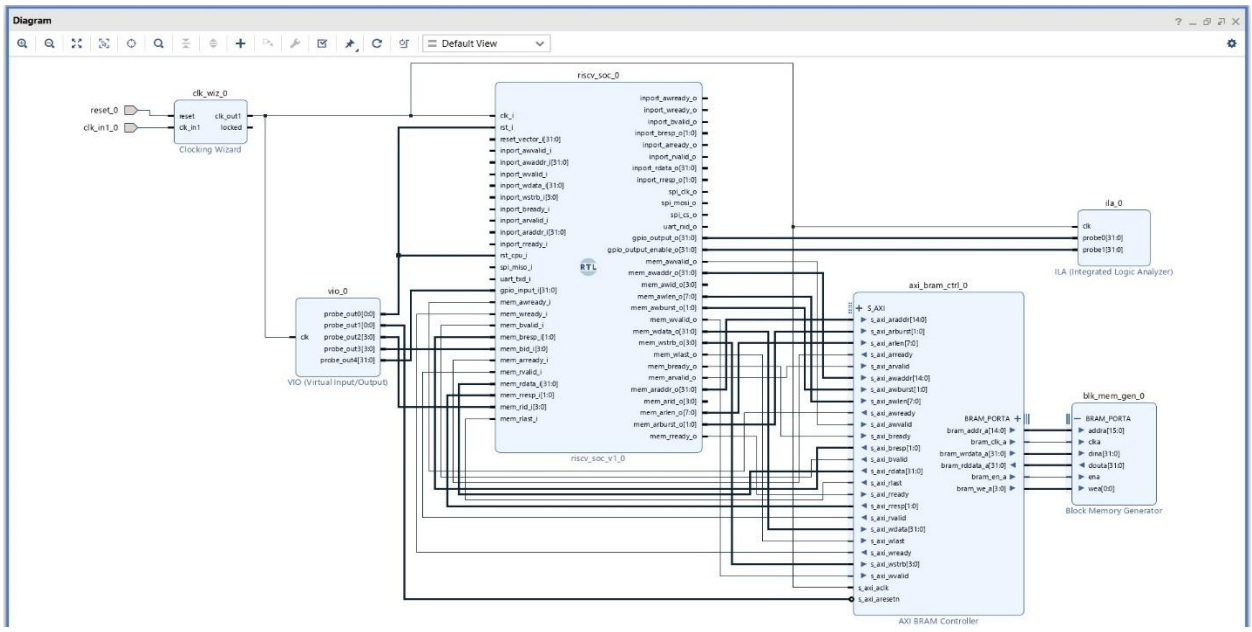
BLOCK DIAGRAM

In order to test the integrated riscv_soc module, we have generated a Block Design instantiating the Integrated Module, Block Memory Generator, AXI BRAM Controller, VIO, ILA and Clocking Wizard.

Active High Reset of Top Module, Active Low Reset of AXI BRAM Controller, GPIO Input, AXI Read ID of Icache and AXI Response ID of Icache are connected as outputs of VIO.

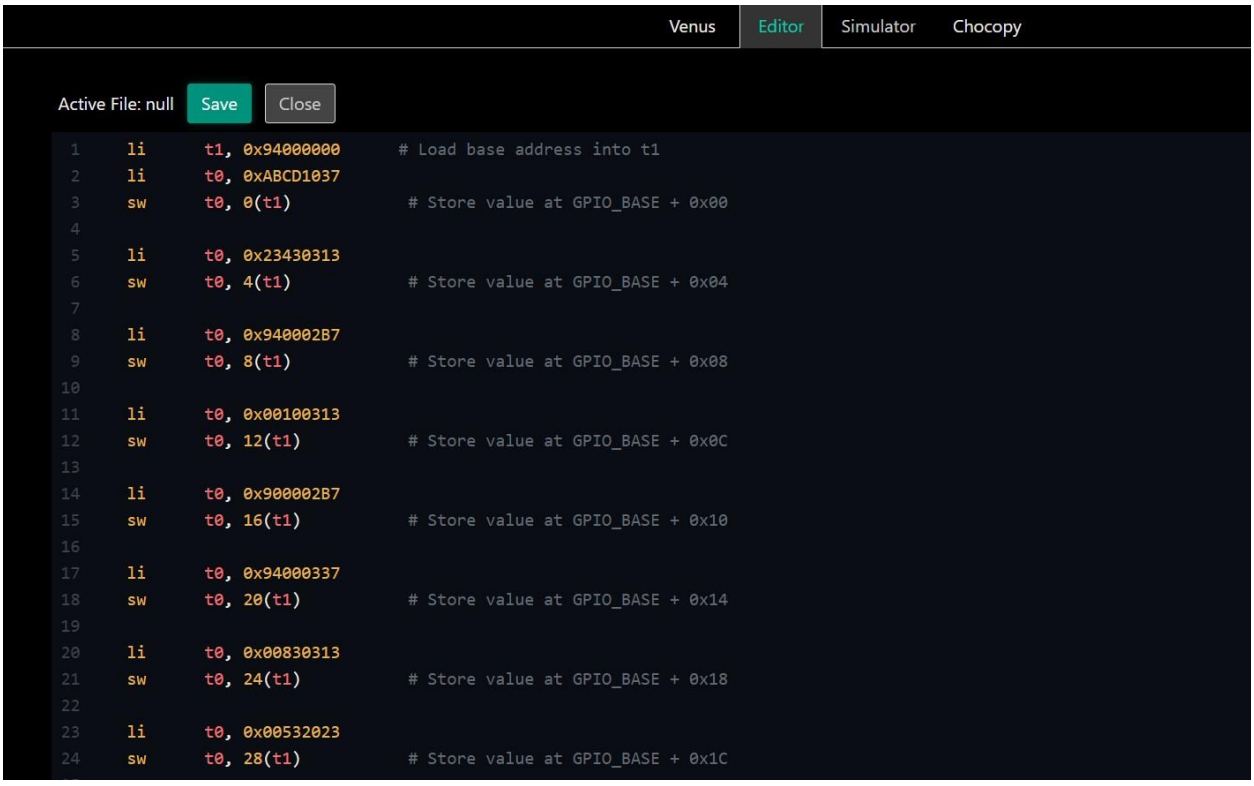
GPIO Outputs are connected to ILA so that they can be analyzed when connected to FPGA.

The Block Diagram of the implementation is shown as under:



MACHINE CODE

In order to test the integrated riscv_soc module with peripherals, a machine code compiled in Venus Compiler is written which is shown as under:




```
1    li      t1, 0x94000000      # Load base address into t1
2    li      t0, 0xABCD1037
3    sw      t0, 0(t1)           # Store value at GPIO_BASE + 0x00
4
5    li      t0, 0x23430313
6    sw      t0, 4(t1)           # Store value at GPIO_BASE + 0x04
7
8    li      t0, 0x940002B7
9    sw      t0, 8(t1)           # Store value at GPIO_BASE + 0x08
10
11   li      t0, 0x00100313
12   sw      t0, 12(t1)          # Store value at GPIO_BASE + 0x0C
13
14   li      t0, 0x900002B7
15   sw      t0, 16(t1)          # Store value at GPIO_BASE + 0x10
16
17   li      t0, 0x94000337
18   sw      t0, 20(t1)          # Store value at GPIO_BASE + 0x14
19
20   li      t0, 0x00830313
21   sw      t0, 24(t1)          # Store value at GPIO_BASE + 0x18
22
23   li      t0, 0x00532023
24   sw      t0, 28(t1)          # Store value at GPIO_BASE + 0x1C
25
26   li      t0, 0x0052A023
27   sw      t0, 32(t1)          # Store value at GPIO_BASE + 0x20
28
29   li      t0, 0xFF5FF06F
30   sw      t0, 36(t1)          # Store value at GPIO_BASE + 0x24
31
32   li      t0, 0x00000000
33   sw      t0, 40(t1)          # Store value at GPIO_BASE + 0x28
34
35   li      a7, 93
36   ecall                       # Exit the program
37
```

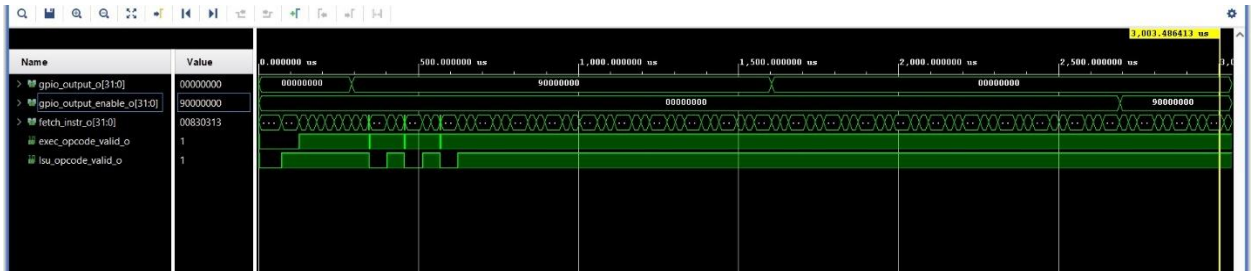
COEFFICIENT FILE

In order to ensure that the integrated soc module is getting instructions from Block Memory Generator in Icache which then passes them to Fetch block to finally get a value stored in provided GPIO Address provides, we have instantiated the Block Memory with a Coefficient File which is shown as under:

```
memory_initialization_radix=16;
memory_initialization_vector=
ABCD1037,
23430313,
940002B7,
00100313,
900002B7,
94000337,
00830313,
00532023,
0052A023,
FF5FF06F,
00000000;
```

BEHAVIORAL SIMULATION WAVEFORM

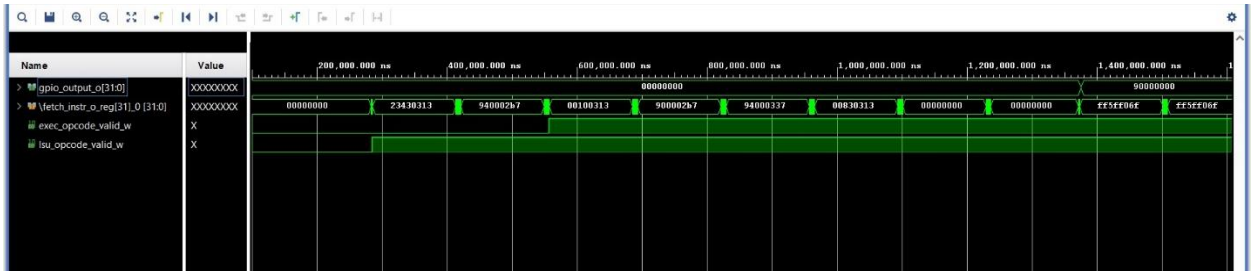
Based on the test cases written to test module's operation, the waveform of the Behavioral Simulation result is obtained as under:



Zoomed in waveform to see fetched instructions more clearly is shown as under:

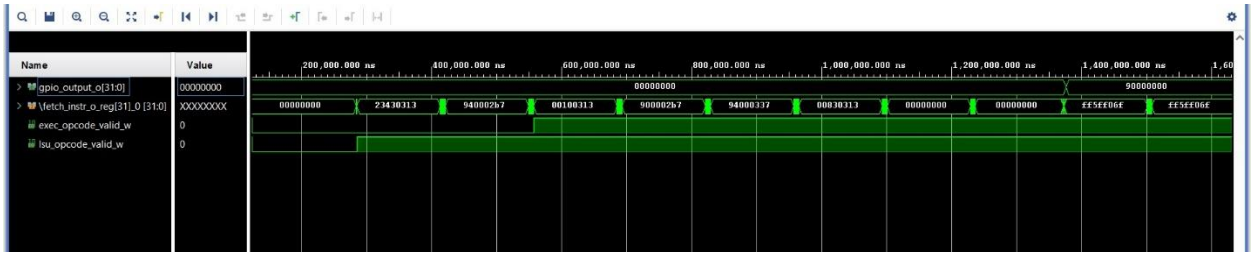
POST SYNTHESIS TIMING SIMULATION WAVEFORM

Based on the test cases written to test module's operation, the waveform of the Post Synthesis Timing Simulation result is obtained as under:



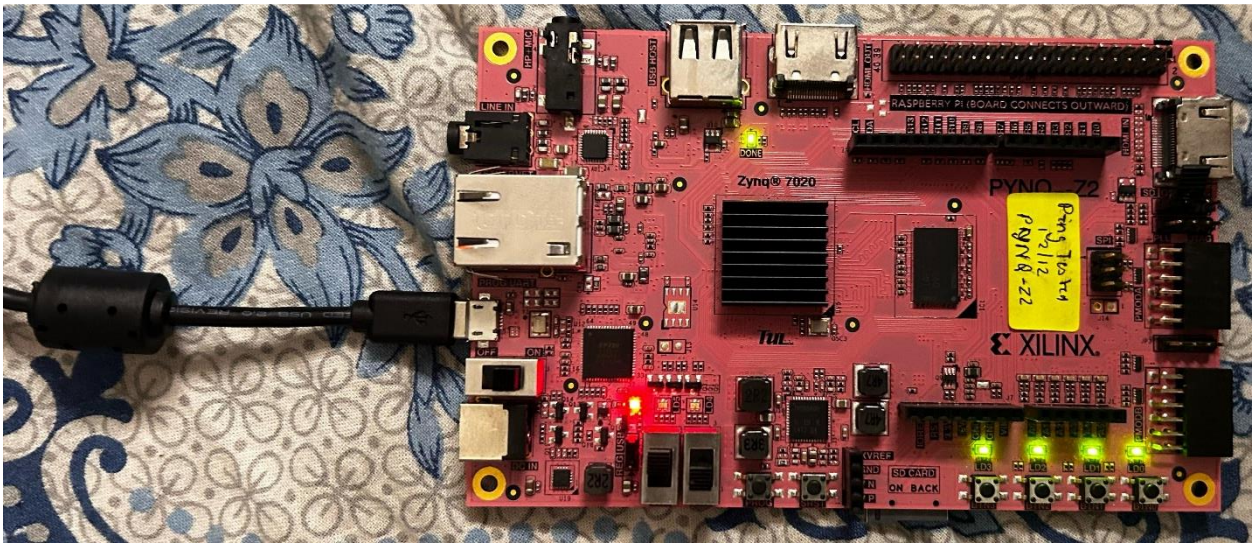
POST IMPLEMENTATION TIMING SIMULATION WAVEFORM

Based on the test cases written to test module's operation, the waveform of the Post Implementation Timing Simulation result is obtained as under:



FPGA IMPLEMENTATION

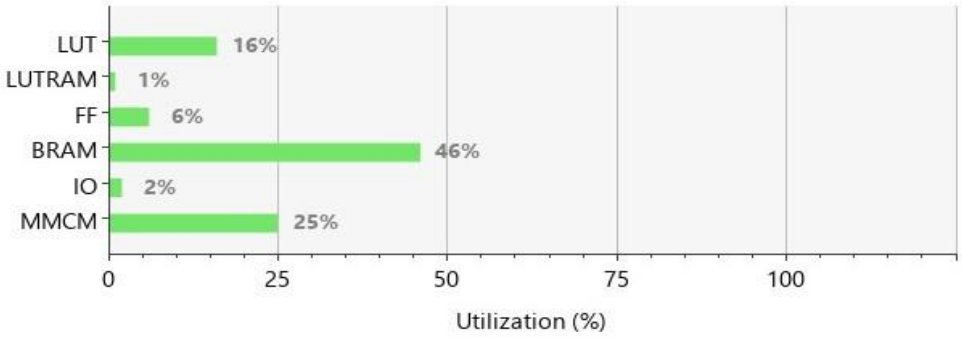
Constraint File is modified to map the port address of LEDs associated with GPIO on PYNQ Z2 Board, the glowing LEDs snapshot is shown below:



RESOURCE UTILIZATION REPORT

After synthesizing and implementing the Verilog code, Resource Utilization Report is obtained as shown in the screenshot attached:

Resource	Utilization	Available	Utilization %
LUT	8543	53200	16.06
LUTRAM	161	17400	0.93
FF	6244	106400	5.87
BRAM	65	140	46.43
IO	2	125	1.60
MMCM	1	4	25.00



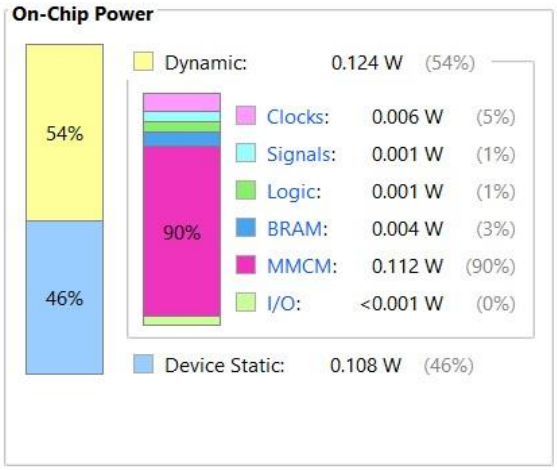
POWER UTILIZATION REPORT

After synthesizing and implementing the Verilog code, Power utilization report is obtained as shown in the screenshot attached:

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

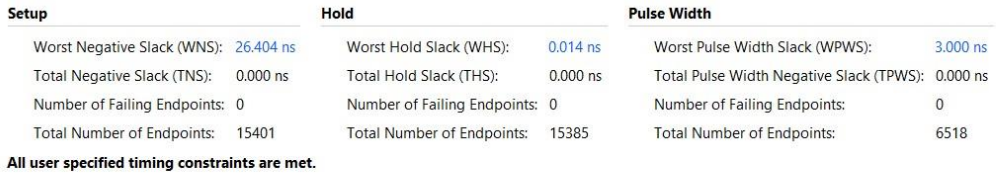
Total On-Chip Power:	**0.232 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**27.7°C**
Thermal Margin:	57.3°C (4.8 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity



TIMING SUMMARY

After synthesizing and implementing the Verilog code, Design Timing Summary is obtained as shown in the screenshot attached:



Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	26.404 ns	Worst Hold Slack (WHS):	0.014 ns	Worst Pulse Width Slack (WPWS):	3.000 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:	15401	Total Number of Endpoints:	15385	Total Number of Endpoints:	6518

All user specified timing constraints are met.

OBSERVATIONS

- The design uses a certain number of LUTs associated with RAM as inputs are fetched through them.
- The design uses **Virtual Input/Output (VIO)**, which are reducing the utilization of IO pins to just 2.
- Worst case negative slack for Setup and Hold conditions is positive which indicates all timing constraints are met satisfactorily.
- MMCM is Mixed Mode Clock Manager which is implemented through **Clocking Wizard** to synchronize the functioning of all modules like Integrated Top Module, AXI BRAM Controller and VIO.

RATNESH UPADHYAY 24M1119

NAMAN AGARWAL 24M1121

SAHIL ASHISH MEHTA 24M1124

SAURAV RAJ 21D070064

MRUDUL JAMBHULKAR 21D070044

PRAYAG MOHANTY 24M1177