UNIVERSITY OF TEXAS AT DALLAS

800 W Campbell Rd, Richardson, TX 75080, USA

EEDG 6370

DESIGN AND ANALYSIS OF
RECONFIGURABLE COMPUTING SYSTEMS
SPRING 2022

DESIGN OF 16-BIT CUSTOMIZABLE MICROPROCESSOR

SUBMITTED BY GROUP: MRS

Team Members:

Gokul Sai Raghunath (GXR200021)

Abhishek Mahesh Kumar (AXM200255)
Aishwarya Surapuram (AXS210196)

ACKNOWLEDGEMENT

We are grateful to Professor Dinesh Bhatia for providing us an opportunity to explore and conduct FPGA based projects.

We also take this opportunity to express our gratitude to Mark Sears for hisguidance in conducting the project.

TABLE OF CONTENT

ABSTRACT

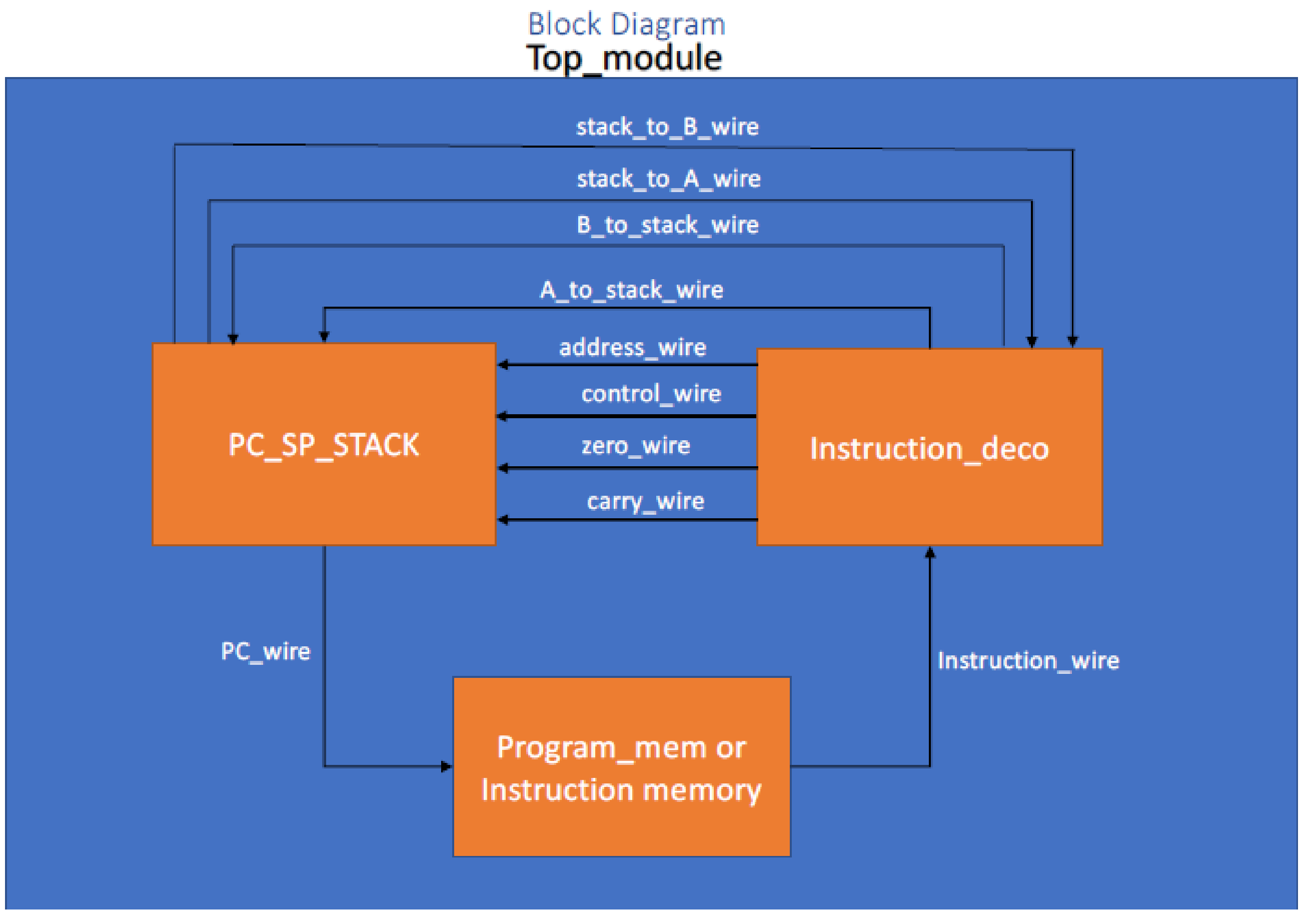
In this project we designed a simple 16-bit customizable microprocessor. The microprocessor can be considered the core for various user specific computing machines. It consists of a set of basic microprocessor features that can be used without any changes for some simple applications or can be extended by the user in many applications' specific directions.

In our MCU, A and B registers are two programmer-visible 16-bit working registers used to hold operands and outcomes of data manipulations. Memory-mapped input/output for communication with input and output devices is essentially a load/store microprocessor architecture with a simple instruction cycle consisting of four machine cycles per instruction; all data transformations are handled in working registers. Support for direct and basic stack addressing modes, as well as implicit addressing mode defining custom instructions and functional blocks that execute custom instructions, can be introduced and implemented using an FPGA.

FEATURES OF OUR MICROPROCESSOR:

- A 16-bit data bus and a 12-bit address bus provide direct access to up to 4096 16-bit memory addresses, as well as 16-bit wide instructions.
- RISC-based architecture
- There are three sorts of instruction sets: implicit (8-bit opcode), non-implicit (4-bit opcode), and user-defined (8-bit Opcode)
- Contains 2 flags. Namely, Zero Flag and Carry Flag
- 4095 stack locations with 16 bits of memory each
- Equipped with an interrupt that is enabled when an ISR is invoked.

BLOCK DIAGRAM



Our architecture is configured so that the instruction fetch cycle occurs at the **'Instruction Memory Block'** at t0 of the clock, the instruction decode cycle occurs at the **'Instruction Decode Block'** at t1 of the clock, and the instruction execution occurs at the **'PC SP STACK'** at t2 of the clock.

1. When a new instruction is fetched from an external memory location pointed to by the program counter, this is referred to as instruction fetch. It takes two machine cycles to complete. The first cycle, TO, is used to move the following instruction's address from the program counter to the address register. The second cycle, TI, is used to read the instruction from memory into the instruction register, IR. Simultaneously, the program counter is increased by one to the value that typically reflects the next instruction address.

2. Instruction decode is the recognition of the operation to be performed as well as the preparation of the effective memory address. This is done in the instruction cycle's third machine cycle, T2.

3. Instruction execution is the process of carrying out the actual operation defined by the operation code. This is accomplished at the fourth machine cycle T3 of the instruction cycle.

INSTRUCTION SET ARCHITECTURE

The program memory or the instruction memory will contain all the instructions to be tested on to our MCU. The instruction decoder decodes the instruction sent by the instructor to the program memory. Depending on the instruction set should be selected, the instruction decoder is sent to the next block for execution.

- ### NON-IMPLICIT INSTRUCTION SET

In this instruction set the first four bits will select the opcode or the operation to execute and the next 12 bits of the instruction denotes the address. This is explicitly used if the data has to be stored to memory, to be loaded from memory, if there is a jump, which is by specifying the address of the subroutine.

OPCODE(4)	ADDRESS(12)

Opcode[15…12]	Mnemonic [11:0]
0000	LDA
0001	LDB
0010	STA
0011	STB
0100	JMP
1000	JSR
1010	PUSHA
1100	POPA
1110	RET

LDA : A➔ M[address]
This instruction does the operation of loading the data memory with the value of the register A to the address specified in the instruction

LDB : **B ➔ M[address]**
This instruction does the operation of loading the data memory with the value of the register B to the address specified in the instruction

STA: **M[address] ➔ A**
This instruction does the operation of loading the register B with the value of the data memory from the address specified in the instruction

STB : **M[Address] ➔ B**
This instruction does the operation of loading the register A with the value of the data memory

from the address specified in the instruction

JMP: **PC ➔ Address**
When this instruction is encountered the Program counter is loaded with a new address of the program memory from which a subroutine is executed. (Note: when a JMP occurs the previous address location of the program memory is erased)

JSR: **Stack ➔ PC, PC ➔ address, SP ➔ SP-1**
When this instruction is encountered the Program counter is loaded with a new address of the program memory from which a subroutine is executed. The previous address location of the program memory is pushed to stack and when RET opcode is encountered the program counter is popped with the address after which the JSR takes place

PUSHA: **Stack ➔ A, SP ➔ SP-1**
This opcode is used to push the value to the stack from Register A

POPA: **SP ➔ SP+1, A ➔ stack**
This opcode is used to pop the value from stack to the Register A

RET: **SP ➔ SP+1, PC ➔ stack**
After the JSR instruction this opcode is to return from the subroutine to the main program

- **IMPLICIT INSTRUCTION SET**

This instruction set is selected when the first four most significant bits are '0111' and the opcode in this instruction is decided by the preceding four bits. They are called implicit as they do the operations implicitly and the instruction does not require for it to be addressable.

OPCODE(8)	NOT USED(8)

0111 0001	**ADD**
0111 0010	**AND**
0111 0011	**CLA**
0111 0100	**CLB**
0111 0101	**CMB**
0111 0110	**INCB**
0111 0111	**DECB**
0111 1000	**CLC**
0111 1001	**CLZ**
0111 1010	**ION**
0111 1011	**IOF**
0111 1100	**SC**
0111 1101	**SZ**

ADD: **A ➔ A+B**
This instruction does the arithmetic addition between A register and B register and stores the result in A register

AND: **A ➔ A AND B**
This instruction does bitwise AND operation between A register and B register and stores the value in A register

CLA: **A ➔ 0**
This instruction clears the data in A register

CLB: **B ➔ 0**
This instruction clears the data in B register

CMB: **B ➔ B'**
This instruction performs bitwise compliment of the value stored in B register

INCB: **B ➔ B+1**
This instruction increments the value in B register by 1.

DECB: **B ➔ B-1**
This instruction decrements the value in B register by 1.

CLC **Carry Flag ➔ 0**
This instruction clears the carry flag.

CLZ: **Zero Flag➔ 0**
This instruction clears the zero flag.

ION: **IEN ➔ 1, enable interrupt**
This instruction is for handling an interrupt and sets the interrupt flag to High, when there is an interrupt, the ISR (Interrupt Service Routine) is invoked and the subroutine pertaining to the interrupt is executed, upon completing the interrupt subroutine the Program counter is loaded with the address to after where the interrupt was handled.

IOF: **IEN ➔ 0, disable interrupt**
This instruction is for handling an interrupt and sets the interrupt flag to Low

SZ: **If Z=1, PC ➔ PC+1; skip if zero is set**
This instruction skips to next instruction in the program memory to the next one if the zero flag is set high.

SC: **If C=1, PC ➔ PC+1; skip if carry set**
This instruction skips to next instruction in the program memory to the next one if the carry flag is set high.

• USER DEFINED INSTRUCTION SET

This instruction set is selected when the first four most significant bits are ‘1111’ and the opcode in this instruction is decided by the preceding four bits. They are also implicit as they do the operations implicitly and the instruction does not require for it to be addressable.

OPCODE(8)	NOT USED(8)

Opcode	Instruction
1111 0000	POPB
1111 0001	CNT
1111 0010	MUL
1111 0011	NEGA

POPB: SP ➔ SP+1, B ➔ stack
This opcode is used to pop the value from stack to the Register B

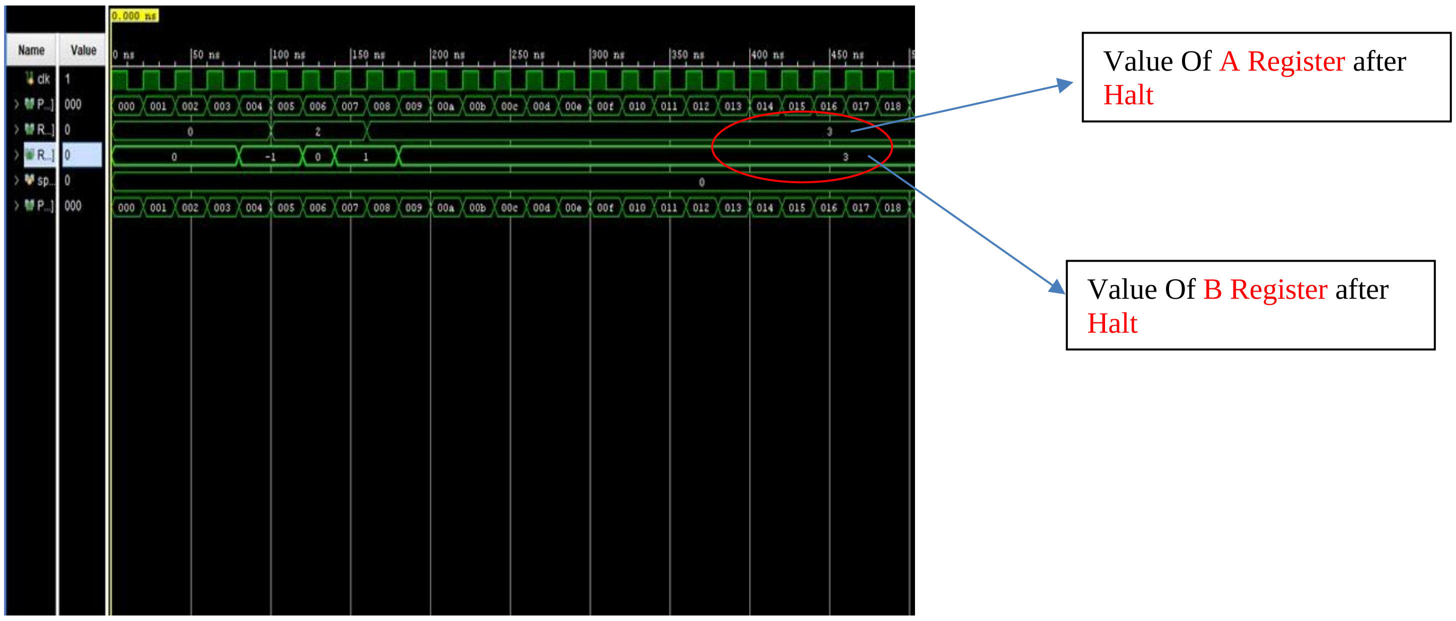
CNT: If Count = number of iterations to brake from loop; PC ➔ PC+1
This opcode is used to check the number of times it traverses through the loop and

MUL : A➔A*immediate value
This opcode is used to multiply an immediate value with A register and store it in A register

NEGA :A➔A’ +1
This opcode is used to take ones compliment of A register or multiply the A register value by -1

SOFTWARE SIMULATION

Test Case :1



Instruction for test

IOF *//interrupt signal is turned off*

CLB *//Clear Value of in Register B*

CLA *//Clear Value of in Register A*

LDB 12'h104 *// Load the value in 12'h104 location of memory to register B (16'hFFFF)*

LDA 12'h102 *//Load the value in 12'h102 location of memory to register A (16'h0002)*

CMB *//Compliment the value in B Register*

INCB *// Increment the value in B register by 1*

ADD *//Add A and B register values and store in A*

LDB 12'h103 *// Load the value in 12'h103 location of memory to register B*

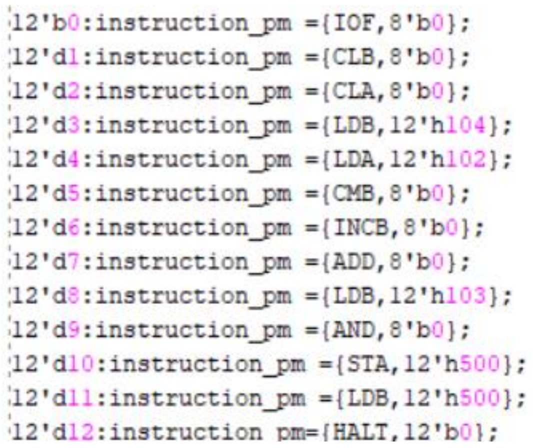
AND *//Add A and B register values and store in A*

STA 12'h500 *//load the value of A to memory in 12'h500 location*

LDB 12'h500 *// Load the value in 12'h500 location of memory to register B (16'h0003)*

HALT 12'b0 *//completely halts the operation by retaining to the same PC value*

Program for test case 1 in program memory

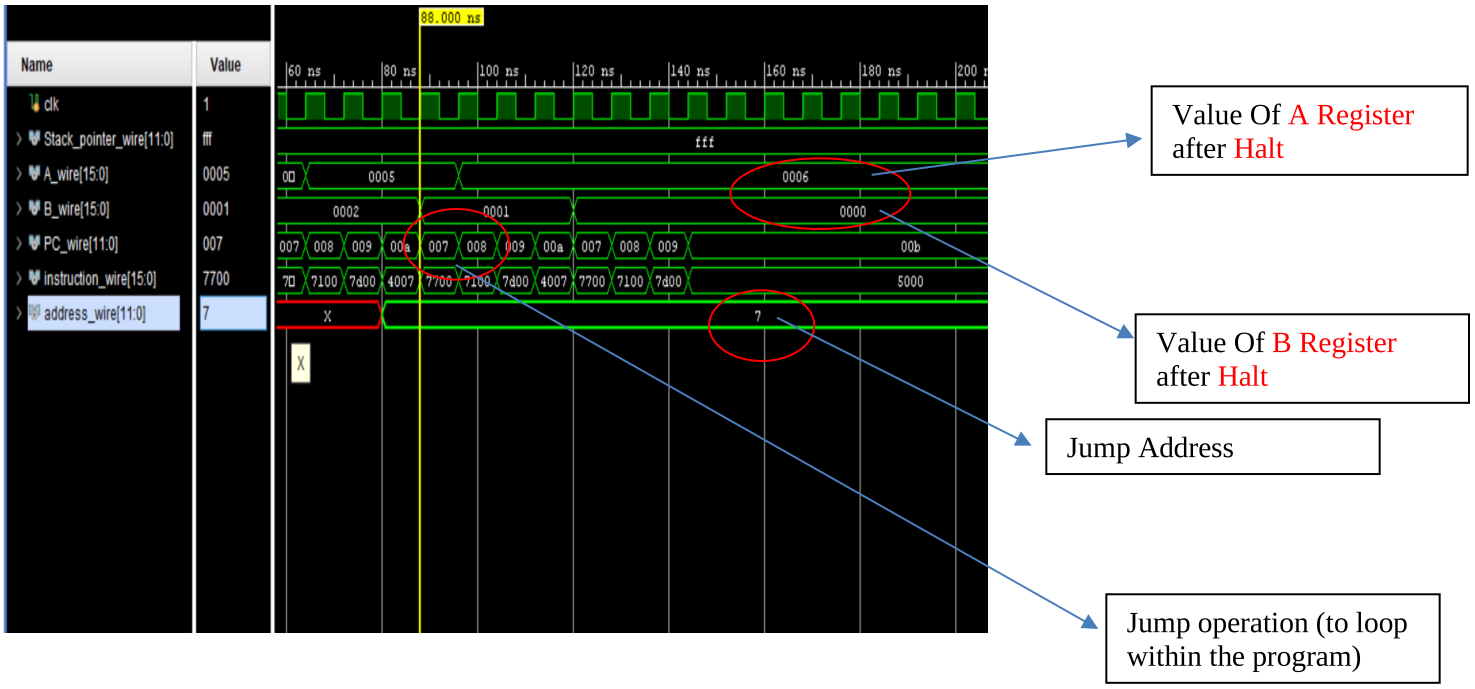


```
12'b0:instruction_pm ={IOF,8'b0};
12'd1:instruction_pm ={CLB,8'b0};
12'd2:instruction_pm ={CLA,8'b0};
12'd3:instruction_pm ={LDB,12'h104};
12'd4:instruction_pm ={LDA,12'h102};
12'd5:instruction_pm ={CMB,8'b0};
12'd6:instruction_pm ={INCB,8'b0};
12'd7:instruction_pm ={ADD,8'b0};
12'd8:instruction_pm ={LDB,12'h103};
12'd9:instruction_pm ={AND,8'b0};
12'd10:instruction_pm ={STA,12'h500};
12'd11:instruction_pm ={LDB,12'h500};
12'd12:instruction_pm={HALT,12'b0};
```

This instruction is to explicitly used to check loading and storing the data and also arithmetic addition. In this test case, the value of A and B register is the same.

A register = B register =16’d3

<u>Test Case :2</u>



<u>*Instruction for test*</u>

IOF *//Interrupt signal is turned off*

CLB *//Clear Value in Register B*

CLA *//Clear Value in Register A*

CLC *//Clear Value in carry flag "C"*

CLZ *//Clear Value in zero flag "Z"*

LDB *//Load Register B with the value present at location 12'h103*

ADD *// Adds values present in register A and B and store backs in A*

DECB *// Decrement the value present in register B by one value i.e., B=B-1*

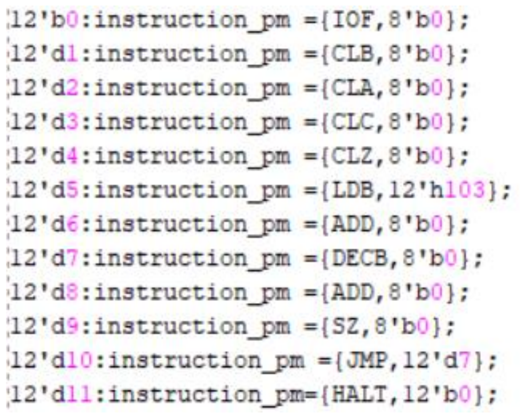
ADD *// Adds values present in register A and B and store backs in A*

SZ *//Skip if Zero i.e., if Z==1 skip next iteration by incrementing PC value.*

JMP 12'd7 //Jump from present PC value to location 7 by assigning it PC

HALT 12'b0 // Halt the simulation at this instruction by not incrementing PC

<u>Program for test case 2 in program memory</u>

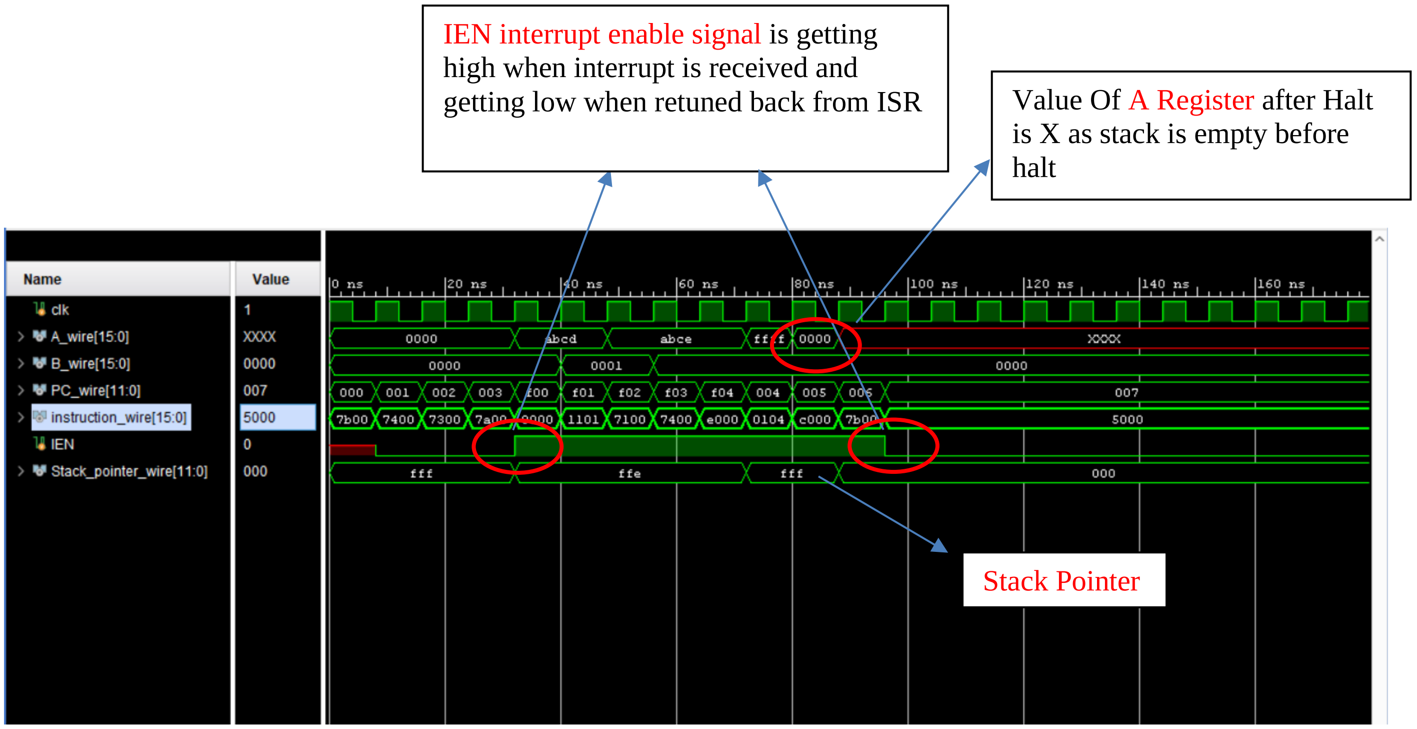


```
12'b0:instruction_pm ={IOF,8'b0};
12'd1:instruction_pm ={CLB,8'b0};
12'd2:instruction_pm ={CLA,8'b0};
12'd3:instruction_pm ={CLC,8'b0};
12'd4:instruction_pm ={CLZ,8'b0};
12'd5:instruction_pm ={LDB,12'h103};
12'd6:instruction_pm ={ADD,8'b0};
12'd7:instruction_pm ={DECB,8'b0};
12'd8:instruction_pm ={ADD,8'b0};
12'd9:instruction_pm ={SZ,8'b0};
12'd10:instruction_pm ={JMP,12'd7};
12'd11:instruction_pm={HALT,12'b0};
```

This instruction is to explicitly used to check the looping operation which done by JMP instruction

A register =16’d6

Test Case :3



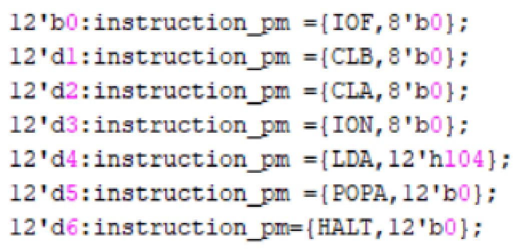
Instruction for test

IOF		*//Interrupt signal is turned off*
CLB		*//Clear Value in Register B*
CLA		*//Clear Value in Register A*
ION		*// Interrupt signal is turned ON PC i.e., Program counter jumps to ISR Interrupt subroutine location in our case PC jumps to 3840*
LDA	12'h104	*//Load Register A with the value present at location 12'h104*
POPA		*//Pop a value from Stack and assign it to register A*
HALT		*// Halt the simulation at this instruction by not incrementing PC*

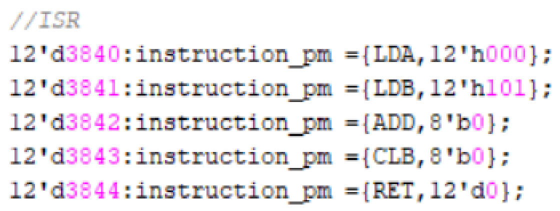
// ISR

LDA	12'h000	*//Load Register A with the value present at location 12'h000*
LDB	12'h101	*//Load Register B with the value present at location 12'h101*
ADD		*// Adds values present in register A and B and store backs in A*
CLB		*//Clear Value in Register B*
RET		*//Return back from ISR to the location where interrupt is generated by popping the address from stack and assigning it back to PC.*

<u>Program for test case 3 in program memory</u>



```
12'b0:instruction_pm ={IOF,8'b0};
12'd1:instruction_pm ={CLB,8'b0};
12'd2:instruction_pm ={CLA,8'b0};
12'd3:instruction_pm ={ION,8'b0};
12'd4:instruction_pm ={LDA,12'h104};
12'd5:instruction_pm ={POPA,12'b0};
12'd6:instruction_pm={HALT,12'b0};
```



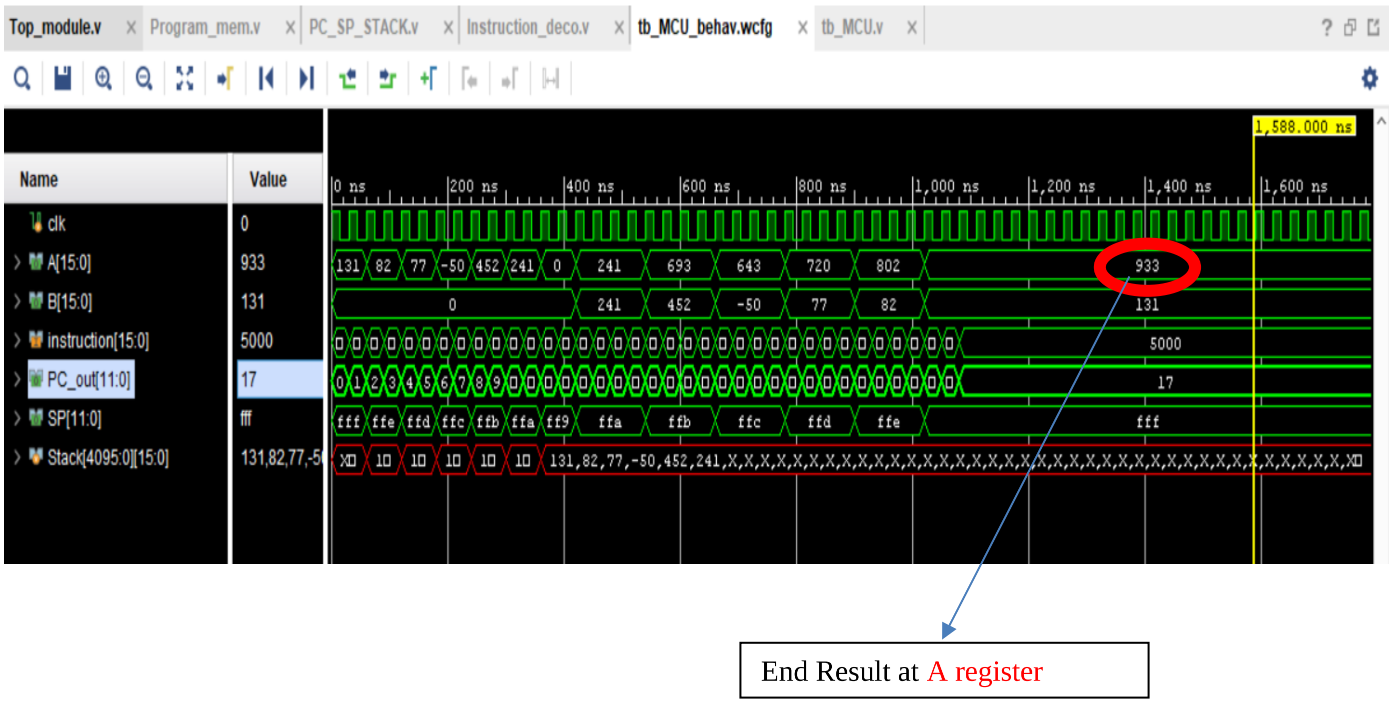
```
//ISR
12'd3840:instruction_pm ={LDA,12'h000};
12'd3841:instruction_pm ={LDB,12'h101};
12'd3842:instruction_pm ={ADD,8'b0};
12'd3843:instruction_pm ={CLB,8'b0};
12'd3844:instruction_pm ={RET,12'd0};
```

This instruction is to explicitly check the working of Interrupt Service Routine

For Validation of our Module we used two test cases which was to check functionality of user defined OPCODE

TEST-1 for checking functionality of CNT and POPB

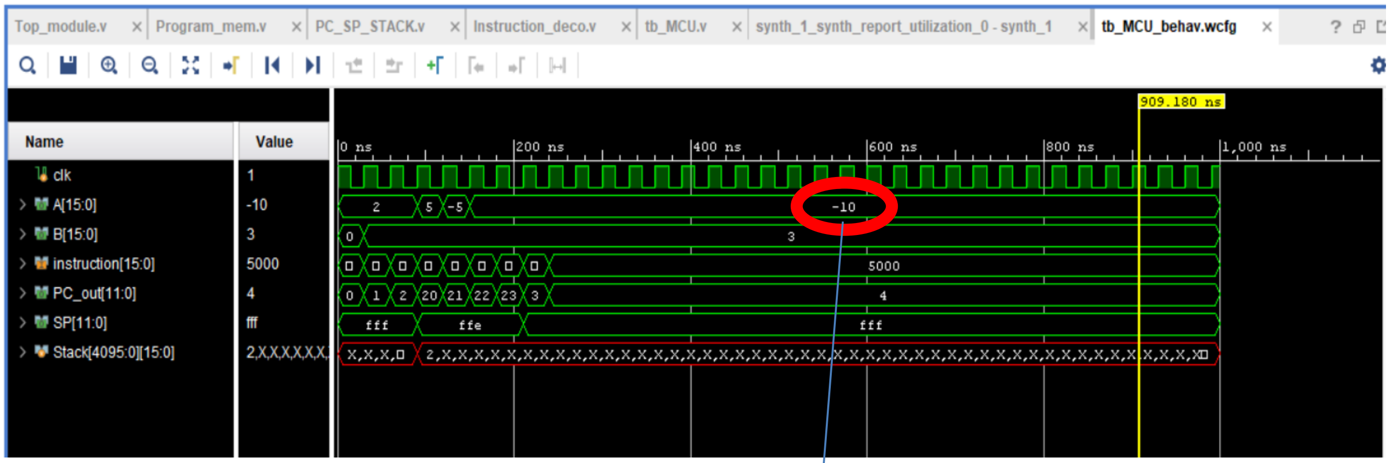
```
LDA      12'h000  //Load Register A with the value present at location 12'h000 i.e., 131
PUSHA 12'b0       //Push the value in A register on to Stack i.e.,131
LDA      12'h001  //Load Register A with the value present at location 12'h000 i.e.,  82
PUSHA 12'b0       //Push the value in A register on to Stack i.e.,82
LDA      12'h002  //Load Register A with the value present at location 12'h000 i.e.,  77
PUSHA 12'b0       //Push the value in A register on to Stack i.e.,77
LDA      12'h003  //Load Register A with the value present at location 12'h000 i.e., 12'hFFCE = -
                   50 (decimal)
PUSHA 12'b0       //Push the value in A register on to Stack i.e.,-50
LDA      12'h004  //Load Register A with the value present at location 12'h000 i.e.,  452
PUSHA 12'b0       //Push the value in A register on to Stack i.e.,452
LDA      12'h005  //Load Register A with the value present at location 12'h000 i.e.,  241
PUSHA 12'b0       //Push the value in A register on to Stack i.e.,241
CLA        8'b0   //Clear Value in Register A
POPB       8'b0   // Pop value from stack and assign to register B = 241
ADD       8'b0    // Adds values present in register A and B and store backs in A i.e., A+B=A=241
CNT       8'd6    // Increment count value for every jump iteration and compare the count value if
                   equals to "6" if true skip next iteration incrementing PC value.
JMP       13      //Jump from present PC value to location 13 by assigning it to PC.
HALT   12'b0      // Halt the simulation at this instruction by not incrementing PC
```



TEST-2 for checking functionality of NEG and MUL

LDA 12'h000 *//Load Register A with the value present at location 12'h000*
LDB 12'h001 *//Load Register B with the value present at location 12'h001*
JSR 20 *// PC i.e., Program counter jumps to ISR Interrupt subroutine location in our case PC jumps to 20*
STA 5 *// Store the value present in Register A in data memory location 5*
HALT 12'b0 *// Halt the simulation at this instruction by not incrementing PC*

//subroutine
ADD 8'b0 *// Adds values present in register A and B and store backs in A i.e., A+B=>A*
NEG 8'b0 *//negates the value in register A i.e A= A*-1*
MUL 8'h2 *//multiply with 2(decimal) and store back to A i.e., A=A*2*
RET 0 *//Return back from ISR to the location where interrupt is generated by popping the address from stack and assigning it back to PC.*



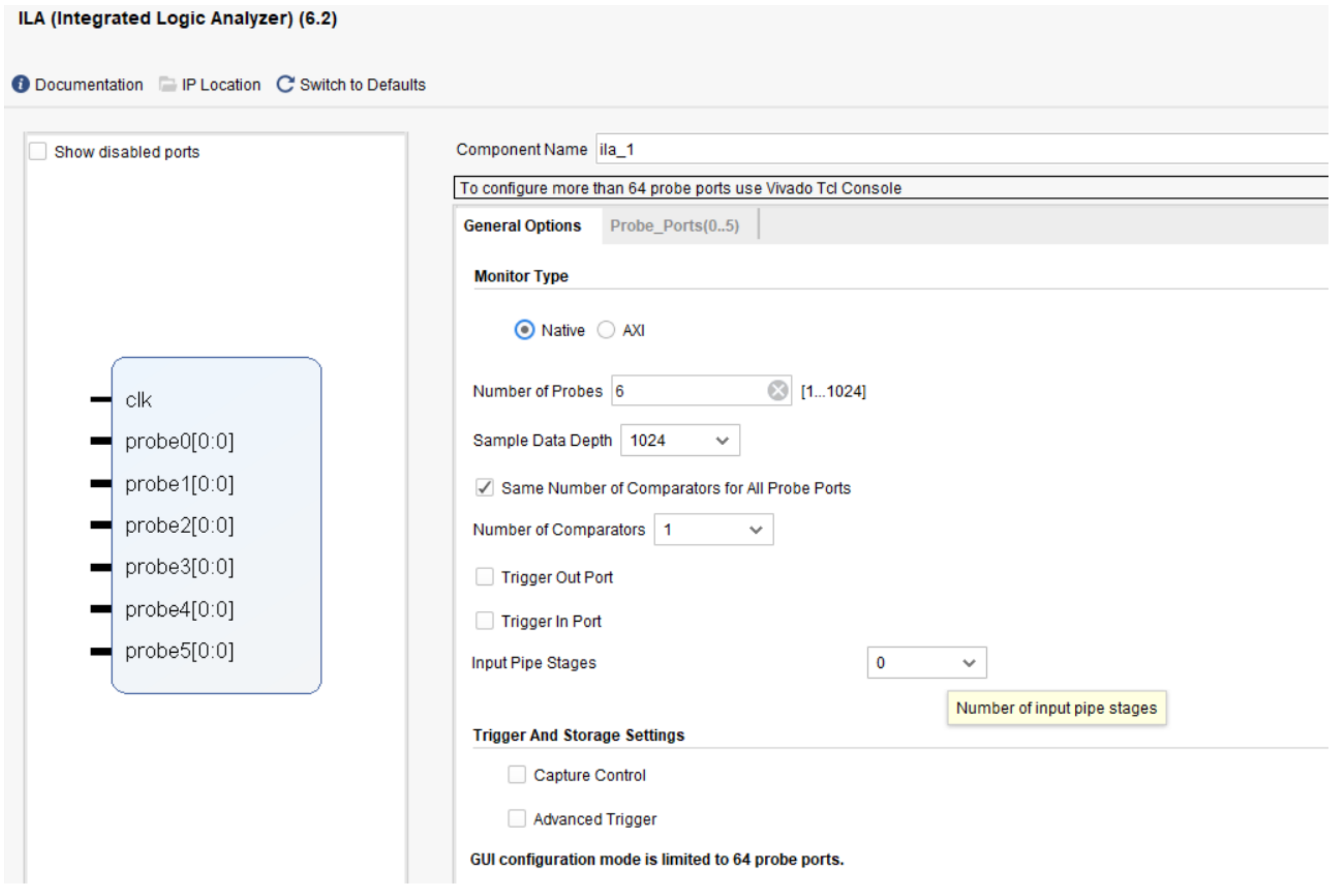
Name	Value
clk	1
A[15:0]	-10
B[15:0]	3
instruction[15:0]	5000
PC_out[11:0]	4
SP[11:0]	fff
Stack[4095:0][15:0]	2,X,X,X,X,X,X,

End Result at A register

HARDWARE DESCRIPTION

The integrated logic analyzer was used to emulate the same result from the FPGA,

ILA: The customizable Integrated Logic Analyzer (ILA) IP core is a logic analyzer core that can be used to monitor the internal signals of a design. The ILA core includes many advanced features of modern logic analyzers, including Boolean trigger equations, and edge transition triggers. Because the ILA core is synchronous to the design being monitored, all design clock constraints that are applied to your design are also applied to the components inside the ILA core



Component Name ila_1

To configure more than 64 probe ports use Vivado Tcl Console

Monitor Type

[✓ Native] AXI

Number of Probes 6 [1...1024]

Sample Data Depth 1024

[✓] Same Number of Comparators for All Probe Ports

Number of Comparators 1

[ ] Trigger Out Port

[ ] Trigger In Port

Input Pipe Stages 0

Number of input pipe stages

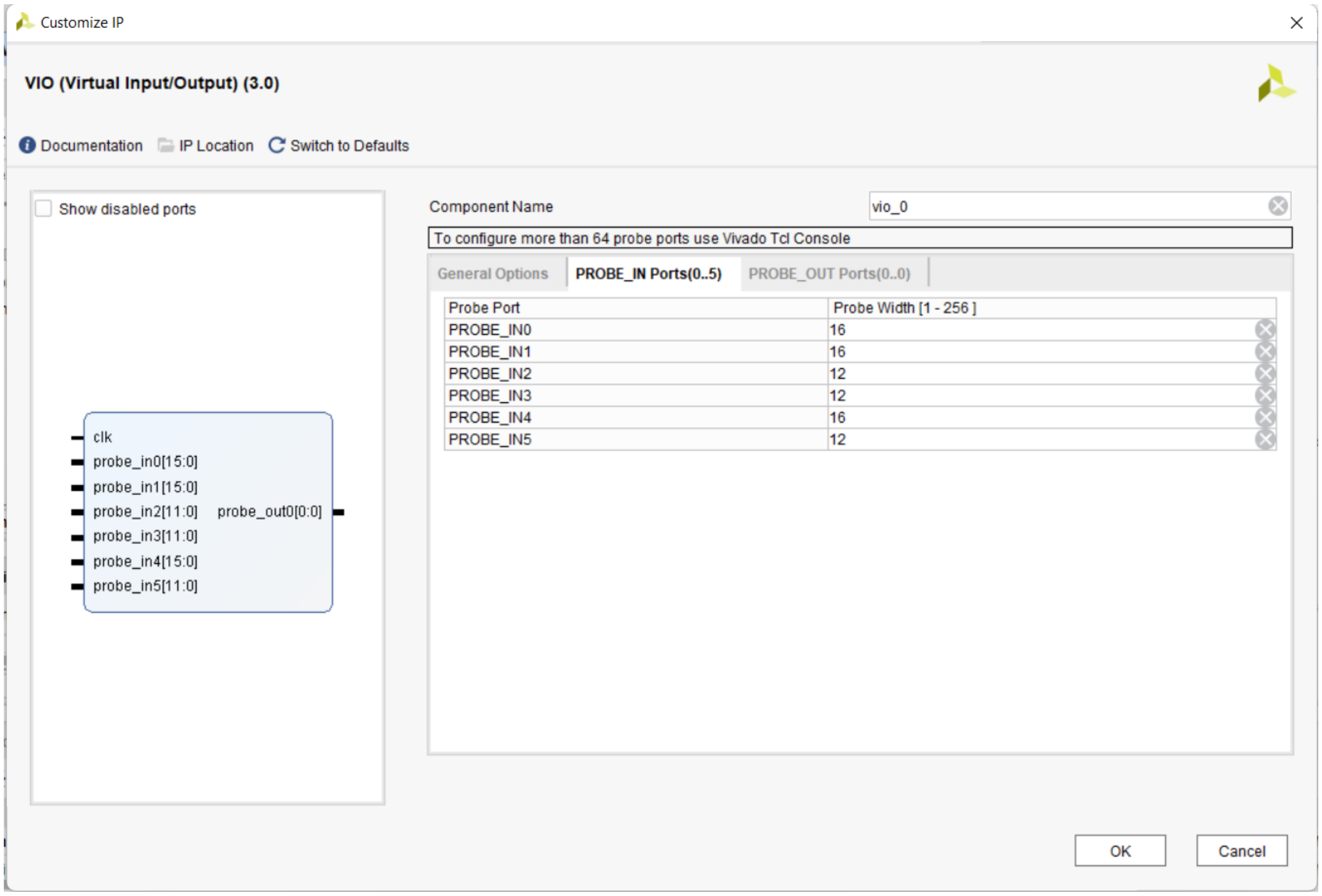
Trigger And Storage Settings

[ ] Capture Control

[ ] Advanced Trigger

GUI configuration mode is limited to 64 probe ports.

VIO: Virtual Input/Output (VIO) core is a customizable core that can both monitor and drive internal FPGA signals in real time. The number and width of the input and output ports are customizable in size to interface with the FPGA design. Because the VIO core is synchronous to the design being monitored and/or driven, all design clock constraints that are applied to your design are also applied to the components inside the VIO core. Run time interaction with this core requires the use of the Vivado® logic analyzer feature.

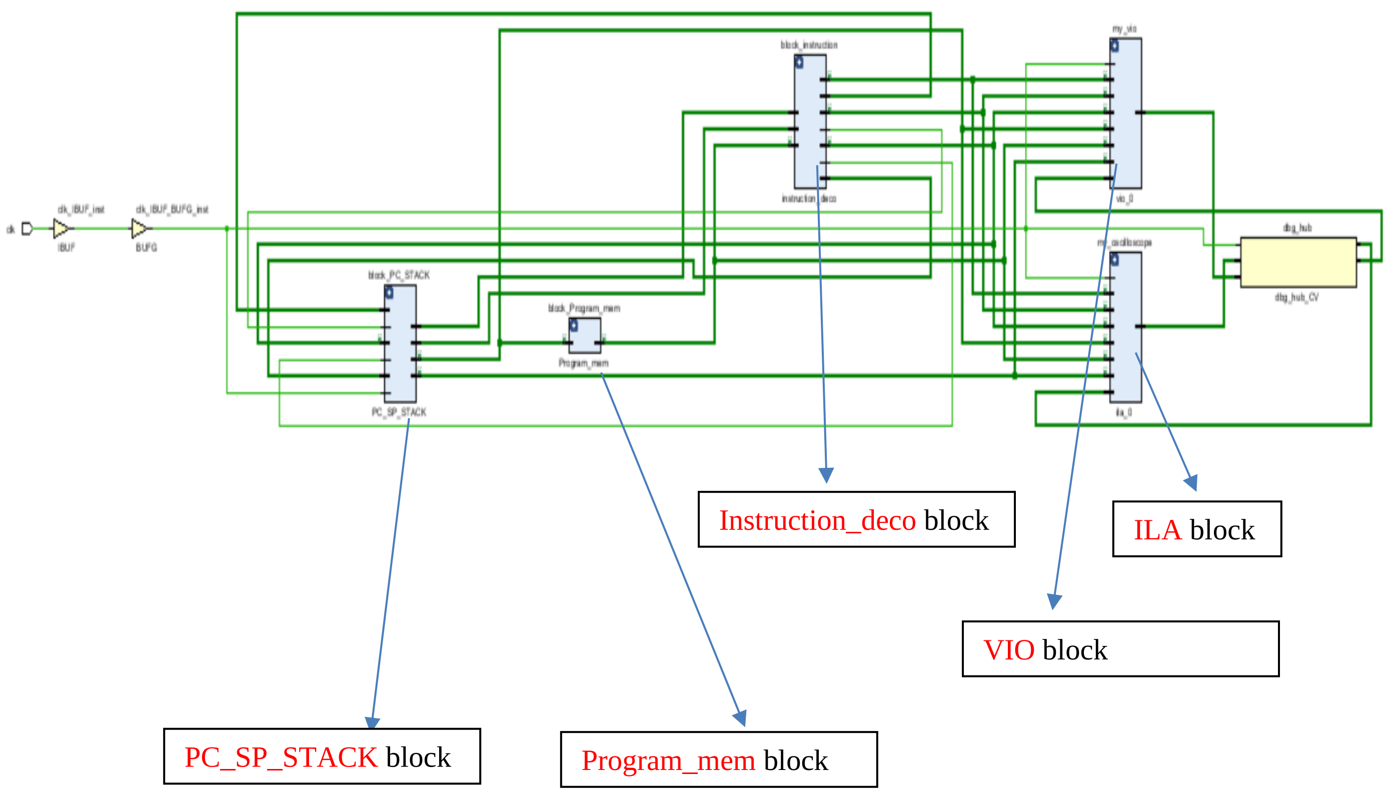


Component Name: vio_0

To configure more than 64 probe ports use Vivado Tcl Console

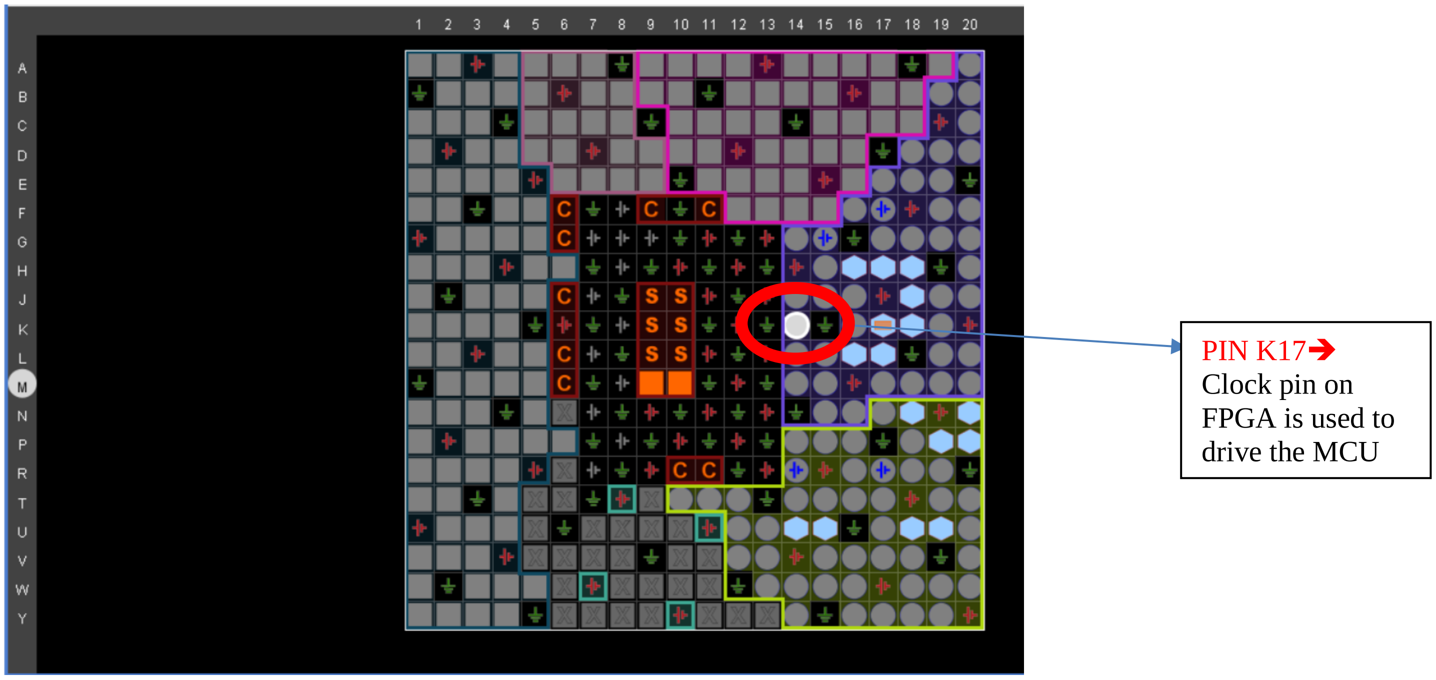
Probe Port	Probe Width [1 - 256]
PROBE_IN0	16
PROBE_IN1	16
PROBE_IN2	12
PROBE_IN3	12
PROBE_IN4	16
PROBE_IN5	12

- **Functional Block Diagram**



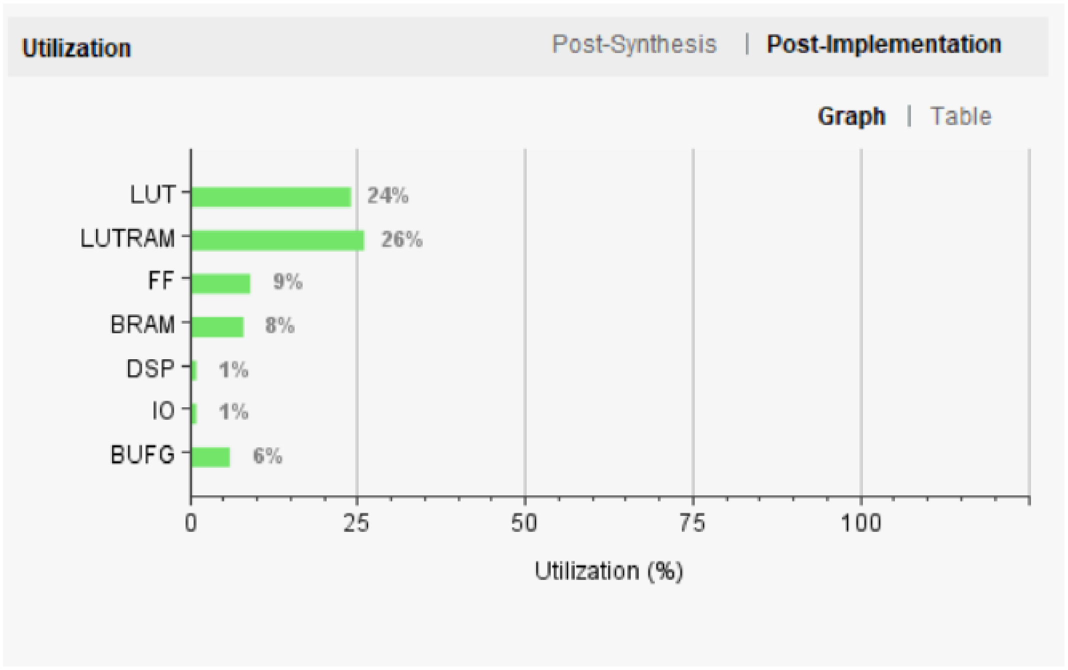
- **Pin Selection For Clock on MCU**

The MCU clock is connected to internal clock of FPGA which is 125Mhz

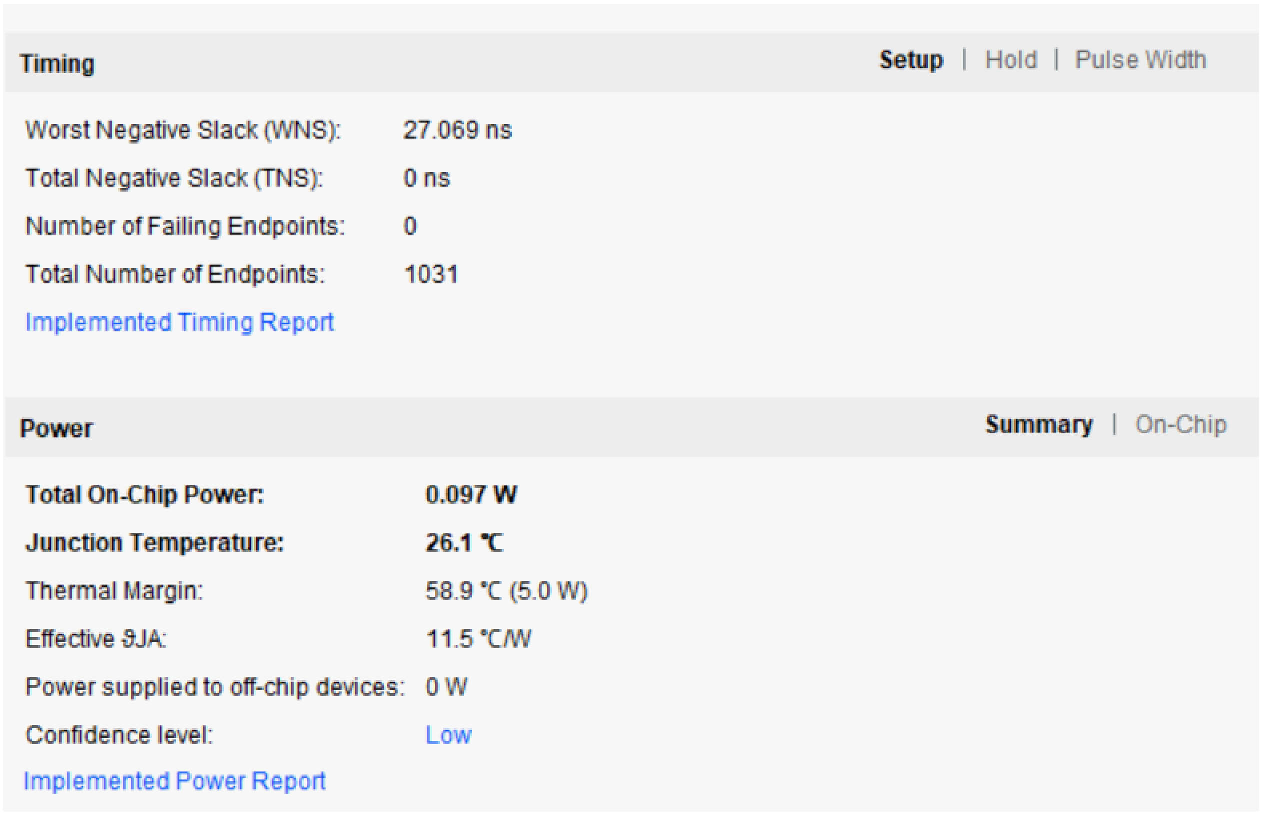


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
A B C D E F G H J K L M N P R T U V W Y
PIN K17➔
Clock pin on FPGA is used to drive the MCU

- **Utilization of FPGA Resources**



- **Timing and Power usage on FPGA**



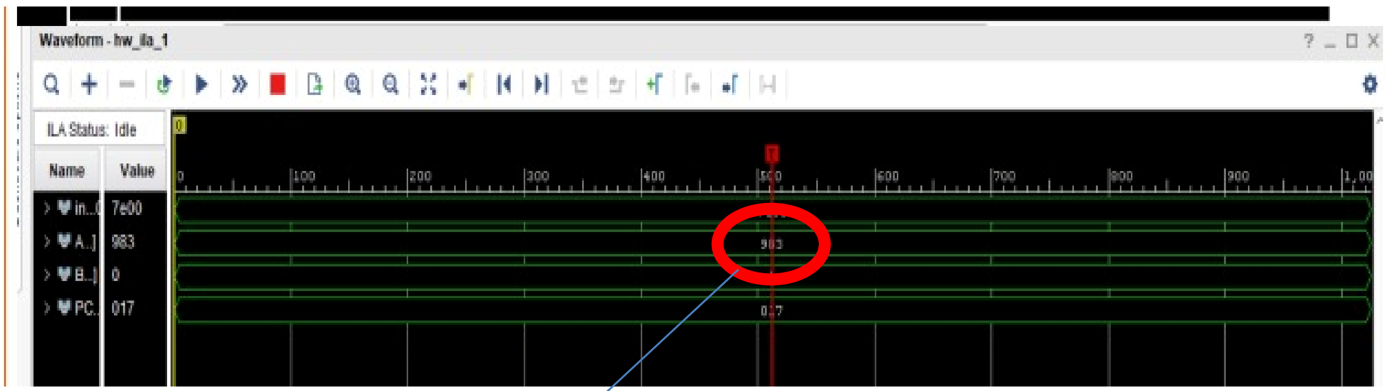
Worst Negative Slack (WNS):	27.069 ns
Total Negative Slack (TNS):	0 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	1031

Implemented Timing Report

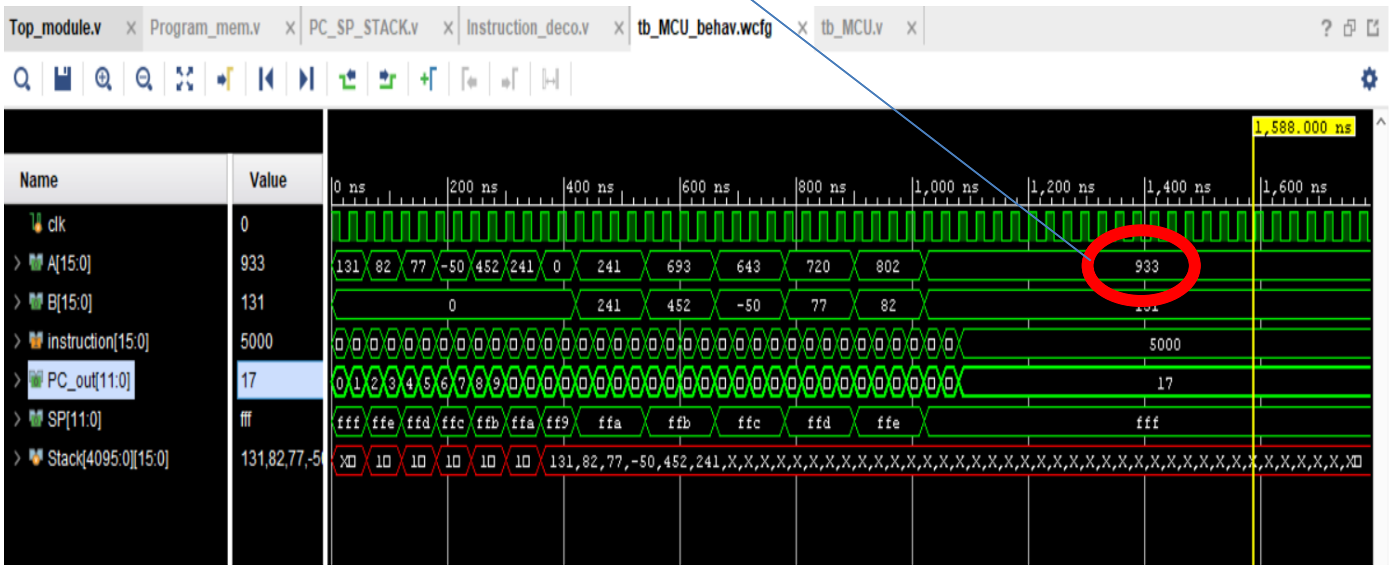
Total On-Chip Power:	**0.097 W**
Junction Temperature:	**26.1 ℃**
Thermal Margin:	58.9 ℃ (5.0 W)
Effective ϑJA:	11.5 ℃/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Implemented Power Report

- **Emulation vs Simulation of test case on FPGA**

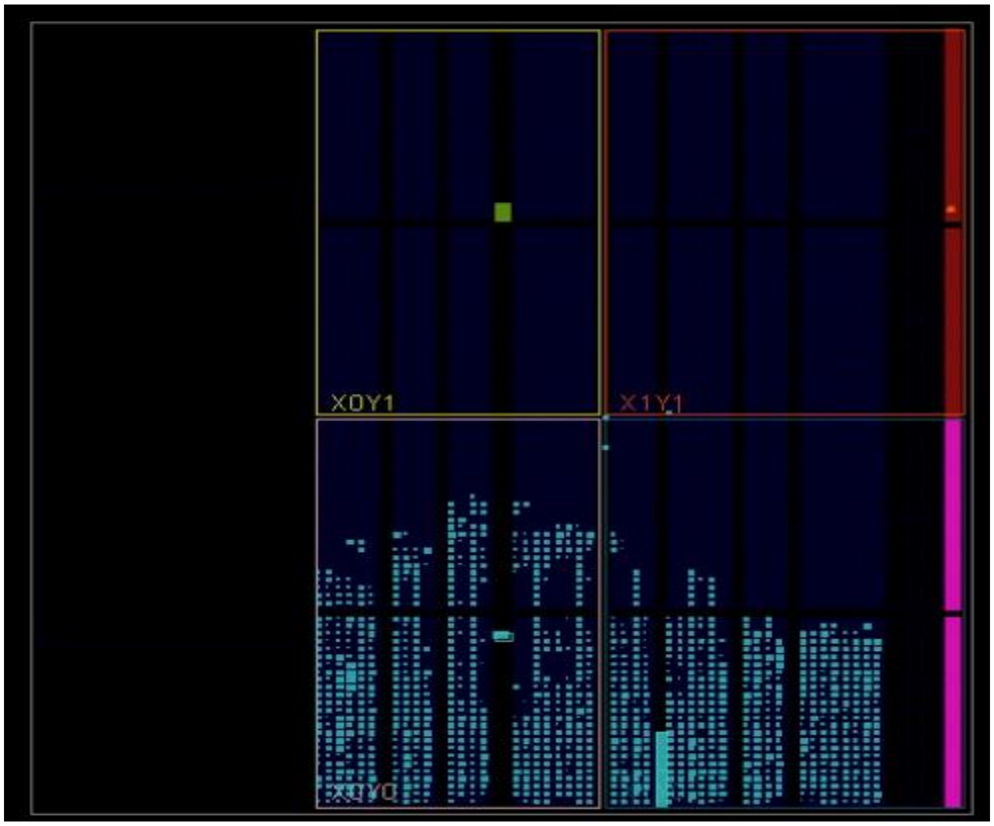


Both Simulation and Emulation on FPGA results to the same value



SYSTEM DESIGN

- **LUT Coverage on ZYBO Z7-10**


X0Y1
X1Y1
X0Y0