NEW YORK UNIVERSITY
TANDON SCHOOL OF ENGINEERING

EL-GY - 6463 ADVANCED COMPUTER HARDWARE DESIGN

Project Title

NYU-6463 MIPS PROCESSOR

Instructor
Prof. Ramesh Karri

Team Members:

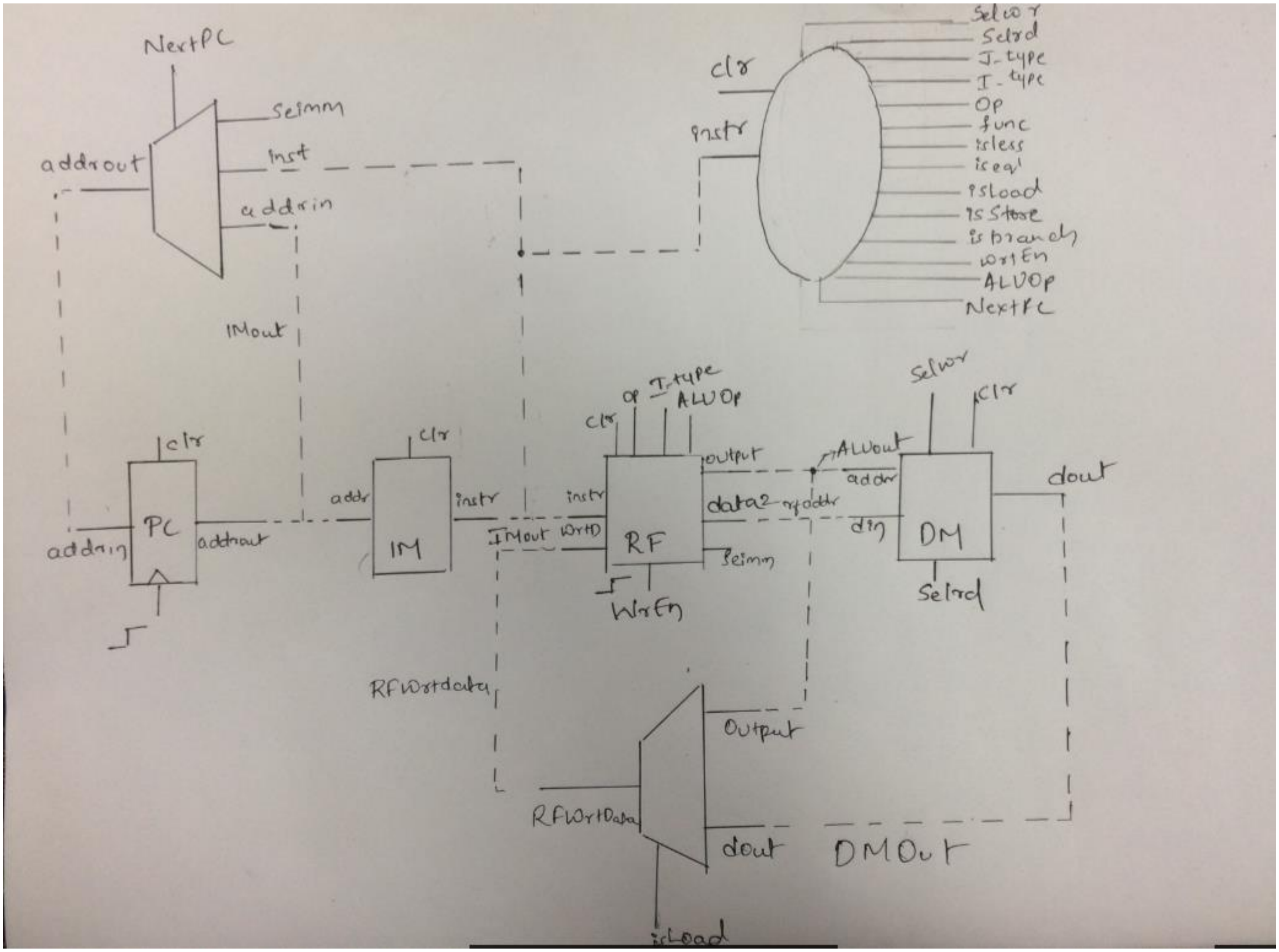
Amit Mohan Redkar (amr1006)

Anant Raghavendra Choudhari(ac6413)

Hrishikesh Garai(hg1230)

Raj Chetan Mehta (rcm445)

1. Processor Design



2.Assembly Codes with Machine Codes:

a. Round Key Generation

ADDI R1, R0, 0	000001 00000 00001 0000000000000000
ADDI R2, R0, 0	000001 00000 00010 0000000000000000
ADDI R3, R0, 0	000001 00000 00011 0000000000000000
ADDI R4, R0, 77	000001 00000 00100 0000000001001101
ADDI R5, R0, 25	000001 00000 00101 0000000000011001
ADDI R6, R0, 3	000001 00000 00101 0000000000011001

```
ADD R7, R0, R2            000001 00000 00101 0000000000011001

ADDI R8, R0, 31           000001 00000 00101 0000000000011001

ADDI R25, R0, 32          000001 00000 00101 0000000000011001

LW R9, 0(R4)              000001 00000 00101 0000000000011001

LW R10, 1(R8)             000001 00000 00101 0000000000011001

LW R11, 2(R8)             000001 00000 00101 0000000000011001


ADD R12, R10, R11       000001 00000 00101 0000000000011001

ADDI R0, R0, 0          000001 00000 00101 0000000000011001

ADD R13, R9, R12        000001 00000 00101 0000000000011001

SHR R15, R13, 28        000001 00000 00101 0000000000011001

SHL R16, R13, 3         000001 00000 00101 0000000000011001

ADDI R17, R0, 26        000001 00000 10001 0000000000011010

OR R9, R16, R15         000001 00000 10001 0000000000011010

LW R18, 0(R17)          000111 10001 10010 0000000000000000

ADD R19, R9, R11        000000 01001 01011 10011 00000 010000

ADDI R20, R0, 0         000001 00000 10100 0000000000000000

ADD R21, R18, R19      000000 10010 10011 10101 00000 010000

AND R22, R19, R8       000000 10011 01000 10110 00000 010010

ADD R23, R0, R21       000000 00000 10101 10111 00000 010000

BEQ R20, R22, 11      001010 10110 10100 0000000000001011

SHL R21, R21, 1       000101 10101 10101 0000000000000001

ADDI R20, R20, 1      000001 10100 10100 0000000000000001

ADD R0, R0, 0         000001 00000 00000 0000000000000000

BNE R20, R22, -4     001011 10110 10100 1111111111111100

ADD R20, R0, 0       000001 00000 10100 0000000000000000

SUB R24, R25, R22   000000 11001 10110 11000 00000 010001

SHR R23, R23, 1     000110 10111 10111 0000000000000001

ADDI R20, R20, 1    000001 10100 10100 0000000000000001
ADDI R0, R0, 0         000001 00000 00000 0000000000000000

BNE R20, R24, -4       001011 11000 10100 1111111111111100

OR R21, R21, R23       000000 10101 10111 10101 00000 010011

SW R9, 0(R4)           001000 00100 01001 0000000000000000

SW R9,1(R8)            001000 01000 01001 0000000000000001

SW R21, 0(R17)         001000 10001 10101 0000000000000000


SW R21,2(R8)           001000 01000 10101 0000000000000010

ADDI R3, R3, 1         000001 00011 00011 0000000000000001

ADDI R2, R2, 1         000001 00010 00010 0000000000000001

ADDI R1, R1, 1         000001 00001 00001 0000000000000001

BNE R2, R5, 1          001011 00110 00011 0000000000000001

ADDI R2, R0, 0         000001 00000 00011 0000000000000000

BNE R1, R4, -40        001011 00101 00010 0000000000000001
```

b. Encryption:

```
ADDI R1 R0             0000001 00000 00001 00000 00000 000000

ADDI R2 R0 12          000001 00000 00010 00000 00000 001100

ADDI R3 R0 29          000001 00000 00011 00000 00000 011101

ADDI R4 R0 31          000001 00000 00100 00000 00000 011111

ADDI R5 R0 2           000001 00000 00101 00000 00000 000010

ADDI R6 R0 3           000001 00000 00110 00000 00000 000011

LW R7 0(R1)            000111 00001 00111 00000 00000 000000

LW R8 1(R1)            000111 00001 01000 00000 00000 000001

LW R9 1(R3)            000111 00011 01001 00000 00000 000001

LW R10 2(R3)           000111 00011 01010 00000 00000 000010

ADD R11 R9 R7          000000 00111 01001 01011 00000 010000

ADD R12 R10 R8         000000 01000 01010 01100 00000 010000

SW R11 19R4)           001000 00100 01011 00000 00000 000001
SW R12 2(R4)            001000 00100 01100 00000 00000 000010

ADDI R1 R1 1            000001 00001 00001 00000 00000 000001

LW R11 1(R4)            000111 00100 01101 00000 00000 000001

LW R12 2(R4)            000111 00100 01100 00000 00000 000010

LW R7 0(R5)             000111 00101 00111 00000 00000 000000

LW R8 0(R6)             000111 00110 01000 00000 00000 000000

ADDI R5 R5 2            000001 00101 00101 00000 00000 000010

ADDI R6 R6 2            000001 00110 00110 00000 00000 000010

NOR R13 R11 R11         000000 01011 01011 01101 00000 010100

NOR R14 R12 R12         000000 01100 01100 01110 00000 010100

AND R15 R11 R14         000000 01011 01110 01111 00000 010010

AND R16 R12 R13         000000 01100 01101 10000 00000 010010

 ADDI R18 R0 31         000001 00000 10010 00000 00000 100000

OR R17 R15 R16          000000 10000 01111 10001 00000 010011

ADDI R19 R0 0           000001 00000 10011 00000 00000 000000

AND R20 R18 R12         000000 10010 01100 10100 00000 010010

ADD R21 R17 R0          000000 10001 00000 10101 00000 010000

BEQ R19 R20 10          001010 10100 10011 00000 00000 001010

SHL R17 R17 1           000101 10001 10001 00000 00000 000001

ADDI R19 R19 1          000001 10011 10011 00000 00000 000001

ADDI R0 R0 0            000001 00000 00000 00000 00000 000000

BNE R19 R20 -4          001011 10100 10011 11111 11111 111100

ADDI R19 R0 0           000001 00000 10011 00000 00000 000000

SUB R20 R18 R20         000000 10010 10100 10100 00000 010001

SHR R21 R21 1           000110 10101 10101 00000 00000 000001

ADDI R19 R19 1          000001 10011 10011 00000 00000 000001

BNE R19 R20 -3          001011 10100 10011 11111 11111 111101

OR R17 R17 R21          000000 10001 10101 10001 00000 010011

ADD R11 R17 r7          000000 10001 00111 01101 00000 010000
```

```
Nor r13 r11 r11         000000 01011 01011 01101 00000 010100
And r15 r11 r14         000000 01011 01110 01111 00000 010010
And r16 r12 r13         000000 01100 01101 10000 00000 010010
or r17 r15 r16          000000 01111 10000 10001 00000 010011
addi r19 r0 0           000001 00000 10011 00000 00000 000000
and r20 r18 r11         000000 10010 01011 10100 00000 010010
add r21 r17 r0          000000 10001 00000 10101 00000 010000
beq r19 r20 10          001010 10100 10011 00000 00000 001010
shl r17 r17 1           000101 10001 10001 00000 00000 000001
addi r19 r19 1          000001 10011 10011 00000 00000 000001
addi r0 r0 0            000001 00000 00000 00000 00000 000000
bne r19 r20 -4          001011 10100 10011 11111 11111 111100
addi r19 r0 0           000001 00000 10011 00000 00000 000001
SUB r20 r18 r20         000000 10100 10100 10010 00000 010001
shr r21 r21 1           000110 10101 10101 00000 00000 000001
addi r19 r19 1          000001 10011 10011 00000 00000 000001
bne r19 r20 -3          001011 10100 10011 11111 11111 111101
and r12 r17 r8          000000 01000 10001 01100 00000 010010
or r17 r17 r21          000000 10101 10001 10001 00000 010011
addi r1 r1 1            000001 00001 00001 00000 00000 000001
sw r11, 1(r4)           001000 00100 01011 00000 00000 000001
sw r12, 2(r4)           001000 00100 01100 00000 00000 000010
Bne r1 r2 -50           001011 00010 00001 11111 11111 001110
HAL                     111111 00000 00000 00000 00000 000000
```

c. Decryption

```
addi r1 r0 12     000001 00000 00001 00000 00000 001100
addi r2 r0 1      000001 00000 00010 00000 00000 000001
addi r3 r0 31     000001 00000 00011 00000 00000 011111
```

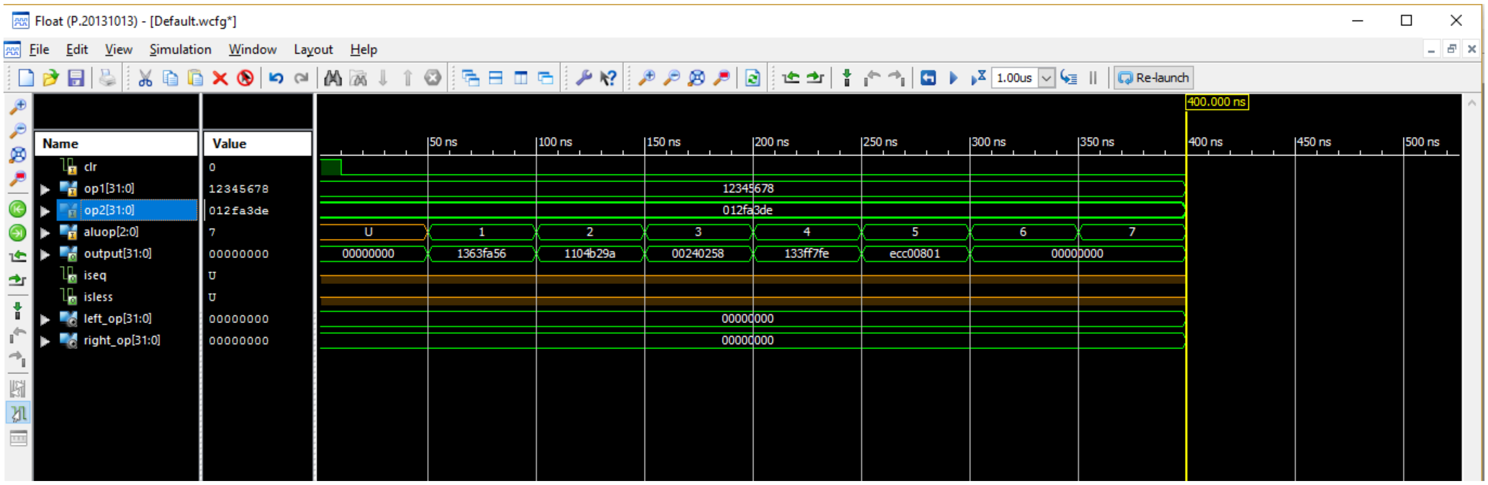
addi r4 r0 25	000001 00000 00100 00000 00000 011001
addi r5 r0 24	000001 00000 00101 00000 00000 011000
lw r6 1(r3)	000111 00011 00110 00000 00000 000001
lw r7 2(r3)	000111 00011 00111 00000 00000 000010
lw r5 0(r4)	000111 00100 00101 00000 00000 000000
lw r7 0(r5)	000111 00101 00111 00000 00000 000000
subi r4 r4 2	000010 00100 00100 00000 00000 000010
subi r5 r5 2	000010 00101 00101 00000 00000 000010
sub r10 r7 r4	000000 00111 00100 01010 00000 010001
addi r11 r0 32	000001 00000 01101 00000 00000 100000
addi r12 r0 0	000001 00000 01100 00000 00000 000000
and r13 r7 r11	000000 00111 01011 01101 00000 010010
add r14 r10 r0	000000 01010 00000 01110 00000 000001
beq r12 r13 8	001010 01101 01100 00000 00000 001000
shr r10 r10 1	000110 01010 01010 00000 00000 000001
addi r12 r12 1	000001 01100 01100 00000 00000 000001
bne r12 r13 -3	001011 01101 01100 11111 11111 111101
addi r12 r0 0	000001 00000 01100 00000 00000 000000
sub r15 r11 r13	000000 01011 01101 01111 00000 010001
shl r14 r14 1	000101 01110 01110 00000 00000 000001
addi r12 r12 1	000001 01100 01100 00000 00000 000001
bne r12 r15 -3	001011 01111 01100 11111 11111 111101
or r10 r10 r14	000000 01010 01110 01010 00000 010011
nor r16 r6 r6	000000 00110 00110 10000 00000 010100
nor r17 r17 r7	000000 10001 00111 10001 00000 010100
and r18 r16 r7	000000 10000 00111 10010 00000 010010
and r19 r17 r6	000000 10001 00110 10011 00000 010010
or r7 r18 r19	000000 10010 10011 00111 00000 010011
sub r20 r6 r1	000000 00110 00001 10010 00000 010001

```
addi r12 r0 0      000001 00000 01100 00000 00000 000000

and r13 r6 r11     000000 00110 01010 01101 00000 010010

add r14 r20 r0     000000 10100 00000 01110 00000 000001

beq r12 r13 8      001010 01101 01100 00000 00000 001000

shr r20 r20 1      000110 10100 10100 00000 00000 000001

add r12 r12 1      000000 01100 01100 00000 00000 000001

bne r12 r13 -3     001011 01101 01100 11111 11111 111101

addi r12 r0 0      000001 00000 01100 00000 00000 000000

sub r15 r11 r13    000000 01011 01101 01111 00000 010001

shl r14 r14 1      000101 01110 01110 00000 00000 000001

addi r12 r12 1     000001 01100 01100 00000 00000 000001

bne r12 r15 -3     001011 01111 01100 11111 11111 111101

or r20 r20 r14     000000 10100 01110 10100 00000 010011

nor r17 r7 r7      000000 00111 00111 10001 00000 010100

and r15 r16 r7     000000 10000 00111 01111 00000 010010

and r19 r17 r6     000000 10001 00110 10010 00000 010010

or r6 r18 r11      000000 10010 01011 00110 00000 010011

sw r6 1(r3)        001000 00011 00110 00000 00000 000001

sw r7 2(r3)        001000 00011 00111 00000 00000 000010

subi r1 r1 1       000010 00001 00001 00000 00000 000001

bne r1 r2 -49      001011 00010 00001 11111 11111 001111

subi r1 r1 1       000010 00001 00001 00000 00000 000001

subi r4 r4 2       000010 00100 00100 00000 00000 000010

subi r5 r5 2       000010 00101 00101 00000 00000 000010

sw r6 1(r3)        001000 00011 00110 00000 00000 000001

sw r7 2(r3)        001000 00011 00111 00000 00000 000010

lw r6 1(r3)        000111 00011 00110 00000 00000 000001

lw r7 2(r3)        000111 00011 00111 00000 00000 000010

lw r8 0(r4)        000111 00100 01000 00000 00000 000000
```

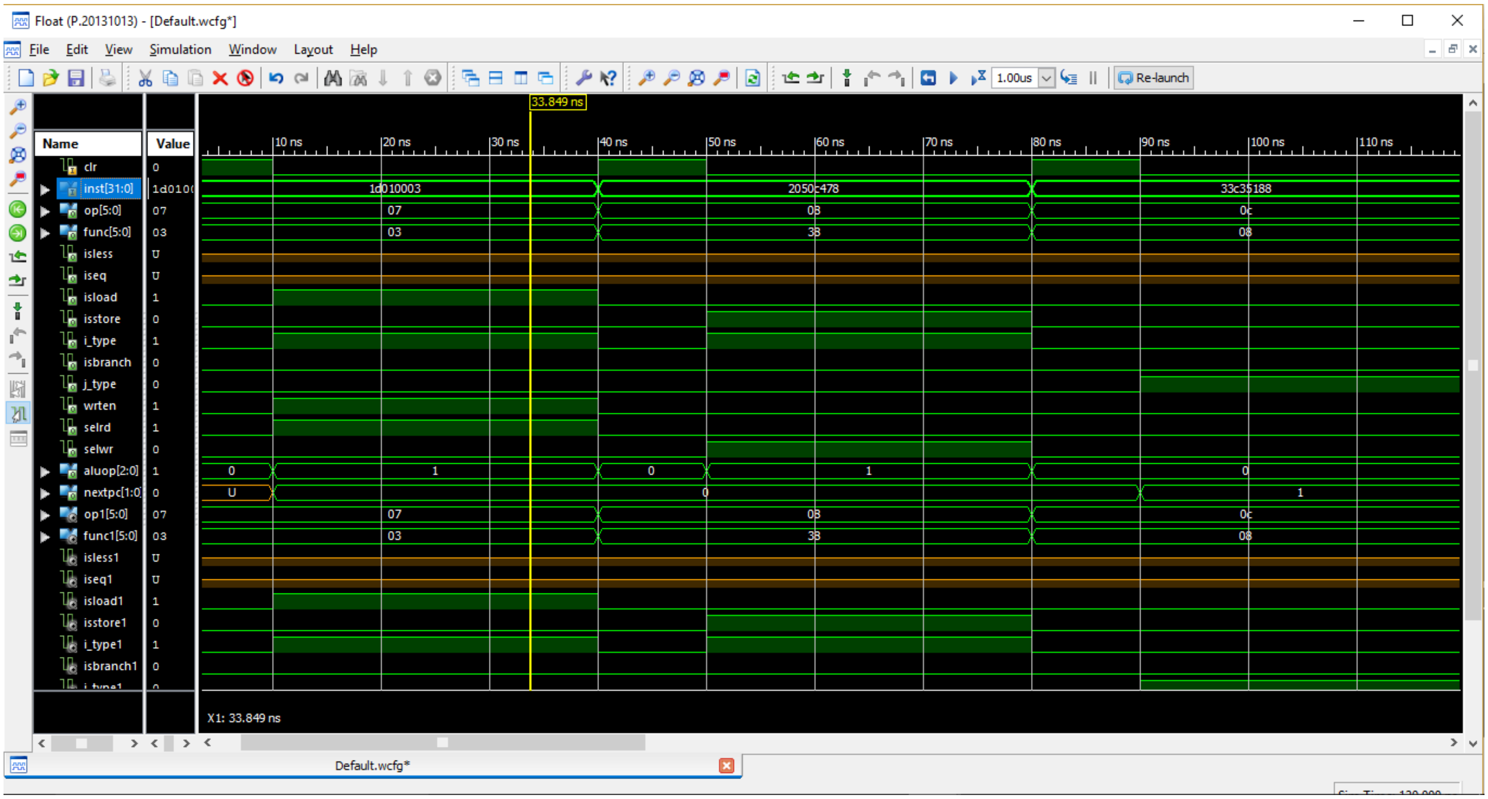
lw r9 0(r5)	000111 00101 01001 00000 00000 000000
sub r7 r7 r8	000000 00111 01000 00111 00000 000001
sub r6 r6 r9	000000 00110 01001 00110 00000 000001
sw r6 1(r3)	001000 00011 00110 00000 00000 000001
sw r7 2(r3)	001000 00011 00111 00000 00000 000010

3. Screenshots of each component working:

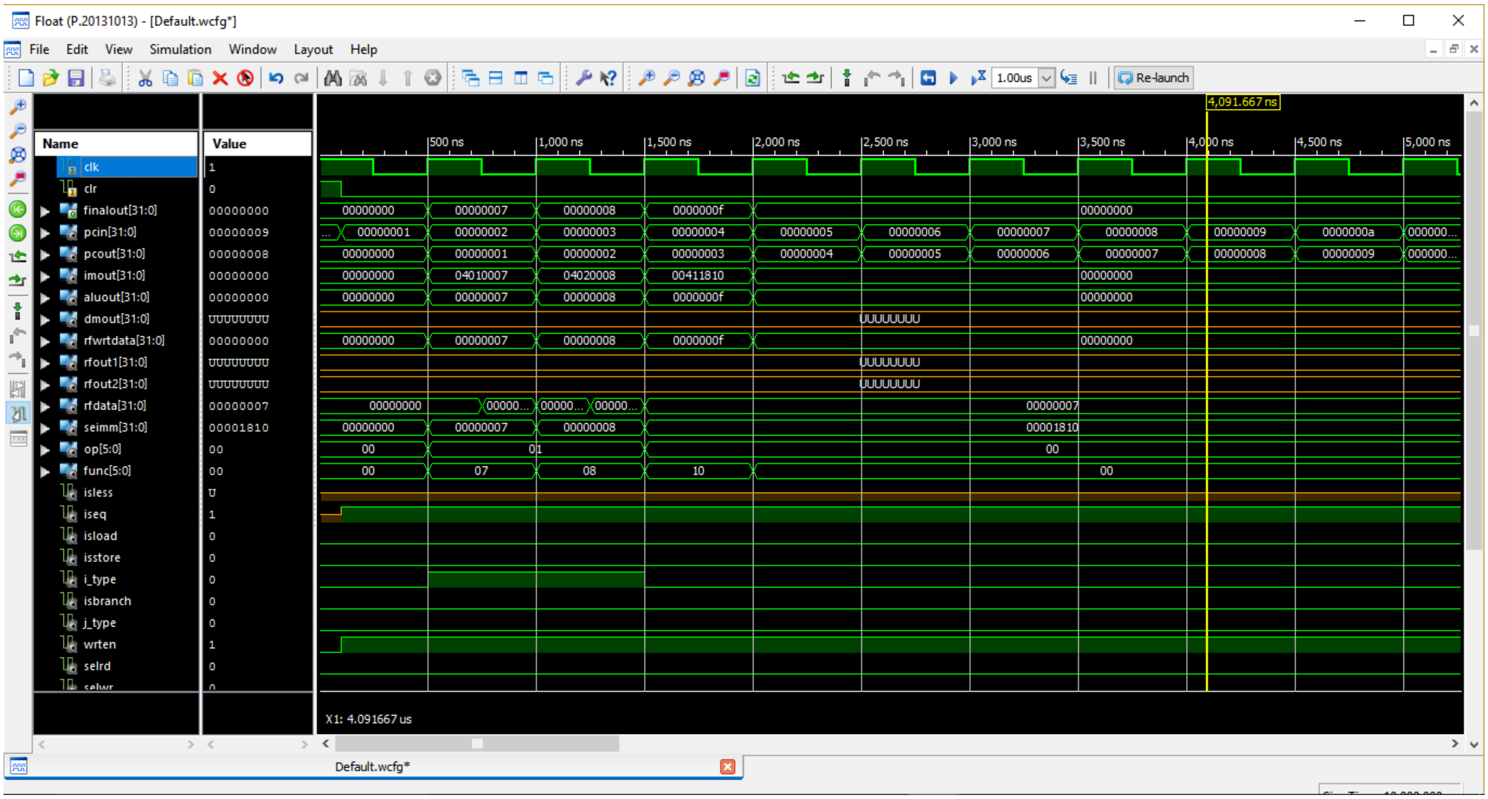
a. ALU Simulation:



b. Decoder Simulation:

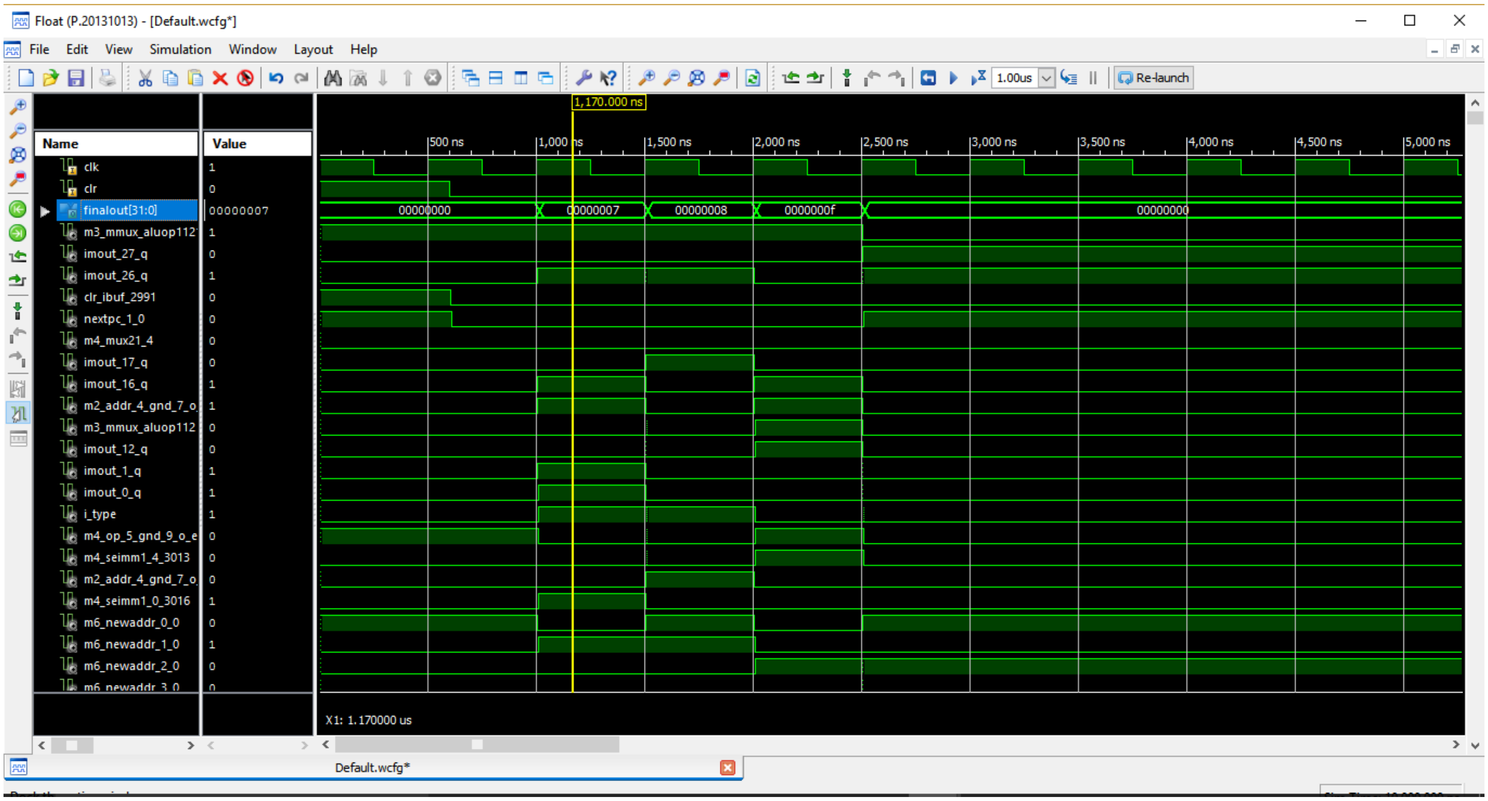


c. Sample Code 1 Functional Simulation


Float (P.20131013) - [Default.wcfg*]
File Edit View Simulation Window Layout Help
1.00us
Re-launch
4,091.667 ns
|500 ns |1,000 ns |1,500 ns |2,000 ns |2,500 ns |3,000 ns |3,500 ns |4,000 ns |4,500 ns |5,000 ns
X1: 4.091667 us
Default.wcfg*

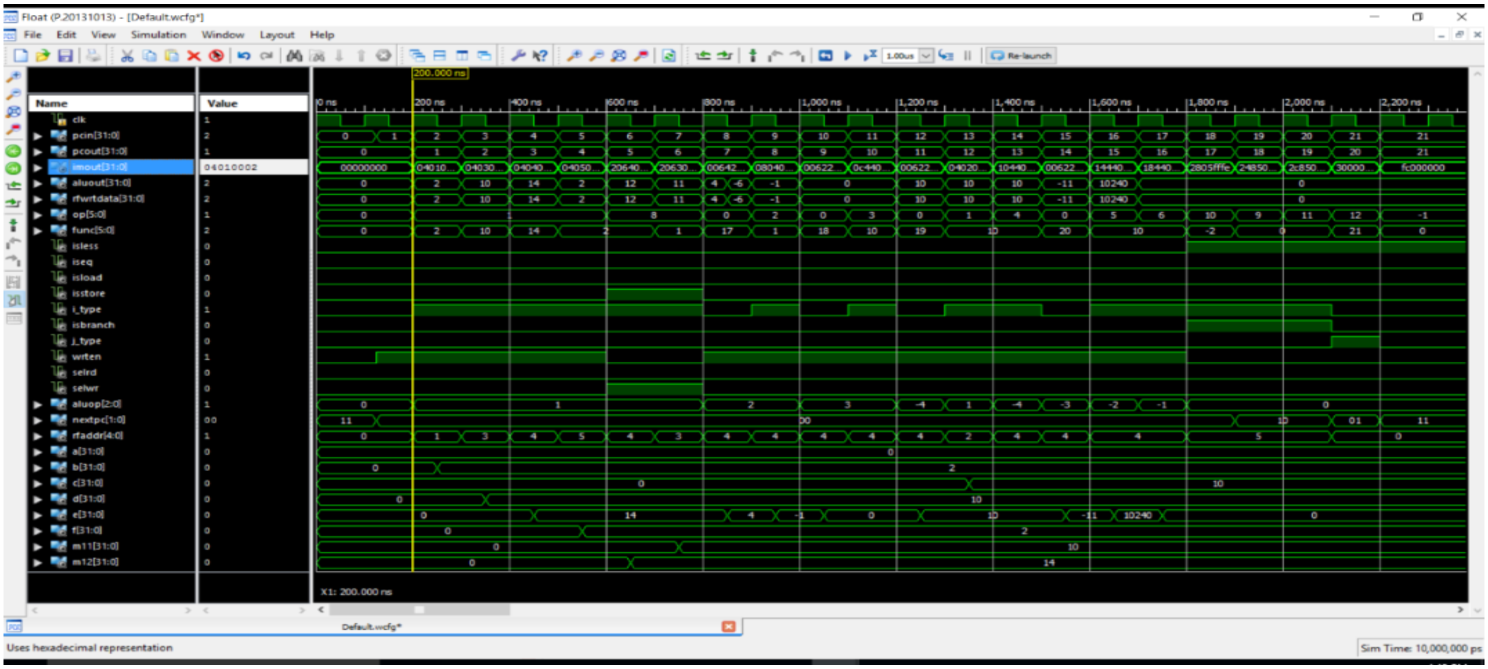
Name	Value
clk	1
clr	0
finalout[31:0]	00000000
pcin[31:0]	00000009
pcout[31:0]	00000008
imout[31:0]	00000000
aluout[31:0]	00000000
dmout[31:0]	UUUUUUUU
rfwrtdata[31:0]	00000000
rfout1[31:0]	UUUUUUUU
rfout2[31:0]	UUUUUUUU
rfdata[31:0]	00000007
seimm[31:0]	00001810
op[5:0]	00
func[5:0]	00
isless	U
iseq	1
isload	0
isstore	0
i_type	0
isbranch	0
j_type	0
wrten	1
selrd	0
selwr	0

d. Sample Code 1 Timing Simulation


Float (P.20131013) - [Default.wcfg*]
File Edit View Simulation Window Layout Help
1.00us
Re-launch
1,170.000 ns
|500 ns |1,000 ns |1,500 ns |2,000 ns |2,500 ns |3,000 ns |3,500 ns |4,000 ns |4,500 ns |5,000 ns
00000000 00000007 00000008 0000000f 00000000
X1: 1.170000 us
Default.wcfg*

Name	Value
clk	1
clr	0
finalout[31:0]	00000007
m3_mmux_aluop112	1
imout_27_q	0
imout_26_q	1
clr_ibuf_2991	0
nextpc_1_0	0
m4_mux21_4	0
imout_17_q	0
imout_16_q	1
m2_addr_4_gnd_7_o	1
m3_mmux_aluop112	0
imout_12_q	0
imout_1_q	1
imout_0_q	1
i_type	1
m4_op_5_gnd_9_o_e	0
m4_seimm1_4_3013	0
m2_addr_4_gnd_7_o	0
m4_seimm1_0_3016	1
m6_newaddr_0_0	0
m6_newaddr_1_0	1
m6_newaddr_2_0	0
m6_newaddr_3_0	0

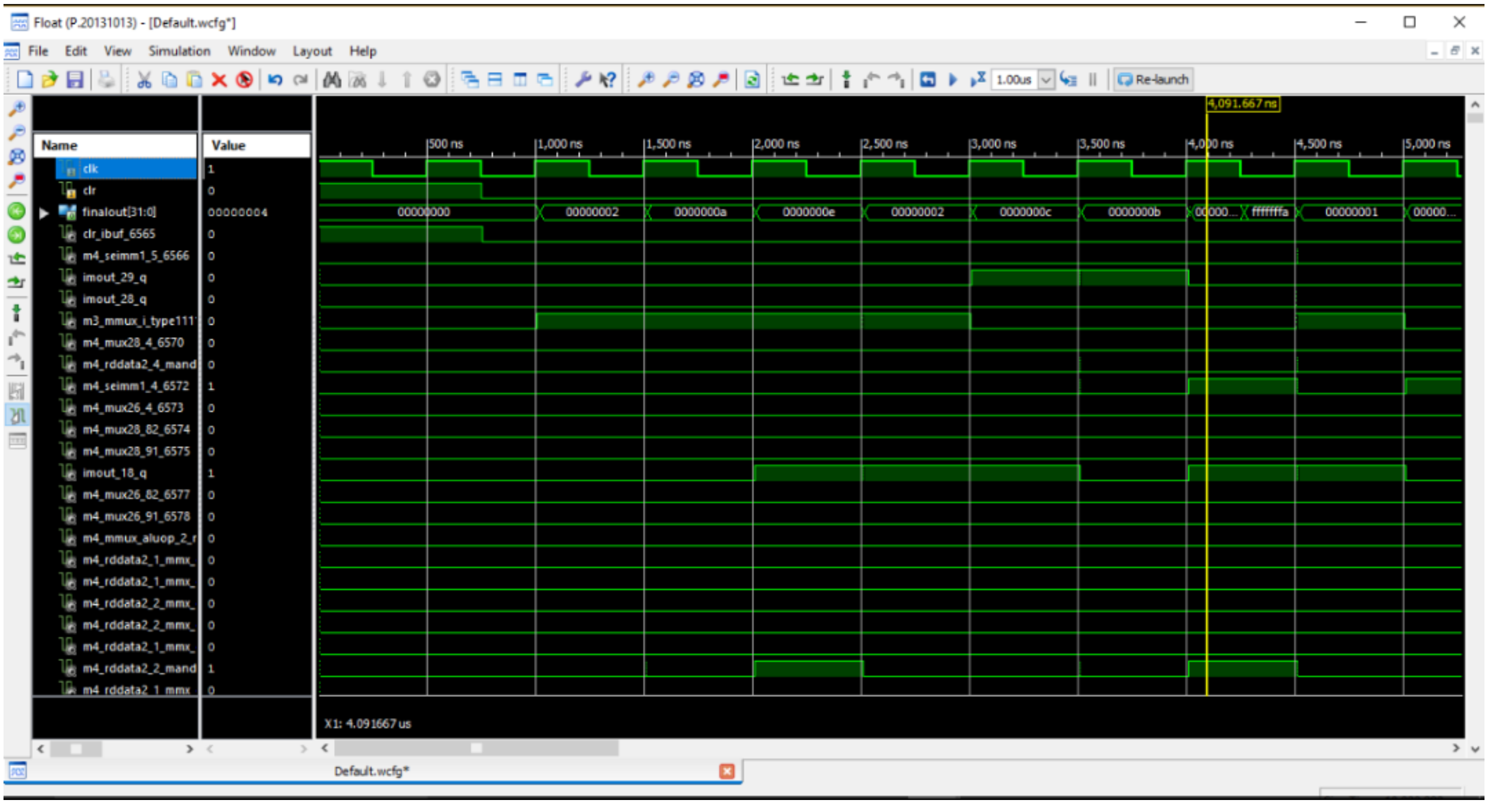
Sample Code 2 : Functional Simulation


Float (P.20131013) - [Default.wcfg*]
File Edit View Simulation Window Layout Help
Re-launch
200.000 ns
X1: 200.000 ns
Default.wcfg*
Sim Time: 10,000,000 ps

Name	Value
clk	1
pcin[31:0]	2
pcout[31:0]	1
imout[31:0]	04010002
aluout[31:0]	2
rfwrtdata[31:0]	2
op[5:0]	1
func[5:0]	2
isless	0
iseq	0
isload	0
isstore	0
i_type	1
isbranch	0
j_type	0
wrten	1
selrd	0
selwr	0
aluop[2:0]	1
nextpc[1:0]	00
rfaddr[4:0]	1
a[31:0]	0
b[31:0]	0
c[31:0]	0
d[31:0]	0
e[31:0]	0
f[31:0]	0
m11[31:0]	0
m12[31:0]	0

Uses hexadecimal representation

Sample 2 : Timing Simulation


Float (P.20131013) - [Default.wcfg*]
File Edit View Simulation Window Layout Help
Re-launch
4,091.667 ns
X1: 4.091667 us
Default.wcfg*

Name	Value
clk	1
clr	0
finalout[31:0]	00000004
clr_ibuf_6565	0
m4_seimm1_5_6566	0
imout_29_q	0
imout_28_q	0
m3_mmux_i_type111	0
m4_mux28_4_6570	0
m4_rddata2_4_mand	0
m4_seimm1_4_6572	1
m4_mux26_4_6573	0
m4_mux28_82_6574	0
m4_mux28_91_6575	0
imout_18_q	1
m4_mux26_82_6577	0
m4_mux26_91_6578	0
m4_mmux_aluop_2_r	0
m4_rddata2_1_mmx_	0
m4_rddata2_1_mmx_	0
m4_rddata2_2_mmx_	0
m4_rddata2_2_mmx_	0
m4_rddata2_1_mmx_	0
m4_rddata2_2_mand	1
m4_rddata2_1_mmx	0

4. High Level Description:

Description:

Implementing MIPS Architecture:

The MIPS Architecture consists of the following components:

Program Counter: The program counter specifies the address of the next instruction that is to be executed. The input to the PC is the next instruction to be executed and the output is also the same. On a clock event, the address is released from the PC to the Instruction Memory. In the case of reset, the PC gets initialised and the output address becomes 0 irrespective of the address input to the PC.

Instruction Memory: The instruction memory gives the instruction set to be executed based on the value received from the Program Counter. The input to the IM is the address received from the PC and the output is the 32-bit instruction set. It does not depend on a clock. On reset, the instruction set fetched is 0.

Decoder: Decoder gets the instruction from the instruction memory output and then based on the opcode and other fields present in the instruction, generates various signals that control the other modules in the design.

Register File: This consists of 32, 32- bit registers. The addresses given in the instruction is used in the RF to fetch the data to be used by the ALU.

ALU: ALU stands for Arithmetic Logic Unit and is used for all the mathematical computations such as ADD, SUB, load, store, ADDI, SUBI and even the calculations needed for the branch and jump instructions. It receives input from the register file.

DM: Data Memory is a RAM that stores all the outputs of the computations and even provide values when needed from a particular location. Using SW instruction we can store a value into the data memory and using the LW instruction, we can load a value from a given data memory location.

Mux: The mux is used to calculate the next address for the Program Counter. Also it is used to select the data to be written in the Register File. If it is a load instruction, the data from Data Memory will be written in the Register File, otherwise the output from ALU will be written to the Register File.

Implementation of RC5 on MIPS ISA

The assembly language code for RC5 is generated and converted to Machine Language code which is used as the Instruction Set for the MIPS ISA. The assembly language and the machine code for the RC5 is shown above.

For an RC5 to work, the two most important inputs required are User Key and Data In. The input can be provided to the MIPS ISA in three ways:

Dynamic Control to the user through FPGA:

The user can dynamically input the User Keys and the Data Input to be encrypted by using the FPGA resources likes Switches and Buttons.

Dynamic Control by generating an Instruction Set for the user key and data in: The user can provide the data and the user key as a part of the instruction set and then the instruction for key generation, encryption and decryption will be run as it is.

Hard-Coding the values directly in the Data Memory: Another option for loading the user key and data input is directly hard coding the data into the Data Memory.

Round Key Generation: Round Key Generation is used to generate the user key that is used to encrypt some user data and the same key is used by the receiver to decrypt the data. So the attacker or any other person who does not have the key cannot get the data. In this project, we used the provided algorithm to generate the key. We had limited instructions, to execute our code so we rotated right by shifting it right first by the number of bit, then subtracting the number from 32, shifted the original value to the left by the new number we obtained by subtracting. Then we or'ed the right shifted and left shifted data to get the new rotated output.

Encryption: Encryption is the process of using an algorithm and keys to change the plaintext or original data into some form (cipher text) that cannot be read by anyone not knowing the key to reverse the cipher text data.

Decryption: Decryption is the process of converting the Cipher text to plain text. The encrypted data generated from the Encryption Module is converted into plain text using an array of numbers (round key) which is used for encryption.

The assembly codes for all the three modules is shown above.

5. Description of Processor interfaces:

We have mapped the 'clr' signal to the FPGA board by using the Button located at the P18 location.

The end result is displayed on the Seven Segment display of the FPGA board.

6. Performance and Area analysis: Sample Code 2

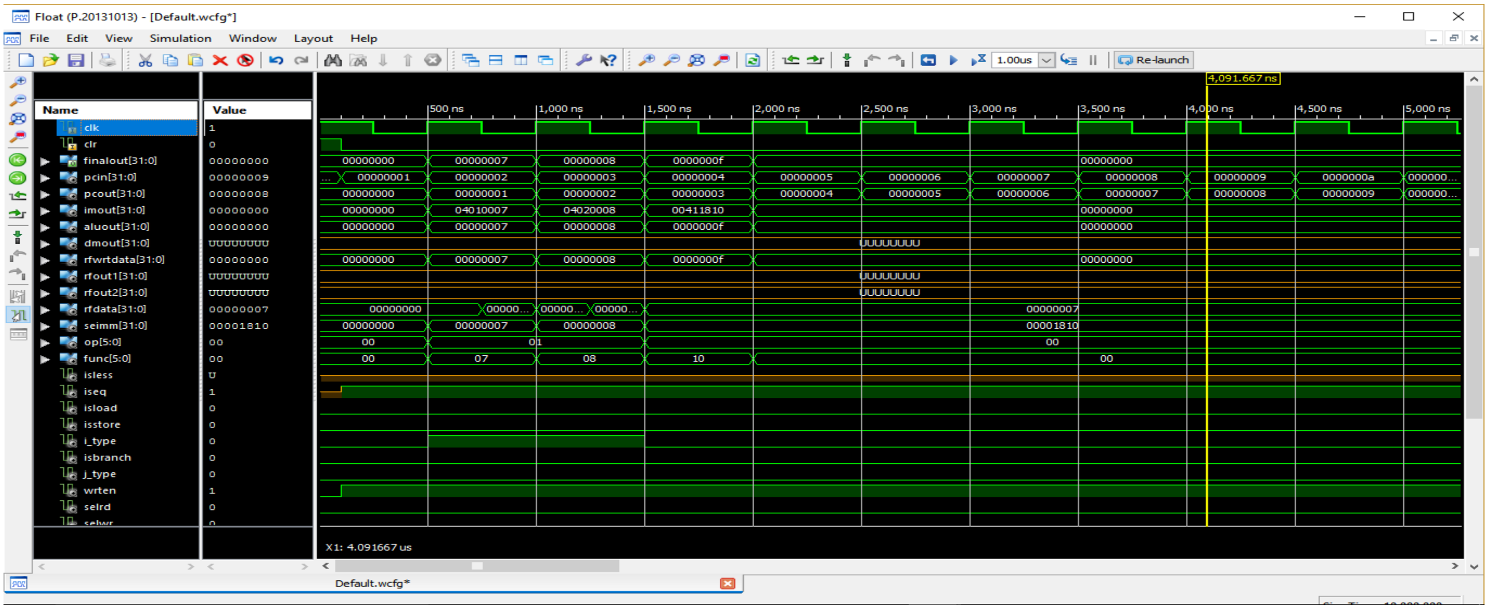
Resources	Post Synthesis	Post PAR
Slice Registers	308	308
Slice LUTs	950	904
IOBs	34	34

Min Period : 4.53ns

Max Freequency: 220.750MHz

7. Verification of the design:

Comparing the outputs of the sample program, we can verify the working of the design as follows:



Name	Value
clk	1
clr	0
finalout[31:0]	00000000
pcin[31:0]	00000009
pcout[31:0]	00000008
imout[31:0]	00000000
aluout[31:0]	00000000
dmout[31:0]	UUUUUUUU
rfwrtdata[31:0]	00000000
rfout1[31:0]	UUUUUUUU
rfout2[31:0]	UUUUUUUU
rfdata[31:0]	00000007
seimm[31:0]	00001810
op[5:0]	00
func[5:0]	00
isless	U
iseq	1
isload	0
isstore	0
i_type	0
isbranch	0
j_type	0
wrten	1
selrd	0
selwr	0

The screenshot depicts the addition of two register file with the values 7 and 8 resulting in 15(f).