TEST REPORT & KGP-RISC PROCESSOR ANALYSIS

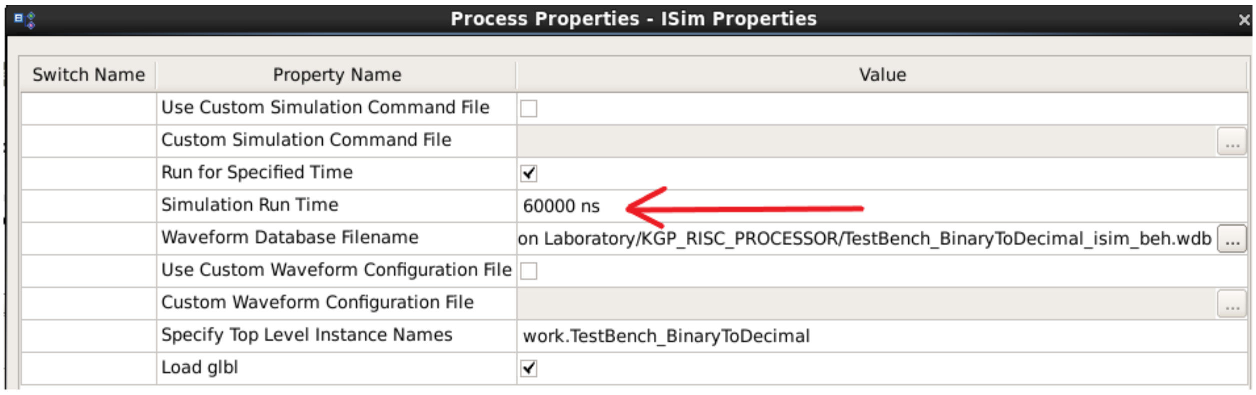
• Algorithm 01 (GCD Computation)

The *ComputeGCD* Algorithm implemented using the ISA of the KGB-RISC Processor is given below:

```
0:     lw $1, 0($0)              // a
1:     lw $2, 4($0)              // b
2:     bz $1, __15__
3:     bz $2, __13__
4:     comp $3, $2       // $3 <- -b
5:     add $3, $1                // $3 <- a - b
6:     bz $3, __13__     // if a == b
7:     bltz __11__       // if a < b
8:     comp $1, $3       // a <- - $3 OR a <- - (a - b)
9:     comp $1, $1       // a <- - a
10:    b __2__
11:    comp $2, $3               // b <- - $3 OR b <- b - a
12:    b __2__
13:    sw $1, 8($0)
14:    b __16__
15:    sw $2, 8($0)
16:    NO OP
```

The above algorithm was then encoded in binary and stored in a .coe file as already explained in README. We simulated this algorithm using our KGP-RISC Processor for different instantiations of the data memories and analysed our results.

Important Note: Since the Algorithm has a loop whose number of iterations depend on the input data, the processor needs to perform computations for enough number of clock cycles. For example, in our TestBench_ComputeGCD.v we have waited for 50000ns for completion of our program and thus



Process Properties - ISim Properties

Switch Name	Property Name	Value
	Use Custom Simulation Command File	☐
	Custom Simulation Command File	
	Run for Specified Time	✓
	Simulation Run Time	60000 ns
	Waveform Database Filename	on Laboratory/KGP_RISC_PROCESSOR/TestBench_BinaryToDecimal_isim_beh.wdb
	Use Custom Waveform Configuration File	☐
	Custom Waveform Configuration File	
	Specify Top Level Instance Names	work.TestBench_BinaryToDecimal
	Load glbl	✓

<u>Simulation time needs to be adjusted to at least 60000ns in the ISIM Simulator.</u>

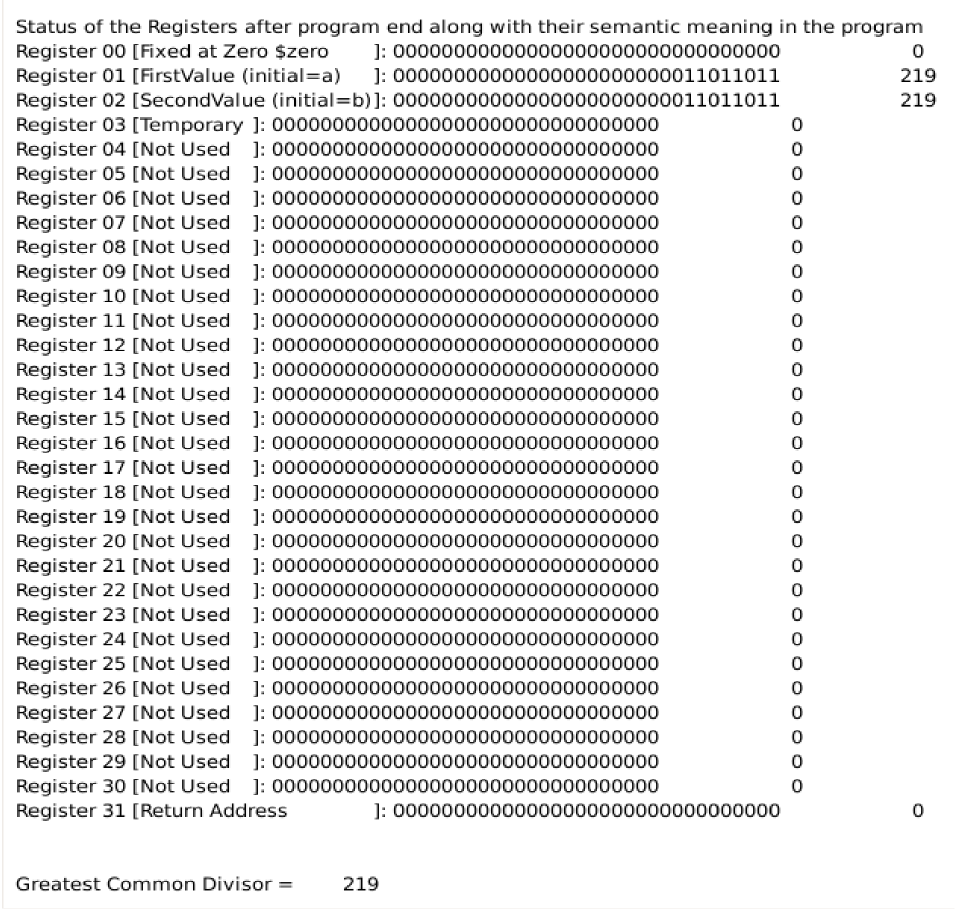
INITIALISE INSTRUCTION MEMORY: Load the instruction memory using the .coe file corresponding to the ComputeGCD Algorithm into the SINGLE_PORT_ROM. (For more details look at the README)

TEST CASE 01

INITIALISE DATA MEMORY: Load the data memory using the .coe file corresponding to __test_case_01__ into the SINGLE_PORT_RAM . (For more details look at the README)

In the first test case data memory is initialised with two values 15549 and 22119. The objective of the program is to load these values from the data memory into the registers, compute their GCD and then store the GCD back into the data memory. As you can see our program has good coverage of all instructions and instruction types.
Below are the snapshots of the results obtained:

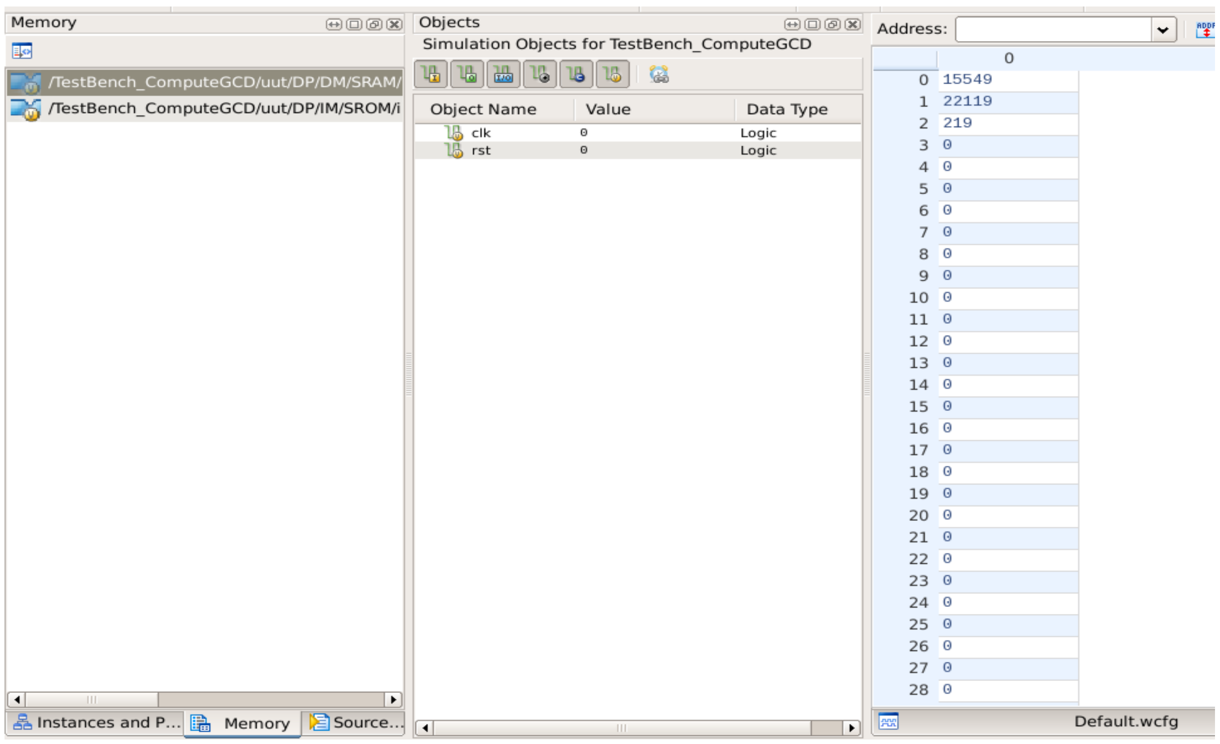
CONSOLE OUTPUT :



```
Status of the Registers after program end along with their semantic meaning in the program
Register 00 [Fixed at Zero $zero       ]: 00000000000000000000000000000000          0
Register 01 [FirstValue (initial=a)    ]: 00000000000000000000000011011011        219
Register 02 [SecondValue (initial=b)]: 00000000000000000000000011011011        219
Register 03 [Temporary  ]: 00000000000000000000000000000000          0
Register 04 [Not Used   ]: 00000000000000000000000000000000          0
Register 05 [Not Used   ]: 00000000000000000000000000000000          0
Register 06 [Not Used   ]: 00000000000000000000000000000000          0
Register 07 [Not Used   ]: 00000000000000000000000000000000          0
Register 08 [Not Used   ]: 00000000000000000000000000000000          0
Register 09 [Not Used   ]: 00000000000000000000000000000000          0
Register 10 [Not Used   ]: 00000000000000000000000000000000          0
Register 11 [Not Used   ]: 00000000000000000000000000000000          0
Register 12 [Not Used   ]: 00000000000000000000000000000000          0
Register 13 [Not Used   ]: 00000000000000000000000000000000          0
Register 14 [Not Used   ]: 00000000000000000000000000000000          0
Register 15 [Not Used   ]: 00000000000000000000000000000000          0
Register 16 [Not Used   ]: 00000000000000000000000000000000          0
Register 17 [Not Used   ]: 00000000000000000000000000000000          0
Register 18 [Not Used   ]: 00000000000000000000000000000000          0
Register 19 [Not Used   ]: 00000000000000000000000000000000          0
Register 20 [Not Used   ]: 00000000000000000000000000000000          0
Register 21 [Not Used   ]: 00000000000000000000000000000000          0
Register 22 [Not Used   ]: 00000000000000000000000000000000          0
Register 23 [Not Used   ]: 00000000000000000000000000000000          0
Register 24 [Not Used   ]: 00000000000000000000000000000000          0
Register 25 [Not Used   ]: 00000000000000000000000000000000          0
Register 26 [Not Used   ]: 00000000000000000000000000000000          0
Register 27 [Not Used   ]: 00000000000000000000000000000000          0
Register 28 [Not Used   ]: 00000000000000000000000000000000          0
Register 29 [Not Used   ]: 00000000000000000000000000000000          0
Register 30 [Not Used   ]: 00000000000000000000000000000000          0
Register 31 [Return Address            ]: 00000000000000000000000000000000          0


Greatest Common Divisor =       219
```

DATA MEMORY AFTER PROGRAM IS RUN:


Memory
/TestBench_ComputeGCD/uut/DP/DM/SRAM/
/TestBench_ComputeGCD/uut/DP/IM/SROM/i
Instances and P...
Memory
Source...
Objects
Simulation Objects for TestBench_ComputeGCD
Address:
Default.wcfg

Object Name	Value	Data Type
clk	0	Logic
rst	0	Logic

	0
0	15549
1	22119
2	219
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	0
13	0
14	0
15	0
16	0
17	0
18	0
19	0
20	0
21	0
22	0
23	0
24	0
25	0
26	0
27	0
28	0

As per the __test_case_01__.coe, data memory is initialised with

```
memory_initialization_radix = 10 ;
memory_initialization_vector =
15549 ,
22119 ,
0;
```

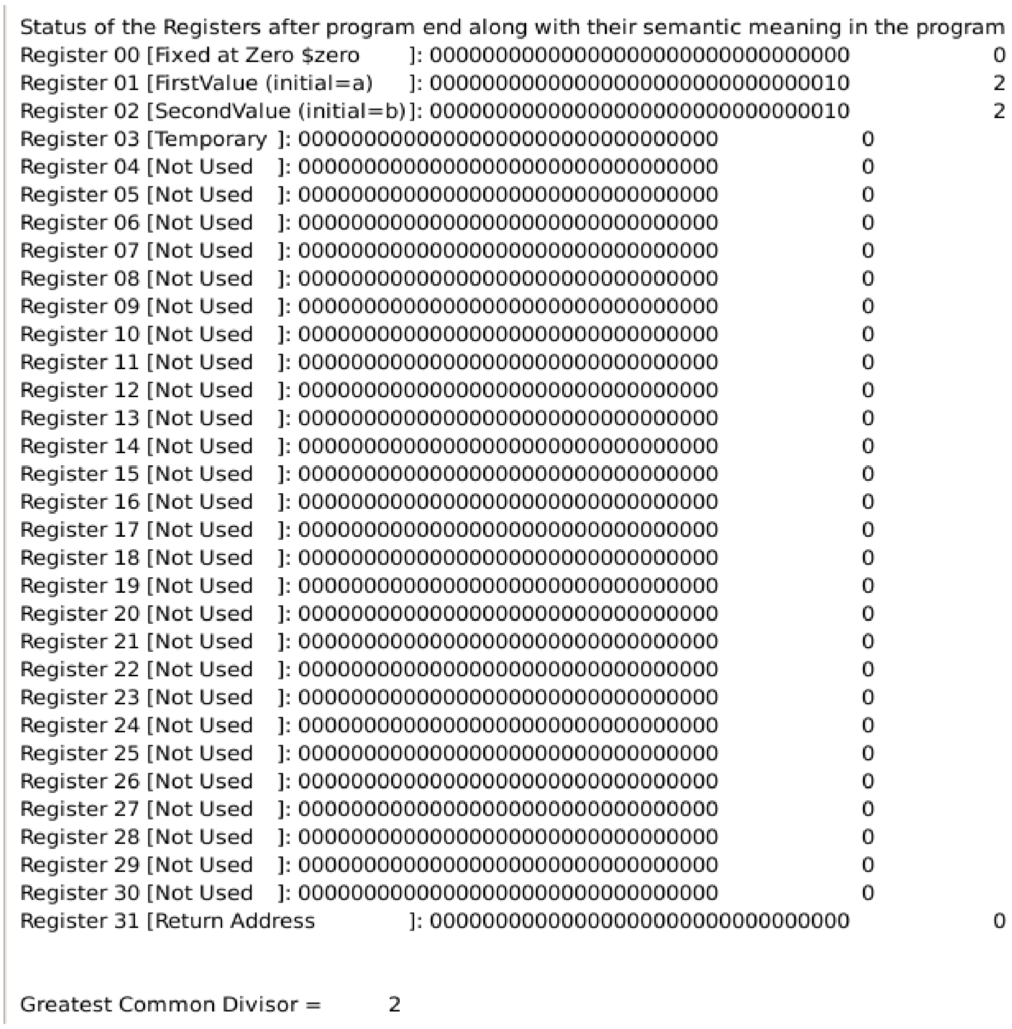
After the program is run, it can be clearly seen that the memory with index 2, has been updated with the GCD of 15549 and 22119 = 219.

TEST CASE 02

INITIALISE DATA MEMORY: Load the data memory using the .coe file corresponding to __test_case_02__ into the SINGLE_PORT_RAM . (For more details look at the README)

In the first test case data memory is initialised with two values 150 and 22. The objective of the program is to load these values from the data memory into the registers, compute their GCD and then store the GCD back into the data memory. As you can see our program has good coverage of all instructions and instruction types.
Below are the snapshots of the results obtained:

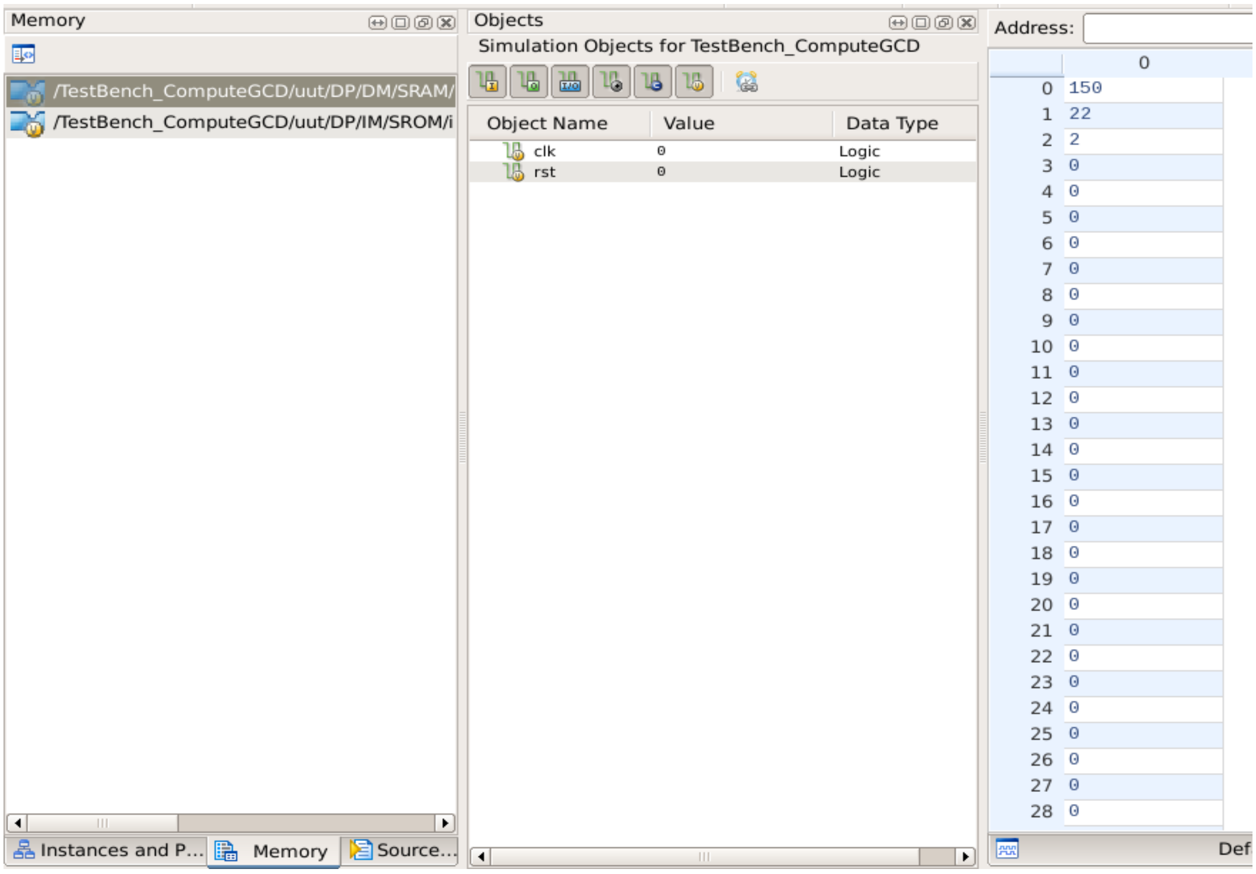
CONSOLE OUTPUT :



```
Status of the Registers after program end along with their semantic meaning in the program
Register 00 [Fixed at Zero $zero        ]: 00000000000000000000000000000000            0
Register 01 [FirstValue (initial=a)     ]: 00000000000000000000000000000010            2
Register 02 [SecondValue (initial=b)]: 00000000000000000000000000000010            2
Register 03 [Temporary  ]: 00000000000000000000000000000000            0
Register 04 [Not Used   ]: 00000000000000000000000000000000            0
Register 05 [Not Used   ]: 00000000000000000000000000000000            0
Register 06 [Not Used   ]: 00000000000000000000000000000000            0
Register 07 [Not Used   ]: 00000000000000000000000000000000            0
Register 08 [Not Used   ]: 00000000000000000000000000000000            0
Register 09 [Not Used   ]: 00000000000000000000000000000000            0
Register 10 [Not Used   ]: 00000000000000000000000000000000            0
Register 11 [Not Used   ]: 00000000000000000000000000000000            0
Register 12 [Not Used   ]: 00000000000000000000000000000000            0
Register 13 [Not Used   ]: 00000000000000000000000000000000            0
Register 14 [Not Used   ]: 00000000000000000000000000000000            0
Register 15 [Not Used   ]: 00000000000000000000000000000000            0
Register 16 [Not Used   ]: 00000000000000000000000000000000            0
Register 17 [Not Used   ]: 00000000000000000000000000000000            0
Register 18 [Not Used   ]: 00000000000000000000000000000000            0
Register 19 [Not Used   ]: 00000000000000000000000000000000            0
Register 20 [Not Used   ]: 00000000000000000000000000000000            0
Register 21 [Not Used   ]: 00000000000000000000000000000000            0
Register 22 [Not Used   ]: 00000000000000000000000000000000            0
Register 23 [Not Used   ]: 00000000000000000000000000000000            0
Register 24 [Not Used   ]: 00000000000000000000000000000000            0
Register 25 [Not Used   ]: 00000000000000000000000000000000            0
Register 26 [Not Used   ]: 00000000000000000000000000000000            0
Register 27 [Not Used   ]: 00000000000000000000000000000000            0
Register 28 [Not Used   ]: 00000000000000000000000000000000            0
Register 29 [Not Used   ]: 00000000000000000000000000000000            0
Register 30 [Not Used   ]: 00000000000000000000000000000000            0
Register 31 [Return Address             ]: 00000000000000000000000000000000            0


Greatest Common Divisor =        2
```

DATA MEMORY AFTER PROGRAM IS RUN:



Memory
/TestBench_ComputeGCD/uut/DP/DM/SRAM/
/TestBench_ComputeGCD/uut/DP/IM/SROM/i
Instances and P...
Memory
Source...
Objects
Simulation Objects for TestBench_ComputeGCD
Address:
Def

Object Name	Value	Data Type
clk	0	Logic
rst	0	Logic

	0
0	150
1	22
2	2
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	0
13	0
14	0
15	0
16	0
17	0
18	0
19	0
20	0
21	0
22	0
23	0
24	0
25	0
26	0
27	0
28	0

As per the __test_case_02__.coe, data memory is initialised with

```
memory_initialization_radix = 10 ;
memory_initialization_vector =
150 ,
22 ,
0;
```

After the program is run, it can be clearly seen that the memory with index 2, has been updated with the GCD of 150 and 22 = 2.

TEST CASE 03

INITIALISE DATA MEMORY: Load the data memory using the .coe file corresponding to __test_case_03__ into the SINGLE_PORT_RAM . (For more details look at the README)

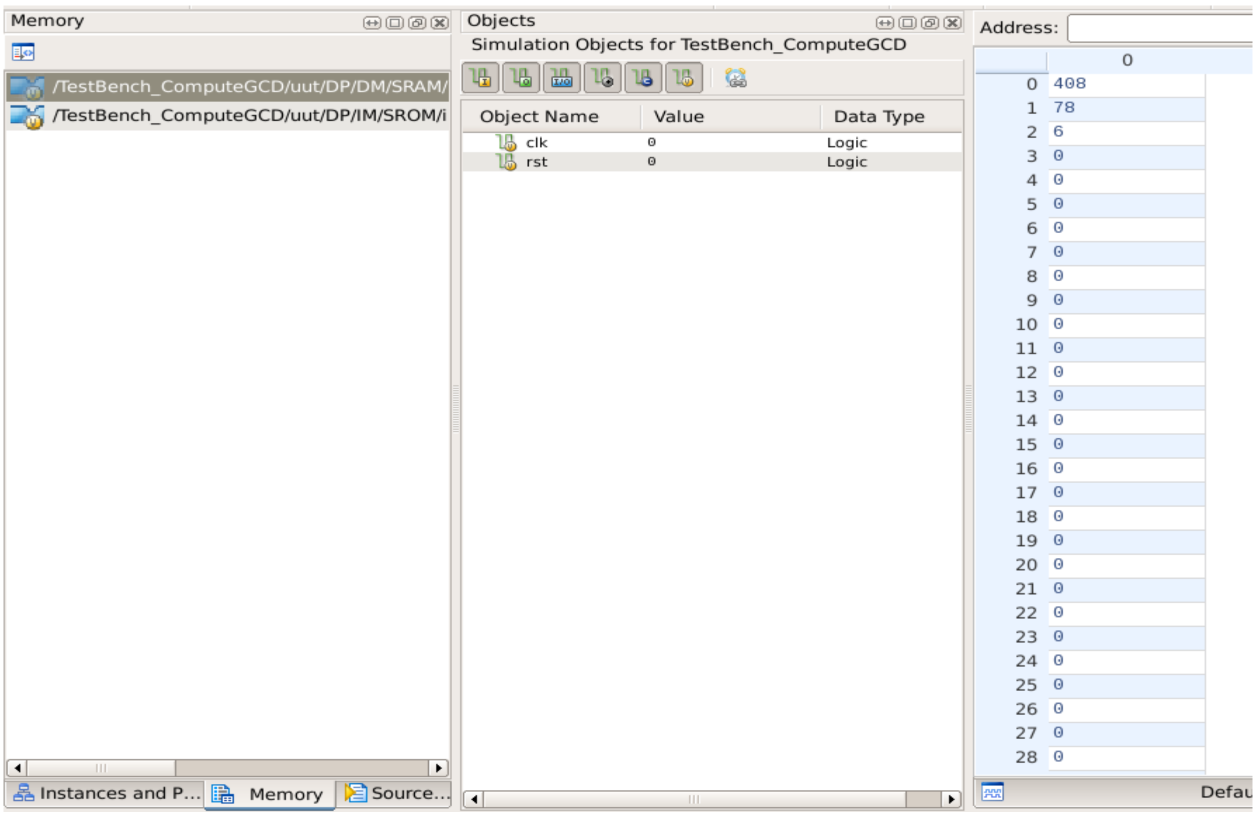
In the first test case data memory is initialised with two values 408 and 78. The objective of the program is to load these values from the data memory into the registers, compute their GCD and then store the GCD back into the data memory. As you can see our program has good coverage of all instructions and instruction types.
Below are the snapshots of the results obtained:

CONSOLE OUTPUT :

```
Status of the Registers after program end along with their semantic meaning in the program
Register 00 [Fixed at Zero $zero       ]: 00000000000000000000000000000000           0
Register 01 [FirstValue (initial=a)    ]: 00000000000000000000000000000110           6
Register 02 [SecondValue (initial=b)]: 00000000000000000000000000000110           6
Register 03 [Temporary ]: 00000000000000000000000000000000           0
Register 04 [Not Used   ]: 00000000000000000000000000000000           0
Register 05 [Not Used   ]: 00000000000000000000000000000000           0
Register 06 [Not Used   ]: 00000000000000000000000000000000           0
Register 07 [Not Used   ]: 00000000000000000000000000000000           0
Register 08 [Not Used   ]: 00000000000000000000000000000000           0
Register 09 [Not Used   ]: 00000000000000000000000000000000           0
Register 10 [Not Used   ]: 00000000000000000000000000000000           0
Register 11 [Not Used   ]: 00000000000000000000000000000000           0
Register 12 [Not Used   ]: 00000000000000000000000000000000           0
Register 13 [Not Used   ]: 00000000000000000000000000000000           0
Register 14 [Not Used   ]: 00000000000000000000000000000000           0
Register 15 [Not Used   ]: 00000000000000000000000000000000           0
Register 16 [Not Used   ]: 00000000000000000000000000000000           0
Register 17 [Not Used   ]: 00000000000000000000000000000000           0
Register 18 [Not Used   ]: 00000000000000000000000000000000           0
Register 19 [Not Used   ]: 00000000000000000000000000000000           0
Register 20 [Not Used   ]: 00000000000000000000000000000000           0
Register 21 [Not Used   ]: 00000000000000000000000000000000           0
Register 22 [Not Used   ]: 00000000000000000000000000000000           0
Register 23 [Not Used   ]: 00000000000000000000000000000000           0
Register 24 [Not Used   ]: 00000000000000000000000000000000           0
Register 25 [Not Used   ]: 00000000000000000000000000000000           0
Register 26 [Not Used   ]: 00000000000000000000000000000000           0
Register 27 [Not Used   ]: 00000000000000000000000000000000           0
Register 28 [Not Used   ]: 00000000000000000000000000000000           0
Register 29 [Not Used   ]: 00000000000000000000000000000000           0
Register 30 [Not Used   ]: 00000000000000000000000000000000           0
Register 31 [Return Address            ]: 00000000000000000000000000000000           0


Greatest Common Divisor =         6
```

DATA MEMORY AFTER PROGRAM IS RUN:



Memory

/TestBench_ComputeGCD/uut/DP/DM/SRAM/

/TestBench_ComputeGCD/uut/DP/IM/SROM/i

Objects

Simulation Objects for TestBench_ComputeGCD

Object Name	Value	Data Type
clk	0	Logic
rst	0	Logic

Address:

	0
0	408
1	78
2	6
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	0
13	0
14	0
15	0
16	0
17	0
18	0
19	0
20	0
21	0
22	0
23	0
24	0
25	0
26	0
27	0
28	0

As per the __test_case_03__.coe, data memory is initialised with

```
memory_initialization_radix = 10 ;
memory_initialization_vector =
408 ,
78 ,
0;
```

After the program is run, it can be clearly seen that the memory with index 2, has been updated with the GCD of 408 and 78 = 6.

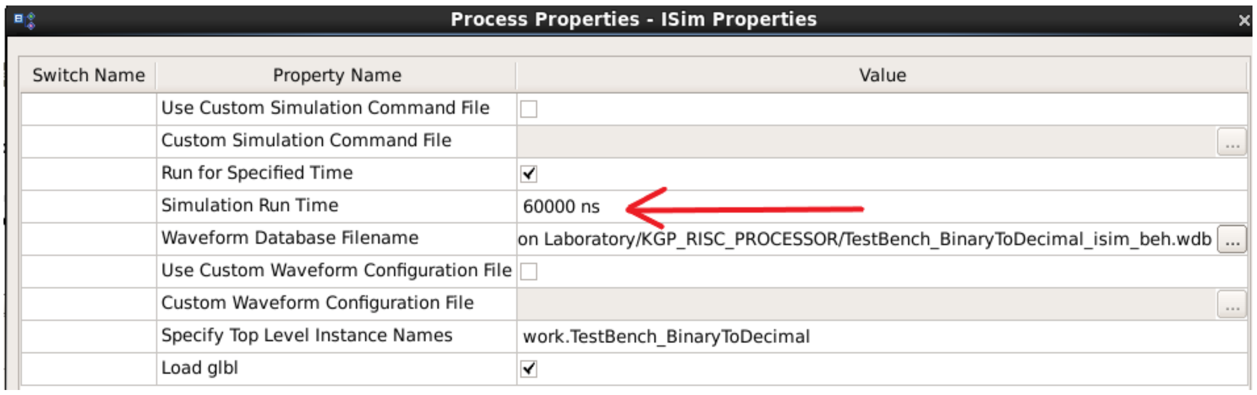
• Algorithm 02 (Binary to Decimal Conversion)

The BinaryToDecimalConverter Algorithm implemented using the ISA of the KGP-RISC Processor is given below:

```
0:  lw $2, 0($1)          // Load the number of bits in the binary number (n)
1:  bltz $2, exit(11)
2:  bz $2, exit(10)
3:  lw $3, 4($1)          // Initialise decimal number to first binary number, because
n is at least 1
4:  addi $2, -1
5:  bz $2, print(6)       // If n-1==0 then print
6:  shll $3, 1            // Shift by 1 bit to the left
7:  lw $4, 8($1)          // Get the next binary digit and add
8:   add $3, $4           // Update decimal number
9:   addi $1, 4           // Increment Stack Pointer
10: addi $2, -1           // Decrement iterator (in this case n)
11: b __5__
12: print: sw $3, 8($1) // Store the decimal number in memory
13: exit: NO OP
```

The above algorithm was then encoded in binary and stored in a .coe file as already explained in README. We simulated this algorithm using our KGP-RISC Processor for different instantiations of the data memories and analysed our results.

Important Note: Since the Algorithm has a loop whose number of iterations depend on the input data, the processor needs to perform computations for enough number of clock cycles. For example, in our TestBench_BinaryToDecimal.v we have waited for 50000ns for completion of our program and thus


Process Properties - ISim Properties

Switch Name	Property Name	Value
	Use Custom Simulation Command File	☐
	Custom Simulation Command File	
	Run for Specified Time	✓
	Simulation Run Time	60000 ns
	Waveform Database Filename	on Laboratory/KGP_RISC_PROCESSOR/TestBench_BinaryToDecimal_isim_beh.wdb
	Use Custom Waveform Configuration File	☐
	Custom Waveform Configuration File	
	Specify Top Level Instance Names	work.TestBench_BinaryToDecimal
	Load glbl	✓

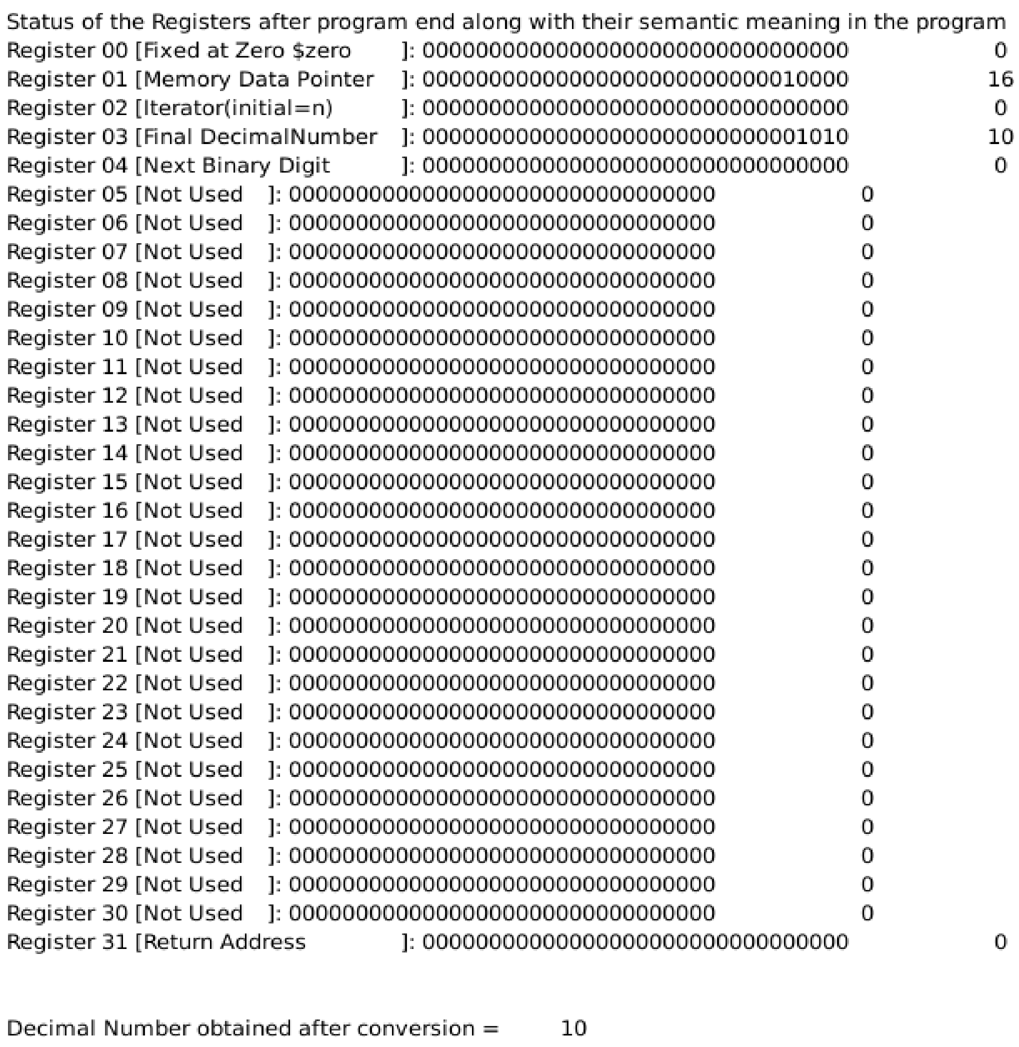
Simulation time needs to be adjusted to at least 60000ns in the ISIM Simulator.
INITIALISE INSTRUCTION MEMORY: Load the instruction memory using the .coe file corresponding to the BinaryToDecimal Algorithm into the SINGLE_PORT_ROM. (For more details look at the README)

TEST CASE 01

INITIALISE DATA MEMORY: Load the data memory using the .coe file corresponding to __test_case_01__ into the SINGLE_PORT_RAM . (For more details look at the README)

In the first test case data memory is initialised with 6 values 5, 0, 1, 0, 1 and 0. The objective of the program is to load these values from the data memory into the registers, convert the binary number to a decimal number and then store the decimal number back into the data memory. As you can see our program has good coverage of all instructions and instruction types. **Note:** The first value 5 tells us the number of binary digits and is followed by the said binary digits.
Below are the snapshots of the results obtained:

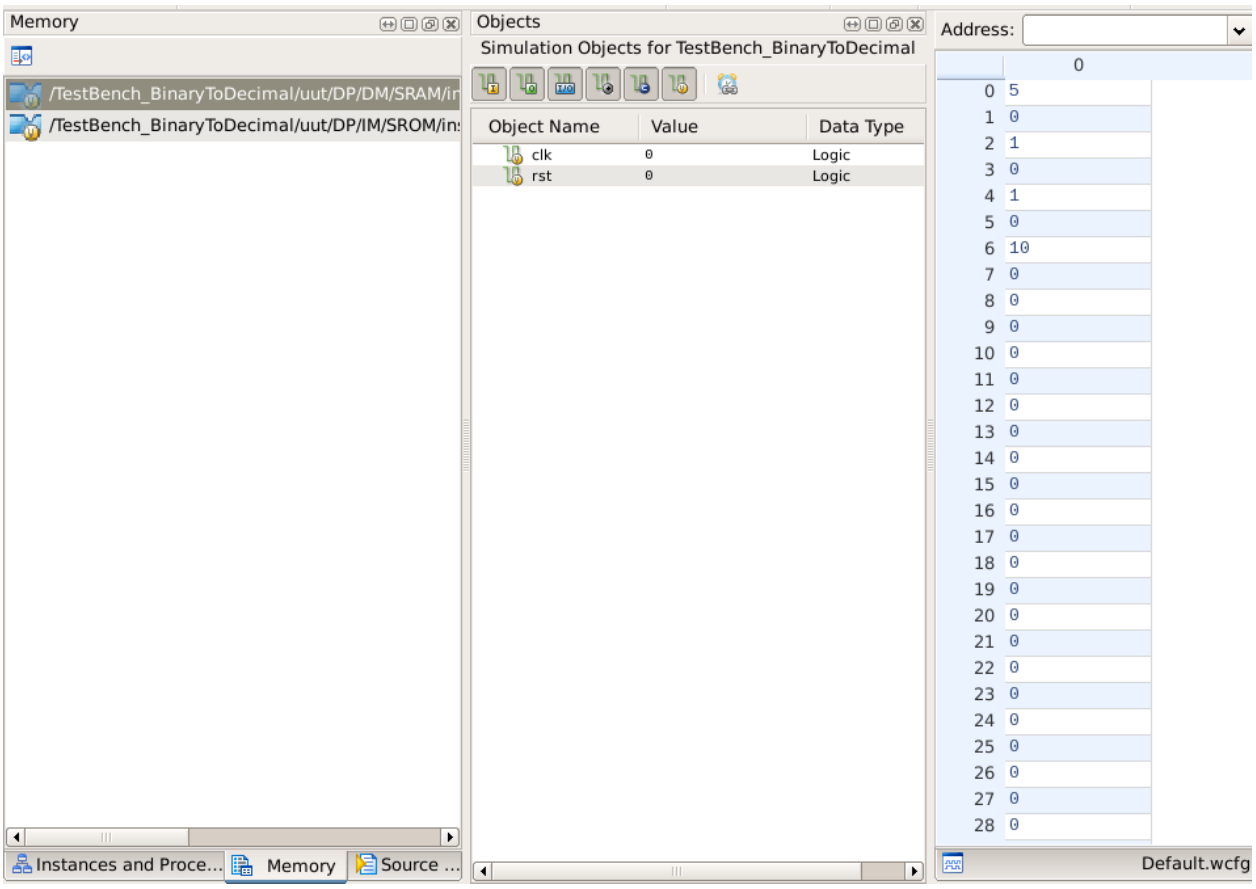
CONSOLE OUTPUT :



```
Status of the Registers after program end along with their semantic meaning in the program
Register 00 [Fixed at Zero $zero      ]: 00000000000000000000000000000000          0
Register 01 [Memory Data Pointer      ]: 00000000000000000000000000010000         16
Register 02 [Iterator(initial=n)      ]: 00000000000000000000000000000000          0
Register 03 [Final DecimalNumber      ]: 00000000000000000000000000001010         10
Register 04 [Next Binary Digit        ]: 00000000000000000000000000000000          0
Register 05 [Not Used    ]: 00000000000000000000000000000000          0
Register 06 [Not Used    ]: 00000000000000000000000000000000          0
Register 07 [Not Used    ]: 00000000000000000000000000000000          0
Register 08 [Not Used    ]: 00000000000000000000000000000000          0
Register 09 [Not Used    ]: 00000000000000000000000000000000          0
Register 10 [Not Used    ]: 00000000000000000000000000000000          0
Register 11 [Not Used    ]: 00000000000000000000000000000000          0
Register 12 [Not Used    ]: 00000000000000000000000000000000          0
Register 13 [Not Used    ]: 00000000000000000000000000000000          0
Register 14 [Not Used    ]: 00000000000000000000000000000000          0
Register 15 [Not Used    ]: 00000000000000000000000000000000          0
Register 16 [Not Used    ]: 00000000000000000000000000000000          0
Register 17 [Not Used    ]: 00000000000000000000000000000000          0
Register 18 [Not Used    ]: 00000000000000000000000000000000          0
Register 19 [Not Used    ]: 00000000000000000000000000000000          0
Register 20 [Not Used    ]: 00000000000000000000000000000000          0
Register 21 [Not Used    ]: 00000000000000000000000000000000          0
Register 22 [Not Used    ]: 00000000000000000000000000000000          0
Register 23 [Not Used    ]: 00000000000000000000000000000000          0
Register 24 [Not Used    ]: 00000000000000000000000000000000          0
Register 25 [Not Used    ]: 00000000000000000000000000000000          0
Register 26 [Not Used    ]: 00000000000000000000000000000000          0
Register 27 [Not Used    ]: 00000000000000000000000000000000          0
Register 28 [Not Used    ]: 00000000000000000000000000000000          0
Register 29 [Not Used    ]: 00000000000000000000000000000000          0
Register 30 [Not Used    ]: 00000000000000000000000000000000          0
Register 31 [Return Address           ]: 00000000000000000000000000000000          0


Decimal Number obtained after conversion =       10
```

DATA MEMORY AFTER PROGRAM IS RUN:


Memory

/TestBench_BinaryToDecimal/uut/DP/DM/SRAM/in

/TestBench_BinaryToDecimal/uut/DP/IM/SROM/in

Objects

Simulation Objects for TestBench_BinaryToDecimal

Object Name	Value	Data Type
clk	0	Logic
rst	0	Logic

Address:

	0
0	5
1	0
2	1
3	0
4	1
5	0
6	10
7	0
8	0
9	0
10	0
11	0
12	0
13	0
14	0
15	0
16	0
17	0
18	0
19	0
20	0
21	0
22	0
23	0
24	0
25	0
26	0
27	0
28	0

As per the __test_case_01__.coe, data memory is initialised with

```
memory_initialization_radix = 10 ;
memory_initialization_vector =
5 ,
0 ,
1 ,
0 ,
1 ,
0 ,
0;
```

After the program is run, it can be clearly seen that the memory with index 6, has been updated with the decimal number depicted by 01010 = 10.

TEST CASE 02

INITIALISE DATA MEMORY: Load the data memory using the .coe file corresponding to __test_case_02__ into the SINGLE_PORT_RAM . (For more details look at the README)

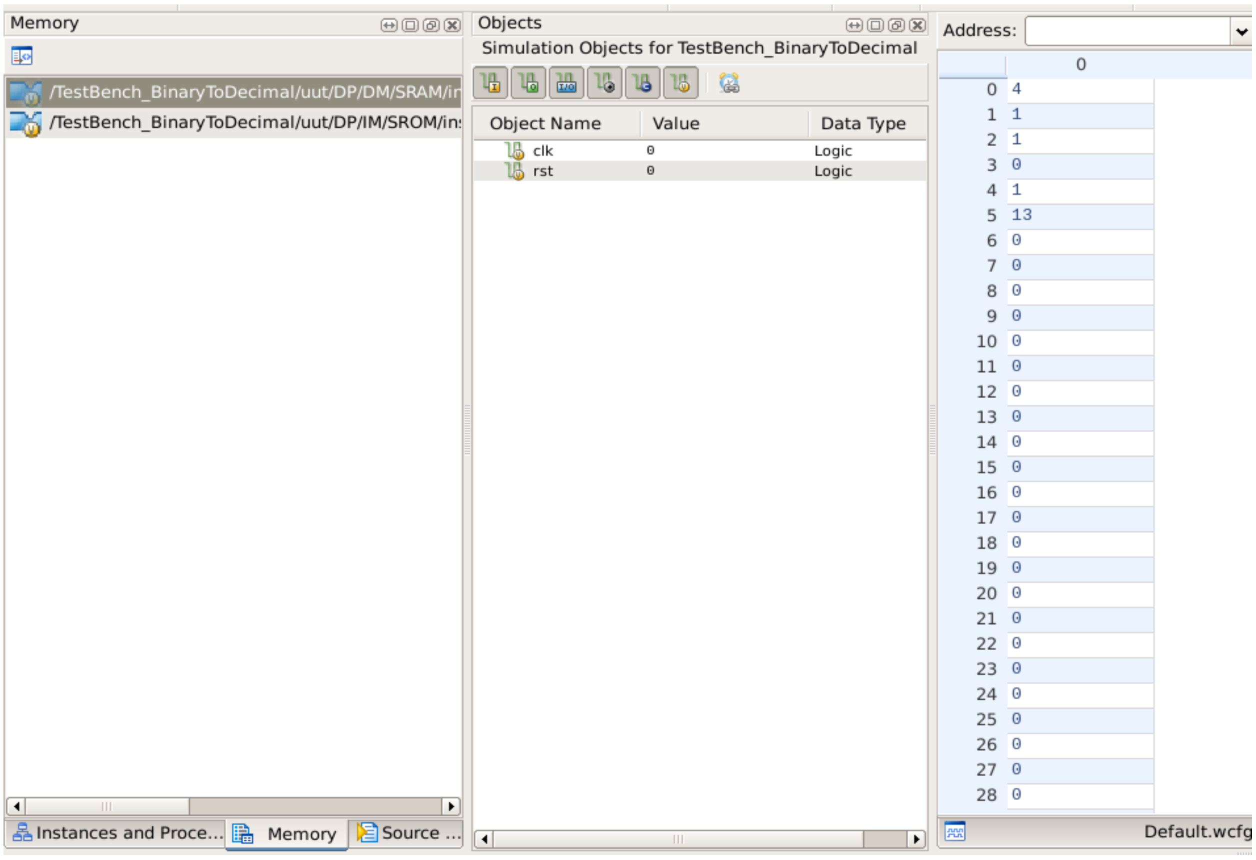
In the first test case data memory is initialised with 5 values 4, 1, 1, 0, and 1. The objective of the program is to load these values from the data memory into the registers, convert the binary number to a decimal number and then store the decimal number back into the data memory. As you can see our program has good coverage of all instructions and instruction types. **Note:** The first value 4 tells us the number of binary digits and is followed by the said binary digits.
Below are the snapshots of the results obtained:

CONSOLE OUTPUT :

```
Status of the Registers after program end along with their semantic meaning in the program
Register 00 [Fixed at Zero $zero      ]: 00000000000000000000000000000000          0
Register 01 [Memory Data Pointer      ]: 00000000000000000000000000001100         12
Register 02 [Iterator(initial=n)      ]: 00000000000000000000000000000000          0
Register 03 [Final DecimalNumber      ]: 00000000000000000000000000001101         13
Register 04 [Next Binary Digit        ]: 00000000000000000000000000000001          1
Register 05 [Not Used    ]: 00000000000000000000000000000000          0
Register 06 [Not Used    ]: 00000000000000000000000000000000          0
Register 07 [Not Used    ]: 00000000000000000000000000000000          0
Register 08 [Not Used    ]: 00000000000000000000000000000000          0
Register 09 [Not Used    ]: 00000000000000000000000000000000          0
Register 10 [Not Used    ]: 00000000000000000000000000000000          0
Register 11 [Not Used    ]: 00000000000000000000000000000000          0
Register 12 [Not Used    ]: 00000000000000000000000000000000          0
Register 13 [Not Used    ]: 00000000000000000000000000000000          0
Register 14 [Not Used    ]: 00000000000000000000000000000000          0
Register 15 [Not Used    ]: 00000000000000000000000000000000          0
Register 16 [Not Used    ]: 00000000000000000000000000000000          0
Register 17 [Not Used    ]: 00000000000000000000000000000000          0
Register 18 [Not Used    ]: 00000000000000000000000000000000          0
Register 19 [Not Used    ]: 00000000000000000000000000000000          0
Register 20 [Not Used    ]: 00000000000000000000000000000000          0
Register 21 [Not Used    ]: 00000000000000000000000000000000          0
Register 22 [Not Used    ]: 00000000000000000000000000000000          0
Register 23 [Not Used    ]: 00000000000000000000000000000000          0
Register 24 [Not Used    ]: 00000000000000000000000000000000          0
Register 25 [Not Used    ]: 00000000000000000000000000000000          0
Register 26 [Not Used    ]: 00000000000000000000000000000000          0
Register 27 [Not Used    ]: 00000000000000000000000000000000          0
Register 28 [Not Used    ]: 00000000000000000000000000000000          0
Register 29 [Not Used    ]: 00000000000000000000000000000000          0
Register 30 [Not Used    ]: 00000000000000000000000000000000          0
Register 31 [Return Address           ]: 00000000000000000000000000000000          0


Decimal Number obtained after conversion =       13
```

DATA MEMORY AFTER PROGRAM IS RUN:



/TestBench_BinaryToDecimal/uut/DP/DM/SRAM/in
/TestBench_BinaryToDecimal/uut/DP/IM/SROM/in:

Simulation Objects for TestBench_BinaryToDecimal

Object Name	Value	Data Type
clk	0	Logic
rst	0	Logic

	0
0	4
1	1
2	1
3	0
4	1
5	13
6	0
7	0
8	0
9	0
10	0
11	0
12	0
13	0
14	0
15	0
16	0
17	0
18	0
19	0
20	0
21	0
22	0
23	0
24	0
25	0
26	0
27	0
28	0

As per the __test_case_02__.coe, data memory is initialised with

```
memory_initialization_radix = 10 ;
memory_initialization_vector =
4 ,
1 ,
1 ,
0 ,
1 ,
0
;
```

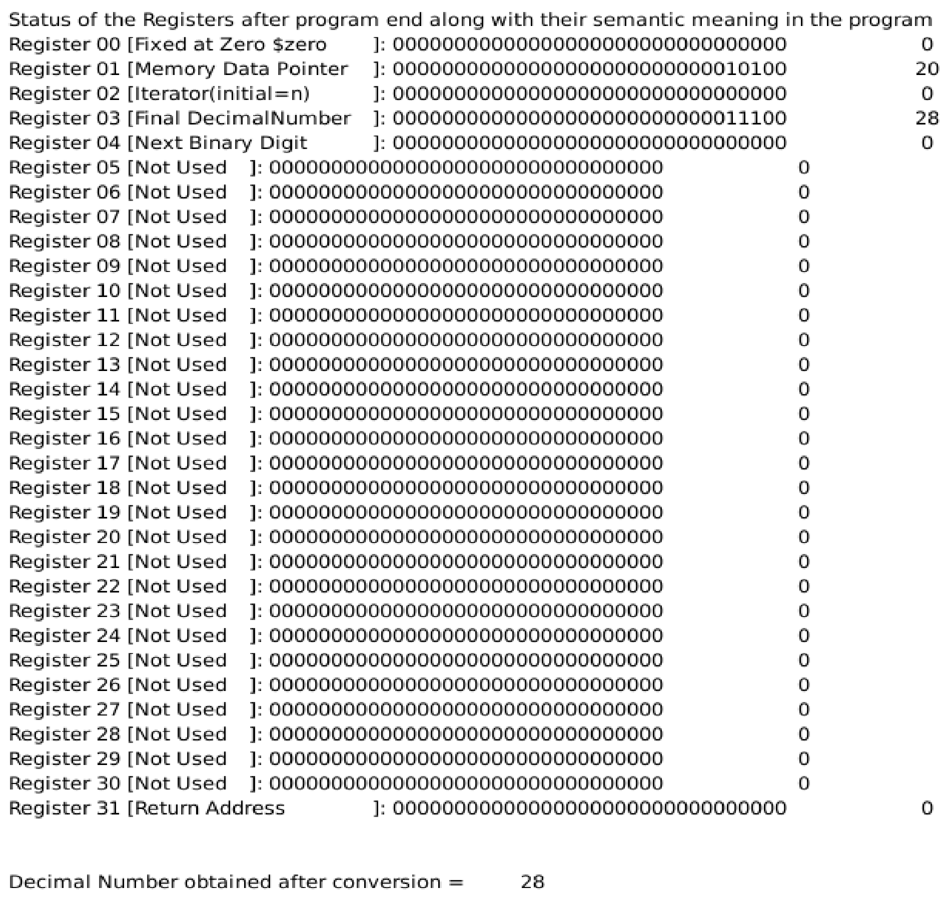
After the program is run, it can be clearly seen that the memory with index 5, has been updated with the decimal number depicted by 1101 = 13.

TEST CASE 03

INITIALISE DATA MEMORY: Load the data memory using the .coe file corresponding to __test_case_03__ into the SINGLE_PORT_RAM . (For more details look at the README)

In the first test case data memory is initialised with 7 values 6, 0, 1, 1, 1, 0, and 0. The objective of the program is to load these values from the data memory into the registers, convert the binary number to a decimal number and then store the decimal number back into the data memory. As you can see our program has good coverage of all instructions and instruction types. **Note:** The first value 6 tells us the number of binary digits and is followed by the said binary digits.
Below are the snapshots of the results obtained:

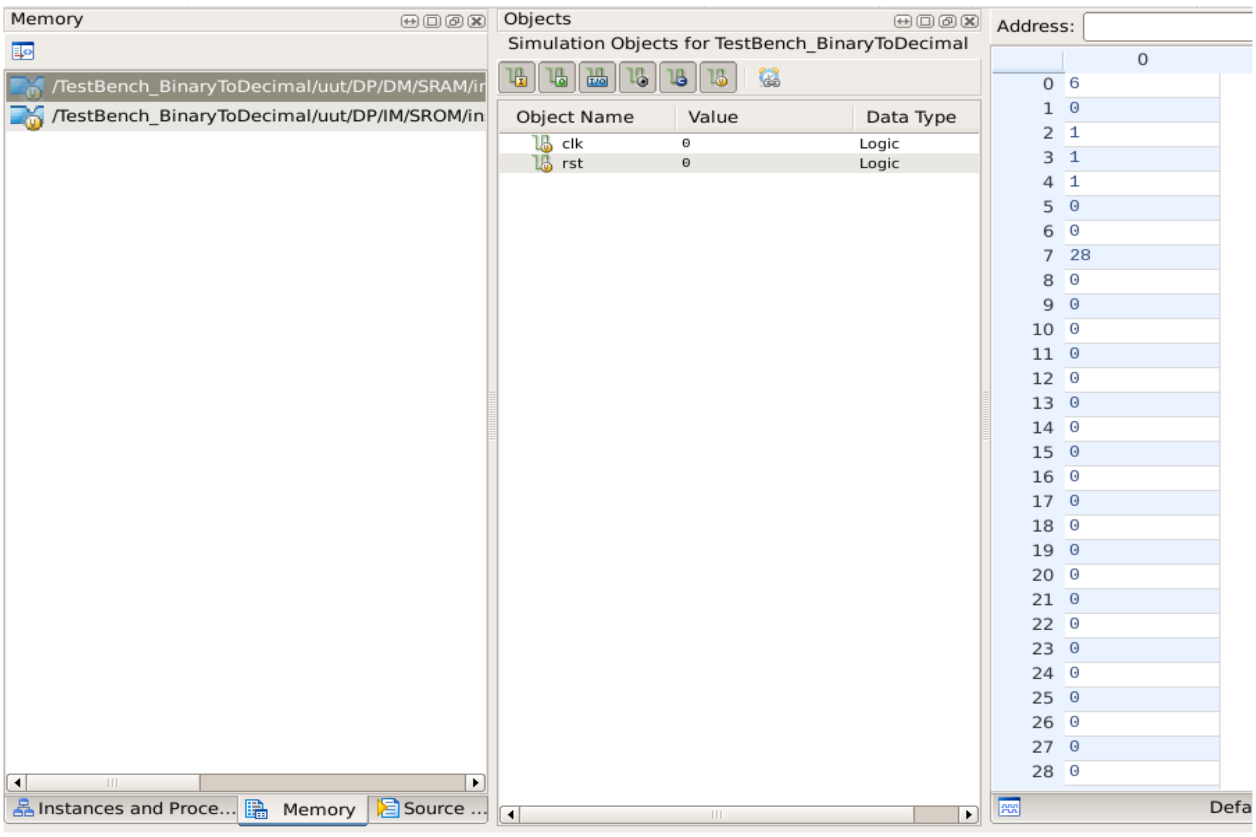
CONSOLE OUTPUT :



```
Status of the Registers after program end along with their semantic meaning in the program
Register 00 [Fixed at Zero $zero     ]: 00000000000000000000000000000000          0
Register 01 [Memory Data Pointer     ]: 00000000000000000000000000010100          20
Register 02 [Iterator(initial=n)     ]: 00000000000000000000000000000000          0
Register 03 [Final DecimalNumber     ]: 00000000000000000000000000011100          28
Register 04 [Next Binary Digit       ]: 00000000000000000000000000000000          0
Register 05 [Not Used   ]: 00000000000000000000000000000000          0
Register 06 [Not Used   ]: 00000000000000000000000000000000          0
Register 07 [Not Used   ]: 00000000000000000000000000000000          0
Register 08 [Not Used   ]: 00000000000000000000000000000000          0
Register 09 [Not Used   ]: 00000000000000000000000000000000          0
Register 10 [Not Used   ]: 00000000000000000000000000000000          0
Register 11 [Not Used   ]: 00000000000000000000000000000000          0
Register 12 [Not Used   ]: 00000000000000000000000000000000          0
Register 13 [Not Used   ]: 00000000000000000000000000000000          0
Register 14 [Not Used   ]: 00000000000000000000000000000000          0
Register 15 [Not Used   ]: 00000000000000000000000000000000          0
Register 16 [Not Used   ]: 00000000000000000000000000000000          0
Register 17 [Not Used   ]: 00000000000000000000000000000000          0
Register 18 [Not Used   ]: 00000000000000000000000000000000          0
Register 19 [Not Used   ]: 00000000000000000000000000000000          0
Register 20 [Not Used   ]: 00000000000000000000000000000000          0
Register 21 [Not Used   ]: 00000000000000000000000000000000          0
Register 22 [Not Used   ]: 00000000000000000000000000000000          0
Register 23 [Not Used   ]: 00000000000000000000000000000000          0
Register 24 [Not Used   ]: 00000000000000000000000000000000          0
Register 25 [Not Used   ]: 00000000000000000000000000000000          0
Register 26 [Not Used   ]: 00000000000000000000000000000000          0
Register 27 [Not Used   ]: 00000000000000000000000000000000          0
Register 28 [Not Used   ]: 00000000000000000000000000000000          0
Register 29 [Not Used   ]: 00000000000000000000000000000000          0
Register 30 [Not Used   ]: 00000000000000000000000000000000          0
Register 31 [Return Address          ]: 00000000000000000000000000000000          0

Decimal Number obtained after conversion =        28
```

DATA MEMORY AFTER PROGRAM IS RUN:


Memory

/TestBench_BinaryToDecimal/uut/DP/DM/SRAM/ir
/TestBench_BinaryToDecimal/uut/DP/IM/SROM/in

Objects
Simulation Objects for TestBench_BinaryToDecimal

Object Name	Value	Data Type
clk	0	Logic
rst	0	Logic

Address:

	0
0	6
1	0
2	1
3	1
4	1
5	0
6	0
7	28
8	0
9	0
10	0
11	0
12	0
13	0
14	0
15	0
16	0
17	0
18	0
19	0
20	0
21	0
22	0
23	0
24	0
25	0
26	0
27	0
28	0

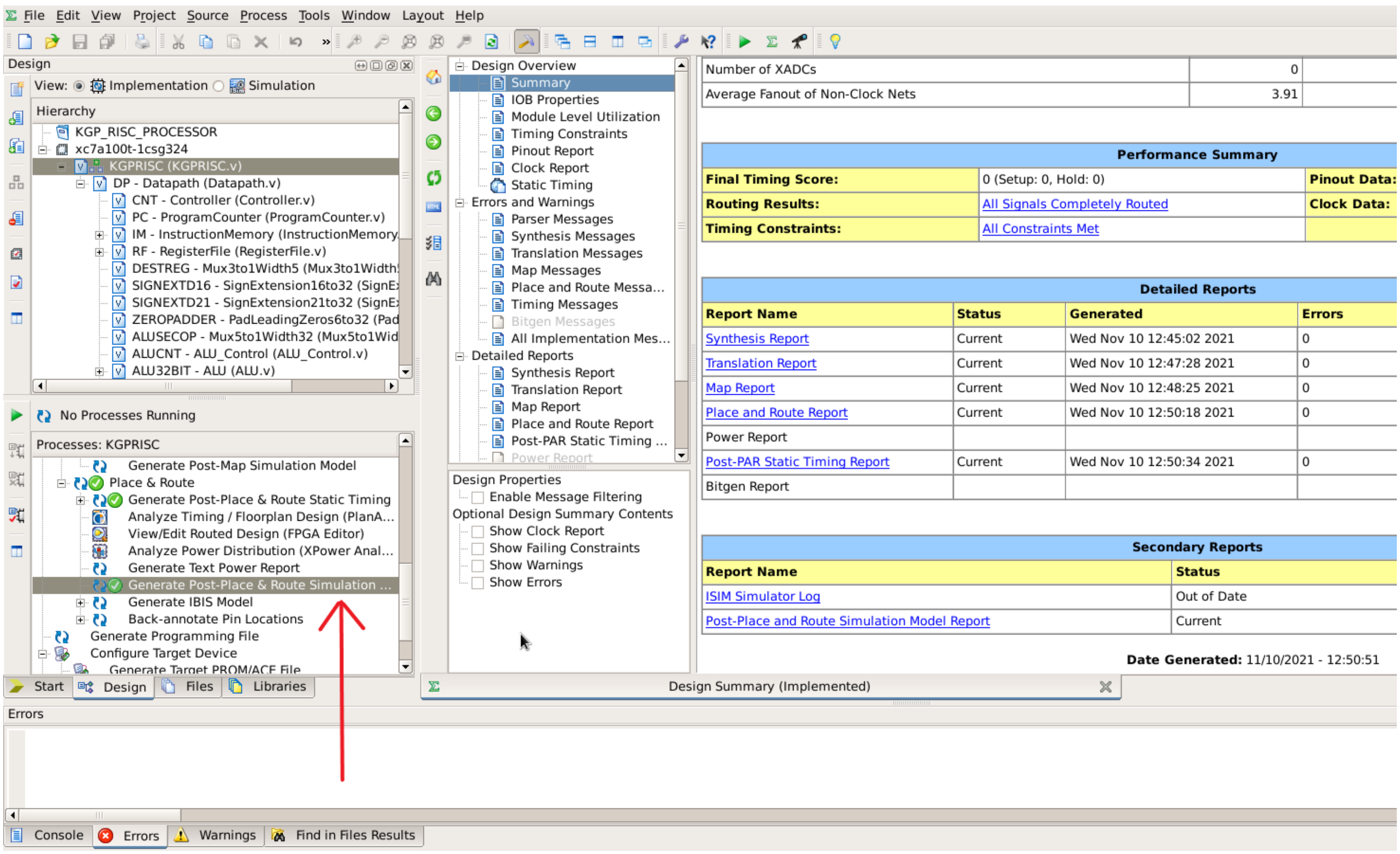
As per the __test_case_03__.coe, data memory is initialised with

```
memory_initialization_radix = 10 ;
memory_initialization_vector =
6 ,
0 ,
1 ,
1 ,
1 ,
0 ,
0 ,
0
;
```

After the program is run, it can be clearly seen that the memory with index 7, has been updated with the decimal number depicted by 011100 = 28.

SUCCESSFULLY GENERATED POST-PLACE & ROUTE SIMULATION MODEL

We have successfully generated the post-place and route simulation model and thus successfully completed the construction of the KGP-RISC Processor.


File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Start Design Files Libraries
Design Summary (Implemented)
Errors
Console Errors Warnings Find in Files Results

Hierarchy
- KGP_RISC_PROCESSOR
- xc7a100t-1csg324
  - KGPRISC (KGPRISC.v)
    - DP - Datapath (Datapath.v)
      - CNT - Controller (Controller.v)
      - PC - ProgramCounter (ProgramCounter.v)
      - IM - InstructionMemory (InstructionMemory
      - RF - RegisterFile (RegisterFile.v)
      - DESTREG - Mux3to1Width5 (Mux3to1Width5
      - SIGNEXTD16 - SignExtension16to32 (SignEx
      - SIGNEXTD21 - SignExtension21to32 (SignEx
      - ZEROPADDER - PadLeadingZeros6to32 (Pad
      - ALUSECOP - Mux5to1Width32 (Mux5to1Wid
      - ALUCNT - ALU_Control (ALU_Control.v)
      - ALU32BIT - ALU (ALU.v)

No Processes Running

Processes: KGPRISC
- Generate Post-Map Simulation Model
- Place & Route
  - Generate Post-Place & Route Static Timing
  - Analyze Timing / Floorplan Design (PlanA…
  - View/Edit Routed Design (FPGA Editor)
  - Analyze Power Distribution (XPower Anal…
  - Generate Text Power Report
  - Generate Post-Place & Route Simulation …
  - Generate IBIS Model
  - Back-annotate Pin Locations
- Generate Programming File
- Configure Target Device
  - Generate Target PROM/ACF File

- Design Overview
  - Summary
  - IOB Properties
  - Module Level Utilization
  - Timing Constraints
  - Pinout Report
  - Clock Report
  - Static Timing
- Errors and Warnings
  - Parser Messages
  - Synthesis Messages
  - Translation Messages
  - Map Messages
  - Place and Route Messa…
  - Timing Messages
  - Bitgen Messages
  - All Implementation Mes…
- Detailed Reports
  - Synthesis Report
  - Translation Report
  - Map Report
  - Place and Route Report
  - Post-PAR Static Timing …
  - Power Report

Design Properties
- Enable Message Filtering

Optional Design Summary Contents
- Show Clock Report
- Show Failing Constraints
- Show Warnings
- Show Errors

Number of XADCs	0
Average Fanout of Non-Clock Nets	3.91

Performance Summary		
Final Timing Score:	0 (Setup: 0, Hold: 0)	Pinout Data:
Routing Results:	All Signals Completely Routed	Clock Data:
Timing Constraints:	All Constraints Met	

Detailed Reports

Report Name	Status	Generated	Errors
Synthesis Report	Current	Wed Nov 10 12:45:02 2021	0
Translation Report	Current	Wed Nov 10 12:47:28 2021	0
Map Report	Current	Wed Nov 10 12:48:25 2021	0
Place and Route Report	Current	Wed Nov 10 12:50:18 2021	0
Power Report			
Post-PAR Static Timing Report	Current	Wed Nov 10 12:50:34 2021	0
Bitgen Report			

Secondary Reports

Report Name	Status
ISIM Simulator Log	Out of Date
Post-Place and Route Simulation Model Report	Current

Date Generated: 11/10/2021 - 12:50:51