Computer Organization and Architecture Laboratory

KGP-RISC

Execution Results

Group 30

VANSHITA GARG - 19CS10064

ASHUTOSH KUMAR SINGH - 19CS30008

Assumptions and Important Points

- All addresses specified in all kinds of branch or jump instructions are treated as absolute addresses (with respect to the instruction memory).
- To accommodate the delay due to the BRAM modules of instruction memory and data memory, the data memory operates on the negative edge of the clock, while all other modules operate on the positive edge.
- The time period of the clock is fixed as 20ns.

Executing Linear Search Algorithm

Assembly Language Code

```
main:
    xor $20, $20        # base address of array = 4 ($20)
    addi $20, 4
    xor $21, $21
    addi $21, 10        # $21 = n = 10
    xor $8, $8          # $8 = i = 0
    lw $9, -4($20)      # $9 = key

fori:
    xor $10, $10
    add $10, $8
    comp $11, $21
    add $10, $11        # $10 = i - n
    bz $10, notFound    # if i == n, jump to notFound

    xor $12, $12
    add $12, $8
    shll $12, 2         # 4 * i
    add $12, $20        # arr + 4 * i
    lw $13, 0($12)      # $13 = arr[i]

    xor $14, $14
    comp $15, $13
    add $14, $9
    add $14, $15        # key - arr[i]
    bz $14, found       # if key == arr[i], jump to found

    addi $8, 1          # i = i + 1
    b fori

found:
```

```
    xor $19, $19
    add $19, $8         # store the index where key is found in $19
    b exit

notFound:
    xor $19, $19
    addi $19, -1        # if key is not found, store -1 in $19

exit:
    xor $16, $16
    addi $16, 1         # to indicate completion
```

The above assembly language code is written according to the instruction set specification provided to us.
Register Usage:

- $20 stores base address of the array
- $21 stores n
- $8 stores the loop variable i
- $9 stores the key to be searched
- $19 stores the index of the element in the array in the case when the key is found, otherwise stores -1.
- $16 stores 1 to indicate completion

Instruction COE File for Linear Search Program

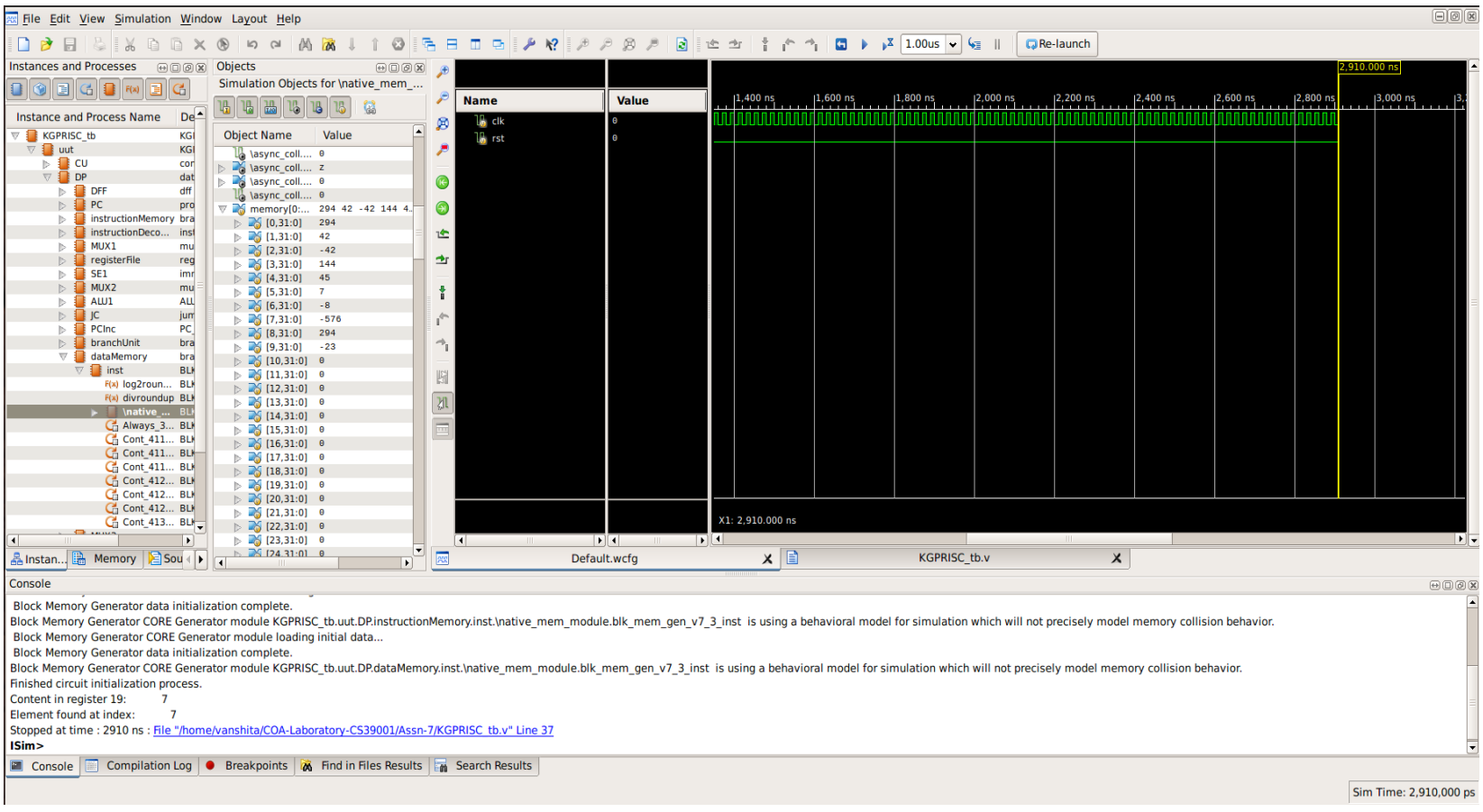
```
memory_initialization_radix=2;
memory_initialization_vector=
00000110100101000000000000000001,
00001110100000000000000000000100,
00000110101101010000000000000001,
00001110101000000000000000001010,
00000101000010000000000000000001,
00010110100010011111111111111100,
00000101010010100000000000000001,
00000000101001000000000000000000,
00000000101110101000000000000001,
00000000101001011000000000000000,
00100001010000000000000001101000,
00000101100011000000000000000001,
00000000110001000000000000000000,
00001001100000000001000000000000,
00000000110010100000000000000000,
00010101100011010000000000000000,
00000101110011100000000000000001,
00000000111101101000000000000001,
00000000111001001000000000000000,
00000000111001111000000000000000,
00100001110000000000000001011100,
00001101000000000000000000000001,
00101100000000000000000000011000,
00000110011100110000000000000001,
00000010011010000000000000000000,
00101100000000000000000001110000,
00000110011100110000000000000001,
00001110011000001111111111111111,
00000110000100000000000000000001,
00001110000000000000000000000001;
```

Data COE file for Linear Search (when element is present in the array)

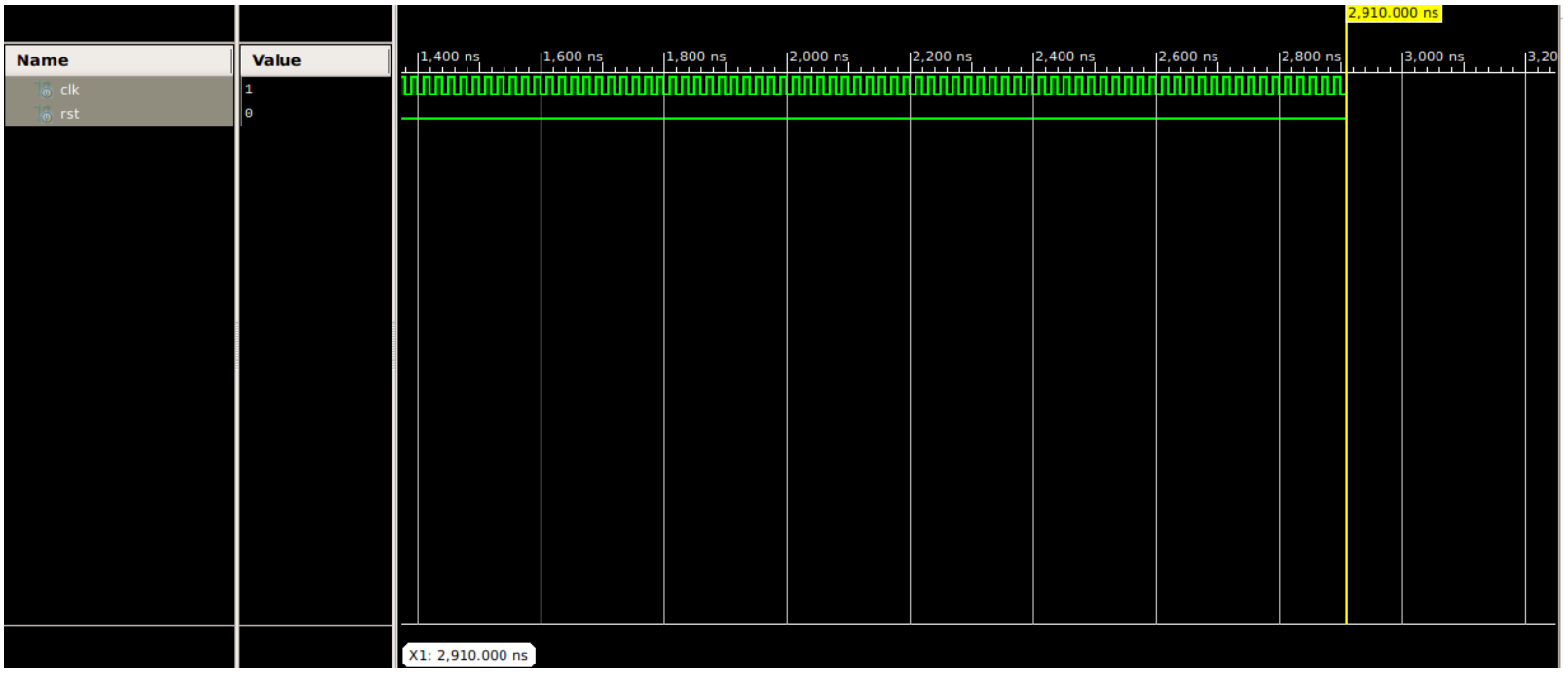
```
memory_initialization_radix=10;
memory_initialization_vector=
294,
42,
-42,
144,
45,
7,
-8,
-576,
294,
-23,
0;
```

Here, the first element in the memory_initialization_vector is the key to be searched, and the next 10 elements are the elements of the array.

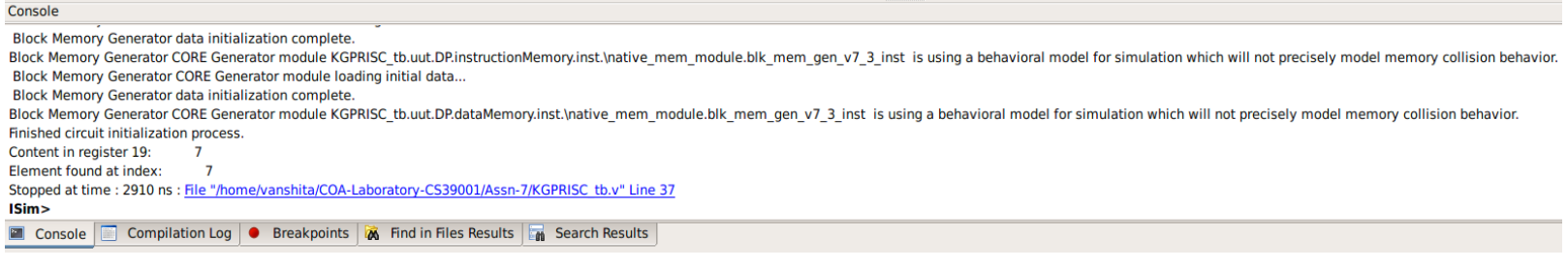
Execution Results for Linear Search (Found Case)



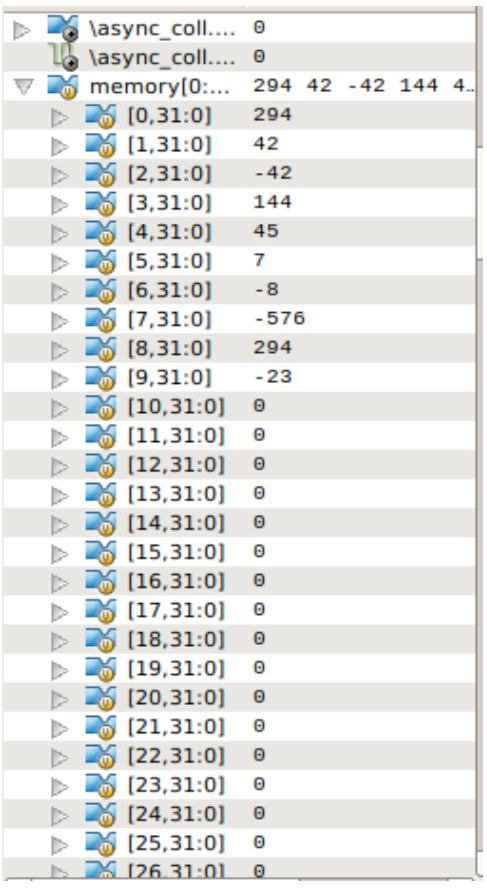
Input Waveforms



Console

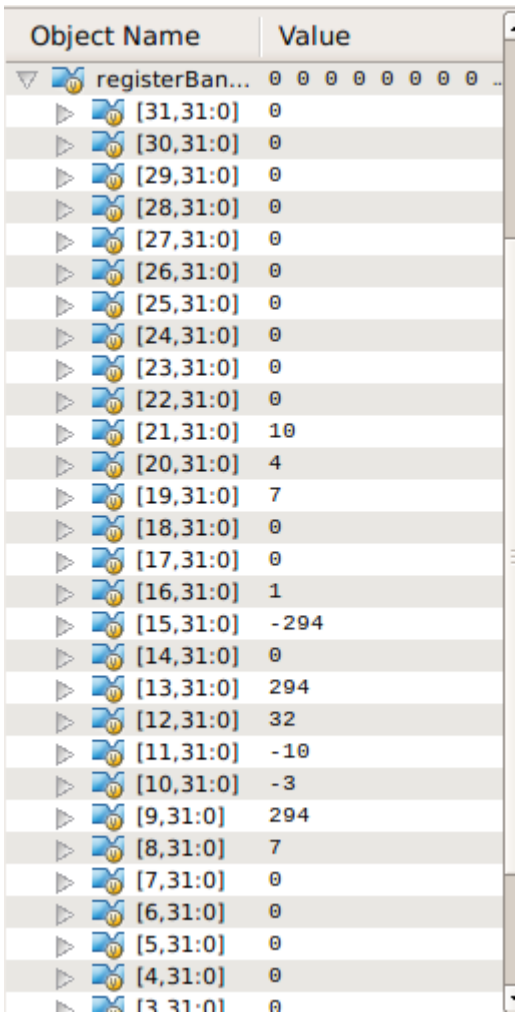


Memory Contents



As we can see, the key to be searched is loaded into the first memory location, and the next 10 locations are filled with the array elements.

Register File Contents



Object Name	Value
registerBan...	0 0 0 0 0 0 0 0 ..
[31,31:0]	0
[30,31:0]	0
[29,31:0]	0
[28,31:0]	0
[27,31:0]	0
[26,31:0]	0
[25,31:0]	0
[24,31:0]	0
[23,31:0]	0
[22,31:0]	0
[21,31:0]	10
[20,31:0]	4
[19,31:0]	7
[18,31:0]	0
[17,31:0]	0
[16,31:0]	1
[15,31:0]	-294
[14,31:0]	0
[13,31:0]	294
[12,31:0]	32
[11,31:0]	-10
[10,31:0]	-3
[9,31:0]	294
[8,31:0]	7
[7,31:0]	0
[6,31:0]	0
[5,31:0]	0
[4,31:0]	0
[3,31:0]	0

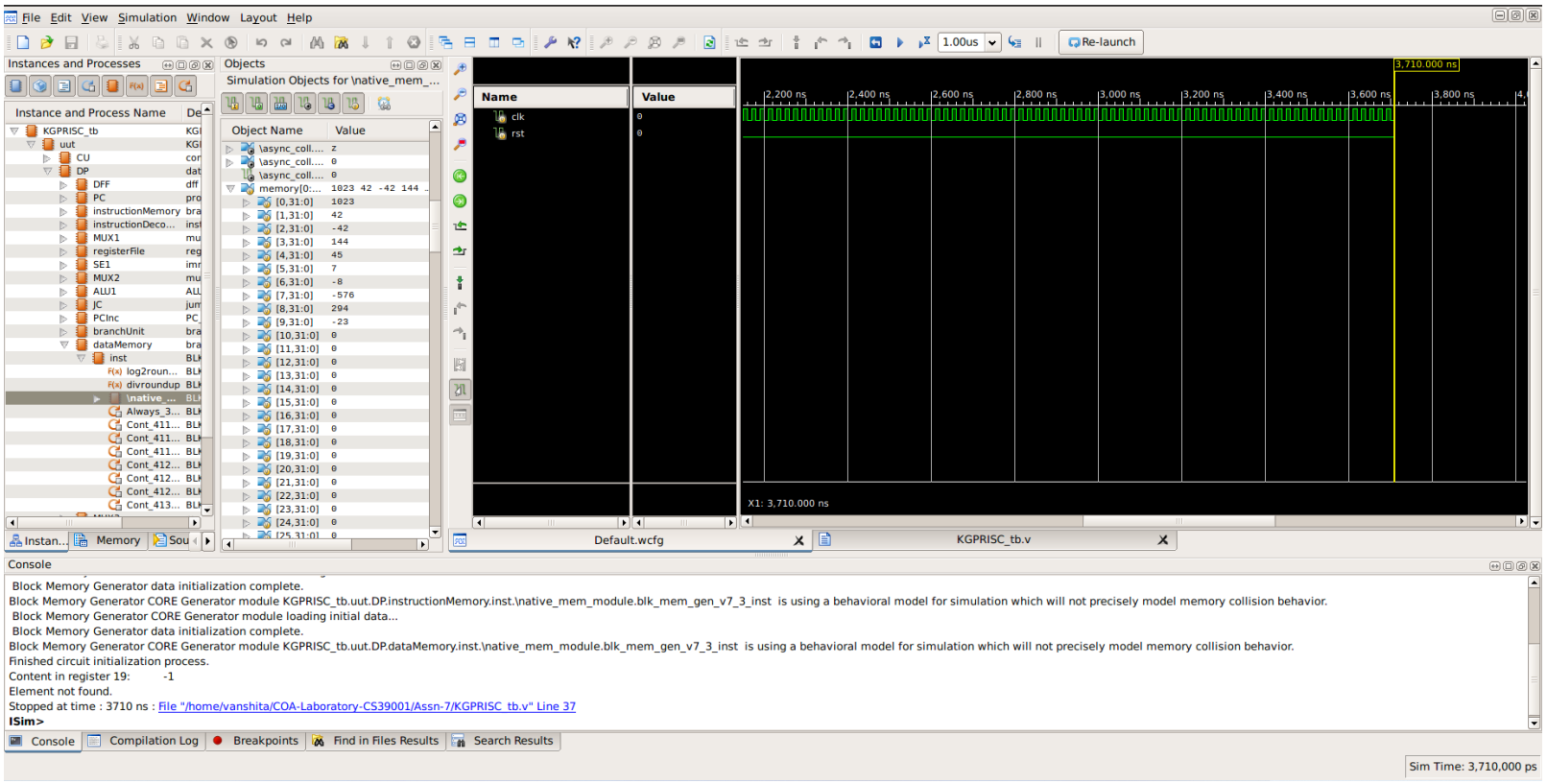
The register $19 stores the index where the element is present, which in this case is index 7.
The value 1 in register $16 indicates the execution was successfully completed.

Data COE file for Linear Search (when element is not present in the array)

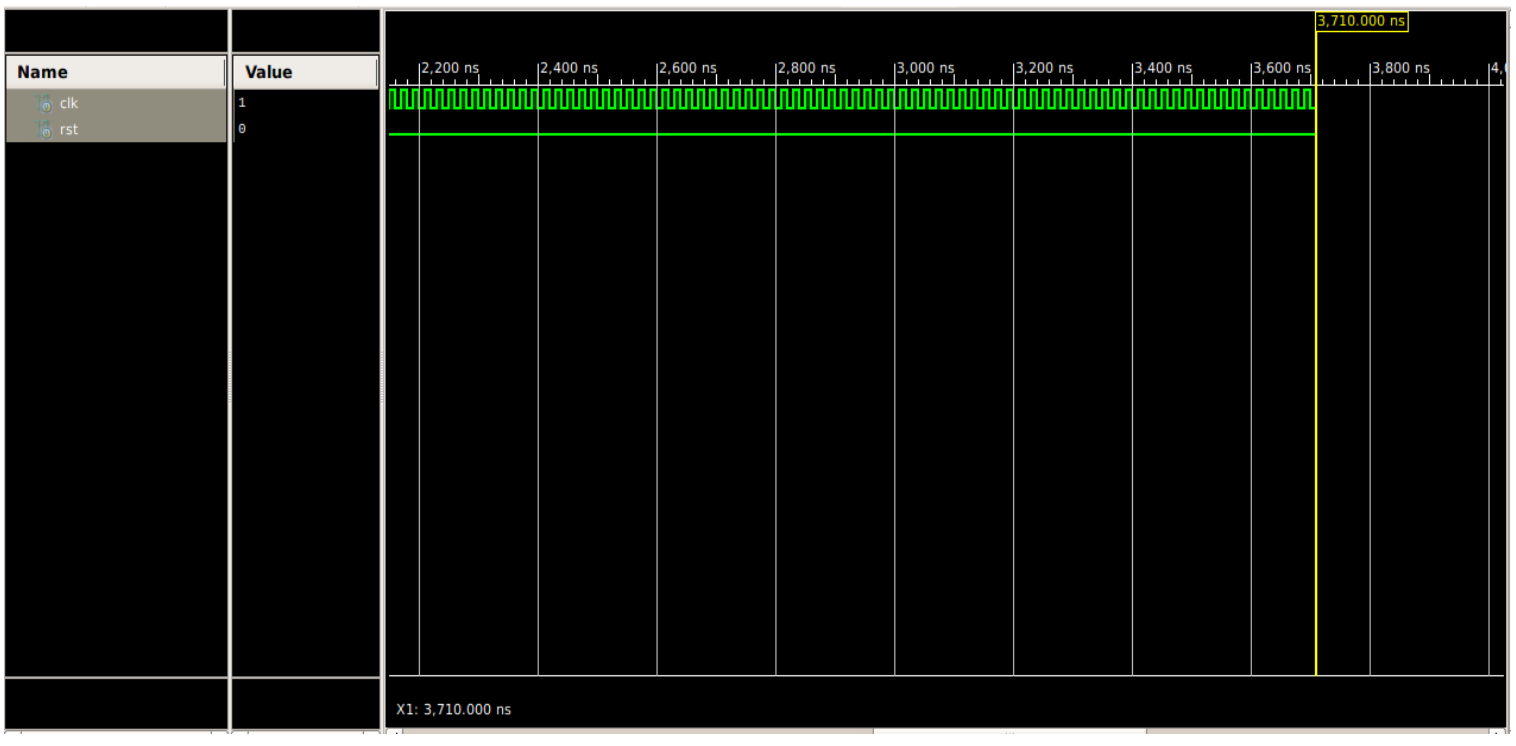
The first element in the memory_initialization_vector is the key to be searched, and the next 10 elements are the elements of the array.

```
memory_initialization_radix=10;
memory_initialization_vector=
1023,
42,
-42,
144,
45,
7,
-8,
-576,
294,
-23,
0;
```

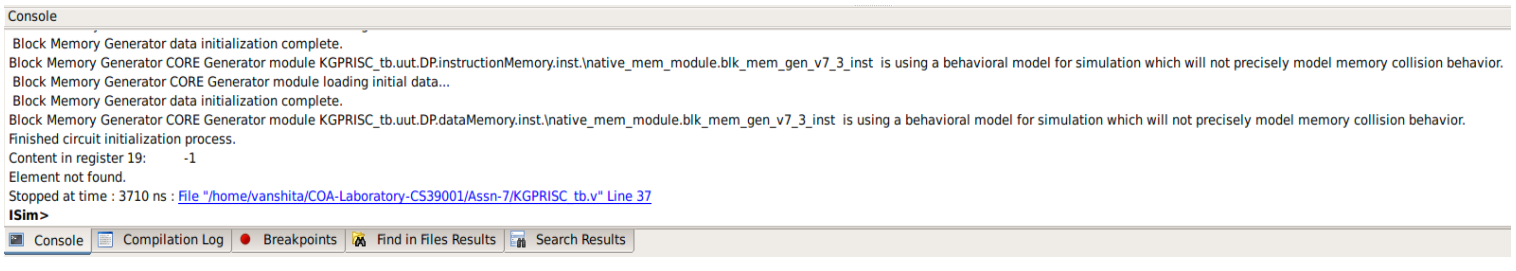
Execution Results for Linear Search (Not Found Case)



Input Waveforms



Console



Memory Contents

Object Name	Value
\async_coll....	z
\async_coll....	0
\async_coll....	0
memory[0:...	1023 42 -42 144 ..
[0,31:0]	1023
[1,31:0]	42
[2,31:0]	-42
[3,31:0]	144
[4,31:0]	45
[5,31:0]	7
[6,31:0]	-8
[7,31:0]	-576
[8,31:0]	294
[9,31:0]	-23
[10,31:0]	0
[11,31:0]	0
[12,31:0]	0
[13,31:0]	0
[14,31:0]	0
[15,31:0]	0
[16,31:0]	0
[17,31:0]	0
[18,31:0]	0
[19,31:0]	0
[20,31:0]	0
[21,31:0]	0
[22,31:0]	0
[23,31:0]	0
[24,31:0]	0
[25,31:0]	0

As we can see, the key to be searched is loaded into the first memory location, and the next 10 locations are filled with the array elements.

Register File Contents

Object Name	Value
registerBan...	0 0 0 0 0 0 0 0 ..
[31,31:0]	0
[30,31:0]	0
[29,31:0]	0
[28,31:0]	0
[27,31:0]	0
[26,31:0]	0
[25,31:0]	0
[24,31:0]	0
[23,31:0]	0
[22,31:0]	0
[21,31:0]	10
[20,31:0]	4
[19,31:0]	-1
[18,31:0]	0
[17,31:0]	0
[16,31:0]	1
[15,31:0]	0
[14,31:0]	1023
[13,31:0]	0
[12,31:0]	40
[11,31:0]	-10
[10,31:0]	0
[9,31:0]	1023
[8,31:0]	10
[7,31:0]	0
[6,31:0]	0
[5,31:0]	0
[4,31:0]	0
[3,31:0]	0

Since the key is not present in the array, the content of register $19 is -1 at the end.
The value 1 in register $16 indicates the execution was successfully completed.

Executing Bubble Sort Algorithm

Assembly Language Code

```
main:
    xor $20, $20        # base address of array = 0 ($20)
    xor $21, $21
    addi $21, 10        # $21 = n = 10
    xor $8, $8          # $8 = i = 0
    xor $9, $9          # $9 = j = 0

fori:
    xor $10, $10
    add $10, $8
    comp $11, $21
    add $10, $11
    addi $10, 1         # $10 = i - (n - 1) = i - n + 1
    bz $10, exitfori    # if i == n - 1, jump to exitfori
    xor $9, $9          # j = 0

forj:
    xor $11, $11
    add $11, $9
    add $11, $10        # $11 = j + i - n + 1
    bz $11, exitforj    # if j == n - i - 1, jump to exitforj

    xor $12, $12
    add $12, $9
    shll $12, 2         # 4 * j
    add $12, $20        # arr + 4 * j
    lw $13, 0($12)      # $13 = arr[j]
    xor $4, $4
    add $4, $12
    addi $12, 4
    lw $14, 0($12)      # $14 = arr[j + 1]
    xor $5, $5
    add $5, $12

    comp $15, $14
    add $13, $15        # arr[j] - arr[j + 1]
    bltz $13, incj
    bz $13, incj
    bl swap             # swap if arr[j] > arr[j + 1]

incj:
    addi $9, 1          # j = j + 1
    b forj

swap:
    lw $18, 0($4)
    lw $19, 0($5)
```

```
    sw $18, 0($5)
    sw $19, 0($4)
    br $31

exitforj:
    addi $8, 1          # i = i + 1
    b fori

exitfori:
    xor $16, $16
    addi $16, 1         # to indicate completion
```

Register Usage:

- $20 stores base address of the array
- $21 stores n
- $8 stores the outer loop variable i
- $9 stores the inner loop variable j
- $16 stores 1 to indicate completion

Instruction COE file for Bubble Sort Program

```
memory_initialization_radix=2;
memory_initialization_vector=
00000110100101000000000000000001,
00000110101101010000000000000001,
00001110101000000000000000001010,
00000101000010000000000000000001,
00000101000010000000000000000001,
00000101001010010000000000000001,
00000101010010100000000000000001,
00000001010010000000000000000000,
00000001011101010000000000000001,
00000001010010110000000000000000,
00001101010000000000000000000001,
00100001010000000000000010100100,
00000101001010010000000000000001,
00000101011010110000000000000001,
00000001011010010000000000000000,
00000001011010100000000000000000,
00100001011000000000000010011100,
00000101100011000000000000000001,
00000001100010010000000000000000,
00001001100000000001000000000000,
00000001100101000000000000000000,
00010101100011010000000000000000,
00000100100001000000000000000001,
00000000100011000000000000000000,
00001101100000000000000000000100,
00010101100011100000000000000000,
00000100101001010000000000000001,
00000000101011000000000000000000,
00000001111011100000000000000001,
00000001101011110000000000000000,
00011101101000000000000010000000,
00100001101000000000000010000000,
00110000000000000000000010001000,
00001101001000000000000000000001,
```

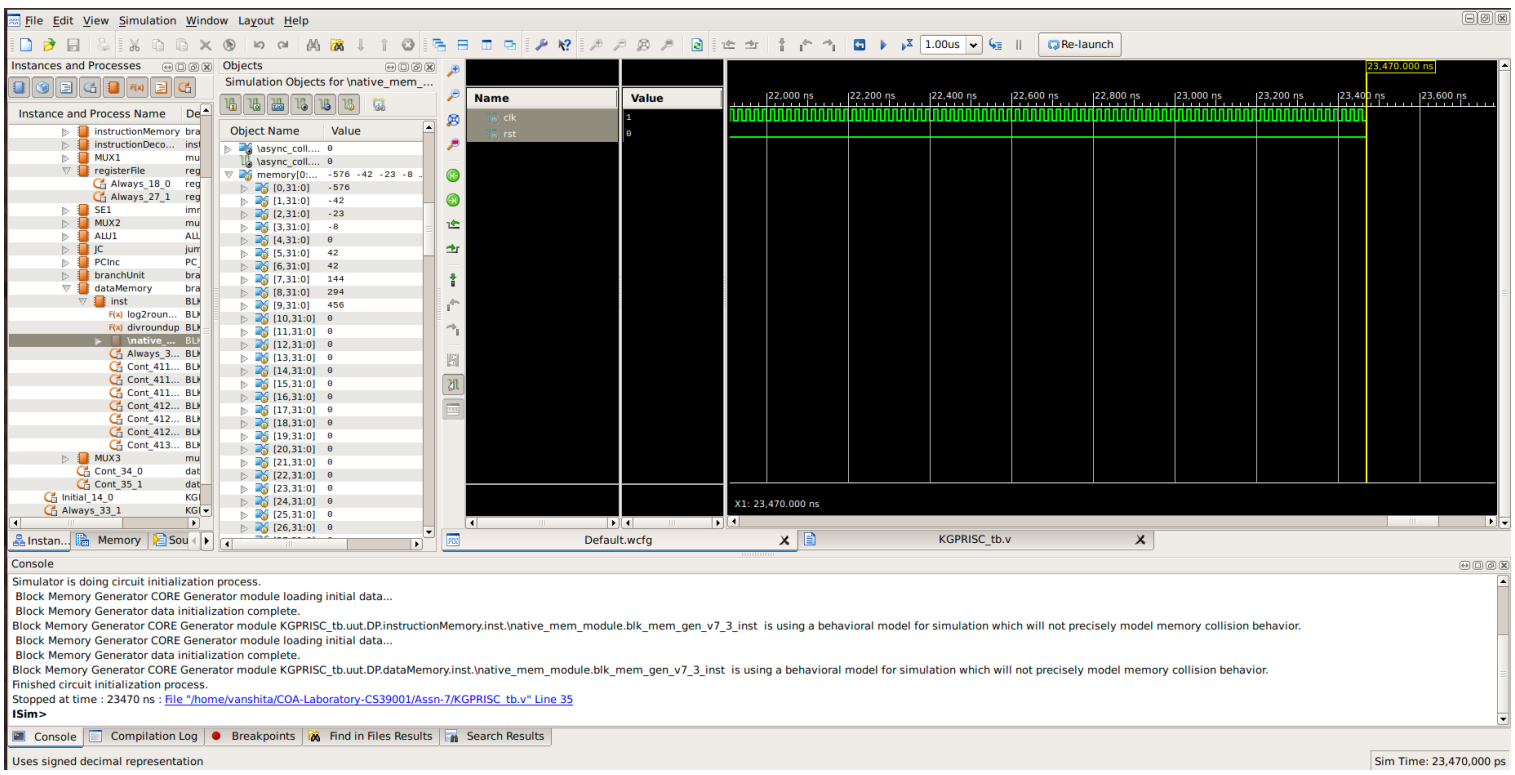
```
00101100000000000000000000110000,
00010100100100100000000000000000,
00010100101100110000000000000000,
00011000101100100000000000000000,
00011000100100110000000000000000,
00101011111000000000000000000000,
00001101000000000000000000000001,
00101100000000000000000000010100,
00000110000100000000000000000001,
00001110000000000000000000000001;
```

Data COE File for Bubble Sort

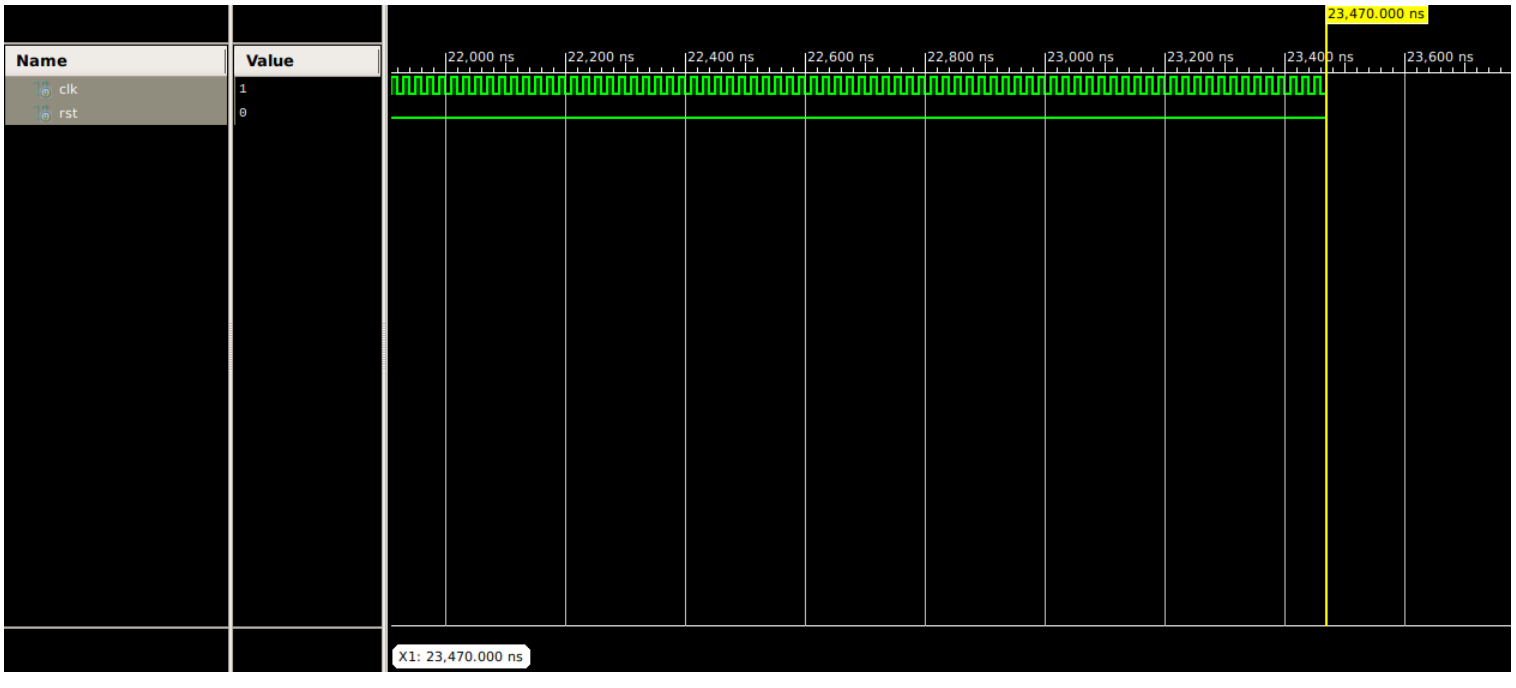
The 10 elements in the memory_initialization_vector are the elements of the array which is to be sorted.

```
memory_initialization_radix=10;
memory_initialization_vector=
42,
-42,
144,
42,
-8,
-576,
294,
-23,
456,
0;
```

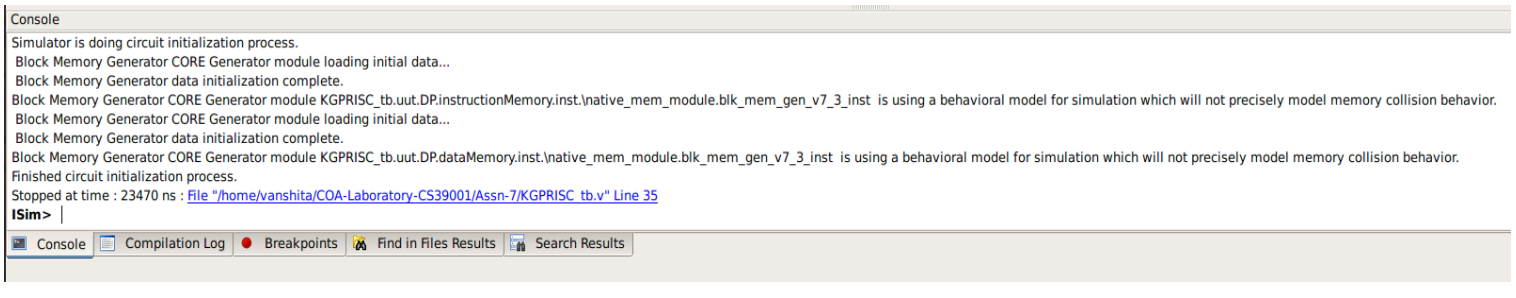
Execution Results for Bubble Sort



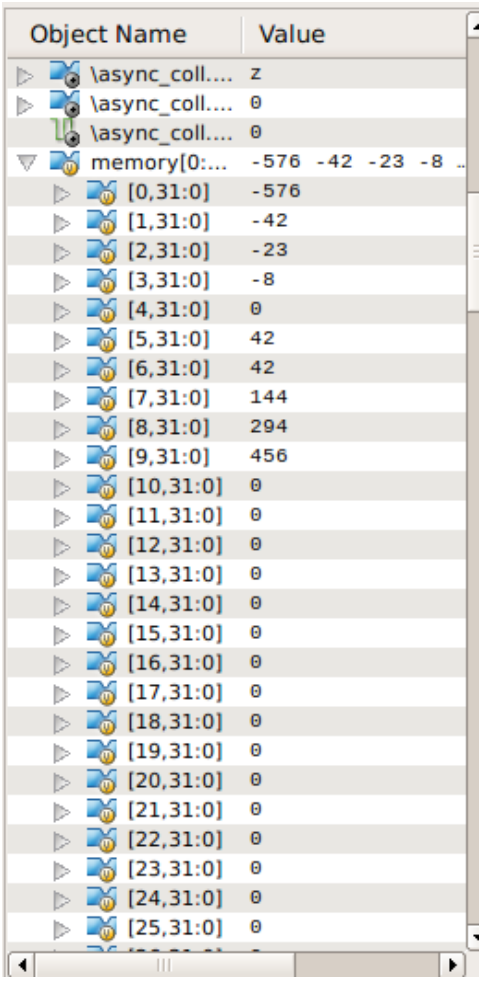
Input Waveforms



Console



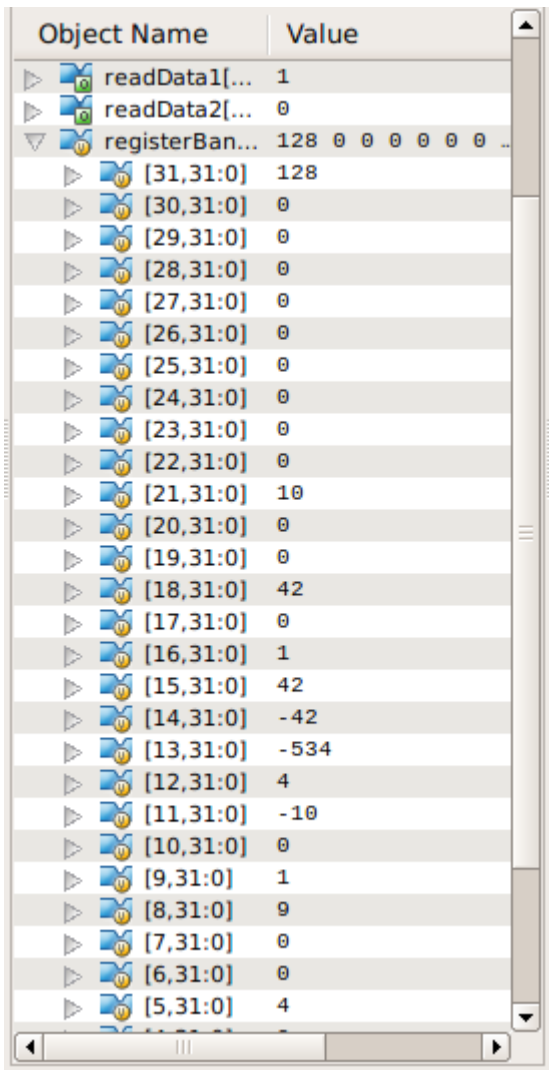
Memory Contents After Execution Ends



Object Name	Value
\async_coll....	z
\async_coll....	0
\async_coll....	0
memory[0:...	-576 -42 -23 -8 ...
[0,31:0]	-576
[1,31:0]	-42
[2,31:0]	-23
[3,31:0]	-8
[4,31:0]	0
[5,31:0]	42
[6,31:0]	42
[7,31:0]	144
[8,31:0]	294
[9,31:0]	456
[10,31:0]	0
[11,31:0]	0
[12,31:0]	0
[13,31:0]	0
[14,31:0]	0
[15,31:0]	0
[16,31:0]	0
[17,31:0]	0
[18,31:0]	0
[19,31:0]	0
[20,31:0]	0
[21,31:0]	0
[22,31:0]	0
[23,31:0]	0
[24,31:0]	0
[25,31:0]	0

We can see that the elements in the data memory from index 0 to 9 are present in sorted order at the end of the execution.

Register File Contents



Object Name	Value
readData1[...	1
readData2[...	0
registerBan...	128 0 0 0 0 0 0 ..
[31,31:0]	128
[30,31:0]	0
[29,31:0]	0
[28,31:0]	0
[27,31:0]	0
[26,31:0]	0
[25,31:0]	0
[24,31:0]	0
[23,31:0]	0
[22,31:0]	0
[21,31:0]	10
[20,31:0]	0
[19,31:0]	0
[18,31:0]	42
[17,31:0]	0
[16,31:0]	1
[15,31:0]	42
[14,31:0]	-42
[13,31:0]	-534
[12,31:0]	4
[11,31:0]	-10
[10,31:0]	0
[9,31:0]	1
[8,31:0]	9
[7,31:0]	0
[6,31:0]	0
[5,31:0]	4

The value 1 in register $16 indicates the execution was successfully completed.