

EE 705 – Course Project
VLSI Design Lab

Hardware Accelerated Sudoku Solver

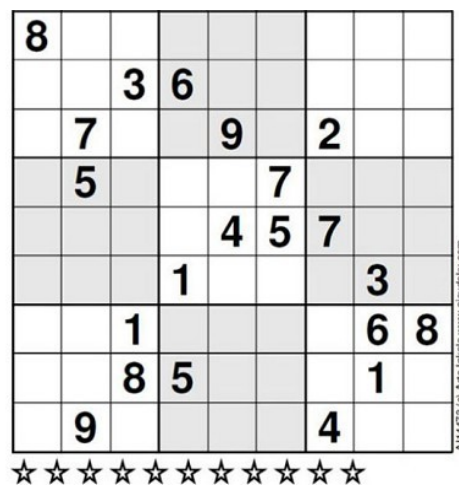
Sourabh Suri
183079015
EE-3

Algorithm

Sudoku contains 81 cells, in a 9×9 grid. Each row, column and block can contain a series number from 1 to 9.

Proper Sudokus have one solution. A brute force algorithm can be used to solve a sudoku on computers. This algorithm visits a cell which is empty and fill it sequentially till a valid fill is not possible. When a valid fill is not found then backtracking is done to change a previously allotted sequential bit to other number. After writing this new number, the solver again attempts to fill all empty cell untill further violatoinis occur or all cell gets filled. This is repeated until the allowed value in the last (81st) cell is a valid guess.

Hardest Sudoku: A typical example to run comparative study,



Reference: <https://www.telegraph.co.uk/news/science/science-news/9359579/Worlds-hardest-sudoku-can-you-crack-it.html>

A simple C code is written to solve sudoku using basic Backtracking problems. A recursive function is defined which stacks up untill an end condition is met. Thus, assigning numbers to next empty cell found in puzzle before checking if its a valid entry or not. A simple check that same entry is not present in its corresponding row, column and block. After checking the number is assigned and again calling the same assignment function recursively to satisfy the end test that this assignment would lead to a solution of puzzle or not. If not, try the next number before checking for its validity.

Hence, simply row and column of unassigned cell and check for conflict from digits one to nine. If there is no conflict at row column and block, If recursion is successful, return true but if not, remove the digit assigned and try another. If all digits fail, return false, thus, this will cause the previous function called in recursion to change the digit it selected then. Untill all the recursions return true a puzzle is still getting solved. When no empty cell left, the puzzle is solved.

This method serves many advantages. Firstly, it guarantees a solution, however, a user should enter a valid puzzle. It serves a simple algorithm. But, it takes long time and is proved to be slow compared to other advanced algorithms. Such an algorithm may typically require as few as 15,000 cycles, or as many as 900,000 cycles to solve a Sudoku.

Solving a sudoku can also be modelled as a constraint satisfaction problem. Reasoning based constraints can be applied to solve as well as model a problem. So, incorporating an algorithm where backtracking and constraint based reasoning would be able to reduce the time to compute and solve all type of sudoku. Following is the above mentioned sudoku game, where zero represents an empty cell.

Puzzle

800000000
003600000
070090200
050007000
000045700
000100030
001000068
008500010
090000400

Row bit map of the above game is shown below,

	Element
Row	1 2 3 4 5 6 7 8 9
1	0 0 0 0 0 0 0 1 0
2	0 0 1 0 0 1 0 0 0
3	0 1 0 0 0 0 1 0 1
4	0 0 0 0 1 0 1 0 0
5	0 0 0 1 1 0 1 0 0
6	1 0 1 0 0 0 0 0 0
7	1 0 0 0 0 1 0 1 0
8	1 0 0 0 1 0 0 1 0
9	0 0 0 1 0 0 0 0 1

Column bit map of the above game is shown below,

	Element
Column	123456789
1	000000010
2	000010101
3	101000010
4	100011000
5	000100001
6	000010100
7	010100100
8	101001000
9	000000010

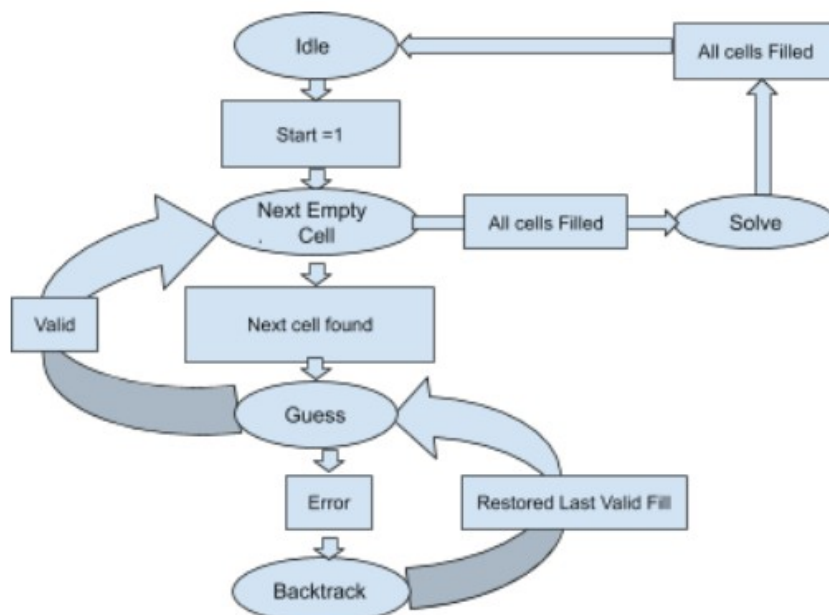
Block bit map of the above game is shown below,

Block	Element								
	1	2	3	4	5	6	7	8	9
1	0	0	1	0	0	0	1	1	0
2	0	0	0	0	0	1	0	0	1
3	0	1	0	0	0	0	0	0	0
4	0	0	0	0	1	0	0	0	0
5	1	0	0	1	1	0	1	0	0
6	0	0	1	0	0	0	1	0	0
7	1	0	0	0	0	0	0	1	1
8	0	0	0	1	1	0	0	0	0
9	1	0	0	1	0	1	0	1	0

Bitmaps are used to constraint a validity of an element for a certian cell. Thus, bitmaps represents a constraint that, such elements are already present in a row or column or block. The puzzle is searched for empty cell.

When a cell is found, a valid element is evaluated based on candidate selection table defined in the FSM later. Thus, a brute force method could be efficient when combined with an elimination method.

Finite State Machine



When the puzzle is loaded, i.e. start=1, the algorithm starts. FSM will goto next empty state where next empty element is determined. It is a simple logic of priority encoder where first zero found element's index are reported as selected row, selected column and selected block. Now, machine proceeds to guess state.

The above mentioned bitmaps are used to generate the candidate selection table. Every bitmap contains the information that whether a particular cell is occupied (1) or empty (0). This information about which element are yet to be occupied in a particular row, column and block. is fed to a priority encoder. Randomly selecting a number would make it difficult to backtrack as then we have to keep a track of it. Also, first number which will always be the least valid number which can be filled onto that cell can be used. It would help in creating logic for backtracking as will be explained further as FSM proceeds. In the above example, the first empty cell was (1,2) thus making candidate selection table for first row, second column and first block.

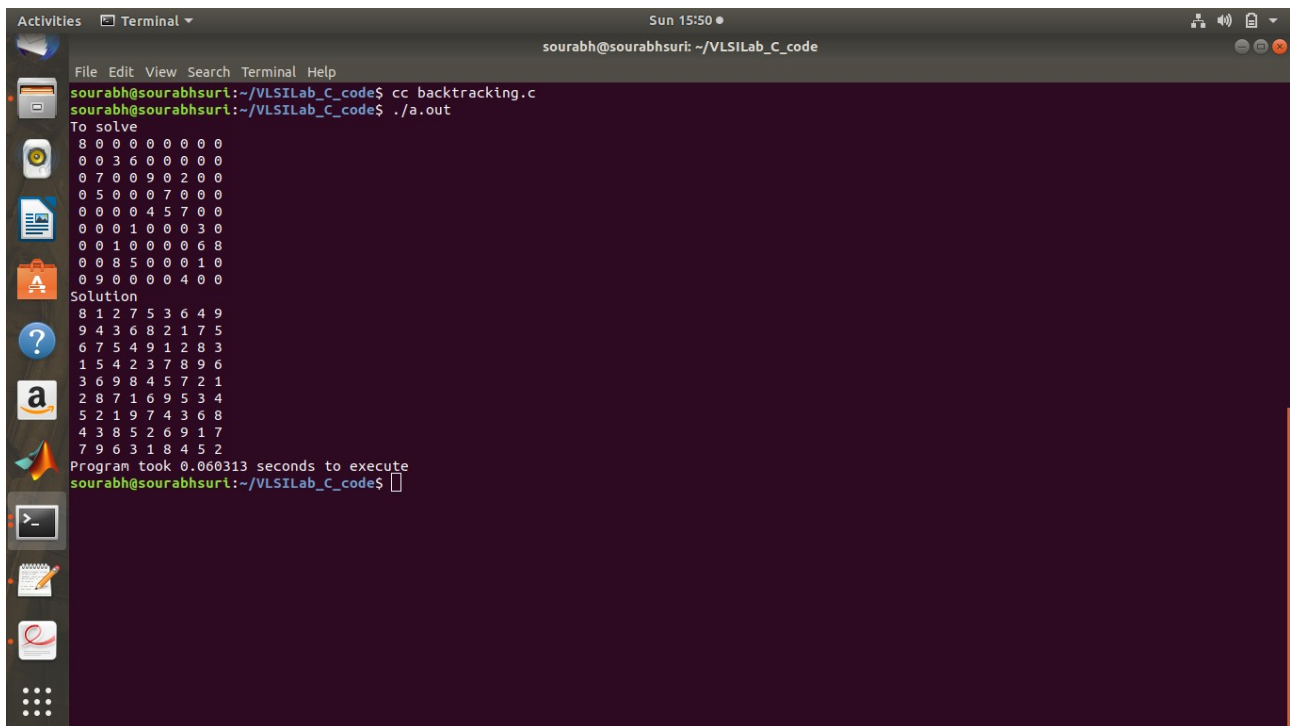
Candidate Selection									
Element	1	2	3	4	5	6	7	8	9
Row	0	0	0	0	0	0	0	1	0
Column	0	0	0	0	1	0	1	0	1
Block	0	0	1	0	0	0	1	1	0
Bitwise OR	0	0	1	0	1	0	1	1	1

This represents that 1,2,4 and 6 are valid elements which can be filled. As mentioned earlier, selecting 1 as valid element FSM proceeds further. When this cell is filled, its bitmaps are also updated correspondingly. Also, this element and its address location is filled onto the stack to remember which element was filled where. This process of finding empty cells and filling repeats until sudoku completes or some violation occurs. When a violation occurs, means that candidate selection table cannot find any zero entry in it. Thus, we need to backtrack and check for the last filled cell. It is backtracked by popping a value from stack where its address was stored. This particular address is read and its element is stored. Its location is cleared and again bitwise ORing test of bitmaps is done to find the valid elements which can be filled. Now a valid symbol other than the one which was already put in it is selected. Thus, next larger element than the one restored from stack is selected from valid list and further filling of puzzle proceeds. Eventually algorithm fills in the last empty cell.

Example Sudoku Under Test:

8								
		3	6					
	7			9		2		
	5				7			
				4	5	7		
			1				3	
		1					6	8
		8	5				1	
	9					4		

C code Implementation

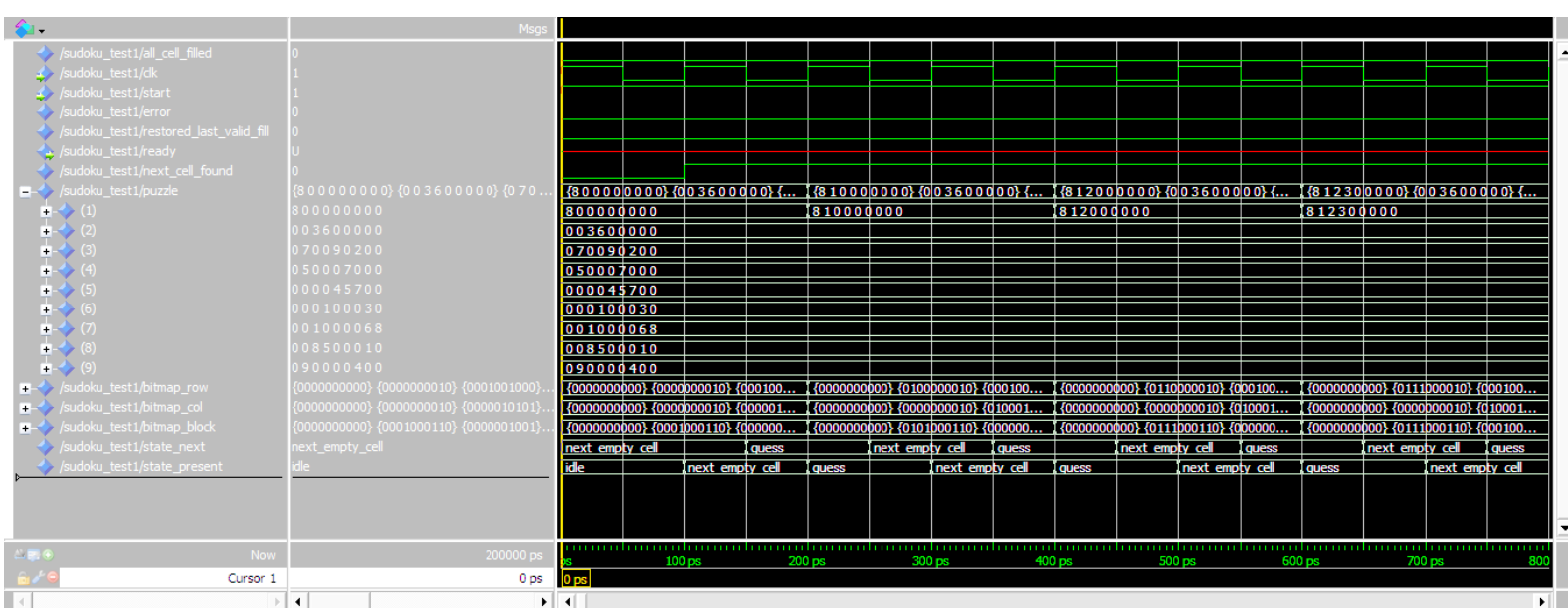


```
Activities Terminal Sun 15:50
sourabh@sourabhsuri: ~/VLSILab_C_code
File Edit View Search Terminal Help
sourabh@sourabhsuri:~/VLSILab_C_code$ cc backtracking.c
sourabh@sourabhsuri:~/VLSILab_C_code$ ./a.out
To solve
8 0 0 0 0 0 0 0 0
0 0 3 6 0 0 0 0 0
0 7 0 0 9 0 2 0 0
0 5 0 0 0 7 0 0 0
0 0 0 0 4 5 7 0 0
0 0 0 1 0 0 0 3 0
0 0 1 0 0 0 0 6 8
0 0 8 5 0 0 0 1 0
0 9 0 0 0 4 0 0
Solution
8 1 2 7 5 3 6 4 9
9 4 3 6 8 2 1 7 5
6 7 5 4 9 1 2 8 3
1 5 4 2 3 7 8 9 6
3 6 9 8 4 5 7 2 1
2 8 7 1 6 9 5 3 4
5 2 1 9 7 4 3 6 8
4 3 8 5 2 6 9 1 7
7 9 6 3 1 8 4 5 2
Program took 0.060313 seconds to execute
sourabh@sourabhsuri:~/VLSILab_C_code$
```

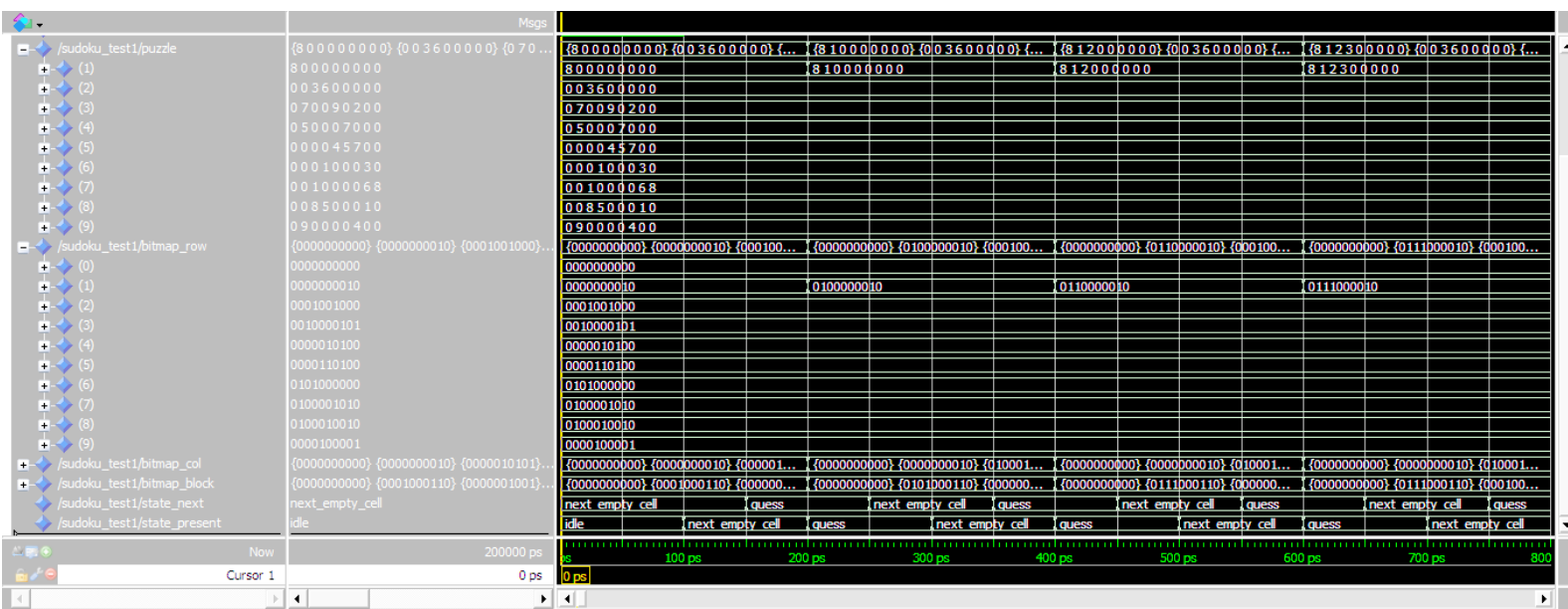
It took 0.060313 seconds to solve this hard sudoku from backtracking. Clock frequency of processor is 2.3GHz.

VHDL Code Implementation

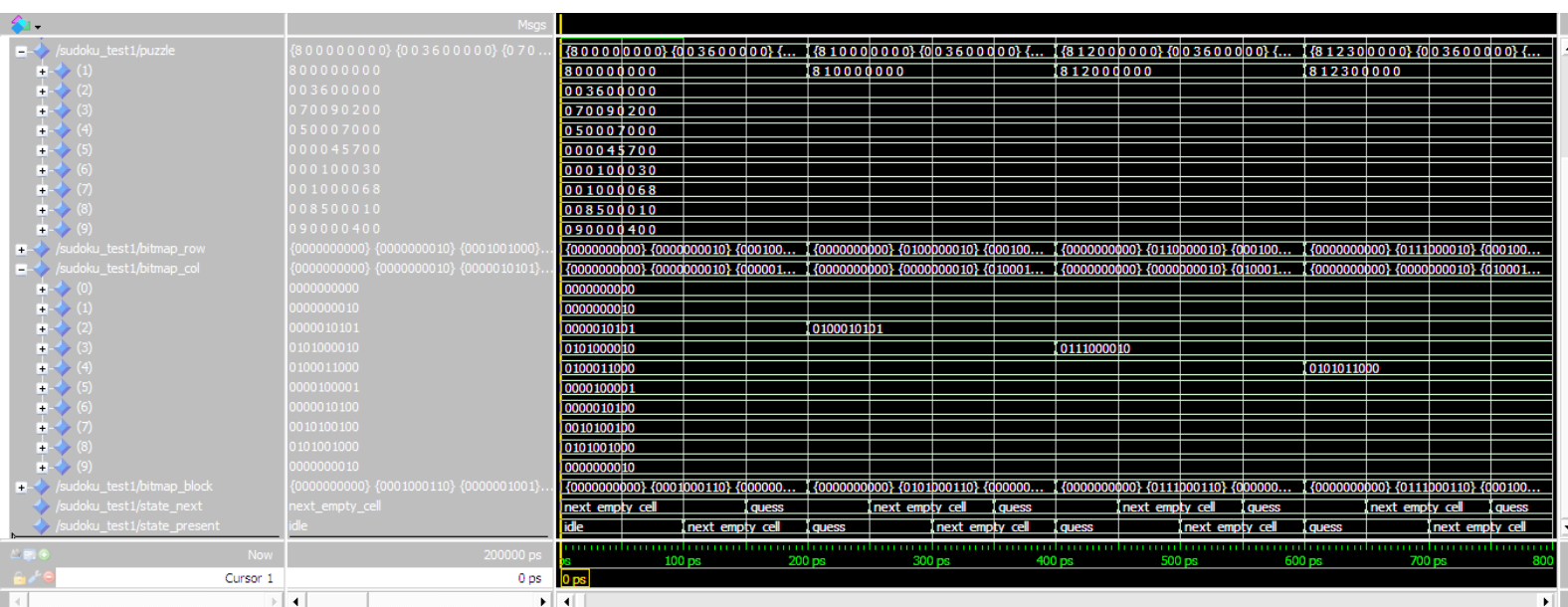
Below Figure shows that how the defined FSM changing states when puzzle has been loaded.



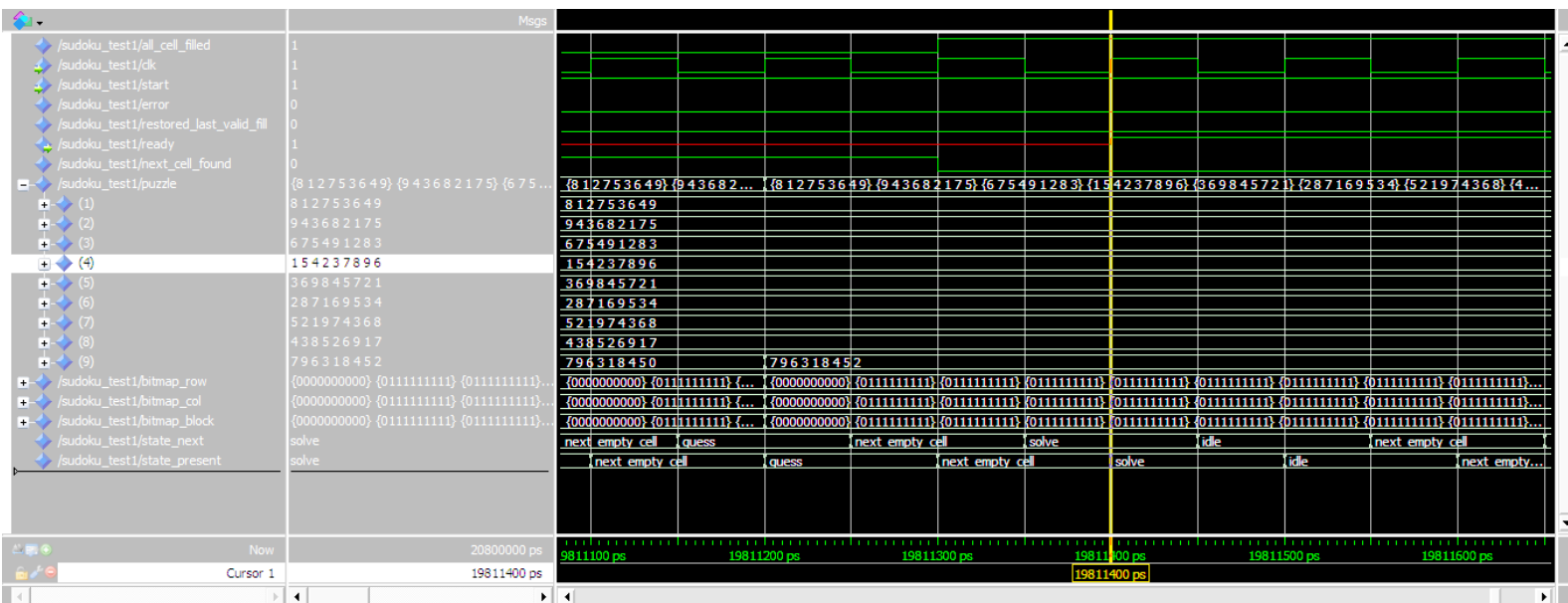
Below shown is bitmap row generated,



Below shown is bitmap column generated,



Below shown is convergence of backtracking and ready bit going high,



Thus, 19.811 micro-seconds took for VHDL code to solve sudoku. Clock Time period was 100 pico second.

For 2.3GHz, 138,690,000 clock cycles are used up by code. However, 198,114 clock cycles used by VHDL.

If clock were 2.3GHz for VHDL, then time taken would be, 86 micro-sec which is less than 60.3 milli-sec as in C code backtracking.

Example, for another typical sudoku,

C code Implementation

```

Activities  Terminal  Mon 02:28
sourabh@sourabhsuri: ~/Desktop/Project/VLSI_Lab

File Edit View Search Terminal Help
sourabh@sourabhsuri:~/Desktop/Project/VLSI_Lab$ ./a.out
To solve
6 0 0 0 0 0 0 0 3
8 0 0 4 5 6 1 0 0
0 5 0 0 0 0 0 0 0
0 1 5 9 0 0 3 0 0
0 0 0 0 1 0 0 0 0
0 6 0 0 8 0 5 0 7
0 0 2 0 0 0 0 0 0
9 0 0 0 0 1 7 4 0
4 7 0 0 9 0 0 0 6

Solution
6 9 4 1 7 8 2 5 3
8 2 3 4 5 6 1 7 9
1 5 7 2 3 9 4 6 8
7 1 5 9 6 2 3 8 4
3 4 8 7 1 5 6 9 2
2 6 9 3 8 4 5 1 7
5 8 2 6 4 7 9 3 1
9 3 6 8 2 1 7 4 5
4 7 1 5 9 3 8 2 6

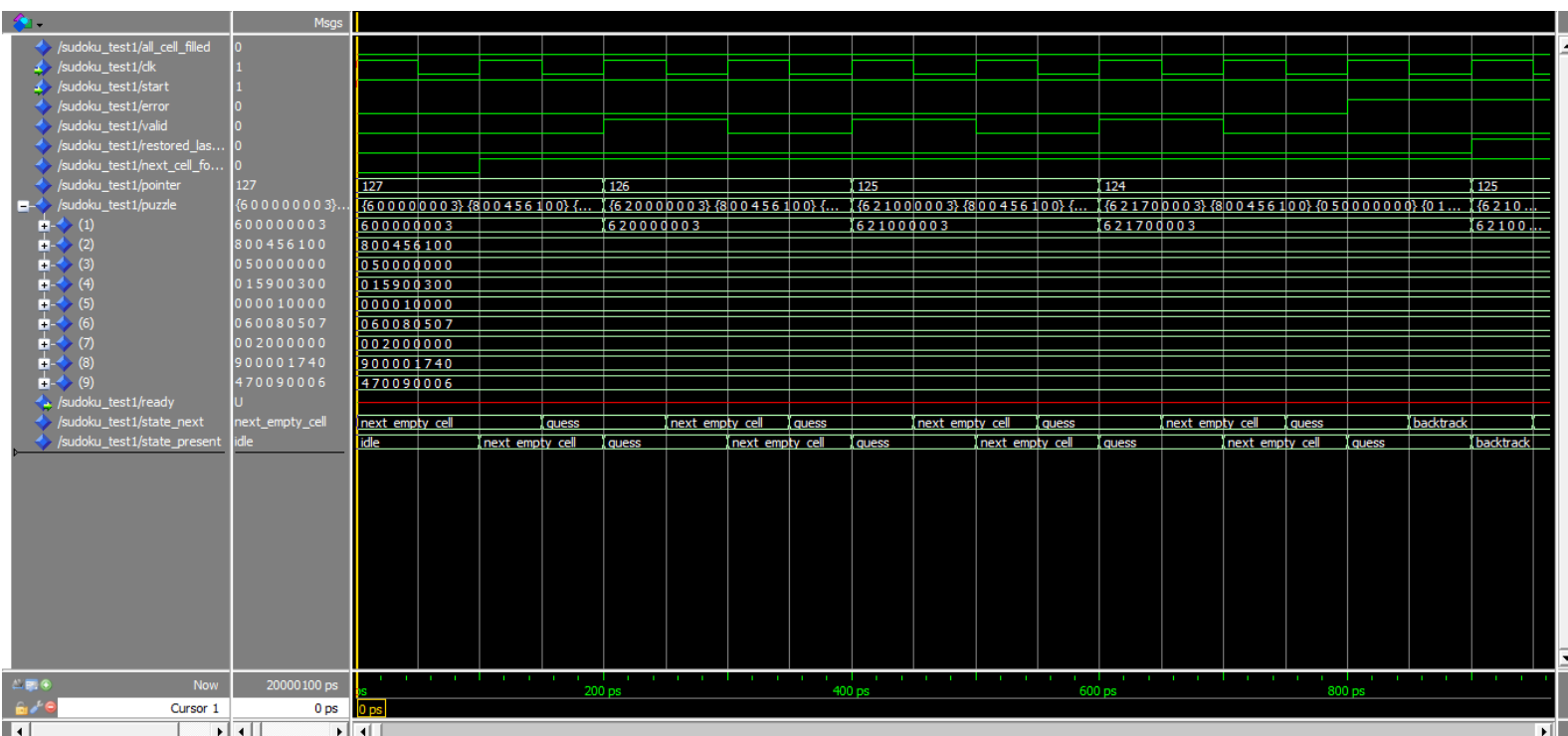
Program took 0.035756 seconds to execute
sourabh@sourabhsuri:~/Desktop/Project/VLSI_Lab$

```

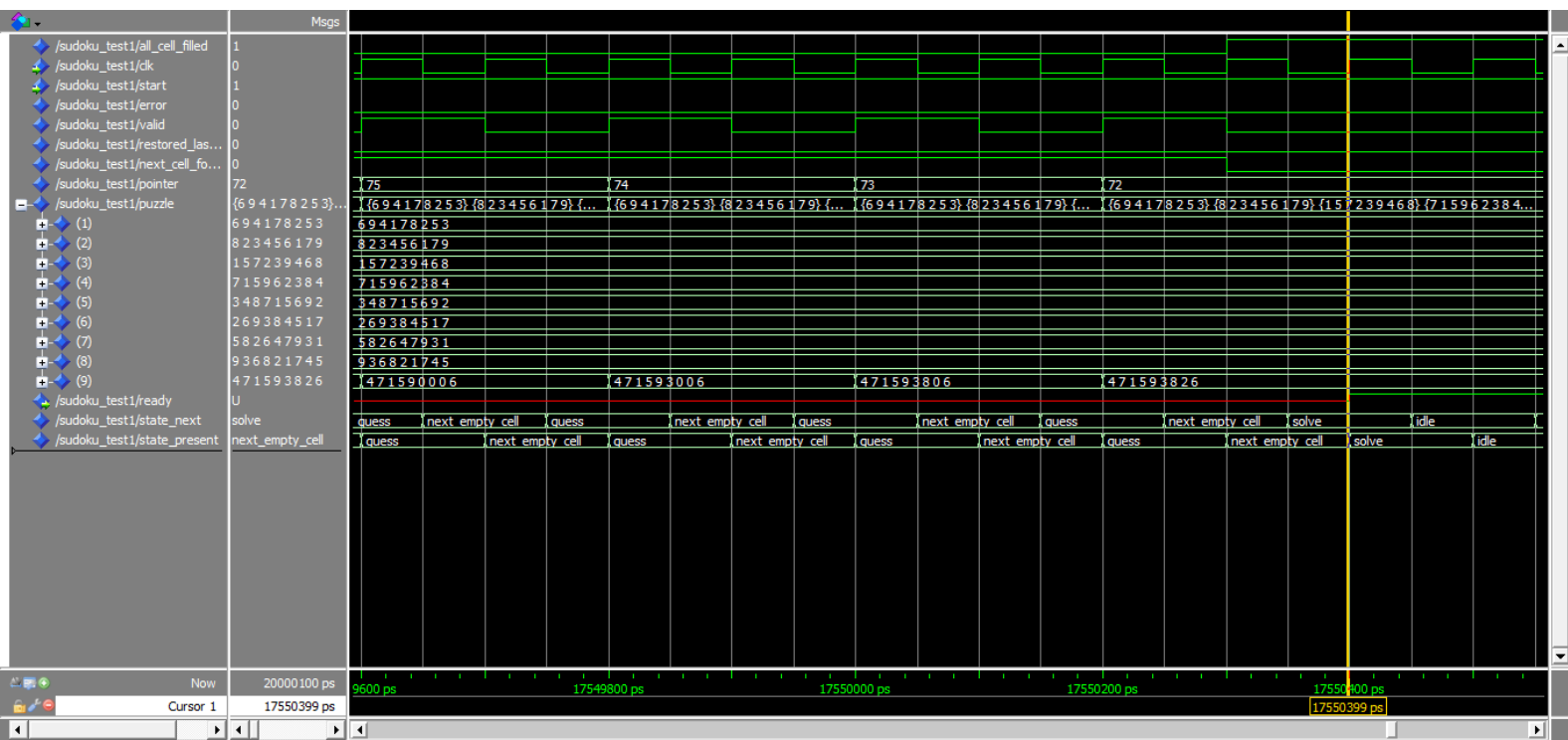
It took 0.035756 seconds to solve this hard sudoku from backtracking. Clock frequency of processor is 2.3GHz.

VHDL Code Implementation

Below Figure shows that how the defined FSM changing states when puzzle has been loaded.



Below shown is time stamp at which puzzle gets solved. Clock frequency = 10GHz.



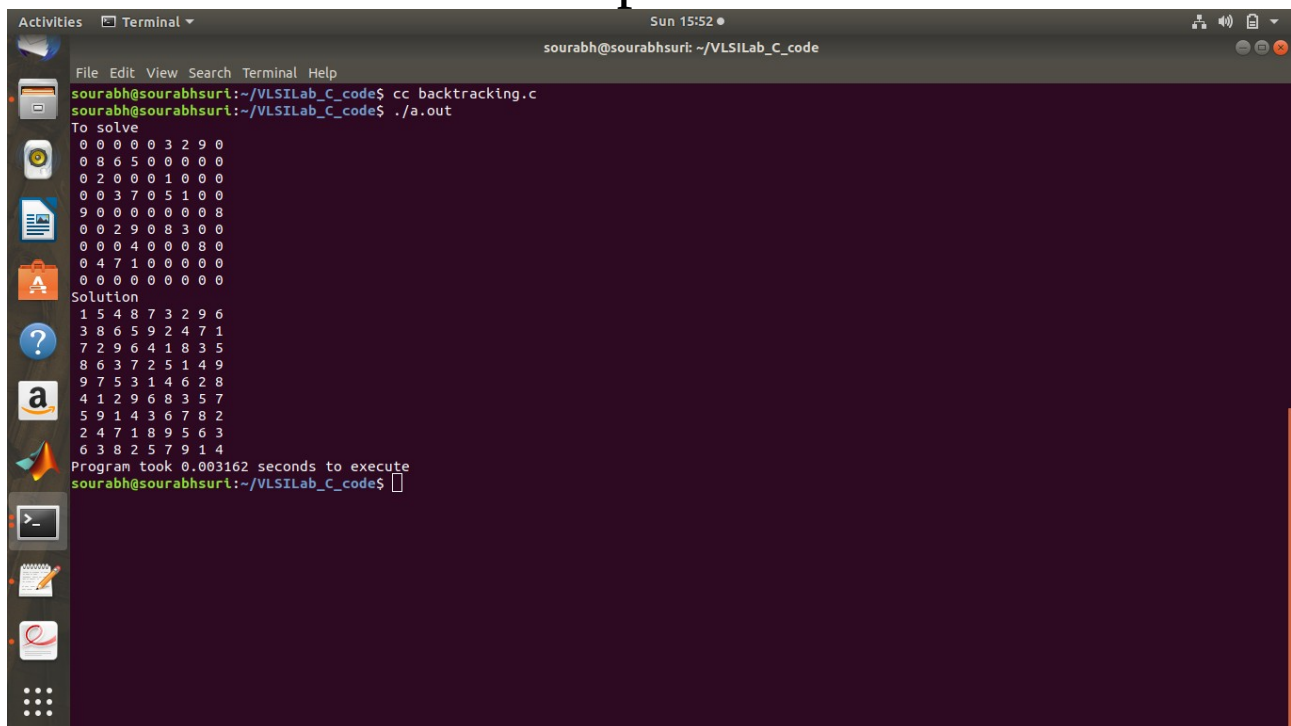
Thus, 17.55 micro seconds took for VHDL code to solve sudoku. Clock Time peroid was 100 pico second.

175,504 clocks were used by VHDL code, while C code took, 15,546,086 clock cycles.

(If clock were, 2.3GHz, then time taken would be, 76 micro-sec which is less than 35 milli-sec as in C code backtracking.)

Another example where easy sudoku is solved,

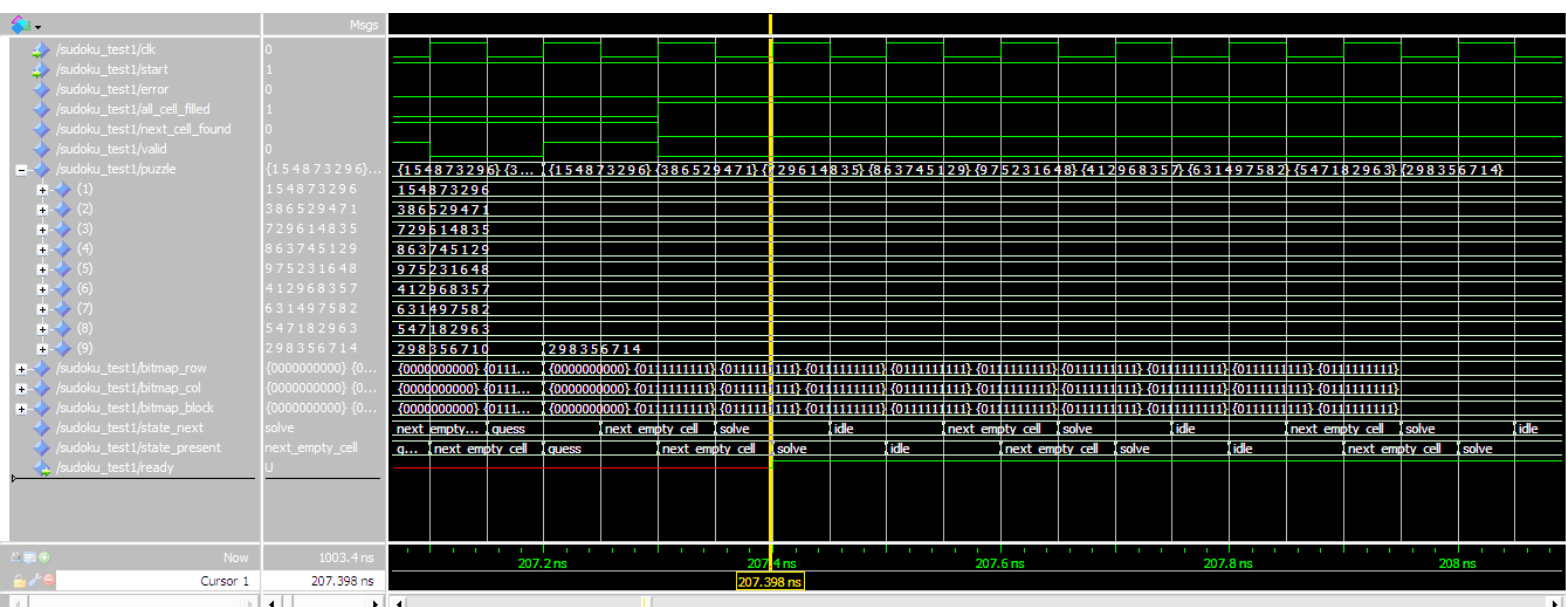
C code Implementation



```
Activities Terminal ▾ Sun 15:52 •
sourabh@sourabhsuri: ~/VLSILab_C_code
File Edit View Search Terminal Help
sourabh@sourabhsuri:~/VLSILab_C_code$ cc backtracking.c
sourabh@sourabhsuri:~/VLSILab_C_code$ ./a.out
To solve
0 0 0 0 0 3 2 9 0
0 8 6 5 0 0 0 0 0
0 2 0 0 0 1 0 0 0
0 0 3 7 0 5 1 0 0
9 0 0 0 0 0 0 0 8
0 0 2 9 0 8 3 0 0
0 0 0 4 0 0 0 8 0
0 4 7 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0
Solution
1 5 4 8 7 3 2 9 6
3 8 6 5 9 2 4 7 1
7 2 9 6 4 1 8 3 5
8 6 3 7 2 5 1 4 9
9 7 5 3 1 4 6 2 8
4 1 2 9 6 8 3 5 7
5 9 1 4 3 6 7 8 2
2 4 7 1 8 9 5 6 3
6 3 8 2 5 7 9 1 4
Program took 0.003162 seconds to execute
sourabh@sourabhsuri:~/VLSILab_C_code$
```

It took 0.003162 seconds to solve this hard sudoku from backtracking. Clock frequency of processor is 2.3GHz.

VHDL Code Implementation



Thus, 207.4 nano-seconds took for VHDL code to solve sudoku. Clock Time peroid was 100 pico second.

VHDL code took 2074 clock cycles while C code took 7,272,600 clock cycles.