

# 8x8 Maze Solver

2010 CVSD Verilog Homework 3 (Updated: 2010/11/09)

## I. Objective

Maze solver is a common yet interesting software programming problem. In this homework, you will be designing an 8x8 maze solver hardware module in *synthesizable* Verilog. Your design's total running (including I/O) cycle will also be scored in addition to its correctness (i.e. *shortest path* is much preferred.) So we do encourage you to explore the possibilities of *fast algorithms* and their *parallel/hardware-oriented* implementations. Furthermore, you need to write and hand in a *testbench file* which can verify your own design.

## II. Description

Figure 1 shows this homework's golden test maze and 3 possible paths. There're totally 64 squares in one 8x8 maze, and each square is represented by a 2-bit value including *free space*, *wall*, *start point* and *end point*. Each maze will only have exactly one start point & one end point. The maze is inputted into your design in a row-by-row manner, and each 16-bit input vector from MSB to LSB represents the squares from left to right. All 8 rows will be inputted *consecutively*.

### Maze Input Sequence:

Row 0 = 16'b01\_01\_01\_01\_01\_01\_00\_00

Row 1 = 16'b01\_10\_00\_00\_01\_00\_00\_00

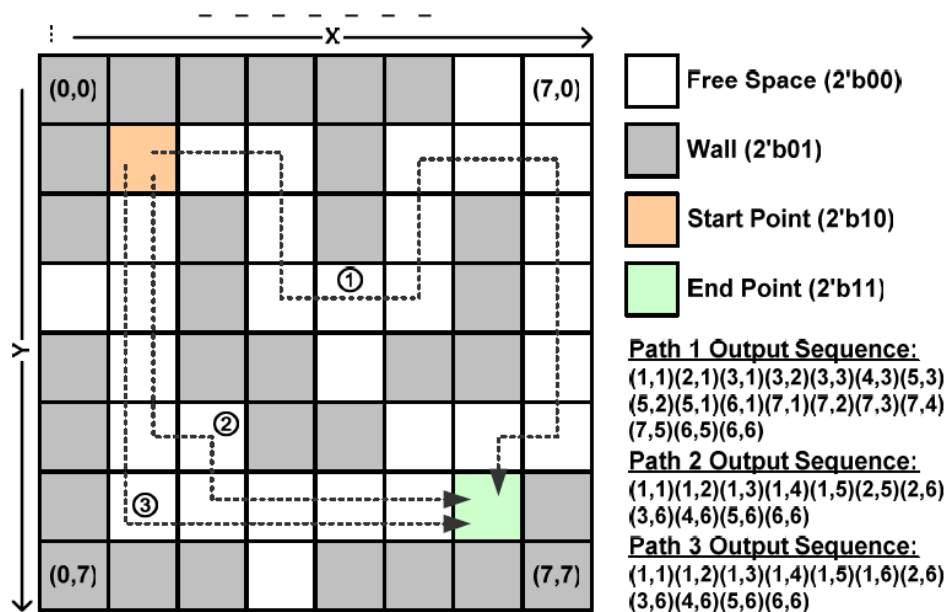


Figure 1: Golden Test Maze and Possible Output Paths

A valid output path is a sequence of (X, Y) coordinates, including *start point*,

some *free spaces*, and finally the *end point*. Each step of the walking pattern is limited to up/down/right/left only, that is, two consecutive output coordinates can only differ by 1 in X or in Y exclusively. Diagonal walk is not allowed. Of course, walking onto the wall or outside the maze is not allowed either. In Fig. 1, paths 2 and 3 are both shortest paths (more preferred), but path 1 is also a correct answer. Note that test mazes may or may not have solution paths connecting the start and end points. In case there is no solution, your solver should be able to tell by outputting the invalid path consisting of the start-point coordinate followed by the end-point coordinate. Worth to note, after outputting one sequence, your design must be able to take a *new turn* of maze solving *without external reset*.

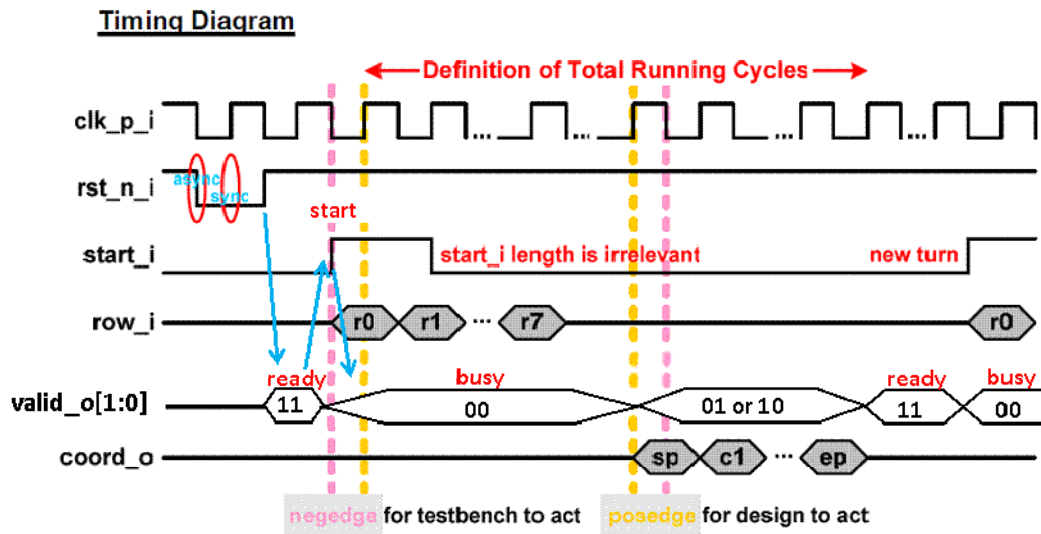
### III. File List & I/O Definition

File list of the homework package:

maze.v	Empty file for you to design. I/O signals cannot be renamed.
maze_tb.v	Empty file for you to write testbench.
report.doc	Template report file. All terms must be filled.

File I/O definition for **maze.v**:

clk_p_i	<i>posedge</i> clock signal. <a href="#">Clock period depend on your design.</a>
rst_n_i	Reset when this signal is high (1'b1). Either async/sync is ok.
start_i	Start to input new maze. First <i>row_i</i> is concurrent with <i>start_i</i> .
row_i[15:0]	Vector of 8 maze squares' types combined together.
valid_o[1:0]	Set 2'b00: when maze is busy Set 2'b01: when maze has a valid path to output at coord_o Set 2'b10: when maze has the invalid path to output at coord_o <a href="#">Set 2'b11: when maze is ready to take input from row_i.</a>
coord_o[5:0]	One output (x, y) combined together. <b>E.g. (2, 7) = 6'b010_111</b>  When there is no valid path, simply output the invalid path consisting of first the start-point coordinate and then the end-point coordinate in two consecutive cycles



**Figure 2: I/O Timing Diagram & Total Running Cycles' Definition**

#### IV. Grading Policy

1. maze.v correctness (including being synthesizable) by TA's testbench: **65%**
2. maze\_tb.v correctness: **10%**
3. maze\_syn.v  $AT = \text{Area}(\mu\text{m}^2) \times \text{Total running time}(\text{ns})$  (after synthesis): **25%**
4. bonus (any special technique that further reduces total cycles, e.g. special walking pattern, decided by TA): **10%**

If your design fails under TA's testbench, you can resubmit a new version, but the 10% score of maze\_tb.v will also be 0. Speed of maze.v is the number of total running (including I/O) time (total cycles x cycle time) as shown in Fig. 2, and remember to define ``timescale` in your code. There's no upper limit on your running cycles, but TA's testbench will still include an over-100000-cycle timeout rule, which should be very sufficient.

#### V. Submission Requirements

File submission:

1. maze.v,
2. maze\_tb.v (set to be able to run for maze.v **without** FSDB dump)
3. maze\_syn.v  
(synthesized netlist by using [TSMC 0.18 um process technology of Synthesis Lab](#))
4. maze\_syn.sdf
5. maze.ddc (Synopsys DC saved file)

Please submit your design in one .zip file with the naming convention:

**StudentID\_HW3.zip** (e.g., R99943001\_HW3.zip).

If your testbench/design needs to include any other file(s), remember to submit them, too. Make sure TAs can directly run **ncverilog maze\_tb.v maze.v** on your files. If you write some modules in other .v files, please *`include* them in your maze.v.

You need to hand in a **hardcopy** of your **report.doc** to **EEII-232**. All blanks must be filled. Also please briefly explain what techniques are applied in your design (figure is ok, too). Otherwise you may not get all the bonus points. In general, your report should be less than **2 pages** (1 page is suggested) with at least 12pt fonts (but not including the parts of synthesis results.) **The total running time in your report is obtained by using the Fig. 1 *golden test maze* in your testbench.**

## **VI. Submission Deadline**

1. Please submit your **.zip** files online using ftp.  
(Deadline: 11/14 Sunday at midnight)
2. Please submit your **hardcopy** of your **report.doc** to EEII-232.  
(Deadline: 11/15 Monday 17:00)