First Design Review

Top-level module
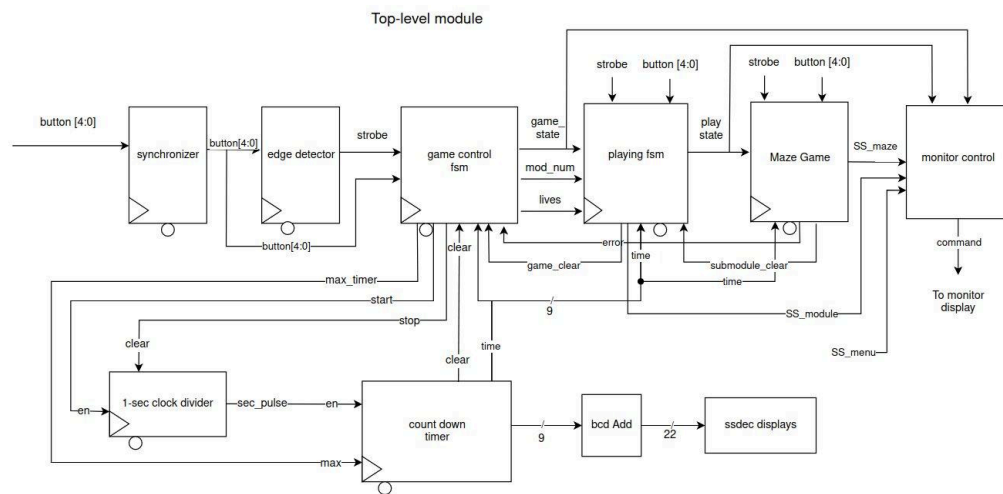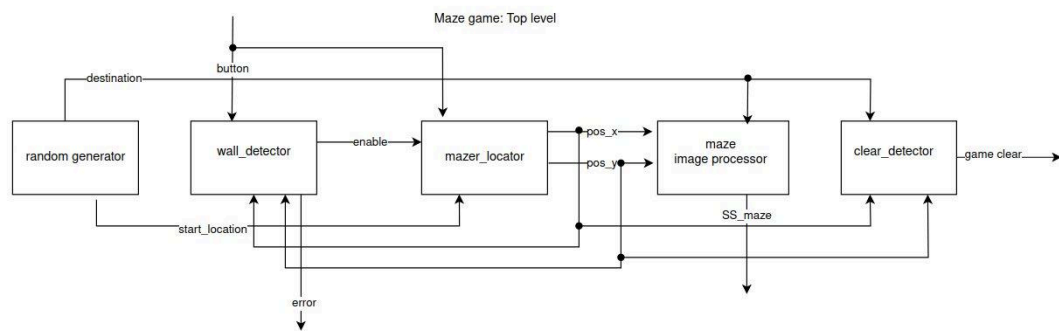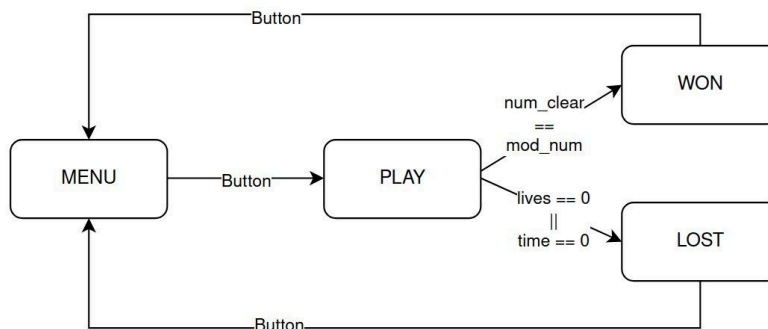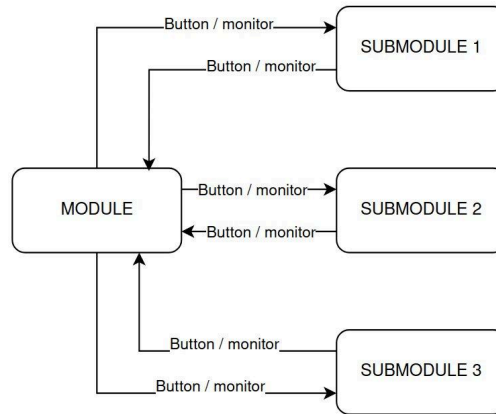

Display 1.1 : Top Level Module (Only Maze Game and the menu states)
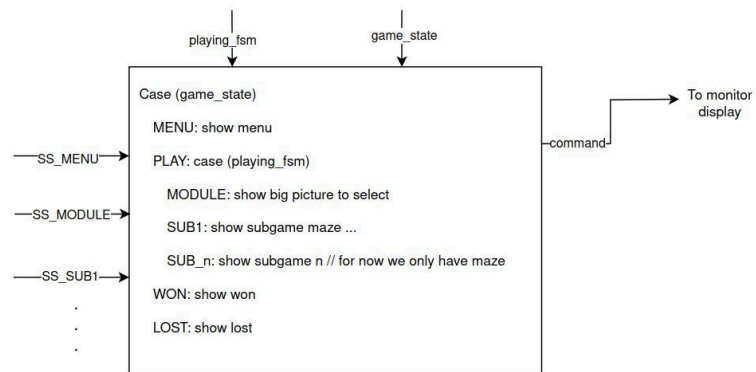
Maze game: Top level

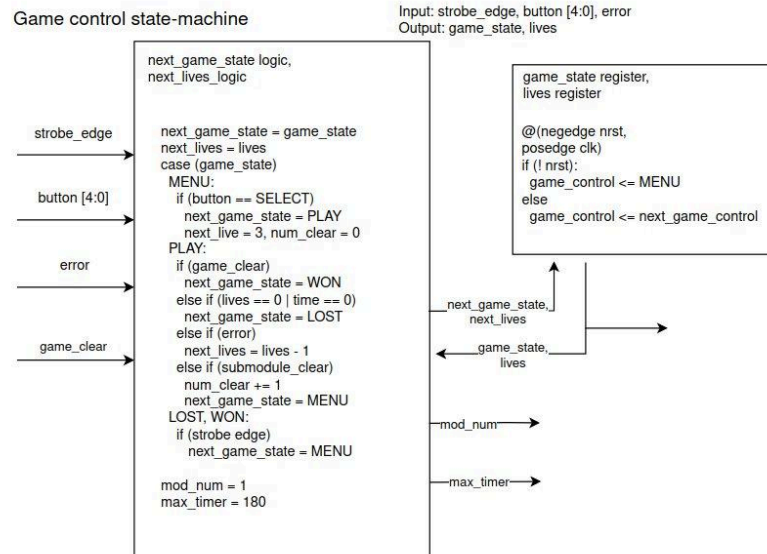Display 1.2 : Maze Game (Focussed specifically on Maze Game)
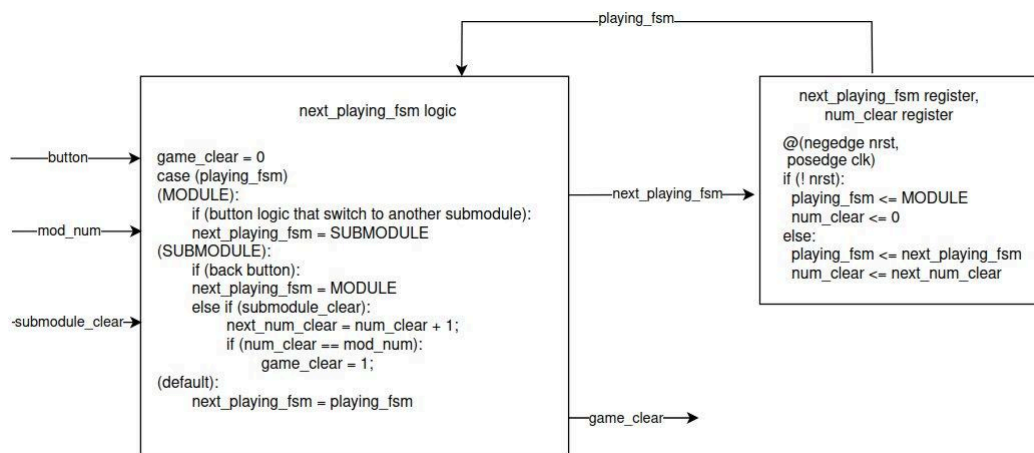

Display 1.3 : Game Control FSM (The game states)
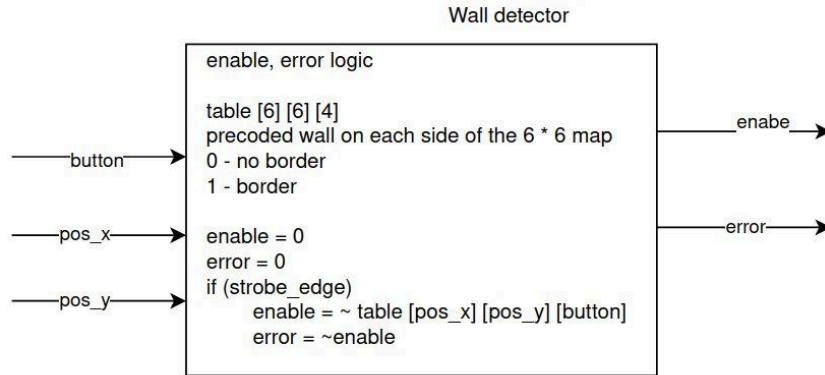
Display 1.4 : Playing FSM (The Game Playing Menu)
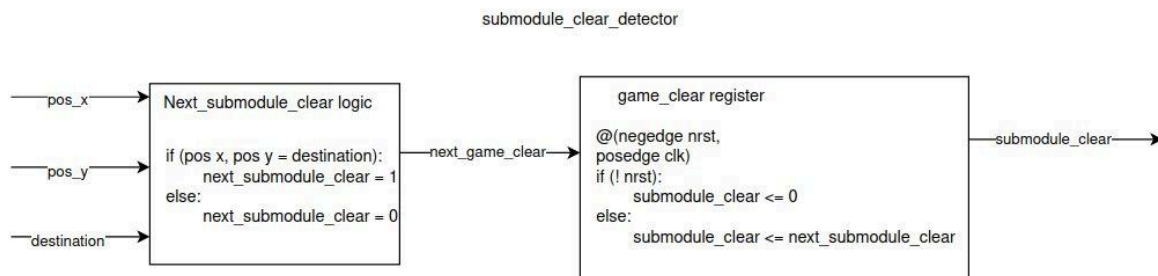

Display 1.5 : Monitor Control (SPI Display)

Game control state-machine

Input: strobe_edge, button [4:0], error
Output: game_state, lives

```
next_game_state logic,
next_lives_logic

  next_game_state = game_state
next_lives = lives
case (game_state)
  MENU:
    if (button == SELECT)
      next_game_state = PLAY
      next_live = 3, num_clear = 0
  PLAY:
    if (game_clear)
      next_game_state = WON
    else if (lives == 0 | time == 0)
      next_game_state = LOST
    else if (error)
      next_lives = lives - 1
    else if (submodule_clear)
      num_clear += 1
      next_game_state = MENU
  LOST, WON:
    if (strobe edge)
      next_game_state = MENU

  mod_num = 1
  max_timer = 180
```

strobe_edge

button [4:0]

error

game_clear

```
game_state register,
lives register

@(negedge nrst,
posedge clk)
if (! nrst):
  game_control <= MENU
else
  game_control <= next_game_control
```

next_game_state,
next_lives

game_state,
lives

mod_num

max_timer

Display 1.6 : Game Control FSM

playing_fsm

```
next_playing_fsm logic

game_clear = 0
case (playing_fsm)
(MODULE):
    if (button logic that switch to another submodule):
        next_playing_fsm = SUBMODULE
(SUBMODULE):
    if (back button):
    next_playing_fsm = MODULE
    else if (submodule_clear):
        next_num_clear = num_clear + 1;
        if (num_clear == mod_num):
            game_clear = 1;
(default):
    next_playing_fsm = playing_fsm
```

button

mod_num

submodule_clear

```
next_playing_fsm register,
num_clear register

@(negedge nrst,
posedge clk)
if (! nrst):
  playing_fsm <= MODULE
  num_clear <= 0
else:
  playing_fsm <= next_playing_fsm
  num_clear <= next_num_clear
```

next_playing_fsm

game_clear

Display 1.7 : Playing FSM

Wall detector

enable, error logic

table [6] [6] [4]
precoded wall on each side of the 6 * 6 map
0 - no border
1 - border

enable = 0
error = 0
if (strobe_edge)
    enable = ~ table [pos_x] [pos_y] [button]
    error = ~enable

button →

pos_x →

pos_y →

→ enabe

→ error

Display 1.8 : Maze Game (Wall Detector)

submodule_clear_detector

Next_submodule_clear logic

if (pos x, pos y = destination):
    next_submodule_clear = 1
else:
    next_submodule_clear = 0

pos_x →

pos_y →

destination →

— next_game_clear —

game_clear register

@(negedge nrst,
posedge clk)
if (! nrst):
    submodule_clear <= 0
else:
    submodule_clear <= next_submodule_clear

— submodule_clear →

Display 1.9 : Puzzle Clear Logic

Maze_locator

start_pos_x
start_pos_y

Next_loc logic

next_pos_x = pos_x
next_pos_y = pos_y
if (enable):
    if (button = UP):
        next_pos_x = pos_x
        next_pos_y = pos_y + 1
    else if (button = RIGHT):
        next_pos_x = pos_x + 1
        next_pos_y = pos_y
    else if (button = DOWN):
        next_pos_x = pos_x
        next_pos_y = pos_y - 1
    else if (button = LEFT):
        next_pos_x = pos_x - 1
        next_pos_y = pos_y

(from wall_detector)

enable →

button →

— next_pos_x

— next_pos_y

loc logic

@(negedge nrst,
posedge clk)

if (! nrst):
    pos_x <= start_pos_x
    pos_y <= start_pos_y
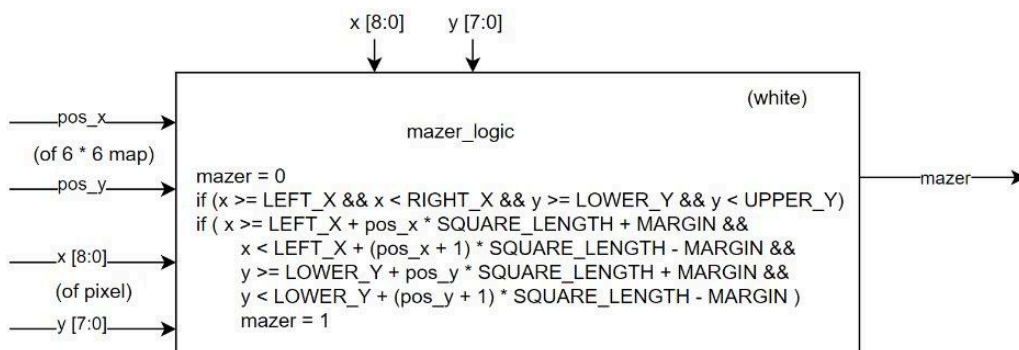else:
    pos_x <= next_pos_x
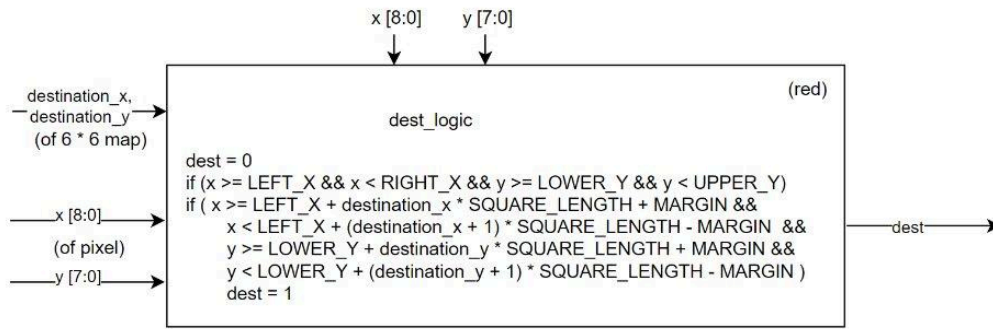   pos_y <= next_pos_y

— pos_x →

— pos_y →

Display 1.10 : Maze Locator
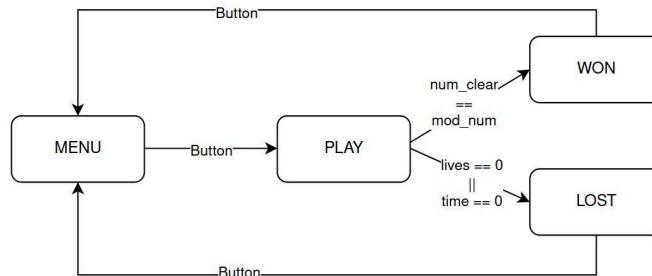
Display 1.11 : Maze Image Processor (Overall)

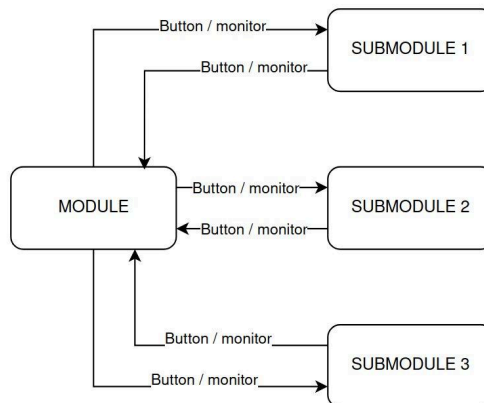Display 1.12 : Maze Image Processor (Mazer Logic Focused)

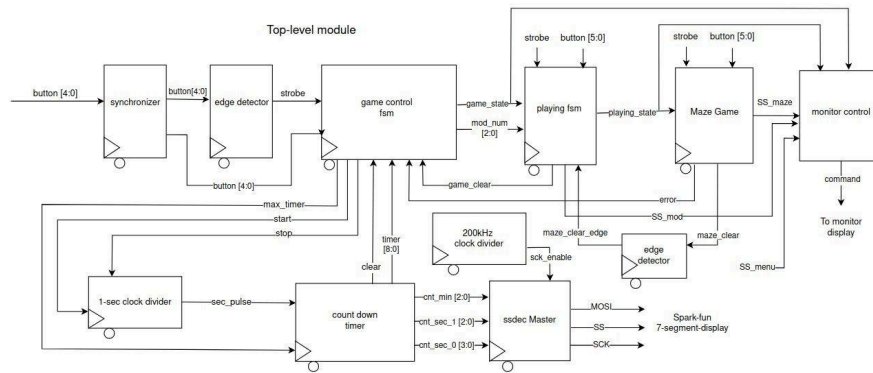Display 1.13 : Maze Image Processor Destination Logic
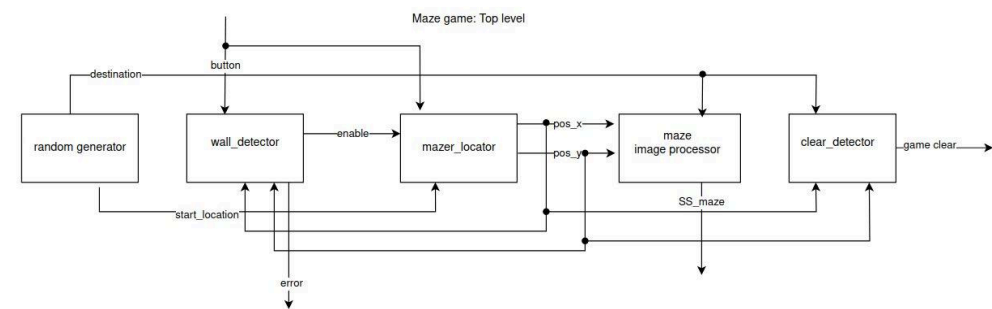
Intermediate Design Review
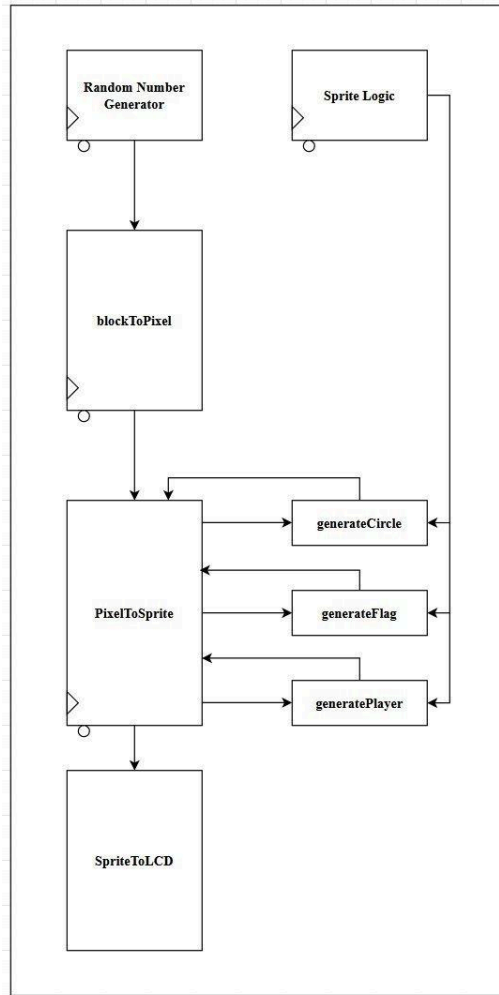

Display 2.1 Game Control FSM (Unchanged)

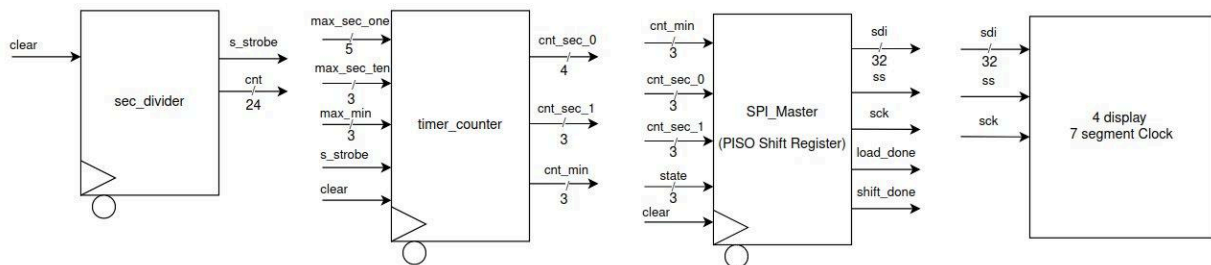Display 2.2 Playing FSM (Unchanged)

Display 2.3 : Unfinished Top Level Diagram (Needed the Monitor Control connected)

Display 2.4 : Rtl Diagram (Unchanged)

Display 2.5 : Extended Top RTL (Attempt for Parallel Display)

Display 2.6 : Timer RTL designs (Unfinished and not connected)

Final Product Rtls