Faculty of Engineering and Technology

Electrical and Computer Engineering Department

ADVANCED DIGITAL DESIGN ENCS3310

COURSE PROJECT

Prepared by

Leen Alqazaqi 1220380

Instructor

Dr. Abdellatif Abu-Issa

Section 4

BIRZEIT /January– 2026

1. FSM Diagram, Encoding, and timers

1.1. FSM diagram

We have three states for the Main Road: GREEN, YELLOW, and RED.

We have three states for the side Road: GREEN, YELLOW, and RED.

We have two states for the pedestrian walk: GREEN and RED

The state of the system is a combination of these states, so there is six states for the system:

1. M-GREEN: main road traffic light is green, all others are red
2. M-YELLOW: main road traffic light is yellow, all others are red
3. ALL-RED: All traffic lights are red
4. S-GREEN: side road traffic light is green, all others are red
5. S-YELLOW: side road traffic light is yellow, all others are red
6. P-GREEN: pedestrian walk light is green, all others are red

Below is the state diagram for the project before encoding:

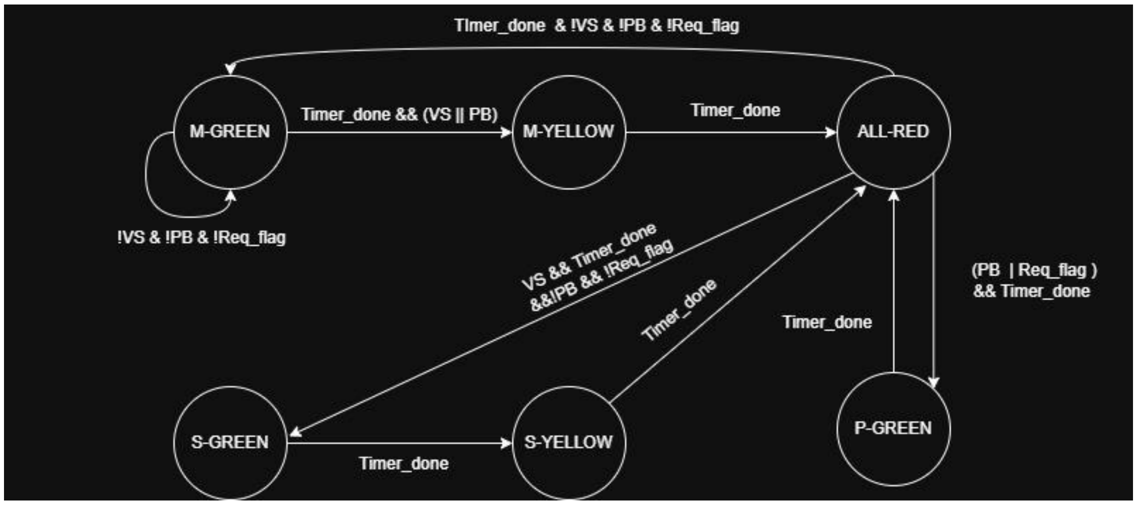


Figure: state diagram

- Timer_done is a flag used to check whether the required time for the current state has elapsed.
- VS is a signal from the side road vehicle sensor and is an input to the system.
- PB is a signal from the pedestrian walk button. it is set to 1 when pressed and 0 otherwise.

- Req_flag is a flag set to 1 if a pedestrian request occurs while the timer is not done. This ensures the request is remembered and served later when appropriate.

1.2. State Table

Then we created the state table to encode the state diagram then convert it into RTL.

Below is the state table:

Current State	Inputs (conditions)	Next State
M-GREEN	timer_done && (PB || VS)	M-YELLOW
M-GREEN	-	M-GREEN
M-YELLOW	timer_done	ALL-RED
ALL-RED	(PB || Req_flag)&& timer_done _	P-GREEN
ALL-RED	timer_done & VS & !PB &!Req_flag	S-GREEN
ALL-RED	timer_done & !PB & !VS &!Req_flag	M-GREEN
S-GREEN	timer_done	S-YELLOW
S-YELLOW	timer_done	ALL-RED
P-GREEN	timer_done	ALL-RED

1.3. FSM States Encoding and outputs encoding

States are encoded using **binary encoding**, while the FSM diagram is kept symbolic for clarity, state encoding provided below:

M-GREEN = 3'b000

M-YELLOW= 3'b001

ALL-RED= 3'b010

S-GREEN= 3'b011

S-YELLOW= 3'b100

P-GREEN= 3'b101

FSM State	Main Light	Side Light	Walk Light
M-GREEN = 3'b000	Green = 00	Red = 01	Red = 1
M-YELLOW= 3'b001	Yellow = 10	Red = 01	Red = 1

ALL-RED= 3'b010	Red = 01	Red = 01	Red = 1
S-GREEN= 3'b011	Red = 01	Green = 00	Red = 1
S-YELLOW= 3'b100	Red = 01	Yellow = 10	Red = 1
P-GREEN= 3'b101	Red = 01	Red = 01	Green = 0

1.4. Reference Model

Below is a table showing the reference model we used inside the test bench to compare our RTL with (expected FSM behavior).

PS	TD	VS	PB / Req	Expected NS	Explanation
M_GREEN	0	X	X	M_GREEN	Timer not done = stay
M_GREEN	1	0	0	M_GREEN	No requests
M_GREEN	1	1	0	M_YELLOW	Side vehicle request
M_GREEN	1	0	1	M_YELLOW	Pedestrian request
M_GREEN	1	1	1	M_YELLOW	Any request triggers transition
M_YELLOW	0	X	X	M_YELLOW	Timer not done
M_YELLOW	1	X	X	ALL_RED	Fixed transition
ALL_RED	0	X	X	ALL_RED	Timer not done
ALL_RED	1	X	1	P_GREEN	Pedestrian has priority
ALL_RED	1	1	0	S_GREEN	Side vehicle request
ALL_RED	1	0	0	M_GREEN	No requests
S_GREEN	0	X	X	S_GREEN	Timer not done
S_GREEN	1	X	X	S_YELLOW	Fixed transition
S_YELLOW	0	X	X	S_YELLOW	Timer not done
S_YELLOW	1	X	X	ALL_RED	Fixed transition
P_GREEN	0	X	X	P_GREEN	Timer not done
P_GREEN	1	X	X	ALL_RED	Pedestrian served

This reference model is used inside the testbench to automatically verify the RTL implementation.

Reset is also considered: when reset is asserted, the next state is **M-GREEN**, and this behavior is verified separately before functional testing begins.

1.5. Timing behavior

Each FSM state is associated with a fixed timing duration, implemented using a counter and a timer_done flag.
The system remains in the current state until the required time expires, ensuring safe traffic operation.

The timing behavior is defined as follows:

- **M-GREEN:**
  The main road remains green for **at least 60 seconds**. If no vehicle or pedestrian requests are present, the system stays in this state.
- **S-GREEN:**
  The side road green state lasts for **40 seconds**.
- **P-GREEN:**
  The pedestrian walk state lasts for **40 seconds**.
- **M-YELLOW and S-YELLOW:**
  Each yellow state lasts for **5 seconds** to allow safe transitions.
- **ALL-RED:**
  The all-red state lasts for **5 seconds** to ensure safety between traffic direction changes.

The timer_done signal becomes active when the counter reaches the specified limit for the current state, allowing the FSM to determine the next state.

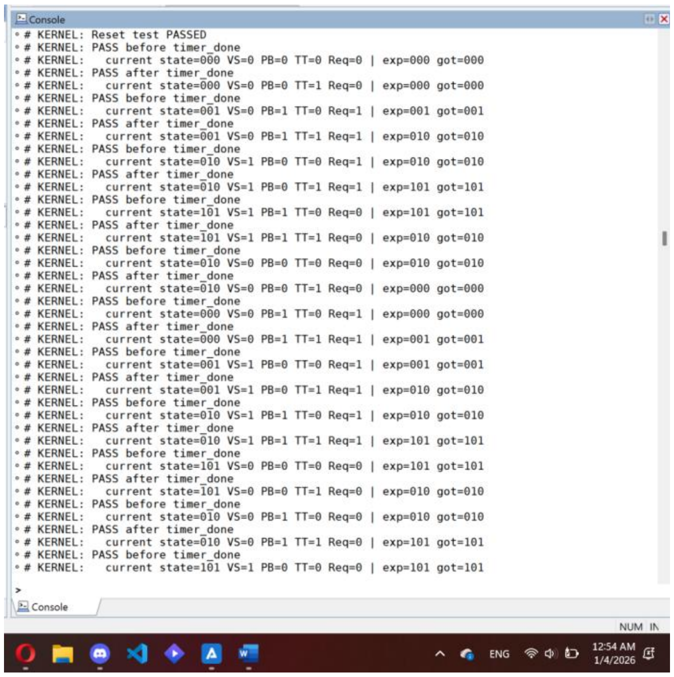
2. Results and waveforms

Simulation results show that the implemented FSM behaves correctly under all tested conditions. The traffic light controller transitions between states as specified, while respecting all timing constraints.

The testbench verifies system behavior both **before** and **after** the timer_done signal is active. Before timer expiration, the FSM remains in the current state, and after timer expiration, the FSM transitions to the correct next state according to the reference model.

All combinations of the vehicle sensor (VS) and pedestrian button (PB) inputs were tested. The results confirm that pedestrian requests are properly stored using the request flag and are served at the appropriate time. The reset functionality was also verified and correctly initializes the system to the **M-GREEN** state.

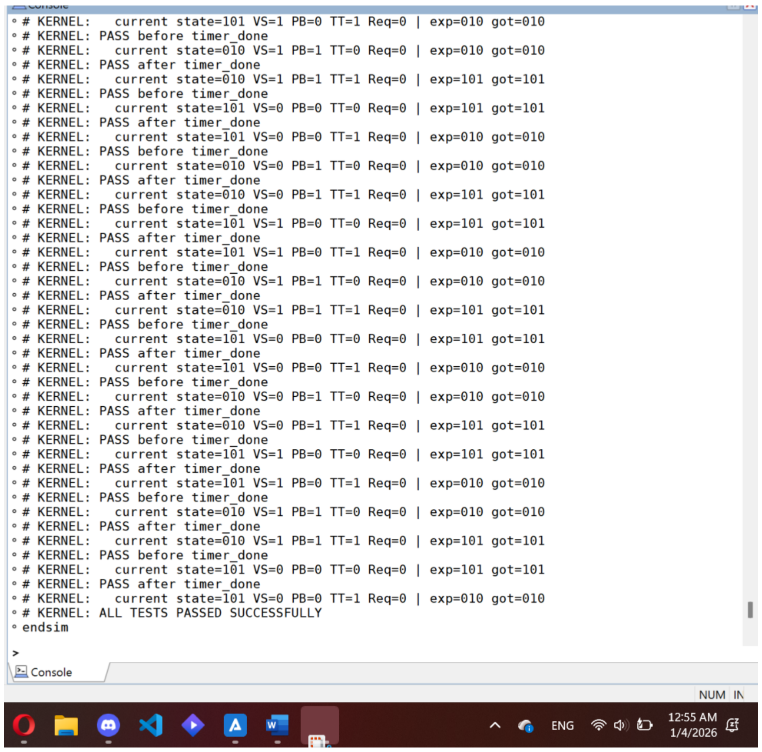
No mismatches were observed between the RTL outputs and the expected behavior generated by the reference model meaning the model is correct. Figures below show the results printing in the cmd after compiling and running the code (the top module is the tb /test bench one).



```
# KERNEL: Reset test PASSED
# KERNEL: PASS before timer_done
# KERNEL:   current state=000 VS=0 PB=0 TT=0 Req=0 | exp=000 got=000
# KERNEL: PASS after timer_done
# KERNEL:   current state=000 VS=0 PB=0 TT=1 Req=0 | exp=000 got=000
# KERNEL: PASS before timer_done
# KERNEL:   current state=001 VS=0 PB=1 TT=0 Req=1 | exp=001 got=001
# KERNEL: PASS after timer_done
# KERNEL:   current state=001 VS=0 PB=1 TT=1 Req=1 | exp=010 got=010
# KERNEL: PASS before timer_done
# KERNEL:   current state=010 VS=1 PB=0 TT=0 Req=1 | exp=010 got=010
# KERNEL: PASS after timer_done
# KERNEL:   current state=010 VS=1 PB=0 TT=1 Req=1 | exp=101 got=101
# KERNEL: PASS before timer_done
# KERNEL:   current state=101 VS=1 PB=1 TT=0 Req=0 | exp=101 got=101
# KERNEL: PASS after timer_done
# KERNEL:   current state=101 VS=1 PB=1 TT=1 Req=0 | exp=010 got=010
# KERNEL: PASS before timer_done
# KERNEL:   current state=010 VS=0 PB=0 TT=0 Req=0 | exp=010 got=010
# KERNEL: PASS after timer_done
# KERNEL:   current state=010 VS=0 PB=0 TT=1 Req=0 | exp=000 got=000
# KERNEL: PASS before timer_done
# KERNEL:   current state=000 VS=0 PB=1 TT=0 Req=1 | exp=000 got=000
# KERNEL: PASS after timer_done
# KERNEL:   current state=000 VS=0 PB=1 TT=1 Req=1 | exp=001 got=001
# KERNEL: PASS before timer_done
# KERNEL:   current state=001 VS=1 PB=0 TT=0 Req=1 | exp=001 got=001
# KERNEL: PASS after timer_done
# KERNEL:   current state=001 VS=1 PB=0 TT=1 Req=1 | exp=010 got=010
# KERNEL: PASS before timer_done
# KERNEL:   current state=010 VS=1 PB=1 TT=0 Req=1 | exp=010 got=010
# KERNEL: PASS after timer_done
# KERNEL:   current state=010 VS=1 PB=1 TT=1 Req=1 | exp=101 got=101
# KERNEL: PASS before timer_done
# KERNEL:   current state=101 VS=0 PB=0 TT=0 Req=0 | exp=101 got=101
# KERNEL: PASS after timer_done
# KERNEL:   current state=101 VS=0 PB=0 TT=1 Req=0 | exp=010 got=010
# KERNEL: PASS before timer_done
# KERNEL:   current state=010 VS=0 PB=1 TT=0 Req=0 | exp=010 got=010
# KERNEL: PASS after timer_done
# KERNEL:   current state=010 VS=0 PB=1 TT=1 Req=0 | exp=101 got=101
# KERNEL: PASS before timer_done
# KERNEL:   current state=101 VS=1 PB=0 TT=0 Req=0 | exp=101 got=101

>
```

Figure: test bench printed results 1



```
# KERNEL:   current state=101 VS=1 PB=0 TT=1 Req=0 | exp=010 got=010
# KERNEL: PASS before timer_done
# KERNEL:   current state=010 VS=1 PB=1 TT=0 Req=0 | exp=010 got=010
# KERNEL: PASS after timer_done
# KERNEL:   current state=010 VS=1 PB=1 TT=1 Req=0 | exp=101 got=101
# KERNEL: PASS before timer_done
# KERNEL:   current state=101 VS=0 PB=0 TT=0 Req=0 | exp=101 got=101
# KERNEL: PASS after timer_done
# KERNEL:   current state=101 VS=0 PB=0 TT=1 Req=0 | exp=010 got=010
# KERNEL: PASS before timer_done
# KERNEL:   current state=010 VS=0 PB=1 TT=0 Req=0 | exp=010 got=010
# KERNEL: PASS after timer_done
# KERNEL:   current state=010 VS=0 PB=1 TT=1 Req=0 | exp=101 got=101
# KERNEL: PASS before timer_done
# KERNEL:   current state=101 VS=1 PB=0 TT=0 Req=0 | exp=101 got=101
# KERNEL: PASS after timer_done
# KERNEL:   current state=101 VS=1 PB=0 TT=1 Req=0 | exp=010 got=010
# KERNEL: PASS before timer_done
# KERNEL:   current state=010 VS=1 PB=1 TT=0 Req=0 | exp=010 got=010
# KERNEL: PASS after timer_done
# KERNEL:   current state=010 VS=1 PB=1 TT=1 Req=0 | exp=101 got=101
# KERNEL: PASS before timer_done
# KERNEL:   current state=101 VS=0 PB=0 TT=0 Req=0 | exp=101 got=101
# KERNEL: PASS after timer_done
# KERNEL:   current state=101 VS=0 PB=0 TT=1 Req=0 | exp=010 got=010
# KERNEL: PASS before timer_done
# KERNEL:   current state=010 VS=0 PB=1 TT=0 Req=0 | exp=010 got=010
# KERNEL: PASS after timer_done
# KERNEL:   current state=010 VS=0 PB=1 TT=1 Req=0 | exp=101 got=101
# KERNEL: PASS before timer_done
# KERNEL:   current state=101 VS=1 PB=0 TT=0 Req=0 | exp=101 got=101
# KERNEL: PASS after timer_done
# KERNEL:   current state=101 VS=1 PB=0 TT=1 Req=0 | exp=010 got=010
# KERNEL: PASS before timer_done
# KERNEL:   current state=010 VS=1 PB=1 TT=0 Req=0 | exp=010 got=010
# KERNEL: PASS after timer_done
# KERNEL:   current state=010 VS=1 PB=1 TT=1 Req=0 | exp=101 got=101
# KERNEL: PASS before timer_done
# KERNEL:   current state=101 VS=0 PB=0 TT=0 Req=0 | exp=101 got=101
# KERNEL: PASS after timer_done
# KERNEL:   current state=101 VS=0 PB=0 TT=1 Req=0 | exp=010 got=010
# KERNEL: ALL TESTS PASSED SUCCESSFULLY
endsim

>
```

Figure: test bench printed results 2

Below is the waveform generated by the RTL code:

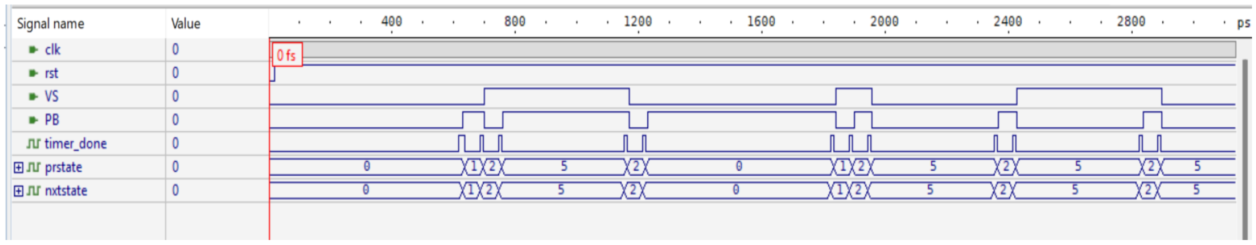


Figure: WaveForm

As we can observe, the system starts in the **M-GREEN** state, which confirms that the active-low reset operates correctly. When the input signals **VS** and **PB** change, no state transition occurs as long as **timer_done** remains low, and the FSM stays in the same state. Once **timer_done** is asserted, a state transition is triggered. It is also noteworthy that nxtstate **precedes** prstate, which is clearly visible when zooming in on the waveform, confirming that the next state is computed combinationally and then registered on the following clock edge.

3. Code

The project source files are attached with the report in the submission message, including the RTL module and the self-checking testbench.