UNIVERSITI TEKNOLOGI MALAYSIA

# Zebra Crossing Simulator on CPLD

Muhammad Mun'im Ahmad Zabidi, Musa Mohd Mokji, Izam Kamisian, Norhafizah Ramli

**Abstract**
A zebra crossing is a type of pedestrian crossing used in many places around the world. It is marked with black and white stripes, resembling the coat of a zebra. Vehicles are supposed to stop so that people can walk across. This document explores the state machine that regulate car traffic.

[1] *Department of Electronic and Computer Engineering, School of Electrical Engineering, Faculty of Engineering*
*__Corresponding author__: munim@utm.my

## 1. Scenario

Figure 1 shows a single lane road with a pedestrian crossing. Road traffic and pedestrians are controlled by light signals. Pedestrians may request to cross the road by pressing a walk button $B$.

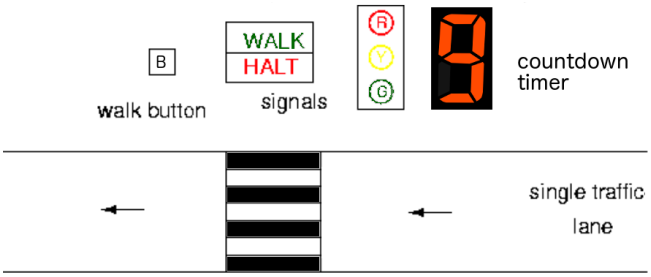



***Figure 1.*** *Conceptual view of zebra crossing.*

The signals that control the lights are defined at the top level entity in Figure 2. The control has a single input button. Three output signals control the car facing lamps: green, yellow and red. Three more outputs control the pedestrian facing lamps, halt, walk, and a set of wires to the 7-segment display.

The traffic lights are controlled by a finite state machine (FSM), with state diagram in Figure 3. The controller has

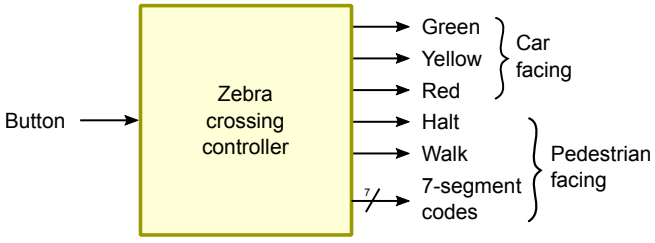



***Figure 2.*** *Conceptual view of zebra crossing controller.*

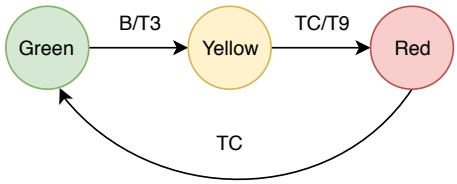



***Figure 3.*** *Zebra crossing state diagram.*

3 states based on the colors visible to the car driver. The default state is Green.

**State Green** The car-facing light is green and the pedestrian facing light is HALT. When someone wants to cross the street, he/she presses the button B. This causes the FSM to jump to state Yellow.

**State Yellow** The car-facing light is yellow and the pedestrian facing light is still HALT. When 3 seconds is up, the FSM jumps to state Red.

**State Red** The car-facing light is red and the pedestrian facing light is WALK. The pedestrian crosses the street at this time. A countdown timer is turned to inform the pedestrian how many seconds are remaining. When 9 seconds is up, the FSM reverts to the green state.

The controller consists of 3 modules: FSM, timer and BCD-to-7-segment decoder, as illustrated in Figure 4. The timer can count 3 down to 0, or 9 down to 0. The output of the timer is always available for debugging.

The signals that affect the state diagram are as follows:

**B (Button)** Pressed by pedestrian to cross the road. This input is only checked in Green state.

**T3** Triggers the timer to count down for 3 seconds. It is activated while transiting from Green state to Yellow

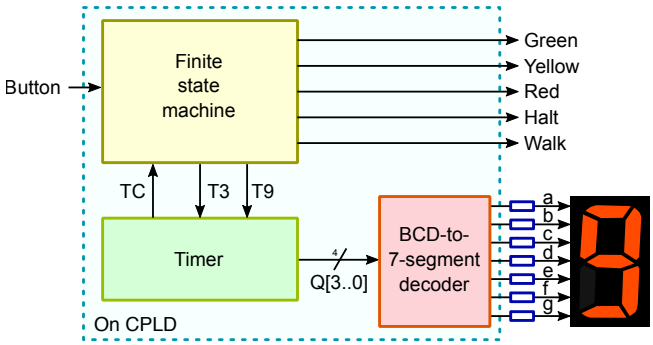



***Figure 4.*** *Main modules of zebra crossing controller.*

state, i.e. this is Mealy type output. Activating T3 lets the FSM stay in Yellow state for 3 seconds.

**T9** Triggers the timer to count down for 9 seconds. It is activated while transiting from Yellow state to Red state, i.e. this is Mealy type output. Activating T3 lets the FSM stay in Red state for 9 seconds.

**TC (Terminal Count)** : High when timer count reaches 0. The FSM stays in Yellow or Red states as long as TC is low.

## 2. Project Settings

1. Create a new project.

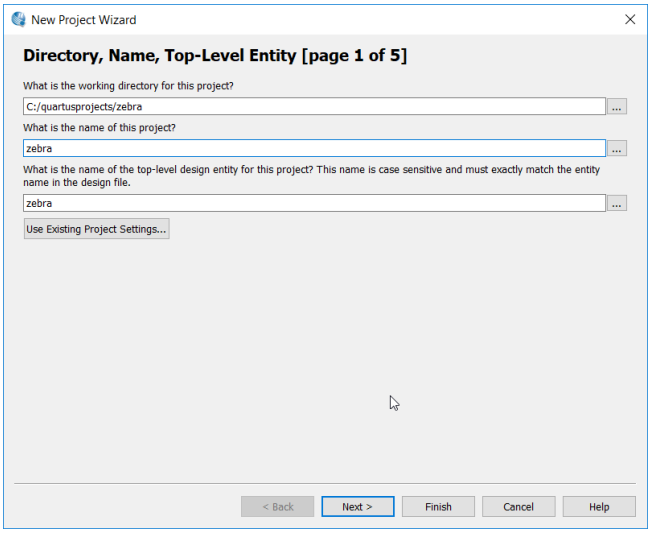



***Figure 5.*** *New project.*

2. Place the project files in the appropriate folder.
3. Click Finish.

## 3. Timer Module

1. Choose **File ➥ New ➥ Block Diagram/Schematic File**.
2. In the Block Diagram editor, place a new **lpm_counter** module.

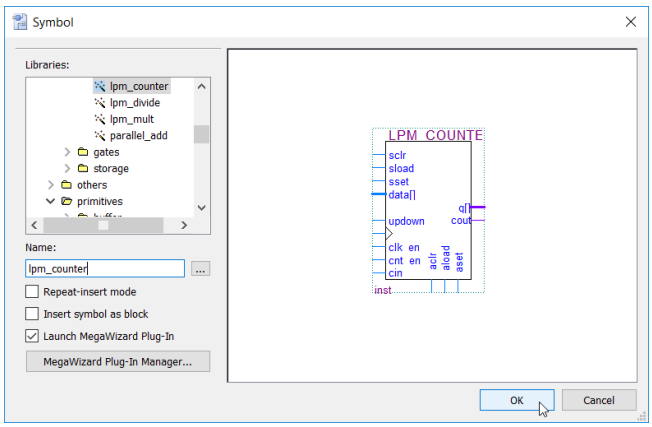



***Figure 6.*** *New project.*

3. In Page 2c of the MegaWizard, choose Verilog. Then click Next >.

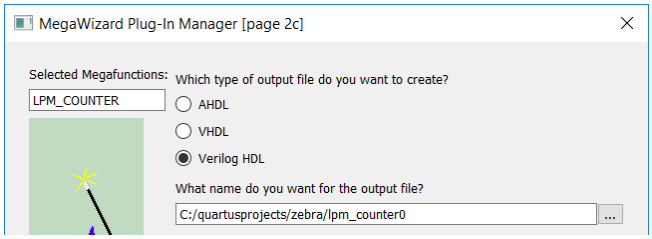



***Figure 7.*** *Choose Verilog.*

4. In Page 3 of the MegaWizard, select 4 bit output and choose Down Only direction. Then click Next >.

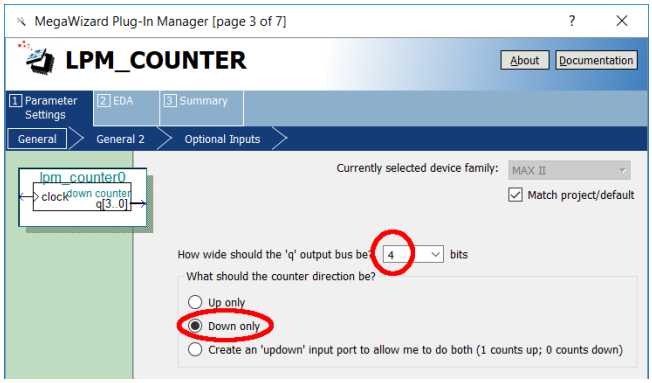



***Figure 8.*** *Select 4 bit output and choose Down Only direction.*

5. In Page 4 of the MegaWizard, click Carry-out. Then click Next >.

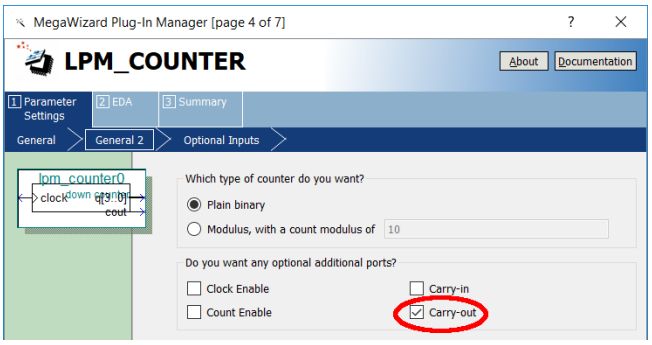



***Figure 9.*** *Click Carry-out.*

6. In Page 5 of the MegaWizard, add a Load input. Then click Finish. On the next page, click Finish again.

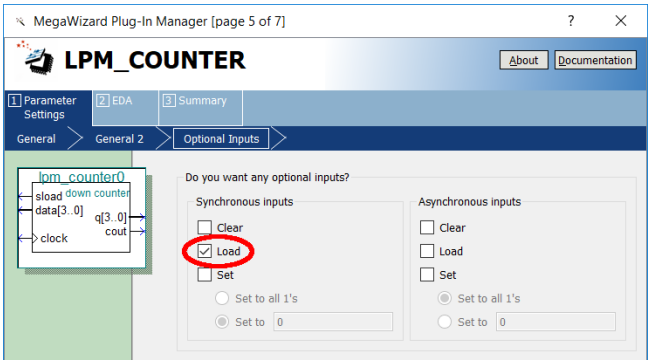



***Figure 10.*** *Click Load.*

7. Place the resulting symbol anywhere convenient.

8. Place the first **lpm_constant** module.

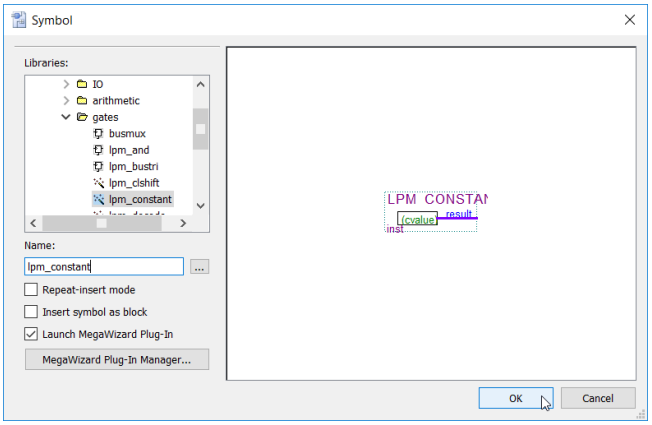



***Figure 11.*** *Placing one out of two lpm_constant symbols.*

9. Set the size to 4 bits and constant value of 3. Click Finish.

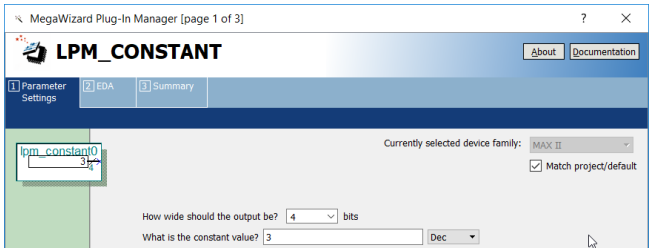



***Figure 12.*** *Constant of 3*

10. Place another **lpm_constant** module. Set the size to 4 bits and constant value of 9. Click Finish.

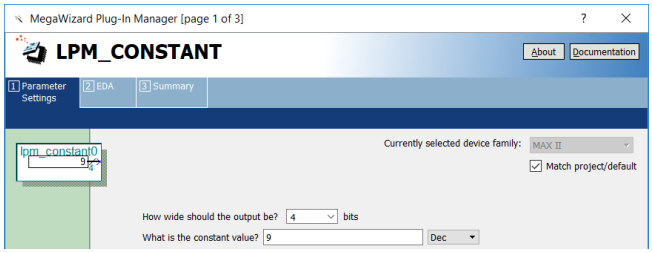



***Figure 13.*** *Constant of 9*

11. Add a **busmux**, an OR gate, 3 input ports and 2 output ports like in Figure 14.

12. Set the timer circuit as top-level entity and compile it.

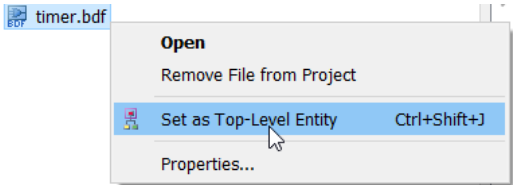



***Figure 15.*** *Set timer as top-level.*

13. Create a waveform file and test the timer. Figure 16 shows the relation between T3/T9 triggers versus TC and counter values.

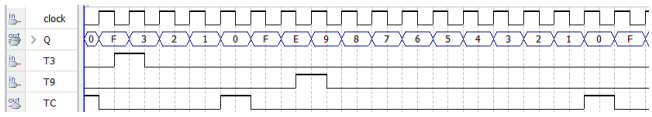



***Figure 16.*** *Timer simulation.*

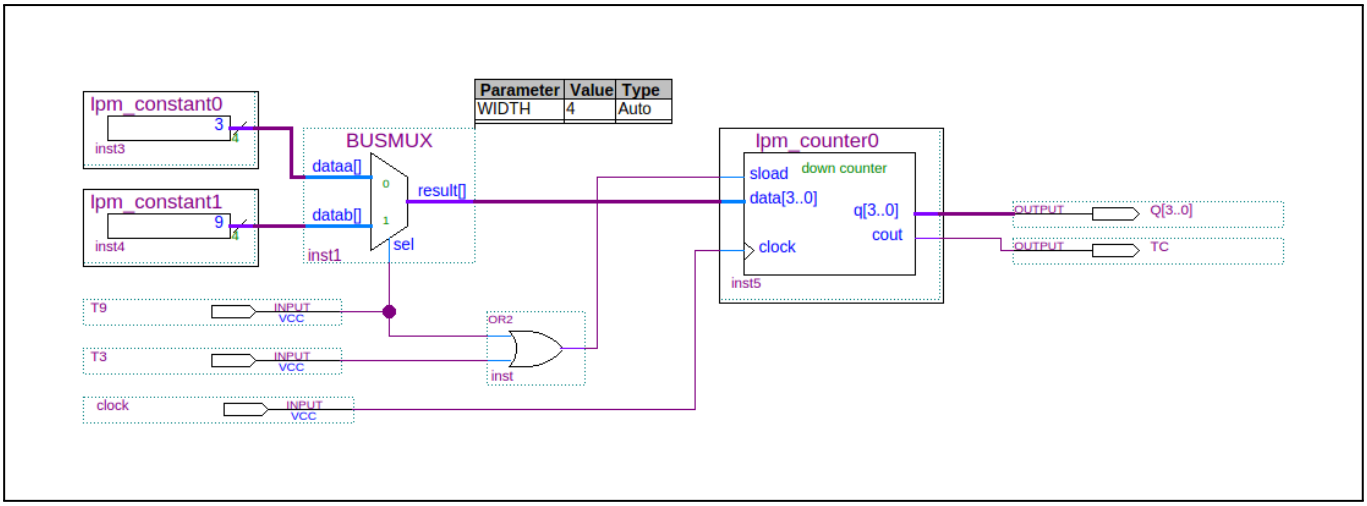



***Figure 14.*** *The timer module.*

14. When no errors, convert the circuit to a symbol file.

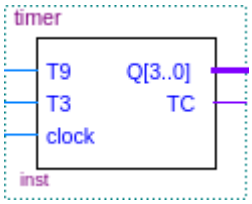
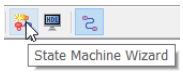



***Figure 17.*** *Timer symbol.*

## 4. FSM Module

1. Choose **File ➥ New ➥ State Machine File**.

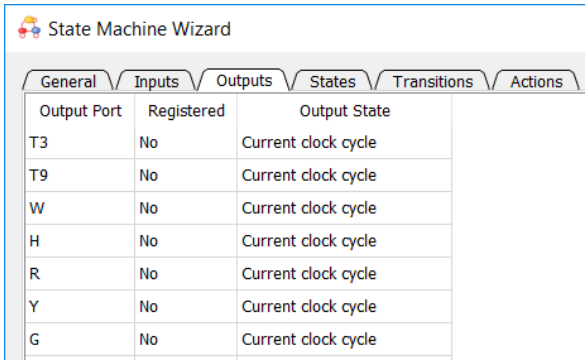
2. In the State Machine Editor, click the State Machine Wizard.



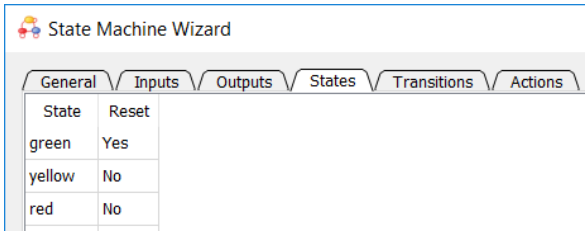

3. Skip the General tab of the State Machine Wizard. Nothing to change there.

4. In Inputs tab of the State Machine Wizard, add TC and B to the list of input ports. The order of ports does not matter.


5. In Outputs tab of the State Machine Wizard, add T3, T9, W, H, R, Y and G to the list of output ports. The order of ports does not matter.




6. In States tab of the State Machine Wizard, add green, yellow and red to the list of states.



7. In Transitions tab of the State Machine Wizard, add the following list of state-to-state transitions. The order does not matter.

    - green goes to yellow when B is received
    - yellow goes to red when TC is received
    - red goes to green when TC is received

State Machine Wizard

General | Inputs | Outputs | States | Transitions | Actions

| Source State | Destination State | Transition (In Verilog or VHDL 'OTHE |
|---|---|---|
| green | yellow | B |
| yellow | red | TC |
| red | green | TC |

8. In Actions tab of the State Machine Wizard, add the following list of outputs. The order does not matter.

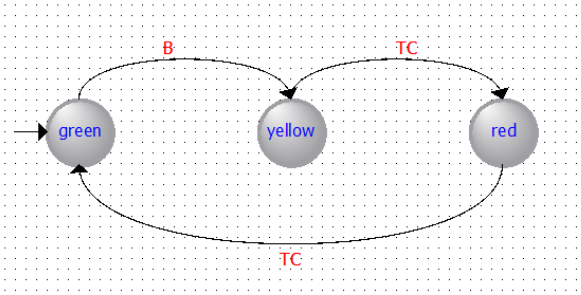
    - T3 is activated in state green if B is received
    - T9 is activated in state yellow if TC is received
    - G and H are activated unconditionally during state green
    - Y and H are activated unconditionally during state yellow
    - R and W are activated unconditionally during state red

State Machine Wizard

General | Inputs | Outputs | States | Transitions | Actions

| Output Port | Output Value | In State | Additional Conditions |
|---|---|---|---|
| T3 | B | green | |
| T9 | TC | yellow | |
| G | 1 | green | |
| Y | 1 | yellow | |
| R | 1 | red | |
| H | 1 | green | |
| H | 1 | yellow | |
| W | 1 | red | |

9. Click OK when all information has been entered.




10. Generate the HDL code.

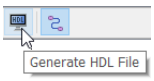



***Figure 18.*** *Generate HDL.*

11. Set the state machine file as the top-level entity.

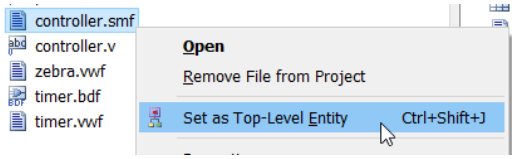



***Figure 19.*** *Set controller as top level entity.*

12. Compile the code.

13. Simulate the controller.

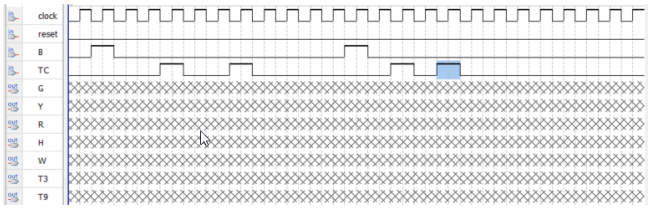



***Figure 20.*** *Controller simulation input.*

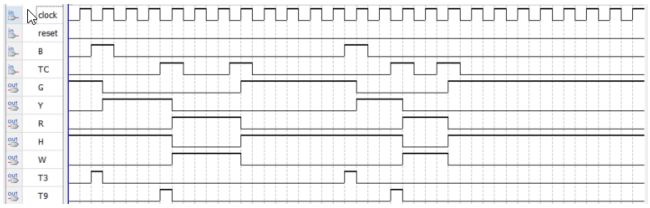



***Figure 21.*** *Controller simulation output.*

14. When no errors in operation, save as a Symbol file.

| Parameter | Value | Type |
|---|---|---|
| green | 0 | Signed Integer |
| yellow | 1 | Signed Integer |
| red | 2 | Signed Integer |

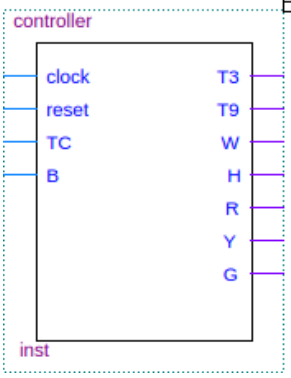



***Figure 22.*** *Controller symbol.*

## 5. Integration & Simulation

1. In a new block diagram file, insert the timer and controller symbols and connect according to Figure 23.
2. Simulate it. You should get Figure 24.

That concludes the construction of modules that can be simulated.

## 6. CPLD

By adding two more modules, the zebra crossing circuit is ready to programmed into a CPLD/FPGA.

The first module to add is the prescaler. This modules slows the clock from 50 MHz to about 1 Hz. The second module is the 7447 BCD-to-7-segment decoder. The output of the decoder provides a convenient human interface but the timing waveform output is not easy to visualize. It is better not to simulate the 7447 module output.

1. Create a Verilog file and enter this code.

```
module prescaler( input clkin, output clkout );
  reg[25:0] count;
  always @(posedge clkin) count <= count + 1;
  assign clkout = count[25];
endmodule
```

2. Save as **prescaler.v** and compile it.
3. Save the prescaler as Symbol file.
4. In a new top-level (non-simulatable) file, Figure 25, insert the prescaler and the 7447 BCD-to-segment decoder modules. Just follow the diagram. Then compile it.
5. In the Pin Assignment menu, set clock to Pin 64. Set all other pins according to the components on your breadboard.
6. Place a switch input, a 7-segment display, 5 color LEDs, and 12 current limiting resistors on your breadboard. Connect them to the CPLD according to Figure 26.

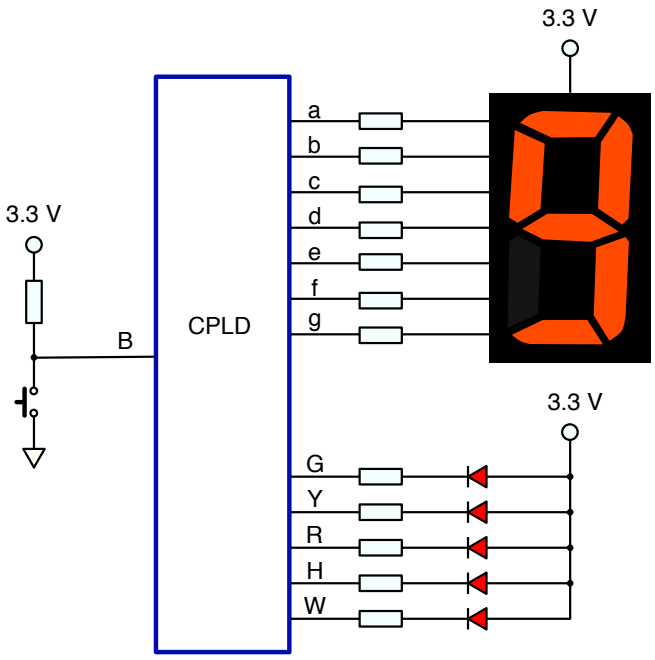



***Figure 26.*** *Zebra controller schematic.*

7. Program the circuit into your CPLD. Run it!

## References

[1] Munim Zabidi, Izam Kamisian, and Ismahani Ismail. *The Art of Digital Design*. 2019.

[2] *Introduction to the Quartus® II Software*. Version 10. Altera. 2010. URL: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/manual/intro_to_quartus2.pdf.

[3] *My First FPGA Design Tutorial*. Altera. July 2008. URL: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/tt/tt_my_first_fpga.pdf.

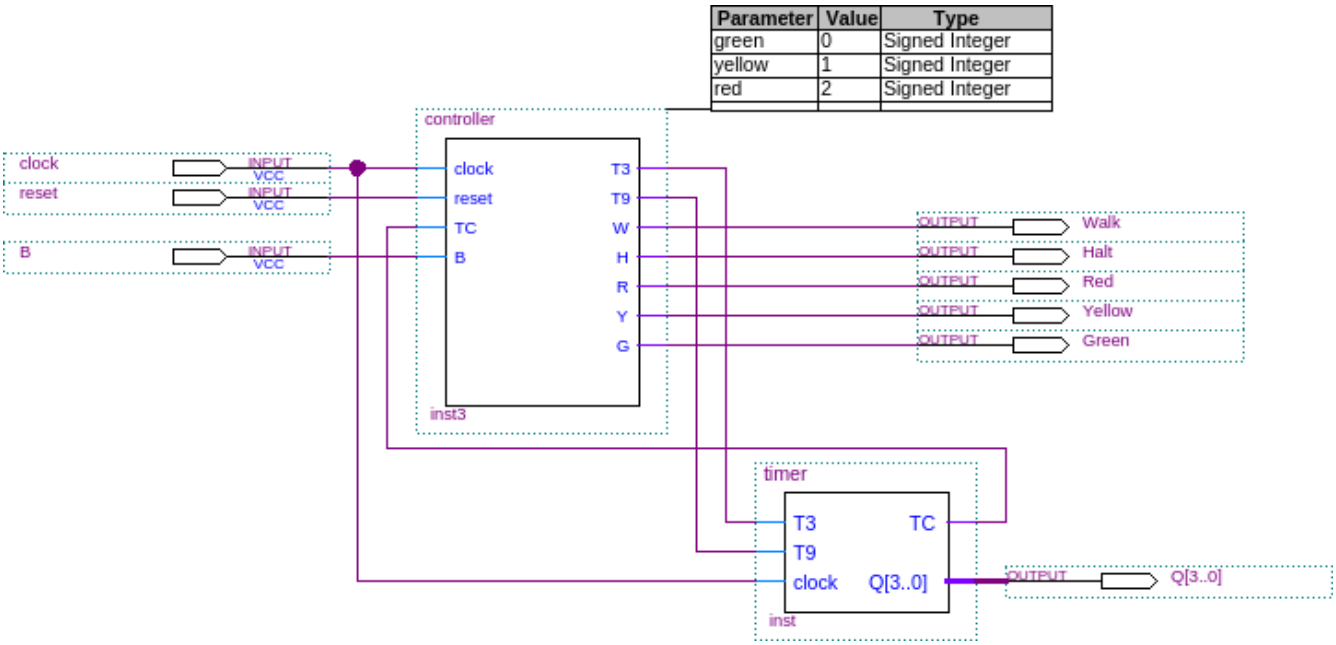



***Figure 23.*** *Top level zebra crossing circuit using full speed clock. The simulation is Figure 24.*

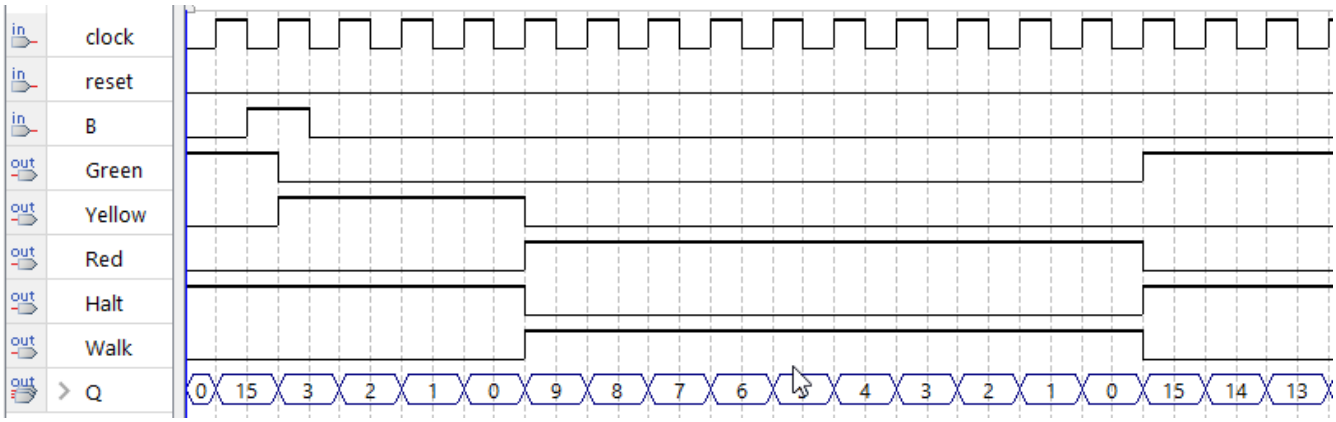



***Figure 24.*** *Zebra crossing simulation. Seven-segment display not added yet.*

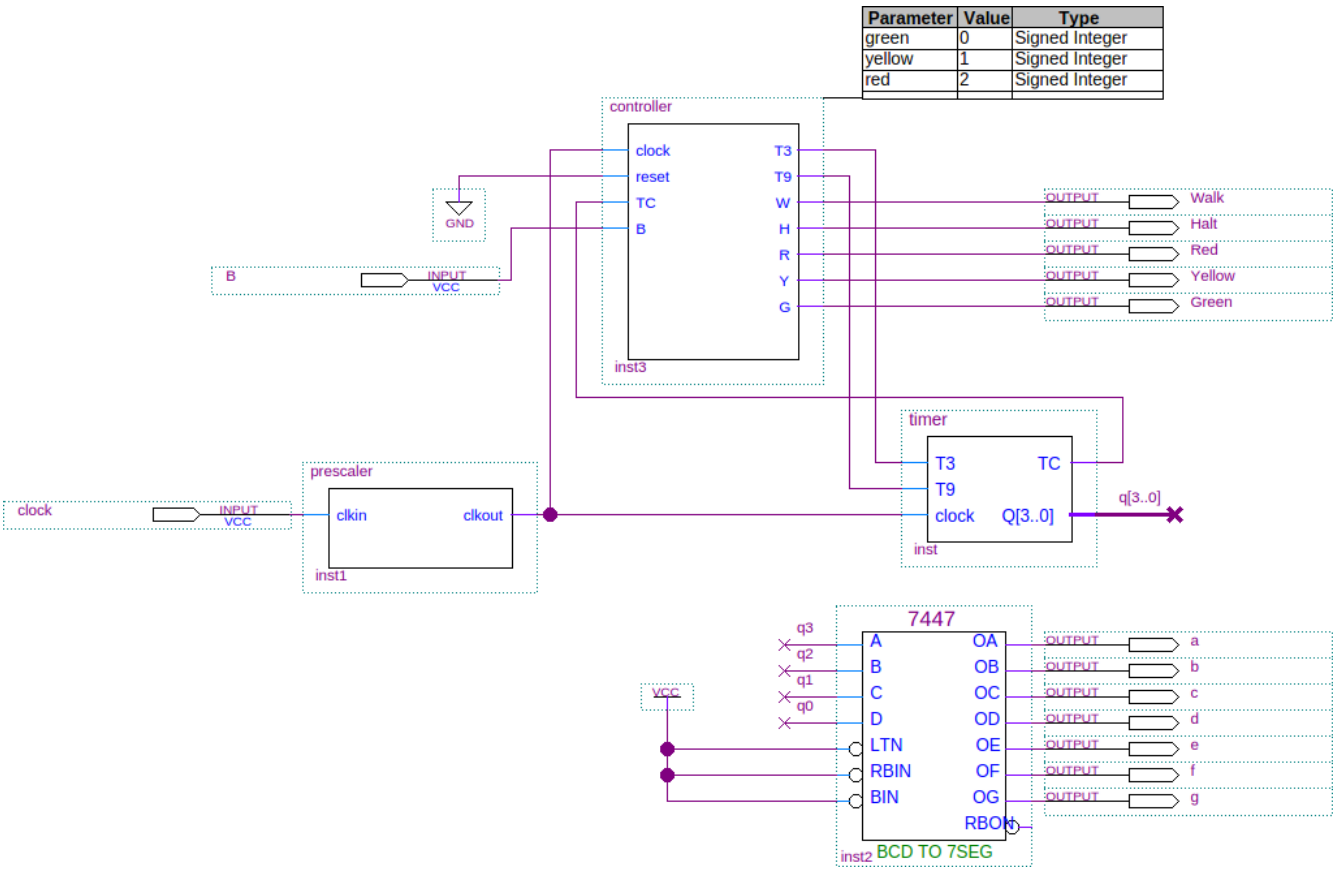



***Figure 25.*** *Zebra crossing circuit with prescaler and 7447 modules. You cannot simulate this circuit. You can only program it to the CPLD/FPGA.*