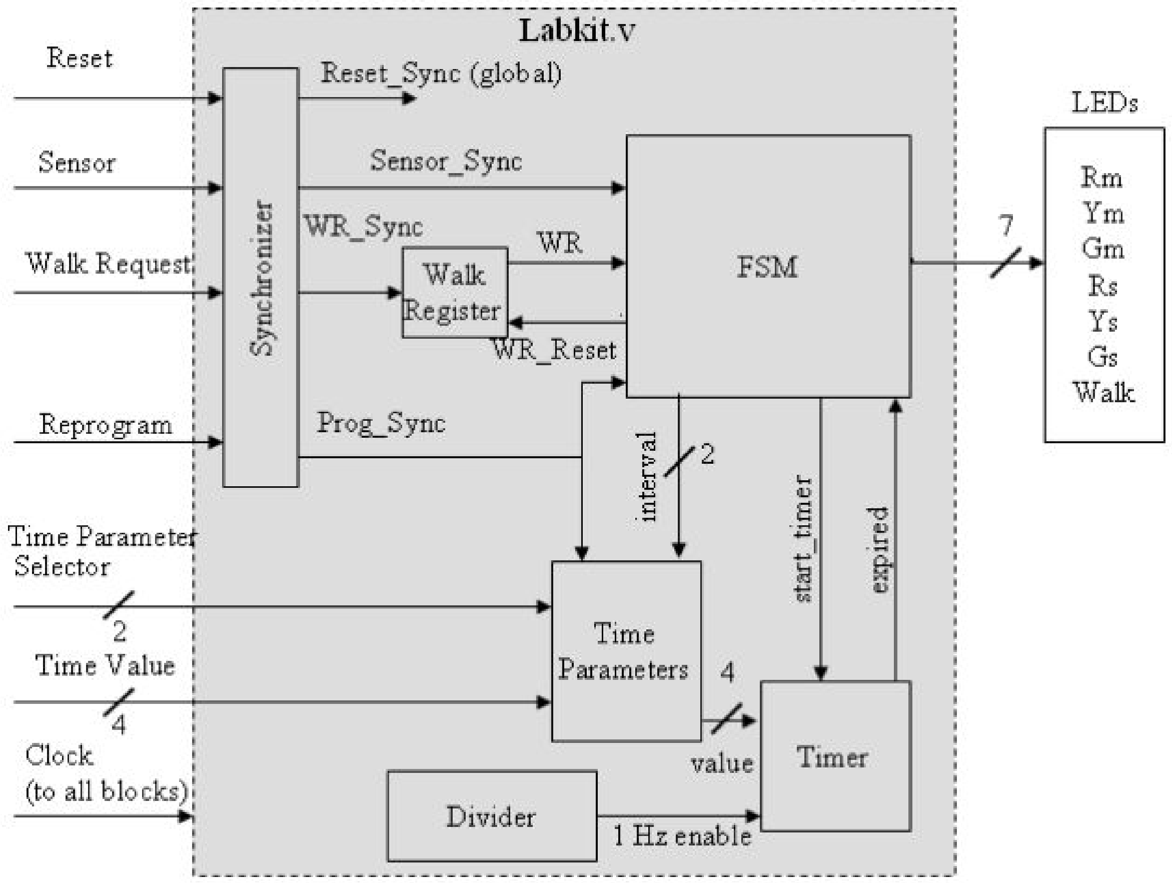
Traffic Light Controller

CS4362 Hardware Description Language

MHM Amrie
150026G

1. Introduction

In this lab, we are supposed to implement a traffic light controller. The design methodology is below described.



All the functionalities of the traffic light controller were segmented as following modules to facilitate the traffic light controller error-prone and convenient of implementing. A details description of each module is given with implementation methodology.

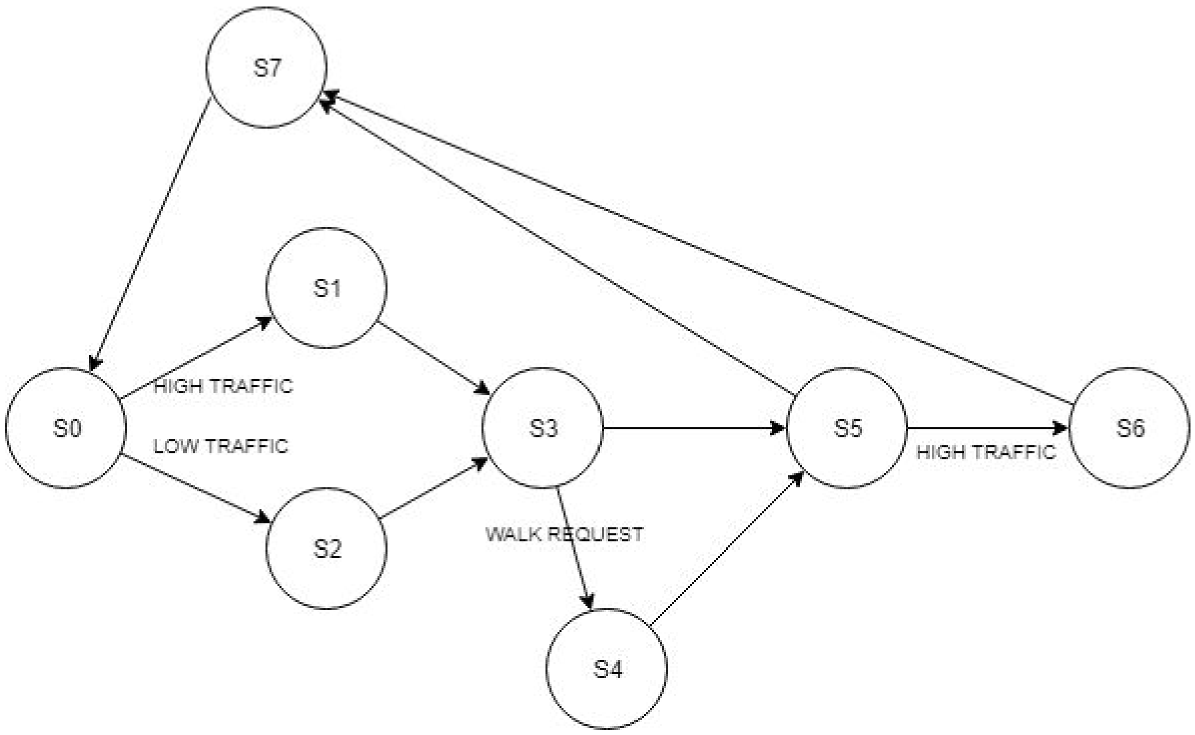
FSM (Finite State Machine)

Description and implementation

Main functions on the Traffic Light Controller are implemented on FSM module. It contains 8 states to govern traffic light controller as per the specifications. In the code implementation at each time, it will check the “Expired” signal line to make a decision about make any state changes or keep the lights according to the states given in the specification.

IO states

Inputs: Walking register value, Sensors signal, Reset signal, expired signal and reprogram
Outputs: Output LEDs, Reset walking register, time value interval



State diagram

State	Description	Next State(Condition)

S0	● Main Green Light(ON)
	● Side Red Light (ON)
	● Reset (OFF)
	● Waiting time: tBASE
		S1(High traffic)
		S2(Low traffic)

S1	● Main Green Light(ON) ● Side Red Light (ON) ● Reset (OFF) ● Waiting time: tEXT	S3

S2	• Main Green Light(ON) • Side Red Light (ON) • Reset (OFF) • Waiting time: tBASE	S3

S3	• Main Yellow Light (ON)
	• Side Red Light (ON)
	• Reset (OFF)
	• Waiting time: tYEL
		S4(Walk request)
		S5(No walk request)

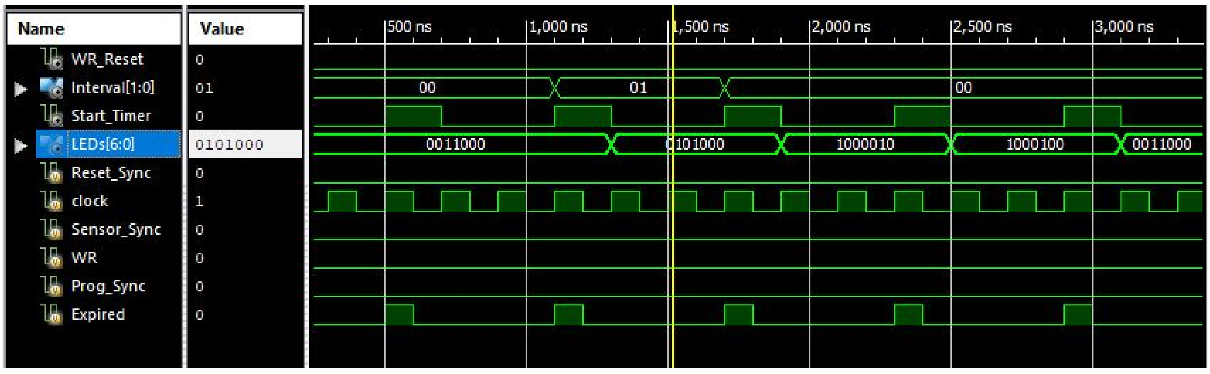
S4	• Walking Lamp (ON) • Reset (OFF) • Waiting time: tEXT	S5

S5	• Main Red Light (ON)
	• Side Green Light (ON)
	• Reset (OFF)
	• Waiting time: tBASE
		S6(High traffic)
		S7(Low traffic)

S6	• Main Red Light (ON) • Side Green Light (ON) • Reset (OFF) • Waiting time: tEXT	S7
S7	• Main Red Light(ON) • Side Yellow Light (ON) • Reset (OFF) • Waiting time: tYELL	S0

States description

Test Bench Simulation



The above figure demonstrates the functionality of the FSM. At each detection in the “Expired”, input line state should be changed as well as the light.
E.g.: state changes between 0011000(S0) to 0101000(S2).

Timer

Description and implementation

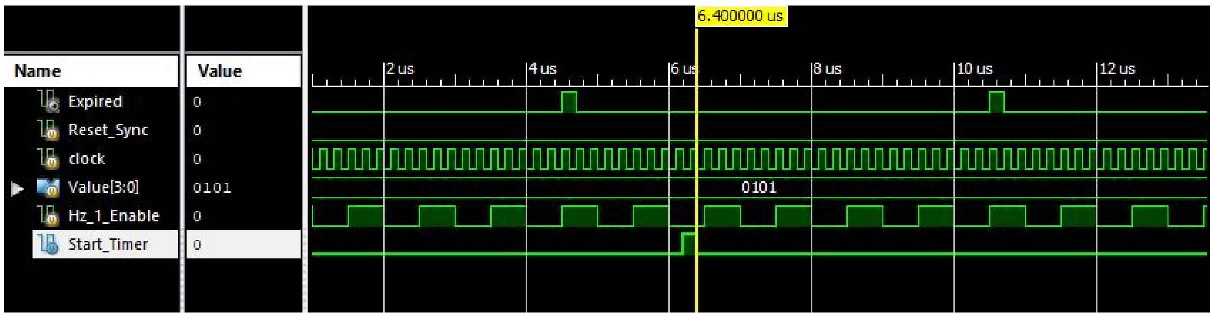
The main functionality of this is to count the time in seconds to facilitate FSM module to change the state with time.

IO states

Input : Start Timer, Value(4bit), 1 Hz Enable,
Output : Expired

Since, there are 2 clocks involved implementation consist of 2 components, one to track the Start Timer and release Expired output line another one to facilitate count the exact time which was provided by the Time Parameter module.

Test Bench Simulation



As shown from the above figure when Start Timer input received a signal from the FSM Timer module start to count some number of time according to the value and release expired as an output for the FSM. In this case, value is 5 and after 5 1 Hz clock tick it will release Expired to the FSM.

Divider

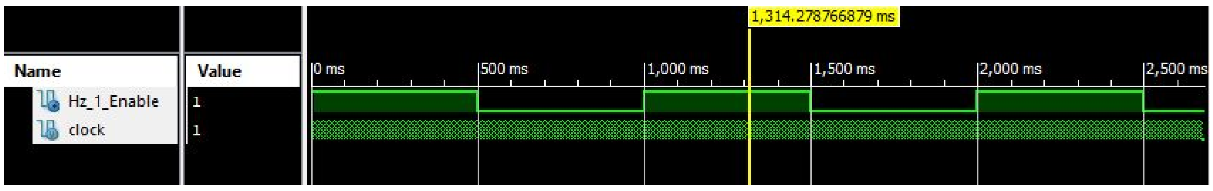
Description and implementation

It is possible to use any frequency as a clock but, when the time is measured there should be a proper way to do it. Dividers' main function is to divide or convert the high frequency to lower frequency like 1Hz. In the code implementation, it will count some particular number of positive edges to calculate to detect 1 second and make the 1Hz clock.

IO states

Input: clock
Output: 1 Hz clock

Test Bench Simulation



From the above figure, it is convenient to see the conversion of the high frequency to 1Hz frequency.

Time Parameter

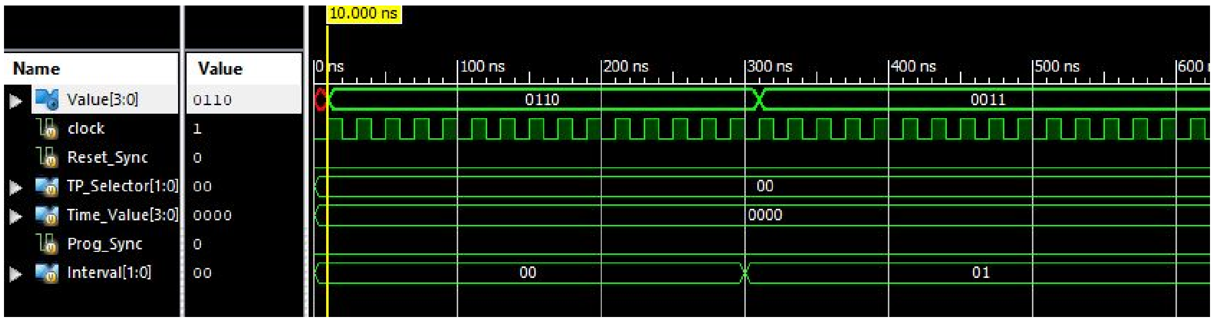
Description and implementation

Time parameter for Green, Red and Yellow light can set through this module. For that, there are 3 inputs (Prog Sync, Time parameter selector and Time Value) by these inputs dynamically change the time parameter according to the situation. In the case of not setting up default values will serve as the time parameter. In the code implementation, at each clock tick, it will check whether the user needs to change the time parameter, failure will cause to pass the default value as per FSM request through the Interval input.

IO states

Inputs: Reprogram, Selecting time interval, setting time interval, time value
Outputs: time value of requested interval

Test Bench Simulation



The above figure is a demonstration of passing value according to a different interval.
E.g.: For Interval 00 the output is 0110(default base value)
For Interval 01 the output is 0011(default Ext value)

Synchronizer

Description and implementation

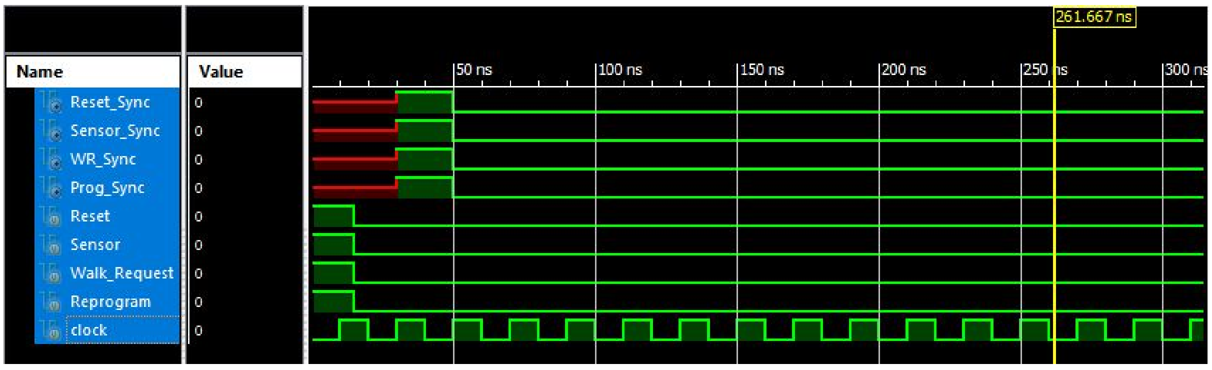
Synchronizer module is supposed to make all the input synchronized with the corresponding clock. All the inputs which are want to be synchronized with the clock will be saved into a register with the next clock tick all inputs that are saved in the register will be released as synchronized with the clock.

IO states

Inputs: Reset, Sensor for traffic, Walk request, Reprogram

Outputs: All synchronized data with the clock

Test Bench Simulation



As the above figure, it is clear that all input (Reset, Walk request, Reprogram) are synchronized with the next clock tick even though those are initiated a different instance of the time.

Walk Register

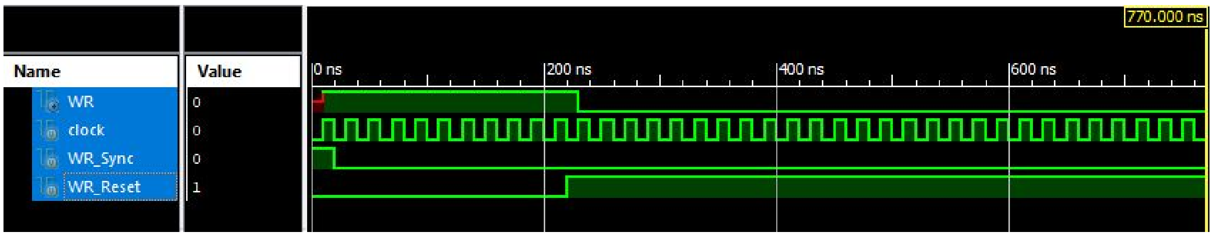
Description and implementation

The Walk Register allows users to set a walk request at any time. There is also a signal controlled by the finite state machine that will be able to reset the register at the end of the actual walk cycle. In code implementation, walk request is saved on a register until reset from the FSM, walk register will maintain its state as high.

IO states

Inputs: Walk request, reset for the FSM
Output: walking register value

Test Bench Simulation



In this simulation, after reset from the FSM, WR value is shown as low that means no pedestrians are available.