Implementation of Car Parking System Using VHDL

Gattadi Vinatha[#1],Kayathi Pooja [#2], R.Poojitha[#3], D.Prudhvi[#4]

#1 Assistant Professor, St.Martins Engineering college, Dhulapally(v), kompally,Secunderabad-500100 Telangana state, India

#2-#4 B.Tech Scholar, St.Martins Engineering college, Dhulapally(v), kompally,Secunderabad-500100 Telangana state, India

[1]vinnu251@gmail.com
[2]poojakayathi21@gmail.com
[3] dontha.prudhvi175@gmail.com
[4]poojithadolly5@gmail.com

***Abstract*— The growing population in India has created many problems - one of the challenging ones being carparking troubles which we confront everyday.This has created a specific need for a system whichcan control and guide the process of car parking. We are implementing car parking system which are muchin vogue – a method of automatically[5]parking and retrieving cars that typically use a system of slots and signalling objects for theretrieve. They serve advantages like safety, saving space,time and fuel as one does not have todrive around for finding empty space . One can design a car parking system using many domains but we have chosen VHDL(VHSIC-HDL) - Very High Speed Integrated Circuit Hardware Description Language domain as it allows the user to implement complex functions and this tool converts described function to logic gates so that one can realize the output better. VHDL is more verbose than similar tools.**

***Keywords*—- VHDL, logic gates, car parking system, simulation**

I. Introduction

Present days usage of motor vehicles are increased day by day, it causes the pollution, traffic congestion and parking problems. To overcome this problem, Parking System is implemented in Finite State Machine(FSM) using VHDL Language.(Very High Speed Integrated Circuit Hardware Description Language). The system has main important modules i.e., identification module and two sensors which are front_ sensor and back _sensor. Identification module which means the security key(password) to enter into the parking slot. In this pragmatic world, several tasks were performed by every individual without being evasive. So, in order to sort out all the tasks for the efficient usage of time, wise steps should be taken to curb the wastage of time at unproductive areas such as at the most frequently performed action, which is the parking of vehicles. So, our paper provides a better alternative to have an efficient usage of time at parking correlated with the security issue which serves at its best. The major discussion involves the following solutions given below for the efficient usage of time which doesn't spare much time for parking purpose and also in order to have a safe park without involving any sort of crashes. Systematic parking with security[2] is the main motto. Security includes the usage of password at the time of park. This VHDL project presents a car parking system in VHDL using Finite State Machine (FSM). VHDL code and test bench for the car parking system is explained in future chapter.

II. ProposedMethod

The system to be designed is a very simple one and its purpose is to introduce the idea of converting a FSM into VHDL. This FSM has four states: A, B, C, and D. The system has one input signal called P, and the value of P determines what state the system moves to next. The system changes state from A to B to C to D as long as the input P is high (1). If P is low, and the system is in state A, B, or C, the state is not changed. If the system is in state D, it changes to B if P is high and to A if P is low. The system also has an output called R which is 1 if in state D, otherwise it is a 0. Figure 1 is the diagram for the FSM, but first here are a few notes about this diagram:

- The circles represent the states
- Arrows between the circles represent the rules for changing from state to state. For example, in this system, the state machine moves from state A to state B if the input P is equal to 1 (otherwise it remains in state A)

- The information underneath the line in the circle represents the output value when in each state.
- The arrow coming from "nowhere" to the A indicates that A is the initial state.

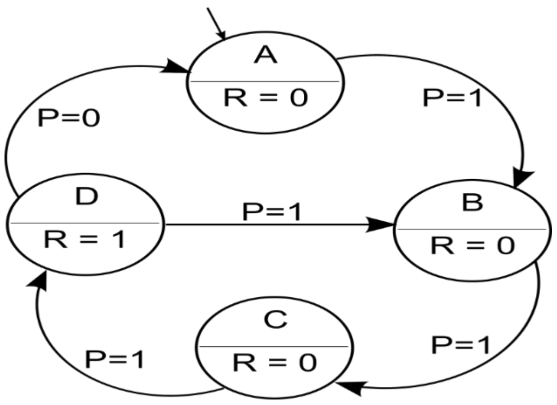


Fig. 1Simple Finite State Machine

This fully defined state machine can very easily be converted into VHDL. It is important to remember that when writing the VHDL code, what you are doing is describing how you want the hardware (i.e., the digital gates) implemented. So, for example, when you define a set of states like A, B, C, and D in this system, those states are going to be represented by bits, and more specifically by the output of flip flops. In a system with four states, like this one, it would be possible to represent those four states with 2 bits (2 flip flops).

There are other ways that the states could be represented too. One of those ways would be to use four bits, where each bit represents a state, but only one bit can be on at a time. So A would be represented by 0001, B by 0010, C by 0100 and D by 1000. One of the good things about using a high level hardware description language is that you can often ignore this level of detail.Figure 2 shows the general idea of the hardware circuitry that will be created when the VHDL code is synthesized to create the hardware.

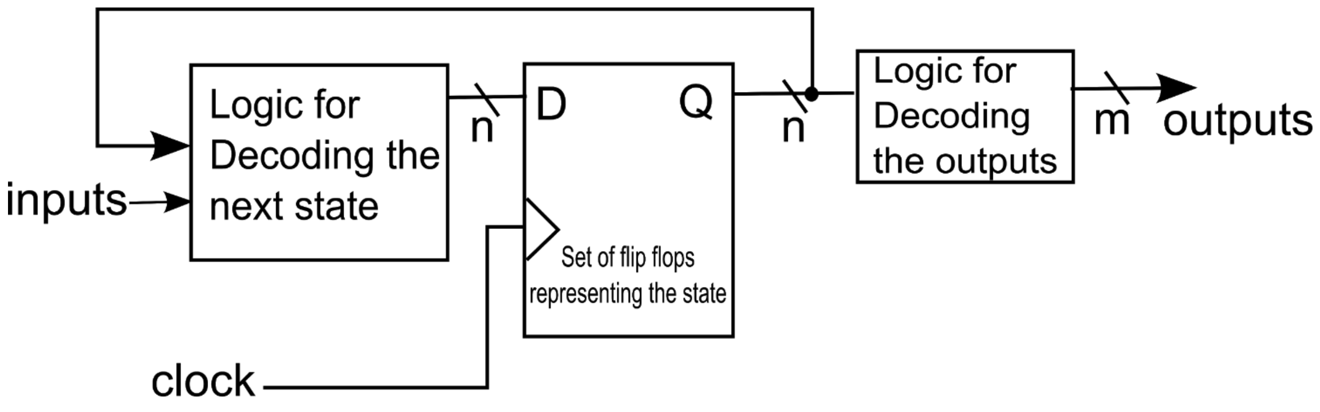

inputs
Logic for Decoding the next state
n
D
Q
Set of flip flops representing the state
n
Logic for Decoding the outputs
m
outputs
clock

Figure.2 Block Diagram Representation of Logic Created for a State Machine

This diagram indicates that there is a set of n flip flops that represent the state. There is also some logic that uses the output of the flip flops and the inputs to the system to determine the next state. Finally, there is some logic that decodes the output values of the flip flops to create the m output signals.

Again, when using a HDL, you can often ignore this level of detail in your design. It is still important to understand what kind of circuitry is created by your HDL because there may come a time when you have to count and minimize the number logic gates in your design. With an understanding of what is created by your HDL statements you can then design to minimize gate creation.

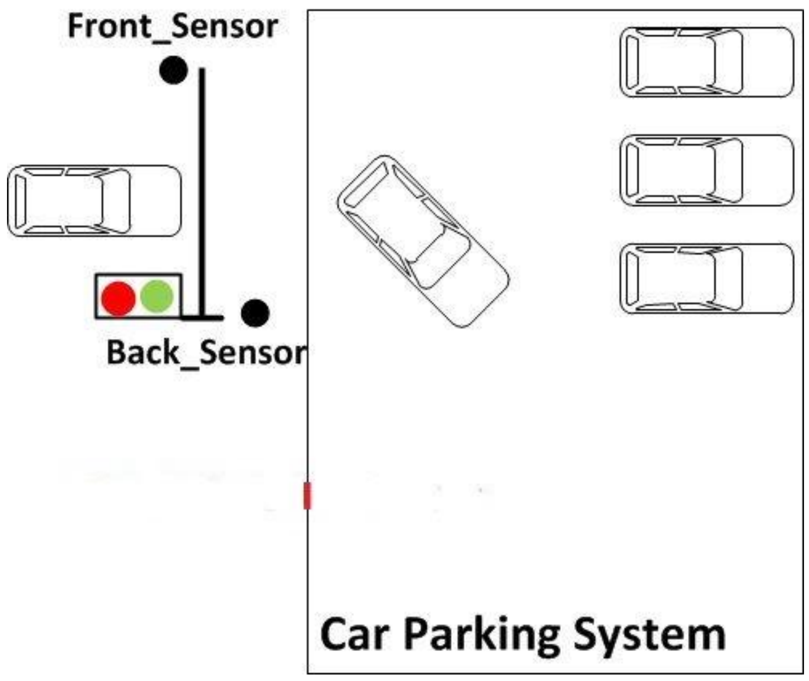


Fig. 3Block Diagram

A. Basic Software Requirements

After creating an account, install Xilinx software: ISE 14.7 from the website at http://www.xilinx.com/ support/download/index.htm For a step by step process of downloading and installing Xilinx ISE Web Pack (student version), go to the appendix at the end of tutorial lFor extra help with the installation, go to: http://www.xilinx.com/support/documentation/dt_ise.htm Xilinx is a powerful software tool that is used to design, synthesize, simulate, test and verify digital circuit designs. The designer (you in this case) can describe the digital design by either using the schematic entry tool or a hardware description language. In this tutorial, we will create VHDL design input files – the hardware description of the logic circuit, compile VHDL source files, create a test bench and simulate the design to make sure of the correct operation of the design (functional simulation). The purpose of this tutorial is to give new users an exposure to the basic and necessary steps to implement and examine your own designs using ISE environment. In this tutorial, we will design one simple module (OR gate); however, in the future, you will be designing such modules and completing the overall circuit design from these existing files. A VHDL input file in the Xilinx environment consists of Entity Declarations: module name and interface specifications (I/O) – list of input and output ports; their mode, which is direction of data flow; and data type. Architecture: defines a component's logic operation. As you will learn (or have learned) in this course, there are different styles for the architecture body:

Behavioral – set of sequential assignment statements
- Data Flow – set of concurrent assignments
- Structural – set of interconnected components

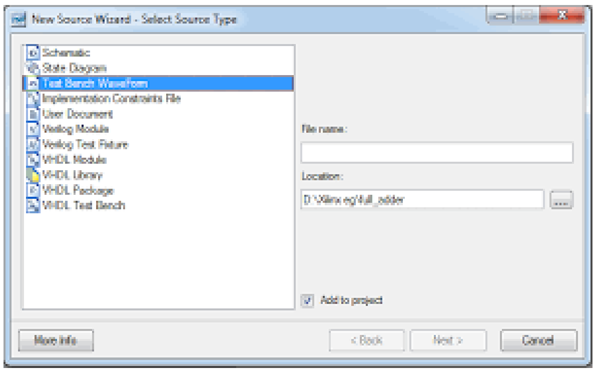
A combination of these could be used, but in this tutorial we will use Dataflow. In its simplest form, the architectural body will take the following format, regardless of the style:

```
architecture
architecture_name of entity_name is
 begin …
-- statement end
architecture_name;
```

III. Simulation of Design

In order to do functional and timing simulation[3], we will create a test bench for our VHDL code which will help in debugging our design. This allows us to verify that our design functions as expected (given inputs in our truth table, we get desired outputs). In order to test the gate completely, we shall provide all the different input combinations.

1. From the tool bar, select Project New Source


New Source Wizard - Select Source Type

Schematic
State Diagram
Test Bench WaveForm
Implementation Constraints File
User Document
Verilog Module
Verilog Test Fixture
VHDL Module
VHDL Library
VHDL Package
VHDL Test Bench

File name:

Location:
D:\Xilinx eg\full_adder

Add to project

Fig. 4Text bench window

2.From the “Select Source Type” options select “VHDL Test Bench”
3. In the “File name” field choose a name that signifies the test bench and adheres to the naming conventions mentioned earlier.Type “testorgate”
4. For the “Location” field, click the browse icon to navigate to the appropriate folder, which should be the same one used forcreating the project.
5. Click “Next”
6. The following window allows you to select which design you want to create a test bench for, in our case “ORgate” since it is the only module we have; however, for your future designs, you can make test benches for individual components of yourdesigns as well as the top-level design which ties it all together.
7. Click “Next”.
8. A summary window like the one shown below will appear, click “Finish”
9. Now you will view the test bench file (testorgate.vhd), shown below, that Xilinx has generated in the workspace window.

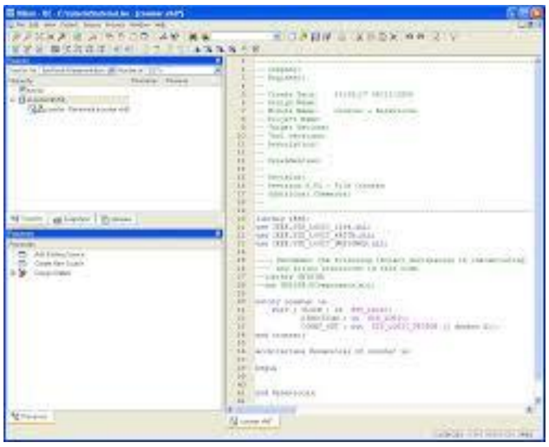


Fig. 5Test bench in VHDL

- Now going to our test bench file, we can see that it consists of the same two main parts of a normal VHDL design, which is the entity and architecture. The entity is left blank because we are simply supplying inputs and observing outputs to the design in test. The architecture part will consist of the design we are testing as a component, input and output signals, a port map of the component for the UUT (Unit Under Test), a process to run the clock and a stimulus process, which will be responsible for running the tests that are written to test the design

10. Let’s modify the default code by removing the highlighted code shown below, which is the clock process that is generated by default, which divides the clock period by two. We also want to remove the stimulus process.
11. Replace the deleted code with the following code segment, which will perform a very simple initial test of the design for simulation by giving different values of inputs. In our modified code, we have chosen to wait for 100 ns, which means the time delay for which the input has to maintain the current value; i.e., after 100 ns have elapsed the next set of values can be assigned to the inputs
12. The test bench file does not appear in the **“Hierarchy”** Pane of the **“Design**” Panel. This is because there is a separate view for implementation and test files. In order to view test files, select the box of **“Simulation**” in the **“View Pane”** of the **“Design”** panel. In the **“Process Pane**,” double click on the “**Behavioral Check Syntax”** to make sure that you didn’t make any syntax errors while making changes.
13. Save your work.
14. Double click on **“Simulate Behavioral Model”** in the “**Process Pane**”, which will open the ISim software with your test bench loaded.
15. ISim simulator window will open with your simulation executed, where you are able to simulate your designs and check for errors. You can step through your VHDL designs and check the states of signals and set the simulation to run for specific period of time. Make sure to check the results of the simulation output against your truth table results to verify the correctness of the design. The resolution of the simulation is set to 1 picoseconds to ensure correct processing of your design.
16. Toget a better view of the simulation waveforms, from the tool bar, click on View Zoom Full View or use F6 or click on the shortcut “**Zoom to Full View**” icon . This will give you a better view of what your simulation is doing

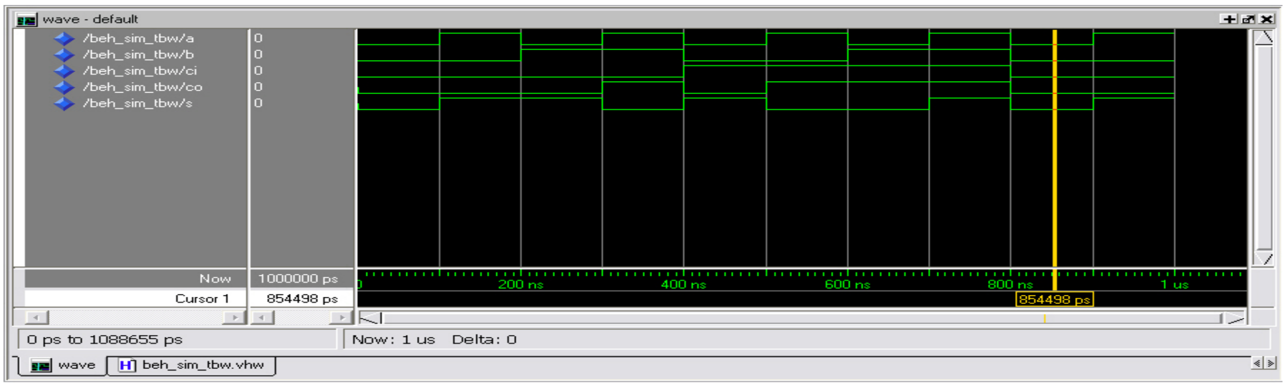


Fig. 6Output waveform

17. In the text box located near the run button, you may specify amount of time for the simulation to run; the button to the left of the box will execute the simulationfor the time you have specified. After setting the new simulation time, click on Re-Start to clear the previous simulation result and then click on Run to start simulating with new time setting. Below is an example of 2us of simulation time:

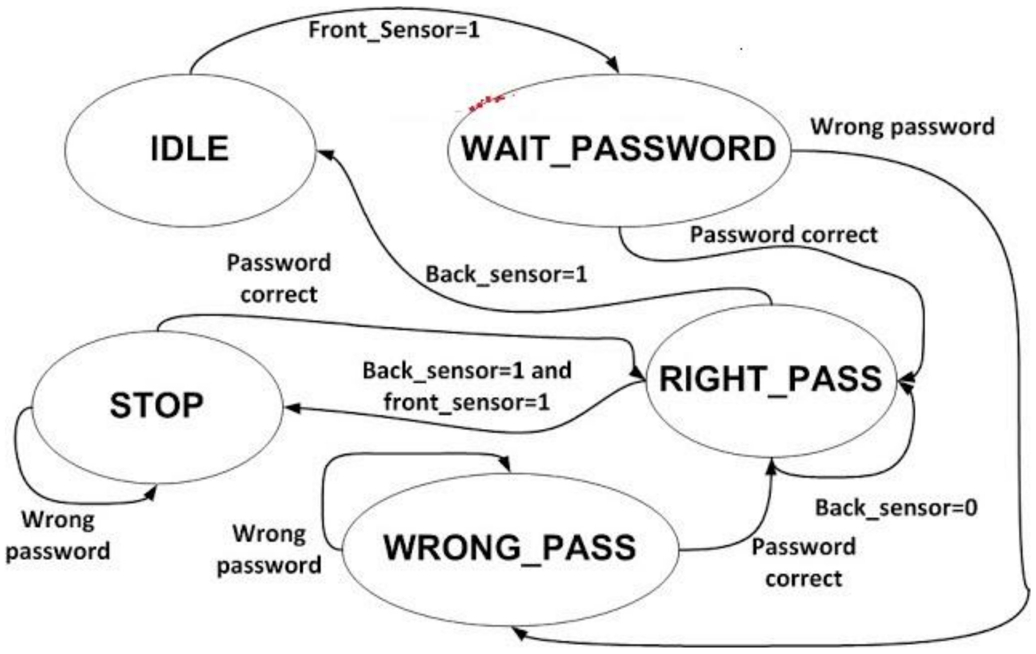


Fig. 7 Flow Chart

IV. RESULTS

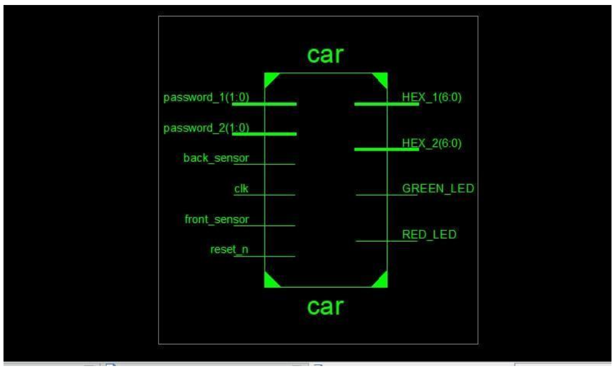


Fig. 7 RTL schematic

The above figure shows the RTL Schematic view of car parking system, passwords 1 &2 ,back_sensor, front_sensor are input signals. Reset_n is control signal,clk is system clock signal.LEDGreen_LED ,Red_LED are output signals, which shows entering the car in the slots.

A. *Output Waveform*

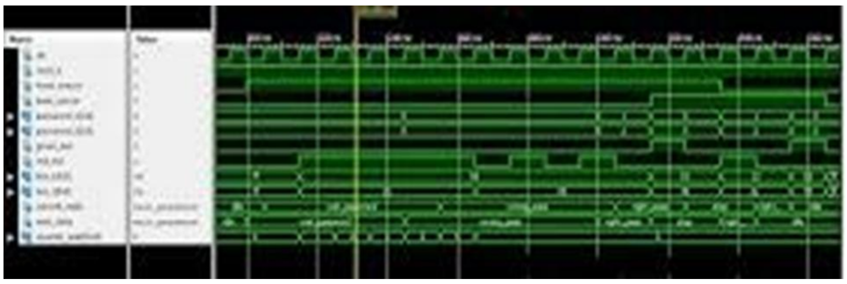


Fig. 8 Output Waveform

The above figure shows the Output Waveform of car parking system.

V. CONCLUSION

The present parking system is implemented using FSMs with the help of Xilinx ISE Design Suite 14.7. The design is verified. State machines increase productivity, reduces cost[4], and accelerates time to market. The designed system can be used for many applications and can easily enhance the number of slot selections. Parking becomes easy by the use of Designed system.The present FSM based parking system using VHDL can be implemented in FPGA with the help of Xilinx ISE Design Suite 14.7 the design is verified on Virtex 5 FPGA kit. State machines increase productivity, reduces cost, and accelerates time to market. FPGA based parking system, gives fast response. The designed[1] system can be used for many applications and can easily enhance the number of slot selections. Parking becomes easy by the use of Designed system

REFERENCES

[1] Du Shaobo; Sun Shibao;,(2012) "The research and design of intellectual parking system based on RFID," Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on, pp.2427-2430.

[2] Gongjun Yan; Weiming Yang; Rawat, D.B.; Olariu, S.,(2011) "SmartParking: A Secure and Intelligent Parking System," Intelligent Transportation Systems Magazine, IEEE , vol.3, no.1, pp.18- 30.

[3] Liang; Zhang Lei; Xiao Jin; ,(2011) "The simulation of an auto-parking system," Industrial Electronics and Applications (ICIEA), 2011 6th IEEE Conference on , pp.249-253.

[4] Soh Chun Khang; Teoh Jie Hong; Tan Saw Chin; Shengqiong Wang;(2010) , "Wireless Mobile-Based Shopping Mall Car Parking System (WMCPS)," Services Computing Conference (APSCC),2010 IEEE Asia-Pacific , pp.573-577.

[5] Gupta, A.; Divekar, R.; Agrawal, M.; ,(2010) "Autonomous parallel parking system for Ackerman steering four wheelers," Computational Intelligence and Computing Research (ICCIC), 2010 IEEE International Conference on , pp.1-6.