

# **Documentation**

## **Cypher**

21 June 2022

## **1 Team members**

1. Sheikh Muhammad Adib Bin Sh Abu Bakar
2. Zafirul Izzat Bin Mohamad Zaidi
3. Muhammad Farid Izwan Bin Mohamad Shabri
4. Muhammad Iqbal Bin Mohd Fauzi

## **2 Introduction**

Parking lots are a common place visited by thousands of people every day. Looking for a parking space could take a lot of time. It is frustrating to drive around looking for a parking lot and end up with no place for parking. This problem is our big interest in our project. We plan to create a system that could help people reduce time in looking for a place for parking by indicating a parking area whether it is already full or not. That means our system is capable of counting the number of entering and leaving cars. The number of used spaces will be displayed at the entrance of the parking area. When there is no free space left, a red light indicator, located near the display unit will be turned on and the entering gate will be closed.

To reduce the cost of building the solution, we use FPGA to prototype our solution since the solution will go through many iterations. This means that a lot of change will be made without spending a lot of money. In order to be able to implement the system in FPGA, we need to use VHDL to model our system and use it for further synthesis to be able to produce the system structure on the logic level and executable model that can be run on the FPGA.

In the next section, we will go through how we design our solution followed by how we organise the project and eventually, the steps of the implementation process

## **3 Concept description**

To build our parking lot system, we reduce the complexity by modelling the structure and behaviour of the parking lot in a few blocks diagram. Our main target was that we could have a concrete system architecture so that we could split the system into a few

components. Modelling diagrams not only helps us in building a system in a more organised way, but it also helps us as a team to work together parallelly.

To do that we first start with a requirement diagram to put every requirement that we want to achieve either functional or non-functional related to the parking lot system that we want to build.

We then create a use case diagram to set the boundaries of the parking lot system. With a use case diagram, we can analyse the components that could exist in the system. After that, we model each action in the use case using an activity diagram with possible scenarios. Next, we analyse all the diagrams to build a concrete parking lot system architecture, which means that we are ready to do the state chart and state machine diagram so that later we can synthesise them using VHDL. The models will be further explained in the implementation section.

The first phase of our project began with a pre-study where basic knowledge of the parking lot was gathered. The aim of the pre-study was to get a summary of our system project in general. The pre-study also enabled the search for content to be used as a guide in this project.

Now, we documented the requirements for this project with the functionalities as shown in Figure 1. One of the requirements of our system is to count cars whenever cars enter and leave. The system also may have a space indicator to help drivers know whether the space is empty or full. Display the number of free spaces is also crucial and lastly, the gate controller can either open or close.

Figure 1: System requirements

To be able to further work with the requirements we form a draft architecture as shown in Figure 2 as a first step for finding the solution to the requirements.

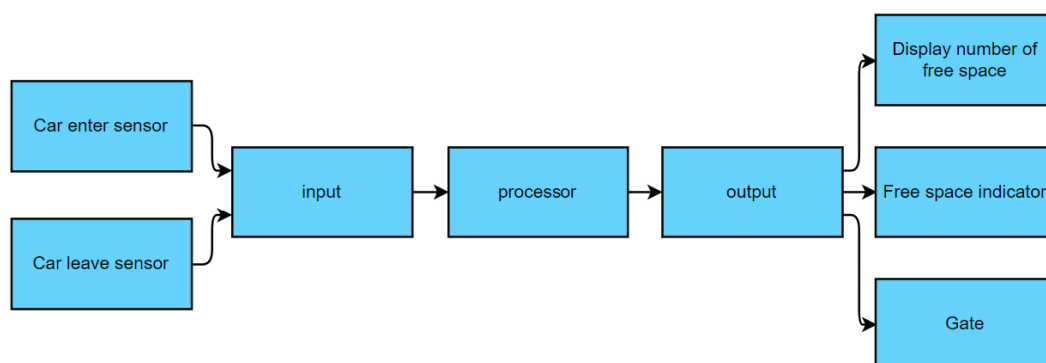


Figure 2: Draft architecture

## 4 Project/Team management

For project management, we use the scrum method where we divide the time that we have to develop this product into 11 cycles and each cycle takes 1 week. To make it more efficient, we have two minimum meetings in each cycle. In the first meeting, we will discuss and brainstorm together the big task that we need to complete in the cycle. As a result, the sub-tasks are created and distributed fairly among the team members. The second meeting is to verify the current progress and make improvements.

Since we use the scrum method, we still help each other in completing each sub-task to ensure the product of each cycle is at its highest and finest quality. The division of the tasks is shown in Table 1.

cycle	Task	Subtask	Actor
1	Idea and discussion	Concept	All members
2-3	Concept model	Use case	Zafirul Izzat
		Activity	Sheikh Adib
		Requirement	Farid Izwan
		Sequence	Muhammad Iqbal
		System architecture	Sheikh Adib
		Draft system architecture	Zafirul Izzat
		Class	Farid Izwan
			Muhammad Iqbal
	FSM discussion and implementation	FSM diagram	All members
4	Component and process analysis	discussion	All members
5-6	Component and	4 bit bcd	Zafirul Izzat

	process implementation using VHDL	Up down counter	Sheikh Adib
		D flip flop	Farid Izwan
		4 bit full adder	Muhammad Iqbal
		8-bit to 3 4-bit bcd	Zafirul Izzat
		3 4-bit bcd scheduler	Sheikh Adib
		Clock divider	Farid Izwan
		comparator	Muhammad Iqbal
		Next state process	Sheikh Adib
7-8	Model upload	Xilinx installation	Zafirul Izzat
			Sheikh Adib
		Code Synthesis	All members
		Code Debugging	All members
		Netlist	Farid Izwan
			Muhammad Iqbal
		Upload Bitstream to FPGA	Sheikh Adib
			Zafirul Izzat
9-10	Hardware Simulation	Output verification	All members
11	Documentation	Concept description	Zafirul Izzat
		Introduction	Sheikh Adib
		Project/Team management	
		Technologies	Farid Izwan
		Implementation	Muhammad Iqbal

			Sheikh Adib
		Use case	Farid Izwan Zafirul Izzat

Table 1: Tasks division

## 5 Technologies

In this section, technological approaches that are used in the Parking Lot System project will be explained. First and foremost, to realise our Parking Lot System in hardware, we model our system using Very High-Speed Integrated Circuit Hardware Description Language (VHDL) by implementing all the components and processes based on the system architecture and system's FSM.

The model is then synthesised using Xilinx ISE and the result, as shown in Figure 20, is then together with the netlist used to produce a bitstream file to be uploaded to the Field Programmable Gate Arrays (FPGA) board. This is to examine whether the outcomes that we want for our Parking Lot System as expected or not on the FPGA board. For example, a 7-segment LED that indicates the number of used spaces really appears correctly on the FPGA board.

After the system is successfully tested, the new component for the system is created on Autodesk EAGLE and named Cypher. This component is used in building the schematic for the system and later on used for the PCB layout. Therefore, all components can be placed and connected to each other clearly. Table 2 describes what are the electronic components that are being used in the Parking Lot System project.

Component	Function	Remarks	Amount used
7 segments LED	Display number of used space		3
LED	Indicate for free space	Red LED on when parking space full	4
		Green LED on when parking space is not full	
	Indicate for gate state	Red LED on when gate close	
		Green LED on when gate open	
Button	Reset	change state to idle	1

		and reset counter	
Switch	Input	car enter or leave signal	4
	Ready	no car near the gate	1

Table 2: Listed electronic components in Parking Lot System

## 6 VHDL Implementation

The project implementation consists of designing the static structure of the system and its environment. This design entails many diagrams in order to make it more clear for implementing the system prototype.

First, the reader will be exposed to the use case model to know the system and the environment in more detail. To make it more concrete, the activity diagram is explained based on the use case model. Last but not least, with the help of the previous model, we generalise our system into the system architecture and modules that are involved in the system.

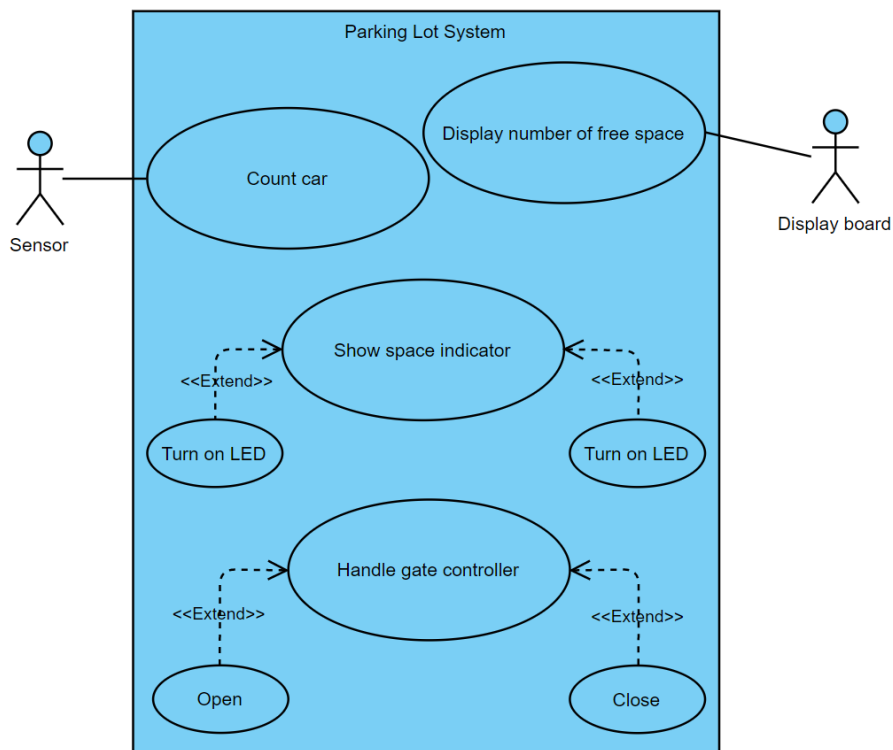


Figure 3: Use case model

The use case diagram for a parking lot system is shown in Figure 3. There are two actors in this diagram: the sensor and the display board. The sensor in this system oversees counting cars as they enter and exit the parking lot. Furthermore, a display board will show the number of available parking spaces for the car. This system also includes an indicator that displays

available space. For instance, if there is free space, the green LED will turn on and the red LED will be turned off; otherwise, if there is no free space, the green LED will turn off and the red LED will turn on. This system will also include a gate controller handle, which will allow the gate to be closed or opened.

We will now proceed to the activity diagram. This activity diagram's purpose is to clearly show how our parking lot system's flow works. There are four diagrams that have been modelled: a counter, the number of cars to be displayed, the condition of the LED (whether on or off), and the action of the gate (closed or open). Figures 4, 5, 6 and 7 depict these activity diagrams.

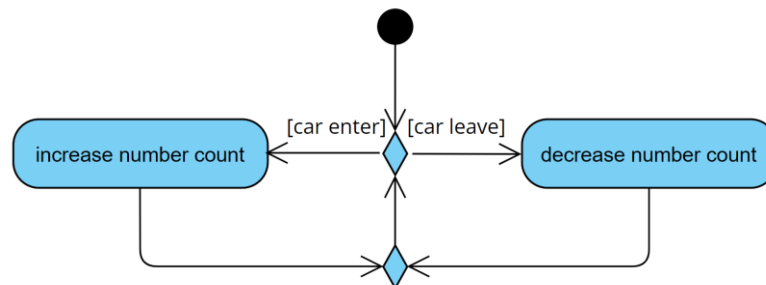


Figure 4: Activity diagram counter

Figure 4 shows the activity diagram when the car is counted. When the car is entering, the counter will add the number to indicate an increasing number of cars entering the parking lot. When the car is leaving, the counter number will decrease. Then it will continuously count when the car is either leave or enter.

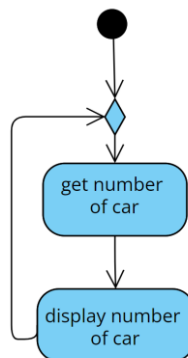


Figure 5: Activity diagram display number

Figure 5 explains the scenario when displaying the number of used parking spaces. The activity diagram describes when the display will get the number of cars entering, then it will be displayed on the display board.

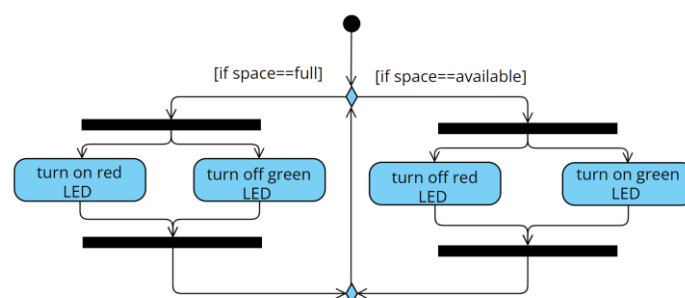


Figure 6: Activity diagram space indicator

Activity diagram as shown in Figure 6 explains how the space indicator works. When there is available parking space in a parking area, the LED will turn green and the red LED will be switched off, so that people know that they can find empty parking space. If the parking space is already fully occupied, then the red LED will be automatically switched on while the green LED will turn off.

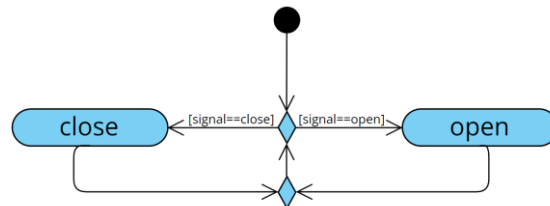


Figure 7: Activity diagram gate controller

Figure 7 explains how the gate controller is being handled. When the open signal is received, the gate will open. If the close signal is received, the gate is closed. The system will keep reading if there are any changes to the signal.

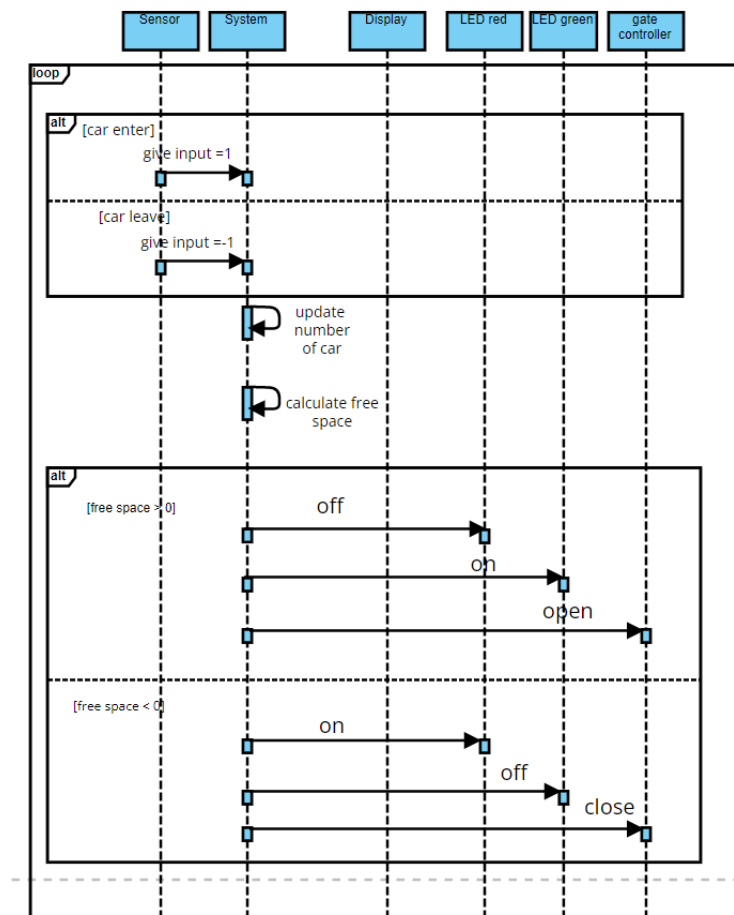


Figure 8: Sequence diagram of parking lot system

The flow of the system will be depicted in this sequence diagram as shown in Figure 8. Sensor, system, display, green LED, red LED and gate controller are the six required objects. Initially, the sensor will inform the system whether the car is entering or exiting the parking lot. If the car enters the parking lot, the system will receive the input value which is 0001. If the car



drives away, the input value will be 0010. The system will then update the number of cars. Following the completion of the process of updating and calculating the number of cars, the system will display the number of used spaces on the display board. If there is free space available, the green LED will turn on, the red LED will turn off, and the system will open the gate. If there is no free space, the green LED will turn off, the red LED will light up, and the gate will close.

Figure 10 shows our final system architecture including every component based on the previous model that we have discussed. The inputs consist of a car enter sensor or car leave sensor, reset, clock and ready. The sensor will detect whether a car leaves the parking lot or enters the parking lot. One more input which is crucial in our system architecture is ready because we want to avoid any miscalculation. For example, when the car enters the parking lot and simultaneously the other car leaves the parking lot. Then, all inputs will go to the next memory state. It also consists of reset, which is to reset the state of each component and the output values. In the FSM part, there are a few modules with different functionality. The up / down counter is used to indicate the number of free spaces on display and the action of the gate. The function of the comparator module depends on the output. For example, the gate will close if the value of cars is equal to 255. Then, based on the diagram, the clock is ported to the clock divider to lower the frequency so that we can see the change in value when our code is implemented on the FPGA board. Furthermore, 8 bit to 4-bit BCD converter is used so that the value can be split into 3 modules of 4-bit BCD. The reason for using a BCD converter is because it is one of the ways of representing a decimal number as a string of bits. The output from those 4-bit BCD will then be displayed in the 7-segment LED. Our system's output also has two green LEDs and a red LED. The LEDs are used to indicate when the gate is closed or open. The other LED is to indicate the number of free spaces. For example, when the space is full, meaning that the value of cars entering is 255, both LEDs will turn red. One indicates the space is full, and the other one indicates the gate is closed.

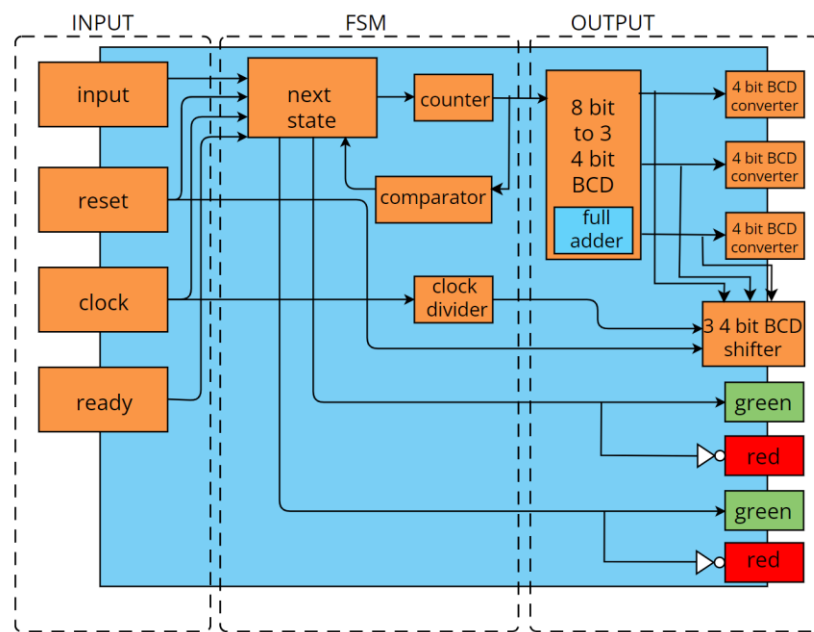


Figure 10: System architecture parking lot system

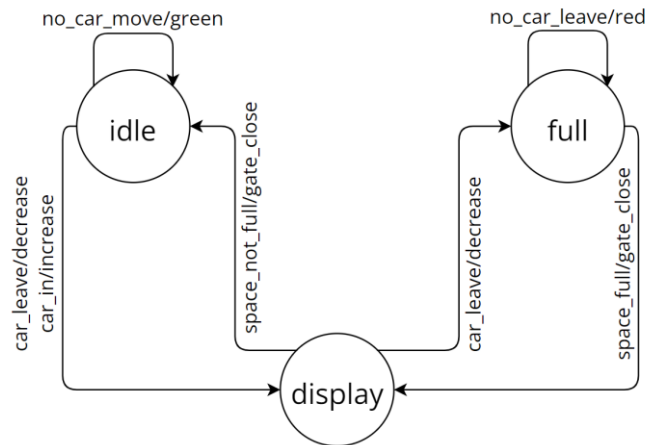


Figure 11: Finite state machine parking lot system

Prior to implementing our parking lot system project in VHDL, a finite state machine must be designed. This finite state machine, which is based on Moore's law, is required to understand how our parking lot system operates. Figure 11 depicts the previously mentioned finite state machine. As illustrated in Figure 11, our system behaves in three states: idle, display, and full. They will each have their own input and output in these three states. Table 3 shows a description of the input and output of these three states. The car's entry and exit will be monitored while it is idle. For example, if no car arrives, the system will remain in an idle state. However, if a car attempts to enter the parking lot, the counter will be raised. In comparison, if the car leaves the parking system, the counter will be reduced. The state will then change from idle to display. The number of cars in this parking lot will be displayed in the display state. It should be noted that if there is still an empty parking space and a car to enter, the system will transition from the display state to the idle state and the gate will remain open. In addition, the green LED will turn on to indicate that there is still free available space. However, if there is no empty space available, it will switch to the full state, close the gate, and turn on the red LED. As a result, no cars are permitted to enter the parking lot. If there are no car leaves, the system will remain in its full state and the red LED will remain on. In contrast, if a car exits, it will return to the display state, with the red LED turning off and the green LED turning on. The gate has been reopened.

State	Input	Output
Idle	Input="0001" indicate car enter	Counter increase
		Enter display state
	Input="0010" indicates car leave	Counter decrease
		Enter display state
	else	Remain in idle

Display	Input space_state ="1" and ready="1"	Gate open
		Enter idle state
		Green LED turn on
	Input space_state ="0" and ready="1"	Gate close
		Enter full-state
		Green LED turn off
	else	Red LED turn off
		Remain in display state
Full	input="0010" indicate car leave	Counter decrease
		Back to display state
		Red LED turn off
		Green LED turn on
	input ≠ "0010"	Remain in full state

Table 3: System FSM state transition

The 7 segment LED is one of the components that we test. This is to ensure that the outcome is true when we display the number of cars entering the parking lot. We can see that in Figure 12 whenever the input is increasing in number, there is a change in the 7 segments led. For example, when the input is 0000 meaning that there is no car, the 7 segments led display number 0.

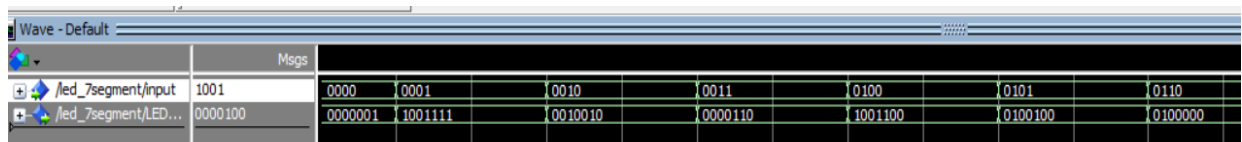


Figure 12: Test bench 7 segments LED

To make the counter readable for the public, the 8-bit binary value should be changed to a decimal number using a 7-segments LED. Since the maximum number is 255, we use 3 7-segments LED. For that reason, we make a component called an 8-bit to 3 4-bit BCD converter where the input is the 8-bit binary number from the counter and the output is the combination



		Msgs	
	/bcd_8_bit/input_8_bit	-No ...	11111111
	/bcd_8_bit/bcd	-No ...	001001010101

Figure 13: Test bench 8-bit to 3 4-bit BCD converter

Since all the 7 segments LED signals on the FPGA that we used for the hardware prototype which is spartan 3 are connected together, we create another component called 7-segments LED scheduler to sequentially choose a 7-segments LED at a time to display the appropriate number. This means a clock must be supplied to the component and it should be at the frequency at which the numbers are seen simultaneously displayed. The result of the component on a test bench is shown in Figure 14.

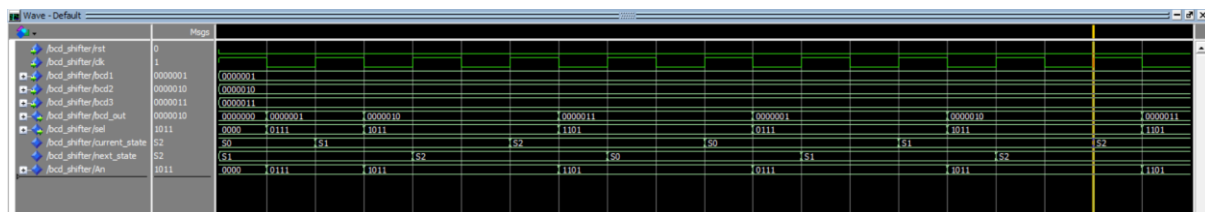


Figure 14: test bench Bcd shifter

An up-down counter will be used to count the number of cars that enter and exit. If cars enter the parking lot system, the counter will increase. In contrast, if the car leaves the parking lot, the counter will be reduced. In Figure 15, we test our counter system using a test bench, and the results are as expected, indicating that the counter will be altered if the car enters or exits the parking lot.

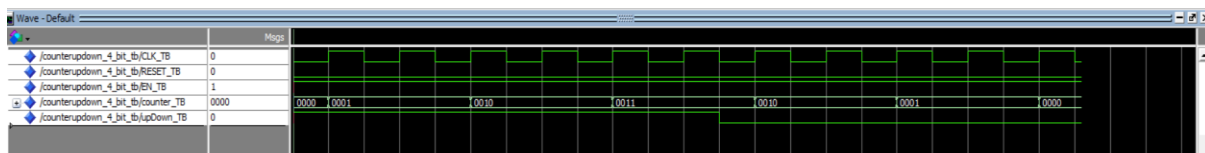


Figure 15: Test bench counter

After testing each component, the result is recorded as we expected. Then, we integrate all of the components into our main system which is the parking lot system. The outcome of the test bench can be referred to in Figure 16.

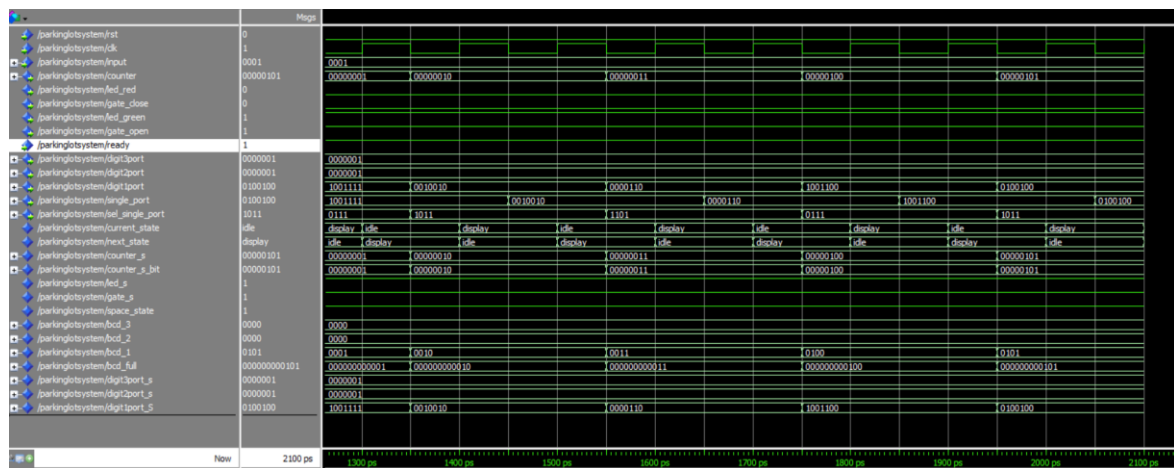


Figure 16: Test bench parking lot system

Ater all the tests meet our expectations, the VHDL model is then synthesised using Xilinx ISE and the result is shown in Figures 17 and 18. This SoC schematic is later on used for the PCB Design. The position of the component used on the FPGA board can be visualised. Apart from them are shown in Figure 19. From the position visualisation, we can also explore the circuit for each input and output as shown in Figures 20, 21 and 22.



Figure 17: Synthesis result

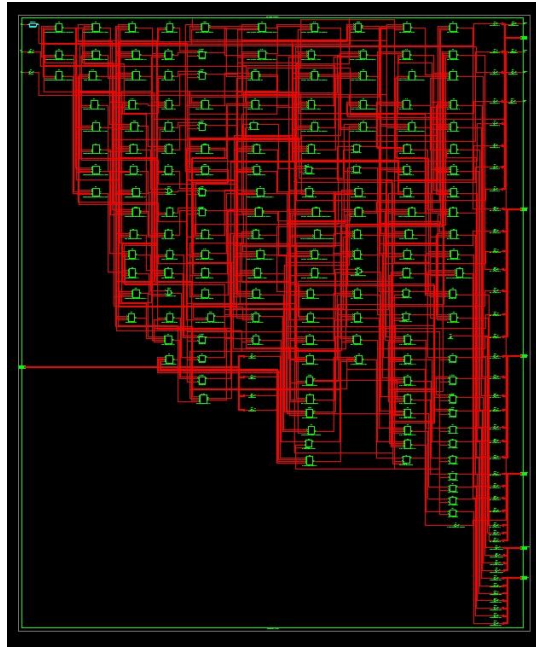


Figure 18: Synthesis result with the detail



Figure 19: Some component position on FPGA board

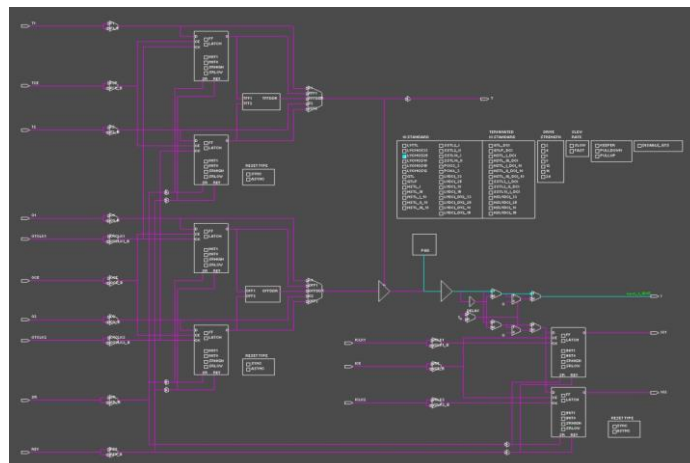


Figure 20: Input[0] circuit

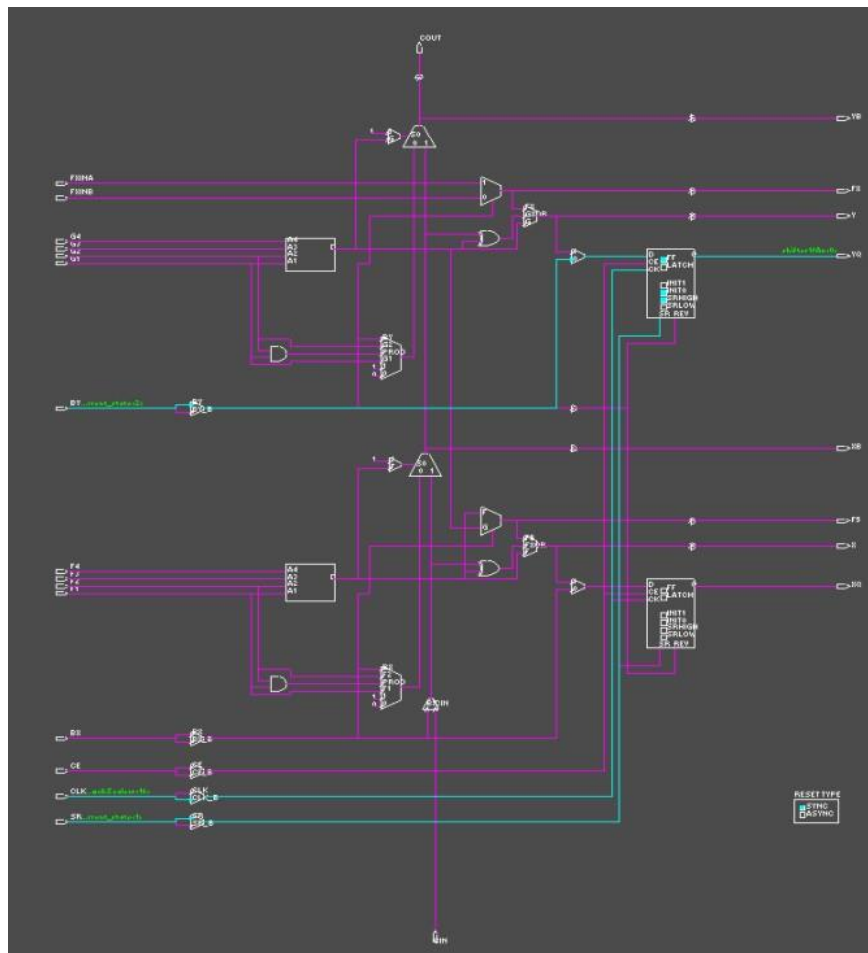


Figure 21: scheduler first selects output circuit

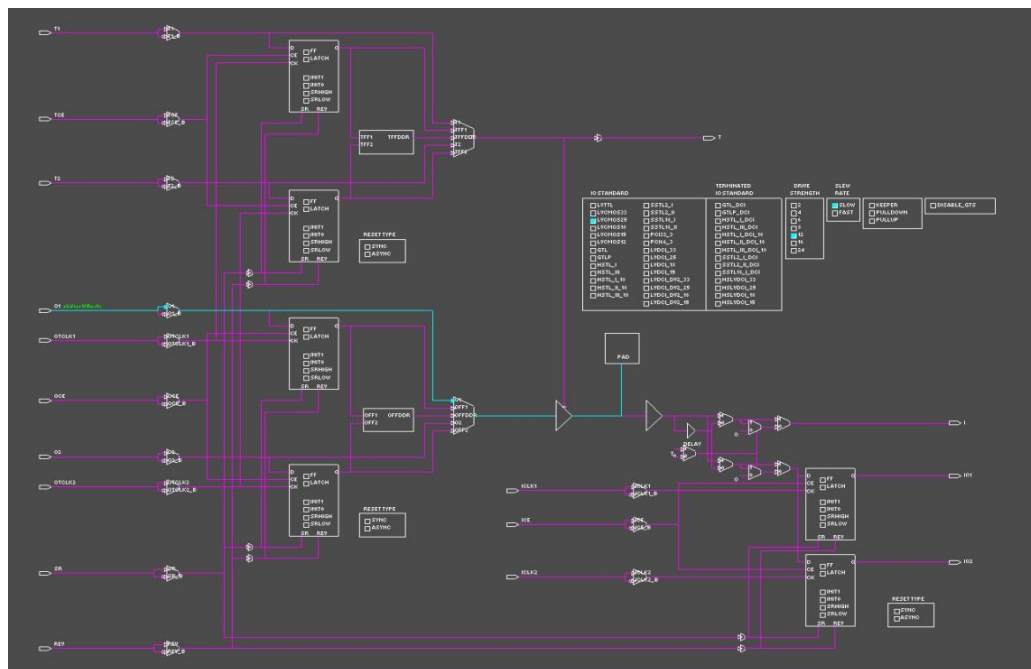


Figure 22: 7 segment LED output



Based on Figure 23, we have successfully implemented our own SoC on an FPGA board. The counter indicates the number of cars entering and leaving the parking lot. As you can see in the diagram, four LEDs are used to indicate free space and gate state. There is also a reset button to reset the value at the counter. Based on our requirement, the input to detect the movement of the car is by using a sensor. But in this case, the input that we are using is a switch to indicate whether the car is entering or leaving. There are two inputs. The first one is when the car is entering, so the counter will increase in value. Next when the car leaves the parking lot. So, the counter will decrease in value.

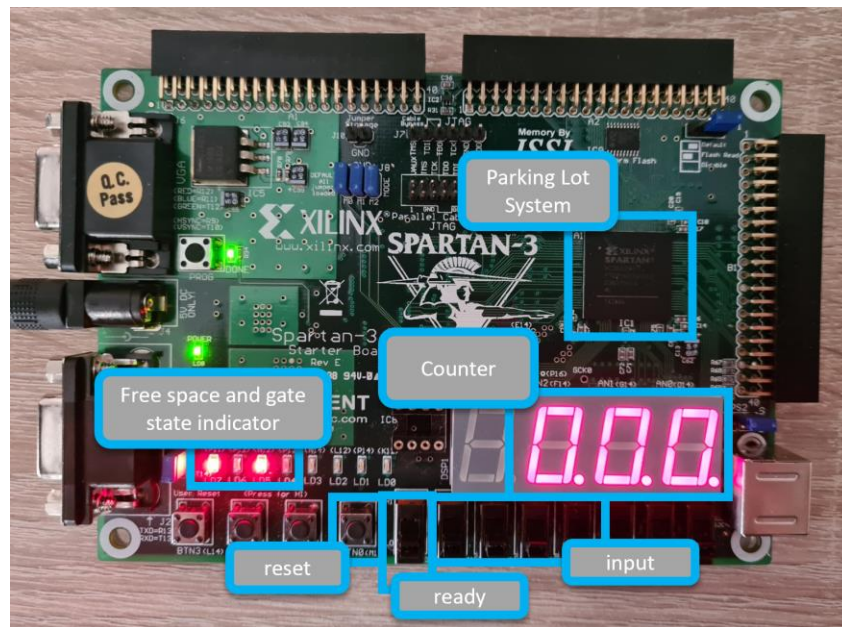


Figure 23: Hardware (FPGA) implementation

## 7 PCB Design

To design our PCB, we start with creating the schematics as shown in Figure 24. The component that we used for our PCB is N55 to provide a clock for the system, cypher 2022 (our own SoC, the schematic is shown in Figures 17 and 18 )to execute our system, pin header for the output to the segment LEDs and input from the sensor and some passive component for the PCB to work properly.

We then use the schematics to produce the PCB layout as shown in Figure 25 and the expected printed PCB will look like in Figure 26. The used material is listed in the BOM file as shown in Table 4.



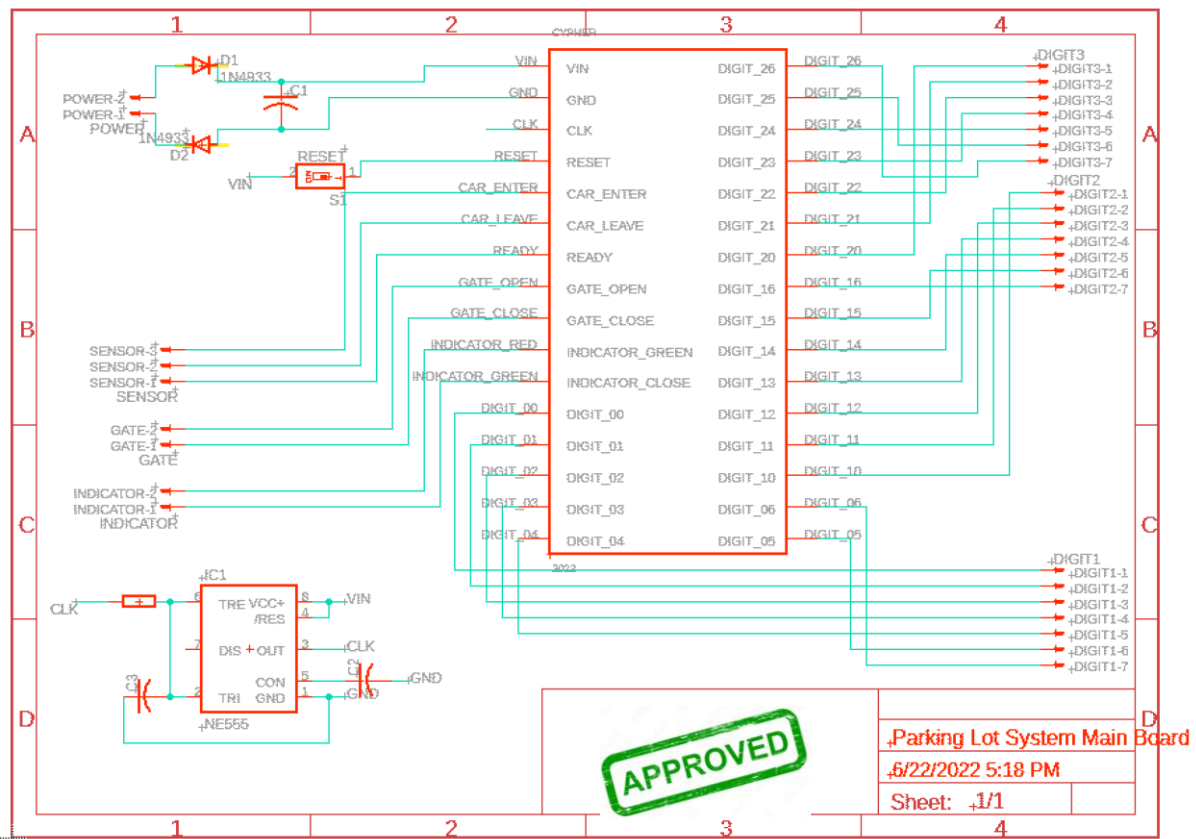


Figure 24: Schematic diagram of parking lot system

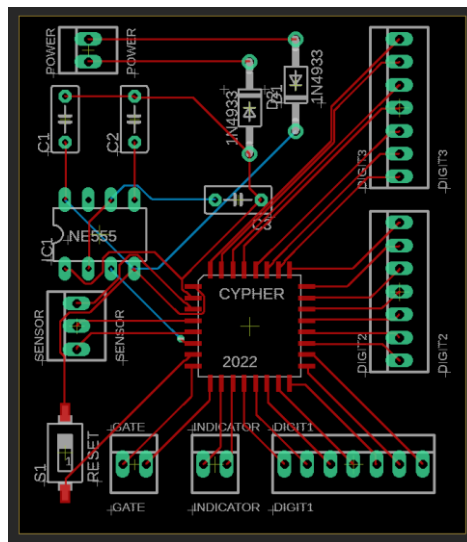


Figure 25: PCB layout

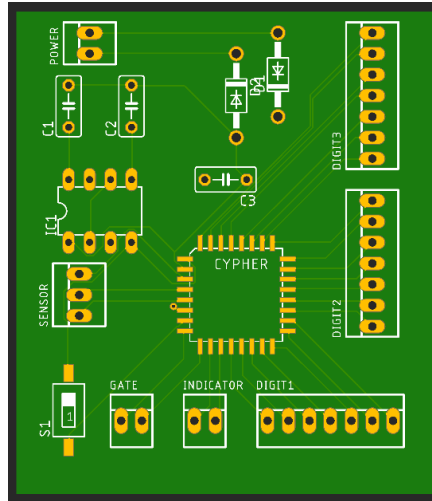


Figure 26: expected Printed PCB

Part	Value	Device	Package	Description	MF	MPN	OC_FARNELL	OC_NEWARK	POPULARITY	SPICEPREFIX
C1		C-US050-030X075	C050-030X075	CAPACITOR, American symbol					5	C
C2		C-US050-030X075	C050-030X075	CAPACITOR, American symbol					5	C
C3		C-US050-030X075	C050-030X075	CAPACITOR, American symbol					5	C
CYPHER	CYPHER2022	CYPHER2022	CYPHER							
D1	1N4933	1N4933	DO41-10	DIODE					2	
D2	1N4933	1N4933	DO41-10	DIODE					2	
DIGIT1	DIGIT1	22-23-2071	22-23-2071	.100" (2.54mm) Center Header - 7 Pin MOLEX		22-23-2071 1654534	56H0445		1	
DIGIT2	DIGIT2	22-23-2071	22-23-2071	.100" (2.54mm) Center Header - 7 Pin MOLEX		22-23-2071 1654534	56H0445		1	
DIGIT3	DIGIT3	22-23-2071	22-23-2071	.100" (2.54mm) Center Header - 7 Pin MOLEX		22-23-2071 1654534	56H0445		1	
GATE	GATE	22-23-2021	22-23-2021	.100" (2.54mm) Center Header - 2 Pin MOLEX		22-23-2021 1462926	25C3832		40	
IC1	NE555	NE555	DIL-08	General purpose bipolar Timer	Arrow Electronics	NE555N 1467742	89K1486		24	
INDICATOR	INDICATOR	22-23-2021	22-23-2021	.100" (2.54mm) Center Header - 2 Pin MOLEX		22-23-2021 1462926	25C3832		40	
POWER	POWER	22-23-2021	22-23-2021	.100" (2.54mm) Center Header - 2 Pin MOLEX		22-23-2021 1462926	25C3832		40	
S1	RESET	SWS001	SMS-001	SMD Dip Switch 1 pol.					9	
SENSOR	SENSOR	22-23-2031	22-23-2031	.100" (2.54mm) Center Header - 3 Pin MOLEX		22-23-2031 1462950	30C0862		35	

Table 4: BOM

## 8 Sources/References

embedded-hardware-lab/Projects-Cypher (github.com)