Parking Management System Project Report

Introduction

This report presents a comprehensive analysis of a Parking Management System designed using Verilog. The system aims to manage the entry and exit of vehicles from a limited-capacity parking lot, and calculate parking fees based on the duration of stay.

The system is designed to track up to three vehicles simultaneously, with the ability to store and retrieve the entry time of each vehicle to calculate the fee upon exit. The system uses a Finite State Machine (FSM) to control entry and exit operations, and provides indicators for the parking lot status (empty or full) and the current number of vehicles.

Project Overview

Project Objective

The main objective of this project is to design and implement a digital system for parking management that can:

1. Track vehicle entry and exit
2. Store the entry time of each vehicle
3. Calculate parking fees upon exit
4. Manage parking capacity and prevent entry when full
5. Provide indicators for parking lot status (empty or full)

Main Components

The system consists of several integrated modules, each responsible for a specific function:

1. **TopLevelModule**: The main module that connects all submodules together
2. **fsm**: Finite State Machine that controls entry and exit operations
3. **counter**: Counter that tracks the number of vehicles in the parking lot
4. **parking_buffer**: Buffer that stores the entry time of each vehicle
5. **timer**: Time generator that tracks the current time

6. **clock_divider**: Frequency divider to generate a slower clock signal
7. **ccost**: Module that calculates parking fees based on entry time and current time

Module Analysis

1. TopLevelModule

Description

This is the main module that connects all submodules together and provides the complete system interface. It routes signals between different modules and coordinates their operation.

Interface

Inputs	Description
clk	Main clock signal
reset	Reset signal
entry	Vehicle entry signal
exit	Vehicle exit signal
Car_Id[1:0]	Vehicle identifier (0-3)

Outputs	Description
Ccost[8:0]	Parking fee
cars_count[1:0]	Number of vehicles in the parking lot
Is_empty	Signal indicating the parking lot is empty
Is_full	Signal indicating the parking lot is full
current_time[7:0]	Current time

Implementation

The module creates and connects all submodules, routing signals between them. It uses internal wires such as `Entry_time`, `write_enable`, `read_enable`, `count_up`,

`count_down`, and `clk_out` to connect the outputs of one module to the inputs of another.

2. FSM (Finite State Machine)

Description

This module is responsible for controlling entry and exit operations. It uses a finite state machine to track the parking lot state and generate appropriate control signals.

Interface

Inputs	Description
clk	Clock signal
reset	Reset signal
entry	Vehicle entry signal
exit	Vehicle exit signal

Outputs	Description
write_enable	Enable writing to the entry time buffer
read_enable	Enable reading from the entry time buffer
count_up	Signal to increment the vehicle counter
count_down	Signal to decrement the vehicle counter

States

States	Description
EMPTY (00)	Parking lot is empty
ONE_CAR (01)	One vehicle in the parking lot
TWO_CARS (10)	Two vehicles in the parking lot
FULL (11)	Parking lot is full (3 vehicles)

Implementation

The module uses registers `current_state` and `next_state` to track the parking lot state. It also uses edge detection technique to generate `entry_pulse` and `exit_pulse` when entry and exit signals change.

The module determines the next state based on the current state and entry/exit signals, and generates appropriate control signals for each state. For example, when a vehicle enters in the EMPTY state, it activates `write_enable` and `count_up`.

3. Counter

Description

This module is responsible for tracking the number of vehicles in the parking lot and generating empty and full signals.

Interface

Inputs	Description
clk	Clock signal
reset	Reset signal
count_up	Signal to increment the counter
count_down	Signal to decrement the counter

Outputs	Description
cars_count[1:0]	Number of vehicles in the parking lot
Is_empty	Signal indicating the parking lot is empty
Is_full	Signal indicating the parking lot is full

Implementation

The module uses a register `cars_count` to track the number of vehicles. It increments the counter when receiving a `count_up` signal and decrements it when receiving a `count_down` signal, while respecting the parking lot limits.

It generates an `Is_empty` signal when `cars_count` equals `MIN_CARS` (00), and an `Is_full` signal when `cars_count` equals `MAX_CARS` (11).

4. Parking_Buffer

Description

This module is responsible for storing and retrieving the entry time of each vehicle.

Interface

Inputs	Description
clk	Clock signal
reset	Reset signal
read_enable	Enable reading from the buffer
write_enable	Enable writing to the buffer
Car_Id[1:0]	Vehicle identifier
current_time[7:0]	Current time

Outputs	Description
Entry_time[7:0]	Vehicle entry time

Implementation

The module uses an array `storage_Time` to store the entry time of each vehicle. When receiving a `write_enable` signal, it stores `current_time` at the location specified by `Car_Id`. When receiving a `read_enable` signal, it retrieves the stored entry time and outputs it on `Entry_time`.

5. Timer

Description

This module is responsible for generating the current time for the system.

Interface

Inputs	Description
clk	Clock signal
reset	Reset signal

Outputs	Description
current_time[7:0]	Current time

Implementation

The module uses a register `current_time` to track time. It increments the time by 1 with each clock pulse, and resets it to 0 when receiving a `reset` signal.

6. Clock_Divider

Description

This module is responsible for dividing the main clock frequency to provide a slower clock signal for the Timer module.

Interface

Inputs	Description
clk	Main clock signal
reset	Reset signal

Outputs	Description
clk_out	Divided clock signal

Implementation

The module uses a counter `counter` to divide the clock frequency. It increments the counter with each clock pulse, and when it reaches a specified value (6,249,999), it toggles `clk_out` and resets the counter to 0.

7. Ccost (Cost Calculation)

Description

This module is responsible for calculating the parking fee based on the difference between the entry time and the current time.

Interface

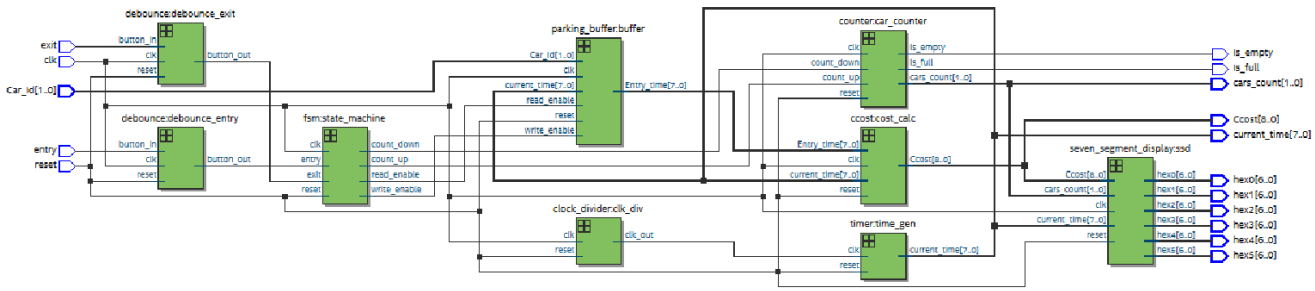
Inputs	Description
clk	Clock signal
reset	Reset signal
current_time[7:0]	Current time
Entry_time[7:0]	Vehicle entry time

Outputs	Description
Ccost[8:0]	Parking fee

Implementation

The module calculates the fee based on the difference between `current_time` and `Entry_time`. If `current_time` is greater than or equal to `Entry_time`, the fee is the direct difference. If `current_time` is less than `Entry_time` (indicating counter overflow), the fee is calculated as `(255 - Entry_time) + current_time + 1`.

Complete System Diagram



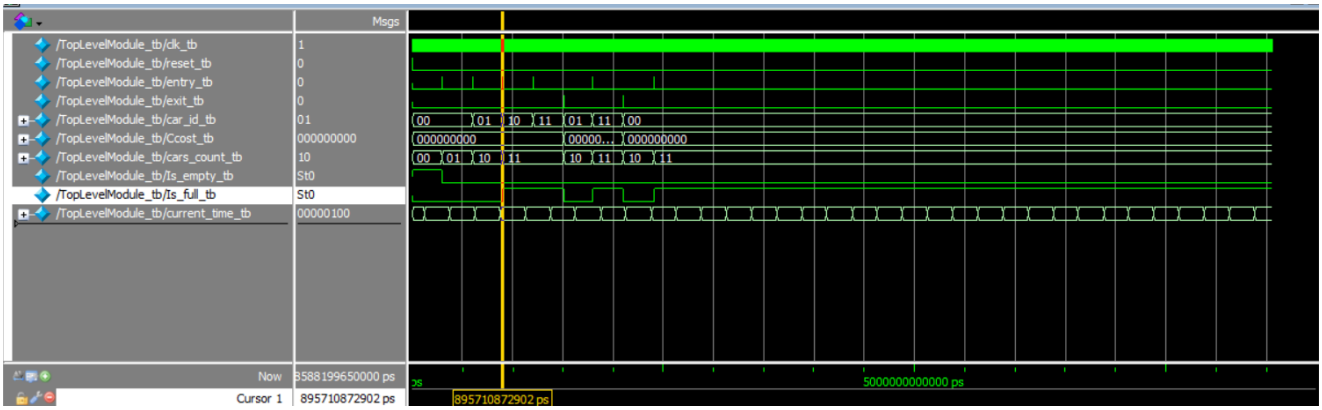
The diagram above illustrates the complete system architecture and how different modules are connected to each other. Notable features include:

1. **Debounce modules on the left side**:
2. `debounce:debounce_exit` processes the exit signal
3. `debounce:debounce_entry` processes the entry signal
4. These modules filter out unwanted signal fluctuations
5. **FSM module in the left center**:
6. `fsm:state_machine` controls the system state
7. Generates `count_up`, `count_down`, `read_enable`, and `write_enable` signals
8. **Parking_buffer module in the center**:
9. Stores vehicle entry times
10. Receives `Car_Id` and current time
11. Provides `Entry_time` upon exit
12. **Counter module in the upper right**:
13. `counter:car_counter` tracks the number of vehicles
14. Generates `is_empty` and `is_full` signals
15. Maintains `cars_count` counter
16. **Ccost module in the right center**:
17. `ccost:cost_calc` calculates the parking fee
18. Uses `Entry_time` and `current_time`
19. **Timer module in the lower right**:
20. `timer:time_gen` generates the current time
21. Depends on the divided clock signal
22. **Clock_divider module at the bottom**:
23. `clock_divider:clk_div` divides the clock frequency
24. Provides `clk_out` to the timer module

25. **Seven_segment_display module on the right side**:

26. Displays cost and current time information

Simulation Results



The image above shows the simulation results of the system, where various signals and their values during system operation can be observed:

1. **Clock and Control Signals**:
2. `/TopLevelModule_tb/clk_tb` : Main clock signal (1)
3. `/TopLevelModule_tb/reset_tb` : Reset signal (0)
4. `/TopLevelModule_tb/entry_tb` : Entry signal (0 with pulses)
5. `/TopLevelModule_tb/exit_tb` : Exit signal (0 with pulses)
6. **Vehicle ID and Count**:
7. `/TopLevelModule_tb/car_id_tb` : Vehicle identifier (01)
8. `/TopLevelModule_tb/cars_count_tb` : Number of vehicles in the parking lot (varies between 00, 01, 10, 11)
9. **Parking Lot Status**:
10. `/TopLevelModule_tb/is_empty_tb` : Empty signal (S0)
11. `/TopLevelModule_tb/is_full_tb` : Full signal (S0)
12. **Cost and Time**:
13. `/TopLevelModule_tb/Ccost_tb` : Parking fee (00000000)
14. `/TopLevelModule_tb/current_time_tb` : Current time (00000100)

Testbench Analysis

TopLevelModule_tb.v

The main testbench file simulates the complete system. It creates clock and control signals, simulates entry and exit operations, and monitors outputs such as vehicle count, parking lot status, and cost.

The test includes: 1. Generating a regular clock signal 2. Applying a reset signal at the beginning 3. Simulating multiple entry and exit operations 4. Changing the vehicle identifier to test entry time storage and retrieval 5. Monitoring changes in vehicle count, parking lot status, and cost

timer_tb.v

The Timer module testbench verifies the correct operation of the time generation unit. It generates a clock signal and monitors the increment of `current_time` with each clock pulse, and verifies its reset when applying a `reset` signal.

Data Flow in the System

Upon Vehicle Entry:

1. The entry signal passes through the debounce module to filter out fluctuations
2. The FSM receives the signal and transitions to the next state (e.g., from EMPTY to ONE_CAR)
3. The FSM generates `write_enable` and `count_up` signals
4. The Counter increments the vehicle count and updates `Is_empty` and `Is_full` signals
5. The Parking Buffer stores the current entry time for the vehicle specified by `Car_Id`

During Parking:

1. The Timer module continues to update the current time
2. The Ccost module calculates the accumulated cost based on the difference between the current time and entry time

Upon Vehicle Exit:

1. The exit signal passes through the debounce module

2. The FSM receives the signal and transitions to the previous state (e.g., from FULL to TWO_CARS)
3. The FSM generates `read_enable` and `count_down` signals
4. The Counter decrements the vehicle count and updates `Is_empty` and `Is_full` signals
5. The Parking Buffer retrieves the entry time of the vehicle specified by `Car_Id`
6. The Ccost module calculates the final fee based on the difference between the current time and entry time

Conclusion

The presented Parking Management System is an integrated digital design that provides an effective solution for managing a limited-capacity parking lot. The system features:

1. Accurate tracking of vehicle entry and exit
2. Storage and retrieval of entry times for each vehicle
3. Calculation of parking fees based on duration of stay
4. Management of parking capacity and prevention of entry when full
5. Clear indicators for parking lot status

The modular design of the system, with its division into specialized modules, makes it easily expandable and modifiable. The system can be modified to support a larger number of vehicles, change the fee calculation method, or add new features such as payment systems or license plate recognition.

The simulation results confirm the correctness of the system design and its ability to operate reliably in various scenarios, making it a practical solution that can be implemented in real-world applications.