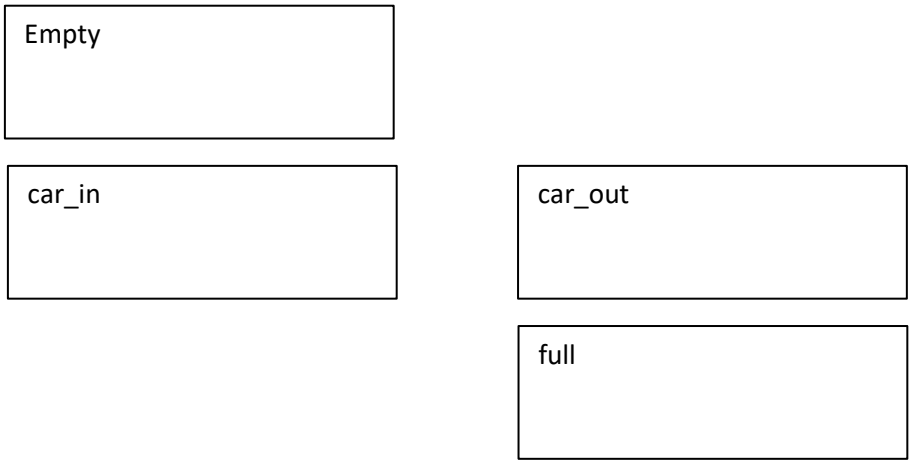
| | |
|---|---|
| محمد عمرو محمد محمود | سكشن 14 |
| محمد مصطفي محمود سلامه | سكشن 14 |
| محمد معين محي الدين | سكشن 14 |

# System block diagrame



| Input | output |
|---|---|
| reset | count |
| Car_in | leds1 |
| Car_out | leds2 |
| button | led_a |
| enable | led_b |
| clk | led_c |
| A | led_d |
| B | led_e |
| C | led_f |
| D | led_g |

Module 1

The code provided contains the design for two modules: up_counter and car_garage_FSM, which are required for the project described in the previous question.

The up_counter module is used to increment or decrement the car count within the garage. The counter is defined with 6 bits and allowed values between 0 and 50 (maximum number of cars in the garage). Input signals passed from the car_garage_FSM module are used to update the counter.

As for the car_garage_FSM module, it defines the possible states of the car garage system using a 2-bit state variable. The system state is changed based on the inputs received. The counter is updated based on the current system state and the inputs received from the buttons and cars (car_in and car_out).

If a car enters and the counter is less than 50, the counter is incremented by one and the system is passed to the Car_input state. If the counter is full (50), the counter will not be incremented when a new car enters, and a message "GARAGE IS FULL" will appear.

If the counter is greater than 0 and less than 50, cars can enter and exit the garage. When a car exits, the counter is decremented by one and the system is passed to the Car_output state. If the counter is zero, the counter will not be decremented when a car exits, and a message "GARAGE IS EMPTY" will appear.

If the counter is 50, new cars will not be allowed to enter, and a message "GARAGE IS FULL" will appear. Cars can exit the garage, and when a car exits, the counter is decremented by one and the system is passed to the Car_output state.

Module 2

This module is called "car_garage" and is used to interface with the car garage system. It has four input ports and three output ports.

The "reset" input is used to reset the system to its initial state. The "button" input is used to detect when a button is pressed or released. The "car_in" input is used to detect when a car enters the garage, and the "car_out" input is used to detect when a car exits the garage.

The "count" output is a 6-bit signal that represents the current count of cars in the garage. The "leds1" and "leds2" outputs are used to display the count on two 7-segment displays.

The module instantiates two other modules: "car_garage_FSM" and "BCD7Seg". The "car_garage_FSM" module is used to define the states of the car garage system and update the "count" variable based on the inputs received. The "BCD7Seg" module is used to display the count on the 7-segment displays.

The module also includes wires to extract the individual digits of the "count" variable (using modulo and integer division operations) and connect them to the "BCD7Seg" module. The outputs of the "BCD7Seg" module are then connected to the "leds1" and "leds2" output ports of this module.

Module 3

This module is called "DUT" (Design Under Test) and is used to test the "car_garage" module. It instantiates the "car_garage" module and provides inputs to it.

The "car_in", "car_out", "reset", and "button" signals are declared as registers, which means that their values can be changed during simulation. The "count", "leds1", and "leds2" signals are declared as wires, which means that their values are read-only and are driven by the "car_garage" module.

The "car_garage" module is instantiated as "c" and connected to the input and output ports of the "DUT" module.

In the "initial" block, the initial values of the input signals are set. The "reset" signal is set to 1 for 50 simulation time units and then set to 0. This is followed by setting the "button" signal to 1.

In the "always" block, a sequence of inputs is provided to the "car_garage" module to simulate the behavior of cars entering and exiting the garage. The input sequence consists of cars entering the garage (car_in=1, car_out=0) for 50 time units, followed by a brief pause (car_in=0, car_out=0) for 1 time unit. This sequence is repeated several times, with the last set of inputs being cars entering and exiting the garage in a complete cycle.

This module can be used to verify that the "car_garage" module is functioning correctly by observing the output signals "count", "leds1", and "leds2" during simulation.

Module 4

This module is called "BCD7Seg" and is used to generate the signals to drive a 7-segment display. It takes four input signals (A, B, C, and D) and provides seven output signals (led_a, led_b, led_c, led_d, led_e, led_f, and led_g).

The input signals represent the binary-coded decimal (BCD) value to be displayed on the 7-segment display. The output signals represent the segments of the display (a, b, c, d, e, f, and g) that need to be turned on or off to display the corresponding BCD value.

The logic for generating the output signals is based on the truth table for a 7-segment display. Each output signal is assigned a logical expression that evaluates to 0 (turn on) or 1 (turn off) based on the input signals.

The "led_a" output signal is generated based on the inputs A, B, C, and D. Similarly, the "led_b" to "led_g" output signals are generated based on different combinations of A, B, C, and D.

The logical expressions used to generate the output signals are implemented using the bitwise NOT (~) and OR (|) operators, as well as the AND (&) operator. The "assign" keyword is used to assign the logical expressions to the output signals.

Overall, this module can be used to drive a 7-segment display to display decimal digits based on the BCD input signals.

Module 5

This module is called "counter_to_decoder" and is used to control the display of the car count on two 7-segment displays. It takes four input signals (clk, reset, car_in, and car_out) and two output signals (leds and leds2).

The module includes an "up_counter" submodule, which is used to increment or decrement the car count based on the inputs received. The "count" output signal of the "up_counter" submodule is connected to two "BCD7Seg" submodules that convert the count to a format that can be displayed on the 7-segment displays.

The module also includes wires to extract the individual digits of the "count" variable (using modulo and integer division operations) and connect them to the "BCD7Seg" submodules. The outputs of the "BCD7Seg" submodules are then connected to the "leds" and "leds2" output ports of this module.

The "enable" input of the "up_counter" submodule is calculated based on the "car_in" input and a signal "full", which is not defined in this module. Presumably, "full" is defined in another module and is used to prevent the counter from incrementing when the garage is full.

Overall, this module can be used to control the display of the car count on two 7-segment displays based on the inputs received from the car garage system.

Result

The code defines a car garage system that uses a finite state machine (FSM) to control the entry and exit of cars and keep track of the number of cars parked in the garage. The FSM has four states: Empty, Car_input, Car_output, and Full, represented by the parameters Empty, Car_input, Car_output, and Full, respectively.

The car_garage_FSM module instantiates an up_counter module that counts the number of cars in the garage and a BCD7Seg module that converts the count into binary-coded decimal (BCD) and displays it on two seven-segment displays.

The car_garage module combines the car_garage_FSM and the BCD7Seg modules, and it also includes inputs and outputs for buttons and LEDs.

Finally, the DUT module instantiates the car_garage module and provides test inputs for simulation. The test inputs simulate cars entering and leaving the garage, and the LEDs display the count of cars in the garage in BCD format.