Faculty of Electrical Engieering

SEEE 3223 MICROPROCESSOR

CAR PARKING SYSTEM

Group:	2	
No.	Name	Matric
1.	MOHAMMED MUBARAK ALKHALIFA	A21EE3008
2.	MOHAMED SHARFI ABDELGADIR	A21EE9137
4.	ISRA ELTAYEB AHMED ELTAYEB	A21EE4016

Lecturer: Dr. Mohd Afzan Othman

1. Table of Contents.

1. Table of Contents. 2
2. Introduction. 3
3. Design Specifications. 4
4. Design Procedure. 4
   4.1 Introduction. 4
   4.2 Flowchart. 5
5. Software Implementation. 7
   5.1 Introduction. **Error! Bookmark not defined.**
   5.2 The System Program or Code. 7
   5.3 Simulation. 10
6. Hardware implementation. 11
   6.1 Introduction. 11
   6.2 Discussion. 11
   6.3 Schematic. 12
7. Discussion. 13
8. Conclusion. 13
9. References. 14

2. Introduction.

The ATmega32 microcontroller features a set of input/output ports that play a crucial role in interfacing with external devices. These ports, labelled as PORTA, PORTB, PORTC, PORTD, provide a convenient way to connect and control peripherals. Each port consists of 8 pins, allowing for a total of 32 I/O pins across the four ports.

Developers can configure these ports as either inputs or outputs based on project requirements. The microcontroller's C programming environment, supported by Microchip's software, simplifies the manipulation of these ports through well-defined registers and libraries.

For instance, by utilizing the DDRx (Data Direction Register) for each port, developers can set whether a particular pin is an input or an output. The PORTx register allows for reading or writing values to the port, while the PINx register provides information about the current state of the pins.

Understanding and effectively utilizing these ports is fundamental for interfacing with components like IR sensors in the parking system project. Proper configuration of ports enables seamless communication and control between the ATmega32 and external devices, contributing to the overall success of the embedded systems project.

3. Design Specifications.

In this phase, we meticulously outlined the design specifications that form the bedrock of our project. These specifications articulate the specific requirements and functionalities anticipated from our system. Key aspects covered in the design specifications include:

Functional Requirements:

Clearly identifying how our system will operate, detailing the counts and functions required with the sensors, and elucidating the approach for displaying numbers on the seven-segment display.

Input and Output Requirements:

Explicitly specifying the types and quantity of inputs and outputs required, including details on the connections, designated ports for input and output, and how they will be utilized. It's important to note that the ports for simulation or software implementations may differ from those used in hardware implementations due to challenges encountered during the hardware implementation phase.

Hardware and Software Constraints:

Enumerating the software and hardware components essential for the system, including the required applications and components. Additionally, addressing power supply considerations for our circuit and identifying any sensors that may be necessary for optimal system functionality.

This comprehensive detailing of design specifications serves as a guidepost, providing a clear roadmap for the development and implementation of our project.

4. Design Procedure.

4.1 Introduction.

Our design procedures commenced by breaking down the main system or program into separate functions. This approach facilitated the distribution of tasks among group members, enabling a more organized and collaborative development process.

Task Distribution Meeting:

To ensure clarity and alignment, a comprehensive meeting was conducted to discuss how each group member would tackle their assigned task. The focus was on understanding the design of each function within the program.

Task Execution Plan:

Following the task distribution, the team outlined a structured plan to guide the development process:

Flowchart Design:

The initial step involved creating a flowchart for the entire system. This visual representation helped in understanding the logical flow of the program and the interaction between different functions.

Circuit Schematic in Proteus:

With the flowchart as a guide, the team proceeded to design the circuit schematic using Proteus. This step allowed for a virtual representation of the hardware, aiding in the identification of potential issues and ensuring compatibility.

Simulation and Troubleshooting:

The designed circuit underwent simulation to validate its functionality. During this phase, the team actively troubleshooted the code and addressed any discrepancies between the expected and observed behavior.

Hardware Implementation:

After successful simulation and code troubleshooting, the next step involved applying the code to the actual hardware. This hardware implementation phase provided a real-world test of the designed system.

By following this systematic approach, the team aimed to streamline the development process, ensuring that each component of the program was well-designed, thoroughly simulated, and robustly implemented on the hardware. This comprehensive methodology aimed to enhance efficiency, collaboration, and the overall success of the project.

4.2 Flowchart.

The flowchart underwent two stages of refinement: initially, a general version was created, followed by a hardware-friendly version. The latter utilized flags and a more detailed chart to address specific issues. It's worth noting that these changes were implemented due to our initial lack of familiarity with interrupts. Subsequently, the integration of interrupts helped simplify our flowchart.

Scenario: Both Sensors Detect a Vehicle:

The original chart checked the first sensor for vehicle detection, initiating a delay and setting an LED high. However, if the second sensor also detected a vehicle simultaneously, the code failed to react and continued decreasing the space until the first sensor ceased detecting a vehicle, revealing a flaw in the logic.

The solution involved modifying the chart to first check if the other sensor detected a vehicle before initiating the delay. This adjustment ensured that the code responded appropriately when both sensors detected a vehicle.

Scenario: Prolonged Sensor Activation:

Even after addressing the initial issue, a new challenge emerged: what if a sensor detected a vehicle for an extended duration? The chart would continually decrease or increase the parking space, presenting an undesirable outcome.

To mitigate this problem, flags were introduced. The chart or code would only start modifying the parking space if the sensor was activated and the flag indicated that it was a new vehicle. Afterward, the code would refrain from making further adjustments until the sensor returned to zero, confirming that the car had passed. The flag would then be reset, allowing the process to be repeated. This approach ensured more controlled and accurate tracking of vehicle presence.

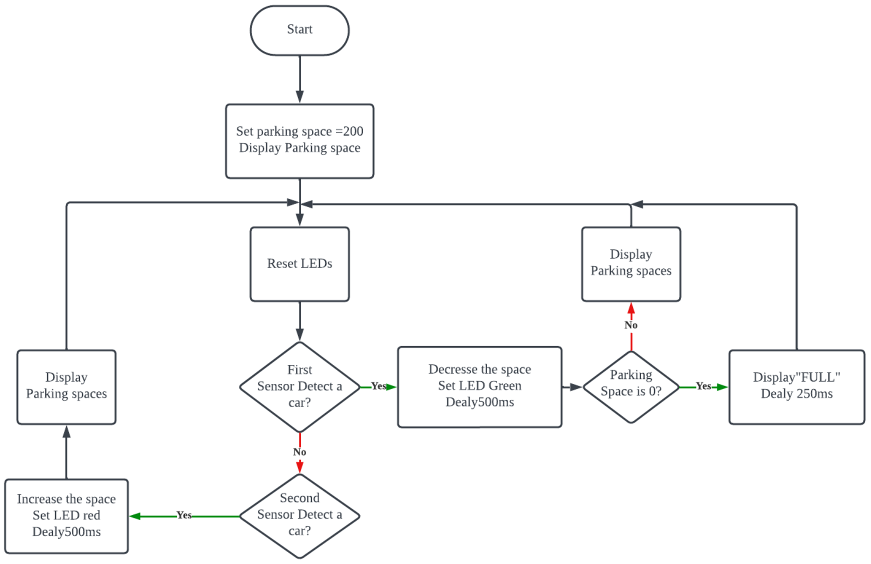


Figure 4-1 Flowchart using Interrupt.

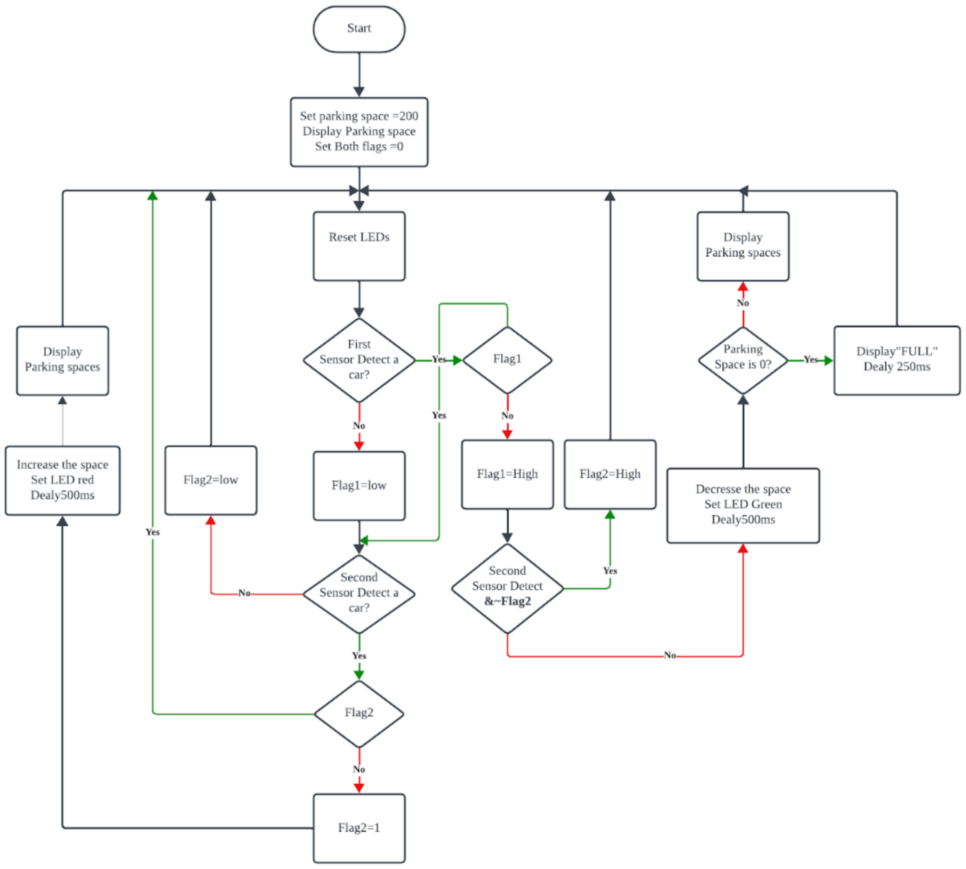


Figure 4-2 Flowchart using GPIO

5. Software Implementation.

5.1 The System Program or Code.

We divided the code into sub functions so that will ease the process.

First declared all libraries that we want in the top of the code like (input and output, delay and interrupt libraries)

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#define F_CPU (1000000)
#include <stdio.h>
```

Then defined variables to enable LEDs in 7-segmnent display like SEG_A will enable A led in 7-segment.

```
// Define the segments for common anode 7-segment display
#define SEG_A   0b00000001
#define SEG_B   0b00000010
#define SEG_C   0b00000100
#define SEG_D   0b00001000
#define SEG_E   0b00010000
#define SEG_F   0b00100000
#define SEG_G   0b01000000
#define SEG_DP  0b10000000
```

Then defined the ports, we chose PORTA as common port which decided which display turns on and PORTC for display LEDs.

```
// Define the common pins for the displays
#define COMMON_PORT  PORTA
#define COMMON_DDR   DDRA
// Define the display pins
#define DISPLAY_PORT  PORTC
#define DISPLAY_DDR   DDRC
// Number of 7-segment displays
#define NUM_DISPLAYS 4
```

Then declare the variables.

```
// Array to hold digits for each display
int a,b,c;
int parking;
int digitsToDisplay[];
parking=200;
```

After that we declared the interrupt (INT0 for decreasing and INT1 for increasing)

```
ISR (INT0_vect){
       led(0);
       parking=parking-1;}
ISR (INT1_vect){
       led(1);
       parking=parking+1;}
```

This function declared to make LEDs blinking with any increase or decrease with 500ms delay.

```
void led (int x){
	if(x==0){
		PORTB=PORTB | 1;
		_delay_ms(500);
		PORTB=0;
	}
	if(x==1){
		PORTB=PORTB | (1<<1);
		_delay_ms(500);
		PORTB=0;
	}
}
```

Also then function for the 7-segment display which makes numbers displayed in 7-segment like when digit =0, in 7-segment will display 0.

```
/ Function to display a digit on a 7-segment display
void displayDigit(uint8_t digit) {
	turn();
	switch (digit) {
		case 0: DISPLAY_PORT = SEG_A | SEG_B | SEG_C | SEG_D | SEG_E | SEG_F; break;
		case 1: DISPLAY_PORT = SEG_B | SEG_C; break;
		case 2: DISPLAY_PORT = SEG_A | SEG_B | SEG_G | SEG_E | SEG_D; break;
		case 3: DISPLAY_PORT = SEG_A | SEG_B | SEG_C | SEG_D | SEG_G; break;
		case 4: DISPLAY_PORT = SEG_B | SEG_C | SEG_F | SEG_G; break;
		case 5: DISPLAY_PORT = SEG_A | SEG_C | SEG_D | SEG_F | SEG_G; break;
		case 6: DISPLAY_PORT = SEG_A | SEG_C | SEG_D | SEG_E | SEG_F | SEG_G; break;
		case 7: DISPLAY_PORT = SEG_A | SEG_B | SEG_C; break;
		case 8: DISPLAY_PORT = SEG_A | SEG_B | SEG_C | SEG_D | SEG_E | SEG_F | SEG_G;
break;
		case 9: DISPLAY_PORT = SEG_A | SEG_B | SEG_C | SEG_D | SEG_F | SEG_G; break;
		default: DISPLAY_PORT = 0; break;  // Turn off all segments for unknown digit
	}
}
```

For the last global function we have turn() which separates the 3 numbers digits in (a,b and c) variable

```
void turn () {
	a = parking /100;
	b = (parking % 100) / 10;  // Tens digit
	c = parking % 10;  // Ones digit
	digitsToDisplay[1] = b;
	digitsToDisplay[2] = c;
	digitsToDisplay[0] = a;
}
```

Then we started with main code, for GICR for enable interrupt and MCUCR for failing edge

```
int main() {

	GICR = (1<<INT0);
	GICR =GICR | (1<<INT1);

	MCUCR |= (1 << ISC01);
	MCUCR &= ~(1 << ISC00);  // Clear the bit

	MCUCR |= (1 << ISC11);
	MCUCR &= ~(1 << ISC10);
       sei();}
```

After that we set the pins as outputs

```
        DDRB= 0xFF;
        COMMON_DDR = 0xFF;  // Set common pins as outputs
        DISPLAY_DDR = 0xFF; // Set display pins as outputs
```

Then we started with infinity loop [while(1)],inside this loop we made if condition which started with case the number of parking is 0 so it will display “FULL” in 7-segment using case function. inside if condition we have For loop to move around the 4 7-segmnet displays.

```
while (1) {
                turn();
                if (parking <=0){
                        // Display "L," "U," "F" on each 7-segment display
                        for (uint8_t display = 0; display < 4; ++display) {
                                COMMON_PORT = 1 << display; // Activate one display at a time
                                switch (display) {
                                        case 0: // "L"
                                        DISPLAY_PORT = SEG_E | SEG_F | SEG_D ;
                                        break;
                                        case 1: // "L"
                                        DISPLAY_PORT =SEG_E | SEG_F | SEG_D  ;
                                        break;
                                        case 3: // "U"
                                        DISPLAY_PORT = SEG_B | SEG_C | SEG_D | SEG_E | SEG_F;
                                        break;
                                        case 2: // "F"
                                        DISPLAY_PORT = SEG_A | SEG_E | SEG_F | SEG_G;
                                        break;
                                }
                        }
```

In the last part of the code is else condition inside it we have For loop to move around the 3 displays with 20ms delay and displayDigit(digitsToDisplay[display]) function to display the values of a, b and c variable in the 7-segment displays.

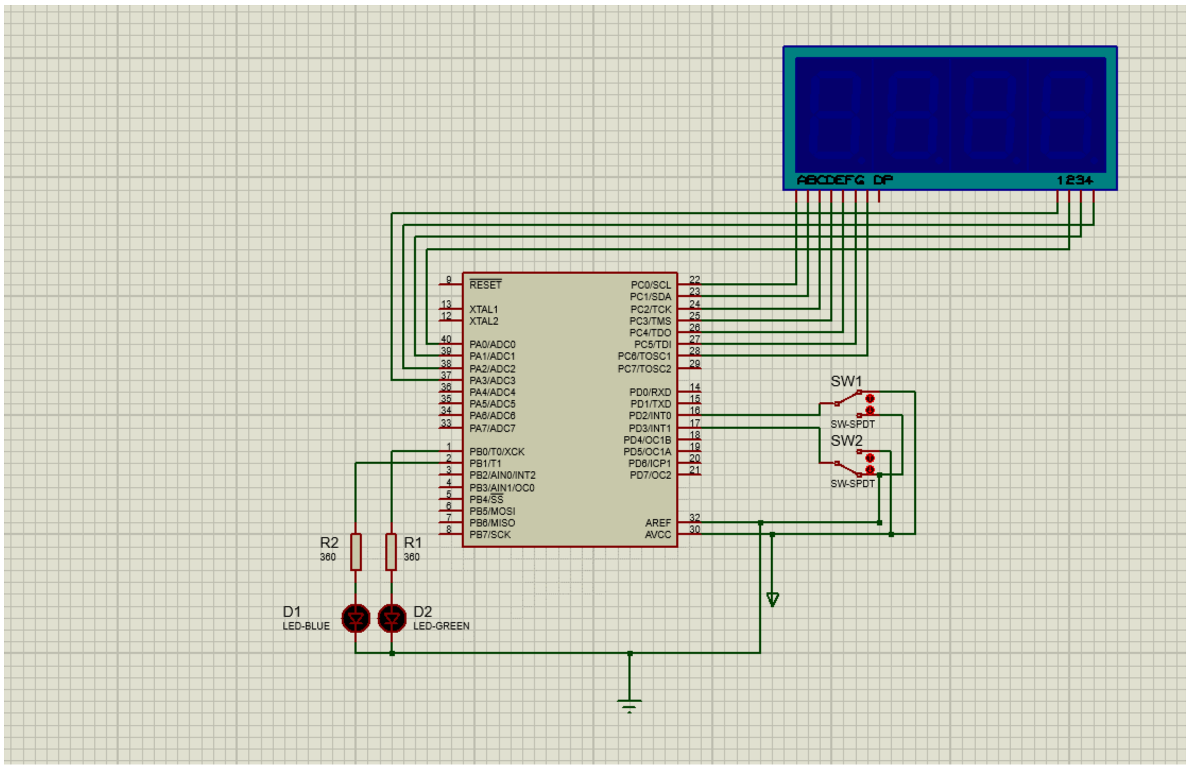
```
else {
                        for (uint8_t display = 0; display < 3; ++display) {

                                COMMON_PORT = ~(1 << display); // Activate one display at a time
                                displayDigit(digitsToDisplay[display]);
                                _delay_ms(20);

                        }
                }
        }
return 0;}
```

5.2 Simulation.

For the simulation part we used **Proteus 8 Demonstration software** which is easy to use and understand.

We should put **parking = 200** to show the function of increasing and decreasing (main task) but we put the value to 3 to show the case of **parking = 0** which display “FULL” in 7-segments and also the function of increasing and decreasing .

Circuit connection:



This the video link for the simulation:

https://youtu.be/tk3uocjjb_Q

6. Hardware implementation.

6.1 Introduction.

The hardware implementation phase involved transforming the simulation into a real-life application. Utilizing the ATmega32 development board, we successfully burned the program hex files onto the ATmega32A chip after installing necessary drivers.

Main Components Used in the Circuit:

Hardware:

- Atmega32 development board
- Atmega32A chip
- USB UART cable
- 2.2uF capacitors
- 330-ohm resistors
- 2 LEDs (green and red)
- 2 IR sensors
- Seven segment display (SH5461AS)

Software:

- Microchip studio (AVR STUDIO 7).
- AVRDUDESS.
- EasyEDA

6.2 Discussion.

Applying the hardware presented a considerable challenge for us, and we encountered several issues during the process.

Interrupt Freeze: We faced a freezing issue in the circuit during interrupts. This problem arose because, in the Proteus simulation software, there was no need to enable the falling-edge interrupt explicitly. The simulation automatically triggered the interrupt on the falling edge. However, when applying the same code to the hardware, configuring it for the falling edge became necessary for proper circuit functionality.

Floating Pin Issue: Prior to using the IR sensor, we experienced a problem where failing to pull up the interrupt pin resulted in an undefined state. This led to erratic behavior in our circuit. The problem was resolved by introducing a pull-up resistor to the interrupt pin or by incorporating the IR sensor later in the circuit.

IR Sensor Multiple Enabling: The IR sensor was found to enable input multiple times due to an effect like debouncing. Minor changes in the infrared signal caused output variations. This issue was mitigated by implementing a 2.2uF capacitor for smoothing.

Segment Activation Problem: When applying the code used in the software to the hardware, we discovered that only half of the seven-segment display lit up, as indicated in figure 6-1. Investigation revealed that pins 3, 4, 5, and 6 of Port C were not functioning correctly. The reason for this discrepancy remained unknown. Whether used as active low or high, the results were the same. Although there was an output voltage, the LED failed to light up, possibly due to a lack of current.

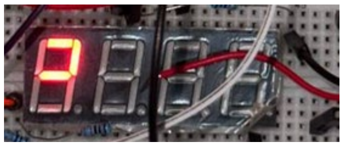
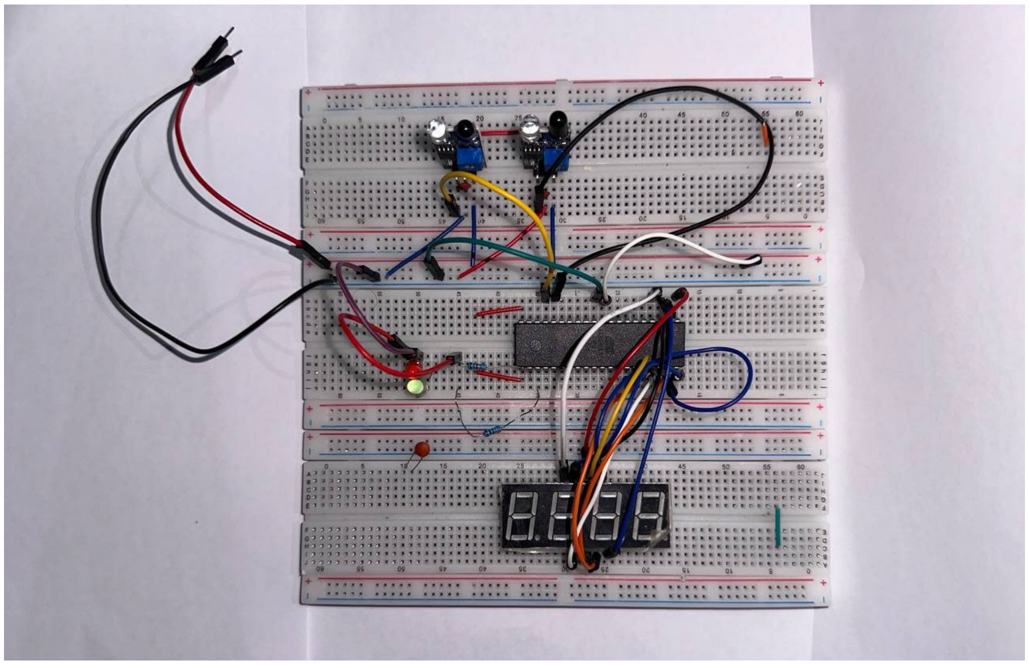
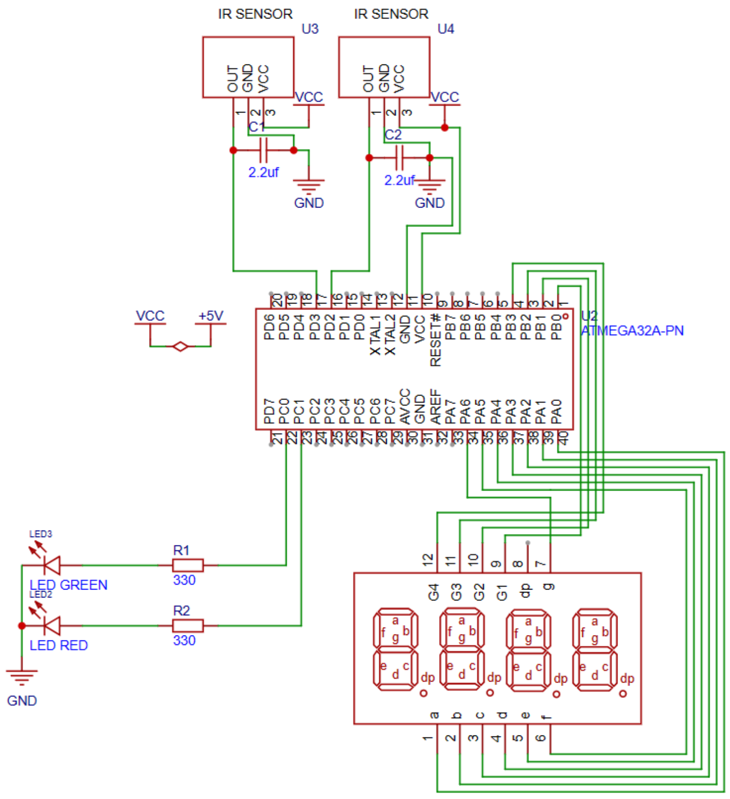


Figure 6-1 Port C Dysfunction

E in Seven Segment Display Dysfunction: It was uncovered that there was a short circuit between E and D1. Consequently, we did not connect E to ATMEGA32. As observed in the video, whenever Display 1 illuminated, it indicated that the selector was grounded. Since it is a common cathode display, the microcontroller would output 0 to D1 or G1, lighting up the display but grounding E. Conversely, if another display was enabled, the first display would set high, leaving E in a high state.

6.3 Schematic.




IR SENSOR
U3
IR SENSOR
U4
OUT GND VCC
VCC
C1
2.2uf
GND
C2
2.2uf
GND
VCC
+5V
U2
ATMEGA32A-PN
PD6 PD5 PD4 PD3 PD2 PD1 PD0 XTAL1 XTAL2 GND VCC RESET# PB7 PB6 PB5 PB4 PB3 PB2 PB1 PB0
PD7 PC0 PC1 PC2 PC3 PC4 PC5 PC6 PC7 AVCC GND AREF PA7 PA6 PA5 PA4 PA3 PA2 PA1 PA0
LED3
LED GREEN
LED2
LED RED
R1
330
R2
330
GND
G4 G3 G2 G1 dp g
a b c d e f

Video link for hardware: https://youtube.com/shorts/1sfywgIIpC0?si=rzouxyJESjkznBiQ

7. Discussion.

In the design process, the flowchart underwent two refinement stages to address issues stemming from a lack of familiarity with interrupts. The hardware-friendly version incorporated flags and a detailed chart, adapting to scenarios such as simultaneous vehicle detection and prolonged sensor activation. These refinements showcased the importance of adaptability and careful consideration in ensuring the code's responsiveness and accurate tracking of vehicle presence.

The hardware implementation of our car parking system encountered substantial challenges. Notably, an interrupt freeze during circuit operation required explicit falling-edge interrupt configuration for proper functionality. Issues related to IR sensors, including multiple enabling and segment activation discrepancies, were mitigated through strategic solutions such as the introduction of a pull-up resistor and a 2.2uF capacitor for smoothing. Additionally, dysfunction in the "E" segment revealed a short circuit, leading to the decision not to connect it to ATMEGA32.

These challenges underscored the complexity of hardware integration and provided valuable insights for future projects. The combination of flowchart refinement and hardware troubleshooting exemplifies the iterative nature of system development, emphasizing adaptability and meticulous consideration for successful implementation.

8. Conclusion.

In conclusion, our team successfully navigated the design process, ultimately delivering a system aligned with the specified requirements. Along the way, we encountered and resolved several challenges during the hardware implementation phase, leading us to a crucial realization. While IR sensors proved effective for a small-scale prototype, their limitations became apparent when considering real-life applications. In response, we recommend the integration of two ultrasonic sensors at each exit – one dedicated to car detection and the other forming a protective loop to ensure gate security and feedback.

Moreover, our consideration extended to potential enhancements for the prototype. Contemplating an additional feature, we explored the incorporation of servo motors to control gates. Unfortunately, due to time constraints, this supplementary add-on remained unrealized. The project's journey underscored the importance of adapting sensor technologies to the practical demands of real-world scenarios and prompted thoughtful consideration of further augmentations for future iterations.

9. References.

1. The art of digital system Design by *(Munim Zabidi, Izam Kamisian , Isamhani Ismail)*
2. *Video link for software simulation:* https://youtu.be/tk3uocjjb_Q
3. *Video link for hardware: https://youtube.com/shorts/1sfywgIIpC0?si=rzouxyJESjkznBiQ*
4. *SH5461AS Datasheet PDF - 4 Digit, 7-Segment Display Datasheetcafe.com https://www.datasheetcafe.com/sh5461as-datasheet-pdf/*
5. *ATmega32 - Microchip Technology https://www.microchip.com/en-us/product/ATmega32*
6. *Debounce Circuit https://circuitdigest.com/electronic-circuits/what-is-switch-bouncing-and-how-to-prevent-it-using-debounce-circuit*