Middle East Technical University
Department of Computer Engineering

CENG 336

Spring 2024-2025

Take Home Exam 3

Introduction

Due: Wednesday, May 28th 2025, 23:59.

Submission: via ODTUClass.

Late Policy: You can extend the deadline by 3 days maximum. If you extend the deadline by n days, you will receive a grade penalty of $5n^2$.

In this assignment, you are expected to implement a parking lot simulation by communicating the PIC microcontroller with a Python program running on a host computer. You are tasked with managing a multistorey parking lot. Your program will direct cars from a queue to available parking spots and offer parking subscriptions. It must also report the current status of the parking lot and calculate parking fees when cars request to leave. Communication with the cars will occur via a serial connection.

The purpose of this assignment is to familiarize you with the Analog-to-Digital Converter (ADC) and Universal Synchronous Receiver Transmitter (USART) modules of our development kits. Any clarifications and revisions to the assignment will be posted to ODTUClass.

Hand Out Instructions

- `simulator.zip`: simulator source files.
- `switchConfiguration.png`: A snapshot of the board with the desired switch configurations of this assignment
- `sample.hex`: A sample .hex file to test the connection between the simulator and the PC.

Simulator

The simulator is written in Python and requires Python 3.10 or higher, with `pygame` and `pyserial` libraries to be installed on your system. The simulator is provided to you in the simulator.zip file.

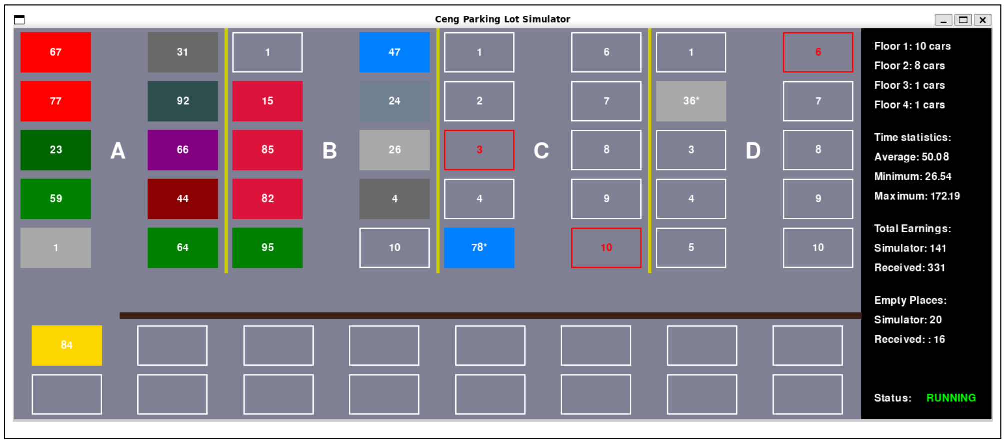
To install `pygame` and `pyserial` you can use the following command:

```
sudo apt-get install python3-pip
pip install pygame pyserial
```

After installing the tools or on Inek machines, you can run the simulator by typing the following command:

```
python3 cengParkSimulator.py
```

Before running the simulator, make sure that your device file name for the serial port is the same as the `DEFAULT_PORT`. If it says 'permission denied', then allow user access to the device file with the chmod command.



```
Floor 1: 10 cars
Floor 2: 8 cars
Floor 3: 1 cars
Floor 4: 1 cars

Time statistics:
Average: 50.08
Minimum: 26.54
Maximum: 172.19

Total Earnings:
Simulator: 141
Received: 331

Empty Places:
Simulator: 20
Received: : 16

Status:    RUNNING
```

Figure 1: Screenshot of the simulator.

Figure 1 displays a screenshot from the simulator. In the simulator screen, we see:

- The average, maximum, and minimum durations between every two messages sent from the PIC
- The current money saved from parking fees
- The status of parking spots across the four levels of the parking lot: A, B, C, and D. The capacity of each level is 10 cars.
  - Full Parking Space: Colored boxes with the car's plate number displayed on top
  - Empty Parking Space: Grey boxes with the slot number displayed on top
  - Reserved Parking Space: Boxes with a red border and the slot number displayed on top
- Waiting queue for the cars
- Cars have unique IDs ranging from 0 to 999. Subscribed cars are indicated by an asterisk (*) following the car ID.
- Seconds elapsed after the simulator started

Operation Modes of The Simulator

The simulator has three operation modes:

- **WAITING**: In this mode, the simulator waits without sending commands or responding to the PIC until the user presses the 'S' button on the computer's keyboard. After the user starts the simulation, the simulator will send a *GO command* over the serial port to the PIC and switch to the RUNNING mode.

- **RUNNING**: In this mode, the simulator expects serial messages from the PIC to update the status of the parking lot. The simulator sends commands whenever a new car arrives to park or requests a subscription. The simulator remains in this mode for 60 seconds. It will then switch to the FINISHED mode. **The microcontroller is expected to send serial messages only during this mode.**
- **FINISHED**: When the simulator enters this mode, it sends an *END command* and terminates further events in the microcontroller. The serial message frequency statistics are also finalized.

The program initially starts in **WAITING** state. In the **RUNNING** state, there are two modes available:

- **Automatic mode (Regular mode):** Activated by pressing the 'A' key on the computer keyboard. In this mode, the simulator generates random events every 100 ms. Possible events include adding a car to the queue, notifying the exit of a parked car, requesting a subscription, or no event at all.
- **Manuel Mode (Debug mode):** Activated by pressing the 'M' key on the computer keyboard. In this mode, there are four buttons you can try:
  - 'R' button: Adds a random car to the queue.
  - 'T' button: Requests removal of random car from the parking lot
  - 'Y' button: Requests subscription of a random car to a random place in the parking lot.
  - 'U' button: Add a random **subscribed** car to the queue.

When the user presses the 'ESC' button on the computer keyboard in any state, the simulation screen turns off and the Python program terminates.

Important Note: Each message begins with the character `$` and ends with `#` , which serve as the delimiter and terminator, respectively. All integers within messages are represented as character strings. For example, the number 8 is sent as the ASCII character `"8"` (hexadecimal 0x38) and the number 1234 is sent as the 4-character string `"1234"`.

Commands

The commands are issued by the car simulator. You need to receive them and act according to the specifications given below.

- ***Go Command:*** When the user starts the simulation in the WAITING mode, the simulator sends a GO command over the serial port to the PIC and switches to the RUNNING mode. This command consists of the following 4 bytes: `$GO#`
- ***End Command:*** When the 60-second period is over, the simulator transitions from the RUNNING mode to FINISHED mode. This is signaled to the PIC with the END command. This command consists of the following 5 bytes: `$END#`

For the following commands, **XXX** represents the license plate number of the cars. These are ASCII characters corresponding to the digits 0 through 9.

- ***Park Command:*** This command is sent when there is a new car in the queue and consists of the following 8 bytes: `$PRKXXX#`
- ***Exit Command:*** This command is sent when a car exits the parking lot and consists of the following 8 bytes: `$EXTXXX#`
- ***Subscription Command:*** This command is sent when a subscription request for a parking space is made. This command consists of the following 11 bytes: `$SUBXXXYZZ#`

  Here **YZZ** specifies the designated parking space. The byte **Y** represents the parking level and can be one of the following: 'A', 'B', 'C', or 'D'. The following 2 bytes **ZZ** identify the specific spot on that level. These are ASCII characters corresponding to the digits 0 through 9, with parking spots numbered from "01" to "10".

Messages

The messages are issued by the PIC program. You must transmit a message every 100 ms. The message content can be one of the following:

- ***Empty Space Message:*** This message provides the current count of empty parking spaces and is sent periodically when no other message is pending. It consists of 7 bytes in the following format: `$EMPCC#`

  Here, **CC** represents the number of available spaces, encoded as ASCII characters corresponding to the digits 0 through 9. Initially, the parking lot has 40 empty spaces.
- ***Parking Space Message:*** This message informs the simulator where to direct the car. It is used in two scenarios: first, when a Park Command is received and there is an available, non-reserved parking space; second, when an Exit Command is received and at least one car is waiting in the queue. This message consists of 11 bytes: `$SPCXXXYZZ#`

  Here, **XXX** represents the license plate number of the car and **YZZ** specifies the designated parking space. The byte **Y** represents the parking level and can be one of the following: 'A', 'B', 'C' or 'D'. The following 2 bytes **ZZ** identify the specific spot on that level. These are ASCII characters corresponding to the digits 0 through 9.
- ***Parking Fee Message:*** When an Exit Command is received, you must inform the simulator of the corresponding parking fee. This message consists of the following 11 bytes: `$FEEXXXMMM#`

  Here, **XXX** represents the license plate number of the car and **MMM** is the parking fee. (see the *Parking Fee Calculation* section for details)
- ***Reserved Message:*** This message is sent when a Subscription Command is received. If the requested parking space is available (empty and non-reserved), the reservation is successful, and you must send a message containing the subscription fee of 50 TL. However, if the space is already reserved or currently occupied, you should respond with a subscription fee of 0 (encoded as '00'). It consists of 10 bytes: `$RESXXXMM#`

  Here, **XXX** represents the license plate number of the car and **MM** indicates the subscription fee, represented as ASCII characters.

In summary, in this simulation the Python program serves as a dummy terminal that sends commands to the PIC microcontroller and visualizes the current state of the parking lot based on the messages that it receives. The microcontoller keeps an internal state of the parking lot and notifies the Python program by responding to its commands via messages.

Some Clarifications

- When a new, unsubscribed car arrives and no parking spaces are available, the car should wait in a FIFO queue. Once a space becomes available, cars from the queue should be accepted in order. The queue should hold a maximum of 16 waiting cars.
- A subscription does not equate to parking the car; it simply reserves a parking space. A subscribed car may park and exit at any time. When a subscribed car parks, a Parking Space Message should be sent with the assigned space. Upon exit, a Fee Message should be sent with a fee of 0 TL. Note that the availability of the parking space is not affected by the exit of a subscribed car. There are no commands for unsubscribing.
- The simulator will not issue Exit Commands for cars that have not been parked. A car must first be parked before it can exit.

Parking Fee Calculation

When an unsubscribed car exits the parking lot, the parking fee should be calculated, and a Fee Message should be sent. Note that subscribed cars are exempt from parking fees. The parking tariff is 1 TL per **250 ms**. The fee is calculated using the following formula:

$$\text{Fee} = \frac{t}{250} + 1,$$

where t is the duration of the parking in milliseconds. For example, if a car stays for 100 ms, the fee is 1 TL. If it stays for 750 ms, the fee is 4 TL.

7-Segment Display and Button Actions

There are two display modes: one shows the total accumulated money, and the other shows the number of empty spaces on each level. You should switch between these modes using a button press.

Initially, the 7-Segment Display should remain blank, and the program should wait for a GO Command to start the simulation. Once the GO Command is received, the display should show the total accumulated money (as illustrated in Figure 2), starting at 0.

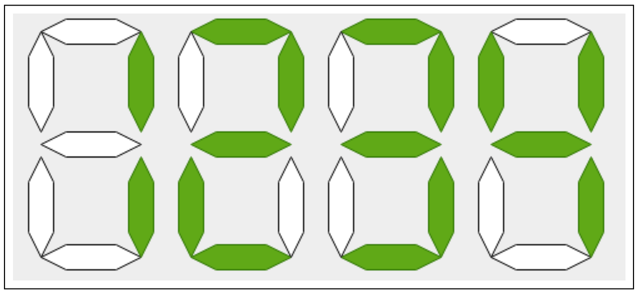


Figure 2: 7-Segment Display showing the total money earned.

You must enable switching between displaying the current money and the number of available parking spaces by releasing the RB4 button. During program execution, each release of the RB4 button should toggle the 7-Segment Display between showing the total money earned and the available parking spaces. **This functionality must be implemented using PORTB on change interrupts.**

The number of available parking spaces will be displayed on a level-by-level basis. In this mode, the rightmost 2 digits of the 7-Segment Display will be used to show the information (as illustrated in Figure 3). Level selection will be determined by the ADC module (refer to the *ADC Module* section for further details).

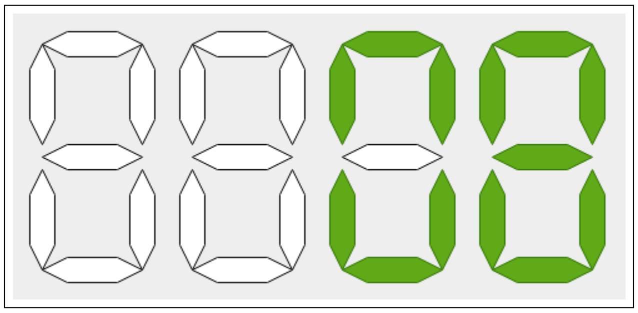


Figure 3: 7-Segment Display showing the number of empty parking spaces on the selected level.

ADC Module

- You will use a 10-bit adjusted ADC. You should also use the ADC interrupt to detect the end of conversion and read the converted value. See the datasheet for acquisition time and clock calculations.
- PORTH4 (AN12) is used for the potentiometer. Don't forget to set it as input.
- The ADC value should be 0 when you turn the ADC potentiometer counterclockwise to its leftmost position, and it should be 1023 when you turn the ADC potentiometer clockwise to its rightmost position.
- ADC potentiometer values will be mapped to levels, according to the table below:

Parking Lot Levels	ADC Range
Level A	$0 \leq X < 256$
Level B	$256 \leq X < 512$
Level C	$512 \leq X < 768$
Level D	$768 \leq X \leq 1023$

Table 1: The ADC Value to Level Mapping Table.

- You should check for a new ADC value at every **500ms**.

Detailed Specifications

- You must code your program in C and compile with the MPLAB XC8 C compiler.
- Your program should be written for PIC18F8722 working at 40 MHz.

- **You MUST properly comment your code, including a descriptive and informative comment at the beginning of your code blocks explaining your design.**
- It is guaranteed that none of the bytes between the delimiters correspond to the ASCII value of `$` or `#`.
- You should send one message at every **100ms**, with an acceptable level of accuracy. You should not block/stop sending messages.
- If there are cars waiting for Parking Space, Parking Fee, and Reserved messages, these messages should be sent in the next available 100 ms slot. If none of these messages need to be sent, an Empty Space message should be transmitted instead.
- You should implement the periodicity of your tasks with timer interrupts. You are free to choose the timer you want to work with.
- Commands sent from the simulator will be spaced at least 100 ms apart.
- You should **ignore** the commands sent from the simulator, if it does not comply with the commands described in *Commands* section.
- For your implementation to be successful, your program should be able to respond to the simulator's needs until the end command is received.
- USART settings should be 115200 bps, 8N1, no parity.

Resources

- Sample program files provided with the homework
- PIC18F8722 Datasheet
- PIC Development Tool User and Programming Manual
- Recitation Documents
- ODTUClass Discussions

Hand In Instructions

- You should submit your code as a single file named the3.zip through ODTUClass. This file should include all of your source and header files.
- By using a text file, you should write ID, name, and surname of **all group members and group number**.
- **Only one of the group members should submit the code**. Please pay attention to this, since if more than one member makes a submission this creates a confusion during grading negatively affecting the grading speed.

Grading

The total of the homework is 100 points. For grading, we will compile and load your program to the development board. Your program will be considered for grading even if it is incomplete. Your grade will also depend on the quality of your design, not just its functional correctness. If there is an unclear part in your code, we may ask any of the group members to describe that code segment. Also, group members may get different grades. We reserve the right to evaluate some or all of the groups to determine the contribution of each group member to the assignment.

- Accuracy of command timings
  - Are you able to send a command every 100 ms? How good are you at it?
- Proper usage of the ADC module
- Proper button use and correct implementation of PORTB interrupt-on-change
- Non-flickering and correct 7-Segment Display
- Proper operation of your program

will be considered while grading.

Hints

- In order to check the serial communication between the simulator and PIC, you can use `sample.hex` file. **This is not an example solution and its messages are not accurate**. Its purpose is only to check the serial communication.

Cheating

We have a zero tolerance policy for cheating. People involved in cheating will be punished according to the university regulations.

Cheating Policy: Students/Groups may discuss the concepts among themselves or with the instructor or the assistants. However, when it comes to doing the actual work, it must be done by the student/group alone. As soon as you start to write your solution or type it, you should work alone. In other words, if you are copying text directly from someone else - whether copying files or typing from someone else's notes or typing while they dictate - then you are cheating (committing plagiarism, to be more exact). This is true regardless of whether the source is a classmate, a former student, a website, a program listing found in the trash, or whatever. Furthermore, plagiarism, even on a small part of the program,m is cheating. Also, starting out with code that you did not write and modifying it to look like your own is cheating. Aiding someone else's cheating also constitutes cheating. Leaving your program in plain sight or leaving a computer without logging out, thereby leaving your programs open to copying, may constitute cheating depending on the circumstances. Consequently, you should always take care to prevent others from copying your programs, as it certainly leaves you open to accusations of cheating. We have automated tools to determine cheating. Both parties involved in cheating will be subject to disciplinary action. [Adapted from `http://www.seas.upenn.edu/~cis330/main.html`]

Using Large Language Models (LLMs) and other AI tools: We note that AI can be a useful learning resource, but if it is used for generating code that you are supposed to write, it can have a detrimental effect on your learning. Therefore, using code from other resources, whether it is generated by humans or AI, is considered cheating.