Middle East Technical University
Department of Computer Engineering

CENG 336

Spring 2023-2024

Take Home Exam 3

Regulations

Due: Thursday, May 30th 2024, 23:55.

Submission: via ODTUClass (NO LATE SUBMISSION)

The purpose of this assignment is to familiarize you with 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 simulator and PC.

Aircraft Autopilot

In this assignment, you are responsible for the flight director program that runs on the most crucial component of the autopilot system. The program will calculate the travelled distance given the speed throughout the flight and end the simulation when the remaining distance become 0. Your mission is to report the current situation of the aircraft by sending messages or to manipulate the flight after receiving messages from the autopilot. Your implementation will be communicating with the autopilot system through serial connection.

The flight will have 3 main phases: altitude change, normal cruise, and manual control. At the start of the simulation, you will receive a message that contains the total distance to the destination point. In order to have a safe flight, you need to be cautious about the reliability of the serial communication with the autopilot system. The autopilot can ask for switching to manual control or to specific altitudes, occasionally. The good news is that the autopilot system will guide you by giving some instructions to follow.

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 autopilot.py
```

Before running the simulator, make sure that you have checked your device file name for the serial port is the same with the `DEFAULT_PORT`, which is `/dev/ttyUSB0` currently, in the JSON file `autopilot-settings.json`.

If it says 'permission denied', then allow user access to device file with chmod command.

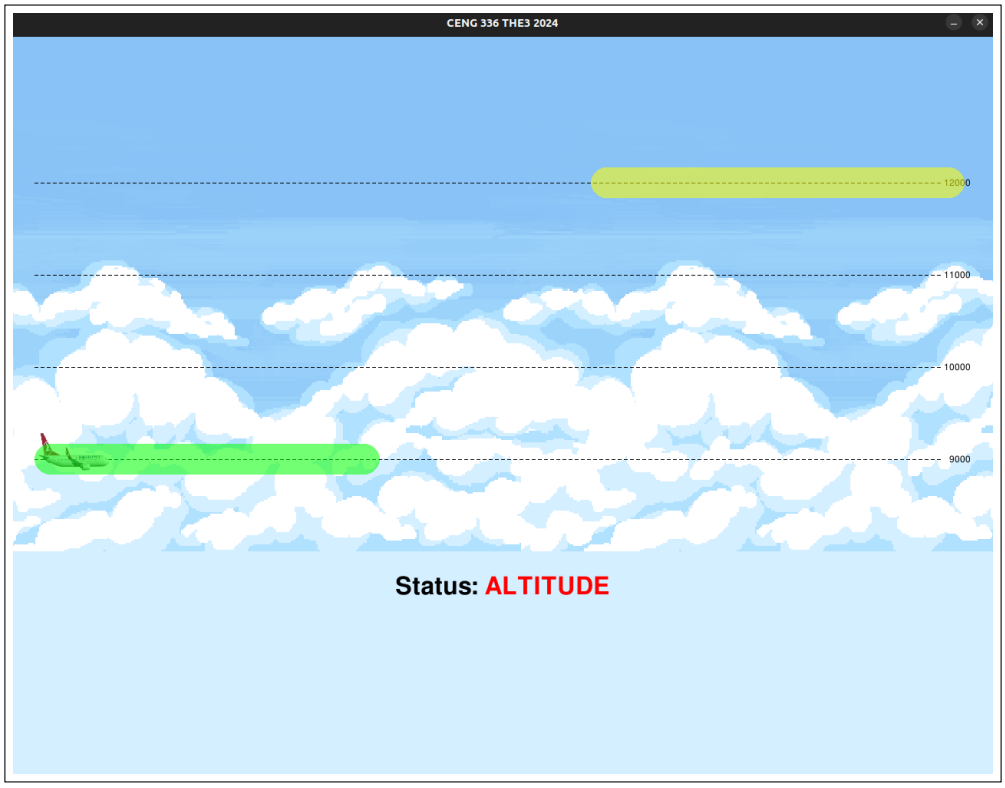

Figure 1: Screenshot of the simulator. Altitude levels and current status value are visible.

Figure 1 display a screenshot from the simulator. Top to bottom, we see:

- Altitude lines for all four altitude levels
- Yellow and green altitude zones. Green means the altitude zone was successful in the last period, yellow means the altitude zone has not arrived yet, and red (not in the screenshot) means the zone has failed in the last period.
- The plane, which is on the altitude level of 9000. This shows the latest read altitude from the board.

- Current state of the simulator, which is one of NORMAL, MANUAL, and ALTITUDE.

Operation Modes of The Simulator

The simulator has three operation modes:

- IDLE: In this mode, the simulator waits without sending commands or responding to the PIC until the user presses 's' button on the computer's keyboard. After the user starts the simulation, the simulator will send a *GO command* over serial port to the PIC and switch to the ACTIVE mode.
- ACTIVE: In this mode, the simulator expects serial commands from the PIC for controlling the flight. The simulator sends periodic messages. The changes in the flight phase will occur in this mode. It stays in this mode for 90 seconds. **The microcontroller is expected to send serial messages only during this mode.**
- END: 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.

When user presses the 'ESC' button on the computer's keyboard, the simulation program terminates.

Important Note: Each message starts and ends with the `$` and `#` characters. We will refer to `$` and `#` as the delimiter and terminator. All integers within messages are hexadecimal representation of the integers in big-endian, i.e. when we refer to number 8, we mean the string of the hexadecimal representation `'08'`. Similarly, when we mean 2748, we mean the 4 character string `'0ABC'`, not the char string `"2748"`. You may use `sscanf(output, "%04x", int_val)` and `sprintf(input, "%04x", &int_val)` functions to encode and decode integers.

Commands

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

- ***GO Command:*** When the user starts the simulation in the IDLE mode, the simulator sends a GO command over the serial port to the PIC and switches to the ACTIVE mode. This command consists of the following 9 bytes: `$GOOXXXX#`

  Here **$**, **G**, **O**, **O**, # are ASCII characters. However, **XXXX** represents the distance to the destination point. The first two **X** represents the most significant byte. You need to calculate the distance upon starting the game.

- ***END Command:*** When the remaining distance reaches 0, the simulator transitions from the ACTIVE mode to END mode. This is signaled to the PIC with the END Command. This response consists of the following 5 bytes: `$END#`

- ***Speed Command:*** This command is sent in response to the sensory responses of the PIC and consists of the following 9 bytes: `$SPDXXXX#`

  Here **$**, **S**, **P**, **D**, # are ASCII characters. However, **XXXX** represents the speed. You will be using the speed information to calculate the remaining distance. You can directly subtract it from the remaining distance.

- ***Altitude Command:*** This command is sent either to stop the altitude messages sent from the PIC or to start periodic altitude messages. This command consists of the following 9 bytes: `$ALTXXXX#`

  Here **$**, **A**, **L**, **T**, # are ASCII characters. However, **XXXX** represents the period in ms. You will be using this period information to sample ADC. You can receive one of the following period options: 0 ms, 200 ms, 400 ms, and 600 ms. For example, you should stop the ADC readings when you receive **$ALT0000#** command.

- ***Manual Control Command:*** This command is sent during the cruise phase and consists of the following 7 bytes: `$MANXX#`

  Here **$**, **M**, **A**, **N**, # are ASCII characters. However, **XX** represents the activation/deactivation switch. When you receive **$MAN01#** command, you need to enable the PORTB interrupt-on-change. The detailed information is given in the *Manual Control* section.

- ***LED Command:*** You will receive this command when the manual control is activated. This command consists of the following 6 bytes: `$LEDXX#`

  Here **$**, **L**, **E**, **D**, # are ASCII characters. However, **XX** indicates the LED that should be turned on. You can receive one of the following options: 0, 1, 2, and 3. The detailed information is given in the *Manual Control* section.

Sensor Responses

The sensor responses are issued by the PIC program. You need to transmit them periodically around 100 ms.

- ***Distance Response:*** This message informs the autopilot about the remaining distance and consists of 9 bytes: `$DSTXXXX#`

  Here **$**, **D**, **S**, **T**, # are ASCII characters. However, **XXXX** represents the remaining distance in km. You need to calculate the remaining distance with the total distance you received when the simulator sent the GO command and the periodic "SPD" commands. You will send the first distance response when the initial speed is 0.

- ***Altitude Response:*** This message is sent periodically when asked by the autopilot and consists of the following 9 bytes: `$ALTXXXX#`

  Here **$**, **A**, **L**, **T**, # are ASCII characters. However, **XXXX** represents the altitude in m. The period of this message is determined by the "ALT" command (issued by the autopilot simulator). The detailed calculation of the altitude will be given in the *Altitude Calculation - ADC Module* section.

- ***Button Press Response:*** This message informs the autopilot when a button is pressed and consists of 7 bytes: `$PRSXX#`

  Here **$**, **P**, **R**, **S**, # are ASCII characters. However, XX represents the PORTB button number and it can be [4,7]. You should issue this message when the corresponding button is **pressed and released**. The detailed information is given in the *Manual Control* section.

An Example Flight Scenario

- The flight simulation will start with a "GO" command.

- The simulator will ask for altitude readings with an "ALT" command. You will also see some altitude directions on the simulator screen.
- The simulator can ask for switching to manual control. The detailed explanation is given in the *Manual Control* section.
- The simulation will end after remaining distance reaches 0 and it will send an "END" command. After receiving the "END" command, you should immediately stop message transmission.

Altitude Calculation - ADC Module

- You will use 10-bit adjusted ADC. You should also use 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 clock-wise to its leftmost position, and it should be 1023 when you turn the ADC potentiometer clock-wise to its rightmost position.
- ADC potentiometer values will be mapped to altitude, according to the table below:

Altitude	ADC Range
12000m	$768 \leq X \leq 1023$
11000m	$512 \leq X < 768$
10000m	$256 \leq X < 512$
9000m	$0 \leq X < 256$

Table 1: The ADC Value to Altitude Mapping Table

- You should start sending the altitude readings when you receive an 'ALT' command with period other than 0.
- You should stop reading the ADC values and sending the altitude information when you receive **$ALT0000#** command.
- Your altitude responses should not disrupt the 100 ms periodicity. For example, if you receive **$ALT0190#** ('0190' string encodes 400 ms) command from the autopilot, your transmitted message pattern should look like the following:

```
$DST0258# $DST0256# $DST0254# $ALT2328# $DST0250# $DST024E# $DST024C# $ALT2328# $DST0248#
```

Manual Control

- **$MAN01#** command will activate the manual control.
- During manual control, you will receive several "LED" commands. These "LED" commands will guide you through the flight until manual control is deactivated.

- You should turn on the LED that is connected to the number you received with the "LED" command. When a LED is turned on, you need to press the connected buttons and send button press responses.

LED Command	LED	Button to press	Button Press Response
$LED00#	clear all	-	-
$LED01#	RD0	RB4	$PRS04#
$LED02#	RC0	RB5	$PRS05#
$LED03#	RB0	RB6	$PRS06#
$LED04#	RA0	RB7	$PRS07#

Table 2: The Connected LEDs and Buttons During Manual Control

- The RB[4-7] buttons will be used. You should send the button press response when a button is **pressed and released**.
- When sending the "PRS" response, you must wait for next transmission time. As in altitude responses, button press responses should not disrupt the 100 ms periodicity.
- When a matching "PRS" response sent, a "$LED00#" command wil be sent to turn off the LEDs.
- You should control RB4 to RB7 button actions in the interrupt routine.
- **$MAN00#** command will deactivate the manual control.
- You will not receive any "LED" commands if manual control is deactivated. Initially, the manual control is deactivated.
- If manual control is **not** activated, RB[4-7] button presses should not send any sensor responses.

Detailed Specifications

- You must code your program in C and compile with 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 ascii value of `$` or `#`.
- You should send your responses at every **100ms**, with an acceptable level of accuracy. You should not block/stop sending sensor responses.
- You should implement the periodicity of your tasks with timer interrupts. You are free to choose the timer you want to work with.
- Manual control will not overlap with the altitude responses. So, "DST" responses can be sent with either "ALT" or "PRS" responses.

- When calculating with the given speed, you can assume that distance will result as an integer.
- 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 autopilots needs until the end command is received.
- USART settings should be 115200 bps, 8N1, no parity.

Resources

- Sample program files provided with 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 as 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 make submission, speed of grading process will be negatively affected.

Grading

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 depends 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 member 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 at every 100 ms, how much are you good at it?
- Proper usage of the ADC module
- Proper button use and correct implementation of PORTB interrupt-on-change
- Proper operation of your program

will be considered while grading.

Hints

- In order to check the serial communication between 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 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 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 upon 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]