Lab #2: Introduction cooperative multitasking using FSM

Leonardo Fusser (1946995)

Objective:

- Get familiar with state machines and cooperative multitasking.
- Structure a real time system program in team.
- Write a UML Finite State Machine.
- Run two SMs simultaneously.

Hardware: Explorer16/32, Rotary O click, 2 USB cables,
Jumper J33 must be set aside.

To hand in:

- See previous lab: these sheets on **Teams** and C code on **GitHub**.
- UML SM diagram.

Constraints:

- All non-automatic variables must be declared static to make them local to the file, i.e., private.
- All private variables must be accessible through public functions only. This way, global broadcast is replaced by public setter-getter functions.

Table 1: examples of file contents

Files	**Content**
`initBoard.c`	Includes its own header file: `initBoard.h` Contain all functions related to initializing the board. Refer to previous labs.
`encoder.c`	Contains implementation of the SM to decode the encoder pulses. Public Setter and Getter function definitions. External global variables are prohibited.
`idle.c`	Contains implementation of a SM that blinks an LED as an indication of how busy the CPU is. A high frequency LED blink would indicate a NOT very busy CPU and a low frequency would indicate a very busy CPU. Hint: use a skip counter.
`soft_com.c`	Contains all function implementations for a software serial RS232 transmission using state machine. Must encapsulate as much as possible.
`queue.c`	Contains queue library.
`initBoard.h`	Refer to previous lab.

`soft_com.h`
`encoder.h`
`idle.h`
`queue.h`
	Includes all public dependencies, public macros, public function prototypes and structure definitions.

`public.h`	Includes all public macros. E.g., LEDs macros, mode macro
	`#ifdef SIMULATION`
	…
	`#else`
	…
	`#endif`
	
	`#ifdef WITH_QUEUE`
	…
	`#else`
	…
	`#endif`

Rotary click theory

Rotary click carries a 15-pulse incremental rotary encoder with detents, surrounded by a ring of 16 LEDs.

Features and usage notes

Rotary encoder:
A single rotation is divided into 15 discrete steps. The encoder outputs A and B signals (out of phase to each other). Two lines are needed for outputting the Encoder info: ENCB OUT and ENCA OUT.

Figure 1 shows the line connections between the Explorer 16/32 board and the Rotary click board.

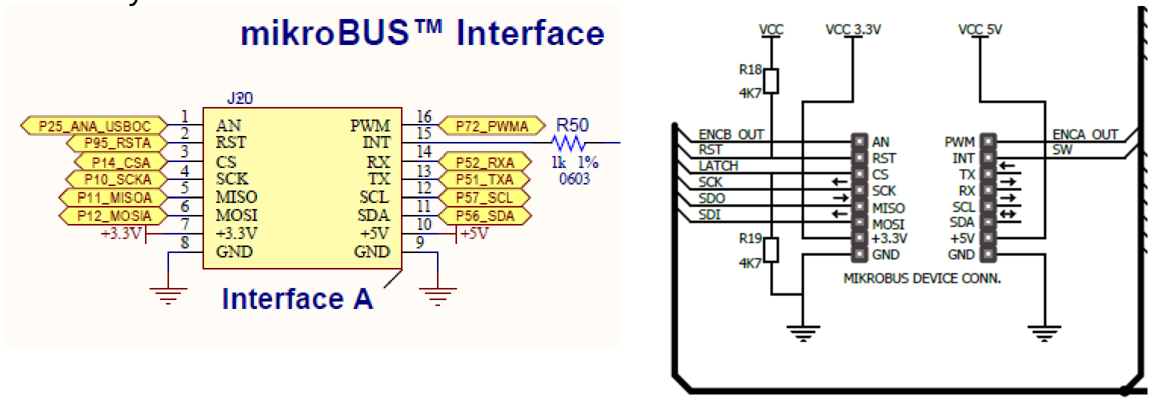


Figure 1 MicroBus interface A on Explorer 16/32 board (left) and the Rotary click pinout (right).

Lab work:

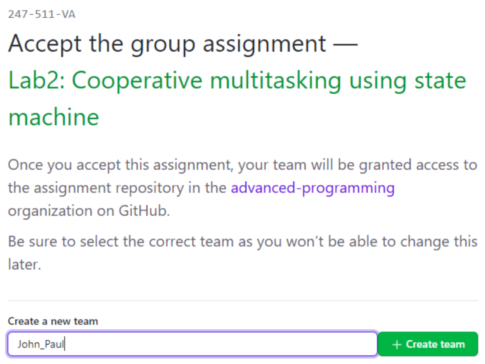
Part1: Teamwork using GitHub

1. Make sure you are logged into GitHub. Copy-paste the following URL to accept an invitation for Lab2_511_A11:

   https://classroom.github.com/g/FGnmqUXW

2. You can **Join an existing team OR create a team** (e.g., John_Paul). The other member of your team will of course join the same team.

   Create a new team name:



247-511-VA

Accept the group assignment —
Lab2: Cooperative multitasking using state machine

Once you accept this assignment, your team will be granted access to the assignment repository in the advanced-programming organization on GitHub.

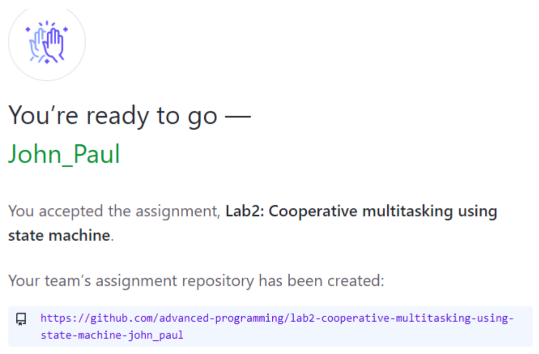
Be sure to select the correct team as you won't be able to change this later.

Create a new team

John_Paul

3. You will get a private new repository for your lab:

https://github.com/advanced-programming/lab2_511_a11- team-name



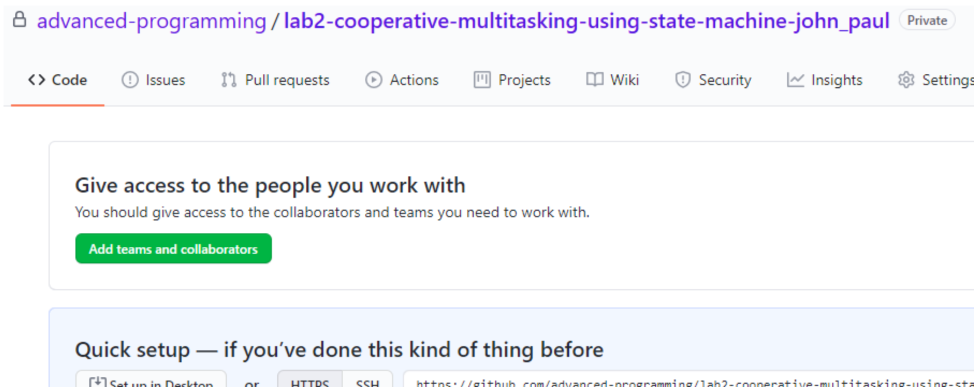
You're ready to go —
John_Paul

You accepted the assignment, **Lab2: Cooperative multitasking using state machine**.

Your team's assignment repository has been created:

https://github.com/advanced-programming/lab2-cooperative-multitasking-using-state-machine-john_paul

4. To add a collaborator, click Add teams and collaborators:



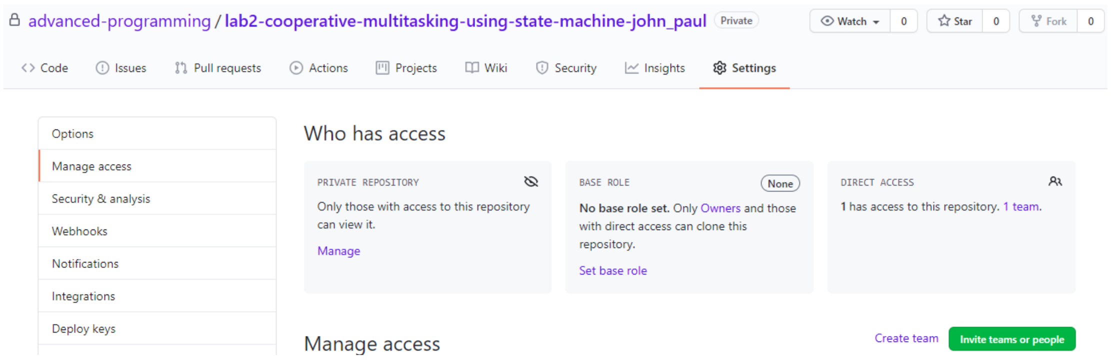
advanced-programming / lab2-cooperative-multitasking-using-state-machine-john_paul Private

Give access to the people you work with

You should give access to the collaborators and teams you need to work with.

Quick setup — if you've done this kind of thing before

5. To invite people, click Invite teams or people:



advanced-programming / lab2-cooperative-multitasking-using-state-machine-john_paul Private

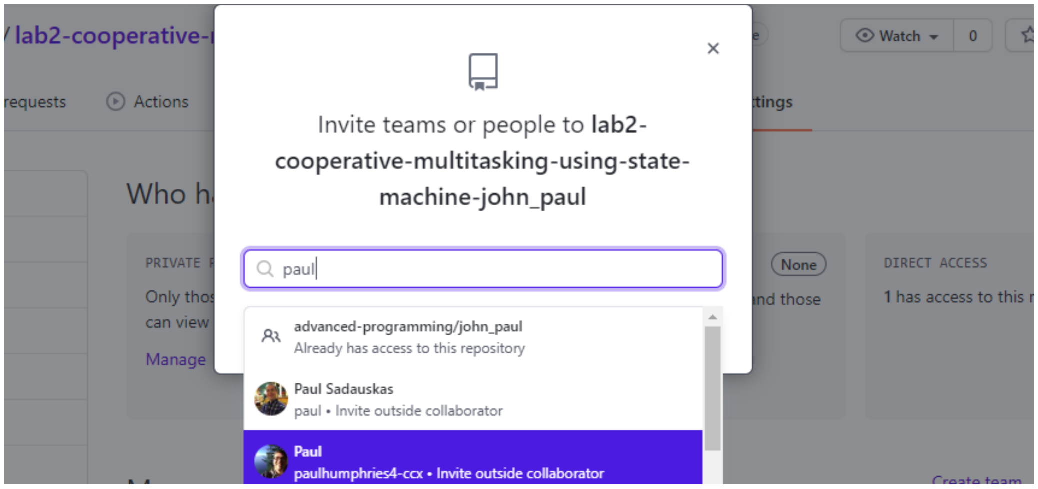
Who has access

PRIVATE REPOSITORY
Only those with access to this repository can view it.
Manage

BASE ROLE None
No base role set. Only Owners and those with direct access can clone this repository.
Set base role

DIRECT ACCESS
1 has access to this repository. 1 team.

Manage access



Invite teams or people to lab2-cooperative-multitasking-using-state-machine-john_paul

paul

advanced-programming/john_paul
Already has access to this repository

Paul Sadauskas
paul • Invite outside collaborator

Paul
paulhumphries4-ccx • Invite outside collaborator

Part 2: Cooperative Multitasking using FSM

In this section you will get familiarised with a design method called state machine and with the concept of multitasking. Friendly cooperative multitasking allows you to split a program into many tasks. If well designed, it will improve performance because each task will get its fair share of time and both tasks will seemingly run in parallel.

The work must be split the master into two branches.

One team member works on the `serialTxTask` task while the other member works on both `encoderTask` and `idleTask` tasks.

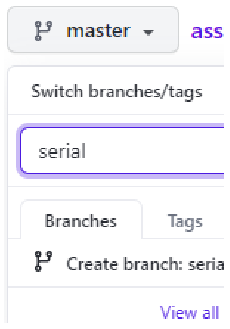
Team member for `encoderTask` must draw a state machine on Visio (or other CAD tool) before starting programming.

With your partner, clone the project.

Clean your cloned code along with your team member. Make sure the project compiles before uploading it.

Commit your clone code and push it to the **master** branch**.**

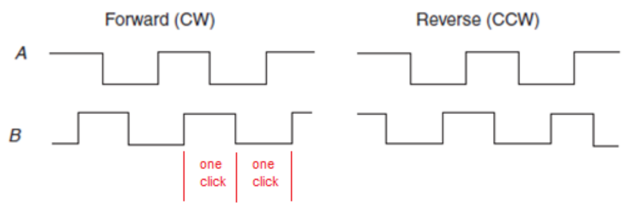
To split into branches: on GitHub web page, each member branches (e.g. branch `encoderTask` and branch `serialTxTask`).



Branch 1: encoderTask ()

Requirement:

1. Increases / decreases a counter by one unit for every CW / CCW click detent.
2. Also, a LED must toggle for every click detent. See timing diagram below.
3. Whenever S3 is pressed, the counter resets to zero.



- In simulator mode:

  Use the stimulus pin/register actions tab. Set it up so two quadrature square wave are generated at 100Hz (take into consideration that the simulator is 10 times slower than normal).

  Once debugged, dump the counter value to the serial console.

Branch 1: idleTask()

Requirement:

A blinking LED indicates how busy the system is.

Notes:

Use UART2 to simulate the LEDs.

Branch 2: serialTxTask()

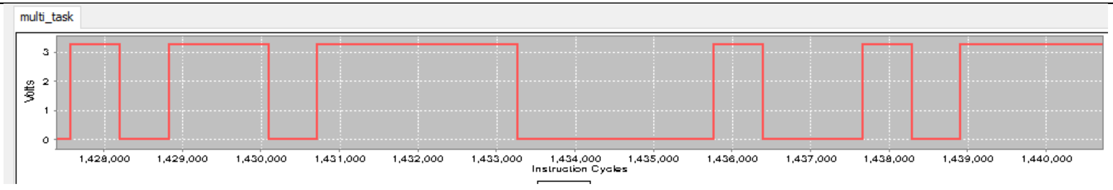
Requirement:

Must generate an 8-bit asynchronous serial communication: no parity and 1 stop bit. See appendix template code.

The bulk of the debugging is done in simulator mode:

Use the simulator logic analyzer.
Set the baud rate at 200000 bps.
Do not run any other task while debugging.
Make the stop bit delay longer (at least 4 times longer).

The following logic analyzer screen shot is the ASCII “Hi” (48 and 69). It clearly shows the longer stop bit delay:



Since the logic analyzer does not work on v5.35, use the breakpoint window, the stopwatch window and the variable window to debug. Set the BAUD rate to 4000 bps.

Once the code has been debugged properly in simulator mode, it is time to test in real time mode – on the target. Set up the system as follows:

Use the oscilloscope.
The baud rate must be set at 500 bps.
Connect to Tera Term terminal emulator.

Public interface:

The communication between the two tasks must be limited to setter-getter functions
e.g., `getCount()`

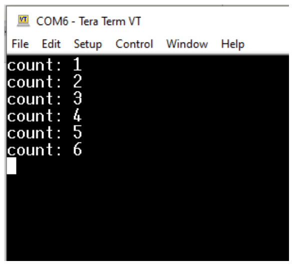
Each member debugs his/her code in simulator mode first. Once fully debugged, it must be tested on the target.

You can disable snippets of code by using macros:

```
#ifdef SIMULATION
   …
#else
…
#endif
```

After the tests, merge the two branches to the master. Together debug on the target.

The final product must display the current count:



```
count: 1
count: 2
count: 3
count: 4
count: 5
count: 6
```

The following rules must be respected:

- Always clone a branch (not the master) because this is hard to switch a branch on MPLABX.
- Commit only the files that were modified/added otherwise it won't be possible to merge/rebase (use CTL key to choose the files to commit). At this point, NEVER COMMIT THE WHOLE PROJECT.
- Commit and push to your branch only when you achieve a milestone.
- Always work on your own branch (not on the master).
- Merge to master only when the code is fully working or when another member requests your code.
- Talk together before merging to master.

Test your code:

Do the two tasks react instantly?
Explain

When the two tasks are done initializing and are up and running, it seems as if the two tasks are running in parallel and they seem to react instantly. There are two tasks that are running, the Serial and Encoder task. The Encoder task is counting a variable called "cnt" and returns the value of "cnt" to the Serial task so that it can be printed serially in a terminal emulator. The Encoder task also prints the value of "cnt" to the LCD and toggles a pair of LEDs on the Explorer 16/32 board at the same time that the Serial task prints the value of "cnt". All of this seems to happen instantly, and they all seem to be running in parallel, but really, they are not. Instead, the two tasks use a very small amount of CPU time to execute and are released once they have completed executing so that the other task can run. This gives the effect as if the two tasks were running in parallel and as if they were reacting instantly.

When you quickly rotate the encoder, does it skip values?
Explain

When the digital potentiometer (encoder) quickly rotates, there is a noticeable negative side-effect in the Serial task output (serial output in terminal emulator). The LCD on the Explorer 16/32 board is able to keep up with the quick rotation, since it accurately displays the full range of the "cnt" variable when the rotation happens, but the Serial task output cannot. Instead, what happens is the Serial task output skips values when a quick rotation occurs on the digital potentiometer. The full range of the "cnt" variable cannot be accurately shown on the Serial task output. This is because, the Serial task is too slow to process all this new data from the Encoder task. To overcome this, a queue will be implemented so that the Serial task can take the appropriate amount of time to display the full range of the "cnt" variable. Even when the queue will be implemented, there will still be a good real-time responsiveness from the system.

Measures

You must provide screenshots of the following measures using the oscilloscope:

1. Encoder channels. It must contain a full cycle of each channel.

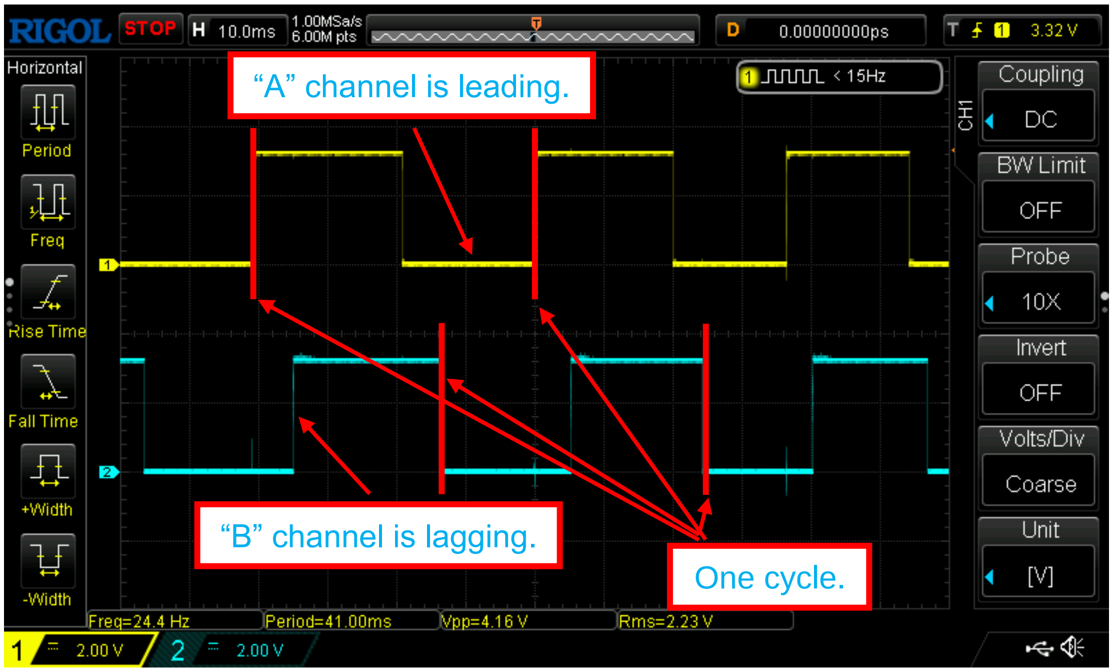


Figure 1. Encoder channels when encoder is running in CW mode.

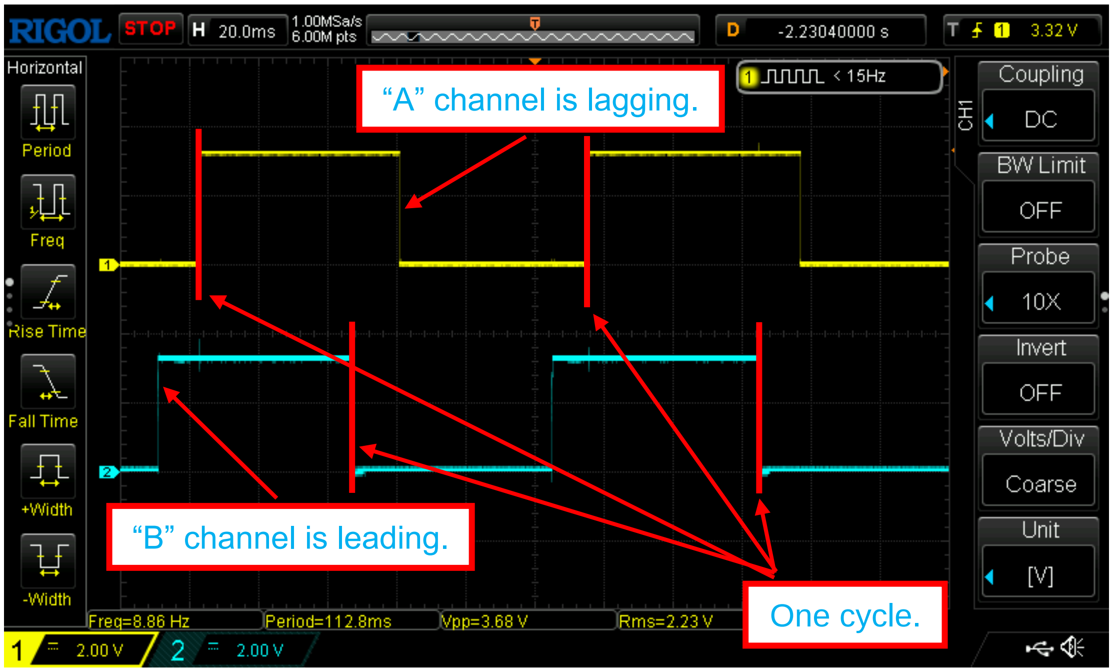


Figure 2. Encoder channels when encoder is running in CCW mode.

2. Serial frame containing at least two characters. The characters must be identified.

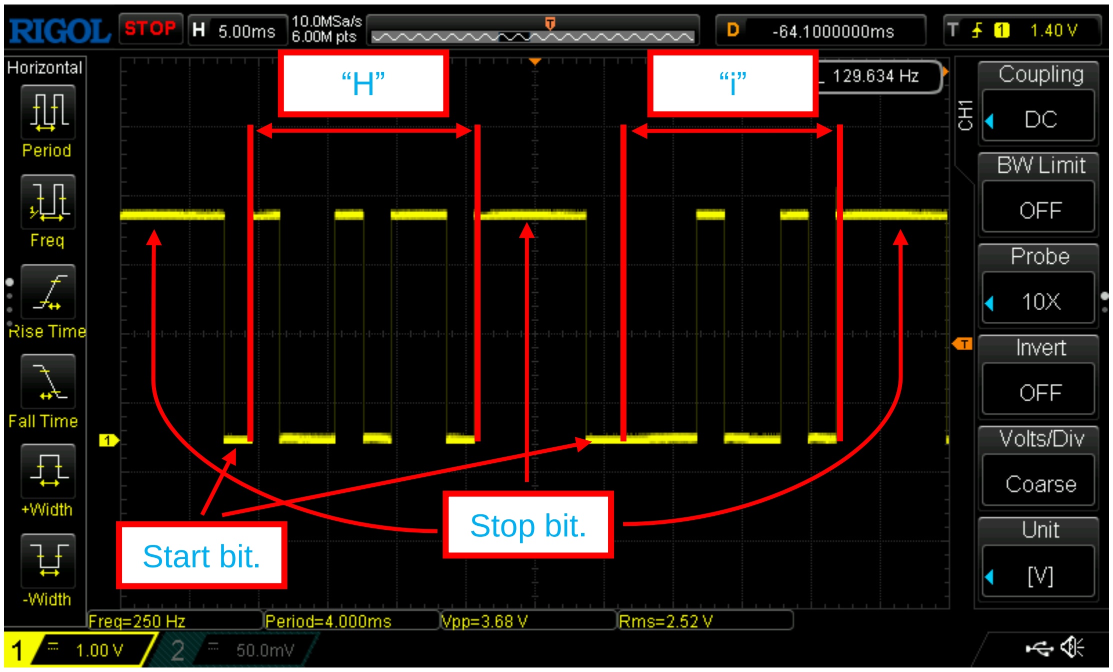


Figure 3. RS232 transmission for the ASCII word "Hi" shown above.

Part 3: Using a queue

In this section you will improve the transmission of data by using a queue.

Working in team, implement a queue to transmit data.

Disable snippets of code by using macros:

```
#ifdef WITH_QUEUE
     …
#else
     …
#endif
```

Test the idle task by adding a 100mS blocking delay whenever a new string is dump to the serial output.

Comment, commit and push the project to GitHub master branch. You must provide a prologue header with a version number (commit only the files modified or added).

You must give a demo for approval.

Check list :

- ☐ Indented code.
- ☐ Commented code – function and file prologs.
- ☐ Versioning.

After lab questions:

1. Explain what you learned in this lab. What went wrong and what did you learn from your mistakes.

   Aside from encountering small syntax errors, some of the main takeaways can be summarized as shown below:
   - Understood how to implement a typical RS232 serial transmission in C using state machines.
   - When problems occurred (logical errors) with the implemented RS232 serial transmission in C, I learned how to use MPLAB's built-in debugging tools and an Oscilloscope to solve these issues.
   - Understood how to code a multitasking system in C using state machines and cooperative multitasking construct.
   - When problems occurred (logical errors) with this multitasking system, I learned how to use MPLAB's built-in debugging tools to solve these issues.
   - Learned how to use public setter-getters in this multitasking system to share data.
   - Learned how to optimize code by removing redundant code and implement "ifdef" statements with appropriate macros (for enable/disable certain parts of code).
   - Learned the advantages of implementing multitasking systems in C using the methods of state machines and cooperative multitasking constructs (for future coding projects involving similar systems).

2- Explain the advantages of state machine and the cooperative multitasking.

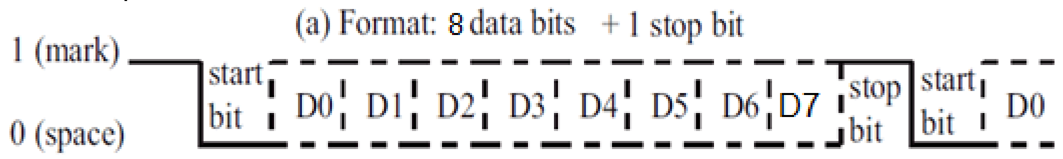
   Some advantages of state machine and cooperative multitasking are as follows:
   - State machine reduces code complexity (no more spaghetti mess in C!).
   - State machines are easy to implement and debug since there are only a finite number of states that need to be implemented in C.
   - Cooperative multitasking is also very easy to implement and is very good in Embedded-like systems where real-time responsiveness needs to be kept (especially for multitasking operations).
   - It is easy to share data in cooperative multitasking systems in which public setter-getter functions are used (with some exceptions: for example, ISRs don't use these functions).

3- Explain the advantages of using a queue specifically for this application?

The main advantage of using a queue specifically for this application is that the system still maintains a good real-time response. This is because the queue is performing a non-blocking operation on the send and receive side of the queue. Another advantage would be that the queue would solve some of the issues we saw when it was not implemented (when the Serial task output would skip the values of the "cnt" variable). The receive end would take whatever time it needs to read the newly received data from the queue and the send side would do just the same (at a much faster rate!).

Appendix

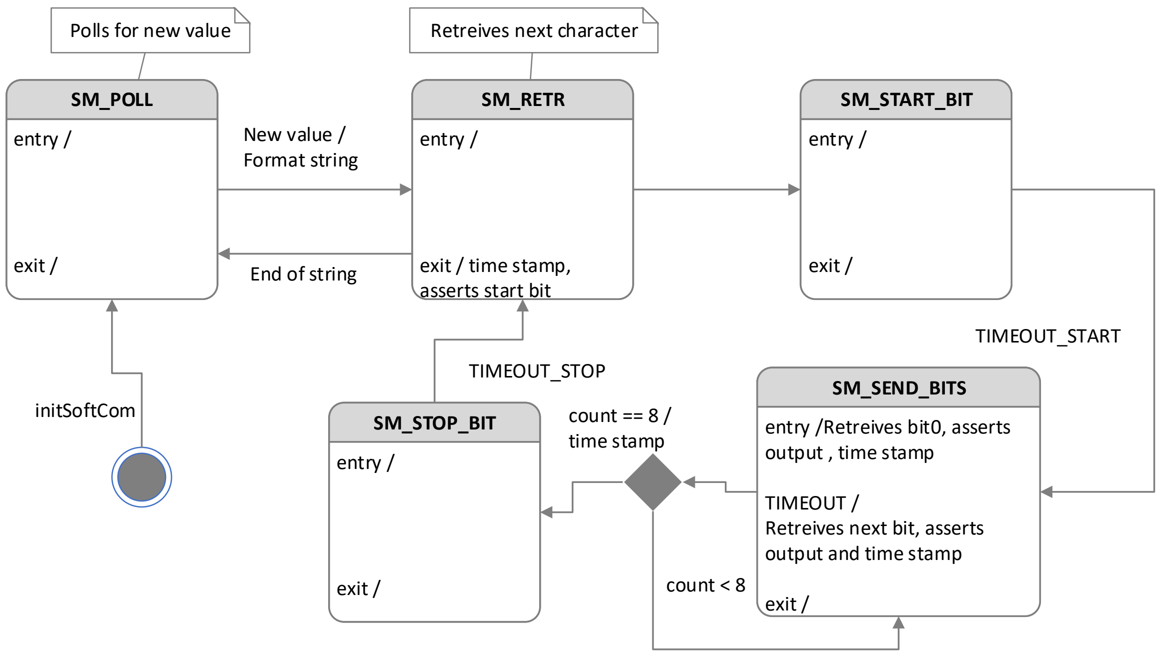
The following is a frame for 8-bit asynchronous serial communication with no parity and 1 stop bit.



Using the UML state machine diagram and the incomplete state machine template, you must implement the frame above.

Restrictions:

You must use the UML state machine diagram and state machine template.
Also, case `SM_SEND_BITS` must use a while loop construct.
You must use macros.



```
/* Initializes resources */
void initSoftCom(void){

}

/* Public interface to communicate with this task */
int getCnt(void){
     return count;
}


void softComTask(void) {
     static int32_t lastTick;
     static enum {SM_POLL, SM_RETR ...} state= SM_POLL;

     switch(state){
        ...

     } // end of case
}// end of softComTask
```