Lab Sheet 6: RTOS Based Systems

1 Introduction and Overview

1.1 Aims

The aim of this lab is to introduce the RTOS. After this lab, you should

- Understand how threads are created and run.
- Understand how threads interact using event flags and message queues

The project includes code for serial communication with the laptop / PC. You **are only** expected to be able to use the API provided. You **are not** expected to understand the code that implements the API[1].

Resources available:

- The RTOS and communication have been introduced in lectures in weeks
- A concept summary describes the structure of RTOS-based systems.
- A 'practical video' on QMPlus on the use of the terminal emulator.

Assessment. If you wish, you can present work on this lab exercise for assessment. Complete any of the following steps (see QM+ for full details and deadlines)

- **Answer the questions on the QMPlus quiz.** *You are recommended to look at the questions about the given code as you go along as answering them will help you complete the rest of the lab.*
- **Demonstration in the lab.** Ask one of the demonstrators to see your code working during the scheduled lab times.
- **Code assessment.** Complete the QM+ assignment to indicate that your code is to be assessed, provided you have demonstrated it. Do not attach any documents. The only way to submit code is to update your repository on GitHub. You can push code as often as you wish, whether or not you would like it assessed.

1.2 Overview of Activities

Perform the following activities:

1. Activity 1: Download, read and run the project using a terminal emulator
2. Activity 2: Modify the project to be increase and decrease the flash rate
3. Activity 3 [Advanced]: Control the timing more accurately when the rate changes

1.3 API Notes

There are notes on the serial communication API in Appendix A.

[1] You are welcome to look at it. We will cover serial communication in the lectures towards the end of term.

2 Activity 1: Download, Read, Connect and Run the Sample Project

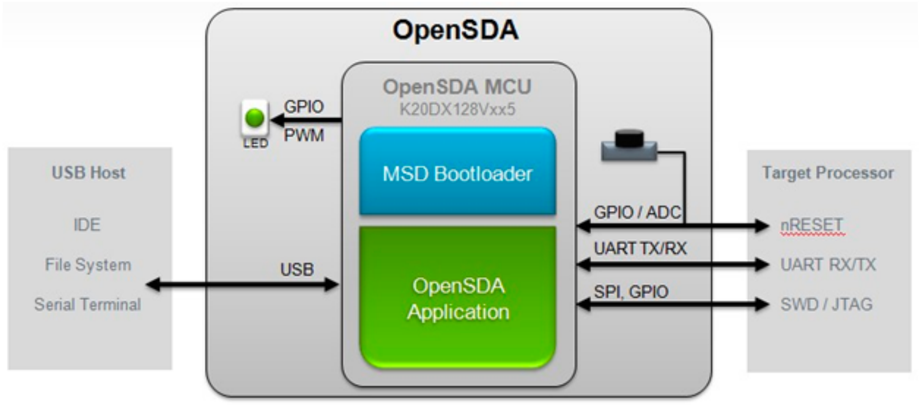
2.1 Download the Sample Project

Download the sample project from the GitHub repository (see URL on the QMPlus page). A terminal emulator is needed to run it (see next section). The behaviour is:

- The green LED can be turned on or off using command entered on the terminal emulator. The command strings are "on", "off" and "reset". Other strings are ignored with an error message.
- When the LED is off, the only acceptable command is "on"; similarly when it is on, "off" is expected.
- If a wrong command is entered, the system enters an error state: the red LED flashes.
- To exit from the error state the "reset" command must be entered.
- The system is initialised in the error state, so the red LED flashes.

2.2 Connect Using a Terminal Emulator

The KL25Z debug adaptor provides a virtual serial connection. A serial communication peripheral on the KL25Z (a UART) is used. This is connected to the OpenSDA debug system which routes the serial data over the USB. This is also a useful debug capability.



On the PC, we need a serial terminal emulator that can display the data. There are many available but coolterm is recommended. It is available from here: https://freeware.the-meiers.org/ It is provided as a no installation zip file so should be possible to run on the lab machines.

The lab machines may have the 'Putty' terminal emulator installed: https://www.chiark.greenend.org.uk/~sgtatham/putty/

Summary Settings

- 8 bits
- No parity
- 115200 baud
- 1 stop bit
- Line mode (send a LF – and may be a CR when return / enter pressed)
- Local echo.

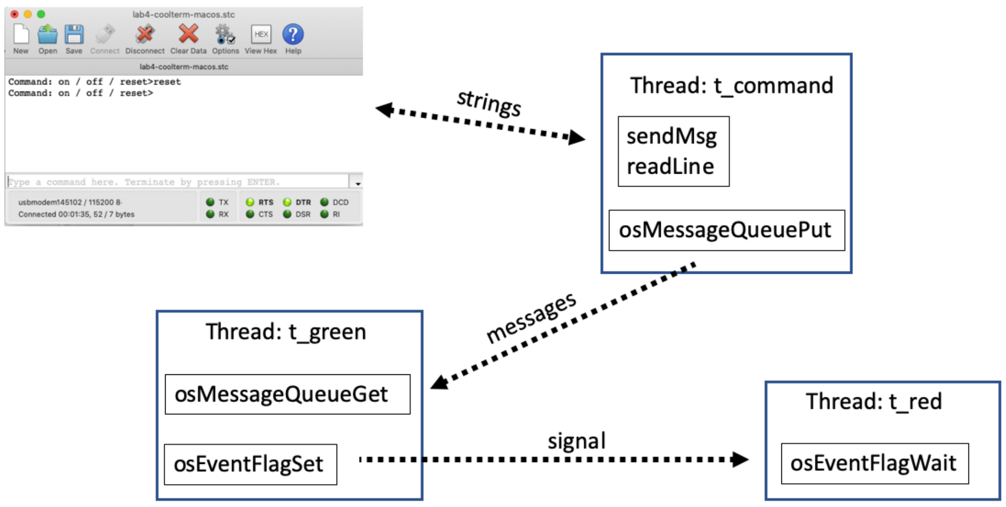
See Appendix B for more details.

2.3 Structure of the Given Program

The program has three threads:

1. The command thread: this communicates with the PC/laptop, sending and receiving text strings
2. The green thread: this controls the green LED. It receives messages from the command thread, turning the LED on and off.
3. The red thread: this controls the red LED. When the green thread signals that an error has occurred, the red LED flashes.

Find all the parts and understand how they relate. The diagram below depicts the structure.



```
Command: on / off / reset>reset
Command: on / off / reset>
```

This structure changes in activity 2.

3 Activity 2: Modify the Project for Varying a Red / Green Flash Rate

3.1 Requirements

Modify the project to achieve the following behaviour:

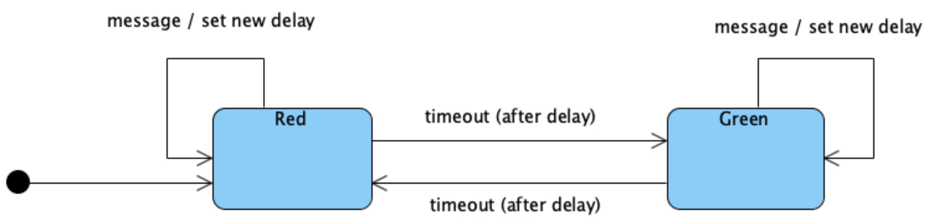
1. The red and green LED alternate, with exactly one LED lit at any moment. The two LEDs have equal on times.
2. The on time can be varied. Eight different on times are available: 0.5 s, 1 s, 1.5 s, 2 s, 2.5 s, 3s, 3.5 s and 4 s.
3. Two commands entered in the serial terminal are used to change the time. The command 'faster' decreases the on time by one step and 'slower' increases the on time by one step. The command should always produce a change: when 'faster' is entered on the shortest time (0.5 s) the list should wrap around so that the new time is 4 s.
   a. Faster: 4s → 3.5s → ... → 0.5s → 4 s → 3.5 s → ...
   b. Slower: 0.5 s → 1s → 1.5s → ... → 4s → 0.5 s

Your code must comply with the guidelines for RTOS-based systems (see web page). In particular:

- Variables cannot be shared between threads; communication must be achieved using Event Flags or message queue.
- You should not use polling to note that a change has occurred.

3.2 Design Requirements

The following outline **state diagram** shows the required design. In each state, wait for a message with a new delay time. Flashing occurs with a timeout; the delay from the message is used to set the timeout.



Code to interpret this state diagram should be run in one of the treads. It is permitted, if you wish, to use event flags (signals) instead of the message.

3.3 Additional Design Hints

The following design hints should make your work easier, but they do not add to the requirements.

Underspecified requirements. The requirements do not cover everything. For example, what is the time period when the system starts? You can make any choice you like.

Two threads. You need at least two threads. Why? The thread blocks waiting to read a line from the terminal, so cannot also flash the LEDs. However, two threads are probably sufficient and more adds complexity. One thread should read lines from the terminal; the other should control the LEDs.

Event flag or message. The two threads should communicate using either a message queue or signals. Do not use both!

Array of times. The times should be held in an array: the content can be values you need in your program. Note that since this data is constant, it can be global. The array could be accessed from either thread; for example, if you decide on using a message then the message can contain the delay.

4 Activity 3: Control the Timing More Accurately when the Rate Changes

4.1 Requirements

A problem with the requirements outlined in the previous section is what happens when the on time is changed. Consider the following scenarios:

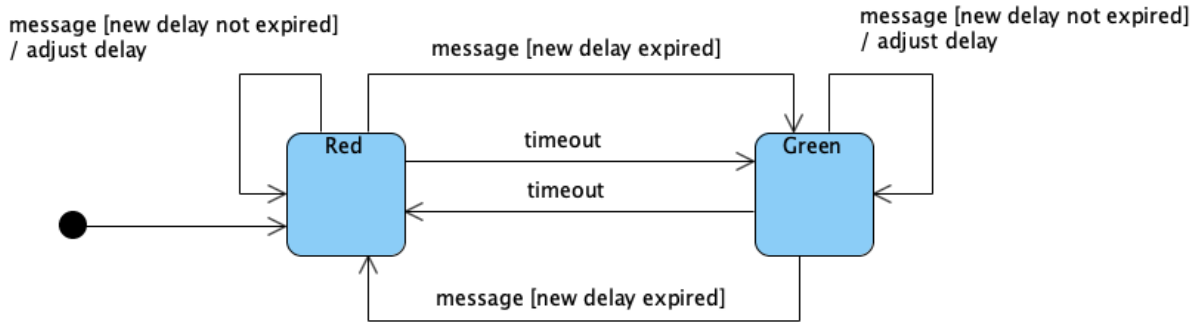
1. Decreasing on-time: suppose that the green LED is on, with an on time for 4 s. At the moment that the time is decreased, 3.9 s has passed. The time is now set to 3.5 s, so that the total time is 3.9 + 3.5 = 7.4 s. So immediately after the decrease, there is an extra-long on-time.
2. Increasing on-time: again, suppose that the green LED has been on for 1.5 s (out of 3.5 s) when the on-time is increased. Just restarting the delay with the new time (4 s) again gives an extra-long time of 5.5 s to the next transition (before the system settles to using 4 s on-time there after).

The additional requirement is to calculate how much of the new on-time has passed:

- If the new on-time has already expired when the change occurs, then immediately change to the new state. This would apply in scenario 1 above: the 3.9 s that have passed is greater than the new on-time of 3.5 s, so the LED that is lit changes immediately.
- The new on-time has not yet been completed. In this case, the additional time should be calculated and this delay used for the first transition for the new on-time. This would apply in scenario 2 above: rather than restarting with a timeout of 4s, using 4s – 1.5s = 2.5 s (for the first delay only) immediately gives the new on-time.

4.2 Design Requirements

The following state diagram shows how to adapt the state model for Activity 2 above.


message [new delay not expired]
/ adjust delay
message [new delay expired]
message [new delay not expired]
/ adjust delay
Red
timeout
Green
timeout
message [new delay expired]

4.3 Additional Design Hints

It is possible to read the tick counter using the call `osKernelGetTickCount`. This reads a counter that counts the number of ticks since the MCU was reset. See https://www.keil.com/pack/doc/cmsis/RTOS2/html/group__CMSIS__RTOS__KernelCtrl.html#gae0fcaff6cecfb4013bb556c87afcd7d2 for full details. The count when the current on-time started can be saved.

5 Appendix A: Serial Communication API

```
bool sendMsg(char *msg, int eol)

     msg: text of message
     eol: whether you want a new line (see code)
     return value: false if message not sent
```

Send a message to the PC / laptop

- Queue a message for transmission over the virtual serial port.
- Non-blocking: small circular queue
- Returns immediately without queuing message if queue full
- Message test not copied from buffer in user thread

```
bool readLine (char *msg, int maxChars)

     msg: buffer for message (size at least maxChars+1)
     maxChars: maximum number of characters from line written
               to buffer
     return value: false if another read already in progress.
```

Read a line from the terminal.

- Single outstanding request
- Blocking: does not return until end of line read
- Reads characters until LF; CR ignored; use with local echo
- Message text written to buffer in user thread

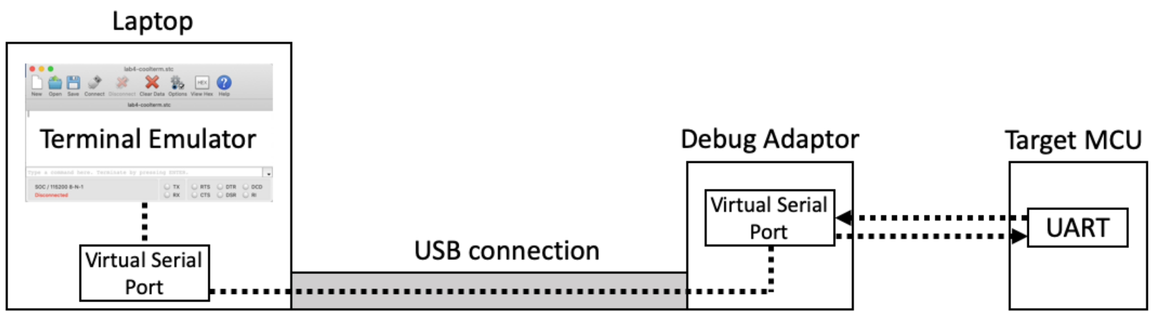
6 Appendix B: using a Terminal Emulator

Background

The original PC had one or more 'COM' ports, with a nine-pin connector. The ports supported the simple point-to-point serial communication standard RS-232. This was originally (in 1960 according to https://en.wikipedia.org/wiki/RS-232) developed for uses such as connecting a terminal to a computer. In the PC, a modem or a mouse might have been connected to the COM port.

Our PCs no longer have physical COM ports. However, the simplicity of RS-232 makes it an attractive way to interface to equipment such as an embedded system (or a laboratory instrument) where the required data rate is not to high. To solve this, virtual serial ports are used. A physical connection is made using a more modern interface (USB, Bluetooth or Ethernet) allowing the end points to interface using RS-232.

The KL25Z has several UART peripherals which are used for simple serial communication. One of these, UART0, is connected to the OpenSDA debug adaptor (also used to download the program). The debug adaptor passes the serial data over the USB connection to the PC/laptop. This has a virtual serial port, through which the data is connected to a program that emulates a terminal. The whole connection is shown below. More background from https://en.wikipedia.org/wiki/Serial_port



Terminal Emulator Configuration

The recommended terminal emulator is Coolterm. This is free and simple to use. It is available from here: https://freeware.the-meiers.org/ and is provided as a no installation zip file so should be possible to run on the lab machines.

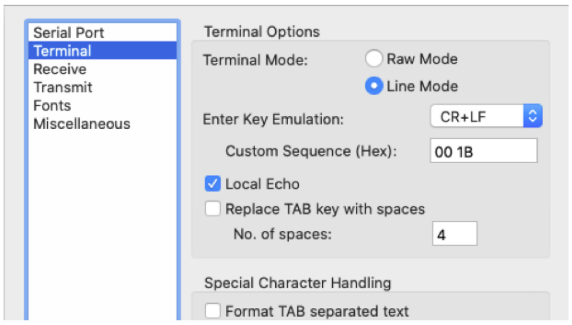
RS-232 has some options and the terminal emulator has to be configured so that the two ends of the conversation agree on the options. A configuration file for Coolterm is provided, but the main points are explained below if you wish to configure another emulator.

Config Option	Value
Port	COM9**
Baud rate	115200
Data bits	8
Parity	None
Stop bits	1
Flow control	None

***The COM port will vary for each computer (depending on what else is connected). Coolterm lists the ones available. Alternatively, see below for how to find the port.*

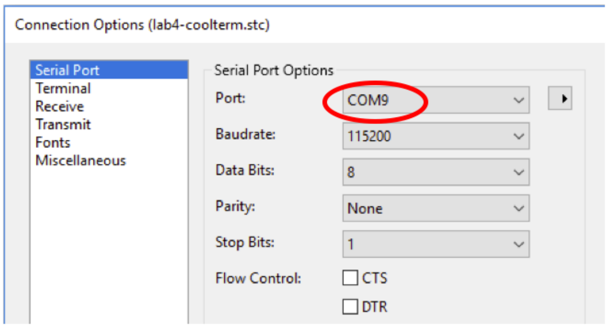
Ports are named by more like files in MacOS and Linux.

The terminal configuration must send a 'line feed' LF when 'enter / return' is pressed. On Coolterm, this is called 'line mode'.


Serial Port
Terminal
Receive
Transmit
Fonts
Miscellaneous

Terminal Options
Terminal Mode: Raw Mode [✓ Line Mode]
Enter Key Emulation: CR+LF
Custom Sequence (Hex): 00 1B
[✓] Local Echo
[ ] Replace TAB key with spaces
No. of spaces: 4

Special Character Handling
[ ] Format TAB separated text


Connection Options (lab4-coolterm.stc)

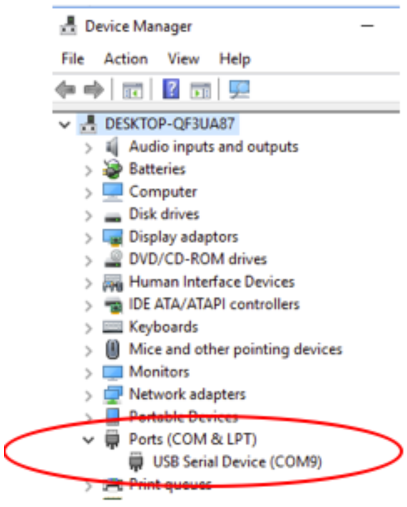
Serial Port
Terminal
Receive
Transmit
Fonts
Miscellaneous

Serial Port Options
Port: COM9
Baudrate: 115200
Data Bits: 8
Parity: None
Stop Bits: 1
Flow Control: [ ] CTS [ ] DTR

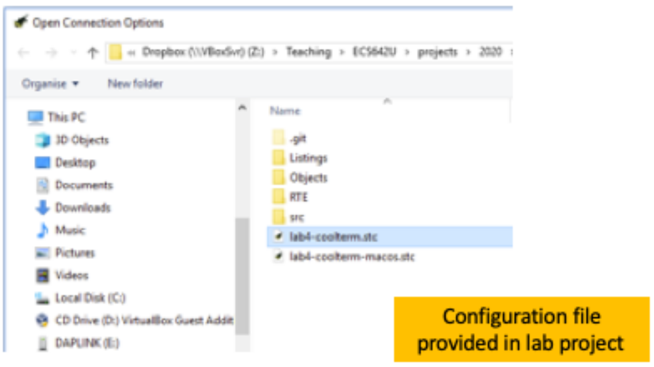
Configuration Options for Terminal and Serial Port

Finding the Port using the Device Manager

The left-hand picture below shows the Windows device manager. The port associated with the USB serial device can be found. A similar capability is available in MacOS.



DESKTOP-QF3UA87
- Audio inputs and outputs
- Batteries
- Computer
- Disk drives
- Display adaptors
- DVD/CD-ROM drives
- Human Interface Devices
- IDE ATA/ATAPI controllers
- Keyboards
- Mice and other pointing devices
- Monitors
- Network adapters
- Portable Devices
- Ports (COM & LPT)
  - USB Serial Device (COM9)
- Print queues


Open Connection Options
Dropbox (\\VBoxSvr) (Z:) › Teaching › ECS642U › projects › 2020
Organise ▾ New folder
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
CD Drive (D:) VirtualBox Guest Addit
DAPLINK (E:)
Name
.git
Listings
Objects
RTE
src
lab4-coolterm.stc
lab4-coolterm-macos.stc
Configuration file provided in lab project