THE UNIVERSITY OF NEWCASTLE AUSTRALIA

School of Electrical Engineering and Computing

Dr Peter Stepien (peter.stepien@newcastle.edu.au)

Assignment 3

Instructions

- Complete all questions.
- Questions are not of equal value.
- To be completed in groups of two.
- Source code due Monday 4 June 2018 11:59pm.
- Late submissions will have 10% deducted per day.
- Demonstrate same week as submission.

Assessment

Code will be assessed using the following criteria:

Component	*Unsatisfactory*	*Improving*	*Satisfactory*	*Excellent*
Specification (60%)	Does not meet specification	Partially meets specification	Meets specification, correct output	Novel extension implemented
Robustness (30%)	No error checking	Partial error checking	Handles provided errors, giving a warning message	Handles additional errors
Code Comments (10%)	No comments	Few comments	Mostly commented	Complete, with clear description of code

Use the template *System Workbench* project provided to write your source code. Information about the templates and how to submit your source code on Blackboard is given on Page 5. The program will be demonstrated to tutors the same week as the due date. The tutors will ask you to explain the operation code and may ask additional questions.

Objectives

The objective is write an application within the constraints of an embedded system targeting the STM32 hardware. The application uses the touch panel and graphics LCD display for user input and output. Data will be read from an analog input and viewed on the LCD display. The data can also be stored on the SD card. The application also makes use of the serial connection.

The application to be implemented is a pulse rate monitor. The analog input is connected to a device that measures blood oxygen saturation levels and the pulse rate can then be determined. The analog signal generated by the analog input is to be plotted on the LCD display and optionally stored to the SD card. Data that has been stored on the SD card can also be retrieved and displayed. A command line interface on the serial port will be used for debugging and to list the contents of the SD card.

This assignment provides practice in writing an embedded application and highlights to issues regarding multiple input and the structure of the software. A Real-Time Operating System (RTOS) will be used for the implementation.

The assignment contains two tasks to be completed. The first covers a *command line interface* and the second covers a *pulse rate monitor*.

Question 1 (40 Marks)

This task makes use of the serial *command line interface* framework implemented previously. The *command line parser* can be used exactly as before, with two additional *commands* to be implemented. One of the additional commands (b) plots the analog input on the LCD display and the other command (c) lists the contents of the SD card. The (a) debug enable command is the same as previously implemented. You can also retain the other commands from the previous assignment if you like. Additional commands that you may want to implement as novel extensions are listed in (d) to (g).

a) Implement a command to turn *debug messages on and off* for Question 2:

- Turning it on will print useful messages while debugging code for Question 2.
- Turning it of will hide debug messages.

This is the same as implemented in Assignment 2. An example of the command operation is:

```
> debug on
Debug messages will be displayed.
> debug off
Debug messages will not be displayed.
>
```

Not shown are the debug messages that would be displayed when it is in the on state. The actual debug messages should be implemented as part of Question 2.

b) Implement the command `analog <time>` to *plot the analog input* for the given period of time `<time>`. An example of the command for a 10 s period of time is:

```
> analog 10
>
```

The software must be able to cope with input that is not valid, such as a time value that is negative or text.

c) Implement the command `ls` to list the contents of the current folder.

An examples of the program output would be:

```
> ls
Pulse     (folder)
Heart     (folder)
data1.csv (203560 bytes)
data2.csv (436344 bytes)
Folders : 2
Files   : 2
>
```

The program can optionally output the number of folders and files found, as shown in the example. This command should use the code written for the previous question.

d) Implement the command `cd <dir>` to change to the folder `<dir>`. This will then become the new current folder. Ensure that an invalid folder name is handled appropriately. Consider also how to move up one level in the folder hierarchy.

An example of the program output would be:

```
> cd Pulse
Current folder : Pulse
>
```

You can optionally print the new current folder as shown in the example. You may also want to extend your program to display the current folder in the prompt, such as in the following example:

```
/> cd Pulse
/Pulse>
```

e) Implement the command `mkdir <dir>` to create a new folder `<dir>`.

An example of the program output would be:

```
> mkdir Heart
New folder : Heart
>
```

You can optionally print the newly created folder as shown in the example.

f) Implement the command `cp <src> <dst>` to copy the file `<src>` to the destination location `<dst>`. If the destination is a folder name, then the file is copied to that folder. If the destination is not a folder name, then a copy of the file is made to the current folder. Consider what happens if the destination file already exists.

Examples of the program output are:

```
> cp file1.csv file2.csv
New file : file2.csv
> cp file1.csv Heart
New file : Heart/Heart.csv
>
```

You can optionally print the newly created file as shown in the example.

g) Implement the command `rm <file>` to delete the file `<file>`. Consider the cases where the file name does not exist or that the file name provided is actually a folder name.

An example of the program output is:

```
> rm file2.csv
Deleted : file2.csv
>
```

You can optionally print the deleted file name as shown in the example.

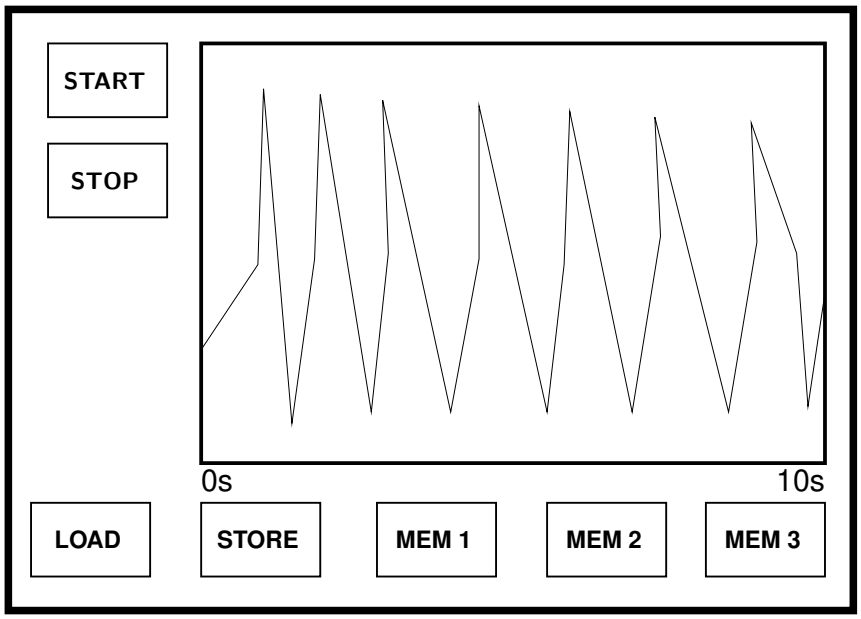
Question 2 (60 Marks)

Implement a pulse rate monitor on the STM32 hardware. Use the LCD screen to display information and use the touch panel for user input. Use the framework implemented for the previous assignment to layout buttons on the screen.

It should be possible to start and stop the pulse rate monitor with the use of two buttons (**START** and **STOP**). This will freeze the display. New data will be displayed at the correct location in time once the display is started again.

The application should allow users to save three sets of recorded pulse rate data to the SD card (**MEM 1**, **MEM 2** and **MEM 3**) and restore the contents to the display. This can be done using two additional buttons, such as (**LOAD** and **STORE**). The method by which loading and storing is implemented is up to you.

An example of the LCD display layout is shown below:



Feel free to change the layout. It is only provided to give you an idea of what the layout may look like. You may also want to look at commercially available products for inspiration.

The command line interface from Question 1 must also run concurrently with the pulse rate monitor. It should be used as a debugging tools, with useful messages being displayed if debugging messages are turned on. The command line interface can also be used to view the files being generated by the pulse rate monitor.

The analog input is configured to sample at a rate of 1 kHz. To save one screen of data to the SD card requires 10 kSamples. Additional features that could be added to the pulse rate monitor include zoom functions, auto scaling and pulse rate calculation.

You are also welcome and encouraged to add any additional features to your design. Have fun coding your application!

Additional Information

The following provides information of writing your source code with *System Workbench* using the project template provided and how to submit your source code and output transcript on Blackboard. An overview of the *Software Development Tools* is also available on Blackboard.

System Workbench Template Project

The template project is available on Blackboard in the folder containing resources for Assignment 3. There is only one template for this assignment, which is for the STM32 hardware:

- `Ass-03-STM32-rXXX.zip`

Unzip the contents into the *System Workbench* workspace and import the project.

The template project is self contained and fully functional, with an example application which uses the resources needed for the assignment, including the use of the various features of the RTOS. Remove the example application code and replace it with your code.

The structure of the code consists of a number of tasks. Also initialised are various RTOS resources that can be used to complete the assignment.

The following source files have been included for you to write code for each question (*this may change)*:

- `Ass-03.h`
- `Ass-03-Task-01.c`
- `Ass-04-Task-02.c`
- `Ass-05-Task-03.c`
- `Ass-05-Lib.c`

The aim of the `Ass-03-Lib.c` file is to place functions that are common to other source files. Note that code from previous assignments can be used. The include file requires updating when additional functions or data structures have been defined which are used in a number of source files.

The baud rate for the UART has been set to 115,200 bps. This will need to be set on PuTTY to correctly communicate with the STM32 hardware. Turn off hardware and software handshaking in PuTTY as it is not configured in the STM32 project.

An SD card will also be required. These have been installed in the STM32 hardware and should remain in the hardware. You can modify the contents of the SD card if you like.

Submitting to Blackboard

The above listed source files form the deliverable for the assignment. A single file needs to be generated using the following from the command prompt in the folder with the source files:

- `type Ass-03*.c Ass-03.h > Ass-03-SID1-SID2.txt`

Replace `SIDn` with the student numbers of the group members. You could also use `cat` instead of `type` under Linux. Note that the submitted file must be clear text.

Analog Input

The analog input has been configured to use DMA transfers from the analog input to memory. This code is available in the template project and is also covered in lectures. *More information will be provided about the analog input.*

FAT File System

A library is provided to access files on the SD card. It is the FatFs *Generic FAT File System Module*. See the template project for example code. Note that the file system has to be mounted first before it can be used. It is also covered in lectures.

If the SD card is removed and inserted while the program is running, the SD card will have to be remounted. There is a switch on the SD card connector that can be used to determine if the SD card is inserted. It is not a requirement to handle this case, but you can experiment if you wish.

CMSIS-RTOS

The CMSIS-RTOS is an operating system abstraction that ARM have written that can be place between the application layer and an operating system. The CMSIS-RTOS will be used on top of FreeRTOS for this assignment.

Information for CMSIS-RTOS can be found on Blackboard in a ZIP file and is also covered in lectures.

References

Additional references:

- CMSIS-RTOS API: Generic RTOS interface for Cortex-M processor-based devices
  Documentation available on Blackboard.
- The FreeRTOS Reference Manual
  http://www.freertos.org/Documentation/FreeRTOS_Reference_Manual_V9.0.0.pdf
- FatFs: Generic FAT File System Module
  http://elm-chan.org/fsw/ff/00index_e.html