School of Science
Computer Science

Loughborough University

21COA202: Embedded Systems Programming

Coursework specification

Iain Phillips

Semester 2 2021-2022

Document history

Version	Date	Comments
V1	2022/02/24	Initial release to students
V2	2022/03/04	Clarifications after forum post. marked in text with (new for v2)

1 Introduction

This coursework forms 100% of the assessment for this module.

There are several parts to the coursework. A main exercise (known as BASIC) and several additional parts to allow you to demonstrate more knowledge (EXTENSIONS). These are all detailed below.

If anything is not clear as you read please read through to the end of the document, it may be clear later. If still not, then post the question into the support forum on LEARN.

2 Data Monitor

The aim of this exercise is to program the Arduino as a Data Monitor and Displayer. It will read data values over the Serial Monitor, process them and then display them on the screen. The Arduino's buttons will allow for interaction with the numbers displayed on the LCD.

First we will introduce the concepts of Data and Channels, then how things should be displayed on the Arduino Screen and the protocol that will be implemented across the Serial Monitor, with some examples.

2.1 Data and Channels

The Data Monitor receives data values over the Serial Interface in a series of channels. These are defined as follows:

- After an initialisation phase, each message is a text string sent as a sequence of characters terminated by an `LF`. These are sent over the serial interface either through the Serial Monitor or from a separate host program.

- Data values will be integers in the range 0-255 presented as text.
- We associate each data value with a channel with the channel id being a letter in the range A-Z. Each channel has a description text.
- Each channel has a maximum and minimum value. These values are provided across the serial monitor and special displays should be used when the received data values exceed the range.

The protocol for communication is detailed later in the document, but first look at what is to be displayed.

2.2 Arduino Display

The Arduino presents data values on its screen and provides an interface via the buttons for the user to examine the most recent values. We describe this interface in this section.

- The display has two lines and shows the value of two channels providing an up-down scrolling facility to see others.
- The display must be of the form:

  ```
  +----------------+
  |^1XXX           |
  |v2YYY           |
  +----------------+
    ^
    \-- note no space between channel letters and value display
  ```

  Note the exact positioning of the characters.
- In this example, `1` and `2` and are replaced by the channel letter and `XXX` and `YYY` by the most recent value. The value must be right justified, if the value is not 3 digits, then move it so the units columns line in the right column. e.g.:

  ```
  +----------------+
  |^A146           |
  |vW  9           |
  +----------------+
  ```
- Program the `UP` and `DOWN` buttons to move up through the channel list in alphabetical order. On reaching the first or last channel remove the `^` or `v`, to indicate there are no more channels to show.
- If the input numbers on any channel fall outside this range then the backlight must change colour: **red**, if there is a value on a channel above the maximum; **green**, if there is one below the minimum; **yellow** if both are true. If all recent data values are in range on every channel then set the backlight to **white**.
  - (New for v2) If a minimum value is received that is larger than the maximum value it must be stored, but not included in the determination of the backlight colour in the paragraph above.
- Pressing and holding the `SELECT` button for longer than one second clears the screen, sets the backlight to purple and displays simply your ID number and nothing else. Releasing it returns the display to normal. During this time your program must continue to read lines from the serial monitor and process them.

2.3 The protocol in more detail

- Synchronisation phase:
  - After your Arduino has booted it must set the backlight to purple and repeatedly send the character `Q` with no following NL or CR character to the Serial interface at a frequency of once per second.
  - After sending, the Arduino must monitor for an incoming character and when an `X` is received synchronisation completes and the main program loop can start up.
  - No newline or carriage return should be sent or expected.
- After synchronisation happens your program must send the string `BASIC` followed by a new line, then set the backlight to white and start the main phase.
- Main phase:
  - The host sends messages to the Arduino over the serial interface. The first character will be one of the following:
  - `C` — this indicates a new channel. It will be followed by the channel letter and a String description. This could be up to 15 characters and extra characters until the LF must be ignored. If the host sends a `C` message an existing channel, the new description must be used.
  - `V` — this indicates a value. It will be followed by a letter, indicating the channel and an integer number: the value. The value will be expressed as text and in the range 0 to 255. Data values received on a channel that has not been created must be ignored.
  - (changed for v2) `X` — specifies the maximum value for a channel. If no `X` line has been received, then assume 255 as default.
  - `N` — specifies the minimum value for a channel. Here assume 0 as default.
  - Both `X` and `N` are followed by the channel letter and value in the same form as the `V` message, with no spaces or other characters. e.g.

    ```
    XA100
    NW100
    ```
  - Any lines not conforming to the protocol must be ignored. However, the text `ERROR:` followed by the ignored string should be written to the Serial Monitor. This will aid debugging.
    - (New for v2) Lines such as `XA` with no value are invalid.
  - In addition, you may send any text beginning with `DEBUG:` to the Serial monitor; it will be ignored by the assessment testing program, but will help your debugging.
  - There must be no whitespace characters in the messages, except in the channel description.
- Unless you implement the EEPROM extension (see below), then on initialisation no channels should be assumed defined and consequently all received data values must be ignored until a suitable `C` message arrives.

2.4 Protocol Examples

Some An example uses of the protocol:

This first example sets up two channels `A` and `B` and gives them the names `Main` and `Secondary`. Data values `100`, `200`, `201`, `202`, `203` and `204` are sent over channel `A` and `5` and `4` over channel `B`.

```
CAMain
CBSecondary
VA100
VA200
VA201
VB5
VA202
VA203
VA204
VB4
```

The following example sets up a single channel `A`, sets maximum to `100`, minimum to `50` and sends data values `90`, `100` and `110`.

```
CAMain
XA100
NA50
VA90
VA100
VA110
```

A python program for running on a host and conforming to the specification will be provided in week 5. You are welcome to modify this for your testing purposes. A version of this program will be used to test your implementations, although the numbers will change.

2.5 Requirements

- You must implement your code using a Finite State Machine (FSM). Include a picture of this machine, listing the states and the state transitions in your documentation. See lab sheet 4.
- Make sure comment your code appropriately. Ideally code should be mostly self commenting through sensible choice of variable and function names and use of macros. However, there will be times when things are less comprehensive, so be sure to highlight theses.
- In your documentation:
  - Write a description of your states, what is being waited for, and the actions take during transition between states.
  - Write 200–500 words of reflection on your code. Include those things that don't work as well as you would like and how you would fix them.

2.6 Assessment

Achieving the above implementation is worth up to 50 marks, if properly documented (20 marks) and coded (30 marks).

3 Extension features

The remaining marks are distributed across the extension features. You may implement as many as you wish. Assume about 40% of each extension's marks will be based on any documentation.

The following extensions allow you to demonstrate more knowledge and in return are work additional marks indicated. If you have implemented any of these then the string BASIC sent after synchronisation must be replaced by a list of the extension names separated by commas and followed by a new line. For example:

```
RECENT,FREERAM,HCI,SCROLL,EEPROM,UDCHARS
```

3.1 UDCHARS

This shows your ability to define your own character forms. [5 marks]

- Define some characters to replace the `v` and `^` with nicer looking arrows.
- In your documentation indicate which lines of the code define the characters.

3.2 FREERAM

This show your ability to read the free SRAM in an Arduino. [5 marks]

- Modify what happens when pressing the `SELECT` button. In addition to the ID number, display the amount of free SRAM.
- In your documentation indicate the parts of the code to display the free SRAM.

3.3 HCI

This shows your ability to select particular values from an array of values. [5 marks]

- When the user presses and releases the `RIGHT` button the display must only display channels where the current value is beyond the maximum. (changed for v2) If none match this criterion, then display nothing.
- When the user presses and releases the `LEFT` button the display must only display channels where the current value is beyond the minimum.
- If in either of the above two states repressing and releasing the same button returns to displaying all the channels.
- If no channels match the criteria, then nothing should be shown. The `v` and `^` arrows should appear based on the subset of channels to display.
- In your documentation show the lines of code and thinking behind the mechanism to display subsets of the list of channels.

3.4 EEPROM

This shows your ability read/write from EEPROM. [5 marks]

- Store the channel letters and descriptions with their maximum and minimum values in the EEPROM so that they can survive power resets.

- Read from the EEPROM on startup and find a mechanism for determining whether the values where written by you or simply left from before.
  - (new for v2) After restarting until a value is received on a particular channel you MUST not include that channel in the maximum and minimum operations.
- In your documentation indicate how you lay out the use of the EEPROM and which lines of code and functions you use to store the information.

3.5 RECENT

This shows your ability to manage larger data sets than single values. [15 marks]

- Store the most recent 64 values and display the average of these as well as the most recent value on the screen. Keep to the format below:

  ```
  +----------------+
  |^1XXX,AVG       |
  |v2YYY,AVG       |
  +----------------+
  ```

  Average is to be rounded to the nearest integer value.
- In your documentation indicate the names and locations of the data structures used to store the recent values.

3.6 NAMES

This shows your ability to store and display other text. [5 marks]

- Display the channel names to the right of the values using the remaining available space.

  ```
  +----------------+
  |^1XXX Main      |
  |v2YYY Secondary |
  +----------------+
  ```

  or if you have implemented RECENT

  ```
  +----------------+
  |^AXXX,AVG Main  |
  |vBYYY,AVG Second|   <--- note name is truncated
  +----------------+
  ```
- In your documentation indicate the data structure you use to store the channel name and how it is printed to the display.

3.7 SCROLL

This shows your ability to implement a more complicated FSM that can take time into account as well as listening to the Serial Monitor and button presses. [10 marks]

- If you have implemented NAMES then when the channel description is too big then add code to scroll it left at 2 characters per second, returning to the start when the full name has been displayed.

- All other functionality should remain implemented as described.
- In your documentation highlight the parts of the state machine required for this particular requirement and the lines of code and functions that carry this implementation.

4 Hints

- SRAM is limited. You want to keep use of SRAM to only what really needs the SRAM. Make use of the FLASH memory for fixed strings, see lab sheet 3.
- Be careful using the String class—this tends to use up RAM.
- On writing to EEPROM remember to use the correct calls to avoid writing when you don't need to.
- In your design of the FSM, consider what events your code will be waiting for. Think about button presses, button releases, something from the serial, timeouts.
- `LF` is the line-feed character. This is sent from Python with a `print('\n')`.{python} or from the Serial monitor by selecting Newline from the drop-down at the bottom of the window.
- You can choose to interact with your program simply through the serial monitor, but it may help you to write some python host code to automate this. See lab sheet 5.

5 Implementation

1. You **must** use (at least one) **Finite State Machines** to implement this specification and these must be described in your final deliverable.
2. You **must** implement the protocols **exactly as described**. If you feel anything is not clear, then please ask. We may use automated testing of your code.

6 Deliverables

This is an individual coursework. Similarity-detection software will be used on both the documentation and the source code as an initial check to see where collusion or copying have taken place. Code may be checked against other submissions this year, previous years and other sources as appropriate.

Your submission needs to include the following:

- Source code of your implementation: a single `.ino` file submitted to `LEARN`.
- A written report—submitted as a PDF to Gradescope. Template files are available; use of these is **compulsory** and details of page tagging will be on `LEARN`.

6.1 Source code

Submit this as the actual `.ino` file, not copy-pasted into the report.

6.2 Written report

A template for the report is available on the Learn page in `markdown`, Word and `LaTeX`—your choice. But remember the final submission must be as a PDF. You must use the provided template—see LEARN.

7 Further information

- You may only use the following libraries (as identified by their header files). You might not need to use all of these.
  - `Wire.h`
  - `Adafruit_RGBLCDShield.h`
  - `utility/Adafruit_MCP23017.h`
  - `EEPROM.h`
  - `avr/eeprom.h`
  - `TimerOne.h`
- Your code has to consist of a single `.ino` file, which needs to compile using the Arduino IDE. If the code fails to compile in this environment, then you risk failing the module.
- Your code has to work with the Arduino Uno and LCD shield that we issued to you. In particular, if you develop your code on a different type of Arduino or with a different type of shield, you risk failing this module.
- We will use various types of software for code similarity detection to compare each submission to all others, to submissions from previous years, and to code from other sources. Please be reminded that this is a individual assessment, which means that group work is not permitted.

8 Grading scheme

Most marks will fall into a scale between 40 (scraped pass) and 80 (impressive). These are explained in the grading descriptors below.

Significantly fewer marks will be awarded if you do not follow the compulsory instructions, for example, not using the template, using other libraries that those specified above.

As a rough guide the following sections indicate what the expected mark ranges.

8.1 Code

Code covers the code itself and includes readability, sensible function names, use of functions and language features as well as whether the functionality is there.

- A (80+) - The project significantly exceeds the minimal specification in the amount of features and in the technical challenge behind these features. All features have been realised at an excellent level. The project implementation carried out at a high-quality level.
- B (70) - The project significantly exceeds the minimal specification in the amount of features or in the technical challenge behind these features. The project implementation has been executed well with some aspects at an excellent level.
- C (60) — The minimal feature set and some non-trivial extra features have been realised at an adequate level. The project implementation and testing is at an adequate level. The interface is easy to understand after some explanation.
- D (50) — The minimal feature set, and some minor extra features have been realised a mostly adequate level. Enough has been done the overall achievement is mediocre.

- E (40) — The minimal feature set has been realised at a somewhat adequate level. The overall implementation and testing of the project is rudimentary.
- F (<40) - Not even the minimum feature set working and nothing to make up for this. The higher end of this range would be used to reward partial attempts with some merit.

8.2 Documentation

Documentation will cover pictures and text in the report and comments in the code.

- A (80+) The documentation is beautiful and describes all aspects of the implementation well.
- B(70) The documentation is very good, including all details, but lacks in presentation and understanding.
- C(60) The documentation includes most of what is necessary, but misses some vital meaning.
- D(50) The documentation describes some aspects well, but these aspects only form some of what is required.
- E(40) The documentation is barely adequate.
- F(<40) Something worse. No documentation will attract zero.