ENGD2103 EMBEDDED SYSTEMS FUNDAMENTALS
Laboratory guidance notes
Week 2: Formative Assignment – Part 2 – Button Counter

Author: M A Oliver, J A Gow, T Allidina *v1.0*

Table of Contents

Version record

Version	Notes	Date	By
1.0	Initial version	14/10/22	MAO

PLEASE READ THIS CAREFULLY

This is the second of a series of guidance documents to get you started with your Embedded Systems Fundamentals projects. They are aligned with the key aspects of the formative coursework specification, and thus this document should be read in conjunction with the coursework specification document. These documents will start off with some fairly close guidance - however as you develop your skills the notes will provide less and less detailed guidance and become more of an overview.

1 Aims and objectives

During the formative assignment, you will be writing three short pieces of code. This document outlines the second part: a button counter.

Prerequisite: you should have completed the assembly of the circuit from Week 1 and the Traffic Lights code from Week 2 before commencing this task.

2 Overview

The purpose of this task is to give you experience of:-

- Interfacing a 7-segment LED display
- Debouncing a switch
- I/O port expansion (i.e. how to deal with situations where there are insufficient port-pins)
- 'Bit-banging' a simple protocol.

3 Task Requirements

In this task, you will be required to implement a counter that counts the number of times a push-button switch is pressed. The least-significant nibble (4-bits) of the count will be displayed on a 7-segment digital display.

The task is to be based around the Arduino Nano that you were provided with. The 7-segment digital display is a Common Cathode device. This will be connected to a 74HC595 shift register. Three lines (CLOCK, DATA and LATCH) of the shift register are connected to three digital pins on the Arduino (as per the schematic from Week 1).

A push-button switch will need to be added to the circuit; this will connect between one of the digital lines of the Arduino and GND. This push-button will require a pull-up resistor. However, it is recommended that you use an internal pull-up.

For this exercise, you will be required to write the Arduino code for driving the counter and debouncing the switch.

For the Arduino code, you will be required to:-

1. Create a table (array) of byte values for displaying the characters '0' to '9' and 'A' to 'F' on the 7-segment digital display.

2. Create a global or static variable called `counter` and initialize it to zero.

3. Write a function such as:-
   ```
   void writeToShiftRegister(byte value)
   ```
   This function will write the byte parameter `value` most significant bit first into the shift register.

4. Write a function:-
   ```
   void waitForKeypress()
   ```
   This function will wait until a debounced keypress of 500ms is detected.

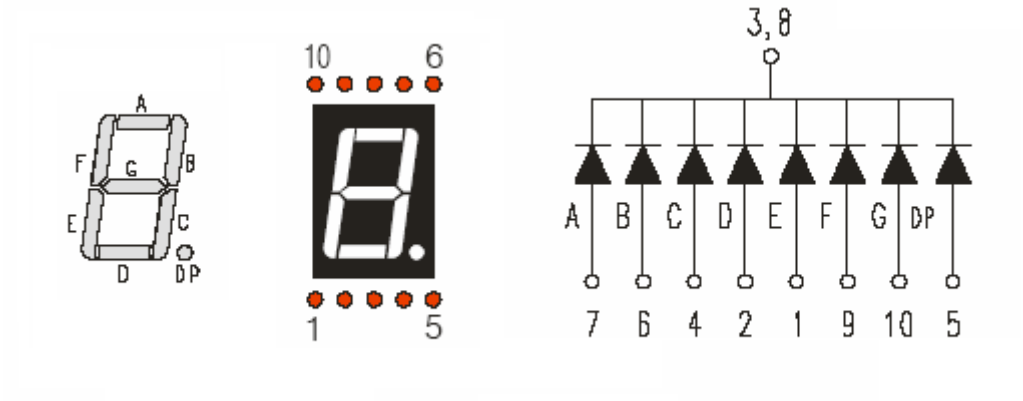
5. The main `loop()` will perform the following operations:-
   - Use the function `writeToShiftRegister()` to display the least-significant nibble of `counter`.
   - Wait for a debounced keypress
   - Increment the value of `counter`

4 The 7-Segment Digital Display

The 7-segment digital display used in this exercise essentially contains 8 LEDs in a single package. Each LED segment requires its own series resistor as is shown in the schematic from Week 1.

The illustration below shows, from left to right:-

- The segment arrangement of the 7-segment display
- The pin numbering of the 7-segment display.
- The internal wiring of the 7-segment display. Note the cathodes are connected together *(i.e. common cathode)*



The following image shows how the characters '0' to '9' and 'A' to 'F' are required to be displayed on the 7-segment digital display.

5 The 74HC595 Shift-register as a Port Expander

Port Expanders – Why?

Suppose you are part of an engineering team. In the engineering lab, the team has developed a fantastic product which is based around a microcontroller. In another part of the building, your main customer meets with your Management and Sales & Marketing teams and explains that they absolutely need an additional 8 status LEDs on the product. Without consulting the Engineering team, Management blindly agrees to this request. The Engineering team is then tasked with implementing this modification. It sounds like a straightforward task until you realize that there are only three unused pins available on the microcontroller. What do you do?

The first option would be to use a microcontroller with more I/O pins. This approach has multiple drawbacks. Firstly, your company might have negotiated an ultra-low price for the existing microcontroller. Secondly, a change in microcontroller will require a significant, if not a complete PCB redesign.

A better option would be to use some form of port expander. This approach would, of course, require a small amount of PCB redesign, but not as drastic as the first approach.

One common form of port expansion comes in the form of I^2C port expanders. These can be driven from two lines from a microcontroller: SDA (data) and SCL (clock). I^2C port expanders can be configured to provide both additional input and output pins. They can also be cascaded to provide more I/O lines if required.

Another approach of port expansion that will be applied in this exercise is the use of the shift-register. The shift-register approach is a relatively inexpensive method for providing additional output lines. Shift-registers can be driven from a microcontroller using three output lines: LATCH, CLOCK and DATA.

Bit-banging – why?

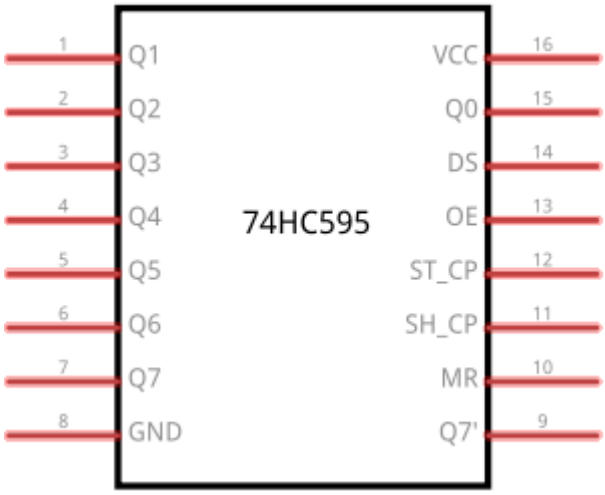
Suppose you want to interface a digital part to your microcontroller. There might not be a peripheral on the microcontroller capable of driving that part directly. Additionally, there may not be any firmware libraries available to drive that part. So what do you do?

The first thing is to find the datasheet for the digital part and look at the timing diagrams. From the timing diagrams, firmware will need to be written for the microcontroller that drives the digital I/O lines in sequence according to the timing diagrams. This approach of driving I/O lines according to a timing sequence is affectionately known as “bit-banging” or “bit-wiggling”.

In this exercise, you will need to drive the LATCH, CLOCK and DATA lines in the correct sequence to write data to the shift-register.

Mastering this technique should give you the confidence to 'bit-bang' other devices you might encounter in the future whilst in industry.

Using the Shift-Register As A Port Expander
The pin-out of the shift-register and the pin definitions are shown below:



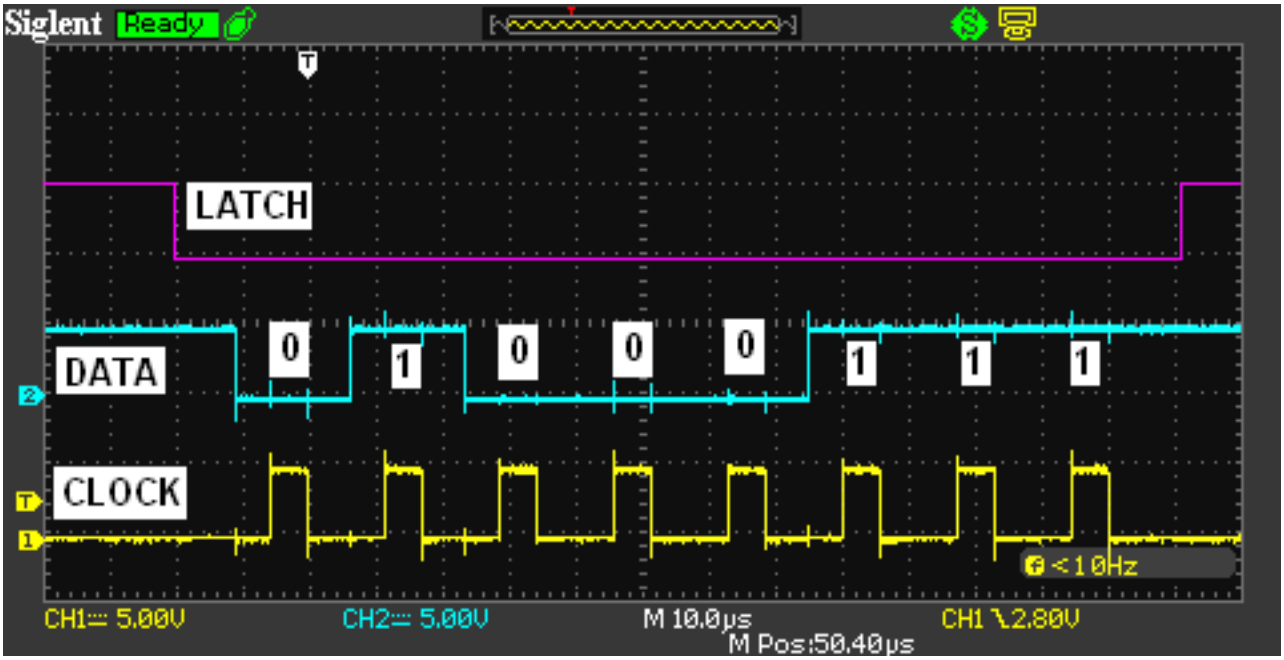
Pin	Label	Notes
1	Q1	Output Q1
2	Q2	Output Q2
3	Q3	Output Q3
4	Q4	Output Q4
5	Q5	Output Q5
6	Q6	Output Q6
7	Q7	Output Q7 (Most-significant bit)
8	GND	Connect to GND
9	/Q7	Inverted output Q7 (not used in this exercise)
10	/MR	Master Reset (Connect to VCC)
11	SH_CP	CLOCK pin
12	ST_CP	LATCH pin
13	/OE	Output Enable (Connect to GND)
14	DS	DATA pin
15	Q0	Output Q0 (Least-significant bit)
16	VCC	Connect to +5V

Driving the Shift-Register

The following approach can write an 8-bit quantity held in a variable called `value` to the shift-register:-

- Declare an 8-bit 'mask' variable, and set the most significant bit high.
  i.e. set `mask` to 0x80 (or 128 in decimal, or B10000000 in binary)
- For each of the 8 bits in `value`
  - If `value` bitwise-ANDed with `mask` is non-zero then
    - Pull the DATA line HIGH
  - Otherwise
    - Pull the DATA line LOW
  - Right-shift `mask` 1 place
  - Toggle the CLOCK line HIGH-then-LOW
- Toggle the LATCH line HIGH-then-LOW

The following oscilloscope trace shows how the `value` 71 (i.e. 0x47 in hexadecimal, or 01000111 in binary) is clocked MSB into the shift-register. It is worthwhile spending some time studying this in conjunction with the method described above. In this example, a slightly different yet valid approach is used to drive the LATCH line. Simply toggling the LATCH line once all 8-bits are clocked-in is a neater approach.



6 Switch Input

This exercise uses a switch that is connected between one of the digital I/O pins on the Arduino and GND.

In the Arduino environment, the following example sets-up Pin A0 as a digital input with no internal pull-up resistor (i.e. a pull-up resistor would need to be added to the circuit)

```
#define SWITCH          A0

void setup()
{
    pinMode(SWITCH, INPUT);
}
```

Alternatively, the following example sets-up Pin A0 as a digital input with internal pull-up resistor (i.e. no pull-up resistor is required in this case). This would be the best approach for this exercise:-

```
#define SWITCH          A0

void setup()
{
    pinMode(SWITCH, INPUT_PULLUP);
}
```

The following function call will return the status of the switch:-

```
digitalRead(SWITCH);
```

This will return HIGH (or 1) if the switch is open with the line pulled-up, or LOW (or 0) when the switch is closed.

One possible approach of performing a simple 'blocking' debounce:-

- Check the status of the switch at short, known regular intervals, T (e.g. 1ms).
- Increment a 'tick-count' variable each time the switch status is read as 0 (i.e. switch depressed)
- Reset the 'tick-count' variable to 0 each time the switch status is read as 1 (i.e. switch open)
- Once ('tick-count' x T) equals the debounce time, then this constitutes a legitimate debounced keypress.

For the purpose of this exercise, a 1ms delay can be generated by using

```
delay(1);
```

This type of delay is known as a **blocking delay**. When a blocking delay is in effect, the microcontroller prevents or blocks any other operation from happening until the delay is completed. Later on in this course, you will learn how to implement non-blocking delays which allow other tasks to process whilst the delay is in progress.

7 Hardware Abstraction Layer (HAL)

A Hardware Abstraction Layer is beneficial for this task; processor-specifics for the 7-segment display and switch(es) can be deployed in the HAL.

You can use the HAL from Task 1 as a start point. The new functionality can simply be added (rather than starting from scratch) to create an extended HAL.

In the second half of the semester, you will engage in a summative assignment. For the summative assignment, a 'bare metal' approach will be used (i.e. digital pins being driven using register access rather than Arduino library functions). All the changes can be made by modifying the HAL accordingly. Additionally, the main code from these formative tasks can be used to test the modified HAL.