

Embedded Systems Fundamentals

ENGD2103

Dr M A Oliver

michael.oliver@dmu.ac.uk

Lecture 2: HAL and Simple Peripherals

Contents

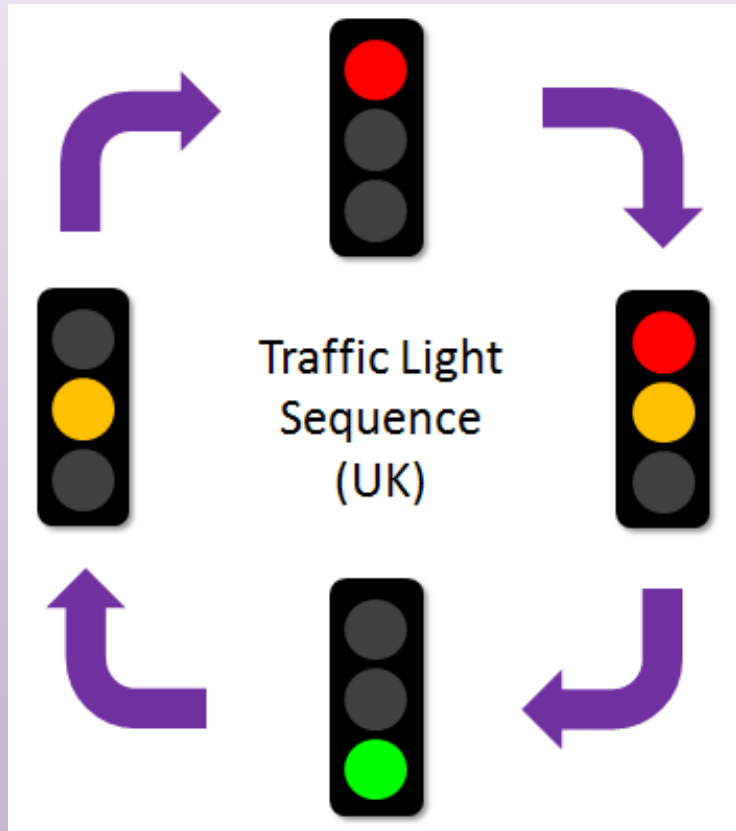
- Traffic Lights Controller
- Hardware Abstraction Layer (HAL)
- Seven Segment Display
- Shift Register
- Pushbutton, pullups and Debounce
- Pushbutton counter exercise

Traffic Lights Controller

This is the first exercise from the formative assessment. The full specification, with tips and recommendations can be found in the Laboratory Notes section on Blackboard.

- Hardware constructed last week from schematic provided.
- Task requires breadboard, microcontroller (Arduino Nano), red, amber and green LEDs, current limiting resistors.
- Write the firmware for a traffic lights controller to control a crossroads.
- Set 1 has *priority* over Set 2 (longer green time on Set 1)
- Observe the sequence of changing lights to see if the sequence is correct and the timings are correct.

Traffic Lights Controller



















Seconds	North-South Road	East-West Road
0	Red	Red
1	Red & Amber	Red
2	Green	Red
3	Green	Red
4	Green	Red
5	Green	Red
6	Green	Red
7	Amber	Red
8	Red	Red
9	Red	Red & Amber
10	Red	Green
11	Red	Green
12	Red	Green
13	Red	Amber

Traffic Lights Controller

For each state:-

- Set each LEDs on or off as required.
- Apply a delay.

The LED blink example from last week provides a good start point.

STATE	LIGHTS SET 1	LIGHTS SET 2	TIME (s)
1			1
2			1
3			5
4			1
5			1
6			1
7			3
8			1

Hardware Abstraction Layer (HAL)

Scenario

- Suppose you have a project containing multiple source code files.
- Suppose some or all of these files contain hardware-specific code.
- You are then told that hardware changes need to be made
 - Could be minor pin assignment changes
 - Could be processor change.
- Every file containing hardware-specific code would need changing.
 - Painstaking
 - Risk of errors / missing something

Hardware Abstraction Layer (HAL)

- Is there a better approach?
 - Yes!
- Create a module called the Hardware Abstraction Layer (HAL)
- Typically contains 2 files:-
 - Header (.h) file for definitions
 - Implementation (.c or .cpp) for functions
- All hardware-specific functionality is contained here.
- Other code modules simply use definitions or function calls from HAL.
- If hardware changes are to be made, simply change the HAL.
 - Good practice, all hardware specifics are in one place
 - Quick, easy and reliable method for making minor hardware changes.
 - Good structure for major changes (e.g. change of processor).

Hardware Abstraction Layer (HAL)

Example: Consider the LED blinker from last week:-

```
#define led 13 // Using the built-in LED.

void setup()
{
    pinMode(led, OUTPUT); // Make the LED pin an output
}

void loop()
{
    digitalWrite(led, HIGH); // Turn the LED on
    delay(500);              // Wait for 500 ms
    digitalWrite(led, LOW);  // Turn the LED off
    delay(200);              // Wait for 200ms
}
```

This code is full of platform-specific functionality. Not at all portable.

Hardware Abstraction Layer (HAL)

Create definitions in the HAL for driving the LED

The following could be incorporated into the header file

```
#define led 13

#define HAL_ledOn      digitalWrite(led, HIGH)
#define HAL_ledOff     digitalWrite(led, LOW)
```

Each definition is for external use should be preceded with HAL_

Each definition should have a descriptive name

Hardware Abstraction Layer (HAL)

- Assuming the HAL is developed in the files hal.h and hal.cpp
- The HAL needs to be `#included` in every module that uses it.

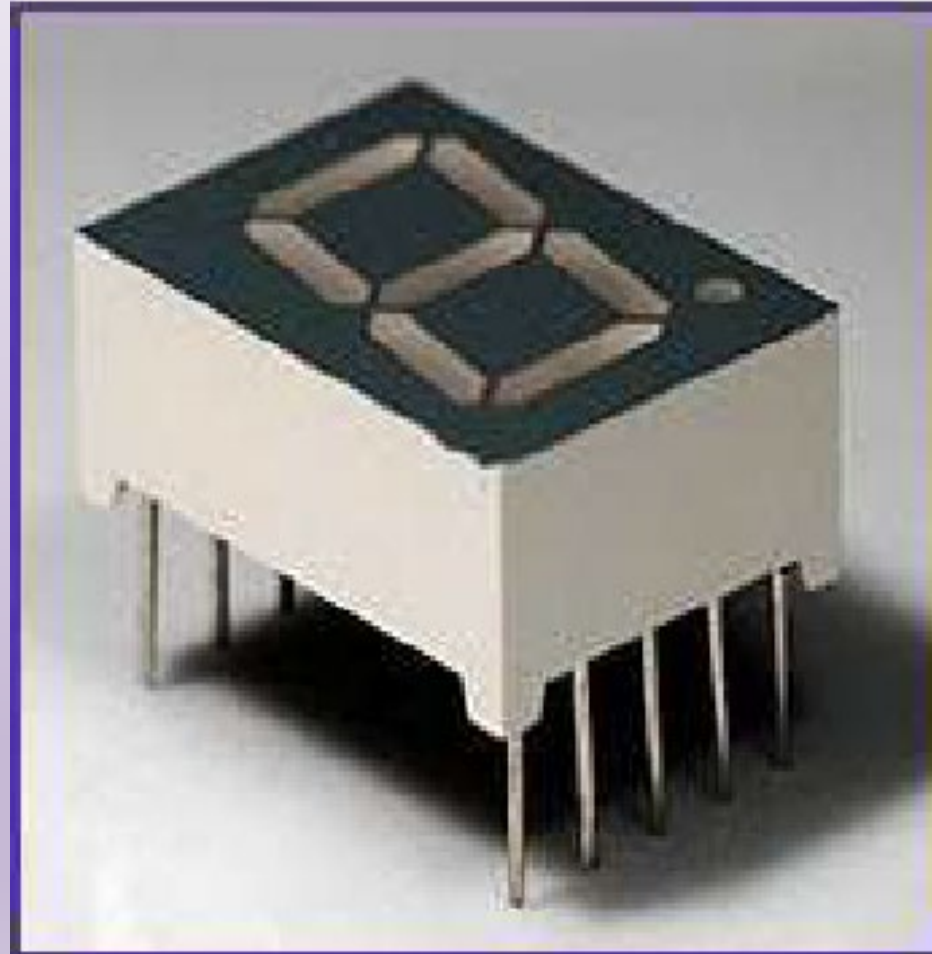
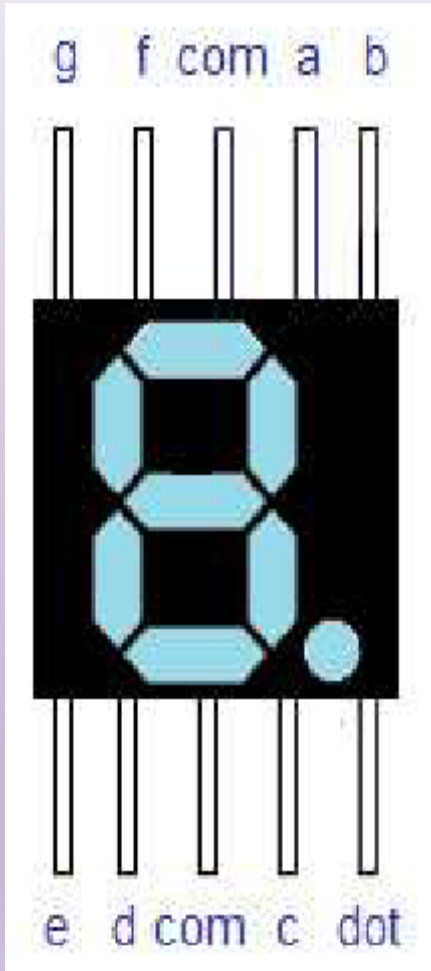
```
#include "hal.h"
```

- Using the definitions from earlier, the `loop()` function can be rewritten:-

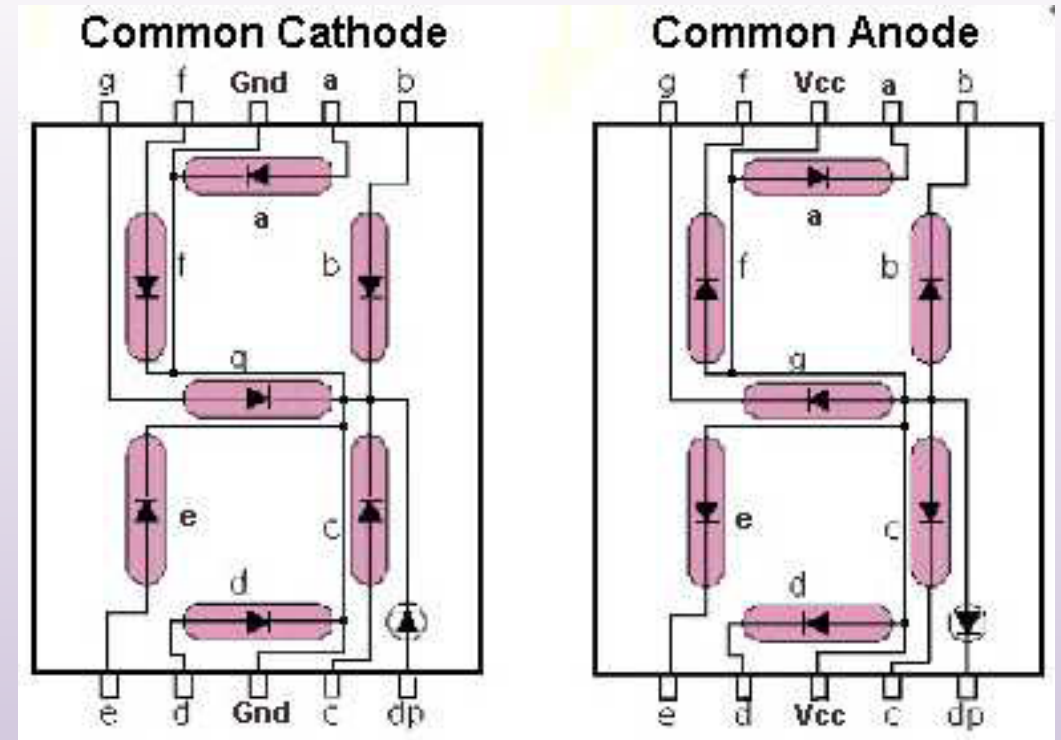
```
void loop()
{
    HAL_ledOn;           // Turn the LED on
    delay(500);           // Wait for 500 ms
    HAL_ledOff;          // Turn the LED off
    delay(200);           // Wait for 200ms
}
```

- In `loop()`, it is clear that
 - Calls are made to turn the LED **on** and **off**.
 - The mechanism for turning the LED on and off is abstracted from the code developer.
- Looking, at `loop()`, are all platform dependencies properly dealt with?
- How would you deal with the `setup()` function?
- Try applying a Hardware Abstraction Layer to your Traffic Lights controller.

Seven Segment Display



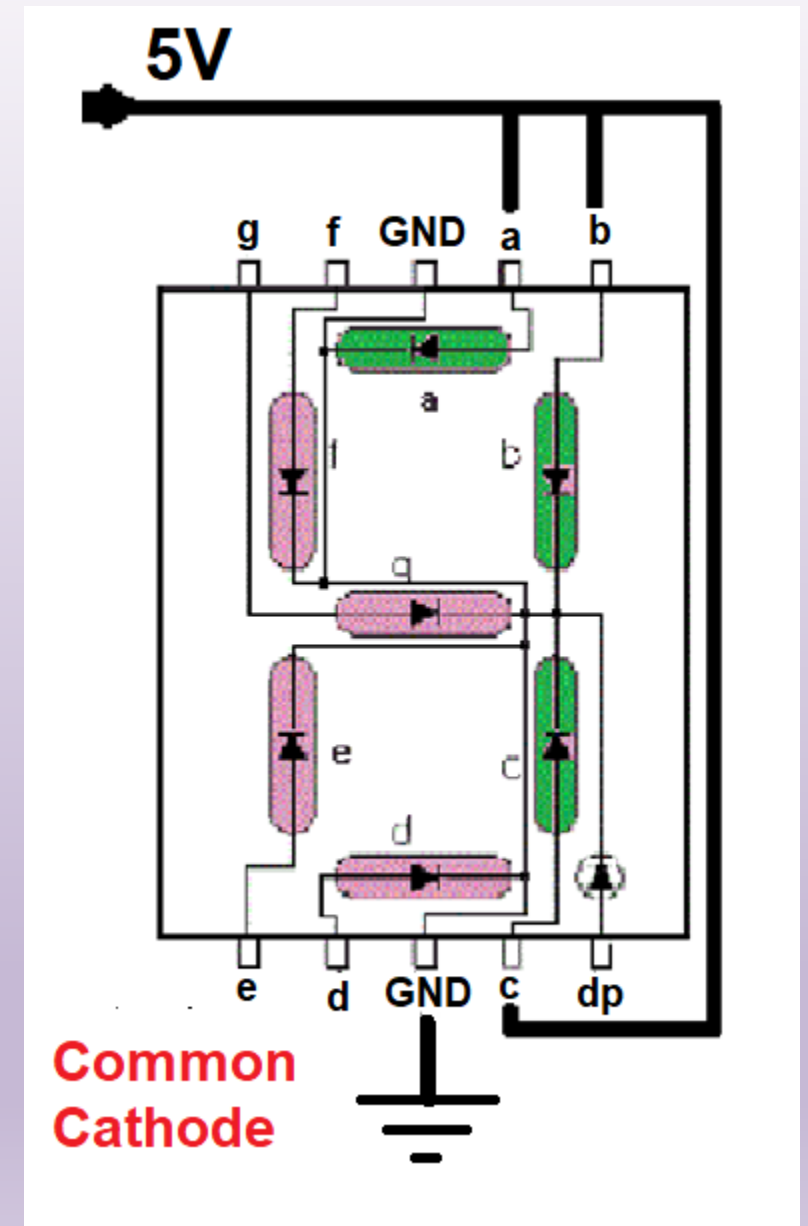
Seven Segment Display



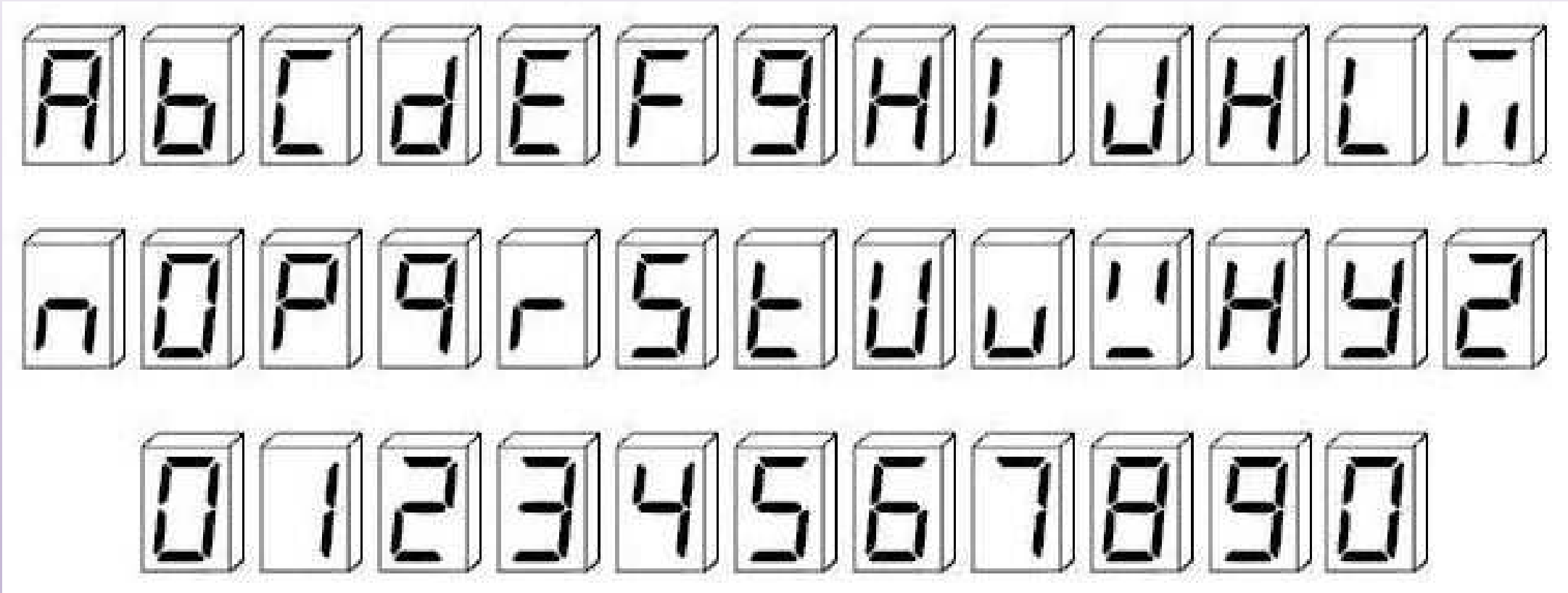
- LEDs like any other just arranged in a configuration
- Two different types: common cathode (ground) and common anode (Vcc)
- Your version is common cathode, i.e. you supply logic 1 to a segment pin to illuminate it, and logic 0 to extinguish it.

Seven Segment Display

- As with all LEDs, you need to protect them with current limiting resistors similarly to your Traffic Light Controllers.
- Images are build by lighting up (in green) some of the segments while keeping the rest dim (in lilac).
- Assume the unconnected pins in the image above are all grounded for the sake of clarity.
- Current limiting resistors are also omitted for the sake of clarity.

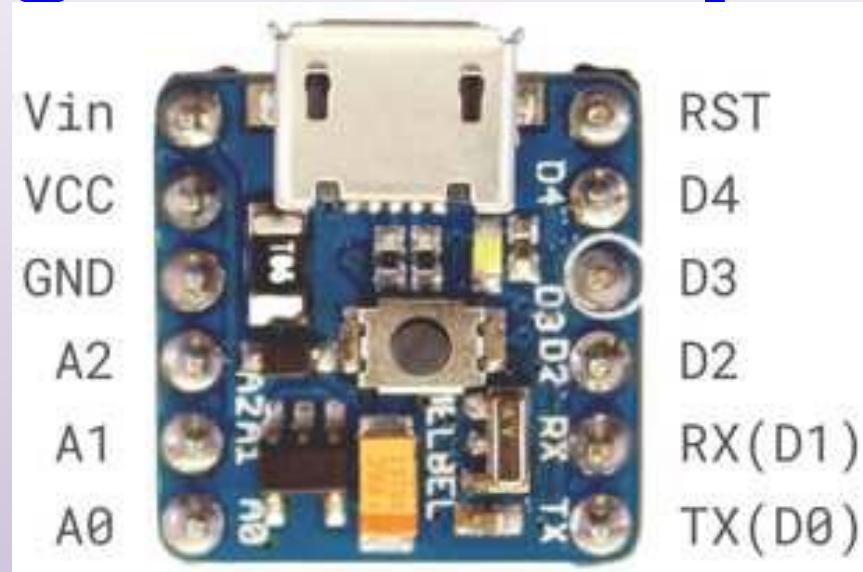


Seven Segment Display



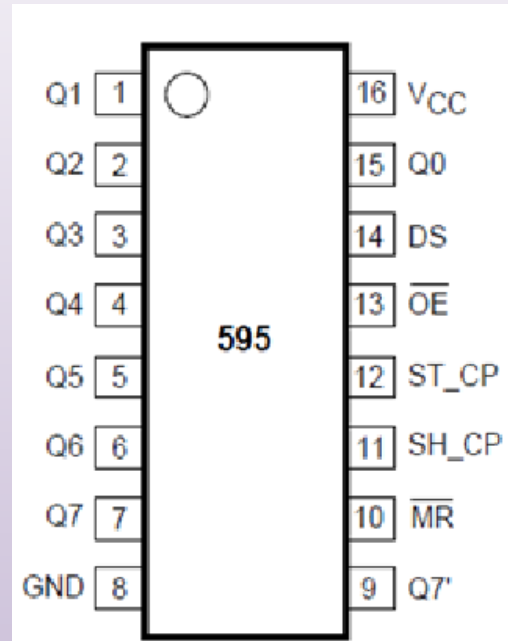
You need to drive 8 pins individually, if you want to display all possible images as per the figure above including the dot segment.

Seven Segment Display



- The PICO board is an example of an Arduino-compatible variant.
- It only has eight I/O pins in total!
- You need to use your I/O pins sparingly.
- Could a 7-segment display be *sensibly* interfaced to this board?

Shift Register 74LS595



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

- Data sheet at <http://www.ti.com/lit/ds/symlink/sn74hc595.pdf>
- Three inputs — we are interested in pins 11 (CLOCK), 12 (LATCH) and 14 (DATA) in green
- Eight outputs — pins 7, 6, 5, 4, 3, 2, 1 and 15 in red

Shift Register 74LS595

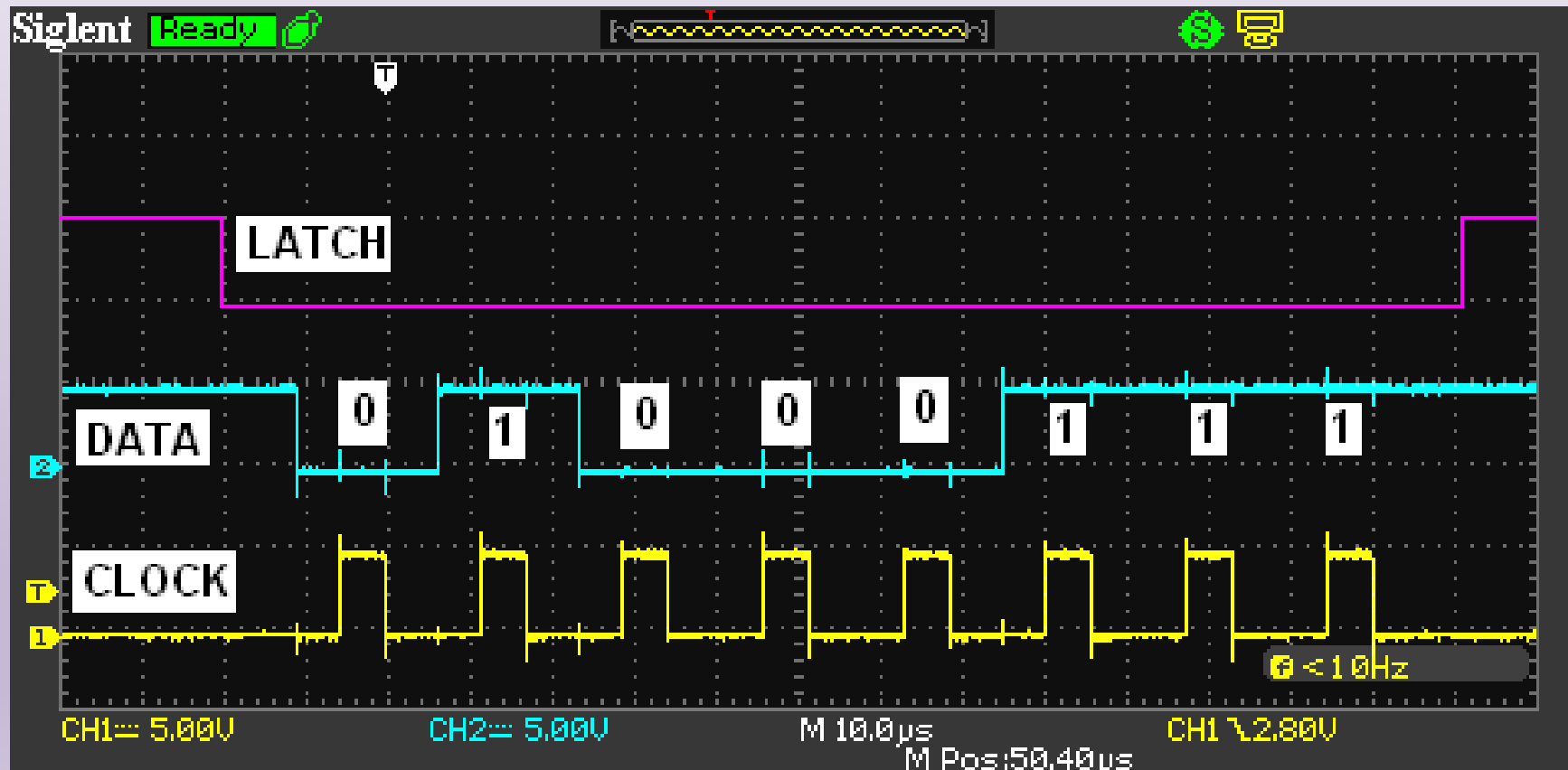
- Functional diagram
 - <http://www.ti.com/lit/ds/symlink/sn74hc595.pdf>, page 12, figure 4and
 - https://assets.nexperia.com/documents/data-sheet/74HC_HCT595.pdf, page 2, figure 1
- 74HC595 is made of a shift register, a storage register (latch) and a 3-state output.
- Connecting pin 13 to GND enables the 3-state output.
- Pin 10 must be connected to Vcc to prevent the register being reset.
- This leaves us with 3 inputs and 8 outputs
 - pin 9 is used for “daisy-chaining” more 595s to get larger registers.

74LS595 Protocol

- LATCH must be LOW, if you want to change what the shift register holds. LATCH goes to HIGH only to push the loaded value from the shift register to the outputs.
- CLOCK must be LOW, if you want to change the DATA. CLOCK goes to HIGH only to push the value of the DATA pin into the least significant bit of the shift register, resulting in all the bits already in the shift register being shifted by one in the direction of the most significant bit. This causes the most significant bit itself to “fall out” on pin 9 for “daisy-chaining”.
- DATA can be changed only when CLOCK is LOW. DATA must be stable when CLOCK is HIGH!

74LS595 Transaction

- Oscilloscope trace for a transfer of a byte to the shift register.



74LS595 'Bit Banging' Code 1

```
#define DATA      8          // DATA pin on the 595
#define CLOCK     12         // CLOCK pin on the 595
#define LATCH     13         // LATCH pin on the 595

void setup() {
    // TO DO:
    // Make the DATA line an output
    // Make the CLOCK line an output
    // Make the LATCH line an output
    // Pull the LATCH line low
    // Pull the CLOCK line low
}
```

74LS595 'Bit Banging' Code 2

```
void toggleLatch()  
{  
    // Pull the LATCH line high  
    // Pull the LATCH line low  
}  
  
void toggleCLK()  
{  
    // Pull the CLOCK line high  
    // Pull the CLOCK line low  
}  
  
void shiftBit(bool data)  
{  
    // Pull the DATA line high or low depending  
    // on the value of data  
    toggleCLK();  
}
```

74LS595 'Bit Banging' Code 3

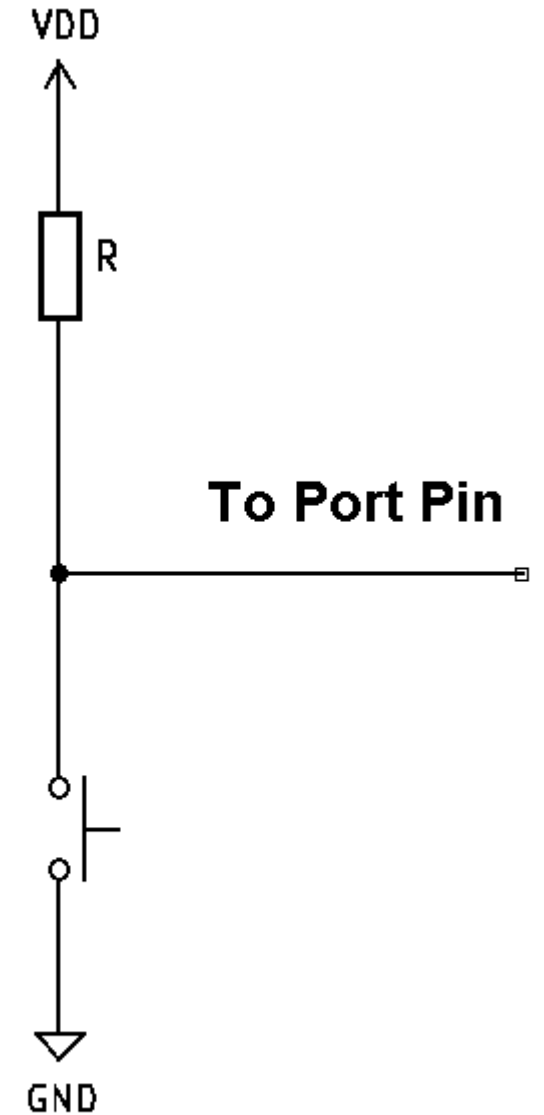
```
void writeToShiftRegister(unsigned char value)
{
    unsigned char mask = B10000000; // use mask to select the bits
                                     // in value one at a time

    // Create a loop to send all 8 bits of value to the shift
    // register. Use the value and mask variables to determine
    // whether each bit is high or low. Use shiftBit() to clock
    // each bit into the shift register

    // Once complete, call toggleLatch() to transfer the shifted
    // data to the outside world
    toggleLatch();
}
```

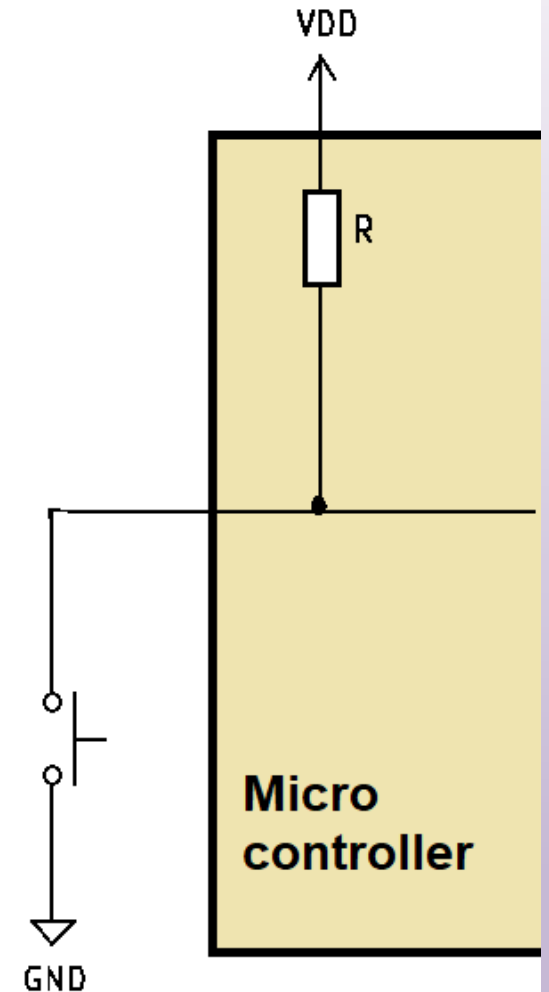
Push Button Hardware

- Switch is connected between digital (input) pin and ground.
- Digital (input) pin is connected to supply via a pull-up resistor.
- When switch is released, the digital pin is connected to supply via the pull-up resistor. i.e. **HIGH, logic 1**.
- When switch is pressed, the digital pin is connected to GND, i.e. **LOW, logic 0**.

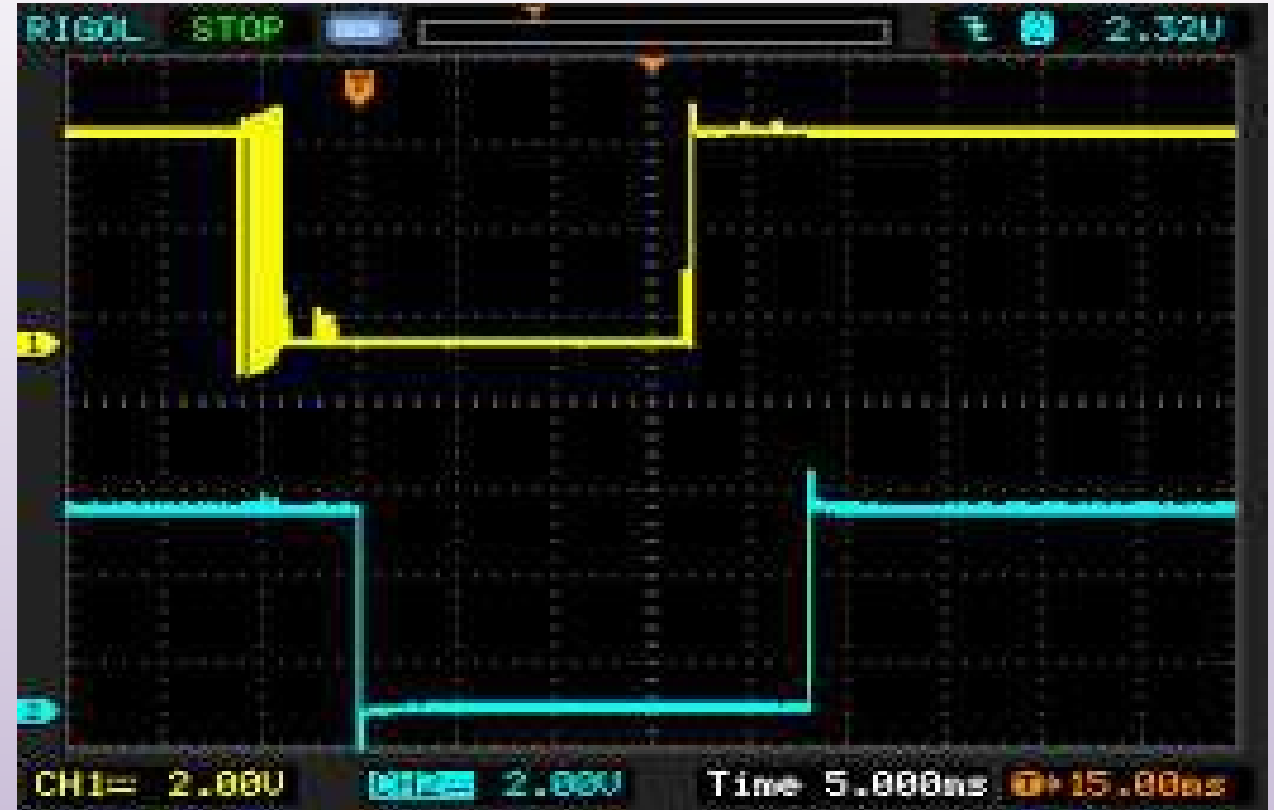


Push Button Hardware

- We will remove the external pullup resistor
- We will instead use the `INPUT_PULLUP` configuration for `pinMode()` which uses the internal pull-up on the Arduino's digital pins.
- This way we do not need the external hardware pullup resistor.

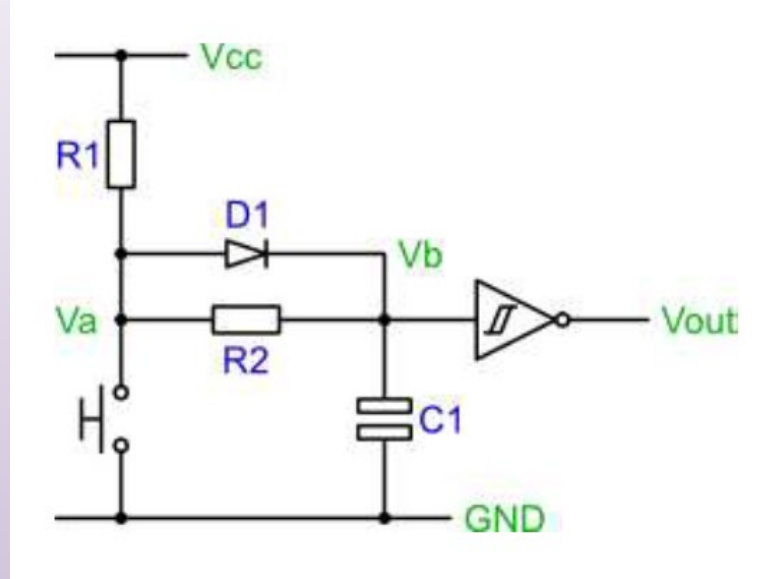


Push Button



- Bouncing behaviour (yellow trace)
- De-bouncing behaviour (blue trace)

Push Button Debouncing Hardware



- Some good notes on de-bouncing can be seen at
<http://www.labbookpages.co.uk/electronics/debounce.html>
<http://www.eng.utah.edu/~cs5780/debouncing.pdf>
- Using extra hardware increases the cost because extra components are required.
- Software de-bouncer is challenging but it is cost-effective.

Push Button Software Debounce: Blocking Debounce

- The following pseudo-code can perform a blocking software debounce:

```
count = N
while count > 0 do
    count = count - 1
    if switch is released then
        count = N
    end if
    wait d milliseconds
End while
```

- Loop exits once switch has been cleanly pressed for $(N \times d)$ milliseconds.
- Code in blue provides delay of $(N \times d)$ milliseconds.
- Code in red resets countdown if switch is released.

Debounced Button Counts

- This summarizes the second part of the Formative assignment.
- Count number of debounced button pushes and ignore button releases
- Display the count as a value between 0 and 15, roll-over to 0 when the count reaches 16
- Values from 0 to 9 to be displayed on the seven segment display as decimal digits
- Values 10, 11, 12, 13, 14 and 15 to be displayed on the seven segment display as the letters A, b, C, d, E and F correspondingly

Summary

- Looked at requirements for Formative Assignment Part 1: Traffic lights.
- Introduction to HAL.
- Considered the 7-segment display and shift-register.
- Considered the push-button and blocking switch debounce.
- Introduced Formative Assignment Part 2: Debounced pushbutton counts.