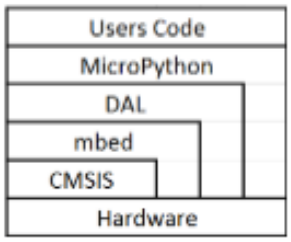
# Software Design Process

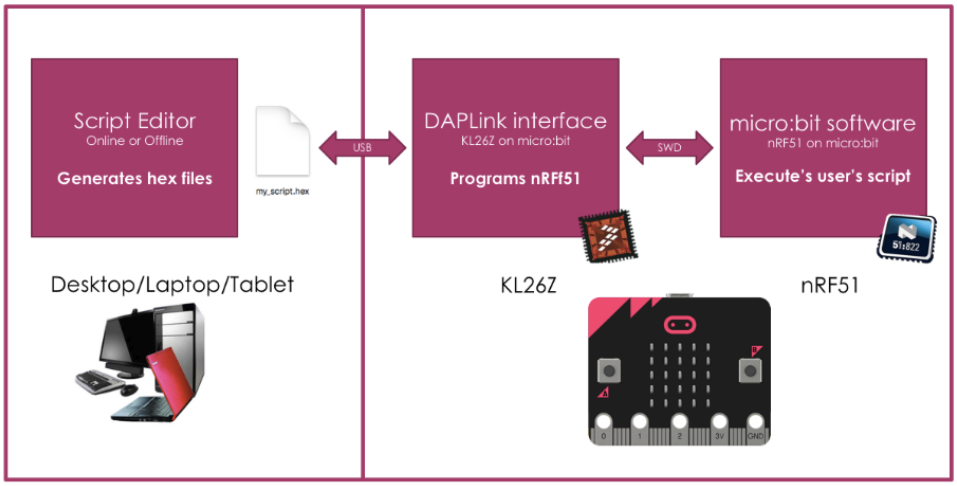
## Objective

Create a method to display student's code from their micro:bit onto a larger mega:bit for demonstration without affecting the micro:bit's performance. This must be done while taking into account the sensors and interactions with additional features such as radio communication. This has to be implemented in a manner such that any student's work can be displayed simply onto the mega:bit's LED matrix without need for additional coding or soldering from the teacher or student.

## Design Context and Software Stack




Several layers exist to translate the user code down to the hardware level. Each ‘sits’ on the top of the next and is procedurally stepped down to the hardware control. However, in some cases, mbed, DAL, and microPython can access the hardware directly. We will concentrate on the online MakeCode editor which differs from the diagram to the left. Instead of the MicroPython layer there are two different layers which connect the user code to the DAL which are called the 'pxt-microbit' and 'microbit' layers. Examples of code being translated through the software stack are at Appendix A.




The micro:bit runtime or device abstraction layer (DAL) contains all the hardware drivers for the micro:bit such as display, radio etc and runtime mechanisms that allow flexible programming of the micro:bit. The script editor generates the .hex files which are then uploaded to the micro:bit to be programmed and executed. The script editor combines the user code with the DAL to form the .hex files. We will insert our code here as it will apply to all user code formats.

## Design Selection Process

1) Accessibility Pin: One of the GPIO pins (p12) is a marked as a ‘reserved accessibility pin’ which by default is not accessible to the user. This was created to leave the Micro:Bit Foundation the option for expansion modules. The theory was for any future peripherals which required data communication this pin would provide a UART connection for data transfer. The Mega:Bit suits this function perfectly.

2) Neopixels: These are addressable multicoloured LEDs that are supported by the micro:bit neopixel library. They can be attached simply with 3 crocodile clip connections for data input, power and ground. A large number of neopixels cannot be powered by the micro:bit and need to be powered externally. The brightness of every LED can be adjusted independently in the software. These could be used for the mega:bit's display however they require strict timings which would require the use of interrupts. This has the potential to interfere with other low level activity on the micro:bit which also require strict timings and interrupts (for example the Bluetooth and RF radios).

3) I2C: The larger LED matrix could be controlled via I2C communication. An I2C LED display driver could be used to drive a matrix of LEDs which would be simple to implement as I2C is natively supported. I2C also has less strict timing requirements as the master can drive the clock at any frequency it would like (up to 400 kHz) in fact if the master has background tasks to do while transmitting it can 'stretch' the clock signal to greater allow for these. I2C bus utilisation must be investigated as we do not want to interfere with the other sensors on the I2C bus.

4) USB: Possibly provide two-way communication to control the mega:bit through the USB port or ‘clone’ the code onto the mega:bit. Two-way communication through the USB port is not feasible but cloning the code potentially is. The code is stored in a particular place in the on-chip memory which could be read and copied to the mega:bit. This however requires a level of 'intelligence' on the mega:bit side, for instance like a Raspberry Pi. This would obviously add cost and be more expensive than the alternative methods.

## Execution of I2C Method

The I2C method was chosen as it provides the simplest and cleanest solution however the accessibility pin may still be used to provide a handshake to confirm when the micro:bit is plugged into the mega:bit.

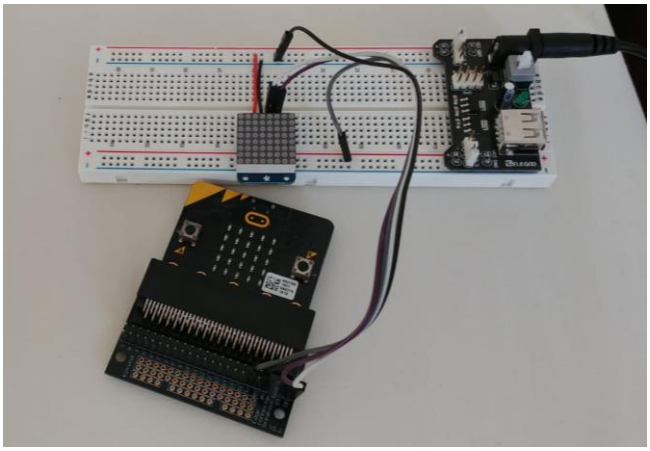
### Micro:Bit I2C Functionality

The I2C communication standard allows for many I2C slaves. The micro:bit has two existing devices on the I2C bus, the compass and accelerometer, we must ensure our use of the bus does not affect the existing functionality. This is done by ensuring there are no addresses clashes with the existing sensors. (See Appendix B for I2C address selection)

It is important to note that the 7-bit addresses found in the device datasheets have to translated with a binary shift left as an 8-bit address is to be used with the micro:bit. For example, the adafruit HT16K33 backpack has an address 0x70, that must be translated to 0xE0.

## Driving LED Matrix using I2C

Components used: Adafruit 0.8" 8x8 Matrix with HT16K33 Backpack, micro:bit edge connector.



An Adafruit backpack[1] was used for convenience as a comprehensive library exists. This means we can quickly and easily create a proof of concept before moving on to creating our own PCBs. The I2C pins of the micro:bit, pin 19 and 20 are accessed using an edge connector. The connections are made as follows:

| Microbit Pin | Backpack Pin |
|---|---|
| **Pin 19** | SCA |
| **Pin 20** | SDA |
| **GND** | GND |
| **3V** | VCC+ |

The DAL (Device Abstraction Layer) code was modified in order to implement the communication with the I2C external LED matrix.

The listing below shows the initialisation of the global variables needed to set up the LED display. The general structure of the code to interface with the HT16K33 LED driver was translated to C++ from a Python library for the backpack[2].

[1] Microbit Playground. (2016). *Control a 8x8 LED matrix with an I2C backpack on the microbit*. [online] Available at: https://microbit-playground.co.uk/components/8x8-matrix-HT16K33-microbit.

[2] https://bitbucket.org/thesheep/microbit-ht16k33/src/a5b7961f0b57ba226ab98edc2c5f8d95d8954d00/ht16k33.py

```
40  //------------------------------------------------------------------------------------------------------------------------
41  #include "MicroBitI2C.h"
42  #include "MicroBitDisplay.h"
43
44  MicroBitI2C i2c(I2C_SDA0, I2C_SCL0);
45
46  char buffer[17] = {};
47  uint8_t address = 0xE0;
48  uint8_t _HT16K33_OSCILATOR_ON = 0x21;
49  uint8_t _HT16K33_BLINK_CMD = 0x80;
50  uint8_t _HT16K33_BLINK_DISPLAYON = 0x01;
51  uint8_t _HT16K33_CMD_BRIGHTNESS = 0xE0;
52  bool i2cPresent = false;
```

## Configuring Display on LED Matrix

The listing below shows the initialisation process of the LED display. After having checked whether the I2C ports are being used (i.e. if a mega:bit is connected), the buffer is filled with zeroes, the oscillator is turned on and an initial blink rate and brightness are set.

```
142 class StartUp
143 {
144 public:
145         StartUp()
146         {
147
148                 if ( i2c.read(0xE0, 0, 1) == 0 ) { i2cPresent = true; }
149
150                 if (i2cPresent) {
151                         buffer[0] = 0x00;
152                         for (uint8_t i = 0; i < 16; i++) {
153                                 buffer[i + 1] = 0x00;
154                         }
155                         char tmp[1] = { _HT16K33_OSCILATOR_ON };
156                         i2c.write(address, tmp, 1);
157                         set_blink_rate(0);
158                         set_Brightness(15);
159                 }
160         }
161 };
```

Eight main display control functions have been implemented:

| Function | Description |
|---|---|
| **set_blink_rate** | Sets the blink rate of the display |
| **set_Brightness** | Sets the brightness of the display |
| **update_Brightness** | Updates the brightness of the display to a new value |
| **fill** | Fills the display |
| **initialise** | Initialises the display (similar to the StartUp class above) |
| **_pixel** | Plots single pixels on the matrix |
| **pixel** | Converts x coordinates and calls _pixel function |
| **bufferClear** | Clears the buffer |

### Replicating All Functionality

The micro:bit display is updated to match a 'buffer' which stores the image/text to be displayed. It works on an interrupt basis and is updated several times per second. The I2C display holds the previous state so does not need to be constantly updated, this would also take up unnecessary processing power and bus utilisation. Therefore, we only update the I2C display whenever the buffer is modified, this is the most efficient way of implementing the code.

## Additional Work

After successfully implementing the Adafruit LED backpack we moved to developing custom PCBs, these used the same HT16K33 chips so the software was unaffected.

### Handshake

A simple handshake was then implemented; using a 555 timer a 1kHz square wave is supplied to the accessibility pin of the micro:bit. When the microbit is powered up, it senses the presence of the square wave on its accessibility pin. If there is no 1kHz signal, the micro:bit will function as per usual. However, if it is present, a boolean variable is set to true and thus whenever the buffer is edited, through an if statement, the I2C display is also updated. This means that extra processing power is not used to send I2C commands when the micro:bit is not plugged into the mega:bit. This can also be adapted to function as an identifier, different signal frequencies indicating different peripherals.

### IS31FL3737

As explained in the hardware section of the GitHub repository (found at https://github.com/LukeB101/Mega-Bit/tree/master/Design%20History/Hardware) after implementing all functionality described above, we investigated using a different chip which runs off 3.3V rather than 5V and allows for other functionality. This chip requires different code which is still under development as the custom PCB did not arrive in time to write the software before university deadlines. This software will be developed outside of this university project.

### Further References:

"Microbit-RTCC-MCP7941X - Program To Interface Bbc Microbit To A MCP79410... | Mbed". *Os.Mbed.Com*, https://os.mbed.com/users/euxton/code/microbit-RTCC-MCP7941X/file/92208e3aae5a/main.cpp/.

"Driving Adafruit I2C 8X8 LED Matrices With The BBC Micro:Bit". *Smythe-Consulting.Com*, 2017, http://www.smythe-consulting.com/2017/03/driving-adafruit-i2c-8x8-led-matrices.html.

Micro:Bit DAL: https://github.com/lancaster-university/microbit-dal

Micro:Bit Software Stack: https://github.com/lancaster-university

# Appendix A

Using the JavaScript editor at **makecode.microbit.org**

The different headings show different layers in the software stack

## JavaScript

Taking the code:

```
basic.showNumber(0)
```

## Pxt-microbit

This is first translated by pxt-microbit/libs/core/basic `void showNumber(int value, int interval = 150) {`

https://github.com/Microsoft/pxt-microbit/blob/master/libs/core/basic.cpp#L19

(as number does not need scroll) in to: `uBit.display.printChar(t.charAt(0), interval * 5);`

uBit is defined in pxt.cpp: `MicroBit uBit;`

https://github.com/Microsoft/pxt-microbit/blob/master/libs/core/pxt.cpp#L4

## microbit

This is passed to the DAL wrapper where the display class is defined: `MicroBitDisplay display;`

https://github.com/lancaster-university/microbit/blob/master/inc/MicroBit.h#L111

## microbit-DAL

This is passed to the DAL where the printChar function is defined:

https://github.com/lancaster-university/microbit-dal/blob/master/source/drivers/MicroBitDisplay.cpp#L606

The printChar function calls the printCharAsync function: `this->printCharAsync(c, delay);`

Which is defined here:

https://github.com/lancaster-university/microbit-dal/blob/master/source/drivers/MicroBitDisplay.cpp#L481

The printCharAsync function calls the image.print function: `image.print(c, 0, 0);`

Which is defined here:

https://github.com/lancaster-university/microbit-dal/blob/master/source/types/MicroBitImage.cpp#L572

and calls: `this->getBitmap()[y1*getWidth()+x1] = (v & (0x10 >> col)) ? 255 : 0;`

## Conclusion

From this and other examples it is clear our software edits must be done within the MicroBitImage.cpp file since this is at the lowest level and deals with the buffer.

# Appendix B

## Adafruit I2C Sensor Addresses

To investigate the possibility of I2C address clashes a table was created displaying I2C sensors on the market and their addresses. Adafruit and SparkFun were chosen specifically as our client indicated any sensors used in schools would be used in a breakout board package and that these are the market leaders.

The table below and the *i2c-addresses.pdf* file show the range of I2C addresses used in their sensors. This data will be used to ensure the Mega:Bit does not clash with many, if any, other devices.

For the IS31FL3737 the address 0xAA was chosen, in 7bit format that is 0x55 which is taken by only one device which has 7 selectable address therefore does not pose any major risk of device clashes.

| Sensor | I2C address | Webpage link |
|---|---|---|
| **TCA9548A I2C Multiplexer** | 0x70 (but can be adjusted from 0x70 to 0x77) | https://www.adafruit.com/product/2717 |
| **Adafruit I2C Controlled + Keypad Shield Kit for 16x2 LCD** | 0x20 | https://www.adafruit.com/product/715 |
| **FLORA Accelerometer/Compass Sensor - LSM303 - v1.0** | 0x32 & 0x3C | https://www.adafruit.com/product/1247 |
| **Flora Lux Sensor - TSL2561 Light Sensor - v1.0** | 0x39, 0x29, 0x49 | https://www.adafruit.com/product/1246 |
| **MCP9808 High Accuracy I2C Temperature Sensor Breakout Board** | Uses any I2C address from 0x18 thru 0x1F | https://www.adafruit.com/product/1782 |
| **Adafruit HTU21D-F Temperature & Humidity Sensor Breakout Board** | 0x40 | https://www.adafruit.com/product/1899 |
| **Adafruit Blue&White 16x2 LCD+Keypad Kit for Raspberry Pi** | 0x20 | https://www.adafruit.com/product/1115 |

| | | |
|---|---|---|
| **LCD Shield Kit w/ 16x2 Character Display - Only 2 pins used! - BLUE AND WHITE** | 0x20 | https://www.adafruit.com/product/772 |
| **RGB LCD Shield Kit w/ 16x2 Character Display - Only 2 pins used! - NEGATIVE DISPLAY** | 0x20 | https://www.adafruit.com/product/714 |
| **MaximIntegrated DS1307 – real time clock** | 0x68 | |
| **Melexis MLX90614 – infrarred thermometer** | 0x5A | |
| **FreeScale MMA8452Q 3-Axis Accelerometer** | 0x1C, 0x1D | |
| **AKM AK9753 IR Sensor** | 0x64H, 0x65H, 0x66H | |
| **AM2315 - Encased I2C Temperature/Humidity Sensor** | 0x5C | https://www.adafruit.com/product/2651 |
| **Adafruit BMP280 I2C or SPI Barometric Pressure & Altitude Sensor** | SPI avoid i2c address collisions | https://www.adafruit.com/product/2651 |
| **Adafruit Mini 8x8 LED Matrix w/I2C Backpack - Ultra Bright White** | between 0x70-0x73 | https://www.adafruit.com/product/1080 |
| **Adafruit Bicolor LED Square Pixel Matrix with I2C Backpack** | between 0x70-0x77 | https://www.adafruit.com/product/902 |
| **Adafruit 16-Channel 12-bit PWM/Servo Driver - I2C interface - PCA9685** | between 0x40-0x7F | https://www.adafruit.com/product/815 |
| **Adafruit 16-Channel 12-bit PWM/Servo Shield - I2C interface** | between 0x60-0x80 | https://www.adafruit.com/product/1411 |
| **ADXL345 - Triple-Axis Accelerometer (+-2g/4g/8g/16g) w/ I2C/SPI** | 0x53 | https://www.adafruit.com/product/1231 |