**UNIVERSITY OF CENTRAL LANCASHIRE**

**School of Engineering**

ER1731 EMBEDDED C ASSIGNMENT, 2017-18

**Design and Development of a Primary Smart Home System using Embedded Systems**

**University of Central Lancashire**

School of Engineering

**Department of Electronic and Robotics Engineering**

**ER1731 Embedded C Assignment**

We confirm that the project report we are submitting is entirely our own work and that any material used from other sources has been clearly identified and properly acknowledged and referenced. In submitting this final version of our report to the Turnitin anti-plagiarism software, we confirm that our work does not contravene the university regulations on plagiarism as described in the Student Handbook. In so doing we also acknowledge that we may be held to account for any particular instances of uncited work detected by the Turnitin anti-plagiarism software, or as may be found by the project examiner or project organiser. We also understand that if an allegation of plagiarism is upheld via an Academic Misconduct Hearing, then we may forfeit any credit for this module or a more severe penalty may be agreed.

**Project title: Design and Development of a Primary Smart Home System using Embedded Systems**

**ABSTRACT**

Advancements in Smart Home technologies have made Home automation a reality. Home automation serves the key aspect of providing assistance to the elderly and disabled individuals using intelligent and interconnected devices which include various sensors and actuators for the purpose of monitoring medical emergencies like falls or seizures. This project is concerned with the design and development of a primary smart home system for automatic heating and lighting control, and fall detection with provisions to access environmental parameter readings like temperature, ambient light levels, etc., by utilizing an STM32 microcontroller. The designing, compiling and debugging of the required system is executed in Mbed compiler which is a cloud based platform for STM32 Nuleo-64 Development Board with STM32F303RE MCU. The embedded project was a success, in that the designed system met the specifications given, however there is significant room for improvement for the system by using technologies like Internet of Things (IoT).

TABLE OF CONTENT

1. INTRODUCTION.…………………………………………………5-6
2. STM32 MICROCONTROLLER…………………………………….6 -10
3. ENVIRONMENTAL PARAMETER READINGS …………………11-13
4. DEBOUNCE FUNCTION FOR INTERRUPT CONTROL…….14
5. AUTOMATIC HEATING CONTROL ……………………………...15-17
6. AUTOMATIC LIGHTING CONTROL…………………………….17-18
7. FALL DETECTION ………………...………………………………19-21
8. COMPONENT COST LIST ………………………………………….21
9. TERMINAL RESULTS …………………………………………….22-26
10. CONCLUSION ………………………………………...................... 27
11. BIBLIOGRAOGHY…………………………………………………28-30
12. APPENDIX………………………………………………………….31-43

12.1 DS1621 FILE

12.2 EMBEDDED CODE FOR SMART HOME SYSTEM

12.3 PIN MAP FOR STM32 NUCLEO-64 BOARD

12.4 DETAILS OF DEVICES ON THE SPECIALLY DESIGNED SHIELD.

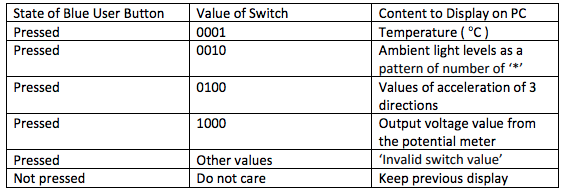
1. **INTRODUCTION**

The report deals with the process of designing, compiling, testing, debugging and developing a primary smart home system according to the specifications given below. Home automation requires usage of main hub or microcontroller to automate the tasks.

* 1. **Overall Aims of the Project**:

1. To understand the operation of an STM32 microcontroller and interpret the software used in such a system.
2. Compile, test and debug embedded C programs using industry de-facto development tools like Keil µVision 5 /Mbed Compiler which includes the standard mbed library necessary for our program.
3. Design hardware and software to meet the specifications of a primary smart home system.
   1. **Specifications for the system:**
4. Automatic heating control is designed to control the state of the thermostat by using the DS1621 sensor to measure temperature and to set the position of the servo motor according to the readings.
5. Automatic lighting control is designed using a light sensitive resistor (LDR) with a set of four red LEDs displaying light patterns corresponding to ambient light levels.
6. Fall detection system is implemented at any acceleration greater than 2g or lesser than -2g using the ADXL335 triple axis analogue accelerometer on the shield to trigger the alarm with RBG LED.
7. Environmental parameter readings:

The readings are summarized as follows:



1. **STM32 MICRO-CONTROLLER**

Microcontroller comprises of core, memory and peripherals. These elements distinguish microcontroller from microprocessor. ARM processor is from the family of [CPUs](https://whatis.techtarget.com/definition/processor) built on the performance based **RISC(reduced instruction set computer) architecture** developed by **Advanced RISC Machines (ARM)**. RISC processors are heavily based on registers. ARM is a 32 bit processor, that is, word size is 32.

ARM has 3 stage pipeline: *fetch*, *decode* and execute.

* Firstly, every instruction stored in memory is fetched.
* The fetched instruction is decoded to understand the operation.
* The instruction is then executed to perform the operation.

In this case, ARM will first fetch one instruction and while it is decoding the first instruction, it will fetch the second instruction. While ARM is executing the first instruction, the second instruction will be decoded and simultaneously the third instruction will be fetched. Thus when the third instruction is being decoded, the second instruction has already reached its execution stage. Thus, all the time wasted in fetching and decoding in different time is avoided.

The ARM Cortex -M4 is an example of HARVARD architecture (that is, programs and data are stored in 2 separate memories).

The STM32 F3 series microcontroller family is based on the ARM Cortex™ M4F core processor which offers a 64 bit MCU portfolio. The interpretation of STM32XXWWRY part number decoding for NUCLEO-F303RE is as follows:

1. XX indicates thefamily of **STM32 MCU product line**
2. WW indicates the subtype with respect to equipment of peripherals.
3. R indicates the package pin count (e.g.: **R for 64 pins**)
4. Y indicates the code size (**E for 512K**)

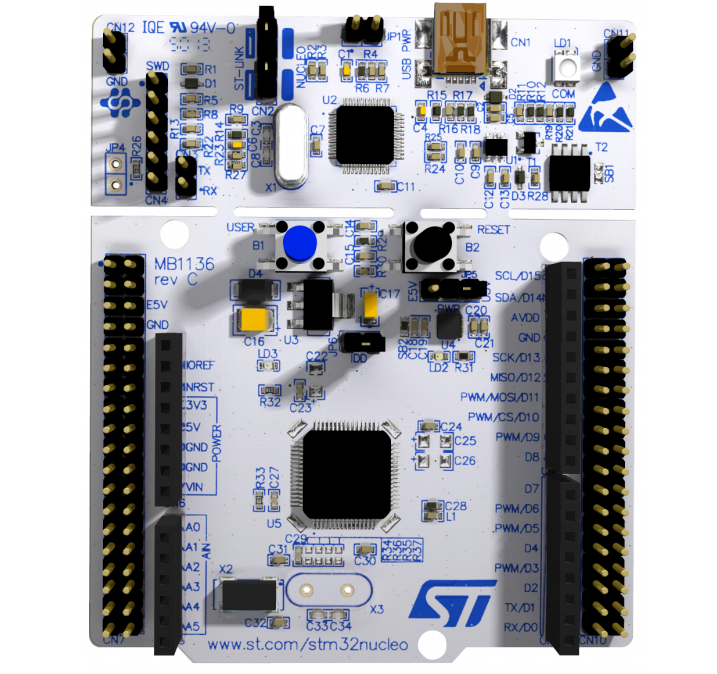
Development processes for embedded systems include programming in low level languages like assembly language or high level Language like C, Python, etc., (Toulson R., Wilmshurst T., 2012).

**Figure 1: The embedded program development cycle** (Toulson R., Wilmshurst T., 2012).

The major difference between mbed’s cloud based IDE to Keil µVision 5 is that Keil is a powerful professional IDE for embedded systems with advanced features like code completion, dynamic syntax checking etc., which allows for offline development whereas, mbed IDE is user friendly.

However, while working in groups it is more convenient to use a cloud based platform like mbed compiler as all the group members can access all the programs at one place, anywhere and anytime. Moreover, the Arm Mbed OS programs can be debugged using linker to a visual debugger like Keil uVision 5 to step through the code.

Moreover, ARM mbed™ Integration with STM32 Nucleo board allows free access to the extensive resources like mbed developer community, mbed cloud-based compiler and the mbed online C/C++ SDK for rapid embedded systems designing and prototyping.



BLACK RESET

BUTTON

BLUE USER

BUTTON

**Figure 2: STM32F303RE Microcontroller.** (STMicroelectronics NV, 2018).

1. **ENVIRONMENTAL PARAMETER READINGS**

The *if-else statement* is used to respond to the user button input. To make a decision within program, if **(button == 0)** is implemented with C equal operator == to check for the user button being pressed. User button needs to be pressed to execute any task for this program.

Note: when **button == 0** means the user button is pressed.

And when **button == 1** means the user button is not pressed.

Once the condition for the user button being pressed is met then the program reads the DIP 4 switches input using the *switch statement* to check bits set in Nibble.

The following code snippet outlines the program to read DIP switches input and execute the cases associate with them.

|  |
| --- |
| BusIn Nibble(D9, D8, D7, D6);  switch(Nibble & Nibble.mask()) {  // read the bus and mask out bits not being used.  case 0x01: … break;  case 0x02: … break;  case 0x04: … break;  case 0x08: … break;  default: printf("Invalid switch value. \n\r");  break;  } |

Note: **BusIn** abstraction is used for putting multiple inputs in a single pass. It is critical to notice the pin ordering during initialization. The pins in this program are initialized in the same order as that of bits.

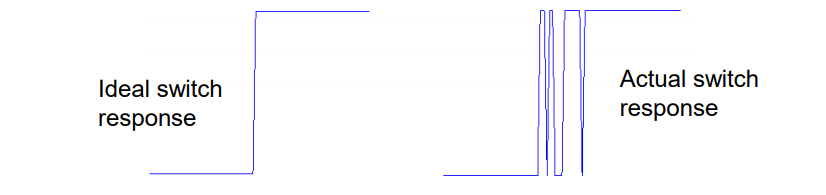
**Table 1 : Switches and user button state values used for smart home system**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **State of Blue User**  **Button** | **Binary value** | **Switch position**  **figure** | **Switch position**  **Case in HEX** | **D9** | **D8** | **D7** | **D6** |
| **pressed** | **0001** |  | **0x01** | **0** | **0** | **0** | **1** |
| **pressed** | **0010** |  | **0x02** | **0** | **0** | **1** | **0** |
| **pressed** | **0100** |  | **0x04** | **0** | **1** | **0** | **0** |
| **pressed** | **1000** |  | **0x08** | **1** | **0** | **0** | **0** |
| **pressed** | **other values**  **for e.g.:**  **(0000),**  **(1001)** |  | **0x00**  **0x09** | **0**  **1** | **0**  **0** | **0**  **0** | **0**  **1** |

1. **DEBOUNCE FUNCTION FOR INTERRUPT CONTROL**

When a single user button press results in multiple interrupts then the tasks to be executed on button press get out of sync with the button. This phenomenon is called "button bouncing". Removal of button bounces enable the system to act as if the user button has an ideal response. Hence, it is essential to debounce the user button with timer interface. (Toulson R., Wilmshurst T., 2012).

**is pressed ()** function considers the user button is pressed to display the value after the debounce time passed has passed 2000 milli-seconds.



**Figure 3:**  **User Button Bouncing.** (Toulson R., Wilmshurst T., 2012).

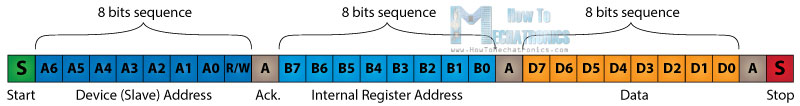
|  |
| --- |
| Timer debounce; // define timer object  // debounce is the local name of the timer  // function definition  void is\_pressed() {  if (debounce.read\_ms() > 2000) { // only allow calling the function is\_pressed() if the debounce time passed has passed 2 seconds.    printf("debounce the user button.\n\r");  debounce.reset(); // Reset the timer to 0 when the function is performed.  }  } |

Note that **function prototype** is not required in the program as the **function definition** is written before **main()** function.

1. **AUTOMATIC** **HEATING CONTROL**

Another feature of the smart home system is the automatic heating control. This feature of the device is connected to a thermostat and when set up properly can help ensure temperature regulation within the house without requiring the user to manually change thermostat levels. The I2C sensor DS1621with built-in processors as well as data registers, serves as the temperature sensor within this system and can be used to detect changes in temperature between -55oC to 125oc with a precision of up to 0.5oC (Maxim Integrated Products Inc. datasheet, 2005). The sensor reads the temperature around the microcontroller (including negative values). Whilst the temperature itself doesn’t change at a high rate, the sensor can detect relatively low changes. This I2C sensor measures temperature (which is an analog quantity) and converts them to digital signals. Digital signals are easier to store, to manipulate and to use for controlling a servo motor. However, after these signals are converted back to analog form to print out the temperature readings to the user, some accuracy and precision can be lost. The I2C protocol follows the following order:

* A start Bit
* 8-bit sequences that contains the address location of destination
* An acknowledge bit
* 8-bit sequence that contains address location of the sensor’s internal register
* An acknowledge bit
* The converted Temperature vales in 8-bit form
* An acknowledge bit
* A stop bit or reset bit.



**Figure 4: I2C Protocol**

As seen from the code in Appendix 2, the digital values read from the DS1621 sensor are printed out as a message.

The home system requires the controller to be on when the temperature falls below 25oC. This ensures that the temperature within the home remains relatively constant at around 25oC. An analog micro-servo is connected to the microcontroller to act as a thermostat controller. A servo is small motor device that has “fans” connected to its motor and its blades can be programmed to be positioned at different angular position. The servo used in this system is the TowerPro analog micro-servo which is a very powerful device that only weight 9grams but has a torque of 1.8 and as speed of 0.1seconds per 60-degree angle (Torq Pro & Tower Pro datasheet., 2014). For this system, the servo moves from an ‘off’ position of 0degrees to a position of 135 degrees when the thermostat is on. The servo remains in the 135-degree position to keep the thermostat on for as long as temperature levels as low and reverts to 0 degrees when home temperature goes over 25oC.

The code uses pulse-width-modulation (PWM) source to set the servo position.

**servo.period\_ms(20); //** This code line sets the period in milli-seconds(int).

**Servo.pulsewidth\_us(2500);**

The effect of this code snippet above will be to set the servo over a 20 milli- second cycle and a 2500 micro second (int) duty cycle, meaning the servo position will rotate to 135 degrees when width of the pulse is set to 2500 micro seconds (int).

Note: Usually servos range is from 1000 to 2000us. 1500 micro-seconds pulse width will set the servo position in the middle of its range (90 degrees), with 1000 micro-seconds to one end (0 degrees) and 2000 micro-seconds to the other end (180 degrees).

It should be noted that upon copy-pasting ° from character map, displays Temperature=28.0 Â° C. However, an unnecessary Â is displayed along with the rest of the sentence. Hence, UTF-16 Encoding for ° is stored in character variable degree and then printed using format specifier %c.

**char deg = 0x00B0;**

Not only is this heating control feature very helpful to ensure comfort within the home but it can also help save energy as well as money as the servo does not turn on the thermostat when it is not needed.

1. **AUTOMATIC LIGHTING CONTROL**

The smart home system is required to be able to automatically adjust light levels within home. This system is set up using a

Light dependent resistor (LDR) and LED lamps. An LDR is a simple low cost analog sensor made of semiconductor. As illumination increases, the LDR resistance decreases; this phenomenon is called *photo conductivity*. LDRs can be very cheap therefore in practice, multiple LDRs can be placed in different parts of the home so that only the rooms that need to be lit will be lit. It should be noted that the LDR output is non-linear. Hence, LDR is not used for precision measurements.

The program uses this property of LDR to vary light intensity of four red Led lights whilst displaying ambient light levels on the terminal using a '\*' pattern to represent light intensity level from 1 to 5 (See appendix 2).

|  |
| --- |
| float brightness;  brightness= 1-ADC.read();  float I\_val;  I\_val =brightness\*100;//Intensity value= I\_val |

From the code above, the value read from the LDR is ADC because this resistor converts analog values (surrounding light level) to digital signal that the processor can then use to operate the LED lights. The level of light intensity of the LEDs can be described as inversely proportional (i.e. if the ADC read value is high, the less light is required in the room at the time). The different light levels can be split in terms of percentages and since the full range scale for read operation is 1.0, the ADC read values is subtracted from 1 and the multiplied by 100 to give the I\_val which is the intensity in terms of brightness of the surrounding at an any given time.

DigitalOut Led1(D10);

DigitalOut Led2(D11);

DigitalOut Led3(D12);

DigitalOut Led4(D13);

As evident form the section of the code above, the LEDs are initialised and declared as DigitalOut therefore each led is controlled as a light level therefore one led on increases light intensity to the next level; this method used over Pulse Width modulation (PWM) (where all LEDs are turned on but different levels of brightness using duty cycle and frequency) because this can help the system be more cost effective as the LEDs have a longer life-span as only minimum required number of LED is used at any one time rather than all four every time.

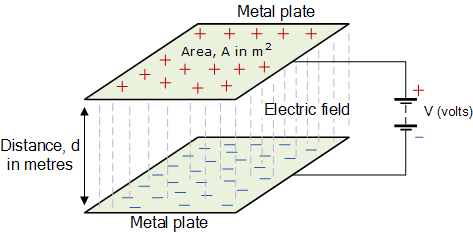
|  |  |  |
| --- | --- | --- |
| **I\_val (%)** | **Number of LEDs turned on** | **Light Level** |
| 0% | 4 | \*\*\*\*\* |
| Under 25% | 3 | \*\*\*\* |
| Under 50% | 2 | \*\*\* |
| Under 75% | 1 | \*\* |
| Over 75% | 0 | \* |

1. **FALL DETECTION**

A key aspect of the smart home system is to provide assistance for disabled and elderly users. Keeping this target demography in mind, it is important to note that these individuals may have limited mobility and are more at risk of falls. The home system includes an accelerometer that detects falls and sends an alert in form of a sequence of changing led lights (perhaps to alert carers). This can prove quite useful as it limits the need for users to be constantly watched which encourages independence whilst ensuring safety.

The use of the RGB LED lights also mean that the user can be easily located even in the dark. The code in appendix 2 enables the microcontroller to set off the alarm only when certain conditions are met, which helps prevent accidental triggers from smaller changes in acceleration (such as sitting down or laying down). The conditions to trigger the alarm is when acceleration falls below -2g or above 2g.

The accelerometer sensor used in the microcontroller is the ADXL335, a triple axis micro-electro mechanical systems device, which work out acceleration based on the changes in capacitance between the plates as the plates move due to force (due to acceleration) being exerted on them. The change in capacitance is converted into analog voltage (From the formula capacitance, C=Q/V (it is clear that capacitance is inversely proportional to voltage). According to manufacturer specification, the output voltage of the accelerometer goes from between 0V to 3.3V. Whereas the acceleration values can range from between 3g and -3g. the formula for calculation of acceleration in g is: **g** = (**A.Read -0.5)\*6** where **A.Read** is the analog read value at g. Using this formula as well the manufacturer's stated values, the expected analog values at 2g and -2g are calculated to be 0.833 at 2g and 0.167 at -2g.



**Figure 5: Capacitor plates and relationship to potential difference.**

(Electronic tutorials, n.d.)

The code then converts the analog read values outputted by accelerometer in to acceleration in all three directions (x, y, z). The acceleration values are then compared to 2g or -2g to check if a fall has been detected. If a fall is detected, the alarm sequence is triggered.

A RGB led is used as the light alarm, this advantageous over a sound alarm as it is not as loud as possibly inconvenient. The alarm protocol is shown below:

* Acceleration value is checked,
* acceleration value is compared to 2g and -2g to detect fall
* if it is not a fall
  + A message is printed showing current acceleration all direction.
* If a fall is detected
  + the message “it’s a fall” is printed alongside the acceleration values
  + The LED lights up in the color Red,
  + After 0.2 seconds, the led turns Blue
  + And 0.2 seconds the Led light turns Green.
* The acceleration values are checked, and the sequence followed as required.

It should be noted that light colour can be varied in order, length of ‘on time’ and in level of brightness as well as mixing colors by lighting up multiple colour at the same time.

This alarm can be beneficial because if implemented correctly can be used in pressure mats within the home as well as within personal alarms for the individuals.

1. **COMPONENT COST LIST**

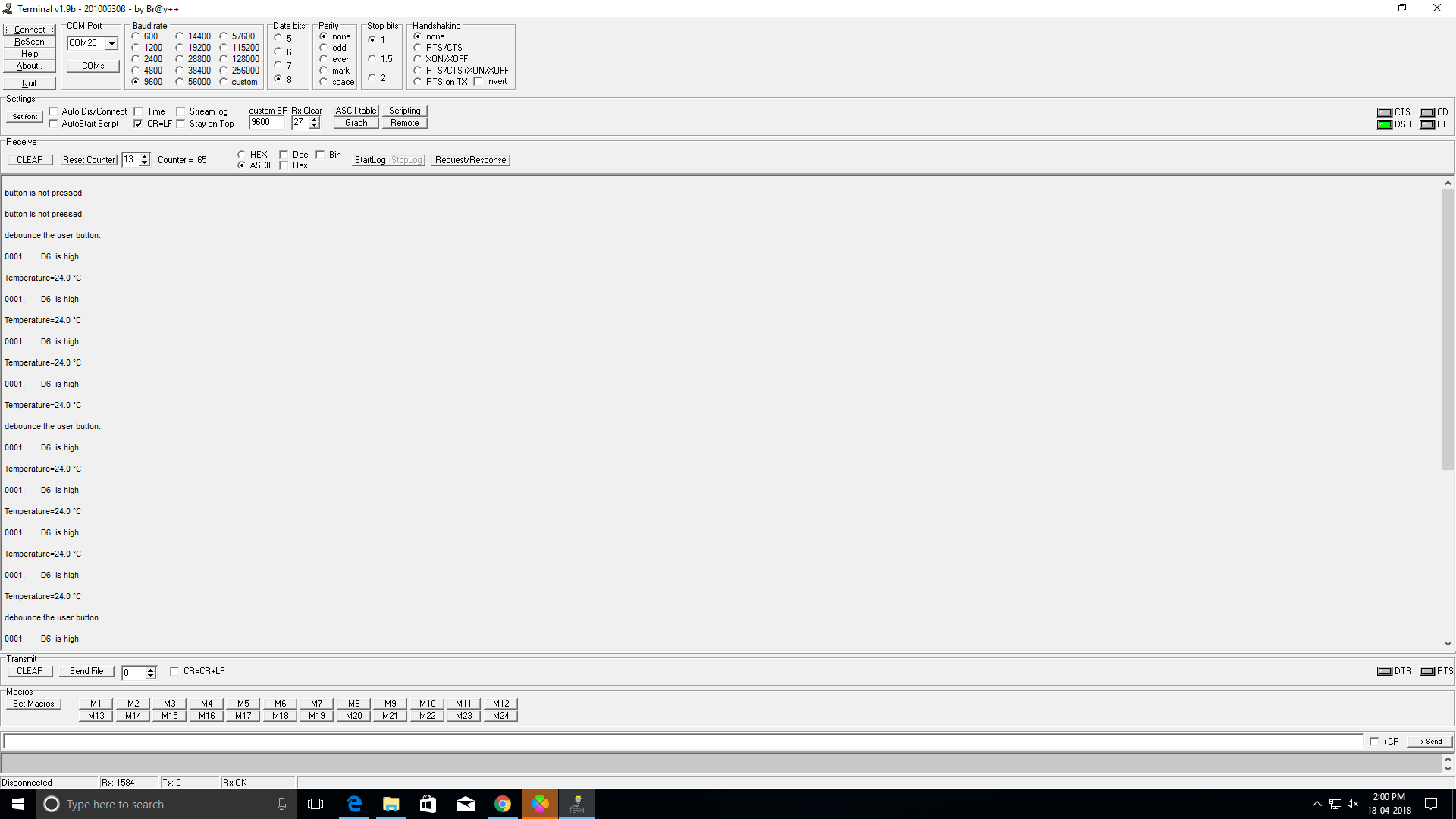
|  |  |
| --- | --- |
| Component/Device | Price |
| Male USB A to Male Mini USB B. | £2.18\* |
| DS1621 | £3.70\* |
| ADXL335 | £4.15\* |
| DIP switches | £0.904\* |
| STM32 Nucleo-64 boards | £12.53\* |
| SG90 Analog micro servo | £3.25 + |
| Single-Turn Cermet Trimmer. | £0.60\* |
| 5mm RGB LED common cathode | £0.73\* |
| Square LED Lights. | £0.20\* |
| NSL-19M51 LDR | £0.827\* |
| Total | **£29.07** |

\*Prices based on buying components as single units at <https://uk.rs-online.com/web/>.

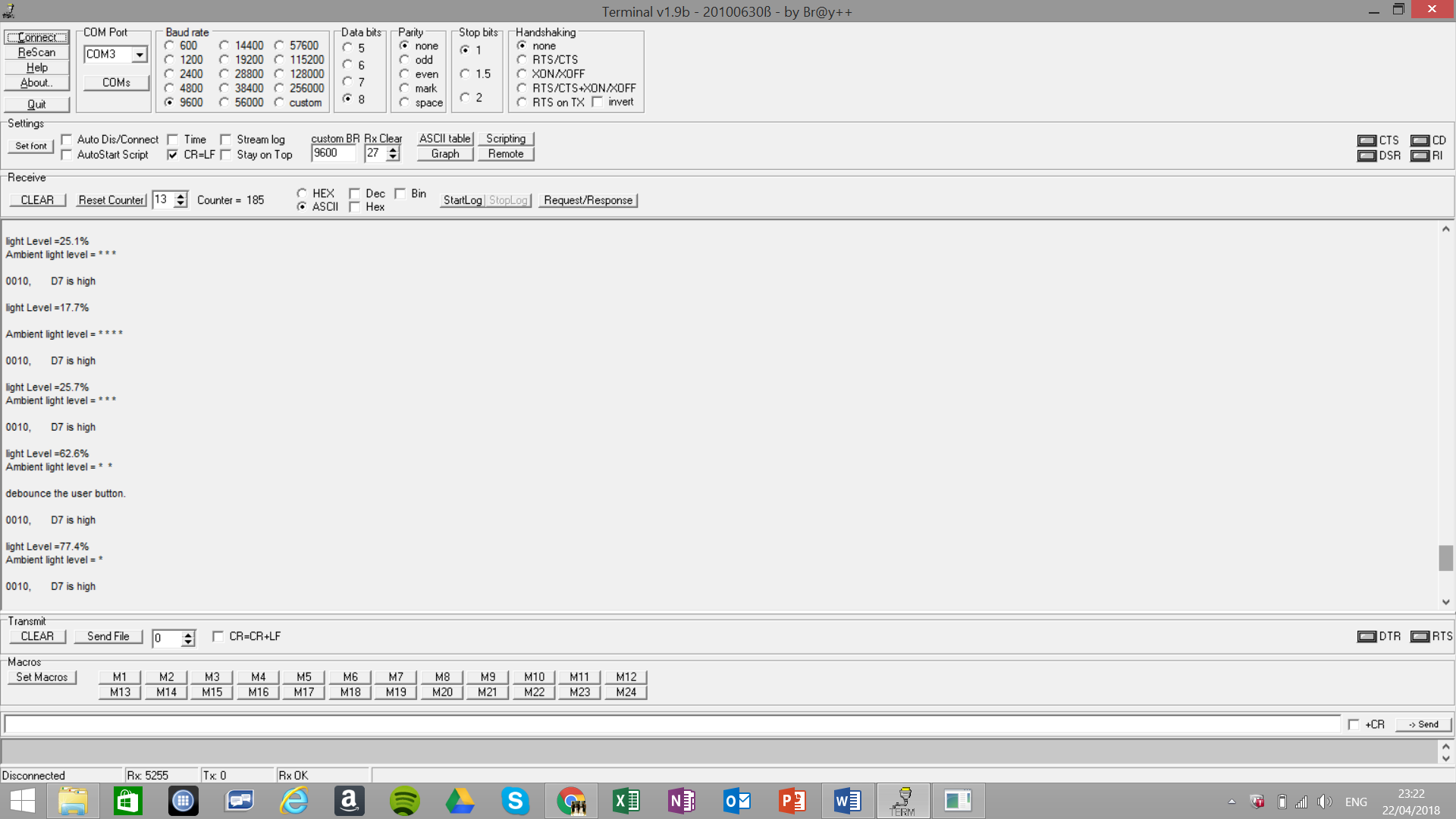
+ Price based on buying component as a single unit at <https://www.servoshop.co.uk>

1. **TERMINAL RESULTS**

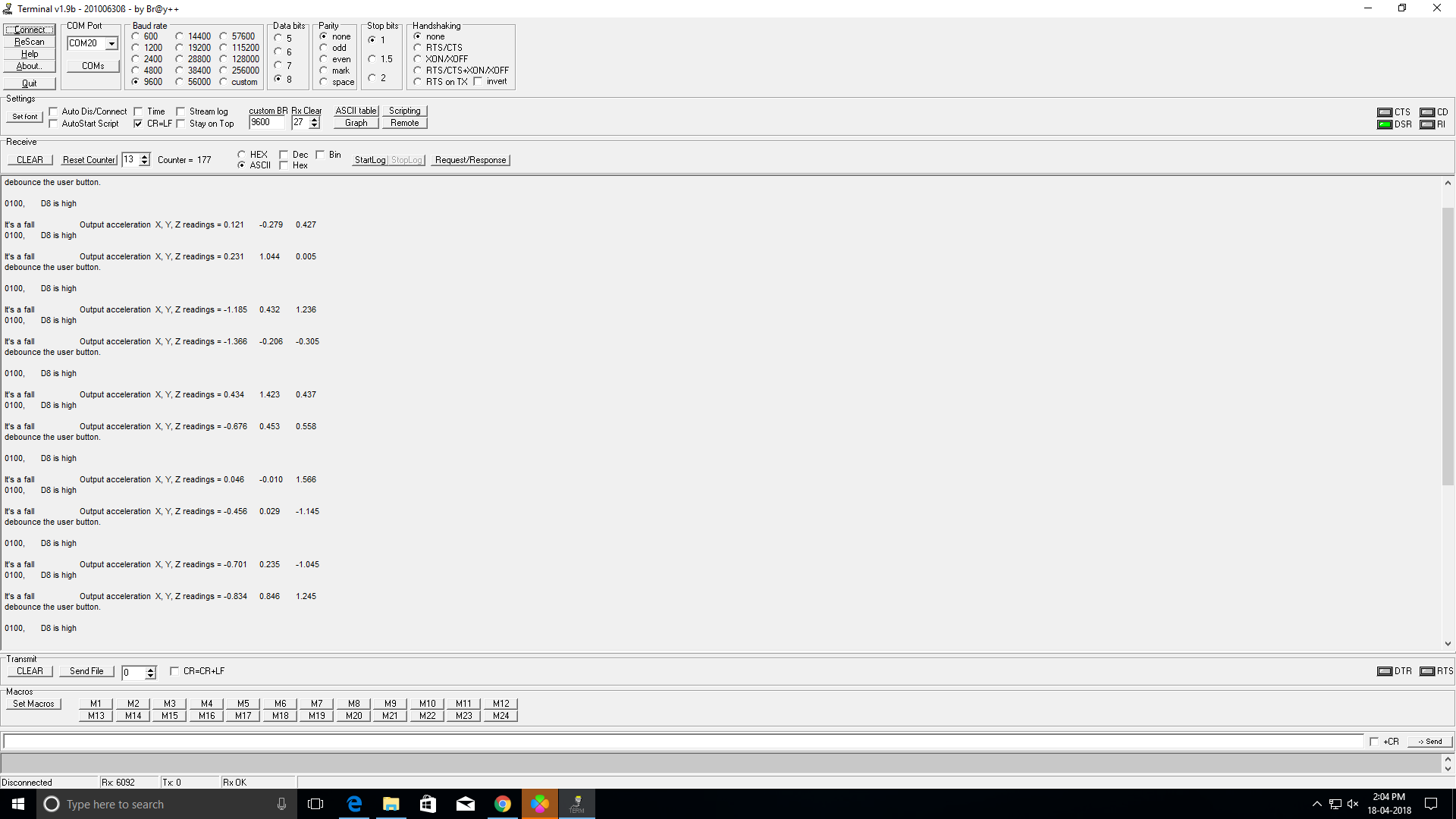
The following screenshots of the results illustrate the output displayed on the terminal for the designed system. The demonstration displaying the results on the STM32 Nucleo 64 development board is included in the video file accompanying this report.

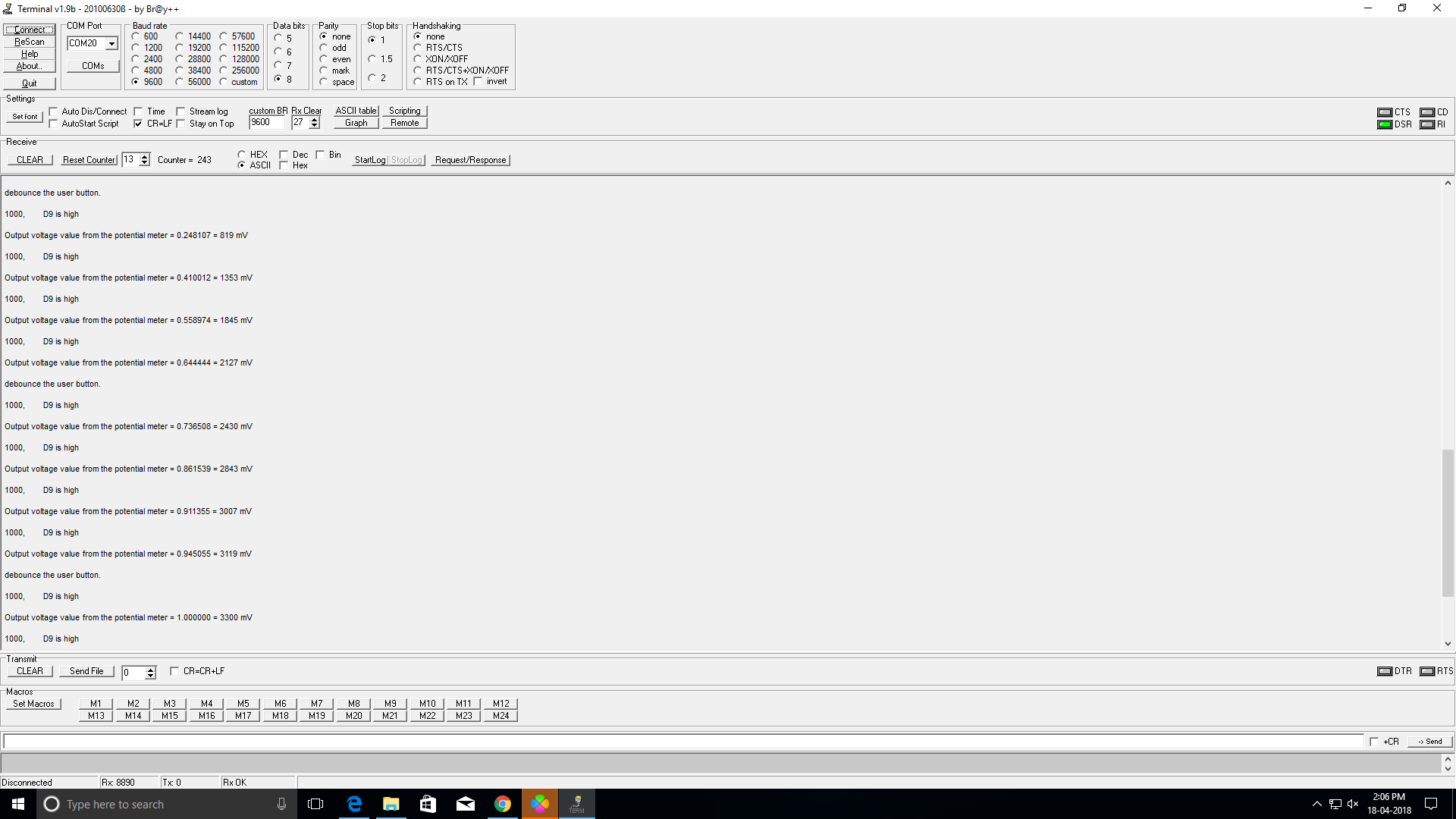
 9.1 Terminal result for switch state 0001 for temperature 

9.2 Terminal result for switch state 0010 for ambient light levels

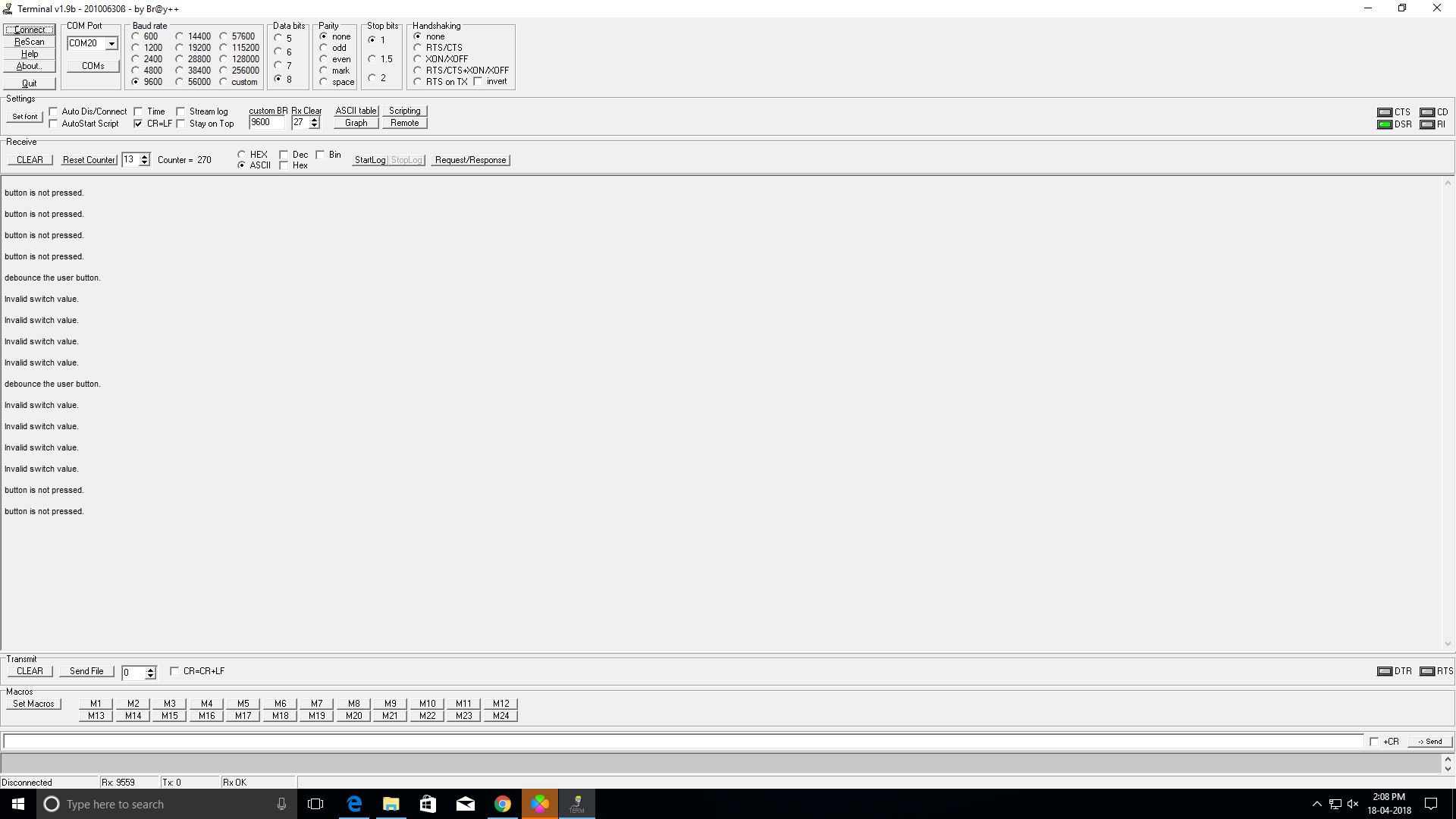


9.3 Terminal result for switch state 0100 for acceleration



* 1. Terminal result for switch state 1000: Output voltage readings form potentiometer.

9.5 Terminal displaying “Invalid switch value.” for any other switch state.



1. **CONCLUSION**

The designed system meets the specifications of a primary smart home system. There is future scope for development by using IoT features to send instant messages on alarm trigger to the Carer, Family and/or doctor. Also the location of the subject could be constantly tracked by using a GPS receiver. Additionally, deploying block chain in Iot enabled home networks could ensure security of the subject information. Overall, with progressing technologies, the home automation should become a cost-effective alternative to assist vulnerable individuals on a daily basis.

1. **BIBLIOGRAPGHY**

[1.] Maxim Integrated Products Inc. (2005) DS1621 Digital Thermometer and Thermostat. Available From: <https://www.promelec.ru/pdf/DS1621.pdf>

[Accessed 20 April 2018]

[2.] Analog Devices, Inc. (2010) ADXL335 Accelerometer. Available From: [http://www.analog.com/media/en/technical-documentation/data-sheets/ADXL335.pdf](http://www.analog.com/media/en/technical-documentation/data-sheets/ADXL335.pdf )

[Accessed 20 April 2018]

[3.] Mouser Electronics, Inc. (2017) DIP switches. Available From: <https://www.mouser.co.uk/datasheet/2/140/N211040-540279.pdf>

[Accessed 20 April 2018]

[4.] Mouser Electronics, Inc. (2017) STM32 Nucleo-64 boards Available From:  <https://www.mouser.co.uk/datasheet/2/389/nucleo-f030r8-973950.pdf>

[Accessed 20 April 2018]

[5.] Torq Pro & Tower Pro. (2014) SG90 Analog micro servo

Available From: <http://www.towerpro.com.tw/product/sg90-analog/>

  [Accessed 20 April 2018]

[6.]  Vishay Intertechnology, Inc. (2012) 3/8" Square (10 mm) Single-Turn Cermet Trimmer.

Available From:  <https://www.rapidonline.com/pdf/68-0600e.pdf>

  [Accessed 20 April 2018].

[7.] Kitronik Ltd  (2018) 5mm RGB LED common cathode. Available From:  <https://www.kitronik.co.uk/pdf/5mm_RGB_led_common_cathode.pdf>

[Accessed 21 April 2018].

[8.]  Avago Technologies (2008) 2.5 mm x 7.6 mm Rectangular LED Lights.

Available from: <https://docs-emea.rs-online.com/webdocs/0020/0900766b80020da9.pdf>

[Accessed 20 April 2018].

[9.]  Luna Optoelectronics (2016) NSL-19M51 LDR. Available from:  <https://docs-emea.rs-online.com/webdocs/1566/0900766b8156674e.pdf>

[Accessed 20 April 2018].

[10.] Toulson R. , Wilmshurst T. (2011), Fast and Effective Embedded Systems Design : Applying the ARM mbed. Oxford OX5 1GB, United Kingdom. Elsevier .Ltd, pp. 70.

[11.] Toulson R., Wilmshurst T.(2012), image, viewed 20 April 2018,

< <https://os.mbed.com/media/uploads/robt/mbed_course_notes_-_timers_and_interrupts.pdf> >

[12.] STMicroelectronics NV(2018), image, viewed 2 March 2018,<<http://www.st.com/content/ccc/resource/technical/document/user_manual/98/2e/fa/4b/e0/82/43/b7/DM00105823.pdf/files/DM00105823.pdf/jcr:content/translations/en.DM00105823.pdf>

[13.] Electronic tutorials. (n.d.). Capacitance and Charge. [online] Available at: <https://www.electronics-tutorials.ws/capacitor/cap_4.html>

[Accessed 21 Apr. 2018].

[14.] HowToMechatronics. (2018). How I2C Communication Works & How To Use It with Arduino. [online] Available at: <https://howtomechatronics.com/tutorials/arduino/how-i2c-communication-works-and-how-to-use-it-with-arduino/>

[Accessed 20 Apr. 2018].

[15.] Mouser Electronics, Inc. (2017) Male USB A to Male Mini USB B. Available From:  https://docs-emea.rs-online.com/webdocs/15fd/0900766b815fd231.pdf

[Accessed 22 April 2018]

1. **APPENDIX**

**Appendix 1:DS1621 FILE**

#include "mbed.h"

I2C i2c(I2C\_SDA, I2C\_SCL); //i2c device constructor

const int addr = 0x9E; //8 bit (addr|0) - all config bits=0 (floating)

//First 4 bits are 1001 (set by manufacturer)

int\* ReadDS1621(void)

{

char cmd[2]; // command buffer (re-usable)

static int data[2] = {0, 0}; //data being read

cmd[0] = 0xAC;

i2c.write(addr, cmd, 1, false); // Access Config

cmd[0] = 0x02; // 2 = continuous convert, POL=1

i2c.write(addr, cmd, 1, false); //continuous conversions

wait\_ms(20); // allow time for ee write

cmd[0] = 0xEE; //238=start conversions - needs 10ms delay

i2c.write(addr, cmd, 1, false); //try sending stop

wait\_ms(20); //20ms

cmd[0] = 0xAA; //to issue 'Read temp'

i2c.write(addr, cmd, 1, false);

i2c.read(addr, cmd, 2, false);

data[0] = (int)cmd[0]; data[1] = (int)cmd[1];

return (int\*)data;

}

**Appendix 2. EMBEDDED CODE FOR SMART HOME SYSTEM**

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

#include "mbed.h"

InterruptIn button(USER\_BUTTON);//Interrupt on user button

Timer debounce; // define timer object

// debounce is the local name of the timer

void is\_pressed() {

if (debounce.read\_ms() > 2000) { // only allow calling the function is\_pressed() if the debounce time passed has passed 2 seconds.

printf("debounce the user button.\n\r");

debounce.reset(); // Reset the timer to 0 when the function is performed.

}

}

//DIP 4 switches

BusIn Nibble(D9, D8, D7, D6); // Change these pins to buttons on your board.

//DigitalOut led(D2); // toggle

extern int \* ReadDS1621(void);

PwmOut servo(D11);

// LDR

AnalogIn ADC(A5);

// 4RED LEDs

DigitalOut Led1(D10);

DigitalOut Led2(D11);

DigitalOut Led3(D12);

DigitalOut Led4(D13);

// RGB LED

DigitalOut led1(D2);

DigitalOut led2(D3);

DigitalOut led3(D4);

// adxl335

AnalogIn analog\_AccelX(A0); // output of x-axis at analog pin A0

AnalogIn analog\_AccelY(A1); // output of y-axis at analog pin A1

AnalogIn analog\_AccelZ(A2); // output of z-axis at analog pin A2

AnalogIn pot(A4);

// main() runs in its own thread in the OS

int main() {

int\* temp;// define address to store the array

float Vin\_r;

float Vin\_v;

while (true)

{

debounce.start(); // Start the timer

if(button == 0)

{

// Button is pressed

is\_pressed(); // considering the user button is pressed to display the value after the debounce time passed has passed 2000 milli-seconds.

// check bits set in Nibble

switch(Nibble & Nibble.mask()) {

// read the bus and mask out bits not being used

case 0x01:

printf("0001, D6 is high \n\r");

temp = ReadDS1621(); //get the temperature data

char deg = 0x00B0; /\*UTF-16 Encoding for ° ...however this gives ==> Warning: Transfer of control bypasses initialization "switch(nibble & nibble.mask()) {" \*/

if(temp[0] >= 25)

{

printf("Temperature=%i.%i %cC \n\r",temp[0],temp[1],deg); // use format specifier %c

//servo = 0 degree position; // thermostat controller is ‘off’.

servo.period\_ms(20);

servo.pulsewidth\_us(1000);

}

else if (temp[0]< 25)

{

printf("Temperature=%i.%i %cC \n\r",temp[0],temp[1],deg); // use format specifier %c

//servo = 135 degree position; // thermostat controller is ‘on’.

servo.period\_ms(20);

servo.pulsewidth\_us(2500);

}

break;

case 0x02:

printf("0010, D7 is high \n\r");

float brightness;

brightness= 1-ADC.read();

float I\_val;

I\_val =brightness\*100;//Intensity value= I\_val where at 100%, all LEDs will be off.

if(I\_val == 0)

{

printf("light Level =%3.1f%%\n\r", I\_val);

printf("Ambient light level = \* \* \* \* \* \n\r");

Led1.write(1); //duty cycle as 20% of 1 period (1 second)

Led2.write(1);

Led3.write(1);

Led4.write(1);

break;

}

else if(I\_val> 0 && I\_val<25)

{

printf("light Level =%3.1f%%\n\r", I\_val);

printf("Ambient light level = \* \* \* \* \n\r");

Led1.write(0); //duty cycle as 20% of 1 period (1 second)

Led2.write(1);

Led3.write(1);

Led4.write(1);

break;

}

else if(I\_val >=25 && I\_val <50)

{

printf("light Level =%3.1f%%\n", I\_val);

printf("Ambient light level = \* \* \*\n\r");

Led1.write(0);

Led2.write(0);

Led3.write(1);

Led4.write(1);

break;

}

else if(I\_val >=50 && I\_val <75)

{

printf("light Level =%3.1f%%\n", I\_val);

printf("Ambient light level = \* \* \n\r");

Led1.write(0);

Led2.write(0);

Led3.write(0);

Led4.write(1);

break;

}

else if (I\_val >=75 && I\_val<100)

{

printf("light Level =%3.1f%%\n", I\_val);

printf("Ambient light level = \* \n\r");

Led1.write(0);

Led2.write(0);

Led3.write(0);

Led4.write(0);

break;

}

else

{

printf("light Level =%3.1f%%\n", I\_val);

wait(0.1);

printf("ERROR!");

break;

}

case 0x04:

printf("0100, D8 is high \n\r");

/\* The code below uses the triple axis analogue accelerometer (adxl335) to detect fall and reperesent an alarm using RGB led.

Alarm is triggered when acceleration values fall above 2g or below -2g.

------------ Using the formula: g = [(Analog.read()-0.5)\* 6] to calculate the expected analogue value for acceleration at 2g and -2g.---------- \*/

float x=analog\_AccelX.read();

float y=analog\_AccelY.read();

float z =analog\_AccelZ.read();

float a=0.833; // where a is expected analog value for acceleration = 2g

float X= (x-0.500)\*6.000;// calculated acceleration based on read values from accelerometer

float Y= (y-0.500)\*6.000;

float Z =(z-0.500)\*6.000;

float b=0.167;// where b is expected analog value for acceleration = -2g

/\*Using Or operation to set the conditions for to trigger the alarm on any of the axis\*/

if (x>a|| x<b|| y>a|| y<b|| z>a|| z<b)

{

printf("It's a fall\t\t Output acceleration X, Y, Z readings = %3.3f\t%3.3f\t%3.3f\t\n", X, Y,Z);

wait(0.2);

led1.write(1); //led1 is red

led2.write(0); //led2 is green

led3.write(0); // led3 is blue

wait(0.2);

led1.write(0);

led2.write(1);

led3.write(0);

wait(0.2);

led1.write(0);

led2.write(0);

led3.write(1);

wait(0.2);

break;

}

else

{

printf(" Output acceleration X, Y, Z readings = %3.3f\t%3.3f\t%3.3f\t\n", X, Y,Z);

// Values of acceleration of 3 directions

wait(0.2);

}

break;

case 0x8:

printf("1000, D9 is high \n\r");

Vin\_r = pot.read(); // Read the analog input value (value from 0.0 to 1.0 = full ADC conversion range)

Vin\_v = Vin\_r \* 3300; // Converts value in the 0V-3.3V range

// Display values

printf("Output voltage value from the potential meter = %f = %.0f mV\n\r", Vin\_r, Vin\_v);

/\* Rotate the poteniometer to left to decrease the vale and to right to increase the output voltage value.

The output voltage values of the potentiometer ranges from 0.000000 (i.e., 0mV) 1.000000(i.e., 3300mV)\*/

break;

default: printf("Invalid switch value. \n\r");

break;

}

}

else

{

printf("button is not pressed.\n\r");

}

wait(0.5);

}

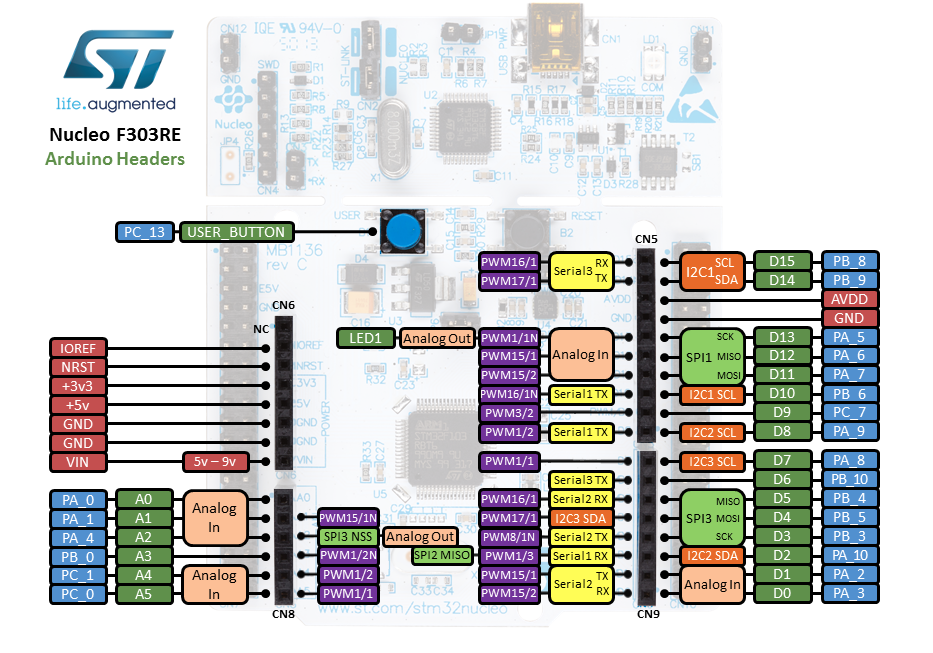
}

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% END OF CODE

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

**Appendix 3: PIN MAP FOR STM32 NUCLEO-64 BOARD**



**Figure 1 : Shield connectors on STM32 Nucleo 303 Microcontroller pin configuration.**

n.d. photograph, viewed 21 April 2018,

< <https://www.tenlong.com.tw/products/9487000878463> >.

**Appendix 4: Details of Devices on the specially designed shield**

