

Sheffield Hallam University
Department of Engineering
BEng (Hons) Computer Systems Engineering
BEng (Hons) Electrical and Electronic Engineering



Activity ID		Activity Title			Laboratory Room No.	Level
Lab 101		A gentle introduction to the STM32F7 discovery board		4302	6	
Term	Duration [hrs]	Group Size	Max Total Students	Date of approval/review	Lead Academic	
1	4	1	25	09-18	Alex Shenfield	

Equipment (per student/group)

Number	Item
1	STM32F7 discovery board lab kit

Learning Outcomes

Learning Outcome	
2	Demonstrate an understanding of the various tools, technologies and protocols used in the development of embedded systems with network functionality
3	Design, implement and test embedded networked devices

A gentle introduction to the STM32F7 discovery board and expansion kit

Introduction

Computing is about more than the PC on your desktop! Embedded devices are everywhere – from wireless telecommunications infrastructure points to electronic point of sale terminals. One definition of an embedded system is:

“An embedded system is a computer system designed to perform one or a few dedicated functions often with real-time computing constraints.”

[\(http://en.wikipedia.org/wiki/Embedded_system\)](http://en.wikipedia.org/wiki/Embedded_system)

In the laboratory sessions for this module you are going to be introduced to the STM32F7 discovery board – a powerful ARM Cortex M7 based microcontroller platform capable of prototyping advanced embedded systems designs. The STM32F7 discovery board includes advanced functionality such as Ethernet connectivity, UART over the USB connection, an LCD screen, and a micro-SD slot. Appendix A shows the various pins that are broken out from the STM32F7 discovery board (onto the Arduino form factor header), and Appendix B provides a schematic showing how these map to the headers on the SHU base board.

This laboratory session aims to introduce you both to some basic electronics principles and to the STM32F7 discovery board (as well as the ST HAL driver library and the rest of the hardware we will be using throughout this year).

Bibliography

There are no essential bibliographic resources for this laboratory session aside from this tutorial sheet. However the following websites and tutorials may be of help (especially if you haven't done much electronics previously or your digital logic and/or programming is a bit rusty):

- <http://www.cs.indiana.edu/~geobrown/book.pdf>¹
- <https://visualgdb.com/tutorials/arm/stm32/>
- http://www.keil.com/appnotes/files/apnt_280.pdf
- <https://developer.mbed.org/platforms/ST-Discovery-F746NG/>

¹ Note: this book is for a slightly different board – however, much of the material is relevant to the STM32F7 discovery

Methodology

Task 1

1. Check that you have all the necessary equipment (see Figure 1)!

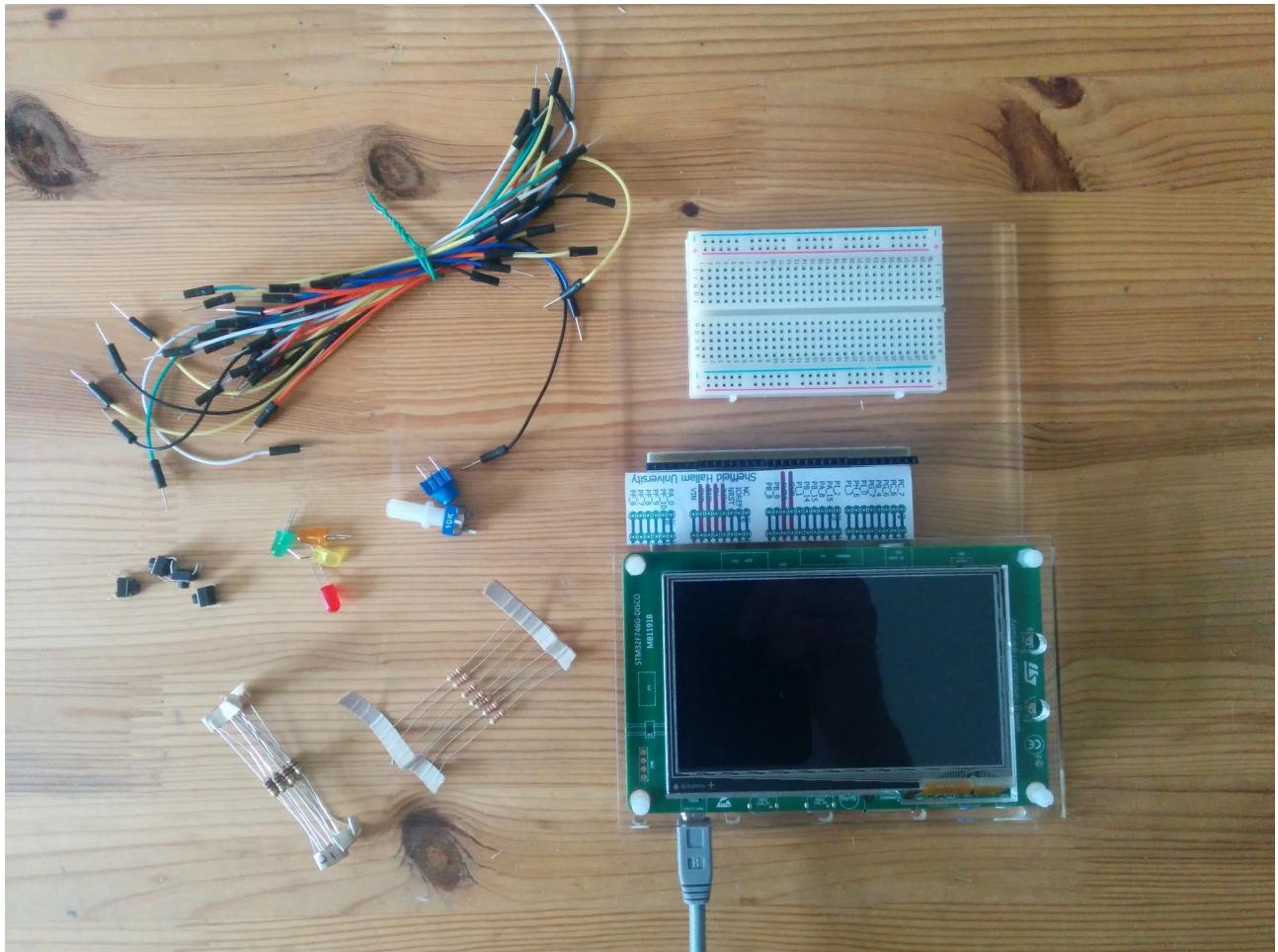


Figure 1 – The necessary equipment for this lab (don't worry if you have LEDs of a different colour)

2. To connect the breadboard to the STM32F7 discovery board you will need some male to male jumper wires. Figure 2 shows some solid core jumper wire.

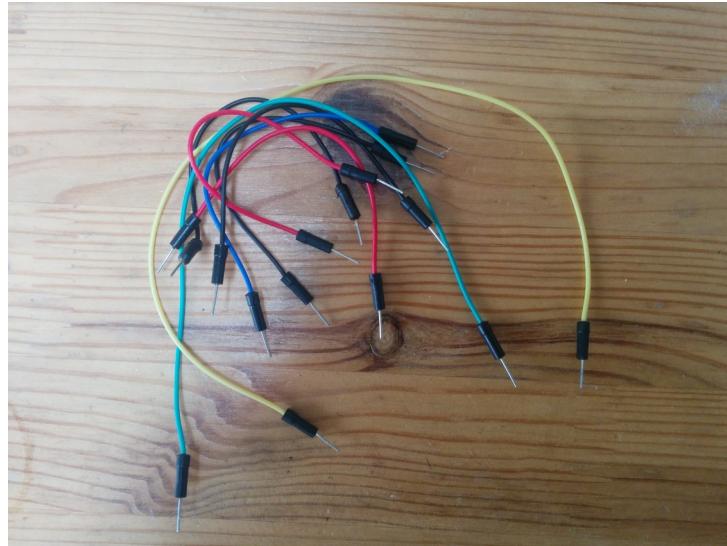


Figure 2 – An example of jumper wires

3. Once you have made a few 3 inch jumper wires it is time to wire up the breadboard. We will connect a 220 ohm resistor from the anode (long leg) of the LED to the positive supply rail on the breadboard, and connect the positive supply rail on the breadboard to PI_1 on the STM32F7 discovery board (the 19th pin from the left on the SHU baseboard kit). We will then wire up the cathode (short leg) on the LED to the negative supply rail on the breadboard and the negative supply on the breadboard to the ground pin on the STM32F7 discovery board. Figure 3 shows the completed and wired up circuit – if you have any questions about this circuit ask a member of staff.

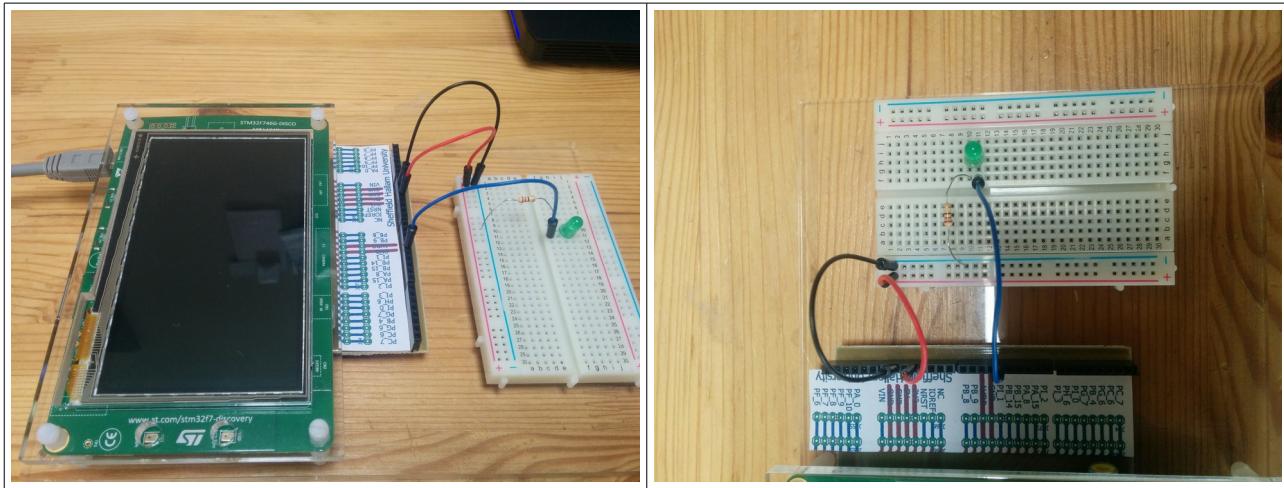


Figure 3 – The completed circuit

4. Now get your circuit checked!

5. We now need to write some code to flash this LED. Figure 4 shows the program logic in pseudo-code, and code listing 1 shows the complete program. A Keil uVision 5 project is available to download from GitHub.

```
Procedure BLINK:  
  
    while not ready:  
        initialise GPIO pin (PI_1)  
    end while  
  
    toggle LED pin (PI_1)  
    wait 1 second  
    repeat  
  
end Procedure BLINK
```

Figure 4 – Pseudo code for the “blink” program

We are going to use this breadboard method of prototyping throughout the rest of the laboratory exercises this year, and for the circuits you will design for your assignment.

Code listing 1:

```
/*
 * main.c
 *
 * this is the main blinky application with no external dependencies apart from
 * the stm32f7xx hal libraries
 *
 * author:     Dr. Alex Shenfield
 * date:       01/09/2017
 * purpose:    55-604481 embedded computer networks : lab 101
 */

// include the hal drivers
#include "stm32f7xx_hal.h"

// map the led to GPIO P11 (this is the inbuilt led by the reset button)
#define LED_PIN          GPIO_PIN_1
#define LED_PORT         GPIOI
#define LED_CLK_ENABLE() __GPIOI_CLK_ENABLE()

// declare our utility functions
void init_sysclk_216MHz(void);
void configure_gpio(void);

// this is the main method
int main()
{
    // we need to initialise the hal library and set up the SystemCoreClock
    // properly
    HAL_Init();
    init_sysclk_216MHz();

    // initialise the gpio pins
    configure_gpio();

    // loop forever ...
    while(1)
    {
        // toggle the led on the gpio pin
        HAL_GPIO_TogglePin(LED_PORT, LED_PIN);

        // wait for 1 second
        HAL_Delay(1000);
    }
}
```

```

// CLOCK

// configure the stm32f7 discovery board to operate at 216MHz
void init_sysclk_216MHz(void)
{
    // set up initialisation structures for the oscillator and the clock
    RCC_OscInitTypeDef rcc_oscillator_config;
    RCC_ClkInitTypeDef rcc_clock_config;

    // call macros to configure the correct power settings on the chip
    __HAL_RCC_PWR_CLK_ENABLE();
    __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

    // configure the oscillator and phase-locked loops so the chip runs at 168MHz
    rcc_oscillator_config.OscillatorType      = RCC_OSCILLATORTYPE_HSE;
    rcc_oscillator_config.HSEState           = RCC_HSE_ON;
    rcc_oscillator_config.HSISState          = RCC_HSI_OFF;
    rcc_oscillator_config.PLL.PLLState       = RCC_PLL_ON;
    rcc_oscillator_config.PLL.PLLSource     = RCC_PLLSOURCE_HSE;
    rcc_oscillator_config.PLL.PLLM          = 25;
    rcc_oscillator_config.PLL.PLLN          = 432;
    rcc_oscillator_config.PLL.PLLP          = RCC_PLLP_DIV2;
    rcc_oscillator_config.PLL.PLLQ          = 9;
    HAL_RCC_OscConfig(&rcc_oscillator_config);

    // activate the OverDrive to reach the 216 MHz frequency
    HAL_PWREx_EnableOverDrive();

    // configure the appropriate clock dividers
    rcc_clock_config.ClockType              = RCC_CLOCKTYPE_HCLK
                                              | RCC_CLOCKTYPE_SYSCLK
                                              | RCC_CLOCKTYPE_PCLK1
                                              | RCC_CLOCKTYPE_PCLK2;
    rcc_clock_config.SYSCLKSource          = RCC_SYSCLKSOURCE_PLLCLK;
    rcc_clock_config.AHBCLKDivider        = RCC_SYSCLK_DIV1;
    rcc_clock_config.APB1CLKDivider       = RCC_HCLK_DIV4;
    rcc_clock_config.APB2CLKDivider       = RCC_HCLK_DIV2;
    HAL_RCC_ClockConfig(&rcc_clock_config, FLASH_LATENCY_7);

}

// HARDWARE

// configure the gpio
void configure_gpio()
{
    // declare a gpio typedef structure that basically contains all the gpio
    // settings and properties you can use
    GPIO_InitTypeDef gpio_init_structure;

    // enable the clock for the led
    LED_CLK_ENABLE();

    // configure the gpio output on the led pin
    gpio_init_structure.Pin    = LED_PIN;
    gpio_init_structure.Mode   = GPIO_MODE_OUTPUT_PP;
    gpio_init_structure.Pull   = GPIO_PULLUP;
    gpio_init_structure.Speed = GPIO_SPEED_FAST;

    // complete initialisation
    HAL_GPIO_Init(LED_PORT, &gpio_init_structure);
    HAL_GPIO_WritePin(LED_PORT, LED_PIN, GPIO_PIN_RESET);
}

```

Task 2

Whilst the code we wrote in task 1 above works fine, we can see that some functions (such as the clock initialisation function to set up the system clock) we are going to be using in pretty much every project we write! So, to aid re-usability and efficiency, I have abstracted some of these commonly used functions into a board support package for some of the kit we have here at Sheffield Hallam University.

For the simple “blink” example in task 1 the use of libraries doesn't make a huge difference (as you can see from the provided project and the code in code listing 2, below) but, as code size and complexity increases, the use of separate source files and libraries to segregate functionality becomes very useful in creating readable and maintainable code (as we'll see in future weeks).

Code listing 2:

```
/*
 * main.c
 *
 * this is the main blinky application (this code is dependent on the
 * extra shu libraries such as the pinmappings list and the clock configuration
 * library)
 *
 * author:     Dr. Alex Shenfield
 * date:       01/09/2017
 * purpose:    55-604481 embedded computer networks : lab 101
 */

// include the hal drivers
#include "stm32f7xx_hal.h"

// include the shu bsp libraries for the stm32f7 discovery board
#include "pinmappings.h"
#include "clock.h"
#include "gpio.h"

// map the led to GPIO P11 (again, this is the inbuilt led)
gpio_pin_t led = {PI_1, GPIOI, GPIO_PIN_1};

// this is the main method
int main()
{
    // we need to initialise the hal library and set up the SystemCoreClock
    // properly
    HAL_Init();
    init_sysclk_216MHz();

    // initialise the gpio pins
    init_gpio(led, OUTPUT);

    // loop forever ...
    while(1)
    {
        // toggle the led on the gpio pin
        toggle_gpio(led);

        // wait for 1 second
        HAL_Delay(1000);
    }
}
```

The only real difference between this code and the code from task 1 is that the commonly used functions (such as initialising the clock and initialising and using the gpio pins) have been abstracted out into libraries (so we can reuse them easily across multiple projects). In the Keil uVision 5 project (available from GitHub), these are contained in the **stm32f7_discovery_bsp_shu_kit** group. The code below illustrates this point:

```
// this is the main method
int main()
{
    // we need to initialise the hal library and set up the SystemCoreClock
    // properly
    HAL_Init();
    init_sysclk_216MHz();

    // initialise the gpio pins
    init_gpio(led, OUTPUT);

    // loop forever ...
    while(1)
    {
        // toggle the led on the gpio pin
        toggle_gpio(led);

        // wait for 1 second
        HAL_Delay(1000);
    }
}
```

Now contained in a separate “clock” library to avoid repetition

Now contained in a separate “gpio” library to avoid repetition

Task 3

In this section we are going to experiment with programming the STM32F7 discovery board to produce different effects with LEDs. First create the circuit shown in Figure 5.

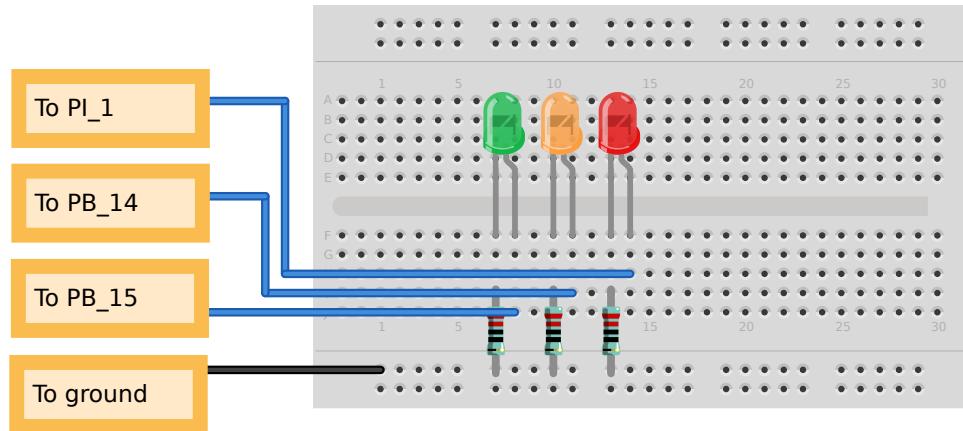


Figure 5 – Controllable LEDs

- a) Now adapt the uVision project from Task 2 to replicate the sequence of a single set of traffic lights. Make sure you tweak the timings of the LEDs so that the delays are as realistic as possible.
- b) As you might have realised whilst creating the traffic light controller in part a), traffic lights usually come in pairs! Write another program (again by adapting the existing project) that acts as a traffic light controller for a pair of lights (as in Figure 6).



Figure 6 – Traffic lights at a two way junction

A table showing the sequence of a sample set of traffic lights is shown below:

Traffic lights 1	Traffic lights 2
Red	Green
Red	Amber
Red	Red
Green	Red
Amber	Red
Red	Red

- c) Now try and create a single LED dice (see Figure 7).

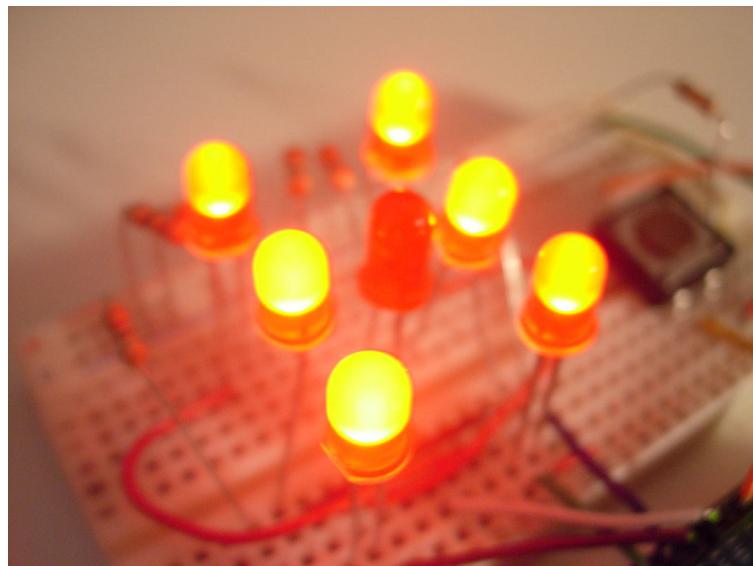


Figure 7– An LED dice

Within the SHU board support package provided, there is a set of functions to deal with the initialisation of the random number generator as well as generate the random numbers. After including the “random_numbers.h” header file and initialising the random number generator, you can use the code:

```
uint32_t rnd = (get_random_int() % 6) + 1;
```

to generate the random roll of the dice.

Task 4

In this task we are going to use a simple pushbutton switch to control an LED.

Firstly create the circuit shown in Figure 8, below. This wires up the two lower rails of the breadboard to the +5V and ground pins on the STM32F7 discovery board (+3.3V to the lower rail and ground to the top rail) allowing us to easily access the +3.3V supply and ground. The pushbutton switch then has one leg connected to +3.3V and the other leg connected to ground via a pull-down resistor (in this case a 10K Ohm one). The other side of this leg is connected to GPIO pin PA_8 on our STM32F7 discovery board. We then connect GPIO pin PI_1 to the LED.

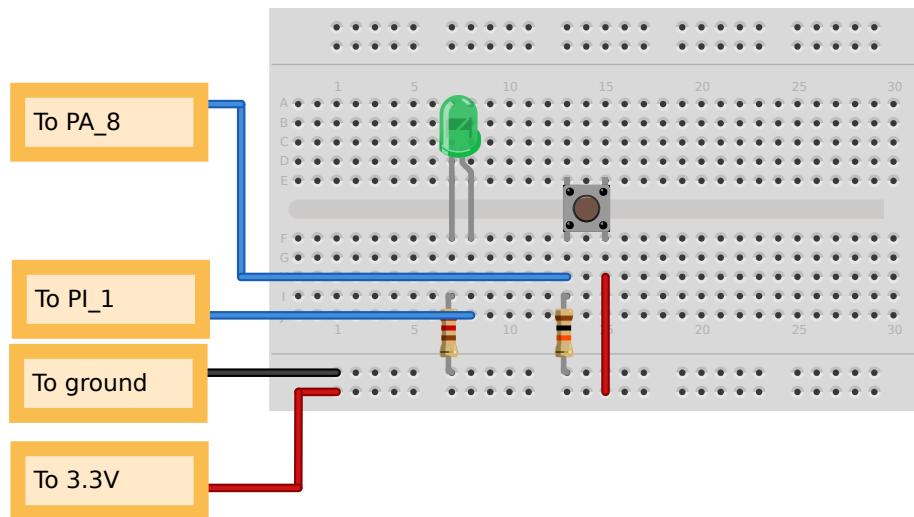


Figure 8 – The switch circuit

What is a “pull-down” resistor?

We could also have a “pull-up” resistor. What is the difference?

Why do we need “pull-up” or “pull-down” resistors?

Now we wish to write some code to control this circuit. When the pushbutton is held down we want the LED to turn on, and then when the pushbutton is released we want the LED to turn off again. See code listing 3 for a simple program to accomplish this².

² Note that a Keil uVision 5 project for this exercise is available to download from GitHub.

Code listing 3:

```

/*
 * main.c
 *
 * this is the main push button application (this code is dependent on the
 * extra shu libraries such as the pinmappings list and the clock configuration
 * library)
 *
 * author:     Dr. Alex Shenfield
 * date:       01/09/2017
 * purpose:   55-604481 embedded computer networks : lab 101
 */

// include the hal drivers
#include "stm32f7xx_hal.h"

// include the shu bsp libraries for the stm32f7 discovery board
#include "pinmappings.h"
#include "clock.h"
#include "gpio.h"

// map the led to GPIO PI_1 (the inbuilt led) and the push button to PI_11
// (the user button)
gpio_pin_t led = {PI_1, GPIOI, GPIO_PIN_1};
gpio_pin_t pbl = {PI_11, GPIOI, GPIO_PIN_11};

// this is the main method
int main()
{
    // we need to initialise the hal library and set up the SystemCoreClock
    // properly
    HAL_Init();
    init_sysclk_216MHz();

    // initialise the gpio pins
    init_gpio(led, OUTPUT);
    init_gpio(pbl, INPUT);

    // loop forever ...
    while(1)
    {
        // if the button is pressed ...
        if(read_gpio(pbl))
        {
            // turn the led on on the gpio pin
            write_gpio(led, HIGH);
        }
        else
        {
            // turn the led off on the gpio pin
            write_gpio(led, LOW);
        }
    }
}

```

Enter this program and make sure everything works. If there are any problems contact a member of staff.

Now you are going to make some modifications to the program to add functionality:

1. Alter the above program to “latch” the button state (i.e. pressing the button turns the LED on, pressing it again turns the LED off).
2. You may have noticed that, in your “latched” button state program, the LED sometimes doesn't behave as expected – this is because the mechanical contacts on the switch may “bounce”, triggering the latch in quick succession. You should add “debouncing” to your program (which means checking the button state twice in a short period of time to ensure that the button was actually pressed and it wasn't just the contacts “bouncing”). Ask a member of staff if this is unclear.
3. Wire up two other LEDs of different colours. Adapt your program to cycle through the LEDs when the button is pressed.

Task 5

In this task we are going to use a potentiometer to control the blinking rate of an LED. To do this you should wire up the circuit shown in Figure 9.

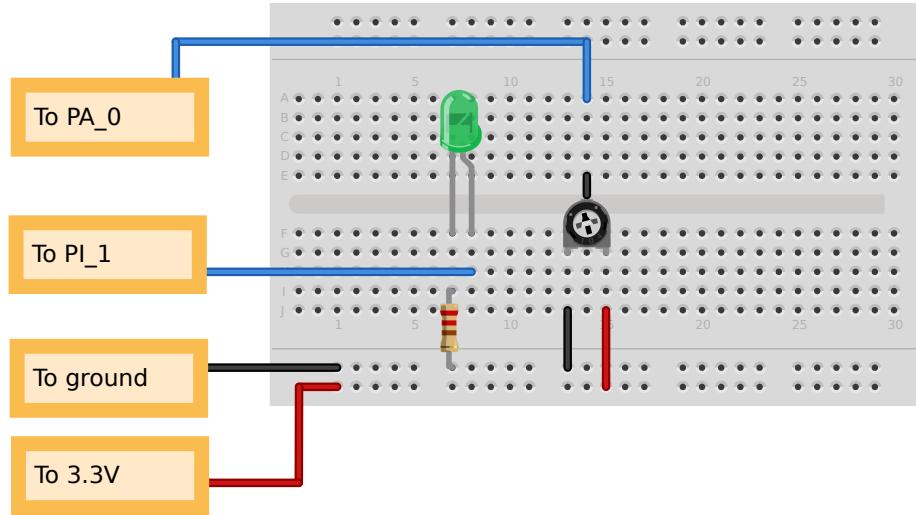


Figure 9 – The blink rate circuit

This circuit has the two rails of the breadboard³ wired up to the +3.3V and ground pins on the STM32F7 discovery board (+3.3V to the lower rail and ground to the upper rail) allowing us to easily access the +3.3v supply and ground. The potentiometer then has one leg connected to +3V another leg connected to ground, and the centre leg (the “wiper”) attached to GPIO pin PA_0. We then connect the LED to GPIO pin PI_1.

We then need to write our program to control the LED. First we have to read the value of the potentiometer via the ADC on the STM32F7 discovery board and then use this value to control the blink rate of the LED. Figure 10 shows the program logic for this example in pseudo-code. Note that in this example we are also going to send the value of the potentiometer to a terminal via the UART (the virtual COM port on the USB port in this case). You will find this functionality extremely helpful in debugging programs of even moderate complexity!

³ Note that the breadboards on the current version of the discovery kit are slightly different to this image and so you will have to adjust the circuit accordingly

```
Procedure ADJUSTABLE_BLINK:  
  
    while not ready:  
        initialise UART 1  
        initialise ADC pin (PA_0)  
        initialise GPIO pin (PI_1)  
    end while  
  
    toggle LED pin (PI_1)  
    delay = read POT pin (PA_0)  
    print "delay" value to the terminal  
    wait "delay" seconds  
    repeat  
  
end Procedure ADJUSTABLE_BLINK
```

Figure 10 – Pseudo code for the “adjustable_blink” program

As you can see, the logic for this program is fairly straightforward; however, the STM32F7 chip is complex and there are many different options available for configuring the ADC. We are only interested in a small subset of these, so I have provided a set of library functions that simplify the configuration process for the ADC. Code listing 4 shows the completed program, and Code listings 5 and 6 show the code for configuring the UART and printing characters and strings to the terminal

A Keil uVision 5 project (containing all the appropriate libraries) is available to download from GitHub.

Code listing 4:

```
/*
 * main.c
 *
 * this is the main analog blink application (this code is dependent on the
 * extra shu libraries such as the pinmappings list, the clock configuration
 * library, the gpio library, and the adc library)
 *
 * author:    Dr. Alex Shenfield
 * date:      01/09/2017
 * purpose:   55-604481 embedded computer networks : lab 101
 */

// include the hal drivers
#include "stm32f7xx_hal.h"

// include the shu bsp libraries for the stm32f7 discovery board
#include "pinmappings.h"
#include "clock.h"
#include "adc.h"
#include "gpio.h"

// include the serial configuration files
#include "serial.h"

// map the led to GPIO P11 and the potentiometer to PA0
gpio_pin_t led = {PI_1, GPIOI, GPIO_PIN_1};
gpio_pin_t pot = {PA_0, GPIOA, GPIO_PIN_0};
```

```
// this is the main method
int main()
{
    // we need to initialise the hal library and set up the SystemCoreClock
    // properly
    HAL_Init();
    init_sysclk_216MHz();

    // initialise the uart, adc and gpio pins
    init_uart(9600);
    init_adc(pot);
    init_gpio(led, OUTPUT);

    // print an initial status message
    printf("we are alive!\r\n");

    // loop forever ...
    while(1)
    {
        // toggle the led pin
        toggle_gpio(led);

        // read the potentiometer and echo that value to the terminal
        uint16_t adc_val = read_adc(pot);
        printf("potentiometer value = %d : delay time = %d ms\r\n",
               adc_val, (adc_val/2));

        // delay for the appropriate time
        HAL_Delay(adc_val / 2);
    }
}
```

Code listing 5:

```
/*
 * serial.h
 *
 * provide some implementation functions for our uart (uart 1 in this case).
 * in this case we are not implementing interrupts at all - just using the
 * uart to output status messages.
 *
 * the stm32f7 board allows us to send messages over the virtual com port
 * (i.e. the usb st-link connection).
 *
 * author:     Dr. Alex Shenfield
 * date:       01/09/2017
 * purpose:    55-604481 embedded computer networks : lab 101
 */

// include the basic headers for the hal drivers
#include "stm32f7xx_hal.h"

// include the standard c io library
#include <stdio.h>

// define the virtual com port gpio pins
#define VCP_RX_Pin          GPIO_PIN_7
#define VCP_RX_GPIO_Port     GPIOB
#define VCP_TX_Pin          GPIO_PIN_9
#define VCP_TX_GPIO_Port     GPIOA

// declare the serial initialisation method
void init_uart(uint32_t baud_rate);
int serial_write(int ch);
```

Code listing 6:

```

/*
 * serial.c
 *
 * provide some implementation functions for our uart (uart 1 in this case).
 * in this case we are not implementing interrupts at all - just using the
 * uart to output status messages.
 *
 * the stm32f7 board allows us to send messages over the virtual com port
 * (i.e. the usb st-link connection).
 *
 * author:     Dr. Alex Shenfield
 * date:       01/09/2017
 * purpose:    55-604481 embedded computer networks : lab 101
 */

// include the relevant header files
#include "serial.h"

// retarget printf for ease of debugging
#define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)

// FUNCTION PROTOTYPES

// configure the hardware part of the uart (e.g. gpio pins and clocks)
void configure_hw_uart(void);

// GLOBAL DECLARATIONS (FOR THIS MODULE)

// uart handle object
UART_HandleTypeDef uart_handle;

// METHODS

// uart initialisation
void init_uart(uint32_t baud_rate)
{
    // configure the hardware
    configure_hw_uart();

    // set up the uart structure - uart 1 xxxx-8N1
    uart_handle.Instance      = USART1;

    uart_handle.Init.BaudRate      = baud_rate;
    uart_handle.Init.WordLength    = UART_WORDLENGTH_8B;
    uart_handle.Init.StopBits     = UART_STOPBITS_1;
    uart_handle.Init.Parity       = UART_PARITY_NONE;
    uart_handle.Init.HwFlowCtl    = UART_HWCONTROL_NONE;
    uart_handle.Init.Mode        = UART_MODE_TX_RX;

    // initialise the uart
    HAL_UART_Init(&uart_handle);
}

```

```

// for the gpio set up stuff we need to look at the alternate function mapping
// table in DM00037051.pdf - pg. 61 onwards

// configure the hardware for the uart (clocks, gpio pins, etc.)
void configure_hardware_uart(void)
{
    // configure the usart 1 clocks (both uart and gpio clocks)
    __HAL_RCC_USART1_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();

    // configure the gpio pins

    // declare a gpio typedef structure that basically contains all the gpio
    // settings and properties you can use
    GPIO_InitTypeDef gpio_init_structure;

    // configure the gpio output on the uart 1 pins (TX=PA9 and RX=PB7)
    gpio_init_structure.Pin      = VCP_TX_Pin;
    gpio_init_structure.Mode    = GPIO_MODE_AF_PP;
    gpio_init_structure.Pull    = GPIO_NOPULL;
    gpio_init_structure.Speed   = GPIO_SPEED_FREQ_LOW;
    gpio_init_structure.Alternate = GPIO_AF7_USART1;
    HAL_GPIO_Init(VCP_TX_GPIO_Port, &gpio_init_structure);

    gpio_init_structure.Pin      = VCP_RX_Pin;
    gpio_init_structure.Mode    = GPIO_MODE_AF_PP;
    gpio_init_structure.Pull    = GPIO_NOPULL;
    gpio_init_structure.Speed   = GPIO_SPEED_FREQ_LOW;
    gpio_init_structure.Alternate = GPIO_AF7_USART1;
    HAL_GPIO_Init(VCP_RX_GPIO_Port, &gpio_init_structure);
}

// LOW LEVEL IO

// put a character onto "the wire"
int serial_write(int ch)
{
    // use the hal uart transmit function to send a character
    HAL_UART_Transmit(&uart_handle, (uint8_t *)&ch, 1, 0xFFFF);

    // return the character
    return ch;
}

// implementation of putchar to retarget printf (we use putchar_prototype here
// because it is easy to make this portable - e.g. by adding conditionals to
// the #define statement)
PUTCHAR_PROTOTYPE
{
    // return the character
    return serial_write(ch);
}

```

Test this program and make sure everything works. If there are any problems contact a member of staff.

Task 6

Create the simple button game circuit shown in Figure 11.

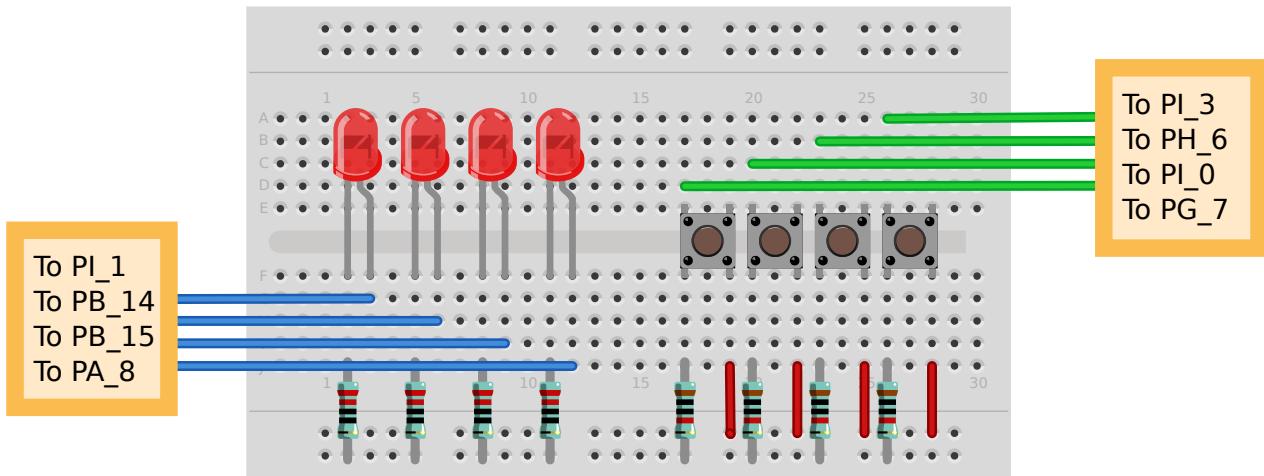


Figure 11 – The button game circuit

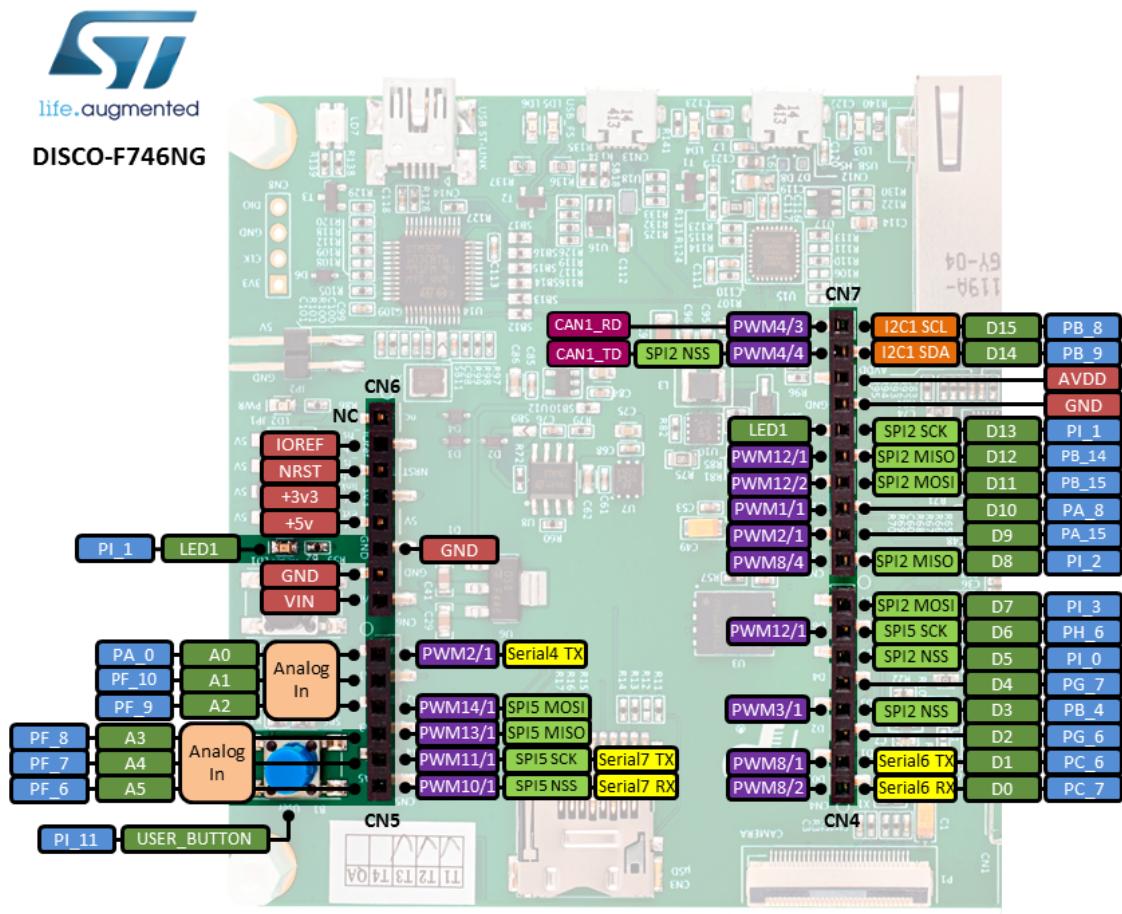
In this simple button game, you have to hit the corresponding button to the randomly selected LED before the LED goes out. You start with 9 lives and you lose a life every time you press the wrong button or fail to press the button before the LED goes out. A partial program (working for two buttons and two LEDs) is available from GitHub.

Once you have tested this program and verified it works OK you should complete the following tasks:

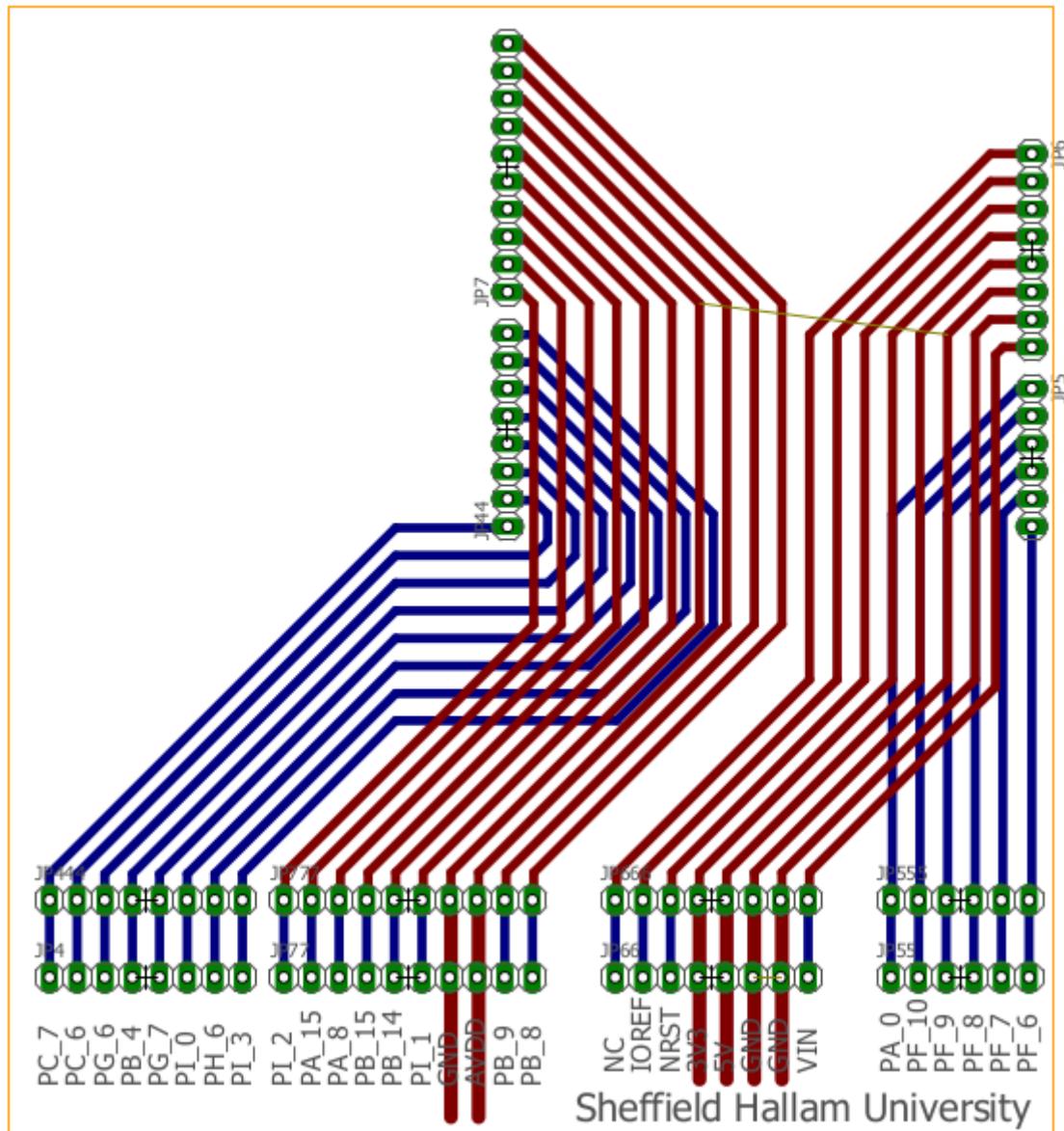
1. Extend this program to work with all 4 LEDs and all 4 buttons shown in the circuit in figure 6.
2. Make the 4 LEDs flash in some sequence when the player loses.
3. Allow the player to restart the game after they lose.
4. Improve the program by making the difficulty (i.e. the speed the LEDs blink at) adjustable using a potentiometer.
5. Improve the program by making it harder as the player scores more points.
6. Using the same circuit and adapting the code, implement a simple version of “Simon Says”⁴

⁴ Simon Says is a game where the player is shown an increasingly complex sequence of “instructions” (i.e. the order the LEDs flash in) and they have to replicate those instructions without making a mistake. More information can be found at: http://en.wikipedia.org/wiki/Simon_%28game%29

Appendix A – The STM32F7 discovery board schematic



STM32F7 discovery board pin outs

Appendix B – The STM32F7 discovery board SHU base board schematic

SHU breakout board for the STM32F7 discovery