1. Introduction

The purpose of this lab is to introduce you to infrared (IR) communication and get familiar with the device driver modules to control other peripherals on the UCFK4. Some sections of this lab will need to be done in pairs so make sure you have a partner to work with - ideally, your project partner.

2. Getting started

2.1. Boot Linux mint and log in.

2.2. Open a bash shell and navigate to the ucfk4 directory that you worked in last week

2.3. Change to, and then list the directory contents of your /labs directory:

```
ls -a.
```

The directories you will work with this week are:

```
/lab3-ex1    /lab3-ex2    /lab3-ex3    /lab3-ex4
```

3. Letters

You are now going to write a program that displays a letter on the LED matrix. To do this we'll have a look at some scrolling messages. We will do this with the tinygl (tiny graphics library) module. This module lets you draw on the LED matrix as you might on a computer screen by calling functions to display lines, shapes, and text[1]. For this lab, we will only use the text functionality.

3.1. Open /lab3-ex1/lab3-ex1.c. First we need to initialise the tinygl module. The following lines will do this:

```
tinygl_init (LOOP_RATE);

tinygl_font_set (&font5x7_1);

tinygl_text_speed_set (MESSAGE_RATE);
```

3.2. The LOOP_RATE is the same as the pacer rate and needs to be passed to the tinygl module so it knows how often its update function is to be called. This is so it can accurately set the speed at which it scrolls text. MESSAGE_RATE will be the speed of the displayed message and is given in characters per 10 seconds.

3.3. Now we need to give tinygl a character to display. You can choose any character here, an uppercase letter will be best. Use the tinygl_text() function and pass it a single character string. Note that this must be a string because tinygl also needs the end of string character ('\0') in order to know when the message ends.

3.4. You will also need to call the tinygl update function in the while loop.

3.5. Compile and program your code. You should now see your chosen character displayed on the LED matrix.

[1] If you are interested in the complete interface of this module, have a look at the utils/tinygl.h file.

3.6. Now we will give tinygl a longer message and make it smoothly scroll the message across the LED matrix. You need to add the following line to the initialisation section of your code.

```
tinygl_text_mode_set (TINYGL_TEXT_MODE_SCROLL);
```

3.7. This changes tinygl from stepping through a message, character by character, to smoothly scrolling it.

3.8. Now we will give tinygl a message to display. Change your tinygl_text() line to pass a longer message, say a couple of words. "Hello world" is popular!

3.9. Recompile and program. You should now see your message scrolling across the display.

4. The navigation switch

The navigational switch lets you perform five different button actions and is convenient for indicating motion, such as cycling through a list. You should recall however that the navigation switch is connected to some of the LED matrix ports. We perform multiplexing on the IO pins to give the appearance of concurrent operation. We will use it to change the letter we display on the LED matrix.

4.1. Open the /lab3-ex2/lab3-ex2.c file and you will see some code to place a character on the LED matrix. Modify this file using the navswitch driver to make the characters change with a north and south press. The navswitch driver operates by periodically updating its internal state with the navswitch_update() function. The navswitch function declarations can be found in /drivers/navswitch.h.

4.2. Compile and program. You should be able to see the character change when you press the navigational switch up or down. Have a look at the available characters by cycling through them all. What happens when you go past the end of the available characters?

5. Infrared transmission

Now we will incorporate some IR communications into our programs. We will send a character using the infrared serial communications driver[2].

5.1. Open the /lab3-ex3/lab3-ex3.c file and you will see a program that allows you to change the character being displayed on the screen. We will be sending the displayed character. Have a look at /drivers/avr/ir_uart.h at how to initialize the IR driver.

5.2. We need to create a button press event for selecting when to send the current character. Use the middle press (down) of the navigational switch. This is the NAVSWITCH_PUSH event. Next, use the ir_uart_putc() function to transmit the current character.

5.3. Compile and program you code. You should now be able to transmit the character that is currently displayed on your LED matrix. However we need a receiver. Get your partner to load the program in the /labs/receiver/ directory onto his/her board. Now you should be able to point your IR transmitter to your

[2] The IR driver API can be found in drivers/avr/ir_uart.h**.**

partner's IR receiver and have the character that you sent appear on the other board. Note that fluorescent lights emit IR radiation, and can cause interference. You may want to turn the lights off when performing this IR communication.

6. Infrared reception

Now we want to include receiving in our own program. This is not too difficult, instead of a button press event we have a character received event. Once a character has been received, the function ir_uart_read_ready_p() will return true[3] and we can then read the character with the ir_uart_getc() function.

6.1. Edit your /lab3-ex3/lab3-ex3.c file and add code to detect a received event and read the received character. Have a look at the demo code in the driver header file for tips on how to do this. The currently displayed character will need to be updated to this new character.

6.2. Compile and program your code. Your partner will need to program the same, or similarly functioning code onto his/her board also. You should be able to select a character and send it to your partner's board, who will then change the character and send it back. You now have two way text communication[4]!

7. Timing multiple tasks

There are many ways that you can synchronise task execution within a program. One way is to construct a scheduler that will call tasks at different rates (see utils/task.h). This task scheduler is used in the apps/stopwatch1/stopwatch1.c program, which implements a simple stopwatch. Your task for this section is to re-implement this program to use the pacer architecture instead of the scheduler.

7.1. Open the apps/stopwatch1/stopwatch1.c file and get a feel of how the program works[5]. Look at the utils/task.h file to see what the functions do. Program this onto your board to see it in action.

7.2. Open /lab3-ex4/lab3-ex4.c. You will see the outline of a pacer program. Implement the functionality of the stopwatch program here without using the scheduler. You will need to copy the functions across from stopwatch1.c to lab3-ex4.c, but also change the __unused__ void *data in the function declaration to void.

7.3. The three tasks need to be executed at different frequencies, so you will need separate variables in your pacer loop to keep track of each task.

7.4. Compile and program your code. If you are successful, the functionality will be the same as the original stopwatch.

[3] A common naming convention for a predicate function, i.e., a function that returns a Boolean value, is to append an _p suffix.
[4] Although somewhat unreliable communication!
[5] This program uses pointers to functions!