

## Lab Report

# Lab Assignment 4, 2023

C-programming with interrupts



***Authors:***

Robin G. Hansen

Michael Daun.

***Course:***

Computer Technology 1

1DT301

## Task 1: Binary counter

**Connect four LEDs in a row to make a binary counter. The counter should count from 0000 to 1111.**

**Connect one button to GP5 and one button to GP6 with the following functions:**

- **Let the button on GP5 increment the counter (increase one step). If increase button is pressed when counter value is 15, nothing should happen!**
- **Let the button on GP6 decrement the counter (decrease one step). If decrease button is pressed when counter value is 0, nothing should happen!**

**Let the counter start at value 0. You must use interrupts to handle the inputs from the buttons! There will probably be problems with bouncing buttons (one button press counts as many) but you can ignore this problem.**

---

### *Solution*

The idea behind our solution was to create a single function for the button callback, which checks whether the increment or decrement button has been pressed. If the increment button is pressed, the counter should increase by one, and it should decrease with the other button. We accomplished this mainly by utilizing functions we had worked with previously, such as `'gpio_init'` and `'gpio_set_dir'`. The functions related to the hardware interrupt are provided by `'gpio_set_irq_enabled_with_callback'`, which is a part of the [Pico SDK](#). This function takes four parameters. The first parameter specifies the pin associated with the interrupt, and we used variables as names for our `'increment_button'` and `'decrement_button'`. The second parameter defines the type of event that will trigger the interrupt. In our case, we defined the interrupt to trigger during the falling edge of the button press. The third parameter is used to enable the interrupt by setting a boolean value. The fourth parameter is the actual callback function that defines what should happen when the button is pressed.

Our function `'button_callback'` takes two parameters. The first parameter is the pin that triggered the callback, and the second parameter is the type of event that triggered it. Inside the function, we have a conditional statement at the beginning that checks whether it's the increment or decrement button that has been pressed. If it's the increment button, the counter should increase as long as the value isn't above 15, which is our maximum value for the 4-bit counter. Similarly, the decrement button should decrease the value by 1, as long as the counter isn't less than 0.

Since we have had some issues with contact bounces we added a print statement to make sure that our counter value in the code matches the value of led lamps. The **'button\_callback'** finally calls the **'binary\_counter'** function that should light the leds corresponding to the current counter value.

The **'binary\_counter'** consists of a series of conditional statements within the **'gpio\_put'** function, determining whether the LED should be set to a high or low value. For example, when the counter is at 9, both outermost lamps should be lit.

The main section contains the function **'tight\_loop\_contents'** in a while loop that keeps our microcontroller in an idle state awaiting interrupts from the buttons.

## *Code*

```
#include <stdio.h>
#include "pico/stdlib.h"
#include "hardware/gpio.h"

#define increment_button 5
#define decrement_button 6
#define LED_1 1 // LSB
#define LED_2 2
#define LED_3 3
#define LED_4 4 // MSB

int counter = 0;

void binary_counter() {
    gpio_put(LED_1, ((counter % 2 == 0) ? 0 : 1));
    gpio_put(LED_2, ((counter > 1 && counter < 4) || (counter > 5 &&
counter < 8) || (counter > 9 && counter < 12) || (counter > 13) ? 1
: 0));
```

```
gpio_put(LED_3, ((counter > 3 && counter < 8) || (counter > 11)) ?
1 : 0);

gpio_put(LED_4, (counter < 8) ? 0 : 1);
}

void button_callback(uint gpio, uint32_t events) {
if (gpio == increment_button && counter < 15) {
counter++;
} else if (gpio == decrement_button && counter > 0) {
counter--;
}
printf("Current counter value: %d \n", counter);
binary_counter();
}

int main() {
stdio_init_all();

gpio_init(increment_button);
gpio_set_dir(increment_button, 0);

gpio_init(decrement_button);
gpio_set_dir(decrement_button, 0);

gpio_init(LED_1);
gpio_set_dir(LED_1, 1);

gpio_init(LED_2);
gpio_set_dir(LED_2, 1);

gpio_init(LED_3);
```

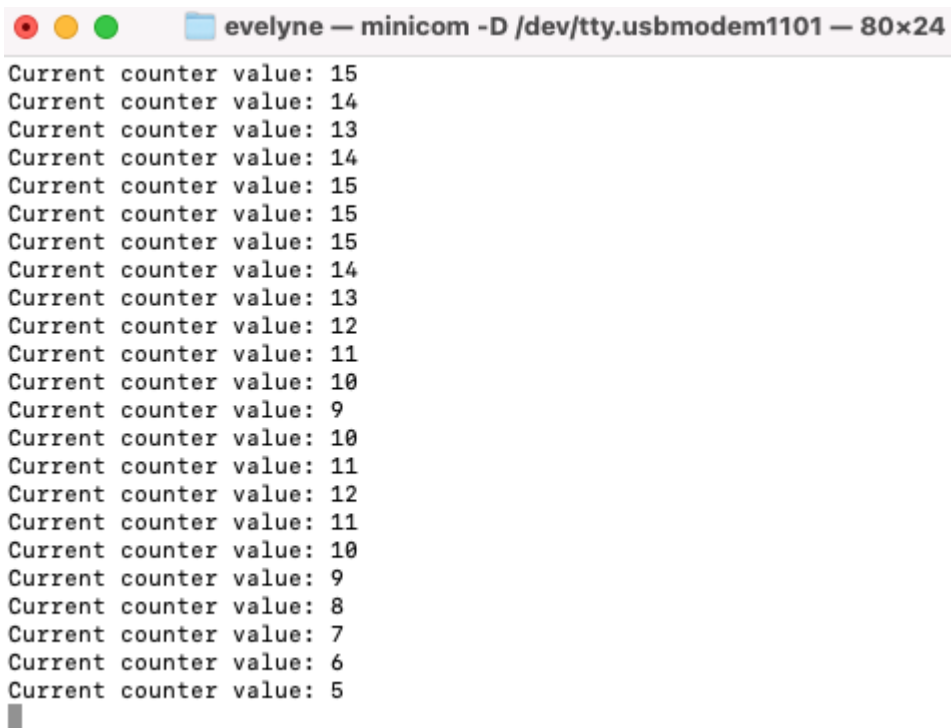
```
gpio_set_dir(LED_3, 1);

gpio_init(LED_4);
gpio_set_dir(LED_4, 1);

gpio_set_irq_enabled_with_callback(increment_button,
GPIO_IRQ_EDGE_FALL, true, &button_callback);
gpio_set_irq_enabled_with_callback(decrement_button,
GPIO_IRQ_EDGE_FALL, true, &button_callback);
while(1) {
tight_loop_contents();
}

}
```

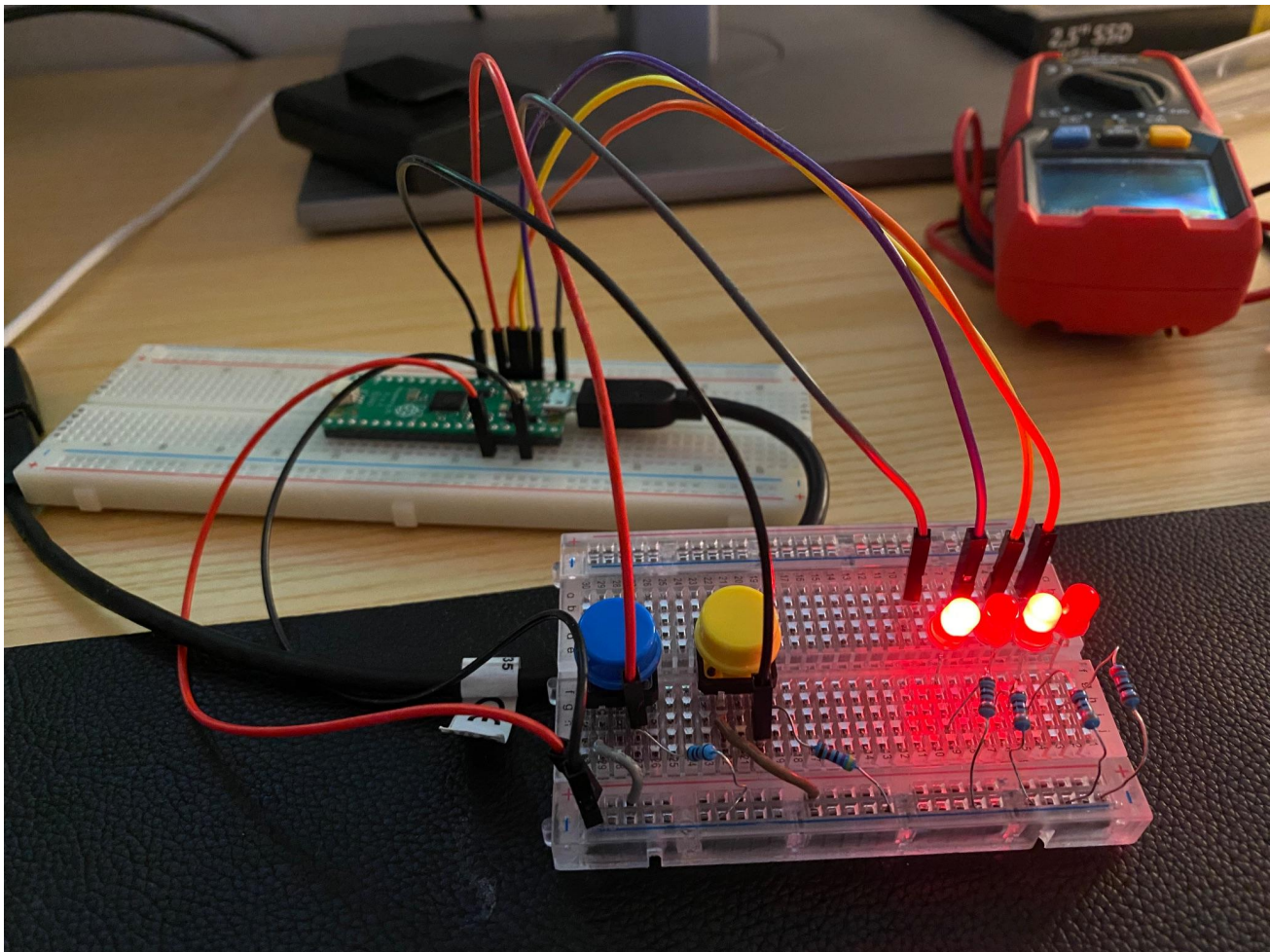
## Screenshot of terminal output using Minicom



```
evelyne — minicom -D /dev/tty.usbmodem1101 — 80x24
Current counter value: 15
Current counter value: 14
Current counter value: 13
Current counter value: 14
Current counter value: 15
Current counter value: 15
Current counter value: 15
Current counter value: 14
Current counter value: 13
Current counter value: 12
Current counter value: 11
Current counter value: 10
Current counter value: 9
Current counter value: 10
Current counter value: 11
Current counter value: 12
Current counter value: 11
Current counter value: 10
Current counter value: 9
Current counter value: 8
Current counter value: 7
Current counter value: 6
Current counter value: 5
```

## *Hardware setup*

In the image below we have the circuit we used for this task. We used breadboards to connect the pushbuttons and the led with our Raspberry Pico. The power rail and ground rail are connected to their respective pins on the Pico. The led lamps are connected in series with protective 330  $\Omega$  resistors to ensure that there is a limited current flow. The pushbuttons are connected with pull down resistors valued at 47k  $\Omega$  each. We used the pull down resistor to ensure that the idle voltage for the Picos input remains at 0 V when the buttons are not actively in use.





## Task 2: Binary counter with reset button

Use the same counter setup as in the previous task, but this time, let the counter increase automatically using a timer interrupt. Also, connect a button to GP0 to reset the counter. You don't need to use the buttons at GP5 and GP6 in this task.

---

### *Solution*

There weren't many changes between Task 1 and Task 2. The main differences were that we no longer needed two buttons, and we had to implement a new callback function for the timer interrupt called "**timer\_callback**". This new function would trigger the "**led\_counters**" function every second and increment our counter by 1 unless the counter had already reached 15. If the counter has reached 15, it will just continuously keep it at 15.

We also modified the functions for increment and decrement to have a single purpose: resetting the counter to 0 whenever we pressed the button. Additionally, we made sure to call the "**led\_counters**" function again to ensure that it updated the values for each LED once the counter was reset to 0.

```
int64_t timer_callback(alarm_id_t id, void *unused) {
    led_counters();
    if (counter == 15) {
        counter = counter;
    } else {
        counter++;
    }
    return 1000 * 1000;
}

void reset_counter() {
    if (gpio_get(RESET_BUTTON) == 1) {
        counter = 0;
        led_counters();
    }
}
```

In order to execute our new functions we used the “**add\_alarm\_in\_us**” function to set the delay for how often we want it to trigger and which callback function to use, in our case the “**timer\_callback**”.

```
add_alarm_in_us(1000 * 1000, timer_callback, NULL, true);
gpio_set_irq_enabled_with_callback(RESET_BUTTON, GPIO_IRQ_EDGE_FALL, true, &reset_counter);

while (1) {
    tight_loop_contents();
}

return 0;
}
```

“The **gpio\_set\_irq\_enabled\_with\_callback**” remains the same as before but instead of having two of them one for each pin (increment button / decrement button) we only use one for the reset\_counter and call it RESET\_BUTTON.

To know the respective parameters and what to insert we used the [documentations](#) as reference along with the help from the lecture slides.



## Code

```
#include <stdio.h>
#include "pico/stdlib.h"
#include "hardware/gpio.h"
#include "hardware/timer.h"
#include "pico/time.h"
#include "hardware/irq.h"

#define RESET_BUTTON 0

#define LED_ONE 16 // LSB
#define LED_TWO 17
#define LED_THREE 19
#define LED_FOUR 20 // MSB

int counter = 0;

// Ternary operator
// condition ? expression_if_true : expression_if_false;

void led_counters() {
    gpio_put(LED_ONE, ((counter % 2 == 0) ? 0 : 1));
    gpio_put(LED_TWO, ((counter > 1 && counter < 4) || (counter > 5 && counter < 8) || (counter > 9 && counter < 12) || (counter > 13)) ? 1 : 0);
    gpio_put(LED_THREE, ((counter > 3 && counter < 8) || (counter > 11)) ? 1 : 0);
    gpio_put(LED_FOUR, (counter < 8) ? 0 : 1);
}

int64_t timer_callback(alarm_id_t id, void *unused) {
    led_counters();
    if (counter == 15) {
        counter = counter;
    } else {
        counter++;
    }
    return 1000 * 1000;
}
```

```
void reset_counter() {
    if (gpio_get(RESET_BUTTON) == 1) {
        counter = 0;
        led_counters();
    }
}

int main() {
    stdio_init_all();

    // Initialize LED pins as output.
    gpio_init(LED_ONE);
    gpio_set_dir(LED_ONE, 1);

    gpio_init(LED_TWO);
    gpio_set_dir(LED_TWO, 1);

    gpio_init(LED_THREE);
    gpio_set_dir(LED_THREE, 1);

    gpio_init(LED_FOUR);
    gpio_set_dir(LED_FOUR, 1);

    // Initialize reset button as input.
    gpio_init(RESET_BUTTON);
    gpio_set_dir(RESET_BUTTON, 0);

    add_alarm_in_us(1000 * 1000, timer_callback, NULL, true);
    gpio_set_irq_enabled_with_callback(RESET_BUTTON, GPIO_IRQ_EDGE_FALL, true, &reset_counter);

    while (1) {
        tight_loop_contents();
    }

    return 0;
}
```

## *Hardware setup*

For the hardware in task 2, we used a setup that was quite similar, but without the requirement of both decrement and increment buttons. Instead, we reduced it to just one button - the reset button. We had to use two smaller breadboards. We used 330-ohm resistors for the LEDs, and for the pull-down resistor, we used a 10k resistor since we didn't have any other options. We also had to use a Pico with the Wi-Fi module, but this did not affect either the code or the pinout configurations we used. All of the materials are our own.

