

LABOTARY A

Embedded System Design and Implementation – EEET2481



Group *Group 19*

Students *Tuan Phung – s3761489*

*Nghia Huu Le - s3654028*

*Trung Nguyen - s3633958*

Lecturer *Andrew Smith*

Table of Contents

[Introduction 4](#_Toc4773659)

[Exercise 1 - Toggling an LED 5](#_Toc4773660)

[Question 1 5](#_Toc4773661)

[Question 2 6](#_Toc4773662)

[Question 3 6](#_Toc4773663)

[Question 4 7](#_Toc4773664)

[Exercise 2 – Creating an LED Chaser 10](#_Toc4773665)

[Question 1 10](#_Toc4773666)

[Question 2 10](#_Toc4773667)

[Question 3 10](#_Toc4773668)

[Question 4 11](#_Toc4773669)

[Exercise 3 - Controlling the LED Brightness 12](#_Toc4773670)

[Question 1 12](#_Toc4773671)

[Question 2 12](#_Toc4773672)

[Question 3 14](#_Toc4773673)

[Source code 14](#_Toc4773674)

[Exercise 4 - Creating an Advanced LED Chaser 17](#_Toc4773675)

[Question 1 17](#_Toc4773676)

[Exercise 5: Controlling an RGB LED 18](#_Toc4773677)

[Indication of the complexity involved in 18](#_Toc4773678)

[Improvements that would enhance the quality of the LED's color being emitted 18](#_Toc4773679)

[Flowchart 19](#_Toc4773680)

[Conclusion 20](#_Toc4773681)

[Lab work and contribution 21](#_Toc4773682)

[References 22](#_Toc4773683)

[Figure 1. GPC12-GPC15 to control LED function. 5](#_Toc4773684)

[Figure 2. VirtualBench Waveform Exercise 1 9](#_Toc4773685)

[Figure 3. VirtualBench result for LED chaser 11](#_Toc4773686)

[Figure 4. VirtualBench result for LED at 50% duty cycle 12](#_Toc4773687)

[Figure 5. VirtualBench duty cycle at 10% 14](#_Toc4773688)

[Figure 6. VirtualBench when the LED increasing in brightness 15](#_Toc4773689)

[Figure 7. VirtualBench when the LED decreasing in brightness 16](#_Toc4773690)

[Figure 8. Flowchart of the algorithm for Exercise 5 19](#_Toc4773691)

[Table 1. PMD register map 6](#_Toc4773692)

[Table 2. LEDs Configuration 10](#_Toc4773693)

# Introduction

General purpose inputs and outputs exist on every embedded system and are used in many different applications for controlling external devices and receiving arbitrary data from the external world. In this laboratory task, the students are required to use on the knowledge has been gained through every lesson to understand how an LED can be operated with an embedded system by controlling the output of GPIO. This means that, the students will have a deep knowledge how an LED can be switched on and off and how the brightness of an LED can be controlled by implementing fundamental principles of an embedded system. While doing the laboratory, all the exercise must be programmed by using the Keil uVision IDE and combined with Nuvoton NUC140 microcontroller.

The main objectives of the lab are:

* Configure Clock Controller to use different clock sources and clock frequencies.
* Configure and control GPIO to output signals to NUC140 MCU external pins.

# Exercise 1 - Toggling an LED

In this exercise, we are required to toggle an LED at a predefined interval.

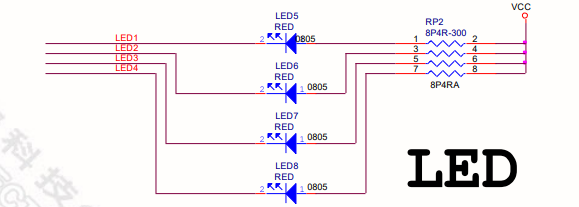
**

Figure 1. GPC12-GPC15 to control LED function.

## **Question 1**

*Open the Nu-LB-NUC140 user manual and go to page 14. Observe how the LED is connected to GPC12*

As we can see from the Figure 1, LED 5 is directly controlled by port C pin 12 of NUC 140 board. The board uses output sink current module to turn on and off LED 5.

***Justification***

According to the electrical schematic, the microcontroller is set to sinking the current from the external source – VCC which returns with the several advantages in terms of the embedded system design.

The microcontroller is responsible for supplying power for a variety of components. In order to solve the shortage of power, the microcontroller is built to sink the current or be provided power from an external source which has greater potential current source. To be more precise, in the above schematic, there is corresponding connection between 5V source, resistor, LEDs and microcontroller in series.

Thus, if the pin connected to a specific LED is pulled low (set the logic value to 0), it would case a potential difference between two nodes in the circuit. Hypothetically, this generates a current flowing from source to the appropriate LED and brightens the LED. In contrast, to turn off the LED, 1 should be set as the output for the microcontroller.

## **Question 2**

*Open the NUC140VE3CN technical reference manual and go to page 186 - GPIO Port X Mode Control Register*

A pin GPIO mode is controlled by the corresponding PMD register which is demonstrated in the map below

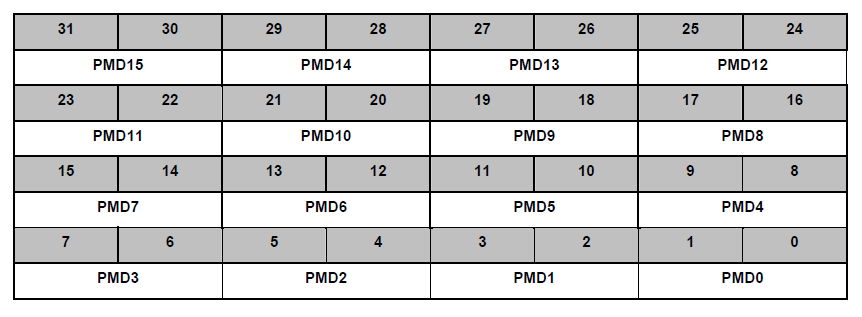


Table 1. PMD register map

As we want to set the mode of GPC12, we need to configure bit 24 and 25 in which 00 means input, 01 means output, 10 means open drain and 11 means quasi-bidirectional mode.

We want to change the mode to output, we need to set those two bits to 01.

## **Question 3**

Assuming that you have already setup a new project in uVision for this laboratory, write code which will configure GPC12 to be an appropriate output pin (Reminder, this code must be inside a function which is defined in a dedicated source file for exercise 1 and this function must be declared inside the associated header file)

Header file (*helper\_ex1.h*)

|  |
| --- |
| #ifndef \_\_HELPEREX1\_\_  #define \_\_HELPEREX1\_\_  void Set\_GPC12\_output();  void Turn\_on\_LED12();  void Turn\_off\_LED12();  void Toggle\_LED12();  #endif |

Source file (helper\_ex1.c)

|  |
| --- |
| #include "NUC100Series.h"  #include "helper\_ex1.h"  void Set\_GPC12\_output() {      PC->PMD &= ~(0x03) << 24;      PC->PMD |= (0x01) << 24;  }  void Turn\_on\_LED12() {      PC->DOUT &= ~(0x01) << 12; //ON = 0  }  void Turn\_off\_LED12() {      PC->DOUT |= (0x01) << 12; //OFF = 1  }  void Toggle\_LED12() {      PC->DOUT ^= (1 << 12); |

Code explanation

|  |
| --- |
| PC->PMD &= ~(0x03) << 24; |

This line of code represents a technique called masking which set the bit 25, 24 to 00.

|  |
| --- |
| PC->PMD |= (0x01) << 24; |

Then we set bit 25, 24 to 01 respectively by OR the current value with the register with the value 1 shift 24 times to the left.

## **Question 4**

In order to see the LED toggling, it is necessary to use a delay function, this function is called Delay\_s(param), where param is a number relating to how long the delay will be

* Define a macro for param (give it an appropriate name), and set the macro value to 500000

|  |
| --- |
| #define DELAY\_PARAM 500000 |

* Write code to turn on the LED connected to GPC12 by writing to the appropriate bit

We defined another function in “helper\_ex1.c” and “helper\_ex1.h”

|  |
| --- |
| void Turn\_on\_LED12() {      PC->DOUT &= ~(0x01) << 12; //ON = 0  } |

main.c

|  |
| --- |
| #include "NUC100Series.h"  #define DELAY\_PARAM 500000  #include "helper\_ex1.h"  #include "misc.h"  int main(void) {      Init\_board();      Turn\_on\_LED12();  } |

* Write additional code to toggle the LED at a fixed rate determined by the macro defined previously

We added a new function to toggle the LED called Toggle\_LED12.

|  |
| --- |
| void Toggle\_LED12() {      PC->DOUT ^= (1 << 12);  } |

^= is XOR manipulation logic, which manipulates with 1 and shifts 12 bits to the left. As we know the XOR logic can be used to toggle a bit

And in main.c file:

|  |
| --- |
| #include "NUC100Series.h"  #define DELAY\_PARAM 500000  #include "helper\_ex1.h"  #include "misc.h"  int main(void) {      Init\_board();      while (1) {          Toggle\_LED12();          Delay\_s(DELAY\_PARAM);      }  } |

* Whilst the delay function above is simple and performs it's task, there are two drawbacks that we could think of:
  + The number will get very bigger when the delay is long
  + This consumes the main thread of the board, which makes the board unresponsive during the loop time.
* Using the Virtual Bench, connect the wire D0 to GPC12 and connect the GND wire to the GND pin on the learning board
  + Display the waveform on the Virtual Bench and use the measurement tools to show the period of the waveform

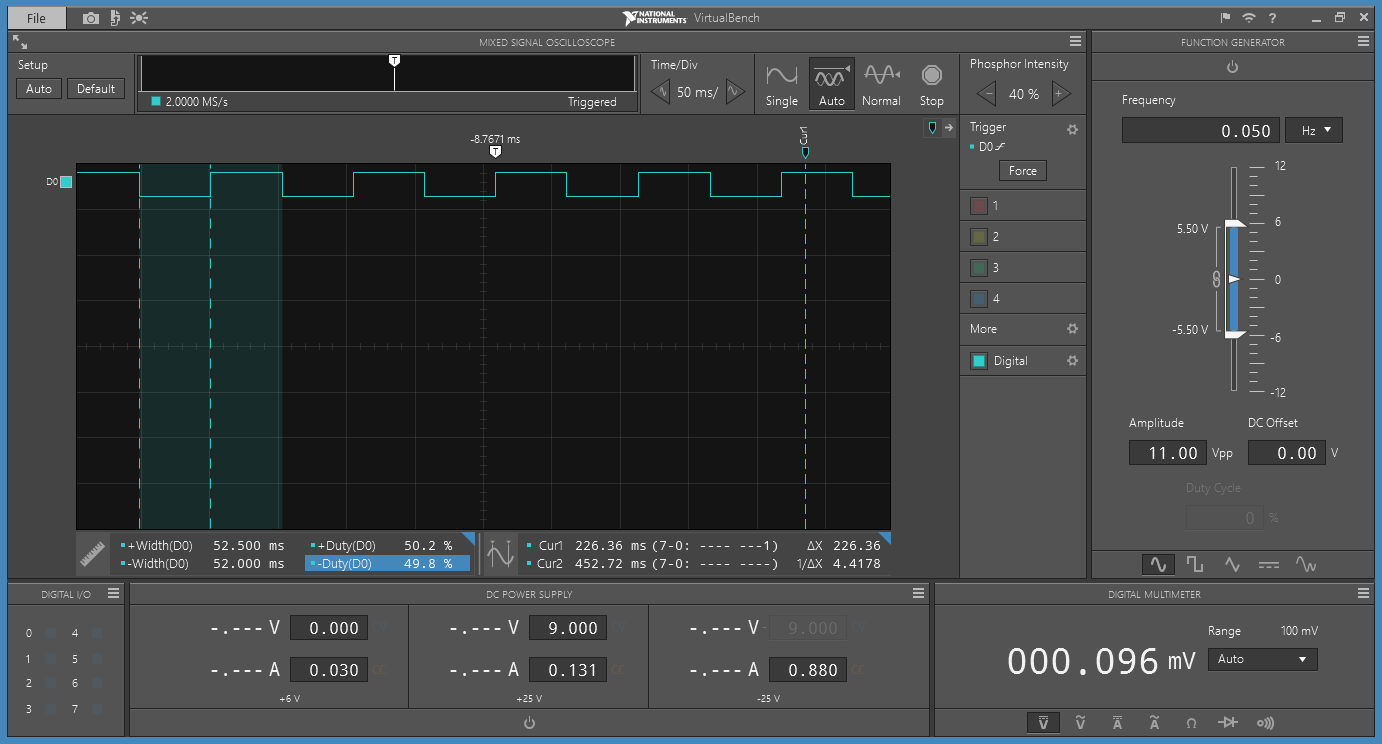


Figure 2. VirtualBench Waveform Exercise 1

* + How long does one delay function take to complete its cycle? Justify your answer

The duration of the delay function equals to the width of D0, which is about 50ms.

# Exercise 2 – Creating an LED Chaser

In this exercise, we extended the previous one and toggled all 4 LEDs on the board in a consecutive sequence.

## **Question 1**

As we could see from the [Figure 1](#_Question_1), the 5-8 LEDs were associated with the following pins

|  |  |
| --- | --- |
| PIN | LED |
| PIN 12 PORTC | LED 5 |
| PIN 13 PORTC | LED 6 |
| PIN 14 PORTC | LED 7 |
| PIN 15 PORTC | LED 8 |

Table 2. LEDs Configuration

## **Question 2**

Using your experience from Step 2 in Exercise 1, write a function to configure the pins, connected to these three remaining LEDs, as appropriate outputs.

|  |
| --- |
| void Set\_output\_mode\_LEDs() {  PC - > PMD &= ~(0xFF ul) << 24;  PC - > PMD |= (0x55 ul) << 24;  } |

## **Question 3**

Using your experience from Step 4 in Exercise 1, write a function to implement the LED chaser. The delay for toggling each LED is achieved by Delay\_s(param), where param is a macro which has the value 500000.

|  |
| --- |
| void LED\_chaser(uint32\_t delay) {  uint8\_t i;  for (i = 0; i < 4; ++i) {  PC - > DOUT ^= 1 << (12 + i);  Delay\_s(delay);  PC - > DOUT ^= 1 << (12 + i);  Delay\_s(delay);  }  } |

And in “main.c” file, we called the function and passed the DELAY\_PARAM macro to it

|  |
| --- |
| int main(void){      Init\_board();      Set\_output\_mode\_LEDs();      while (1) {          LED\_chaser(DELAY\_PARAM);  }  } |

## **Question 4**

Display the waveforms of each LED on the virtual bench. You will need to connect the wires on Virtual Bench D0, D1, D2 and D3 to the appropriate pins on the Nu-LB-NUC140 learning board. The GND wire on the Virtual Bench must be connected to a Ground pin on the Nu-LB-NUC140.

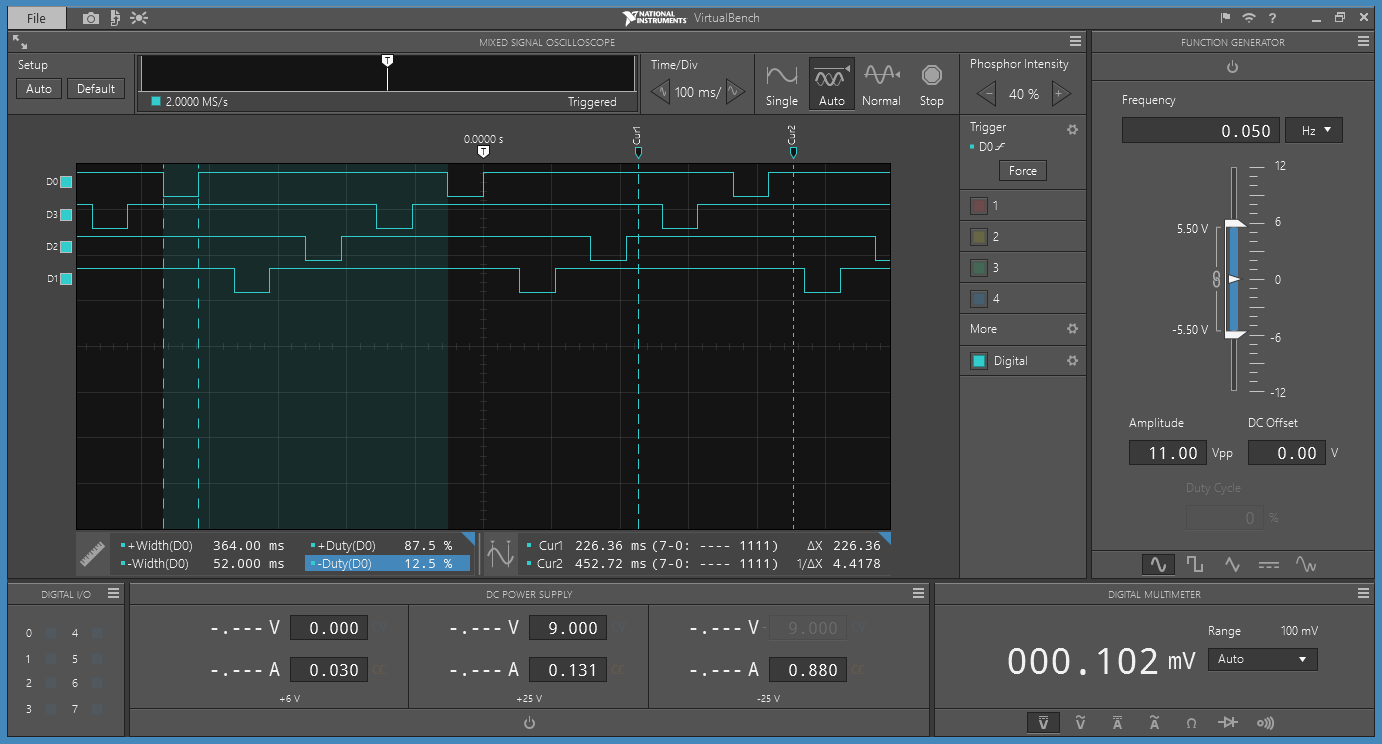


Figure . VirtualBench result for LED chaser

# Exercise 3 - Controlling the LED Brightness

## **Question 1**

Assume that the switching frequency for the LED is 1kHz. From your answer in Step 5 from Exercise 1, estimate the value of the delay macro to achieve a switching period of 1kHz.

1kHz is equivalent to **1000000** ns. Therefore, the value of the delay macro would be **10000.**

## **Question 2**

Write software code which switches the LED at a frequency of 1kHz with the duty cycle set to 50%.

* Build and run your program. What do you observe about the LED?

The LED is “half-illuminated”, which is less bright than the LED at 100%.

* Use the Virtual Bench to confirm that the LED is operating at 1kHz at 50% duty cycle.

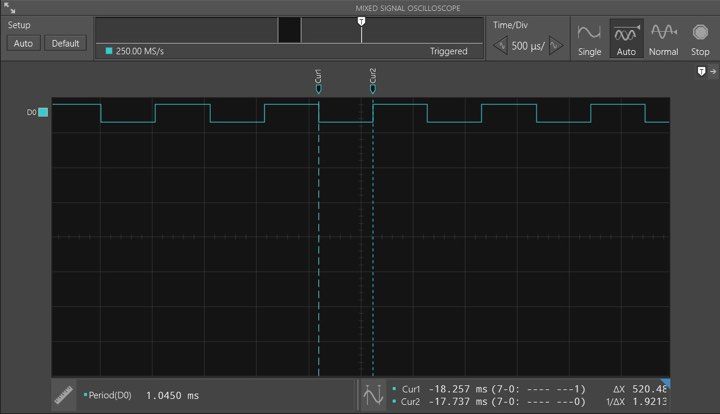


Figure . VirtualBench result for LED at 50% duty cycle

* How can you change the delay function parameters to change the duty cycle?

We could change the delay function to return different values depends on the current state off the LED.

Proposed function

|  |
| --- |
| #include "helper\_ex3.h"  #include "misc.h"  #define STATE\_ON 0  #define STATE\_OFF 1  void Delay\_ms(uint32\_t count, int duty, int state) {      if (state == STATE\_ON) {          count = count / 100 \* duty;      } else {          count = count / 100 \* (100 - duty);      }      Delay\_s(count);  } |

* What delay parameters are required to achieve a duty cycle of 10% - Enter these parameters into your program, then build and run your program.

|  |
| --- |
| #define DELAY\_PARAM 10000  #define DUTY\_CYCLE 10  int main(void) {  Init\_board();  Set\_GPC12\_output();  int state = 0;  while (1) {  Toggle\_LED12();  Delay\_ms(DELAY\_PARAM, DUTY\_CYCLE, state);  state = !state;  }  } |

* What do you observe about the LED?

A lower level brightness of the LED, which is significantly noticeable compared to it when at 100% and 50%.

* Use the Virtual Bench to confirm the frequency and duty cycle.

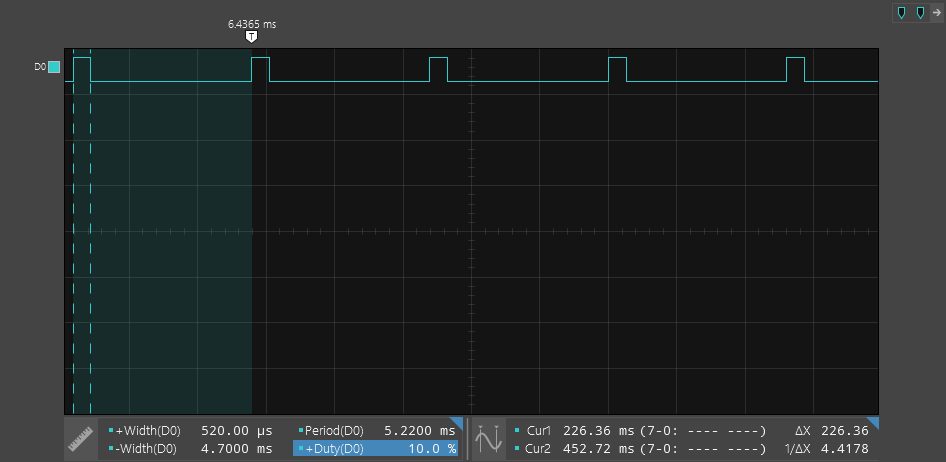


Figure . VirtualBench duty cycle at 10%

## **Question 3**

* Write a function which gradually turns on the LED and then gradually turns off the LED. Keep the switching frequency at 1kHz.

### Source code

|  |  |
| --- | --- |
| |  | | --- | | void Adv\_adjust\_brightness() {  Init\_board();  Set\_GPC12\_output();  while (1) {  uint32\_t i;  for (i = 0; i <= 10000; i++) {  Toggle\_LED12();  Delay\_s(i);  Toggle\_LED12();  Delay\_s((10000 - i));  }  for (i = 10000; i > 0; i--) {  Toggle\_LED12();  Delay\_s(i);  Toggle\_LED12();  Delay\_s((10000 - i));  }  }  } | |

* Use the Virtual Bench to show the behavior of the the LED when it is being turned on and when the LED is being turned off.
  + LED increasing in brightness



Figure . VirtualBench when the LED increasing in brightness

* + LED decreasing in brightness



Figure . VirtualBench when the LED decreasing in brightness

# Exercise 4 - Creating an Advanced LED Chaser

## **Question 1**

What effect (if any) does the LED switching frequency have on the light being emitted from the LED?

As making the pulse frequency appear as a continuous light output, the “ON-OFF” flashing condition does not affect what is seen by the human eye as it “fills” in the gaps between “ON” and “OFF” light pulses, providing it is high enough. Therefore, a continuous light of the same average intensity is less bright than the pulse at a frequency of 100Hz or more.

# Exercise 5: Controlling an RGB LED

According to the schematic, the RGB LED color is determined by the GPIO PIN A value, PIN 12, 13 and 14.

* **GPA 12**: Blue 0 = on, 1 = off
* **GPA 13**: Green 0 = on, 1 = off
* **GPA 14**: Red 0 = on, 1 = off

## **Indication of the complexity involved in**

There are 3 colors which are Red, Blue and Green and each of them could take 2 possible states 0 (on) and 1 (off).

Thus, the possible combinations for the states are 23 = 8

And to generate every possible color, we need to go through all possible states of 3 LEDs and varies the duty cycle for each color when its LED is on. This can be seen more clearly from our flowchart.

## **Improvements that would enhance the quality of the LED's color** **being emitted**

To enhance the quality of the LED’s color being emitted, we could increase the frequency of the LED

## **Flowchart**

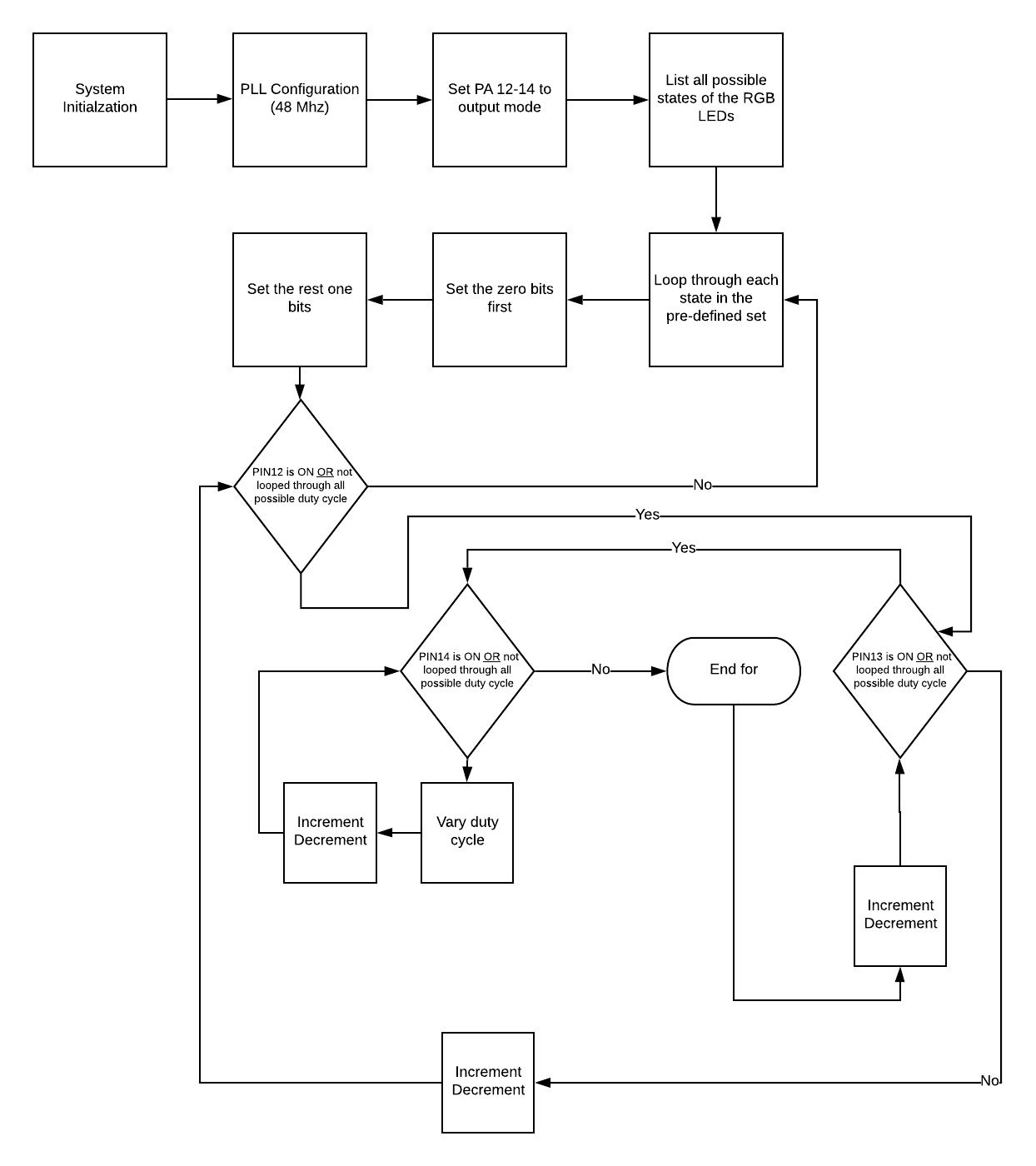


Figure . Flowchart of the algorithm for Exercise 5

# Conclusion

By completing the laboratory, the students are now be confident on programming an embedded system using C, preprocessing directives, using multiple header and source files and controlling LEDs with an embedded system using GPIO. One more thing that all the students are now familiar with combining C code to Nuvoton NUC140 development and analyzing waveform by using Virtual Bench.

Specific key learning achievements include:

* Learned the basic structure of an embedded system program code by using C.
* Preprocessing directives.
* Comfortable with using multiple header and source files.
* Controlling LEDs with an embedded system using GPIO.

# Lab work and contribution

|  |  |  |
| --- | --- | --- |
| **Member name** | **Work contribution** | **Work contributions in words** |
| Phung Minh Tuan | 33% | Tuan is doing exercise 1 and 5. Also, he writes for his own report for the exercise 1 and 5. |
| Le Huu Nghia | 33% | Nghia is doing exercise 2 and 3. Also, he writes for his own report for the exercise 2 and 3. |
| Trung Nguyen | 33% | Trung is doing exercise 4. Also, he writes for his own report for the exercise 4. |

# References

[1] Andrew Smith. Clock System & GPIO. Available: https://rmit.instructure.com/courses/55465/files/5617879/download?wrap=1. [Accessed: 25- March- 2019].

[2] NUC130/NUC140 Technical Reference Manual. Available: https://rmit.instructure.com/courses/55465/files/5638356/download?verifier=LdJjAu4ogz8E1w8TUSrIAVuoX9E4FxotJIbIQgCT&wrap=1. [Accessed: 25- March- 2019].

[3] ElectronicsTutorial. (2019). The Light Emitting Diode. [online] Available at: https://www.electronics-tutorials.ws/diode/diode\_8.html [Accessed 25 Mar. 2019].

[4] ElectronicsTutorial. (2019). The Light Emitting Diode. [online] Available at: https://www.electronics-tutorials.ws/diode/diode\_8.html [Accessed 25 Mar. 2019].

[5] Andrew Smith. Getting started with Keil & the Nu-LB-NUC140 Learning Board. Available:

https://rmit.instructure.com/courses/59115/pages/t1-getting-started-with-keil-and-the-nu-lb-nuc140-learning-board?module\_item\_id=1602209

[Accessed: 25-March-2019].

[6] Andrew Smith. The Build Process. Available:

https://rmit.instructure.com/courses/59115/pages/t2-the-build-process?module\_item\_id=1605731

[Accessed: 25 Mar 2019].

[7] Andrew Smith. Fundamental of ARM. Available:

https://rmit.instructure.com/courses/59115/pages/t3-lecture-resources?module\_item\_id=1536288

[Accessed: 25 Mar 2019]

[8] Andrew Smith. General purpose Input/Output. Available:

https://rmit.instructure.com/courses/59115/pages/t4-lecture-resources?module\_item\_id=1619028

[Accessed: 25 Mar 2019]

[9] Andrew Smith. Memory Map & Clock System. Available:

<https://rmit.instructure.com/courses/59115/pages/t5-lecture-resources?module_item_id=1536299>

[10]