Hi, and welcome to our repo. This repo is prepared for educational purposes. I hope it will be useful and give you an opportunity to learn something and improve yourself. So let's Start!!!!!!!

You can read this also from the pdf for **readme.pdf** !!! (The pdf is auto generated so sorry if some parts look bad)

1.Introduction

1.1 Stm32g031k8 nucleo pinout

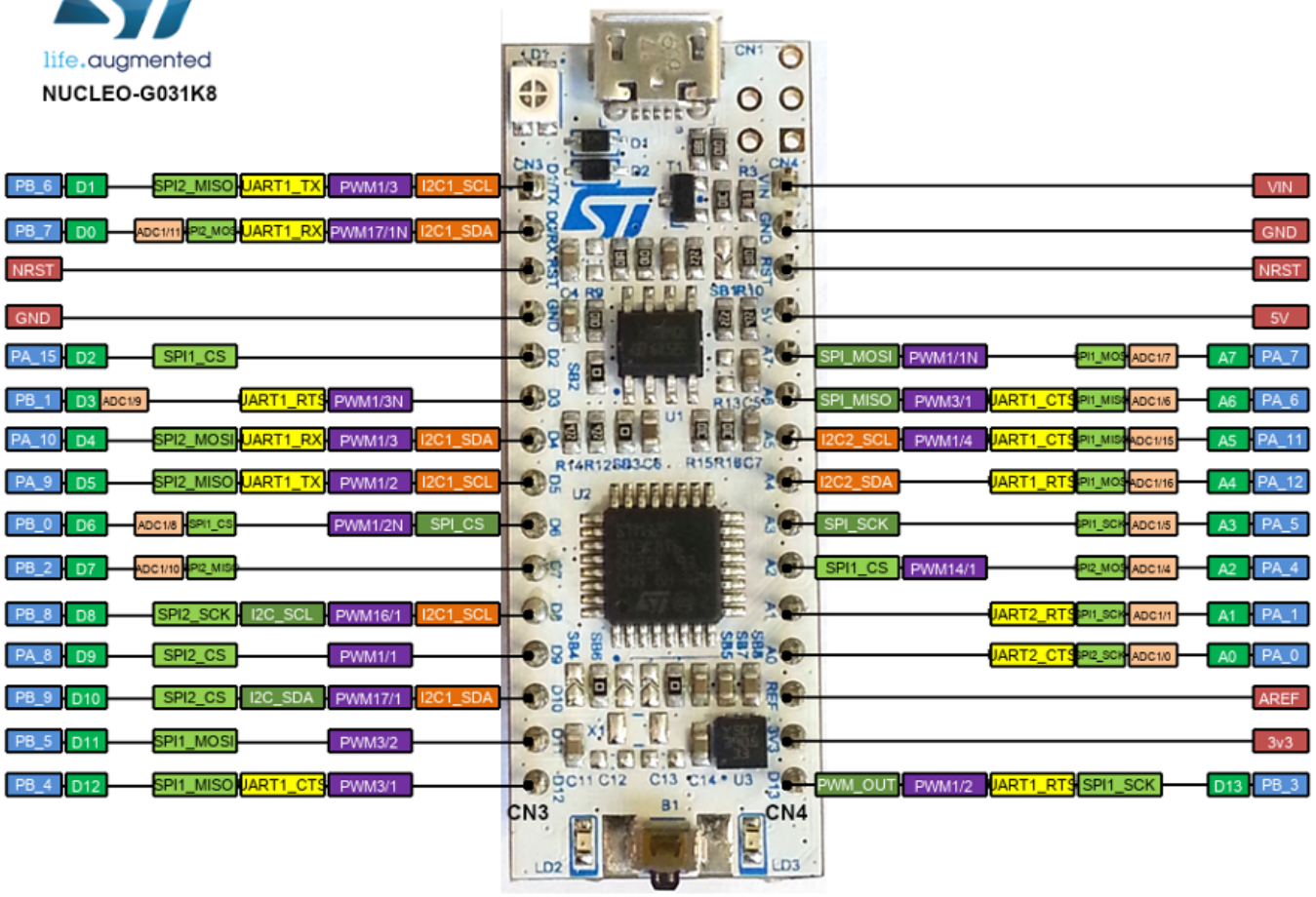

Figure 1:Stm32g031k8 nucleo board pinout

As shown at Figure1, the pins can used for multiple purposes. The only digital pins are labeled with Dx and pins that can read analog signals with ADC is labeled with Ax.

1.2 On board IC’s

Figure 2: Stm32g031k8 nucleo board back

The board include a usb port, this port allows code running and debugging using ST- link. For ST-link a stm32f103 IC is used and is connected to the back of the nucelo board. The stm32g0 IC operate at 3.3V because of it a 5V 3.3V regulator appear on the board. A 16Mhz oscillator appear to clock the stm32g0.

1.3 Peripherals and their usage

1.3.1 GPIOs:

GPIOs (general-purpose input/output) port handles both incoming and outgoing digital signals. As an input port, it can be used to communicate to the CPU the ON/OFF signals received from switches, or the digital readings received from sensors.

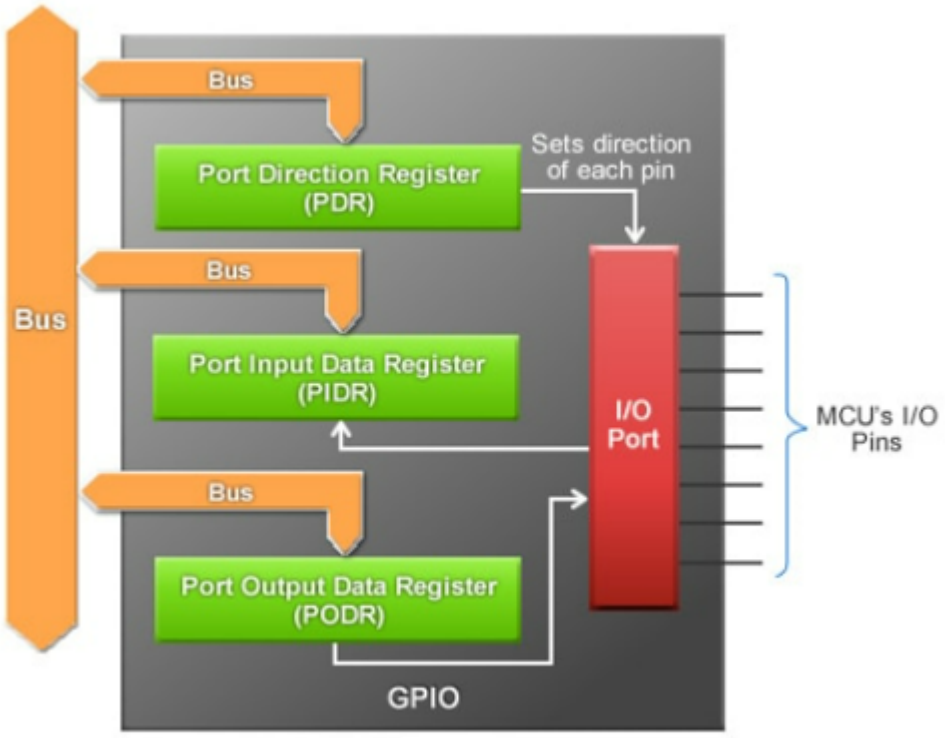

Figure 3: GPIO structure

1.3.2 SRAM:

SRAM (static RAM) is a type of random-access memory (RAM) that retains data bits in its memory if power is being supplied. Unlike dynamic RAM (DRAM), which must be continuously refreshed, SRAM does not have this requirement, resulting in better performance and lower power usage.

1.3.3 FLASH:

The STM32G0 embeds up to 128 Kbytes of single-bank Flash memory. The Flash memory interface manages all memory access (read, programming and erasing) as well as memory protection, security and option bytes.

1.3.4 DMA:

The direct memory access (DMA) controller is a bus master and system peripheral with single-AHB architecture. With 5 channels, it performs data transfers between memory-mapped peripherals and/or memories, to offload the CPU.

1.3.5 DMAMUX:

The DMAMUX request multiplexer enables routing DMA request lines from the peripherals to the DMA controllers in the products.

1.3.6 CRC:

The CRC (cyclic redundancy check) calculation unit is used to get a CRC code using a configurable generator polynomial value and size, perform data integrity checks of off-/on-chip memories as a background task without CPU intervention.

1.3.7 RNG:

Provides random numbers which are used when producing an unpredictable result is desirable.

1.3.8 RCC:

The **RCC** peripheral is used to control the internal peripherals, as well as the reset signals and clock distribution. The **RCC** gets several internal (**LSI**, **HSI** and **CSI**) and external (**LSE** and **HSE**) clocks. They are used as clock sources for the hardware blocks, either directly or indirectly

1.3.9 USART (universalsynchronous asynchronous receiver transmitter)

The USART is a hardware that enables the device to communicate using serial protocol.

1.3.10 ADC:

The analog-to-digital converters allows the microcontroller to accept an analog value like a sensor output and convert the signal into the digital domain

1.3.11 SPI(internal Standard Peripheral Interface):

Provides. simple communication interface allowing the microcontroller. to communicate with external devices.

1.3.12 TIM:

TIM peripheral is a multi-channel timer unit. It's include advanced-control timers, general-purpose timers and basic timers.

1.3.13 I2C:

The I2C is a multi-master, multi-slave, synchronous, bidirectional, half-duplex serial communication bus.

1.3.14 RTC:

RTC, is a digital clock with a primary function to keep accurate track of time even when a power supply is turned off or a device is placed in low power mode.

1.3.15 PLL:

PLL is a clock generation engine in the MCU which is used to generate the clock speed which is much higher than the internal HSI or external clock

2 INTRODUCTION TO PROGRAMMMING Registers (RCC, GPIO)

2.1 RCC Configuration I/O port clock enable register (RCC_IOPENR)

The register shown in Figure 1 is defined as RCC_IOPENR, reset and clock setting is activated by making the related bits 1.

5.4.13 I/O port clock enable register (RCC_IOPENR)

Address: 0x34

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	GPIOFEN	GPIOEEN(1)	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN
										rw	rw	rw	rw	rw	rw

1. Only significant on devices integrating the corresponding peripheral, otherwise reserved. Refer to *Section 1.4: Availability of peripherals*.

Bits 31:6 Reserved, must be kept at reset value.

Bit 5 **GPIOFEN:** I/O port F clock enable
This bit is set and cleared by software.
0: Disable
1: Enable

Bit 4 **GPIOEEN:** I/O port E clock enable(1)
This bit is set and cleared by software.
0: Disable
1: Enable

Bit 3 **GPIODEN:** I/O port D clock enable
This bit is set and cleared by software.
0: Disable
1: Enable

Bit 2 **GPIOCEN:** I/O port C clock enable
This bit is set and cleared by software.
0: Disable
1: Enable

Bit 1 **GPIOBEN:** I/O port B clock enable
This bit is set and cleared by software.
0: Disable
1: Enable

Bit 0 **GPIOAEN:** I/O port A clock enable
This bit is set and cleared by software.
0: Disable
1: Enable

Figure 4: RCC_IOPENR register.

2.2 GPIO CONFIGURATIONS

2.2.1 Peripherals addresses

GPIOX base addresses were defined as GPIOX_BASE and these addresses were accessed to the registers that make the settings on the X ports. It is seen that there is an offsett of 0x400 between the ports.

Table 6. STM32G0x1 peripheral register boundary addresses

Bus	Boundary address	Size	Peripheral	Peripheral register map
-	0xE000 0000 - 0xE00F FFFF	1MB	Cortex®-M0+ internal peripherals	-
IOPORT	0x5000 1800 - 0x5FFF FFFF	~256 MB	Reserved	-
	0x5000 1400 - 0x5000 17FF	1 KB	GPIOF	*Section 7.4.12 on page 248*
	0x5000 1000 - 0x5000 13FF	1 KB	GPIOE	*Section 7.4.12 on page 248*
	0x5000 0C00 - 0x5000 0FFF	1 KB	GPIOD	*Section 7.4.12 on page 248*
	0x5000 0800 - 0x5000 0BFF	1 KB	GPIOC	*Section 7.4.12 on page 248*
	0x5000 0400 - 0x5000 07FF	1 KB	GPIOB	*Section 7.4.12 on page 248*
	0x5000 0000 - 0x5000 03FF	1 KB	GPIOA	*Section 7.4.12 on page 248*

Figure 5: GPIOX address map.

2.2.2 GPIO MODER REGISTER

Using GPIOX_BASE addresses, the modes of the pins in the MODER registry can be set by accessing the MODER registry with offset 0x00.

7.4.1 GPIO port mode register (GPIOx_MODER) (x =A to F)

Address offset:0x00

Reset value: 0xEBFF FFFF for port A

Reset value: 0xFFFF FFFF for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODE15[1:0]		MODE14[1:0]		MODE13[1:0]		MODE12[1:0]		MODE11[1:0]		MODE10[1:0]		MODE9[1:0]		MODE8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODE7[1:0]		MODE6[1:0]		MODE5[1:0]		MODE4[1:0]		MODE3[1:0]		MODE2[1:0]		MODE1[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **MODE[15:0][1:0]:** Port x configuration I/O pin y (y = 15 to 0)
These bits are written by software to configure the I/O mode.
00: Input mode
01: General purpose output mode
10: Alternate function mode
11: Analog mode (reset state)

Figure 6: GPIOx_MODER Register

2.2.3 GPIO ODR REGISTER

Using GPIOX_BASE addresses, the ODR register is accessed with 0x14 Offset and the pins are powered by assigning a value of 1 to the relevant bits in the ODR register.

7.4.6 GPIO port output data register (GPIOx_ODR) (x = A to F)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OD15	OD14	OD13	OD12	OD11	OD10	OD9	OD8	OD7	OD6	OD5	OD4	OD3	OD2	OD1	OD0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **OD[15:0]:** Port output data I/O pin y (y = 15 to 0)
These bits can be read and written by software.
Note: For atomic bit set/reset, the OD bits can be individually set and/or reset by writing to the GPIOx_BSRR register (x = A..D, F).

Figure 7: GPIOx_ODR register

2.2.4 GPIO IDR REGISTER

Using GPIOX_BASE addresses, the IDR register is accessed with 0x10 Offset and the pins state can read by read this register.

7.4.5 GPIO port input data register (GPIOx_IDR) (x = A to F)

Address offset: 0x10

Reset value: 0x0000 XXXX

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ID[15:0]:** Port x input data I/O pin y (y = 15 to 0)
These bits are read-only. They contain the input value of the corresponding I/O port.

Figure 8: GPIOx_IDR register

3 ASSEMBLY_001_led

To turn on the LED connected to the PA8 pin, the clock of the GPIOA was first activated by set GPIOAEN bit (bit 0) in RCC_IOPENR register. Then, bits 12-13 were set to 01 in MODER for PA8 to be output. Finally, the LED was powered by setting OD8 to 1 in ODR.

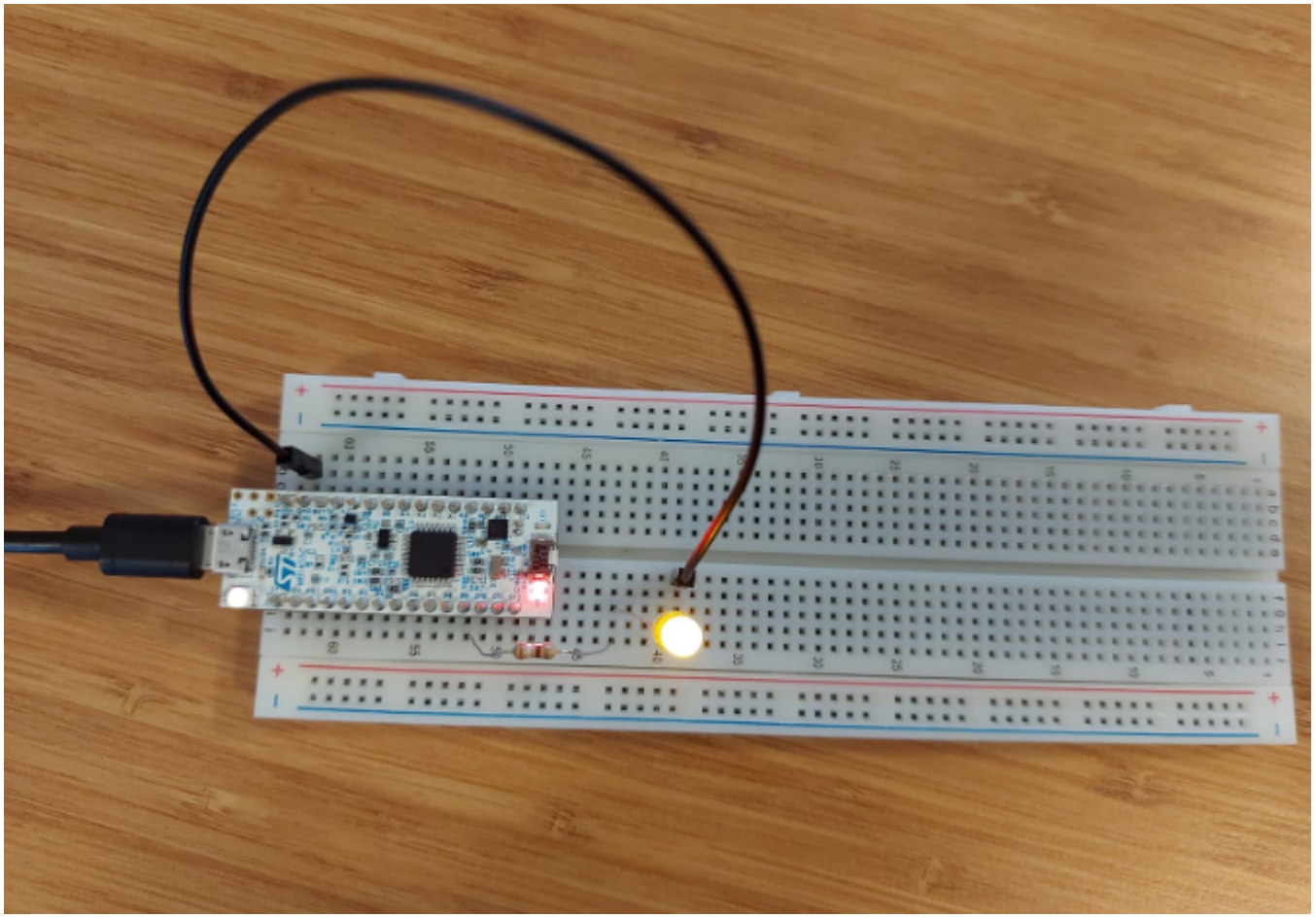

Figure 9: Led on at PA8 pin.

4 ASSEMBLY_002_GPIO_functions

This project includes a multi usage assembly gpio functions. You can take a look to improve yourself and it can be used later.

5 ASSEMBLY_003_ledon

To turn on the LED connected to the PA11, PA12, PB4, PB5 pin, the clock setting of GPIOA and GPIOB was first activated. Then, MODE Pin values were set to 01 in MODER

to make the pins output. Finally, the relevant pins in ODR were set to 1 and the LED was powered.

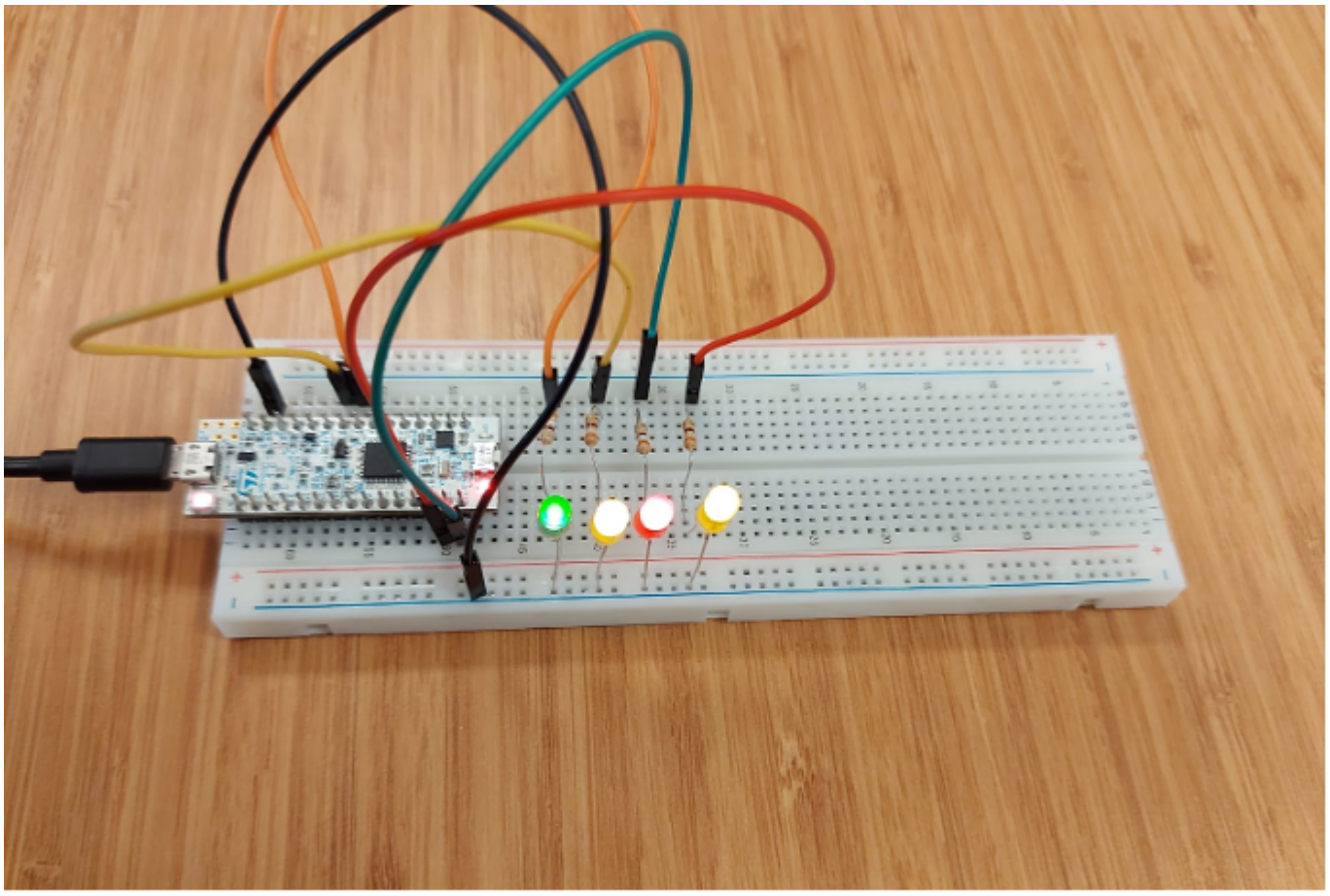

Figure 10: Four led on

6 ASSEMBLY_004_ledblink

6.1 Clock settings

In the beginning, the main function configures the microcontroller by enabling the clock for GPIOA (RCC_IOPENR register). To do this GPIOAEN is set to 1.

6.2 GPIO (PA8) configuration

The pin PA8 is set to output mode by using GPIOA_MODER register. To do this the Mode 8 bits is set to 01.

6.3 Toggling led and delay loop

Following the configuration, the ***blink_loop*** is initiated.This loop toggles the state of the LED on pin PA8 by reading the current state from the Output Data Register (GPIOA_ODR), XORing it with a mask that represents the state of the LED and storing the result back to the ODR. This effectively toggles the LED. The mask is 0x100, it makes only the bit 8 (OD8 bit) toggle and don't effect to other bits.

After toggling the LED state, the program introduces a delay by executing a simple countdown loop (delay_loop). The **DELAY_FREQ** constant determines the duration of the delay, adjusting the speed of the LED blinking.** DELAY_FREQ **is loaded to r0 and subtract one with ***subs ***instruction, reference to Cortex-M0 Technical Reference Manual (figure 10) it will take one cycle than with ***bne*** instruction if r0 not equal to 0 they will go again to the countdown loop (delay_loop), reference to Cortex-M0 Technical Reference Manual it will take three cycle . The loop continues until the countdown reaches zero. Then go again to blink_loop label and toggle the led. So the delay_loop function execution time is 4 cycle (1 from subs, and 3 from ben). Assume the Clock is 16Mhz so 16M cycle appear in 1 second, so for waiting for 1 second we have to do a process of 16M cycle, for do this we made the loop count for 4M times, that's make 16M cycle. The code with **DELAY_FREQ** = 4Mhz was tried, and a clock was kept, the result of the observation was that the LED was delayed by less than 1 second, faster than expected. Therefore, dividing the frequency by 3 was tried and the code was run. When done this way, it was observed that the delay was 1 second. Although we cannot fully explain the reason for this, it seems that it may be because the loop execution time is just 3 cycles.

Subtract	Negate	`RSBS Rd, Rn, #0`	1

Branch	Conditional	`B <cc> <label>`	1 or 3[d]
	Unconditional	`B <label>`	3
	With link	`BL <label>`	4
	With exchange	`BX Rm`	3
	With link and exchange	`BLX Rm`	3

Figure 11: Time of instructions Cortex-M0 Technical Reference Manual

7 ASSEMBLY_006_onbordled_blink

The onboard led is connected to PC6 pin. The led is blinking by set the pin high and low with 1s intervals.

7.1 Flowchart

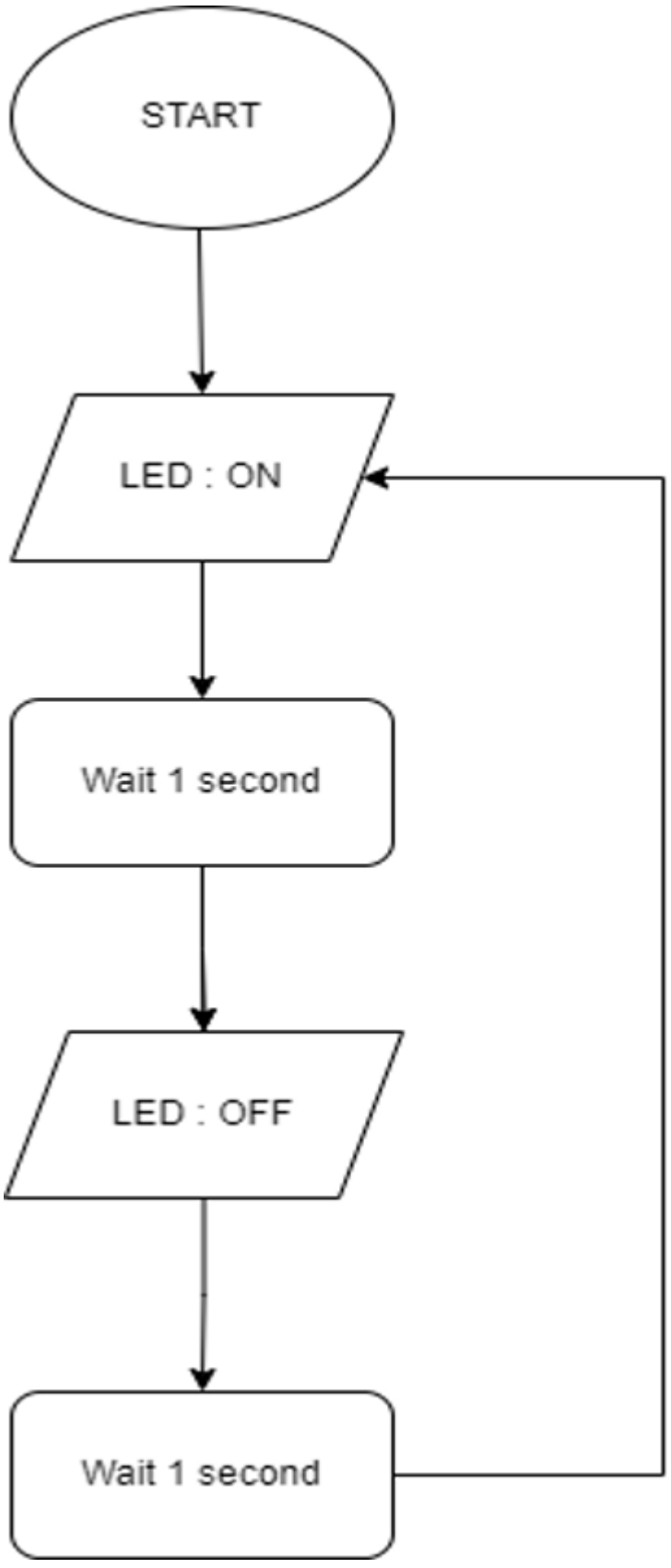

Figure 12: flowchart of ASSEMBLY_006_onbordled_blink

8 ASSEMBLY_007_onboardled_on_with_button

A button is connected to PA0 pin with pull down mode. When the button is pressed, the input PA0 pin becomes 1 and the LED connected to the PC6 pin on the board lights up. When the button is not pressed, the PA0 pin becomes 0 and the LED connected to the PC6 pin on the board does not light.

8.1 Schematic

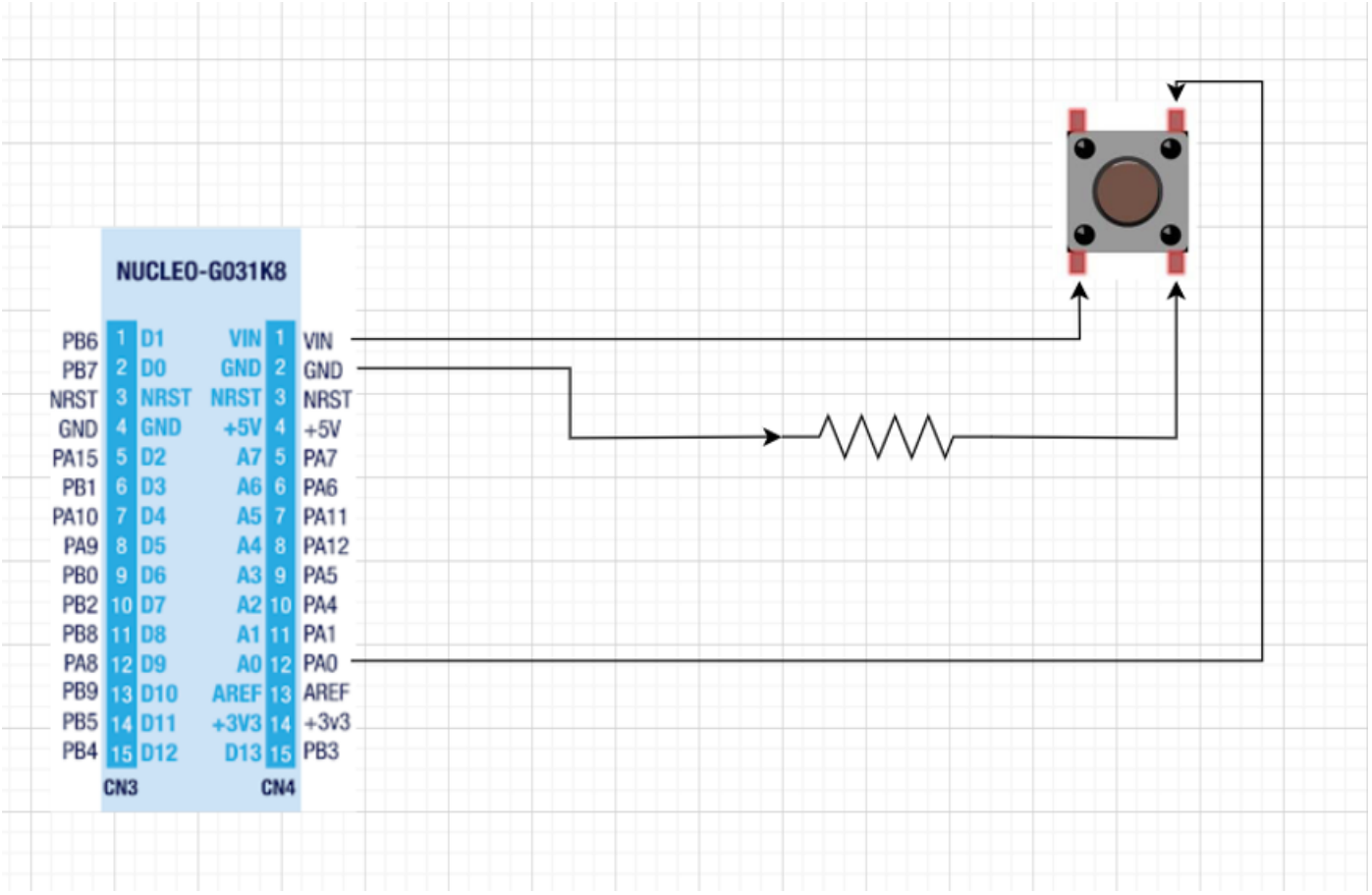

Figure 13: Schematic of ASSEMBLY_007_onboardled_on_with_button

8.2 Flowchart

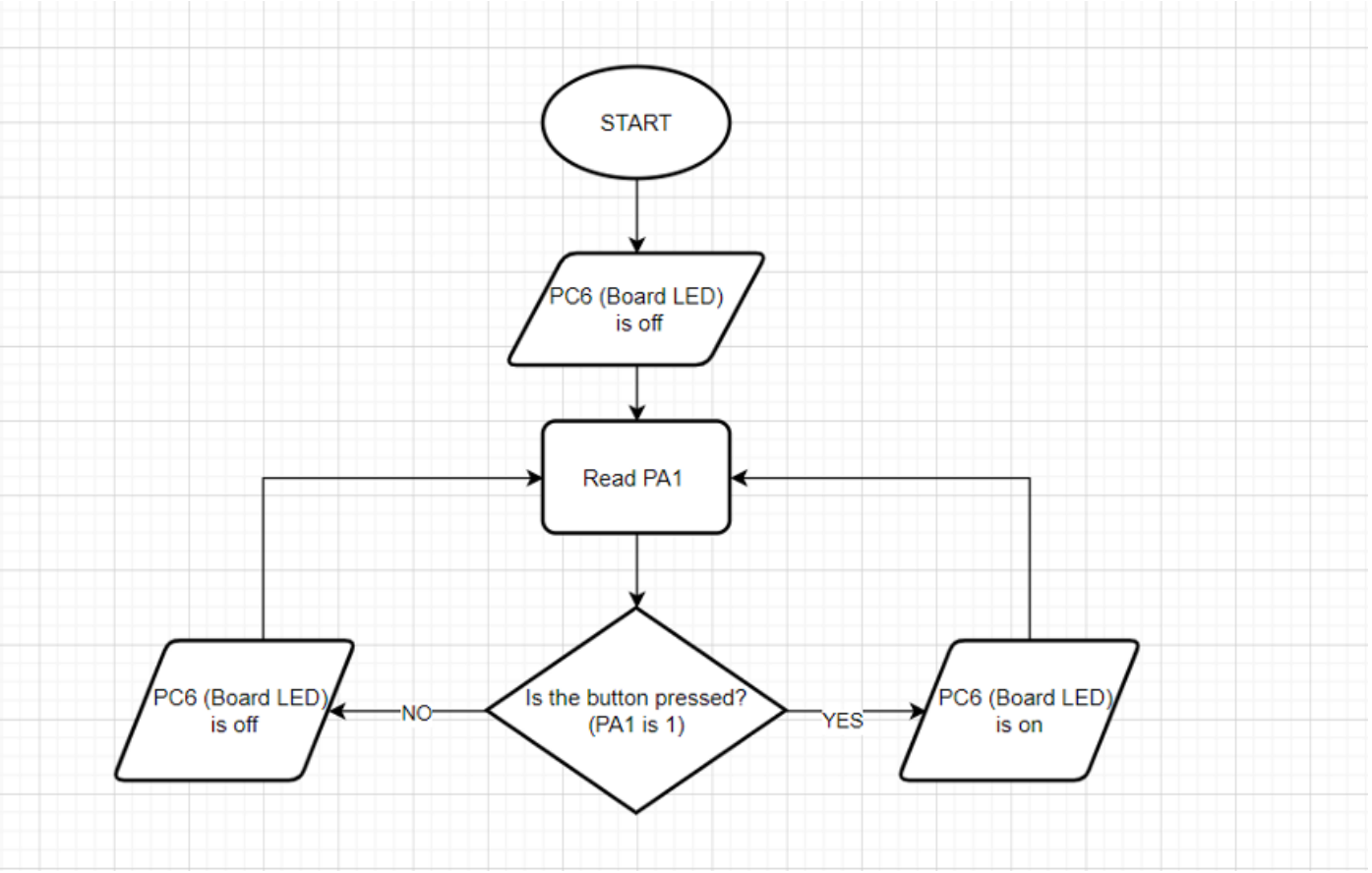

Figure 14: Flowchart of ASSEMBLY_007_onboardled_on_with_button

9 ASSEMBLY_008_8led_blink

The 8 pins of the B port were set to output mode and connected to the LEDs. LEDs were switched on and off with 1 second intervals.

9.1 Schematic

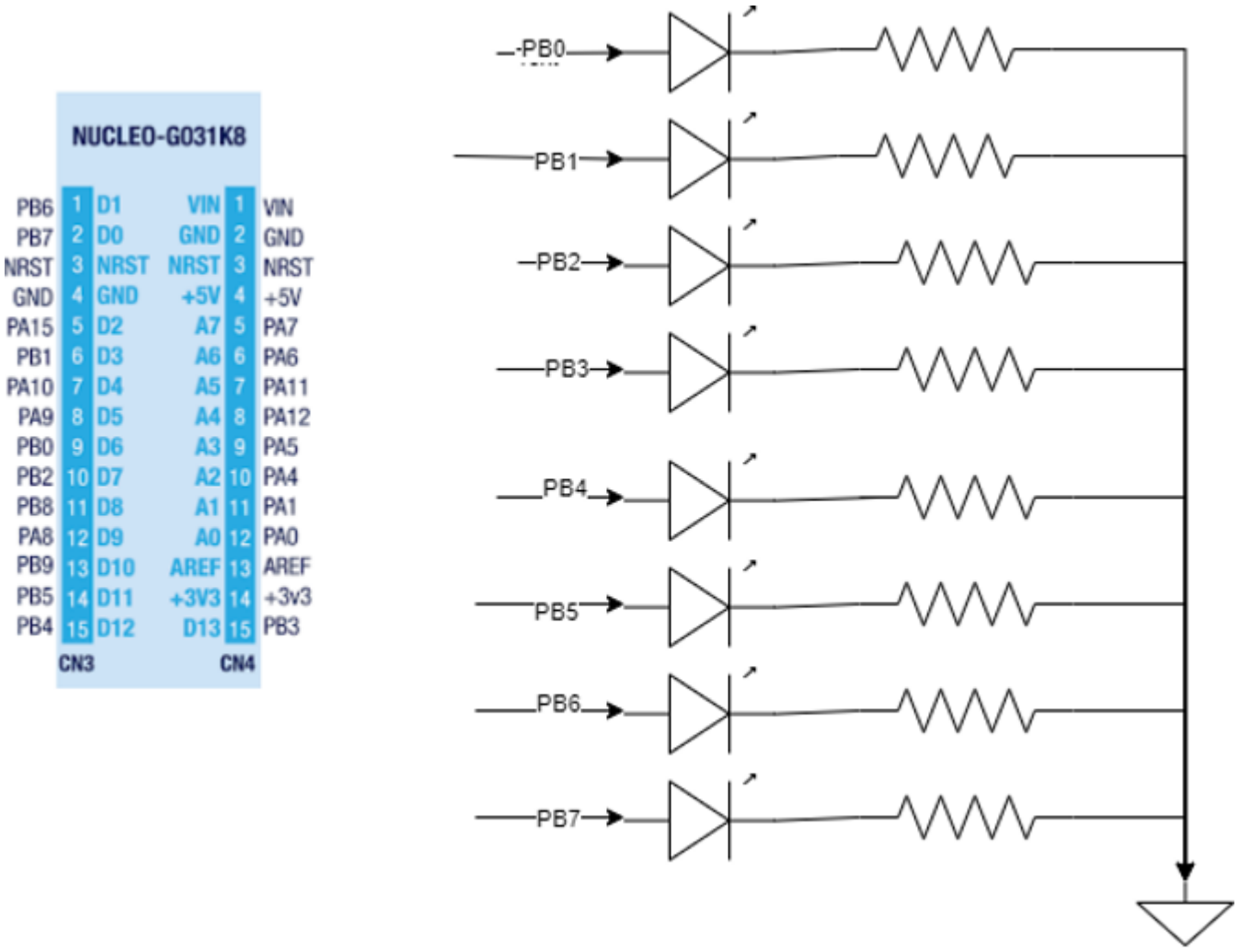

Figure 15: Schematic of ASSEMBLY_008_8led_blink

9.2 Flowchart

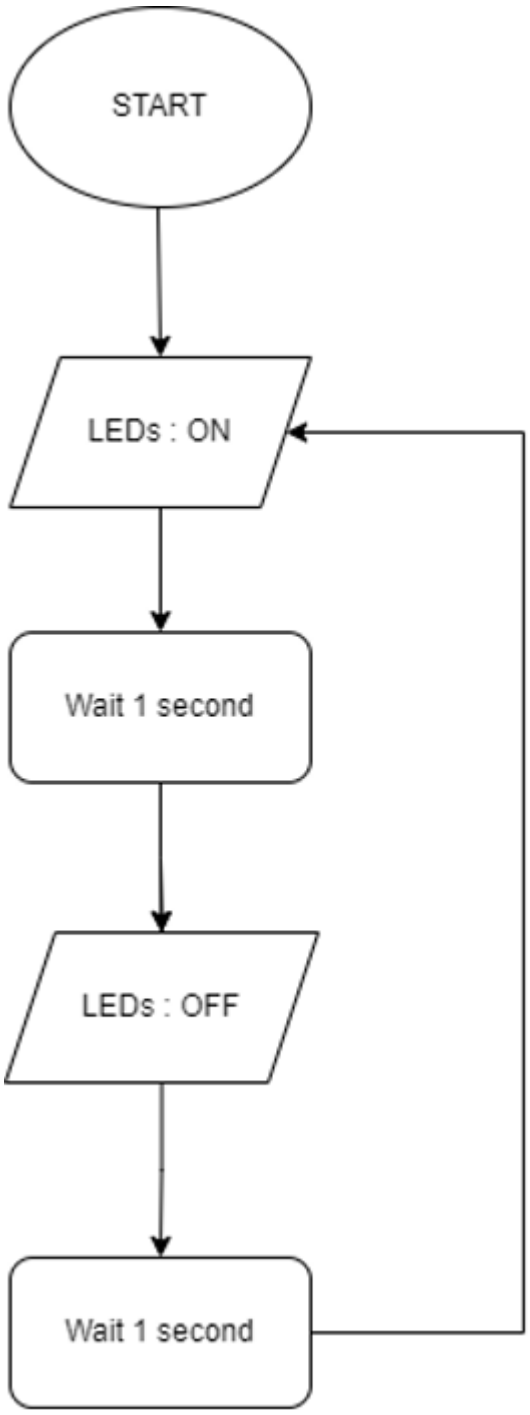

Figure 16: Flowchart of ASSEMBLY_008_8led_blink

10 ASSEMBLY_009_led_rotate

A button is connected to PA8 with pull down mode and the leds is connected to port B starting from PB0 to PB7. First starting from LED0, LED3 was switched on with 100 ms intervals. Then a shift pattern is set 0000 0111 for 8 bit as initial condition, 1 for leds on and 0 for off. Because registers are 32 bit we will do the pattern as 0000 0111 0000 0111 0000 0111 0000 0111.Than using rotate instruction, the shifting pattern is implemented. When the button pressed the shifting pattern change.

10.1 Schematic

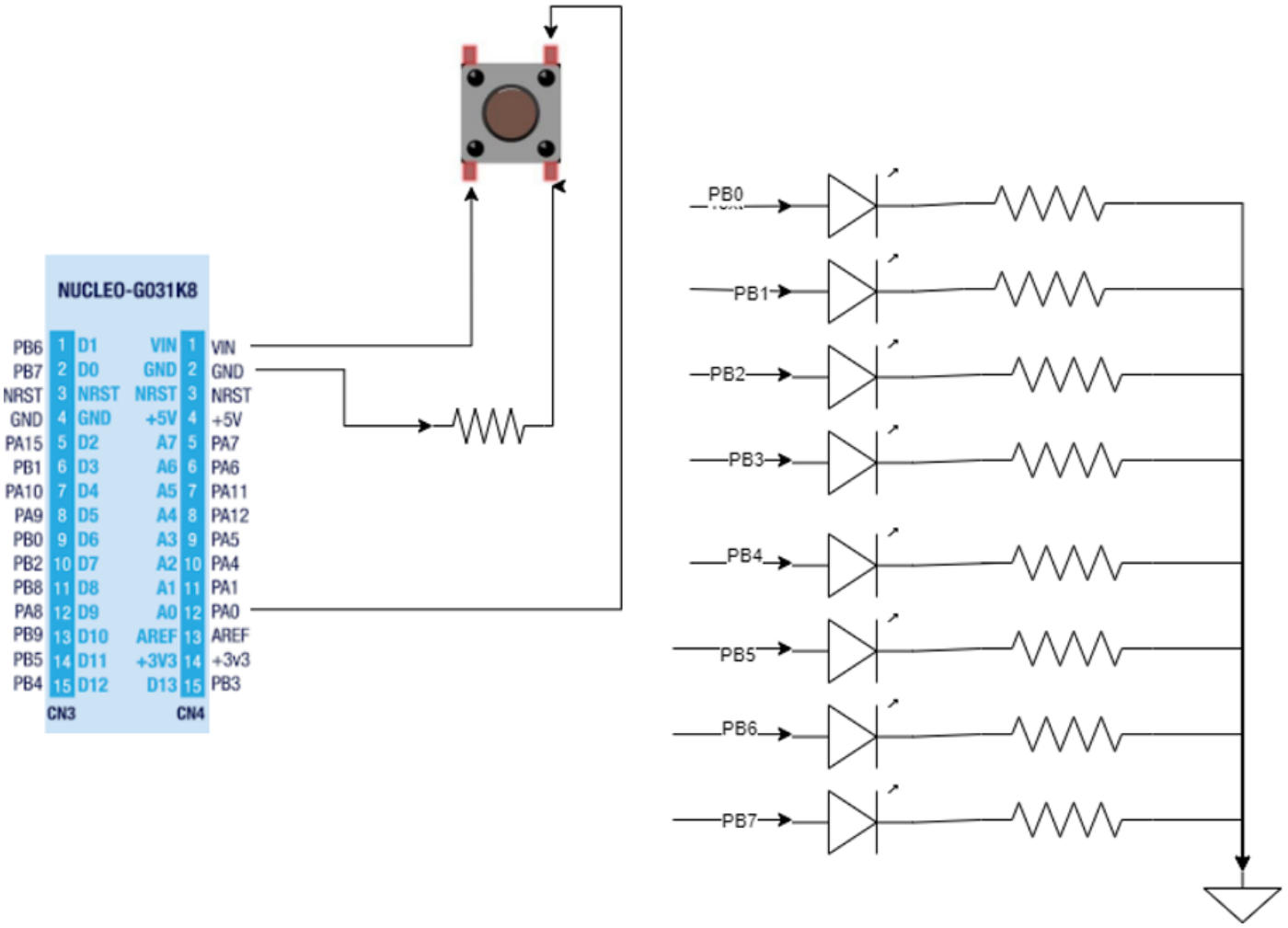

Figure 17: Schematic of ASSEMBLY_009_led_rotate

10.2 Flowchart

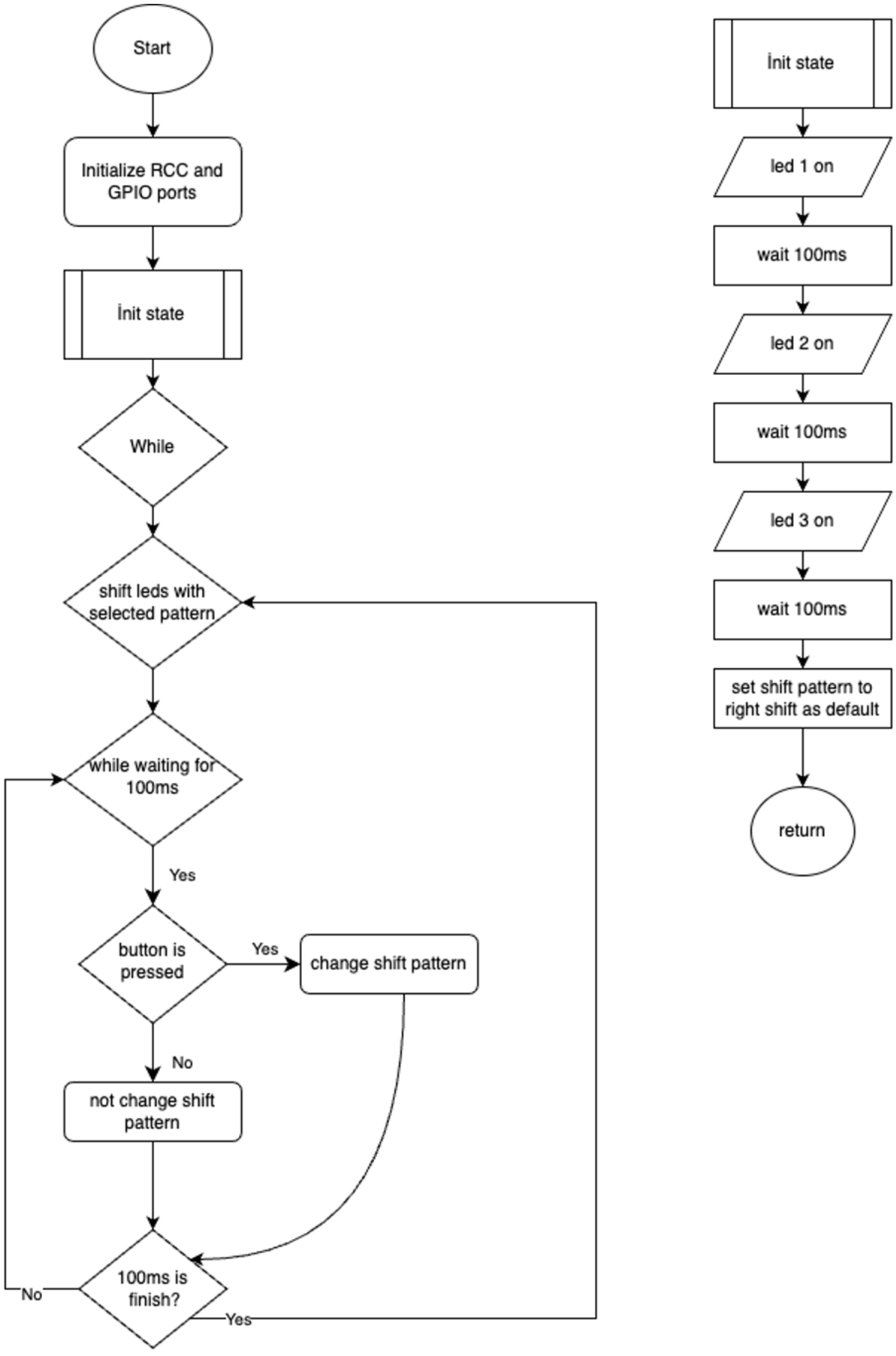

Figure 18: Flowchart of ASSEMBLY_009_led_rotate

11 ASSEMBLY_010_knight_rider

Leds was connected to the pins of port B. Firstly, starting from LED0, LED3 was switched on with 100ms intervals. Then the led numbers were shifted to the right with 100ms intervals. With the algorithm created, when LED8 was reached, it was shifted towards LED0, and when LED0 was reached, it was shifted towards LED8.

11.1 Schematic

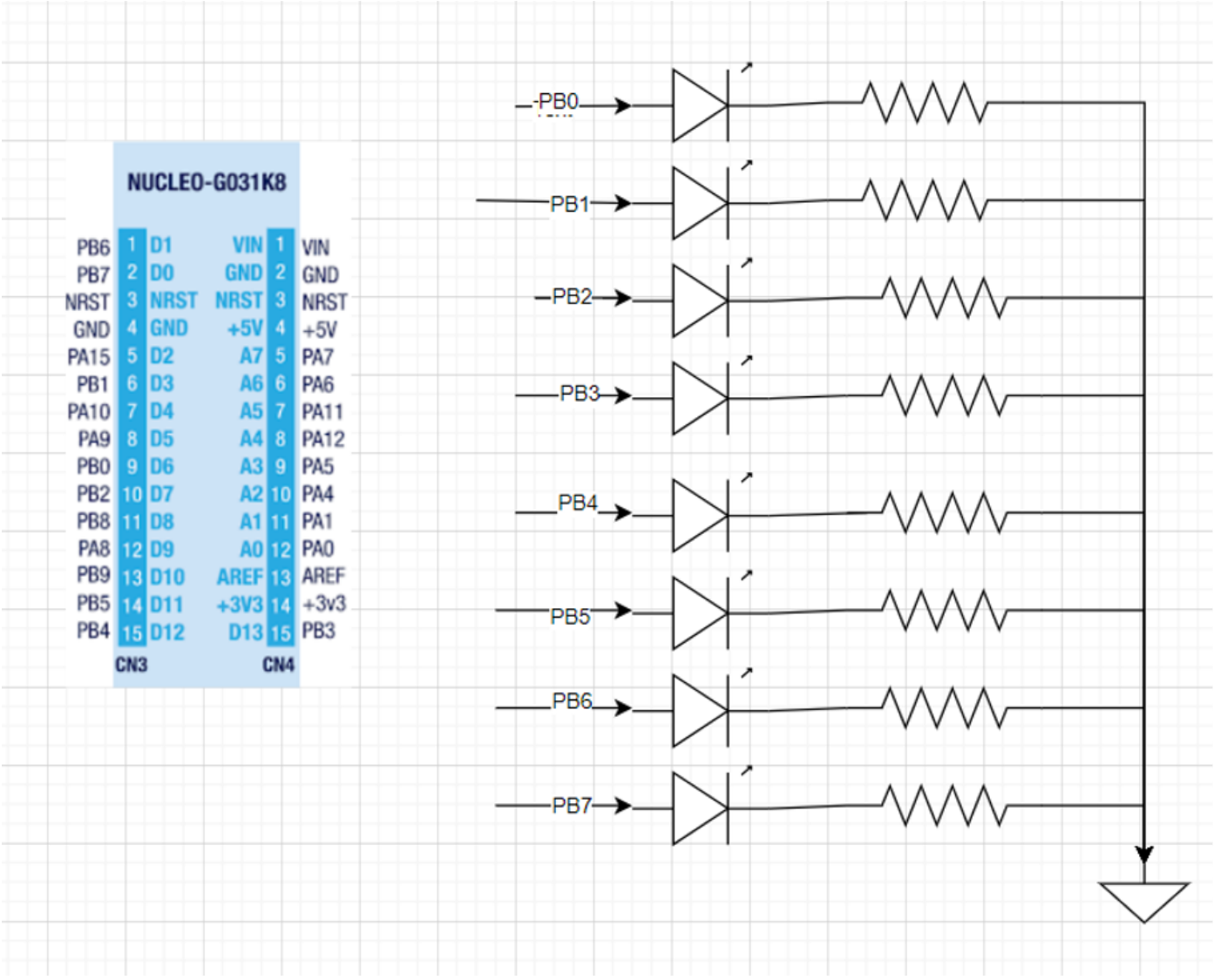

Figure 19: Schematic of ASSEMBLY_010_knight_rider

11.2 Flowchart

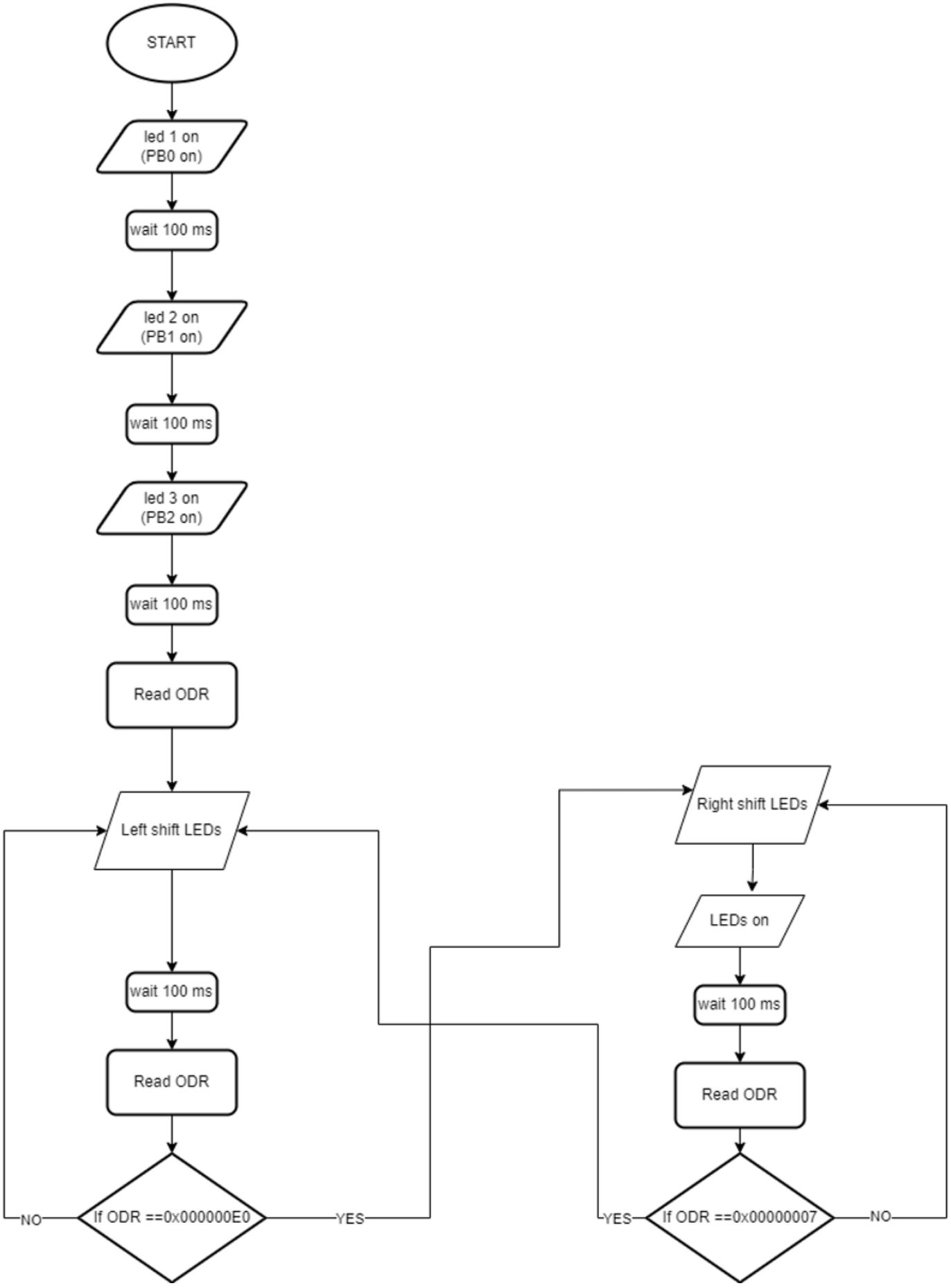

Figure 20: Flowchart of ASSEMBLY_010_knight_rider

12 REGC_001_ledblink_systick

As a software method we can record SysTick values before and after delay. Hence the difference between these values can be calculated. An oscilloscope can be used as a hardware method. Calculating the time difference between the toggled signal frequency at the start and end of the delay. -oscilloscope used for measure the frequencies-

12.1 Flowchart

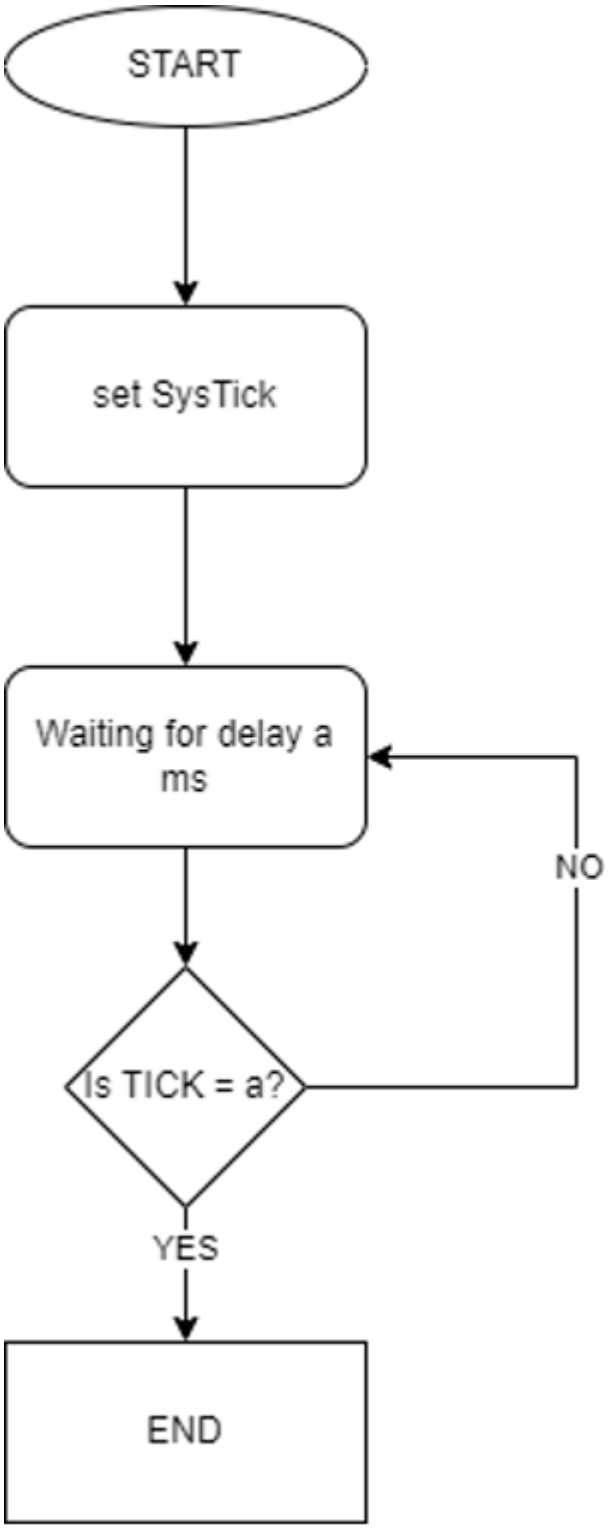

Figure 21: Flowchart of REGC_001_ledblink_systick

12.2 Some important registers

> I cant find it in the stm32g0 referance manuel or datacheet so i tahe these photoes from stm32f4 datasheet.

4.4.2 SysTick Reload Value Register

The SYST_RVR register specifies the start value to load into the SYST_CVR register. See the register summary in Table 4-32 on page 4-33 for its attributes. The bit assignments are:

Table 4-34 SYST_RVR register bit assignments

Bits	Name	Function
[31:24]	-	Reserved.
[23:0]	RELOAD	Value to load into the SYST_CVR register when the counter is enabled and when it reaches 0, see *Calculating the RELOAD value*.

Calculating the RELOAD value

The RELOAD value can be any value in the range `0x00000001-0x00FFFFFF`. A start value of 0 is possible, but has no effect because the SysTick exception request and COUNTFLAG are activated when counting from 1 to 0.

The RELOAD value is calculated according to its use. For example, to generate a multi-shot timer with a period of N processor clock cycles, use a RELOAD value of N-1. If the SysTick interrupt is required every 100 clock pulses, set RELOAD to 99.

Figure 22: Reload register

4.4.3 SysTick Current Value Register

The SYST_CVR register contains the current value of the SysTick counter. See the register summary in Table 4-32 on page 4-33 for its attributes. The bit assignments are:

Table 4-35 SYST_CVR register bit assignments

Bits	Name	Function
[31:24]	-	Reserved.
[23:0]	CURRENT	Reads return the current value of the SysTick counter. A write of any value clears the field to 0, and also clears the SYST_CSR COUNTFLAG bit to 0.

Figure 23: Current value register

4.4.1 SysTick Control and Status Register

The SysTick SYST_CSR register enables the SysTick features. The register resets to `0x00000000`, or to `0x00000004` if your device does not implement a reference clock. See the register summary in Table 4-32 for its attributes. The bit assignments are:

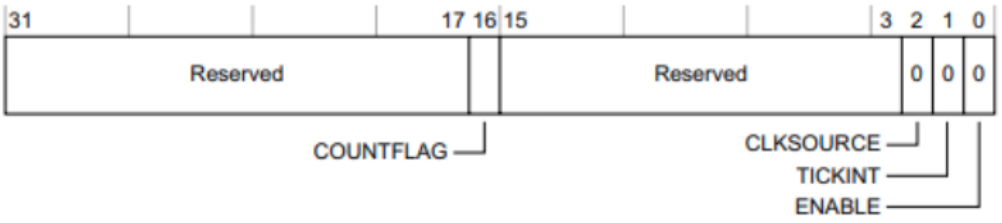

Table 4-33 SysTick SYST_CSR register bit assignments

Bits	Name	Function
[31:17]	-	Reserved.
[16]	COUNTFLAG	Returns 1 if timer counted to 0 since last time this was read.
[15:3]	-	Reserved.
[2]	CLKSOURCE	Indicates the clock source: 0 = external clock 1 = processor clock.
[1]	TICKINT	Enables SysTick exception request: 0 = counting down to zero does not assert the SysTick exception request 1 = counting down to zero asserts the SysTick exception request. Software can use COUNTFLAG to determine if SysTick has ever counted to zero.
[0]	ENABLE	Enables the counter: 0 = counter disabled 1 = counter enabled.

When ENABLE is set to 1, the counter loads the RELOAD value from the SYST_RVR register and then counts down. On reaching 0, it sets the COUNTFLAG to 1 and optionally asserts the SysTick depending on the value of TICKINT. It then loads the RELOAD value again, and begins counting.

Figure 24: Systick Control and Status Register

13 REGC_002_timer_interrupt

The TIM2 timer interrupt is used to change the state of the LED by tracking the time. The EXTI interrupt is used to increase or decrease the blinking rate of the LED when a button is pressed. As a result, it realizes an application that uses timer and external interrupts to control the LED to blink at a specific speed.

13.1 Flowchart

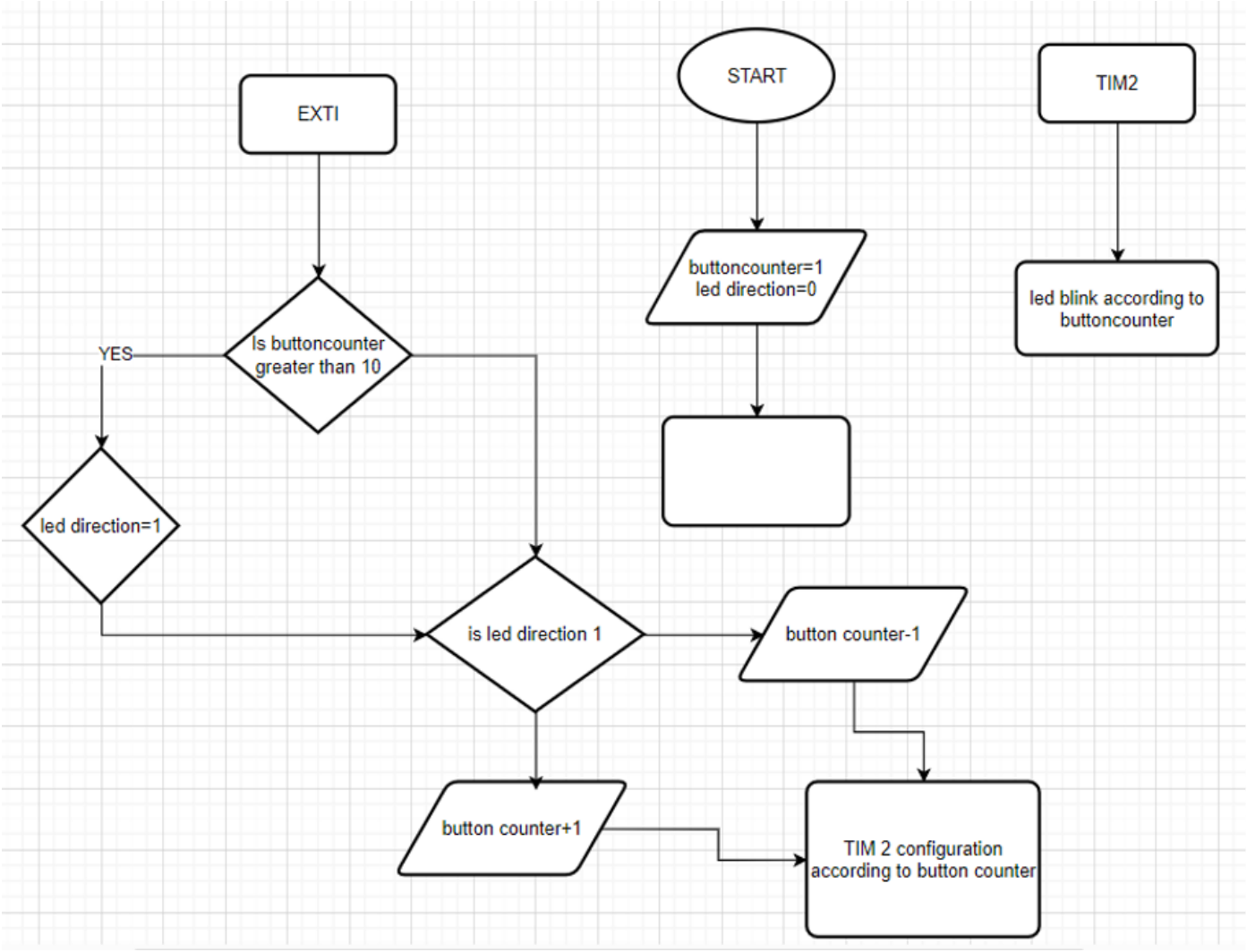

Figure 25: Flowchart of REGC_002_timer_interrupt

13.2 Schematic

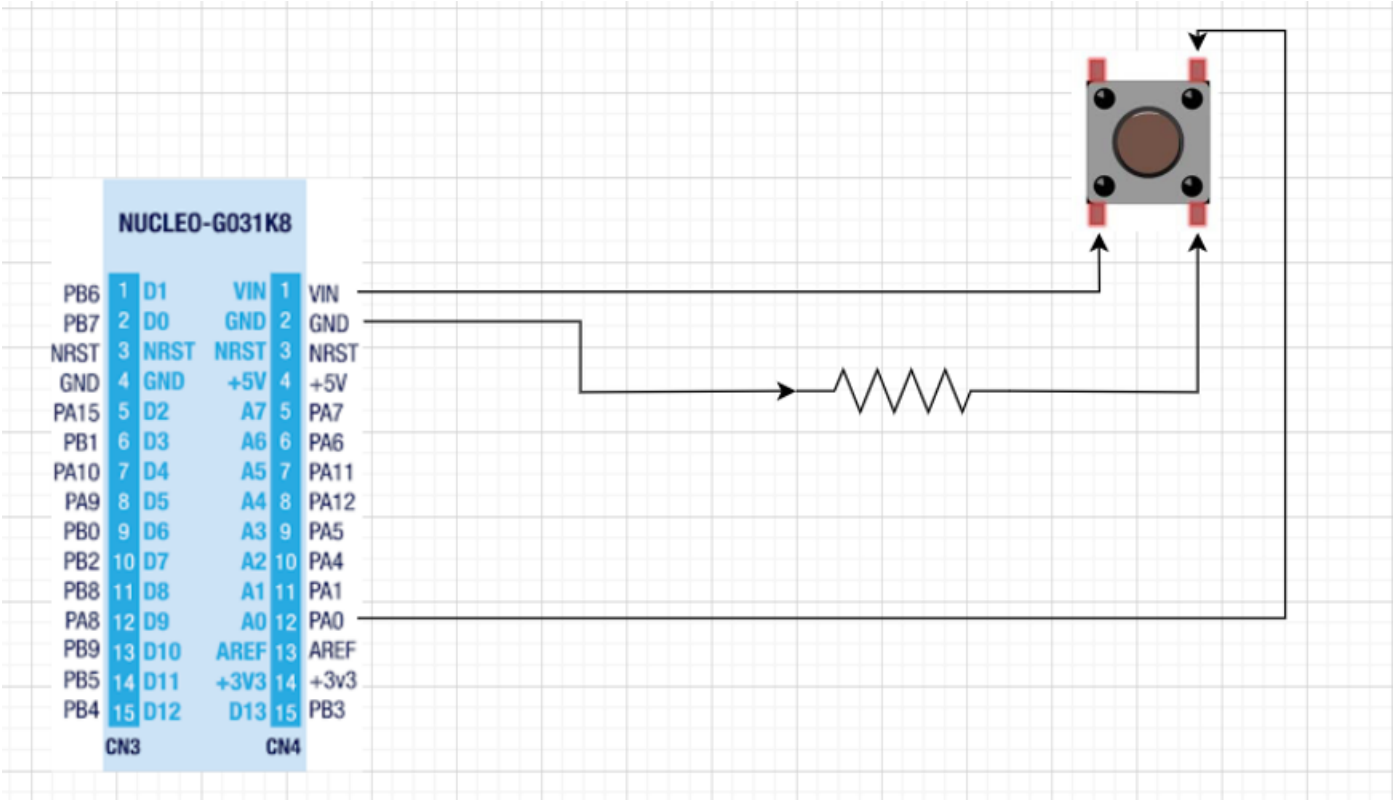

Figure 26: Schematic of REGC_002_timer_interrupt

13.3 Important registers

13.3.1 TIM2 registers

22.4.1 TIMx control register 1 (TIMx_CR1)(x = 2 to 4)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	UIFRE MAP	Res.	CKD[1:0]		ARPE	CMS[1:0]		DIR	OPM	URS	UDIS	CEN
				rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:12 Reserved, must be kept at reset value.

Bit 11 **UIFREMAP**: UIF status bit remapping
0: No remapping. UIF status bit is not copied to TIMx_CNT register bit 31.
1: Remapping enabled. UIF status bit is copied to TIMx_CNT register bit 31.

Bit 10 Reserved, must be kept at reset value.

Bits 9:8 **CKD[1:0]**: Clock division
This bit-field indicates the division ratio between the timer clock (CK_INT) frequency and sampling clock used by the digital filters (ETR, TIx),
00: $t_{DTS} = t_{CK\_INT}$
01: $t_{DTS} = 2 \times t_{CK\_INT}$
10: $t_{DTS} = 4 \times t_{CK\_INT}$
11: Reserved

Bit 7 **ARPE**: Auto-reload preload enable
0: TIMx_ARR register is not buffered
1: TIMx_ARR register is buffered

Bits 6:5 **CMS[1:0]**: Center-aligned mode selection
00: Edge-aligned mode. The counter counts up or down depending on the direction bit (DIR).
01: Center-aligned mode 1. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx_CCMRx register) are set only when the counter is counting down.
10: Center-aligned mode 2. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx_CCMRx register) are set only when the counter is counting up.
11: Center-aligned mode 3. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx_CCMRx register) are set both when the counter is counting up or down.

Note: It is not allowed to switch from edge-aligned mode to center-aligned mode as long as the counter is enabled (CEN=1)

Bit 4 **DIR**: Direction
0: Counter used as upcounter
1: Counter used as downcounter

Note: This bit is read only when the timer is configured in Center-aligned mode or Encoder mode.

Figure 27: Control register 1 of TIMx

This register is used in the code to enablle or disable timer.To understand the usage you can take a look to **Start_TIM2(void)** function.

22.4.14 TIMx prescaler (TIMx_PSC)(x = 2 to 4)

Address offset: 0x28

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSC[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency CK_CNT is equal to f_{CK_PSC} / (PSC[15:0] + 1).
PSC contains the value to be loaded in the active prescaler register at each update event (including when the counter is cleared through UG bit of TIMx_EGR register or through trigger controller when configured in “reset mode”).

Figure 28: TIMx Prescaler register

22.4.12 TIMx counter [alternate] (TIMx_CNT)(x = 2 to 4)

Bit 31 of this register has two possible definitions depending on the value of UIFREMAP in TIMx_CR1 register:

- This section is for UIFREMAP = 0
- Next section is for UIFREMAP = 1

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNT[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 **CNT[31:16]**: Most significant part counter value (TIM2)

Bits 15:0 **CNT[15:0]**: Least significant part of counter value

Figure 28: TIMx Counter register (CNT)

22.4.15 TIMx auto-reload register (TIMx_ARR)(x = 2 to 4)

Address offset: 0x2C

Reset value: 0xFFFF FFFF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ARR[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 **ARR[31:16]**: High auto-reload value (TIM2)

Bits 15:0 **ARR[15:0]**: Low Auto-reload value

ARR is the value to be loaded in the actual auto-reload register.

Refer to the *Section 22.3.1: Time-base unit on page 627* for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is null.

Figure 29: TIMx auto reload register

22.4.4 TIMx DMA/Interrupt enable register (TIMx_DIER)(x = 2 to 4)

Address offset: 0x0C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TDE	Res.	CC4DE	CC3DE	CC2DE	CC1DE	UDE	Res.	TIE	Res.	CC4IE	CC3IE	CC2IE	CC1IE	UIE
	rw		rw	rw	rw	rw	rw		rw		rw	rw	rw	rw	rw

Bit 15 Reserved, must be kept at reset value.

Bit 14 **TDE**: Trigger DMA request enable

0: Trigger DMA request disabled.

1: Trigger DMA request enabled.

Bit 13 Reserved, must be kept at reset value.

Bit 12 **CC4DE**: Capture/Compare 4 DMA request enable
0: CC4 DMA request disabled.
1: CC4 DMA request enabled.

Bit 11 **CC3DE**: Capture/Compare 3 DMA request enable
0: CC3 DMA request disabled.
1: CC3 DMA request enabled.

Bit 10 **CC2DE**: Capture/Compare 2 DMA request enable
0: CC2 DMA request disabled.
1: CC2 DMA request enabled.

Bit 9 **CC1DE**: Capture/Compare 1 DMA request enable
0: CC1 DMA request disabled.
1: CC1 DMA request enabled.

Bit 8 **UDE**: Update DMA request enable
0: Update DMA request disabled.
1: Update DMA request enabled.

Bit 7 Reserved, must be kept at reset value.

Bit 6 **TIE**: Trigger interrupt enable
0: Trigger interrupt disabled.
1: Trigger interrupt enabled.

Bit 5 Reserved, must be kept at reset value.

Bit 4 **CC4IE**: Capture/Compare 4 interrupt enable
0: CC4 interrupt disabled.
1: CC4 interrupt enabled.

Bit 3 **CC3IE**: Capture/Compare 3 interrupt enable
0: CC3 interrupt disabled.
1: CC3 interrupt enabled.

Bit 2 **CC2IE**: Capture/Compare 2 interrupt enable
0: CC2 interrupt disabled.
1: CC2 interrupt enabled.

Bit 1 **CC1IE**: Capture/Compare 1 interrupt enable
0: CC1 interrupt disabled.
1: CC1 interrupt enabled.

Bit 0 **UIE**: Update interrupt enable
0: Update interrupt disabled.
1: Update interrupt enabled.

Figure 29: TIMx Interrupt enable register (DIER)

It is used to update interrupt enable.

13.3.2 EXTI registers

EXTI->EXTICR[1]

13.5.11 EXTI external interrupt selection register (EXTI_EXTICRx)

Address offset: 0x060 + 0x4 * (x - 1), (x = 1 to 4)

Reset value: 0x0000 0000

EXTIm fields contain only the number of bits in line with the nb_ioport configuration.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EXTIm+3[7:0]								EXTIm+2[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTIm+1[7:0]								EXTIm[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 23:16 **EXTIm+2[7:0]:** EXTIm+2 GPIO port selection (m = 4 * (x - 1))
These bits are written by software to select the source input for EXTIm+2 external interrupt.
0x00: PA[m+2] pin
0x01: PB[m+2] pin
0x02: PC[m+2] pin
0x03: PD[m+2] pin
0x04: reserved
0x05: PF[m+2] pin
Others reserved

Bits 15:8 **EXTIm+1[7:0]:** EXTIm+1 GPIO port selection (m = 4 * (x - 1))
These bits are written by software to select the source input for EXTIm+1 external interrupt.
0x00: PA[m+1] pin
0x01: PB[m+1] pin
0x02: PC[m+1] pin
0x03: PD[m+1] pin
0x04: reserved
0x05: PF[m+1] pin
Others reserved

Bits 7:0 **EXTIm[7:0]:** EXTIm GPIO port selection (m = 4 * (x - 1))
These bits are written by software to select the source input for EXTIm external interrupt.
0x00: PA[m] pin
0x01: PB[m] pin
0x02: PC[m] pin
0x03: PD[m] pin
0x04: reserved
0x05: PF[m] pin
Others reserved

Figure 30: Extenal interrupt control register (EXTICR)

- **EXTICR (External Interrupt Configuration Register)** : This register is used to select the GPIO port that is connected to the EXTI line. The STM32 microcontrollers have multiple EXTICR registers (EXTICR1, EXTICR2, etc.), each controlling a group of EXTI lines.
- `EXTI_EXTICR1_EXTI0_0` : This specific bit setting in the `EXTICR[1]` register connects EXTI line 0 to GPIO port A. The notation `[1]` seems to be a mistake here since EXTI0 configuration should be in `EXTICR[0]` (assuming EXTICR registers are zero-indexed in your environment). The correct approach to connect EXTI1 to GPIOA Pin 0 would typically involve `EXTICR[0]` and setting the appropriate bits for GPIOA.

EXTI->RTSR1

Figure 31: Extenal interrupt rising trigger selection register (EXTI_RTSR1)

- **RTSR1 (Rising Trigger Selection Register 1)** : This register is used to enable rising edge triggers on the EXTI lines. Setting a bit in this register selects the corresponding EXTI line to be sensitive to rising edges.
- `EXTI_RTSR1_RT0` : Setting this bit enables EXTI line 0 to trigger an interrupt request on a rising edge.

EXTI->FTSR1

13.5.2 EXTI falling trigger selection register 1 (EXTI_FTSR1)

Address offset: 0x004

Reset value: 0x0000 0000

Contains only register bits for configurable events.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	FT20	Res.	FT18	FT17	FT16
											rw		rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FT15	FT14	FT13	FT12	FT11	FT10	FT9	FT8	FT7	FT6	FT5	FT4	FT3	FT2	FT1	FT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:21 Reserved, must be kept at reset value.

Bit 20 **FT20:** Falling trigger event configuration bit of configurable line 20(1).
This bit enables/disables the falling edge trigger for the event and interrupt on the corresponding line.
0: Disable
1: Enable
The FT20 bit is only available in STM32G0B1xx and STM32G0C1xx. It is reserved in all the other devices.

Bit 19 Reserved, must be kept at reset value.

Bits 18:0 **FTx:** Falling trigger event configuration bit of configurable line x (x = 18 to 0)(1).
Each bit enables/disables the falling edge trigger for the event and interrupt on the corresponding line.
0: Disable
1: Enable
The FT18 and FT17 bits are only available in STM32G071xx and STM32G081xx as well as STM32G0B1xx and STM32G0C1xx. They are reserved in STM32G031xx and STM32G041xx as well as STM32G051xx and STM32G061xx.

1. The configurable lines are edge triggered, no glitch must be generated on these inputs.
If a falling edge on the configurable line occurs during writing of the register, the associated pending bit is not set.
Falling edge trigger can be set for a line with rising edge trigger enabled. In this case, both edges generate a trigger.

Figure 32: Extenal interrupt falling trigger selection register (EXTI_FTSR1)

- **FTSR1 (Falling Trigger Selection Register 1)** : Similar to RTSR1, but for falling edge triggers.
- `EXTI_FTSR1_FT0` : Clearing this bit (`&= ~EXTI_FTSR1_FT0`) ensures that EXTI line 0 will not trigger an interrupt request on a falling edge, making it sensitive to rising edges onlyEXTI->IMR1
- **IMR1 (Interrupt Mask Register 1)** : This register is used to enable or disable interrupt requests from the EXTI lines.
- `EXTI_IMR1_IM0` : Setting this bit enables interrupt requests from EXTI line 0. This means that when the selected edge (rising, in this case) is detected on EXTI line 0, an interrupt request will be generated.

13.4 Frequency Calculation

- **PSC (Prescaler Register)** : This register determines the division factor applied to the timer's clock. By slowing down the clock, the timer can count at a slower rate, allowing for longer intervals.
- **CNT (Counter Register)** : Acts as the core of the timer, incrementing on each clock cycle. When it reaches the value in the ARR, an event can be triggered.
- **ARR (Auto-Reload Register)** : Specifies the value to which the CNT register resets after reaching its maximum count. This defines the period of the timer's cycle.

The frequency of the timer can be calculated using the formula:

`Frequency = Clock / ((PSC + 1) * (ARR ))`.

Adjusting the PSC and ARR values allows for fine-tuning of the timer's frequency for various applications.

The clock i 16 Mhz for this application.

14 REGC_003_seven_segment

In this problem, pushbutton control was provided on the seven segment display using external interrupts. When the button is pressed, the led on and the display starts to increase starting from 0, when it reaches 999, it stops and the led off. If the button is pressed again during the process, the display will reset and start over.

14.1 Flowchart

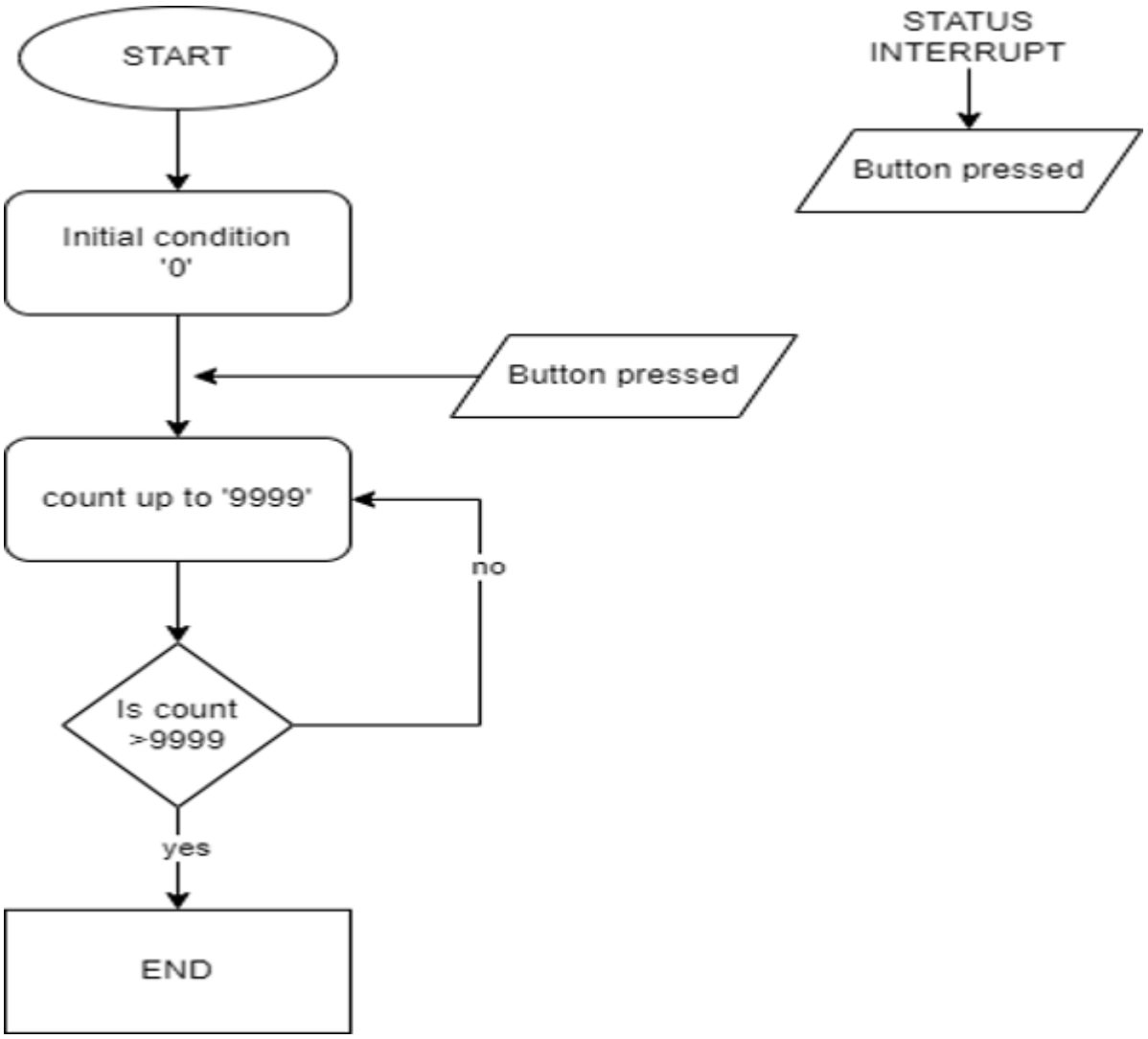

Figure 33: Flowchart of REGC_003_seven_segment

14.2 Schematic

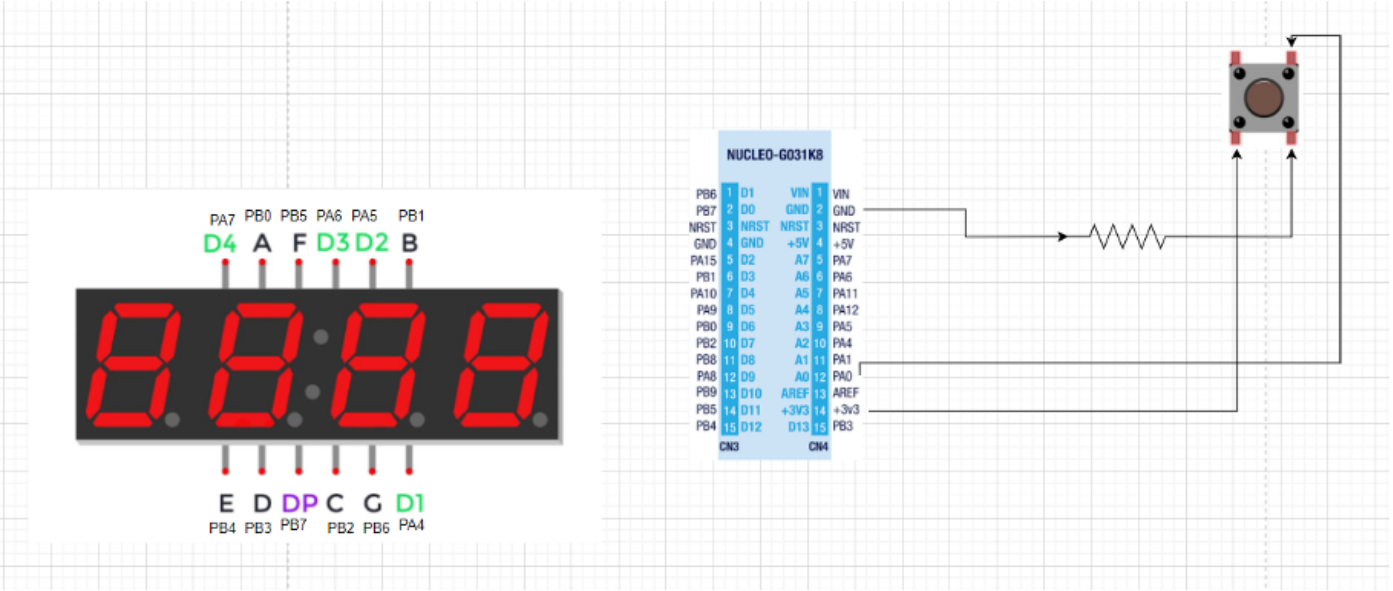

Figure 34: Schematic of REGC_003_seven_segment

15 REGC_004_ledblink_watchdog

For the independent watchdog timer, the time is set according to the following parameters and the blinking of the led is tested. If the set time is exceeded and the watchdog is not refreshed, the system will be reset.

15.1 Important registers

28.4.1 IWDG key register (IWDG_KR)

Address offset: 0x00

Reset value: 0x0000 0000 (reset by Standby mode)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
KEY[15:0]															
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **KEY[15:0]:** Key value (write only, read 0x0000)

These bits must be written by software at regular intervals with the key value 0xAAAA, otherwise the watchdog generates a reset when the counter reaches 0.
Writing the key value 0x5555 to enable access to the IWDG_PR, IWDG_RLR and IWDG_WINR registers (see *Section 28.3.4: Register access protection*)
Writing the key value 0xCCCC starts the watchdog (except if the hardware watchdog option is selected)

Figure 35: IWDG key Register (IWDG_KR)

28.4.2 IWDG prescaler register (IWDG_PR)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PR[2:0]		
													rw	rw	rw

Bits 31:3 Reserved, must be kept at reset value.

Bits 2:0 **PR[2:0]:** Prescaler divider

These bits are write access protected see *Section 28.3.4: Register access protection*. They are written by software to select the prescaler divider feeding the counter clock. PVU bit of the *IWDG status register (IWDG_SR)* must be reset in order to be able to change the prescaler divider.

000: divider /4
001: divider /8
010: divider /16
011: divider /32
100: divider /64
101: divider /128
110: divider /256
111: divider /256

Note: Reading this register returns the prescaler value from the V_{DD} voltage domain. This value may not be up to date/valid if a write operation to this register is ongoing. For this reason the value read from this register is valid only when the PVU bit in the IWDG status register (IWDG_SR) is reset.

Figure 36: IWDG Prescaler Register (IWDG_PR)

28.4.3 IWDG reload register (IWDG_RLR)

Address offset: 0x08

Reset value: 0x0000 0FFF (reset by Standby mode)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	RL[11:0]											
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:12 Reserved, must be kept at reset value.

Bits 11:0 **RL[11:0]:** Watchdog counter reload value

These bits are write access protected see *Register access protection*. They are written by software to define the value to be loaded in the watchdog counter each time the value 0xAAAA is written in the *IWDG key register (IWDG_KR)*. The watchdog counter counts down from this value. The timeout period is a function of this value and the clock prescaler. Refer to the datasheet for the timeout information.
The RVU bit in the *IWDG status register (IWDG_SR)* must be reset to be able to change the reload value.

Note: Reading this register returns the reload value from the V_{DD} voltage domain. This value may not be up to date/valid if a write operation to this register is ongoing on it. For this reason the value read from this register is valid only when the RVU bit in the IWDG status register (IWDG_SR) is reset.

Figure 37: IWDG Reload Register (IWDG_RLR)

15.2 Independent Watchdog time calculation

- Setting IWDG time base:
  - IWDG time base prescaled from LSI clock (32 kHz)
    - 7 pre-dividers: 4 to 256 selectable by IWDG_PR register (and 12-bit watchdog counter reload value, RLR[11:0])
  - Setting the IWDG timeout by using the following formula:

$$t_{IWDG} = t_{LSI} \times 4 \times 2^{PR} \times (RL + 1)$$

where t_{LSI} = 1/32000 = 31.25 µs, PR and RL are fields of IWDG registers

Figure 38: time calculation

15.3 Flowchart

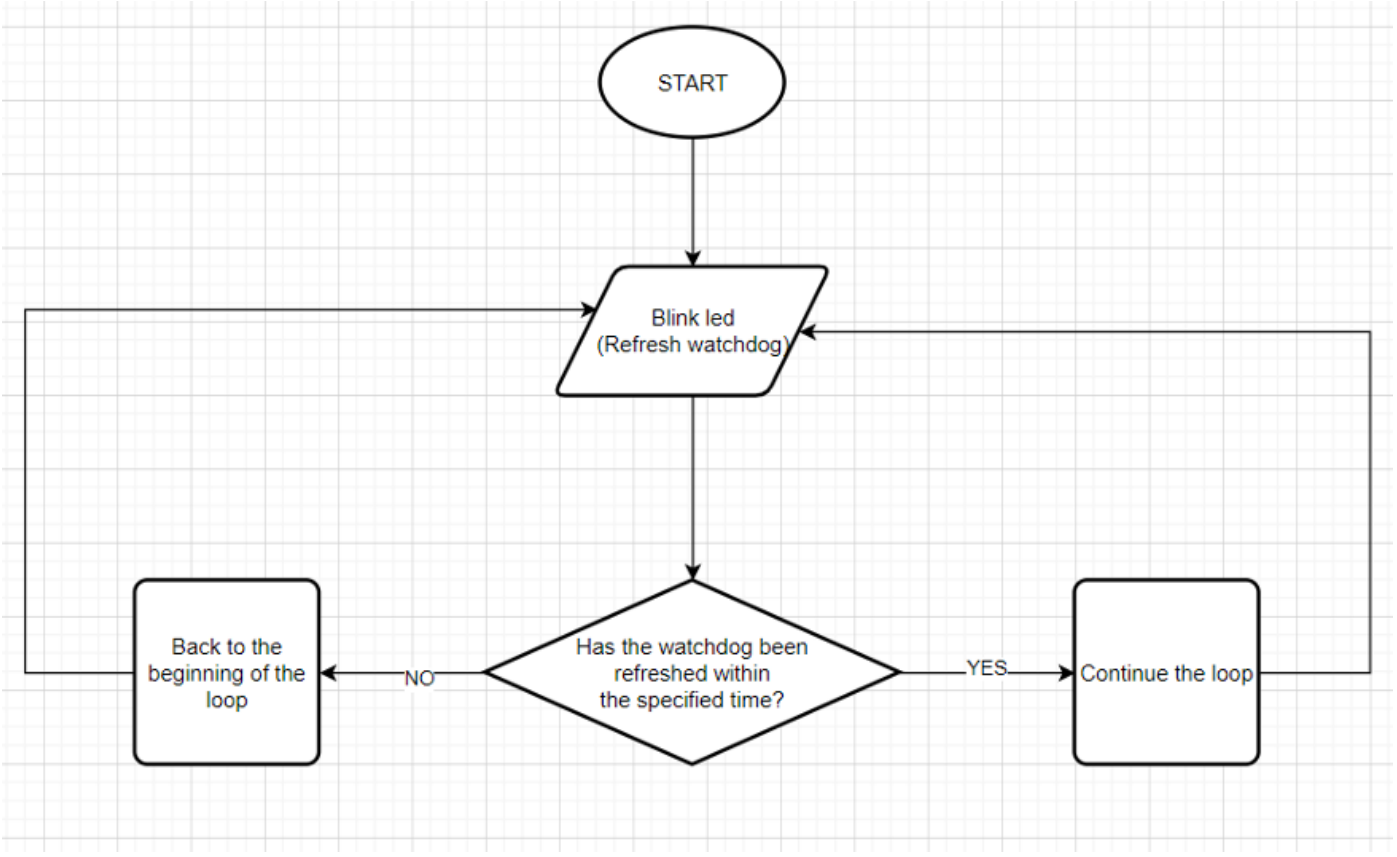

Figure 39: flowchart of REGC_004_ledblink_watchdog

16 REGC_005_seven_segment_with_watchdog

The system was tested by adding watchdog timer while working as in REGC_003_seven_segment . If the watchdog timer time is made 16 s and watchdog refresh is not done within this time, the screen is reset and starts counting from the beginning.

16.1 Flowchart

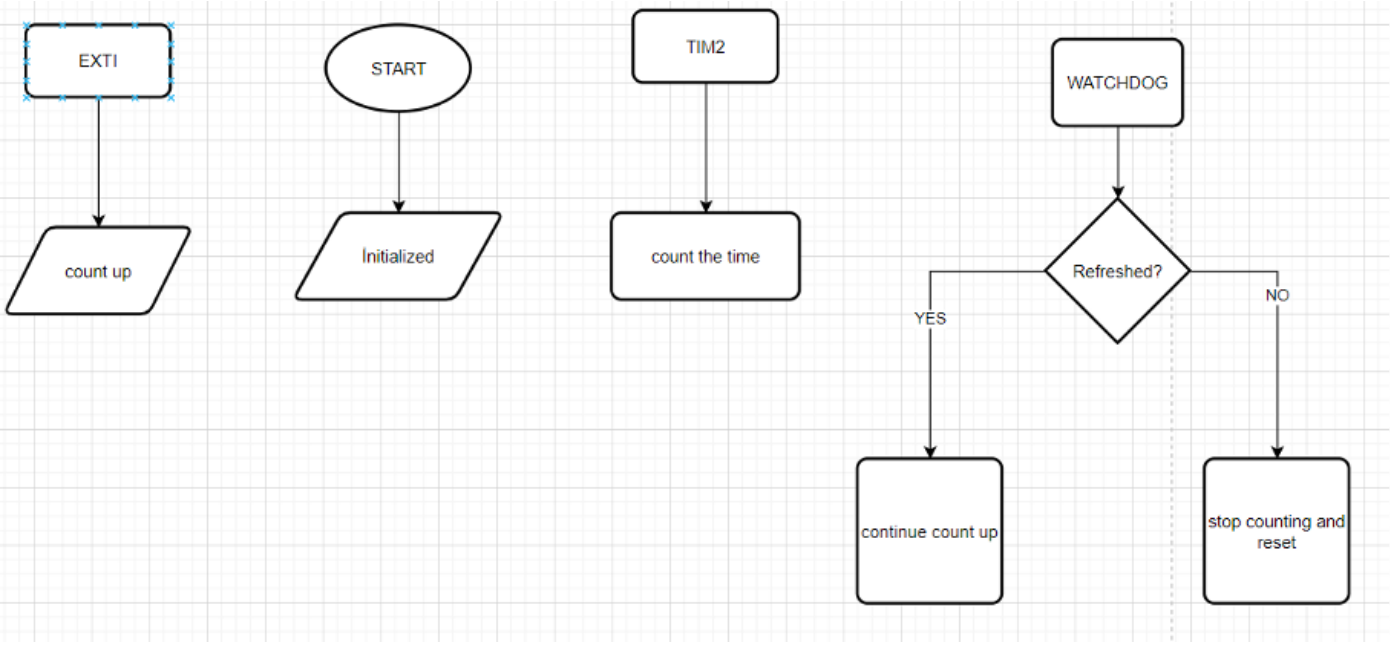

Figure 40: flowchart of REGC_005_seven_segment_with_watchdog

16.2 Schematic

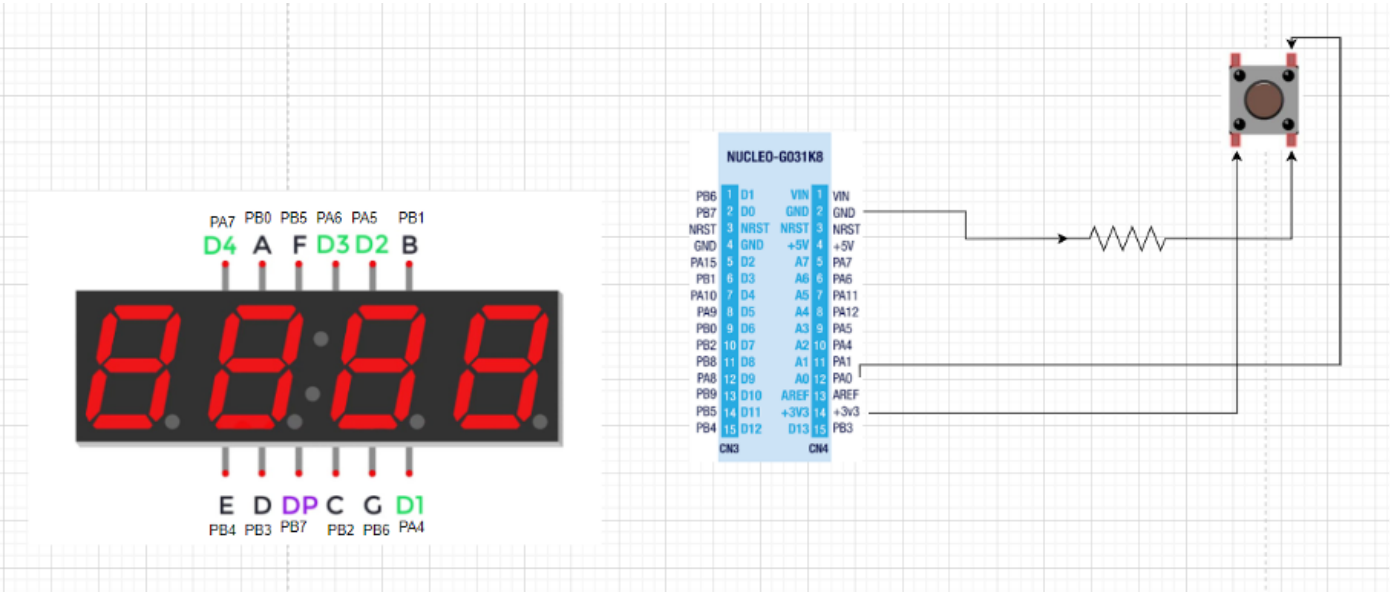

Figure 41: Schematic of REGC_005_seven_segment_with_watchdog

17 REGC_006_uart

The aim of this project is creating the functions of transmitting and receiving data.

When we observe the code, first GPIOA, USART2 module clocks and PA2, PA3 pins are enabled from RCC. Then PA2 and PA3 pins settled as alternate function for the USART2 module. CR1 control register choose from datasheet at enabling receive and transmit

from USART2 module. TDR and RDR registers used at data transmit and receive processes respectively. Baud rate settled to 9600 bps assuming clock is running at 16Mhz. Value chosen according to the formula of baud rate and EISA RS-232 communication Standard.

Baud Rate = APBxCLK / USARTx

```
Some of the standard baud rates: 300, 600,
```

1200, 2400, 4800, 9600, 14400, 19200, 38400, 57600, 115200, 230400, 460800.

17.1 Flowchart

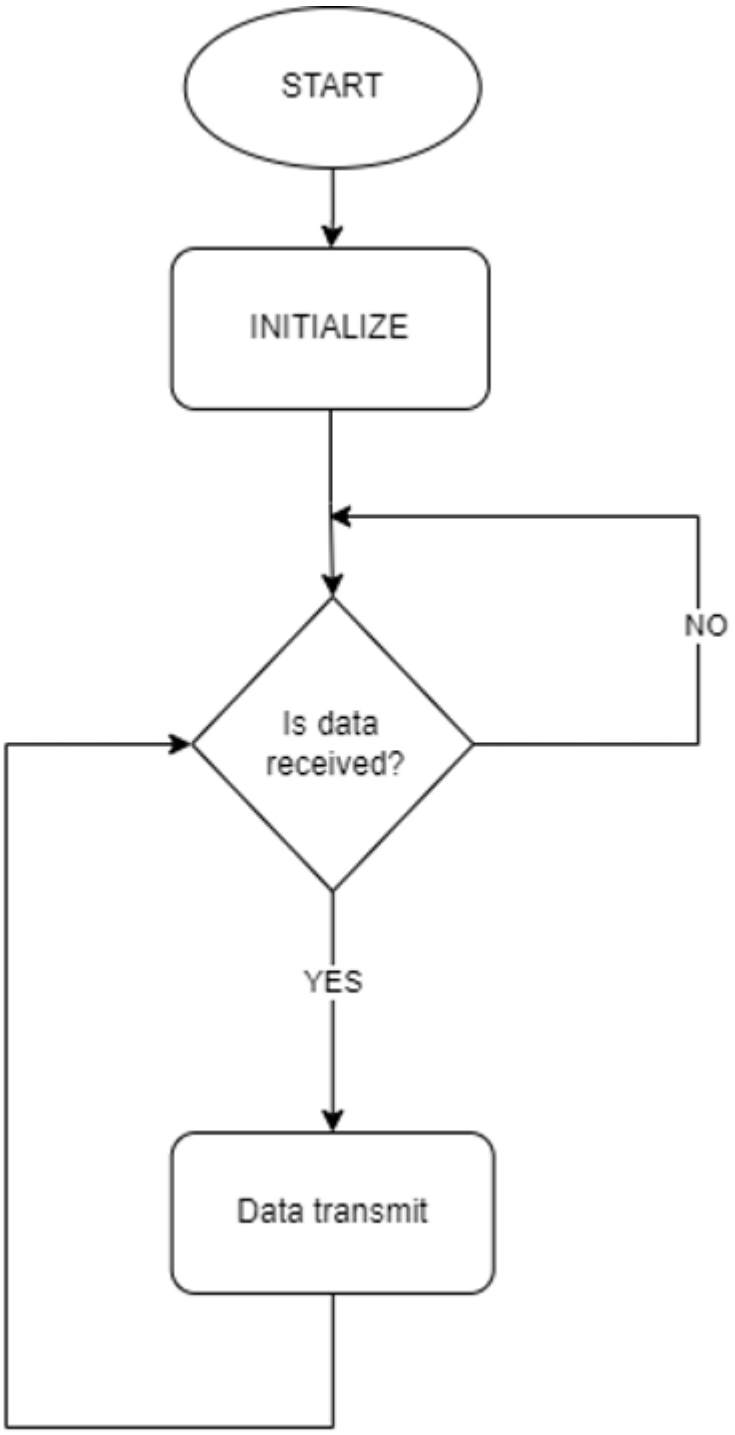

Figure 42: Flowchart of REGC_006_uart

18 REGC_007_pwm

Pulse Width Modulation is a technique used to encode information in the duration of a pulse signal. By changing the duty cycle of the PWM signal LED brightness can be changed.

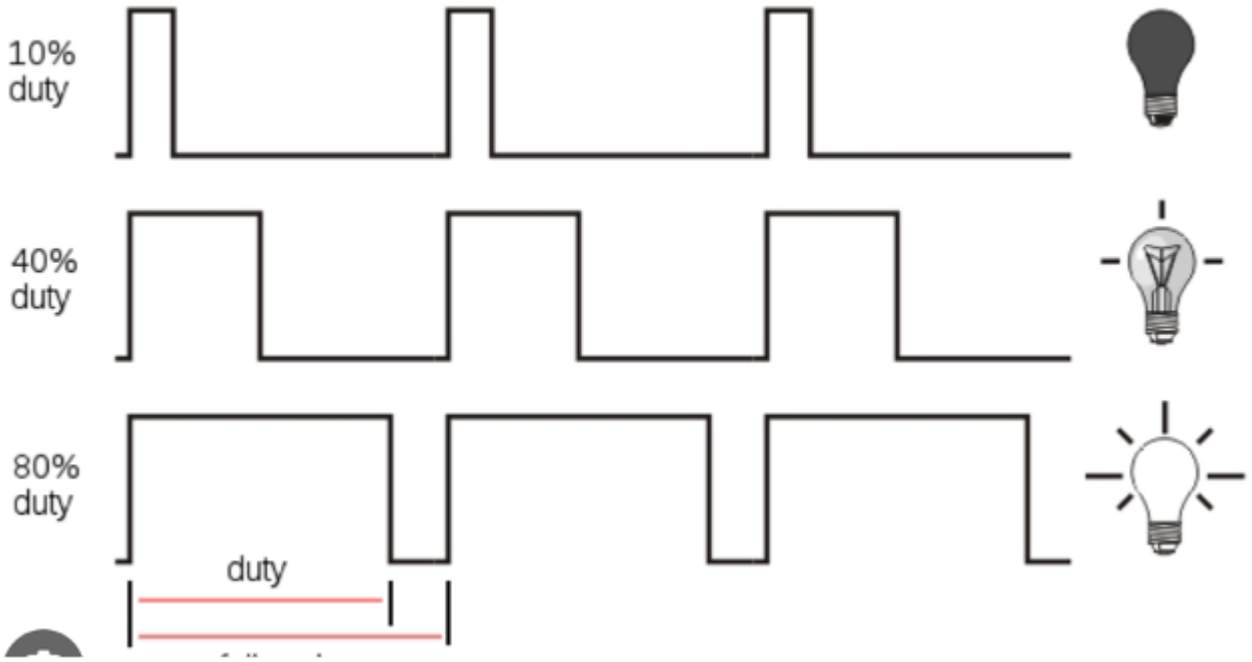

Figure 43: Duty cycle led brightness relationship

PWM involves rapidly toggling a digital signal between high and low states. The frequency of the PWM signal can calculated by the above formula:

$$PWM\ frequency = \frac{APB\ clock\ frequency}{ARR * (PSC + 1)}$$

Figure 44: PWM frequency formula

To set the PWM frequency to 1 Khz, because the clock is default 16Mhz, PSC register value is set as 15, and ARR register value as 1000. To set the duty cycle the CCR register value can changed. The duty cycle percentage can be calculated by the above formula:

$$Duty\ Cycle = \frac{CCR}{ARR} * 100\ \%$$

Figure 44: PWM Duty cycle formula

So as an example, to set duty cycle to 40% CCR value must be equal to 400.

TIM2 is used to generate the PWM signal, and the interrupt is activated. In the interrupt the duty cycle of the PWM signal is updated. By using the Systick timer a counter is increased and decreased. The counter is set as PWM duty cycle in the TIM2 interrupt. So, every 1ms the counter increase by 1, this is equal to 0.1% of duty cycle.

18.1 Flowchart

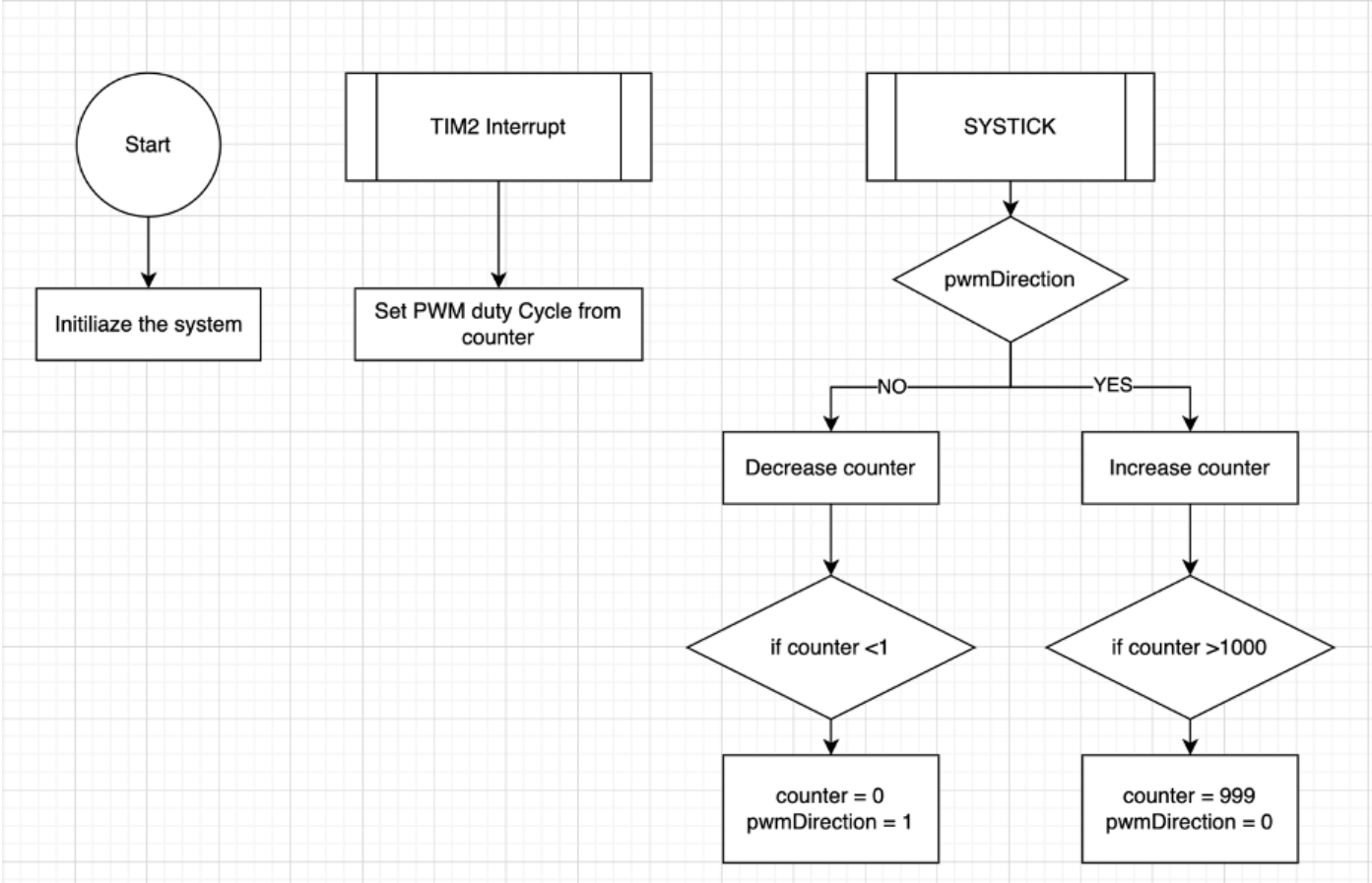

Figure 45: Flowchart of REGC_007_pwm

18.2 Schematic

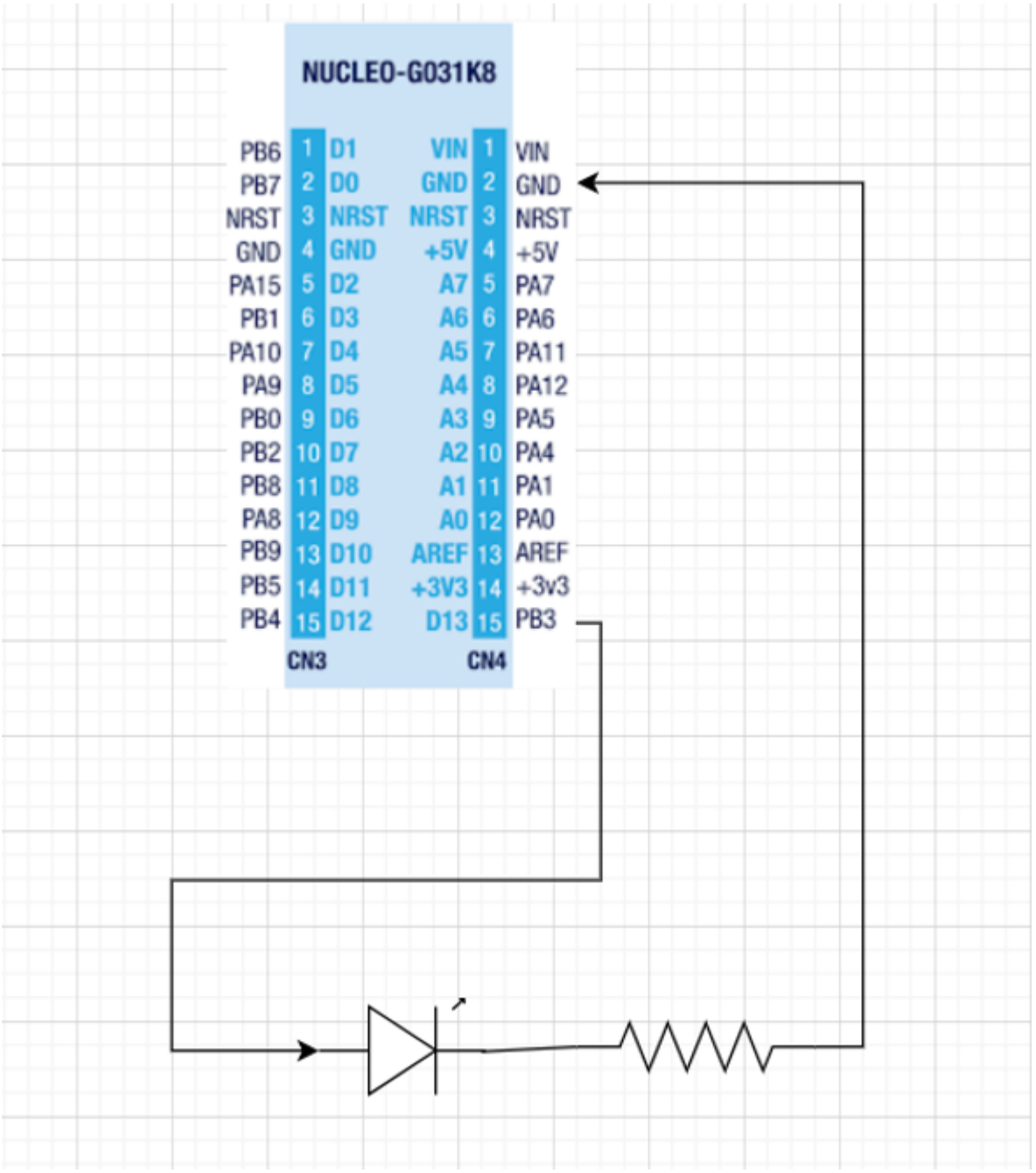

Figure 46: Schematic of REGC_007_pwm

19 REGC_008_withkeypad

The columns of the keypad were set as output and rows as input. Columns were connected to EXTI and keypad keys were controlled. First the column was determined from the incoming interrupt, then the rows were controlled and the key pressed was determined. Significant numbers entered were stored until "# came. Significant values between 0 and 100 affected the duty cycle, while other meaningless values and signs gave a warning. PWM was controlled with TIM2, UART was controlled with TIM3.

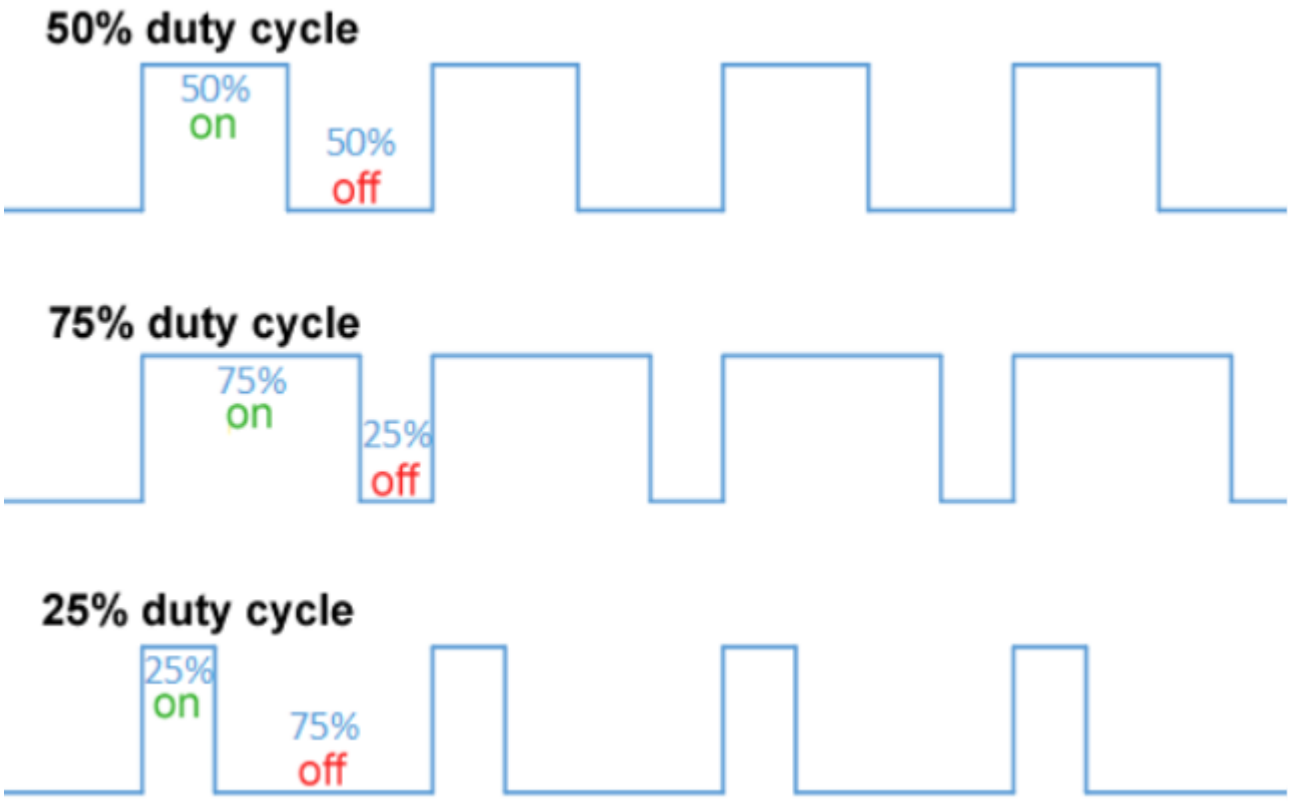

Figure 47 : Duty cycle of pwm signal

The internal structure of 4X4 KEYPAD MODULE is shown below.

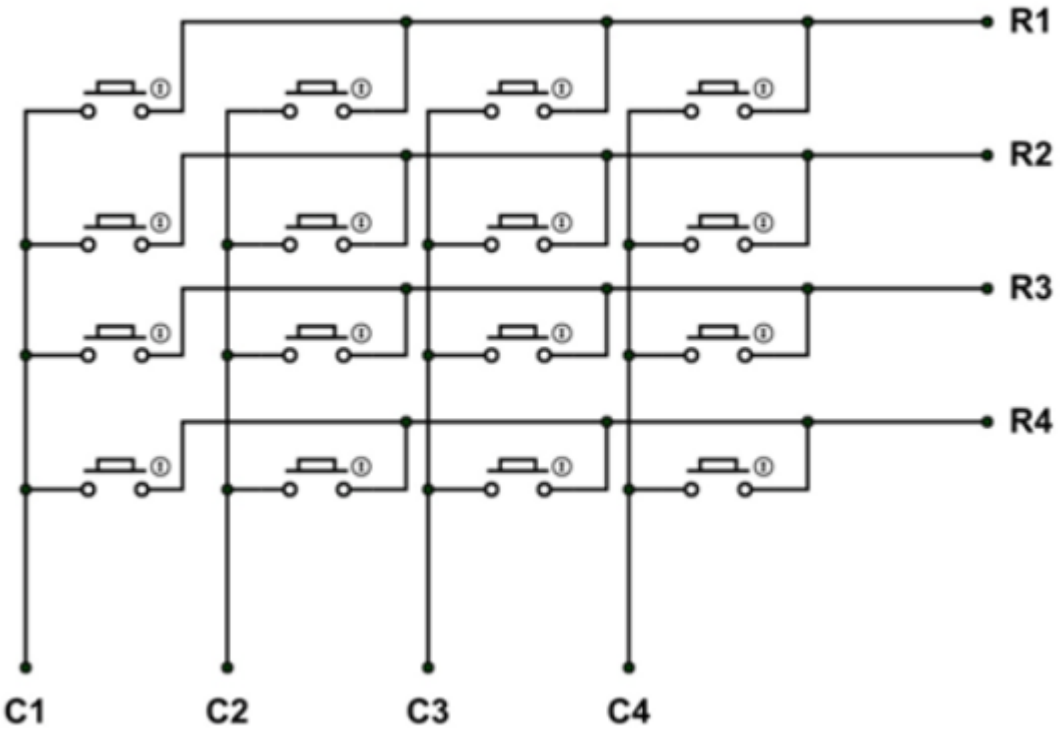

Figure 48 : Keypad circuit model

18.1 Flowchart

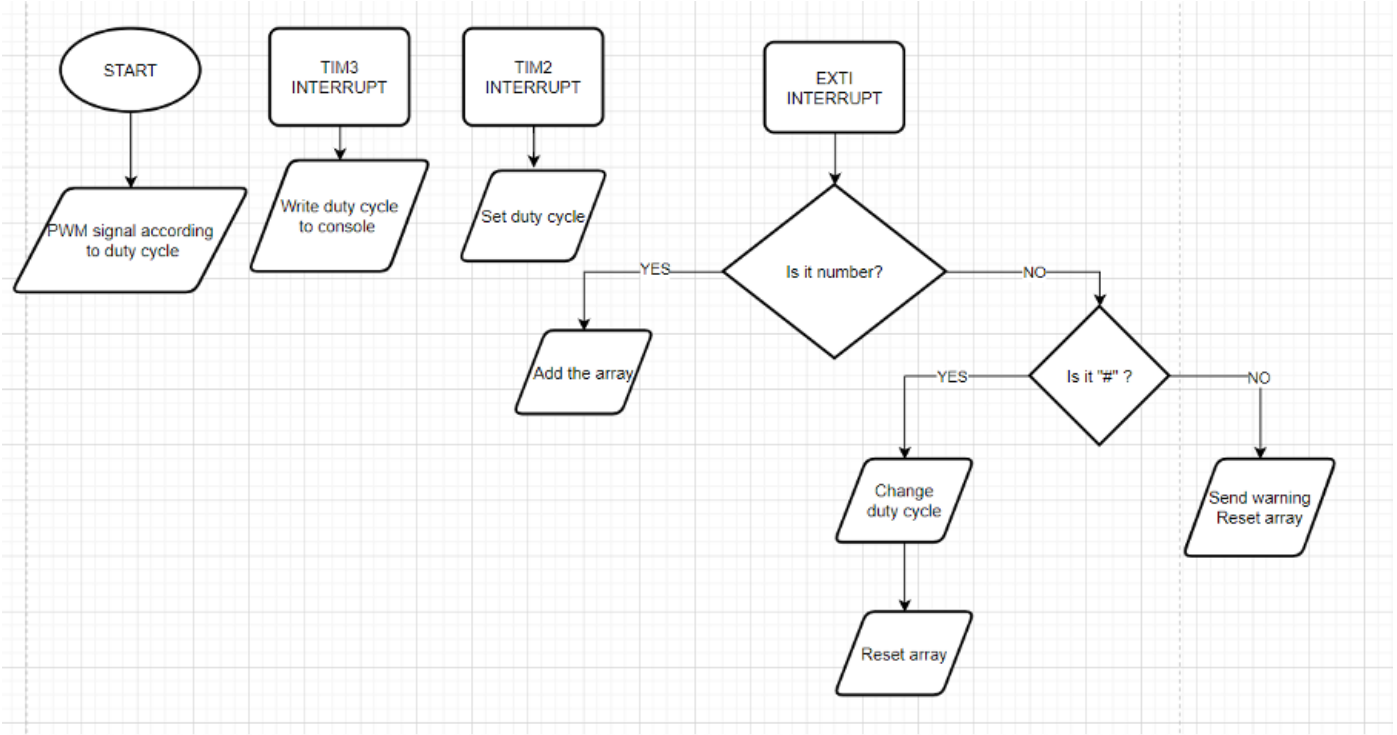

Figure 49 : Flowchart of REGC_008_withkeypad

18.2 Schematic

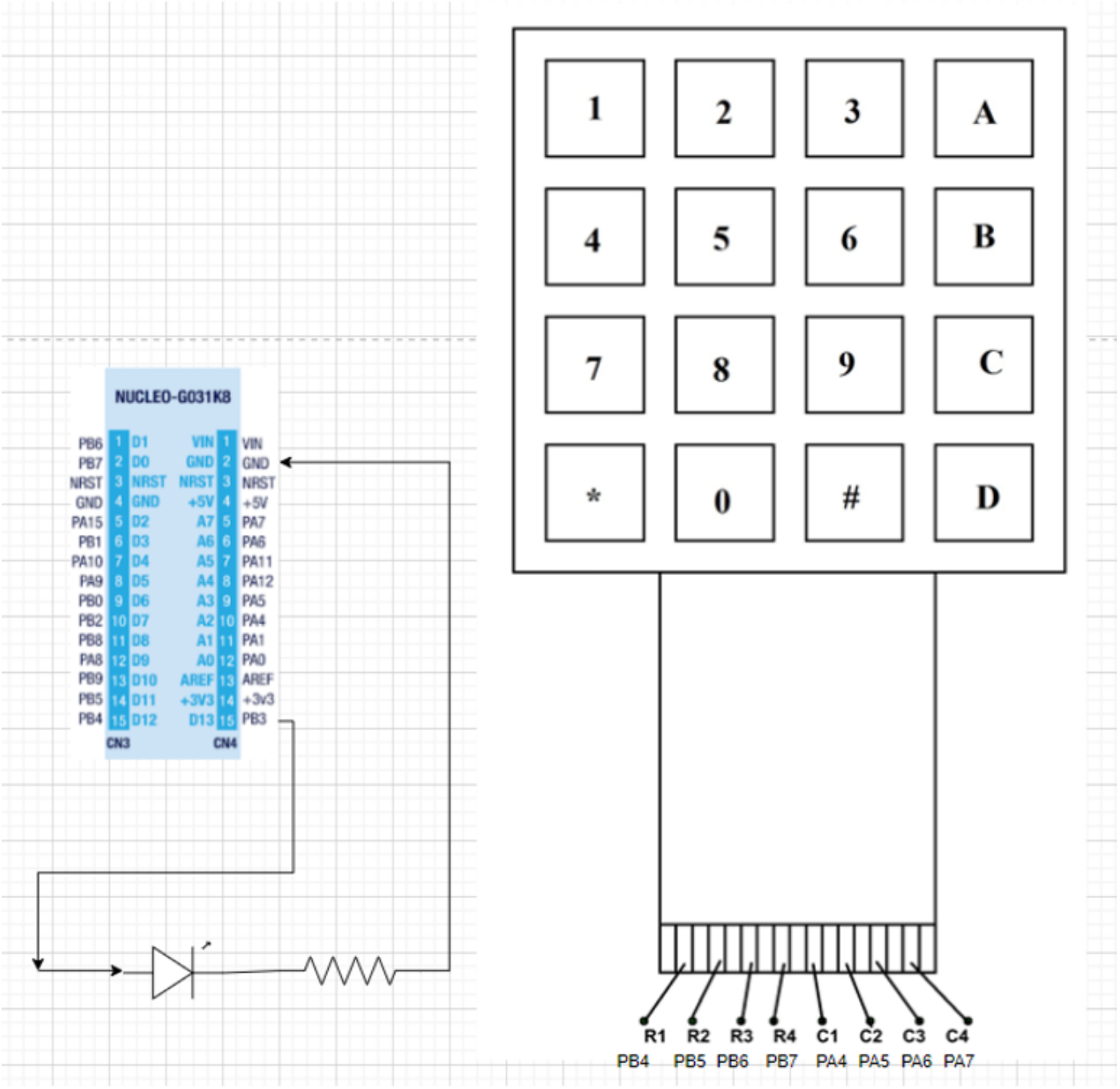

Figure 50: Schematic of REGC_008_withkeypad

20 REGC_009_ADC_POT_PWM_led_control

```
The STM32G0 microcontroller reads analog
```

signal with ADC from the pot connected to pin PA12, converts the ADC values to voltage, and set the duty cycles of the 2 PWM channels according to the ADC value. Each led is connected to PWM channel at pins PA15 and PB3. When the pot is turned to the left the ADC value will be 0 and the left led will bright with a full brightness (full duty cycle) when

turning to the right the brightness of the left led decrease. At the middle the left led turn off, then when turning to the right the right led will turn on from low to high brightness according to the PWM signal duty cycle.

20.1 Flowchart

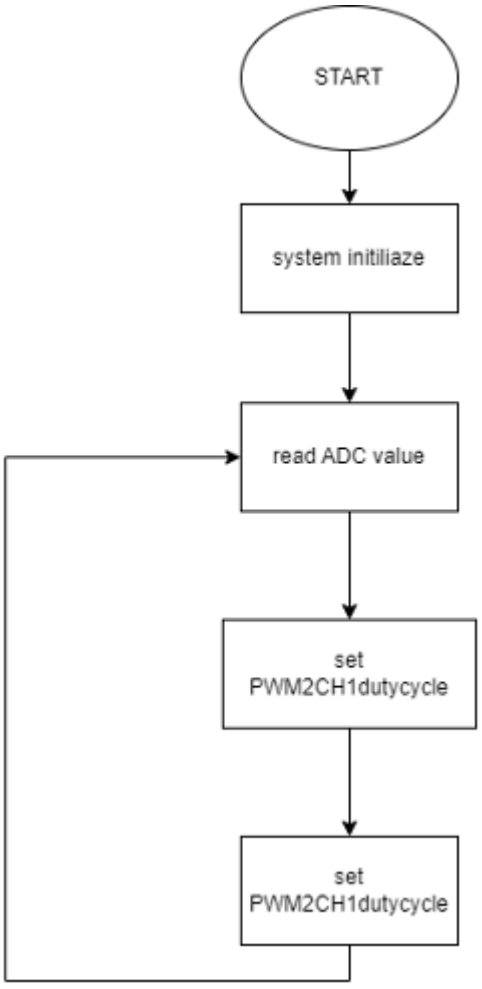

Figure 51 : Flowchart of REGC_009_ADC_POT_PWM_led_control

20.2 Schematic

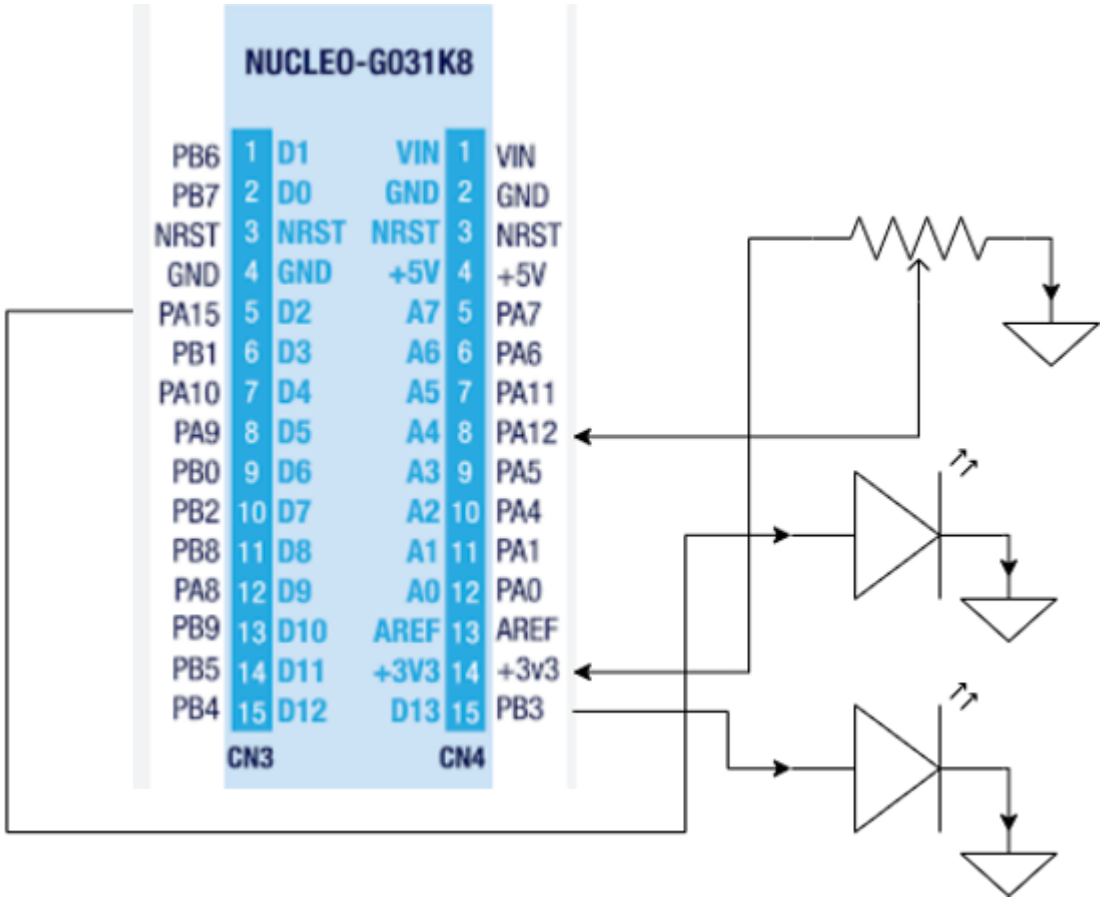

Figure 52 : Schematic of REGC_009_ADC_POT_PWM_led_control

21 REGC_012_knock_detector

In this problem, the sound sensor and the IMU sensor is combined to detect knock and write the number of the knocks to the seven-segment display. The IMU sensor data, acceleration and gyro for each axis is read using I2C protocol. The gyro is measuring the angle velocity from 0 to 250 degree per second. Always the sensor is read, and smooth data is obtained by using exponential smoothing, by a factor of 0.001. When the instant value of the gyro in one of any axes is far then the smooth gyro data 20 degree/second, it's mean that a vibration detected and it's that the IMU say that is a possible knock. Then the sound data is read from sound sensor if a high sound detected that is a possible knock. If at the same time the IMU sensor and the sound sensor detect a possible knock that's mean that's really knock and the knock counter increase. Then the timer flag is set to zero and timer 3 counter set to 0. After 100ms the timer flag will set to 1 inside the interrupt. The knock check will only be done if the flag value is 1. This method avoid blocking delay and the debouncing of knock checking.

To make this project REGC_010_sound_sensor_ADC and REGC_011_mpu6050 is used.

21.1 Flowchart

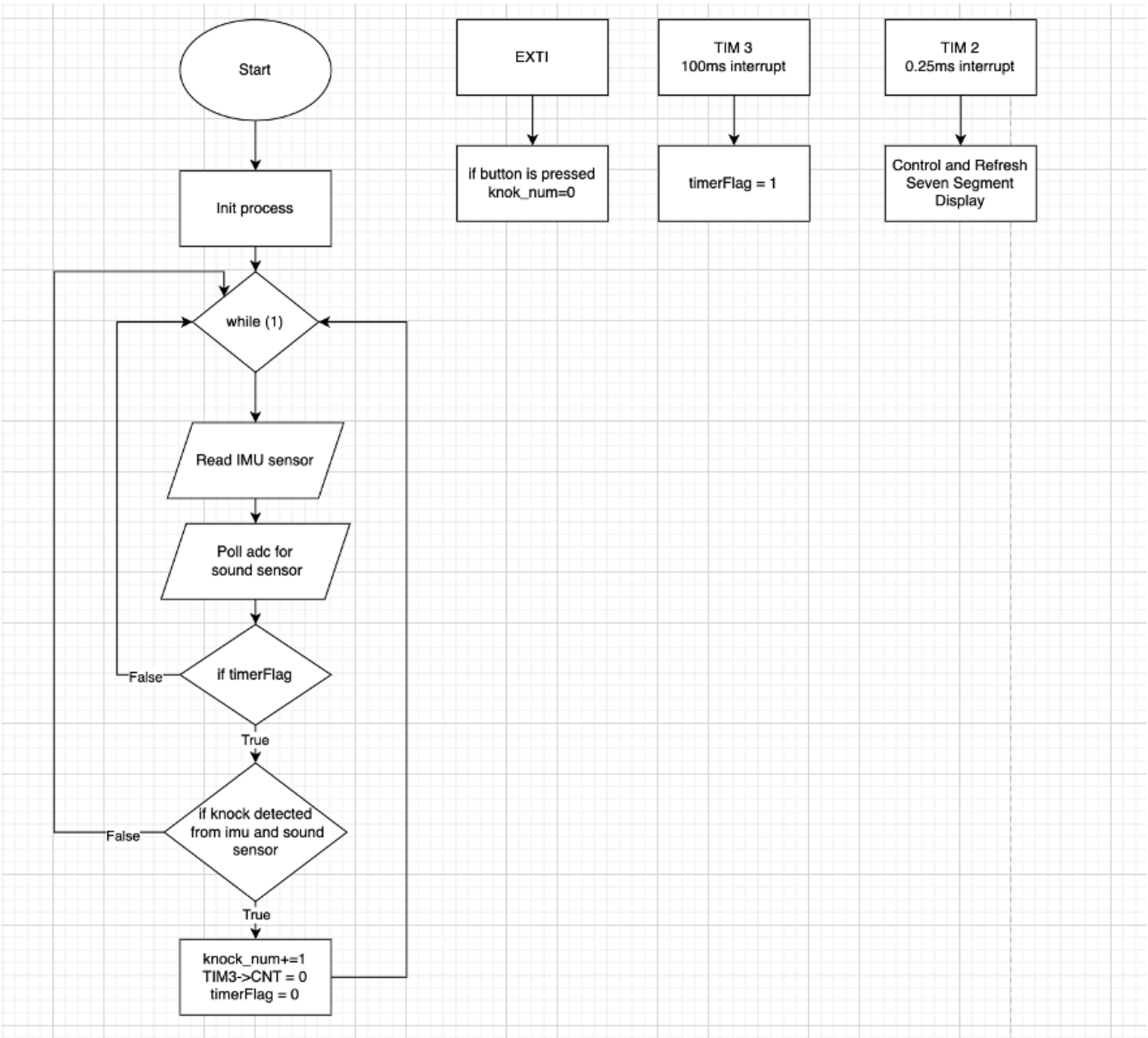

Figure 53 : Flowchart of REGC_012_knock_detector

21.2 Schematic

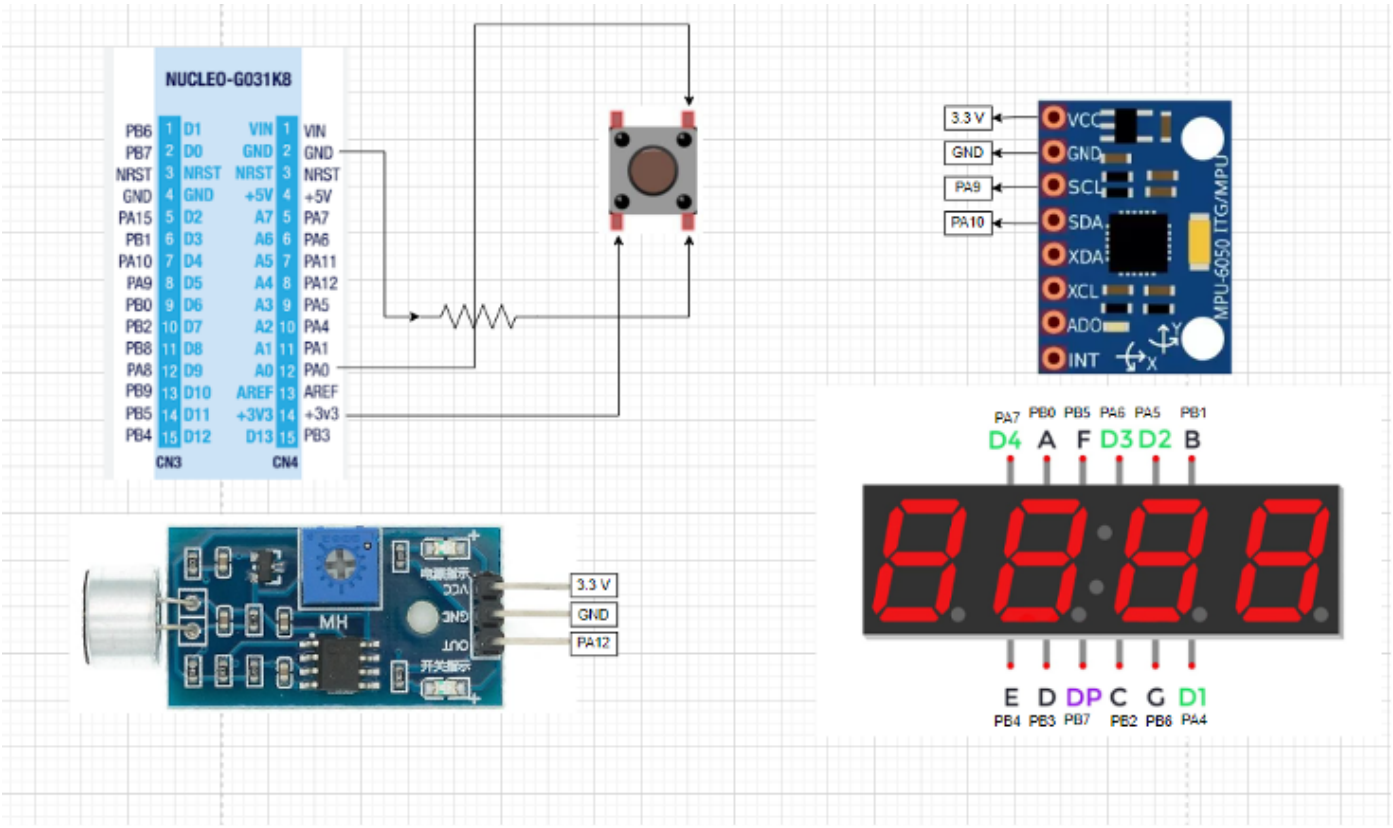

Figure 54 : Schematic of REGC_012_knock_detector