logo, grafik, grafik tasarım, metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**GEBZE TECHNICAL UNIVERSITY**

**ELECTRONIC ENGINEERING**

ELM334

Microprocessors Laboratory

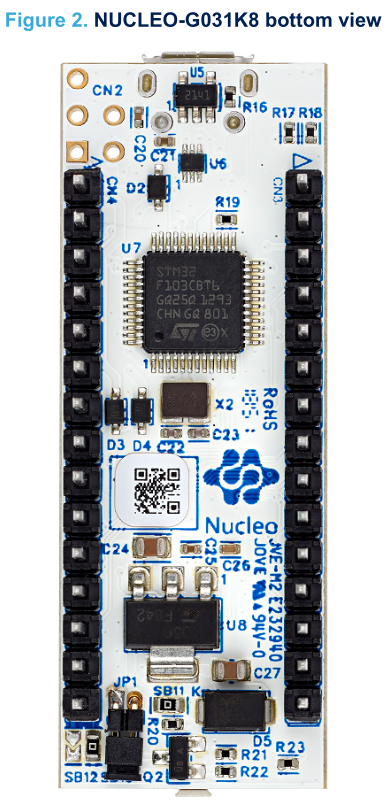
LAB 1 Experiment Report

| 1) 210102002037– Beyzanur Cam |
| --- |
| 2) 210102002026 – Efe Bayrakçeken |
| 3) 200102002027 – Erhan Gök |

**Problem 1.** In the Nucleo G031K8 board, identify all the ICs and explain their usage. Also explain all the connected peripherals and their pin connections with the microcontroller.

metin, devre, elektronik mühendisliği, ekran görüntüsü içeren bir resim

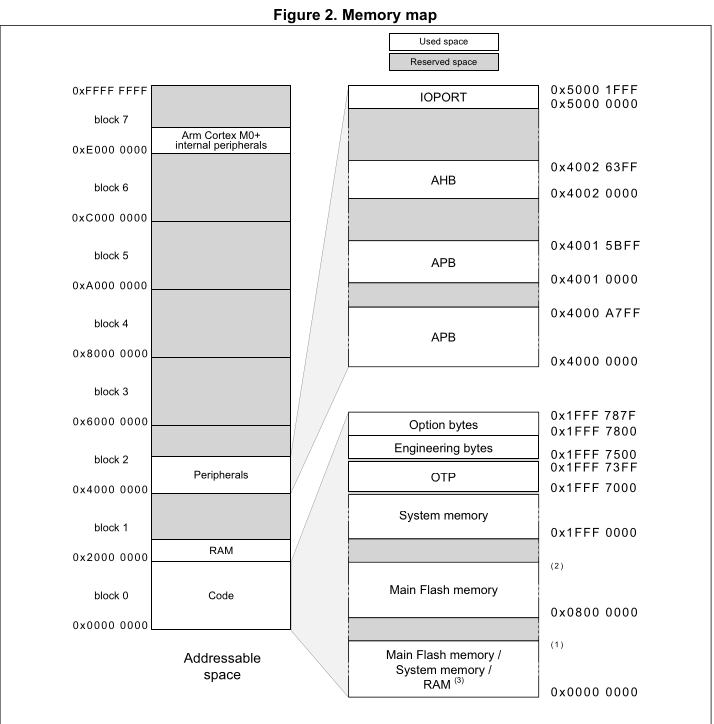
Açıklama otomatik olarak oluşturuldu 



**Problem 2.** Implement assembly code that will light up 1 LED connected to pin PA8. For this problem (and lab) you do not need to worry about clocking the peripherals (and that is fine if you don't even know what that means).

In order to turn a led connected to PA8 on the processor, we have to first enable the RCC clock. Then set the GPIO mode to output. Then we can finally turn on the led.

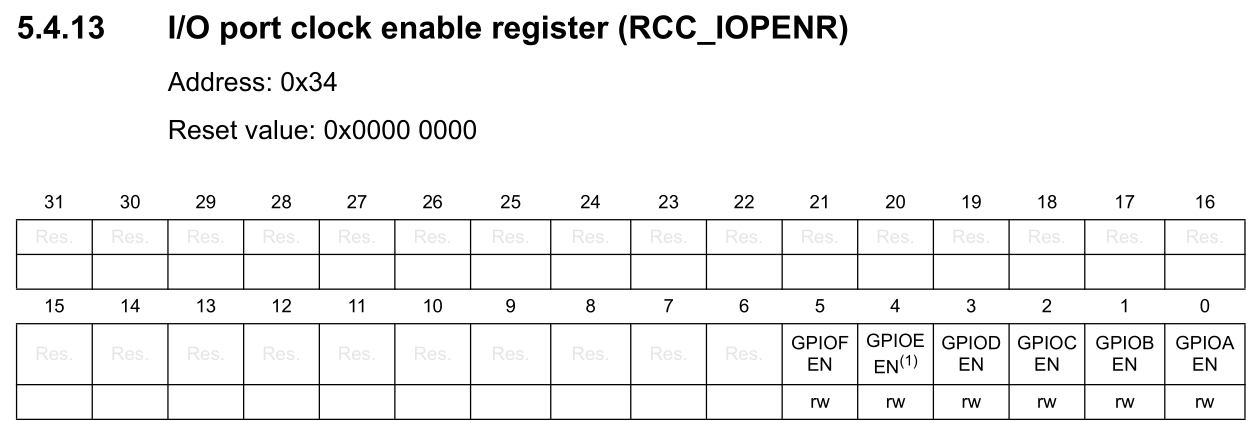
To do all of these, first we need to find the memory addresses related to the RCC clock with relevant values. According to the memory map of the arm architecture, the memory addresses that are related to RCC are located between the AHB(advanced high-performance bus) addresses.



The exact addresses can be seen at table 6 of the reference manual. I/O port reset register has a 0x24 bit offset compared to the RCC base value. The bit 0 needs to be set to enable the GPIOA.

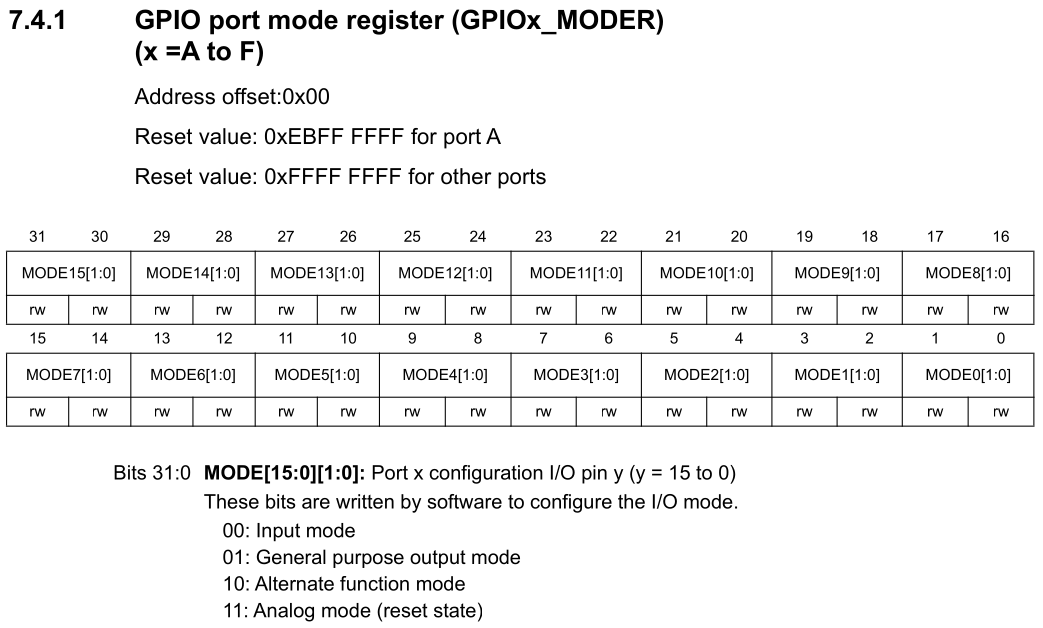
After the RCC configuration is done, GPIO can be set to output mode. The base value for the GPIOA again can be seen at Table 6 of the reference manual.

****



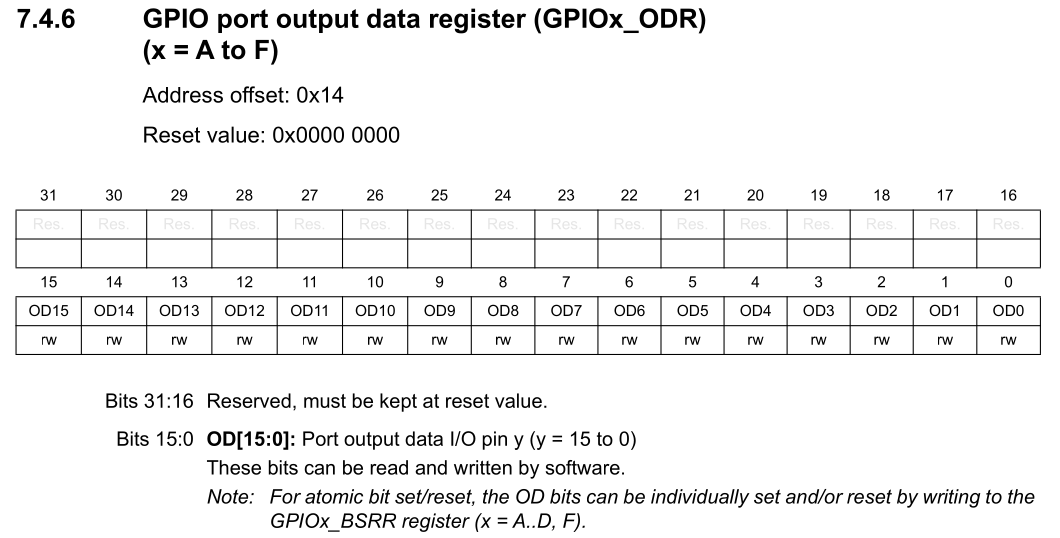
In order to enable the clock of the port A, we need RCC\_ BASE address and the offset of RCC\_IOPENR register. We can obtain the “RCC\_BASE” address from Table 6 of STM32G0 reference manual which is “0x40021000“ and also we can obtain “RCC\_IOPENR” offset from part 5.4.13 which is “0x34” . In STM32CubeIDE, this addresses set as variables like this:

| .equ RCC\_BASE, (0x40021000) // RCC base address  .equ RCC\_IOPENR, (RCC\_BASE + (0x34)) // RCC IOPENR register offset |
| --- |

****

Then the base address of the GPIOA can be set as variables. The addresses related to the GPIOA\_BASE can be found at Table 6 of STM32G0. Then the address related to the mode of PA8 GPIO can be set as per the section 7.4.1 of the reference manual. Lastly GPIO port output data register (GPIOA\_ODR) can be declared.

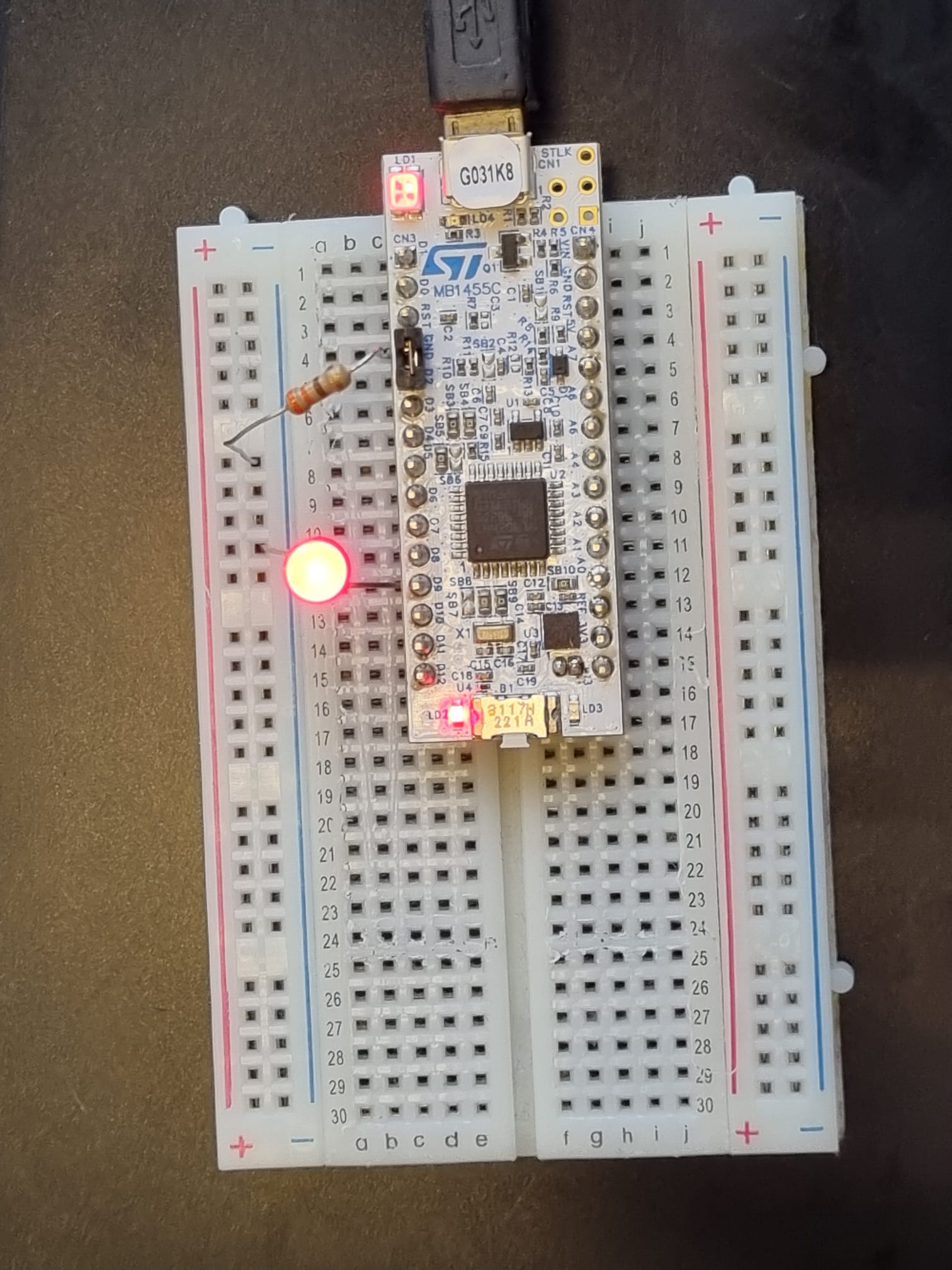
| .equ GPIOA\_BASE, (0x50000000) //GPIOA Base Adress  .equ GPIOA\_MODER, (GPIOA\_BASE + (0x00)) // GPIOA MODER register offset  .equ GPIOA\_ODR, (GPIOA\_BASE + (0x14)) // GPIOA ODR register offset |
| --- |

****

Then the main loop can be finally written. All the explanations for the code have been provided as comment lines with the code.

| **main:**  //////Enable RCC\_IOPENR For A//////  ldr r0,=RCC\_IOPENR //Load RCC\_IOPENR adress to r0  ldr r1, [r0] //Load content of the adress in r0  movs r2, #0x01 //0000 0001, For enabling RCC mask  orrs r1, r1, r2 //Set first bit  str r1, [r0] //Enable RCC mask for Port A    ///////////GPIOx\_MODER/////////  ldr r0,=GPIOA\_MODER //Load GPIOA\_MODER adress to r0  ldr r1,[r0] //Load content of the adress in r0  ldr r2,=0x30000 //For setting MODE8 Pin  mvns r2,r2 //r2=~r2  ands r1, r1, r2 //Clear bit r2 and save to r1  ldr r2, =0x10000 //For output mode set to 01 for MODE8  orrs r1,r1,r2  str r1, [r0] //Store r1 to GPIOA\_MODER adress    ///////////GPIOA\_ODR//////////  ldr r0,=GPIOA\_ODR  ldr r1, [r0] //Get GPIOA\_ODR content  ldr r2,=0x100 //GPIOA\_ODR mask for setting Pin8  orrs r1, r1, r2  str r1, [r0] //Enable led  /\* for(;;); \*/  b . |
| --- |

Then if we run the code, the led lights up!



The led is connected to PA8 which corresponds to D9 on the board with a 390Ω resistor and to the GND of the board. And after running the Assembly code on STM32G0 board, the led was turned on.

**Problem 3.** Implement assembly code that will light up 4 LEDs connected to pins PA11, PA12, PB4, PB5.

For this question the only difference from problem 2 will be opening port B and connecting extra leds for each port(A and B). So the coding will be very similar.

Setting the base addresses of GPIOA and GPIOB can be declared with the same logic that was used in problem 2.

| .equ GPIOA\_BASE, (0x50000000) //GPIOA Base Adress  .equ GPIOB\_BASE, (0x50000400) //GPIOB Base Adress  .equ GPIOA\_MODER, (GPIOA\_BASE + (0x00)) // GPIOA MODER register offset  .equ GPIOB\_MODER, (GPIOB\_BASE + (0x00)) // GPIOB MODER register offset  .equ GPIOA\_ODR, (GPIOA\_BASE + (0x14)) // GPIOA ODR register offset  .equ GPIOB\_ODR, (GPIOB\_BASE + (0x14)) // GPIOB ODR register offset |
| --- |

The relevant IO Clocks can be enabled with the mask 0x3 which sets the first bits to 1.

| //////Enable RCC\_IOPENR For A AND B//////  ldr r0,=RCC\_IOPENR  ldr r1, [r0]  movs r2, #0x03 //0000 0011, For enabling RCC mask for A and B port  orrs r1, r1, r2  str r1, [r0] //Enable RCC mask |
| --- |

| ///////////GPIOA\_MODER/////////  ldr r0,=GPIOA\_MODER  ldr r1,[r0]  ldr r2,=0x03C00000 //0011 1100 0000 0000 0000 0000 0000  bics r1, r2 //1100 0011 1111 1111 1111 1111 1111  //xx00 00xx xxxx xxxx xxxx xxxx xxxx  ldr r2, =0x1400000  orrs r1,r1,r2 //xx01 01xx xxxx xxxx xxxx xxxx xxxx  str r1, [r0]  ///////////GPIOB\_MODER/////////  ldr r0,=GPIOB\_MODER  ldr r1, [r0]  ldr r2, =0xF00 //1111 0000 0000  bics r1, r2 //0000 1111 1111  //0000 xxxx xxxx  ldr r2, =0x500 //0101 0000 0000  orrs r1, r1, r2 //0101 xxxx xxxx  str r1, [r0] |
| --- |

The only difference with problem 2 is that the bics instruction was used. This instruction can do the clearing of the bits in a single instruction instead of 2 instructions that are needed with doing a mvn and ‘ands’ operation.

| ///////////GPIOA\_ODR//////////  ldr r0,=GPIOA\_ODR  ldr r1, [r0]  ldr r2,=0x1800 //0001 1000 0000 0000  orrs r1, r1, r2  str r1, [r0]  ///////////GPIOB\_ODR//////////  ldr r0,=GPIOB\_ODR  ldr r1, [r0]  ldr r2,=0x30 //0011 0000  orrs r1, r1, r2  str r1, [r0] |
| --- |

As expected, when the code above is run, all the leds light up!

****

All the leds are connected to the relevant pins with a 390Ω resistor connected to the ground.

**Problem 4.** Implement a delay routine and have 1 LED described on Problem 2 toggle in roughly 1 second intervals. For this assume the clock is running at 16 Mhz. (It will be impossible to single step through 1 second delay routine, so utilize breakpoints)

We assumed that the processor runs at 16mhz, so a FREQ value was declared. Then the count of cycles needed for a loop was declared. This will be explained later in the report.

| .equ FREQ, (16800000)  .equ CYCLE\_COUNT, (FREQ/4) |
| --- |

The code provided is the delay function. The function subtracts 1 to a known value. When the value of r7 becomes 0, the zero flag of the alu is raised and the branch instruction returns to the value that is stored in the link register. The subs instruction just takes 1 clock cycle but the bx instruction requires 3 clock cycles to run. Therefore the clock frequency was divided by 4 to get the appropriate delay required for 1 second loops.

| **delay\_func:**  subs r7, #1  bne delay\_func  ldr r7, =CYCLE\_COUNT  bx lr |
| --- |

This code part is mostly the same with problem 1. The only difference is we set r7 to CYCLE\_COUNT we calculated before. In order for the delay function to work r7 must be set to CYCLE\_COUNT, otherwise the register starts from -1 and in the delay function it starts to subtracts from -1. So the necessary condition(Z=1) for the delay function to return to the loop is never met.

| ///////////GPIOA\_ODR//////////  ldr r0,=GPIOA\_ODR  ldr r1, [r0]  ldr r2,=0x100  ldr r7, =CYCLE\_COUNT //Set R7 to CYCLE\_COUNT |
| --- |

For the final part we have to create a loop to call function when needed. When led lights up “bl” (branch & link) saves the program counter to the link register and makes code to go to the delay\_func. After “delay\_func" is executed, bx recovers the program counter from the link register and brings code back to the loop. Then the bits are cleared and the led turns off. Then the same execution goes on and on. And because of the aim of delay\_func our eyes can catch the led’s status.

| **loop:**  orrs r1, r1, r2  str r1, [r0] //Turn on LED  bl delay\_func  bics r1, r1, r2  str r1, [r0] //Turn off LED  bl delay\_func  b loop |
| --- |

# 

# References:

<https://developer.arm.com/documentation/ddi0337/e/BABBCJII>

<https://github.com/fcayci/stm32g0>

[STM32G0x1 advanced Arm®-based 32-bit MCUs - Reference manual - AKA The bible](https://www.st.com/resource/en/reference_manual/rm0444-stm32g0x1-advanced-armbased-32bit-mcus-stmicroelectronics.pdf)

[STM32G0 Nucleo-32 board - User manual](https://www.st.com/resource/en/user_manual/um2591-stm32g0-nucleo32-board-mb1455-stmicroelectronics.pdf)