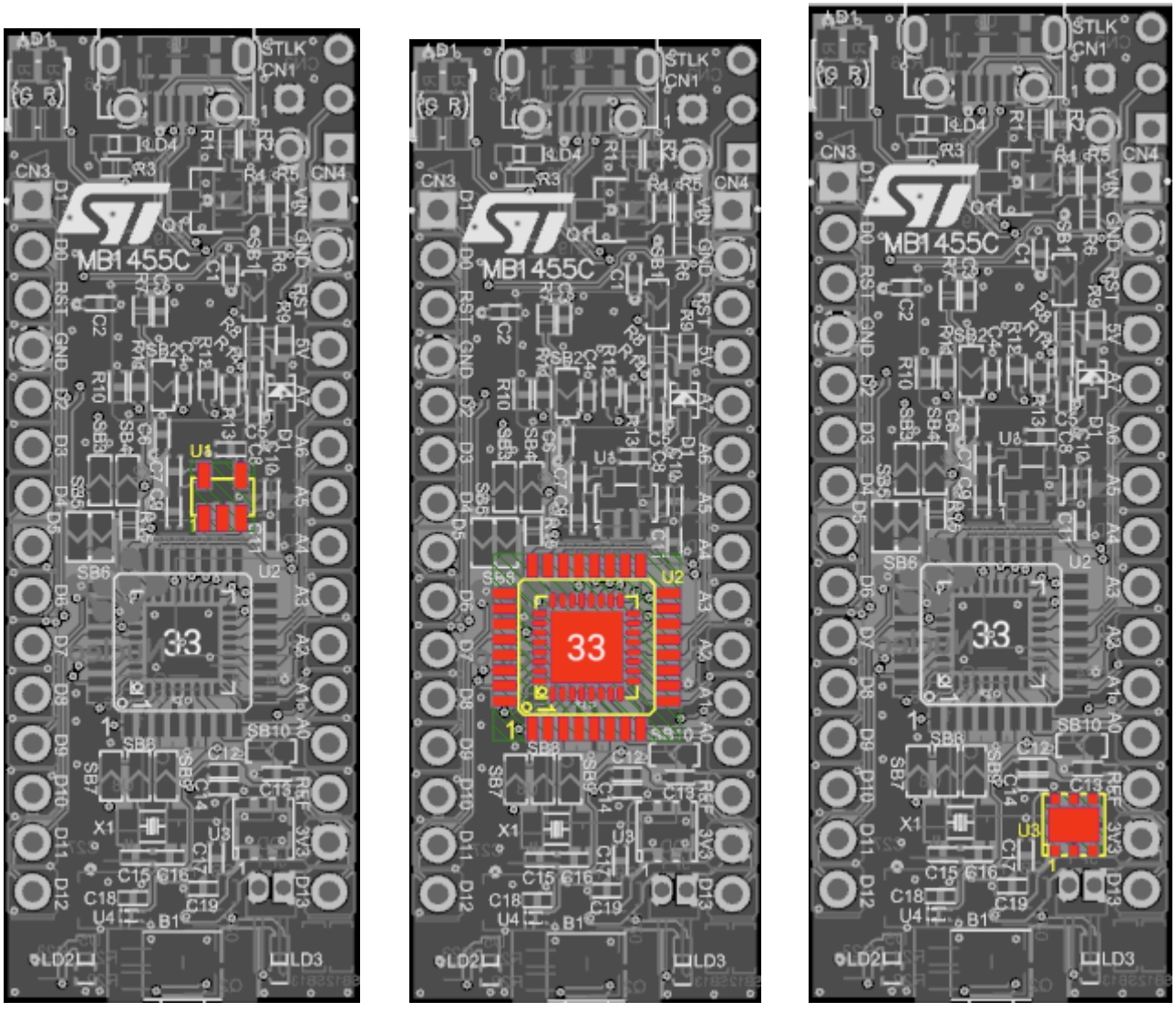
ELM 334 - LAB #1

Ömer Emre Polat
1801022037

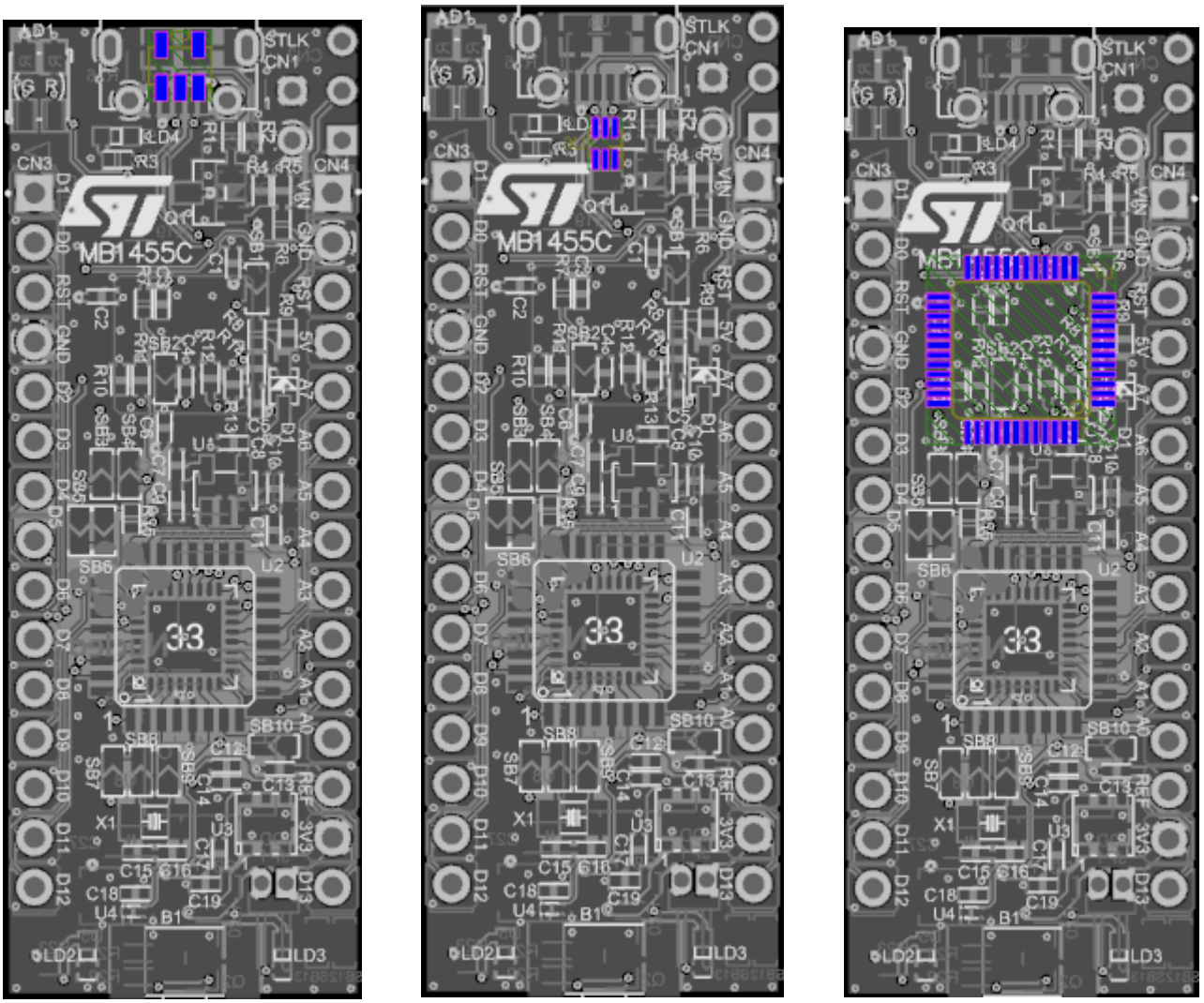
A. Problem 1

IC’s are identified using an official circuit model of the STM32G031K8T6U board and a model preview software.



From left to right

1- LDK120M33R
   200mA low quiescent current very low noise LDO
2- STM32G031K8T6U
   Cortex M0+ microcontroller
3- LD39050PU33R
   Low quiescent current and low noise voltage regulator

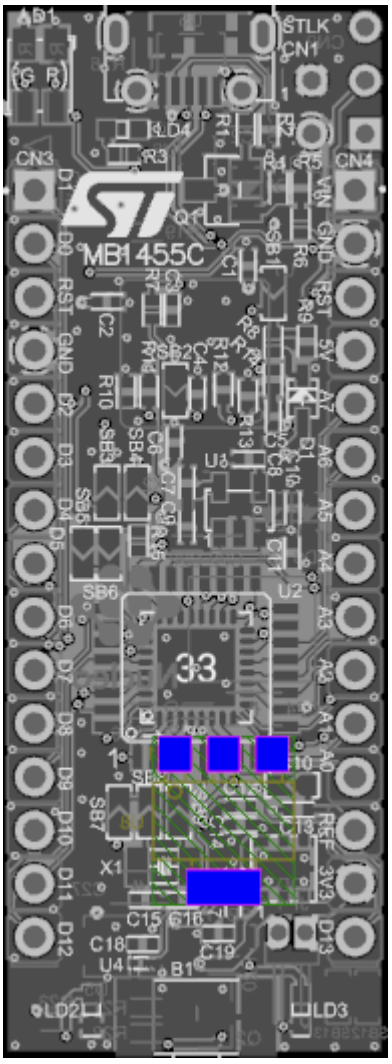


From left to right

1- STMPS2141STR
   Enhanced single channel switch

2- USBLC6-2P6
   USB OTG FS ESD protection

3- STM32F103CBT6
   Arm based 32-bit MCU



From left to right

1- LD1117S50TR
5V REG LDO SOT223

B. Problem 2

For this problem we will have to look up for the value of GPIOA base address from the RM044.

The following table gives the boundary addresses of the peripherals.

Table 4. 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 6.4.12 on page 211*
	0x5000 1000 - 0x5000 13FF	1 KB	Reserved	-
	0x5000 0C00 - 0x5000 0FFF	1 KB	GPIOD	*Section 6.4.12 on page 211*
	0x5000 0800 - 0x5000 0BFF	1 KB	GPIOC	*Section 6.4.12 on page 211*
	0x5000 0400 - 0x5000 07FF	1 KB	GPIOB	*Section 6.4.12 on page 211*
	0x5000 0000 - 0x5000 03FF	1 KB	GPIOA	*Section 6.4.12 on page 211*

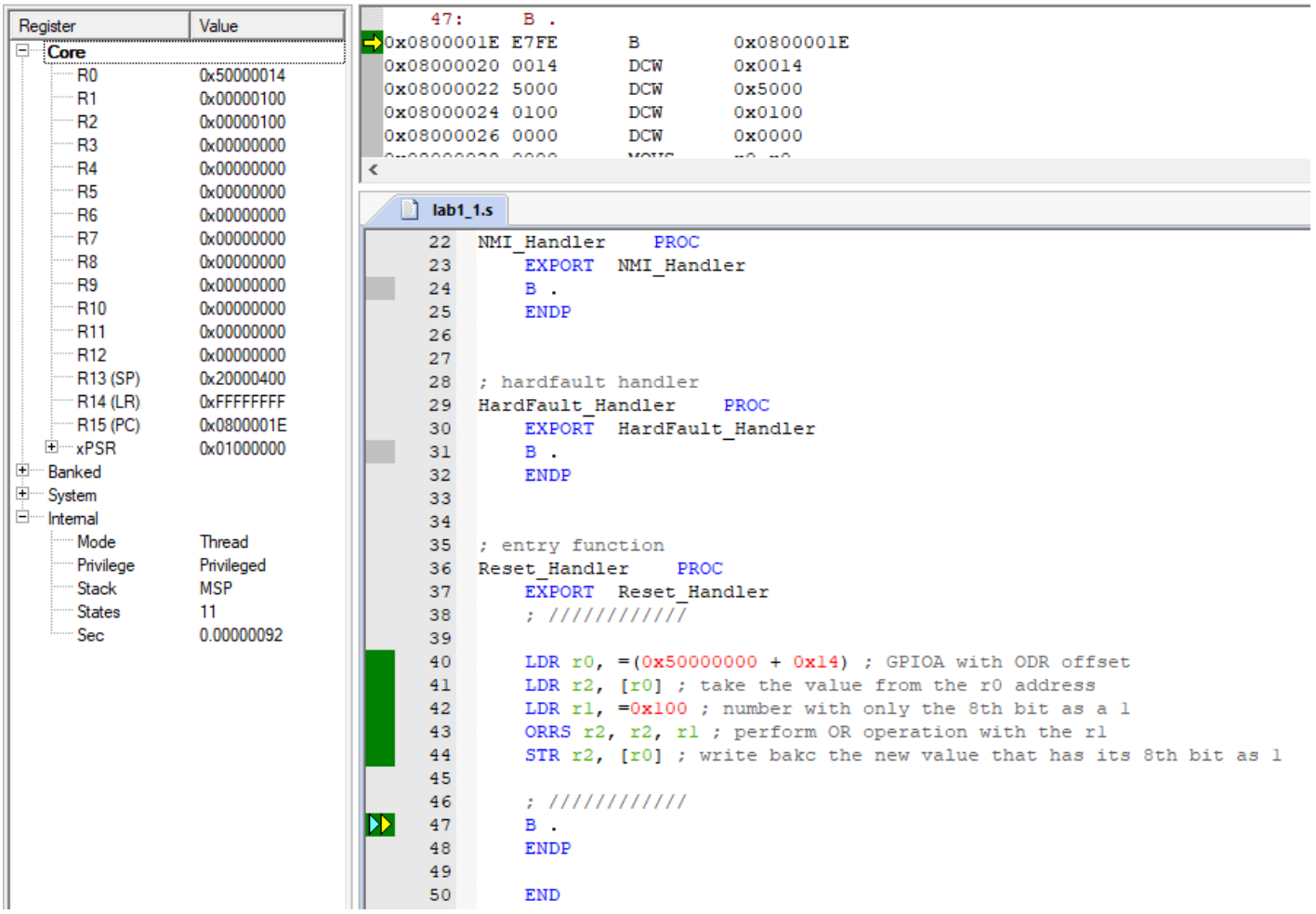
Now we need to find the ODR offset value from the RM044.

0x14	**GPIOx_ODR** (where x = **A..D, F**)	R(15)	R(14)	R(13)	R(12)	R(11)	R(10)	R(9)	R(8)	R(7)	R(6)	R(5)	R(4)	R(3)	R(2)	R(1)	R(0)	OD15	OD14	OD13	OD12	OD11	OD10	OD9	OD8	OD7	OD6	OD5	OD4	OD3	OD2	OD1	OD0
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

After finding the correct address of the value for peripherals we can write the code for the LED code.

```
Stack_Size      EQU     0x00000400

                AREA    STACK, NOINIT, READWRITE, ALIGN=3
Stack_Mem       SPACE   Stack_Size
__initial_sp

    THUMB

    AREA    RESET, DATA, READONLY
    EXPORT  __Vectors

__Vectors
    DCD     __initial_sp                ; Top of Stack
    DCD     Reset_Handler               ; Reset Handler
    DCD     NMI_Handler                 ; NMI Handler
    DCD     HardFault_Handler           ; Hard Fault Handler

    AREA    |.text|, CODE, READONLY


; nmi handler
NMI_Handler    PROC
    EXPORT  NMI_Handler
    B .
    ENDP
```

```
; hardfault handler
HardFault_Handler    PROC
    EXPORT  HardFault_Handler
    B .
    ENDP


; entry function
Reset_Handler    PROC
    EXPORT  Reset_Handler
    ; ////////////

      LDR r0, =(0x50000000 + 0x14) ; GPIOA with ODR offset
    LDR r2, [r0] ; take the value from the r0 address
    LDR r1, =0x100 ; number with only the 8th bit as a 1
    ORRS r2, r2, r1 ; perform OR operation with the r1
    STR r2, [r0] ; write bakc the new value that has its 8th bit as 1

    ; ////////////
    B .
    ENDP

    END
```



As we can see, the register r2 contains the value of the GPIOA with ODR offset address. 0x0100 has its 8th bit as a 1.

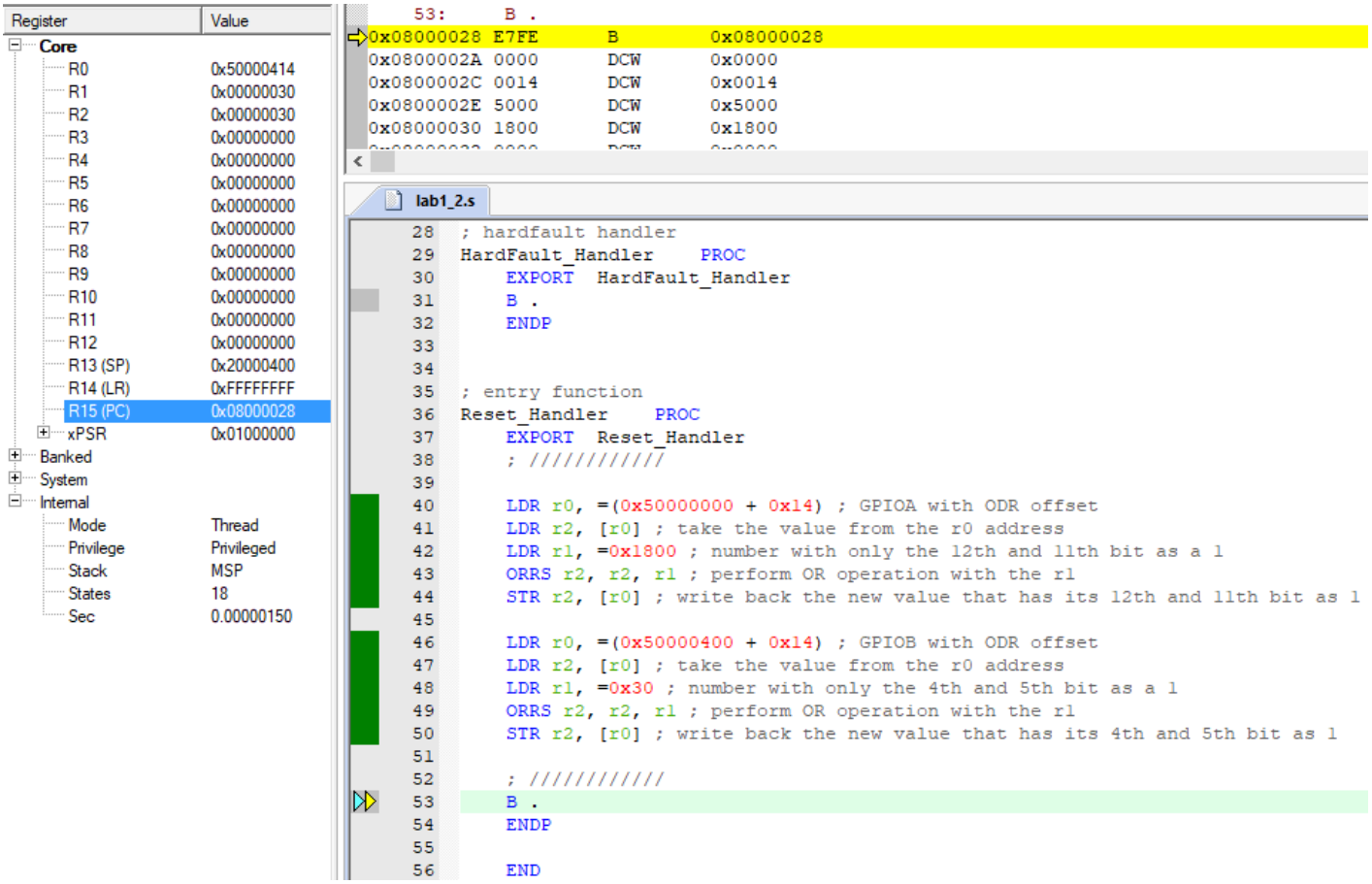
C. Problem 3

We can use the same code above with a little modification to write addresses and ORR values to light up multiple bits.

```
Stack_Size      EQU     0x00000400

                AREA    STACK, NOINIT, READWRITE, ALIGN=3
Stack_Mem       SPACE   Stack_Size
__initial_sp

    THUMB

    AREA    RESET, DATA, READONLY
    EXPORT  __Vectors

__Vectors
    DCD     __initial_sp                ; Top of Stack
    DCD     Reset_Handler               ; Reset Handler
    DCD     NMI_Handler                 ; NMI Handler
    DCD     HardFault_Handler           ; Hard Fault Handler

    AREA    |.text|, CODE, READONLY


; nmi handler
NMI_Handler    PROC
    EXPORT  NMI_Handler
    B .
    ENDP


; hardfault handler
HardFault_Handler    PROC
    EXPORT  HardFault_Handler
    B .
    ENDP


; entry function
Reset_Handler    PROC
    EXPORT  Reset_Handler
    ; ////////////

    LDR r0, =(0x50000000 + 0x14) ; GPIOA with ODR offset
    LDR r2, [r0] ; take the value from the r0 address
    LDR r1, =0x1800 ; number with only the 12th and 11th bit as a 1
    ORRS r2, r2, r1 ; perform OR operation with the r1
    STR r2, [r0] ; write back the new value that has its 12th and 11th bit as 1

      LDR r0, =(0x50000400 + 0x14) ; GPIOB with ODR offset
    LDR r2, [r0] ; take the value from the r0 address
    LDR r1, =0x30 ; number with only the 4th and 5th bit as a 1
```

```
    ORRS r2, r2, r1 ; perform OR operation with the r1
    STR r2, [r0] ; write back the new value that has its 4th and 5th bit as 1

    ; ////////////
    B .
    ENDP

    END
```

As stated above we use two different GPIO addresses and ORR values to light up multiple bits at different address values. We can determine the ORR values by the high bit positions. For example in the GPIOA part we use a number that's 12^{th} and 11^{th} bits are 1 and all the other bits are zero. Perfoming ORR operation on the address value with this number ensures that the other on or off values wont be lost after rewriting the address to light up the led.



As expected after modifying the W/O addresses on the linker settings, we don't get any write or read violation errors and the program works as expected.

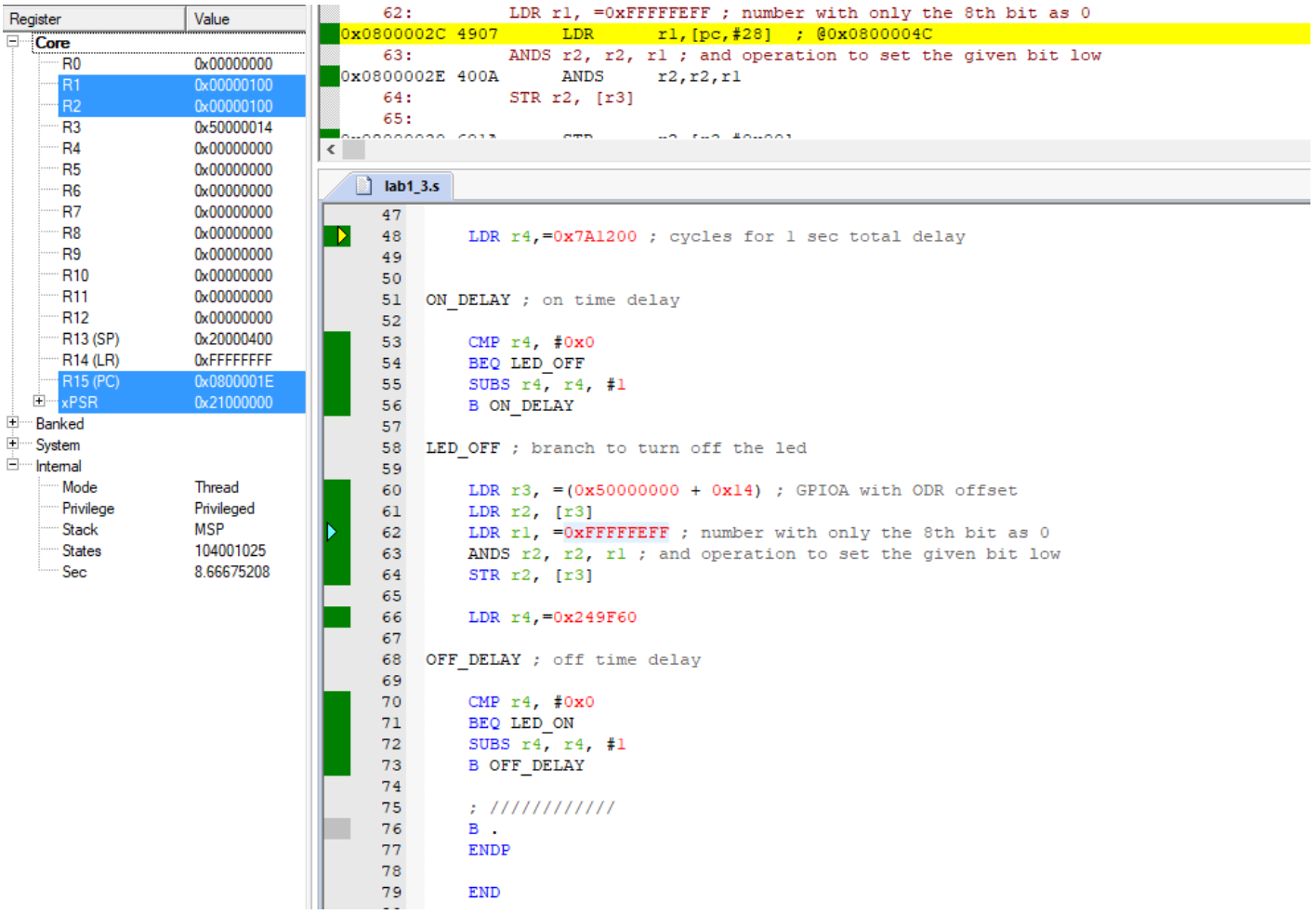
D. Problem 4

With this problem we have to implement a led delay loop that will turn the led on wait for around 1 seconds turn the led off wait for additional 1 seconds and jump back to the beginning.

```
Stack_Size      EQU     0x00000400

                AREA    STACK, NOINIT, READWRITE, ALIGN=3
Stack_Mem       SPACE   Stack_Size
__initial_sp

    THUMB

    AREA    RESET, DATA, READONLY
    EXPORT  __Vectors

__Vectors
    DCD     __initial_sp                ; Top of Stack
    DCD     Reset_Handler               ; Reset Handler
    DCD     NMI_Handler                 ; NMI Handler
    DCD     HardFault_Handler           ; Hard Fault Handler

    AREA    |.text|, CODE, READONLY


; nmi handler
NMI_Handler    PROC
    EXPORT  NMI_Handler
    B .
    ENDP


; hardfault handler
HardFault_Handler    PROC
    EXPORT  HardFault_Handler
    B .
    ENDP


; entry function
Reset_Handler    PROC
    EXPORT  Reset_Handler
    ; ////////////

LED_ON ; branch to turn the led on

      LDR r3, =(0x50000000 + 0x14) ; GPIOA with ODR offset
      LDR r2, [r3]
      LDR r1, =0x100 ; number with only the 8th bit as 1
      ORRS r2, r2, r1 ; or operation to set the given bit high
      STR r2, [r3]

      LDR r4,=0x7A1200 ; cycles for 1 sec total delay
```

```
ON_DELAY ; on time delay

      CMP r4, #0x0
      BEQ LED_OFF
      SUBS r4, r4, #1
      B ON_DELAY

LED_OFF ; branch to turn off the led

      LDR r3, =(0x50000000 + 0x14) ; GPIOA with ODR offset
      LDR r2, [r3]
      LDR r1, =0xFFFFFEFF ; number with only the 8th bit as 0
      ANDS r2, r2, r1 ; and operation to set the given bit low
      STR r2, [r3]

      LDR r4,=0x249F60

OFF_DELAY ; off time delay

      CMP r4, #0x0
      BEQ LED_ON
      SUBS r4, r4, #1
      B OFF_DELAY

    ; ////////////
    B .
    ENDP

    END
```

With the code written we can test it if it loops continuously.



As we can see after doing multiple loops the code still works with the proper branch jumps.

References

---

[1]. RM0444 https://www.st.com/resource/en/reference_manual/dm00371828-stm32g0x1-advanced-armbased-32bit-mcus-stmicroelectronics.pdf