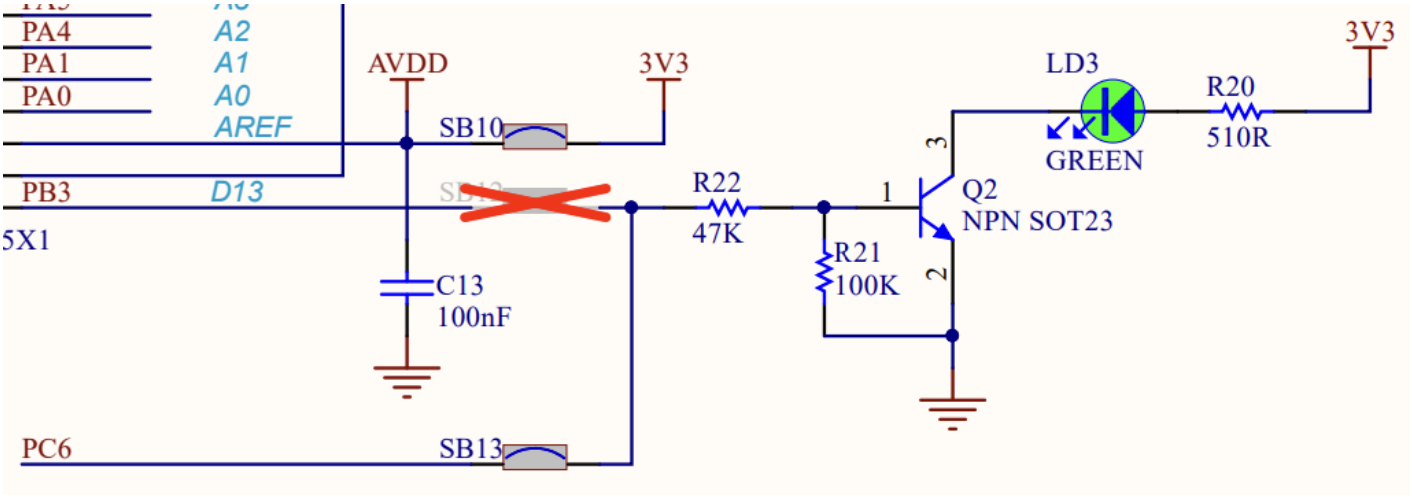
ELM 334 - LAB #2

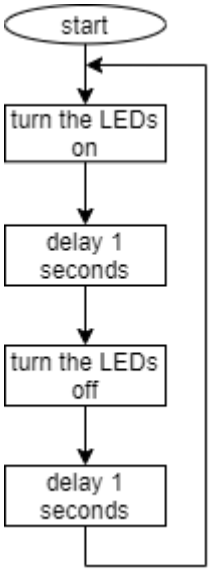
Ömer Emre Polat
1801022037

A. Problem 1

In this problem we have to turn on and off the onboard green LED. This can be done after looking up to the schematics of the stm32g031 board.



As we can see the Pin C6 controls the led turn on with the help of the NPN BJT transistor.



Here's a basic flowchart of the code working on normal conditions.

Now the base addresses can be searched in RM0444.

6.4 GPIO registers

This section gives a detailed description of the GPIO registers.

For a summary of register bits, register address offsets and reset values, refer to *Table 33*.

The peripheral registers can be written in word, half word or byte mode.

6.4.1 GPIO port mode register (GPIOx_MODER) (x =A to D, 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)

5.4.12 I/O port clock enable register (RCC_IOPENR)

Address: 0x34

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access

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	Res.	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN
										rw		rw	rw	rw	rw

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

6.4.5 GPIO port input data register (GPIOx_IDR) (x = A to D, 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.

6.4.6 GPIO port output data register (GPIOx_ODR) (x = A to D, 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).

After looking for all the addresses we will use, now all that's left is to write a code that uses this information.

```
.syntax unified
.cpu cortex-m0
.fpu softvfp
.thumb


/* make linker see this */
.global Reset_Handler

/* get these from linker script */
.word _sdata
.word _edata
.word _sbss
.word _ebss


/* define peripheral addresses from RM0444 page 57, Tables 3-4 */
.equ RCC_IOPENR,    (0x40021034)
```

```
.equ GPIOC_MODER,   (0x50000800)
.equ GPIOC_ODR,     (0x50000814)


/* vector table, +1 thumb mode */
.section .vectors
vector_table:
    .word _estack             /*     Stack pointer */
    .word Reset_Handler +1    /*     Reset handler */
    .word Default_Handler +1  /*       NMI handler */
    .word Default_Handler +1  /* HardFault handler */
    /* add rest of them here if needed */


/* reset handler */
.section .text
Reset_Handler:
    /* set stack pointer */
    ldr r0, =_estack
    mov sp, r0

    /* initialize data and bss
     * not necessary for rom only code
     * */
    bl init_data
    /* call main */
    bl main
    /* trap if returned */
    b .


/* initialize data and bss sections */
.section .text
init_data:

    /* copy rom to ram */
    ldr r0, =_sdata
    ldr r1, =_edata
    ldr r2, =_sidata
    movs r3, #0
    b LoopCopyDataInit

    CopyDataInit:
        ldr r4, [r2, r3]
        str r4, [r0, r3]
        adds r3, r3, #4

    LoopCopyDataInit:
        adds r4, r0, r3
        cmp r4, r1
        bcc CopyDataInit

    /* zero bss */
    ldr r2, =_sbss
    ldr r4, =_ebss
```

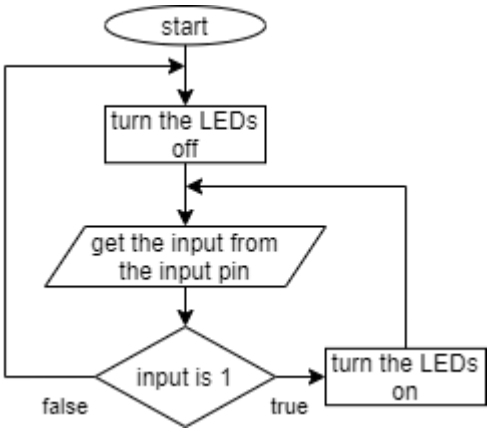
```
    movs r3, #0
    b LoopFillZerobss

    FillZerobss:
        str  r3, [r2]
        adds r2, r2, #4

    LoopFillZerobss:
        cmp r2, r4
        bcc FillZerobss

    bx lr


/* default handler */
.section .text
Default_Handler:
    b Default_Handler


/* main function */
.section .text
main:
    /*//////ENABLE GPIOC CLOCK//////*/
    ldr r1, =(RCC_IOPENR) /* RCC with IOPENR offset */
    ldr r0, [r1]

    ldr r2, =0x4
    orrs r0, r0, r2 /* change the corresponding bit to 1 */
    str r0, [r1] /* enable GPIOC clock */

    /*//////SETUP C6 MODER//////*/
    ldr r1, =(GPIOC_MODER)
    ldr r0, [r1]

    ldr r2, =0x2000
    mvns r2, r2
    ands r0, r0, r2
    ldr r2, =0x1000
    orrs r0, r0, r2 /* ANDS and orrs to turn the C6 to output mode */
    str r0, [r1] /* writing back the modified MODER bits */


    LED_ON: /* branch to turn the led on */

    ldr r1, =GPIOC_ODR
    ldr r0, [r1]
    ldr r2, =0x40
    orrs r0, r0, r2
    str r0, [r1]

    ldr r4,=0xC35000 /* 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 r1, =(GPIOC_ODR) /* GPIOC with ODR offset */
ldr r0, [r1]
ldr r2, =0x40
mvns r2, r2
ands r0, r0, r2 /* and operation to set the given bit low */
str r0, [r1]

ldr r4,=0xC35000

OFF_DELAY: /* off time delay */

cmp r4, #0x0
beq LED_ON
subs r4, r4, #1
b OFF_DELAY

/* for(;;); */
b .

/* this should never get executed */
nop
```

The code works by first enabling the GPIOC clock. Then it modifies the GPIOC_MODER address to set the given pin as an output pin. After the pin setup is done it can continue to the LED on and off loop.

B. Problem 2

For this problem lets start by creating a basic flowchart for the code.



Here we can see the basic operations of the code.

After the flowchart we will have to look up for the value of GPIOA base address from the RM0444.

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 RM0444.

0x14	**GPIOx_ODR** (where x = **A..D, F**)																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.

```
.syntax unified
.cpu cortex-m0
.fpu softvfp
.thumb


/* make linker see this */
.global Reset_Handler

/* get these from linker script */
.word _sdata
.word _edata
.word _sbss
.word _ebss


/* define peripheral addresses from RM0444 */
.equ RCC_IOPENR,    (0x40021034)

.equ GPIOA_MODER,   (0x50000000)
.equ GPIOC_MODER,   (0x50000800)

.equ GPIOA_IDR,     (0x50000010)
.equ GPIOA_ODR,     (0x50000014)

.equ GPIOC_IDR,     (0x50000810)
.equ GPIOC_ODR,     (0x50000814)
```

```
/* vector table, +1 thumb mode */
.section .vectors
vector_table:
    .word _estack             /*     Stack pointer */
    .word Reset_Handler +1    /*     Reset handler */
    .word Default_Handler +1  /*       NMI handler */
    .word Default_Handler +1  /* HardFault handler */
    /* add rest of them here if needed */


/* reset handler */
.section .text
Reset_Handler:
    /* set stack pointer */
    ldr r0, =_estack
    mov sp, r0

    /* initialize data and bss
     * not necessary for rom only code
     * */
    bl init_data
    /* call main */
    bl main
    /* trap if returned */
    b .


/* initialize data and bss sections */
.section .text
init_data:

    /* copy rom to ram */
    ldr r0, =_sdata
    ldr r1, =_edata
    ldr r2, =_sidata
    movs r3, #0
    b LoopCopyDataInit

    CopyDataInit:
        ldr r4, [r2, r3]
        str r4, [r0, r3]
        adds r3, r3, #4

    LoopCopyDataInit:
        adds r4, r0, r3
        cmp r4, r1
        bcc CopyDataInit

    /* zero bss */
    ldr r2, =_sbss
    ldr r4, =_ebss
    movs r3, #0
    b LoopFillZerobss

    FillZerobss:
```

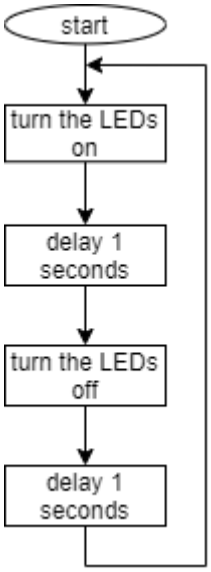
```
        str  r3, [r2]
        adds r2, r2, #4

    LoopFillZerobss:
        cmp r2, r4
        bcc FillZerobss

    bx lr


/* default handler */
.section .text
Default_Handler:
    b Default_Handler


/* main function */
.section .text
main:
    /*//////ENABLE GPIOA CLOCK//////*/
    ldr r1, =(RCC_IOPENR) /* RCC with IOPENR offset */
    ldr r0, [r1]

    ldr r2, =0x1 /* A port enable */
    orrs r0, r0, r2 /* change the corresponding bit to 1 */
    str r0, [r1] /* enable GPIOC clock */

    /*//////ENABLE GPIOC CLOCK//////*/
    ldr r1, =(RCC_IOPENR) /* RCC with IOPENR offset */
    ldr r0, [r1]

    ldr r2, =0x4 /* C port enable */
    orrs r0, r0, r2 /* change the corresponding bit to 1 */
    str r0, [r1] /* enable GPIOC clock */

    /*//////SETUP A6 MODER//////*/
    ldr r1, =(GPIOA_MODER)
    ldr r0, [r1]

    ldr r2, =0x3000
    mvns r2, r2
    ands r0, r0, r2
    str r0, [r1] /* writing back the modified MODER bits */

    /*//////SETUP C6 MODER//////*/
    ldr r1, =(GPIOC_MODER)
    ldr r0, [r1]

    ldr r2, =0x2000
    mvns r2, r2
    ands r0, r0, r2
    ldr r2, =0x1000
    orrs r0, r0, r2 /* ands and orrs to turn the C6 to output */
    str r0, [r1] /* writing back the modified MODER bits */
```

```
LED_OFF:

ldr r1, =(GPIOC_ODR) /* GPIOC with ODR offset */
ldr r0, [r1]
ldr r2, =0x40
mvns r2, r2
ands r0, r0, r2 /* and operation to set the given bit low */
str r0, [r1]

CHECK_INPUT_LOOP_1:

ldr r1, =(GPIOA_IDR) /* GPIOA with IDR ofset */
ldr r0, [r1]
movs r2, #0x40
ands r0, r0, r2
cmp r0, r2
beq LED_ON
b CHECK_INPUT_LOOP_1

LED_ON: /* branch to turn the led on */

ldr r1, =GPIOC_ODR
ldr r0, [r1]
ldr r2, =0x40
orrs r0, r0, r2
str r0, [r1]

CHECK_INPUT_LOOP_2:

ldr r1, =(GPIOA_IDR) /* GPIOA with IDR ofset */
ldr r0, [r1]
movs r2, #0x40
mvns r3, r2
orrs r0, r0, r3
cmp r0, r3
beq LED_OFF
b CHECK_INPUT_LOOP_2

/* for(;;); */
b .

/* this should never get executed */
nop
```

The code first sets the GPIOA and GPIOC ports clocks for reading and writing data to them. Then the GPIOC and GPIOA MODER's are set to certain values that correspond to input or output states. After the pins are set and ready to be used, the LED blinker code is run.

C. Problem 3

For this problem we can set all the pins at the same time for MODER modification. We know all the other addresses so we can use the information to write the code.



The only difference from the first problem will be the ports and the multiple LEDs.

Now we can write the code according to the information.

```
.syntax unified
.cpu cortex-m0
.fpu softvfp
.thumb


/* make linker see this */
.global Reset_Handler

/* get these from linker script */
.word _sdata
.word _edata
.word _sbss
.word _ebss


/* define peripheral addresses from RM0444 */
.equ RCC_IOPENR,    (0x40021034)

.equ GPIOA_MODER,   (0x50000000)
.equ GPIOC_MODER,   (0x50000800)

.equ GPIOA_IDR,     (0x50000010)
.equ GPIOA_ODR,     (0x50000014)

.equ GPIOC_IDR,     (0x50000810)
.equ GPIOC_ODR,     (0x50000814)


/* vector table, +1 thumb mode */
```

```
.section .vectors
vector_table:
    .word _estack             /*     Stack pointer */
    .word Reset_Handler +1    /*     Reset handler */
    .word Default_Handler +1  /*       NMI handler */
    .word Default_Handler +1  /* HardFault handler */
    /* add rest of them here if needed */


/* reset handler */
.section .text
Reset_Handler:
    /* set stack pointer */
    ldr r0, =_estack
    mov sp, r0

    /* initialize data and bss
     * not necessary for rom only code
     * */
    bl init_data
    /* call main */
    bl main
    /* trap if returned */
    b .


/* initialize data and bss sections */
.section .text
init_data:

    /* copy rom to ram */
    ldr r0, =_sdata
    ldr r1, =_edata
    ldr r2, =_sidata
    movs r3, #0
    b LoopCopyDataInit

    CopyDataInit:
        ldr r4, [r2, r3]
        str r4, [r0, r3]
        adds r3, r3, #4

    LoopCopyDataInit:
        adds r4, r0, r3
        cmp r4, r1
        bcc CopyDataInit

    /* zero bss */
    ldr r2, =_sbss
    ldr r4, =_ebss
    movs r3, #0
    b LoopFillZerobss

    FillZerobss:
        str  r3, [r2]
```

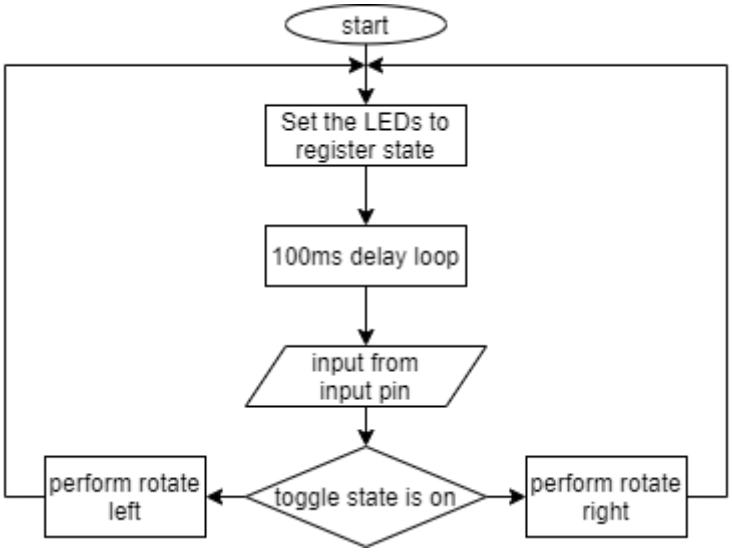
```
        adds r2, r2, #4

    LoopFillZerobss:
        cmp r2, r4
        bcc FillZerobss

    bx lr


/* default handler */
.section .text
Default_Handler:
    b Default_Handler


/* main function */
.section .text
main:

    /*//////ENABLE GPIOA CLOCK//////*/
    ldr r1, =(RCC_IOPENR) /* RCC with IOPENR offset */
    ldr r0, [r1]

    ldr r2, =0x1 /* for A ports */
    orrs r0, r0, r2 /* change the corresponding bit to 1 */
    str r0, [r1] /* enable GPIOC clock */

    /*//////SETUP A PINS MODER//////*/
    ldr r1, =(GPIOA_MODER)
    ldr r0, [r1]

    ldr r2, =0xAAAA0
    mvns r2, r2
    ands r0, r0, r2
    ldr r2, =0x55550
    orrs r0, r0, r2
    str r0, [r1] /* writing back the modified MODER bits */


    LED_ON: /* branch to turn the led on */

    ldr r1, =(GPIOA_ODR) /* GPIOA with ODR offset */
    ldr r0, [r1]
    ldr r2, =0xFF0 /* to turn off all 8 bits */
    orrs r0, r0, r2 /* orr operation to set the given bits high */
    str r0, [r1]

    ldr r4, =0xC35000 /* 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 r1, =(GPIOA_ODR) /* GPIOA with ODR offset */
ldr r0, [r1]
ldr r2, =0xFF0 /* to turn off all 8 bits */
mvns r2, r2
ands r0, r0, r2 /* and operation to set the given bits low */
str r0, [r1]

ldr r4, =0xC35000

OFF_DELAY: /* off time delay */

cmp r4, #0x0
beq LED_ON
subs r4, r4, #1
b OFF_DELAY

/* for(;;); */
b .

/* this should never get executed */
nop
```

The code starts with enabling GPIOA pins for clock. After that the GPIOA MODER values are modified to make all the [A4-A11] pins output mode. Lastly the LED on and off code is modified to turn on and off all the [A4-A11] pins.

D. Problem 4

With this problem we have to implement an input read and toggle setting. After setting all the pins for output and input values, we can write algorithm to shift the bits and light the LED's accordingly.



As we can see from the basic flowchart the code will work by having the state of the LEDs memorized in a state register and we will perform rotate instruction on that register to shift the LEDs.

```
.syntax unified
.cpu cortex-m0
.fpu softvfp
.thumb


/* make linker see this */
.global Reset_Handler

/* get these from linker script */
.word _sdata
.word _edata
.word _sbss
.word _ebss


/* define peripheral addresses from RM0444 */
.equ RCC_IOPENR,    (0x40021034)

.equ GPIOA_MODER,   (0x50000000)
.equ GPIOB_MODER,   (0x50000400)

.equ GPIOA_IDR,     (0x50000010)
.equ GPIOA_ODR,     (0x50000014)

.equ GPIOB_IDR,     (0x50000410)
.equ GPIOB_ODR,     (0x50000414)


/* vector table, +1 thumb mode */
.section .vectors
vector_table:
    .word _estack             /*     Stack pointer */
    .word Reset_Handler +1    /*     Reset handler */
    .word Default_Handler +1  /*       NMI handler */
    .word Default_Handler +1  /* HardFault handler */
    /* add rest of them here if needed */


/* reset handler */
.section .text
Reset_Handler:
    /* set stack pointer */
    ldr r0, =_estack
    mov sp, r0

    /* initialize data and bss
     * not necessary for rom only code
     * */
    bl init_data
    /* call main */
    bl main
    /* trap if returned */
    b .
```

```
/* initialize data and bss sections */
.section .text
init_data:

    /* copy rom to ram */
    ldr r0, =_sdata
    ldr r1, =_edata
    ldr r2, =_sidata
    movs r3, #0
    b LoopCopyDataInit

    CopyDataInit:
        ldr r4, [r2, r3]
        str r4, [r0, r3]
        adds r3, r3, #4

    LoopCopyDataInit:
        adds r4, r0, r3
        cmp r4, r1
        bcc CopyDataInit

    /* zero bss */
    ldr r2, =_sbss
    ldr r4, =_ebss
    movs r3, #0
    b LoopFillZerobss

    FillZerobss:
        str  r3, [r2]
        adds r2, r2, #4

    LoopFillZerobss:
        cmp r2, r4
        bcc FillZerobss

    bx lr


/* default handler */
.section .text
Default_Handler:
    b Default_Handler


/* main function */
.section .text
main:

    /*//////ENABLE GPIOA CLOCK//////*/
    LDR r1, =(RCC_IOPENR) /* RCC with IOPENR offset */
    LDR r0, [r1]

    LDR r2, =0x1 /* A port enable */
```

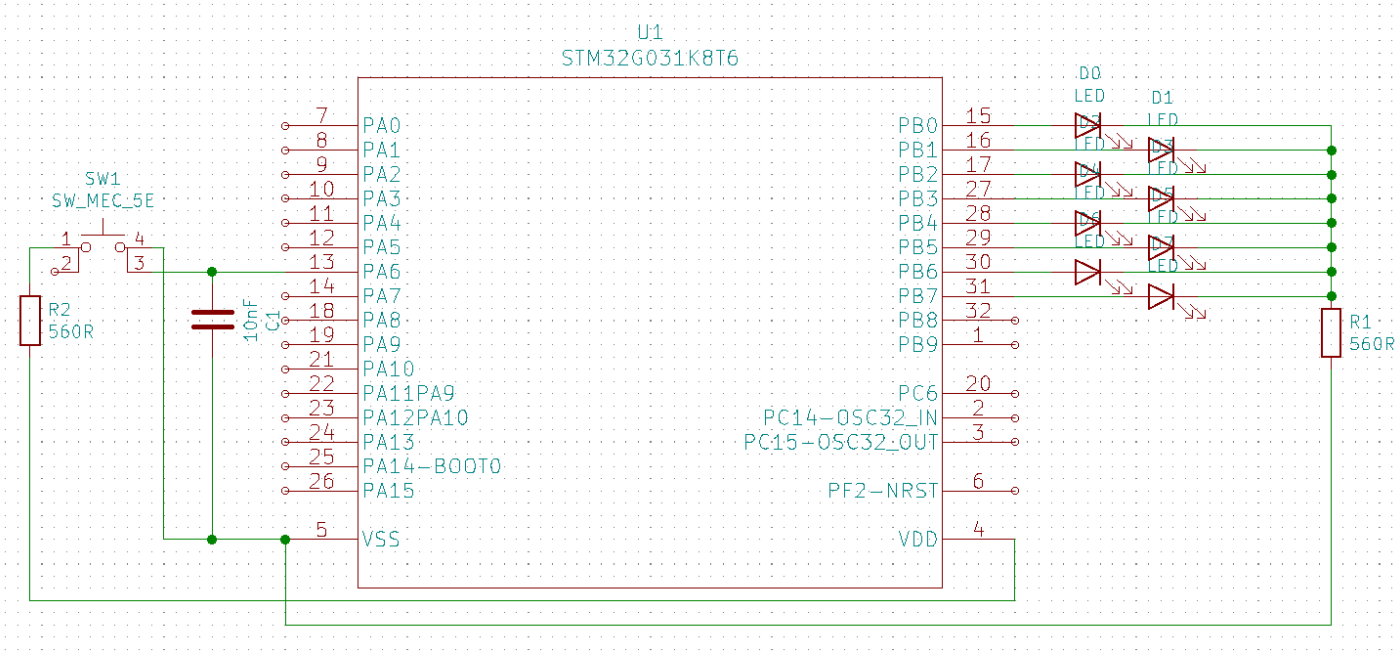
```
    ORRS r0, r0, r2 /* change the corresponding bit to 1 */
    STR r0, [r1] /* enable GPIOA clock */

    /*//////ENABLE GPIOB CLOCK//////*/
    LDR r1, =(RCC_IOPENR) /* RCC with IOPENR offset */
    LDR r0, [r1]

    LDR r2, =0x2 /* for B port */
    ORRS r0, r0, r2 /* change the corresponding bit to 1 */
    STR r0, [r1] /* enable GPIOB clock */

    /*//////SETUP A6 MODER//////*/
    LDR r1, =(GPIOA_MODER)
    LDR r0, [r1]

    LDR r2, =0x3000
    MVNS r2, r2
    ANDS r0, r0, r2
    STR r0, [r1] /* writing back the modified MODER bits */

    /*//////SETUP B PINS MODER//////*/
    LDR r1, =(GPIOB_MODER)
    LDR r0, [r1]

    LDR r2, =0xAAAA
    MVNS r2, r2
    ANDS r0, r0, r2
    LDR r2, =0x5555
    ORRS r0, r0, r2
    STR r0, [r1] /* writing back the modified MODER bits */

    /*//////SETUP VARIABLES//////*/
    LDR r1, =GPIOB_ODR /* GPIOB with ODR offset */
    LDR r0, =0x83838383 /* rotate value */
    MOVS r2, #0x1 /* rotate counter */
    MOVS r5, #0x0 /* toggle register */

LED_SET: /* branch to set the led */

    LDR r3, =0xFF
    ANDS r3, r3, r0 /* only take the first 8 bits */
    STR r3, [r1]

    LDR r4,=0x138800 /* cycles for 100ms total delay */

ON_DELAY: /* on time delay */

    SUBS r4, r4, #1
    CMP r4, #0x0
    BNE ON_DELAY

    /*//////INPUT PIN CHECK//////*/
    LDR r7, =(GPIOA_IDR) /* GPIOA with IDR ofset */
    LDR r6, [r7]
    MOVS r3, #0x40
```

```
    ANDS r6, r6, r3
    CMP r6, r3
    BEQ TOGGLE /* if input is present toggle the register */
    B TOGGLE_CHECK /* else go to toggle check */

TOGGLE:
    /* toggle the toggle register */
    MVNS r5, r5

TOGGLE_CHECK:
    /* if the toggle register is set rotate left else rotate right */
    CMP r5, #0x0
    BNE ROTATE_LEFT /* if toggle is not 0 then rotate left */

ROTATE_RIGHT:
    /* rotates right once */
    RORS r0, r0, r2
    B LED_SET

ROTATE_LEFT:
    /* rotating right (period-1) times will be the same as rotating 1 left */
    MOVS r3, #0x7
    RORS r0, r0, r3
    B LED_SET

    /* for(;;); */
    b .

    /* this should never get executed */
    nop
```

The code starts by setting the input and output pins. After all the pins are rightly set, the code continues to set the variables for shift values, toggle status and shift amount. Using these values it first sets the initial 3 LEDs to on mode. Then it continues to the delay loop that will take around 100ms. After the delay loops it check for the input pin if it is set or not and flips the toggle status accordingly. Lastly after the toggle status check it jumps to either right rotate or left rotate branch.

The basics behind using the rotate algorithm built into arm could be explained like this. Because we're setting 8 pins we can use the same ports and 8 bits a row. We use the B ports for this because the PA3 is absent on the external pins. We run into the problem of memory values being 32 bits and that the rotating cant be done only on the first 8 bits. For this problem the solution is to use extending it to 32 bits using 4 copies of the same bit pattern. Example(8bit: 10000011 32bit: 10000011 10000011 10000011 10000011). Now even if we use the ROR instruction once we get the same result of a single rotate right. The last problem we have to solve is to rotate the bits to the left when there are no rotate left instruction built in. For this we can use the (period-1) times rotation to get the same result of a single backward rotation.

For the schematic KiCAD is used to visualize the component plaecements.



Here's the schematic for the component placements. R1 is a high watt resistor and C1 is a bypass filter to reduce input noise.

References

---

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