임베디드시스템설계

Embedded System Capstone Design

ICE3015-001

과제 3

실습팀 4

정보통신공학과 12181855 황용하

정보통신공학과 12191720 곽재현

정보통신공학과 12191765 박승재

Introduction

ATmega4809 는 A 부터 F 까지의 Port 가 있고, 각 Port 에는 8 개의 Pin 이 있다.

6.1 Peripheral Module Address Map

The address map shows the base address for each peripheral. For complete register description and summary for each peripheral module, refer to the respective module chapters.

Table 6-1. Peripheral Module Address Map

Base Address	Name	Description	28-Pin	32-Pin	40-Pin	48-Pin
0x0400	PORTA	Port A Configuration	X	X	X	X
0x0420	PORTB	Port B Configuration				X

각 Port 와 Pin 에 접근하기 위해서는 Base Address 와 Offset 을 이용해야 한다.

예를 들어 Pin 에 값을 읽거나 쓰기 위해서는 Data Direction 을 지정해야 하는데, Port A 의 Data Direction 주소는 0x0400 + 0x00 = 0x0400 이다.

15.5.1 Data Direction

Name: DIR
Offset: 0x00
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
	DIR[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 7:0 – DIR[7:0] Data Direction
This bit field controls output enable for the individual pins of the Port.
Writing a '1' to PORTx.DIR[n] configures and enables pin n as an output pin.
Writing a '0' to PORTx.DIR[n] configures pin n as an input-only pin. Its properties can be configured by writing to the ISC bit in PORTx.PINnCTRL.
PORTx.DIRn controls only the output enable. Setting PORTx.DIR[n] to '1' does not disable the pin input.

```
#define _PORTA_DIR (unsigned char*) (0x0400)

int main(void) {
    *_PORTA_DIR |= 0b10000000; // output PIN7
    *_PORTA_DIR &= ~0b01000000; // input PIN6
    return 0;
}
```

DIR 의 1 은 Output 을 의미하고, 0 은 Input 을 의미한다. Port Register 의 종류는 다양하게 존재하며 avr 헤더에 미리 정의되어 있기 때문에 우리는 #define 을 이용해 레지스터 주소를 직접 정의할 필요는 없다. 하지만, Base Address 와 Offset 을 필요한 레지스터 주소를 계산하는 방법은 알고 있어야 한다.

해당 핀으로 전압을 출력하고 싶다면 OUT 을 1 로 설정하면 된다. _PORTA_OUT 은 Port A 의 Base Address 가 0x0400 이고 Output Value 의 Offset 이 0x04 이므로 0x0404 주소에 값을 넣어주면 된다.

15.5.5 Output Value

Name: OUT
Offset: 0x04
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
	OUT[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 7:0 – OUT[7:0] Output Value
This bit field defines the data output value for the individual pins of the port.
If OUT[n] is written to '1', pin n is driven high.
If OUT[n] is written to '0', pin n is driven low.
In order to have any effect, the pin direction must be configured as output.

```
#define _PORTA_OUT (unsigned char*) (0x0404)
*_PORTA_OUT |= 0b10000000; // PIN7 HIGH
```

PORTA_OUT 역시 avr 헤더에 정의되어 있다. 따라서 #define 없이 아래와 같이 쓸 수 있다.

```
PORTA_OUT |= PIN7_bm;
```

avr 헤더에 정의된 값을 이용해 Port 를 지정할 때는 *을 붙이지 않는다. PIN7_bm 는 0b10000000 와 동일한 값으로, 마찬가지로 avr 헤더에 정의되어 있다.

값을 읽을 때는 Input Register 를 이용한다. Port A 의 Base Address 가 0x0400 이고 Input Value 의 Offset 이 0x08 이므로 0x0408 주소에서 값을 읽어오면 된다.

15.5.9 Input Value

Name: IN
Offset: 0x08
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
	IN[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 7:0 – IN[7:0] Input Value
This register shows the value present on the pins if the digital input driver is enabled. IN[n] shows the value of pin n of the Port. If the digital input buffers are disabled, the input is not sampled and cannot be read.
Writing to a bit of PORTx.IN will toggle the corresponding bit in PORTx.OUT.

아두이노 Due 와 마찬가지로 4809 에서도 Input Pullup 을 사용할 수 있다.

```
PORTA_DIR &= ~PIN6_bm; // input PIN6
PORTA_PIN6CTRL |= PORT_PULLUPEN_bm; // pullup PIN6
if (PORTA_IN & PIN6_bm) { // check PIN6 HIGH
```

PORTA_PIN6CTRL 에 PORT_PULLUPEN_bm 을 넣어주게 되면 Pin 6 을 Pullup 하게 된다.

15.5.12 Pin n Control

Name: PINCTRL
Offset: 0x10 + n*0x01 [n=0..7]
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
	INVEN				PULLUPEN	ISC[2:0]		
Access	R/W				R/W	R/W	R/W	R/W
Reset	0				0	0	0	0

Bit 7 – INVEN Inverted I/O Enable

Value	Description
0	Input and output values are not inverted
1	Input and output values are inverted

Bit 3 – PULLUPEN Pullup Enable

Value	Description
0	Pullup disabled for pin n
1	Pullup enabled for pin n

Pin n Control 의 3 번 비트를 1 로 설정하면 Pullup 이 활성화된다. `PORT_PULLUPEN_bm` 는 해당 비트만 1 로 설정된 상수이다.

`util/delay.h` 헤더를 이용하면 원하는 시간만큼 MCU 의 동작을 일시정지할 수 있다. 강의자료에는 `avr/delay.h` 헤더를 사용하지만, 최신버전 Microchip Studio 에서 컴파일하면 `avr/delay.h` 대신 `util/delay.h` 헤더를 사용하라고 경고가 나온다.

`_delay_ms` 와 `_delay_us` 가 있는데, 각각 ms(밀리초), us(마이크로초) 단위로 동작을 일시정지 시킨다. 함수의 입력 인자의 타입은 double 이라 실수값을 집어넣어도 동작한다.

```
#define F_CPU 3333333

#include <avr/io.h>
#include <util/delay.h>

int main(void) {
    PORTA_DIR |= PIN7_bm; // output PIN7
    while (1) {
        PORTA_OUT |= PIN7_bm;
        _delay_ms(1000);
        PORTA_OUT &= ~PIN7_bm;
        _delay_ms(1000);
    }
    return 0;
}
```

위는 Pin 7 에 연결된 LED 를 1 초마다 깜빡이는 코드이다. `PORTA_DIR` 으로 Pin 7 을 Output 으로 사용하는 것을 지정하고, `PORTA_OUT` 으로 출력값을 설정했다. `F_CPU` 는 MCU 의 주파수를 정의한 것인데, 4809 는 20MHz 를 사용하고 6 배 Scaler 가 들어가 있기 때문에 20000000/6=3333333 를 정의했다. `F_CPU` 가 올바르게 설정되어 있지 않다면, delay 함수들은 정상적으로 동작하지 않는다.

Problem

1. Implement the _delay_ms, _delay_us functions directly (Search on google).
2. Implement the traffic lights by using PORT register, toggle each 3s.
3. Using the IN register, modify the code so that when the button is pressed, the toggle time changes sequentially to 1, 3, 5 seconds.
4. Analyze the following code and print out the results.

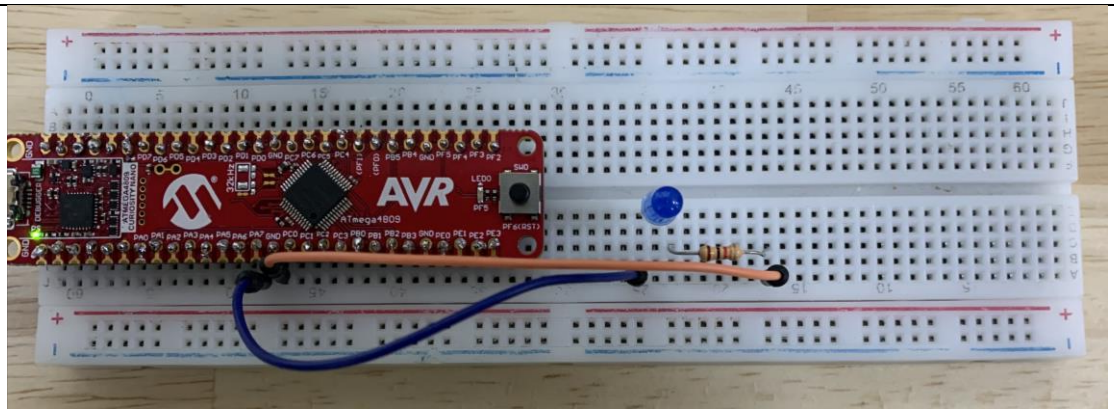
Result

Problem 1

```c
#define F_CPU 3333333

#include <avr/io.h>
// #include <util/delay.h>

void delay_ms(double ms) {
    const long long tick = (long long) ms * 42;
    for (volatile long long i = 0; i < tick; i += 1);
}

void delay_us(double us) {
    const long long tick = (long long) us * 0.042;
    for (volatile long long i = 0; i < tick; i += 1);
}

int main(void) {
    PORTA_DIR |= PIN7_bm;
    while (1) {
        PORTA_OUT |= PIN7_bm; // HIGH
        delay_ms(3000);
        PORTA_OUT &= ~PIN7_bm; // LOW
        delay_ms(3000);
    }
    return 0;
}
```

```c
#define F_CPU 3333333

#include <avr/io.h>
```

```c
// #include <util/delay.h>

void delay_ms(double ms) {
    const long long tick = (long long) ms * 42;
    for (volatile long long i = 0; i < tick; i += 1);
}

void delay_us(double us) {
    const long long tick = (long long) us * 0.042;
    for (volatile long long i = 0; i < tick; i += 1);
}


int main(void) {
    PORTA_DIR |= PIN7_bm;
    while (1) {
        PORTA_OUT |= PIN7_bm; // HIGH
        delay_us(5000000);
        PORTA_OUT &= ~PIN7_bm; // LOW
        delay_us(5000000);
    }
    return 0;
}
```

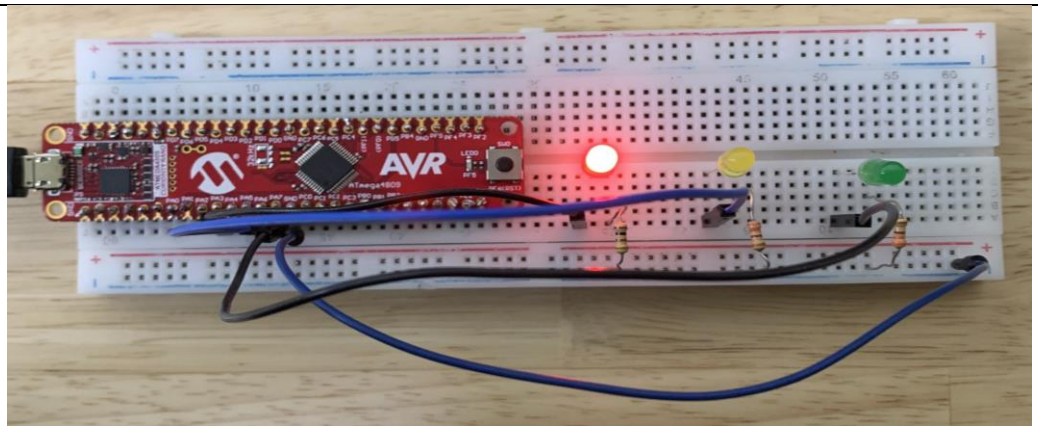


첫 번째 코드는 LED 를 달아서 3000ms 간격으로 깜빡이는지 스톱워치로 측정하며 `tick` 에 곱해지는 계수를 찾았다. 밀리 초에 42 를 곱한 횟수만큼 반복문으로 돌리니 0.1ms 가 나왔다. `volatile` 은 컴파일러가 코드 최적화를 금지하는 문법인데, 컴파일러 최적화로 인해 반복문이 삭제되는 것을 막기 위해 넣었다.

두 번째 코드는 LED 를 달아서 5000000us 간격으로 LED 가 깜빡이는지 앞서 구한 밀리 초에 곱해진 계수를 활용해, `tick` 에 곱해진 계수에 1/1000 을 곱해 마이크로 초 계수를 구했다. 마이크로 초에 0.042 를 곱한 횟수만큼 반복문으로 돌리니 0.1us 가 나왔다.

Problem 2

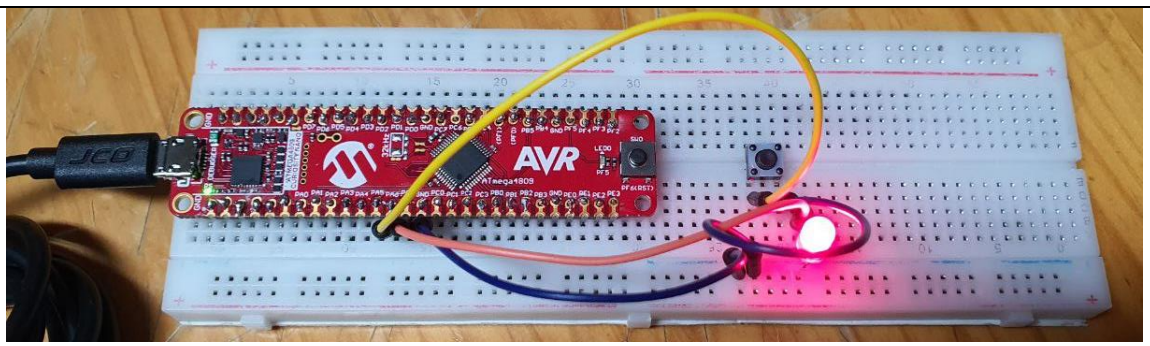
```c
#define F_CPU 3333333

#include <avr/delay.h>
#include <util/io.h>

int main(void) {
    PORTA_DIR |= PIN3_bm | PIN5_bm | PIN7_bm; // output LED

    while (1) {
        PORTA_OUT |= PIN3_bm;
        _delay_ms(3000);
        PORTA_OUT &= ~PIN3_bm;
        _delay_ms(3000);

        PORTA_OUT |= PIN5_bm;
        _delay_ms(3000);
        PORTA_OUT &= ~PIN5_bm;
        _delay_ms(3000);

        PORTA_OUT |= PIN7_bm;
        _delay_ms(3000);
        PORTA_OUT &= ~PIN7_bm;
        _delay_ms(3000);
    }
    return 0;
}
```



위의 코드는 3 개의 LED 를 사용해서 3 초 간격으로 점멸하는 코드이다. 해당 코드에서 `while` 내부를 살펴보면, `PIN3_bm`, `PIN5_bm`, `PIN7_bm` 를 통해 `PORTA` 에서 3 번 핀, 5 번 핀, 7 번 핀을 `output LED` 로 설정하고, 3 초 간격으로 점멸하는 코드이다.

Problem 3

```c
#define F_CPU 3333333

#include <avr/io.h>
#include <util/delay.h>

int clicked = 0; // boolean
int duration = 1000; // [ms]
int tick = 0; // [ms]

int main(void) {
    PORTA_DIR |= PIN7_bm; // output LED
    PORTA_DIR &= ~PIN6_bm; // input Switch
    PORTA_PIN6CTRL |= PORT_PULLUPEN_bm;

    while (1) {
        if (!clicked && (PORTA_IN & PIN6_bm) == 0) { // pressed
            clicked = 1;
            duration = (duration + 2000) % 6000; // 1s 3s 5s repeat
            tick = -100; // delay 100ms
            PORTA_OUT &= ~PIN7_bm; // LED off
        } else if (clicked && (PORTA_IN & PIN6_bm) != 0) { // released
            clicked = 0;
        }

        if (tick == 0) {
            PORTA_OUTTGL |= PIN7_bm; // LED toggle
        } else if (tick == duration) {
            tick = -duration;
        }
        _delay_us(5300); // 5.3ms left
        tick += 10; // 10ms
    }
    return 0;
}
```

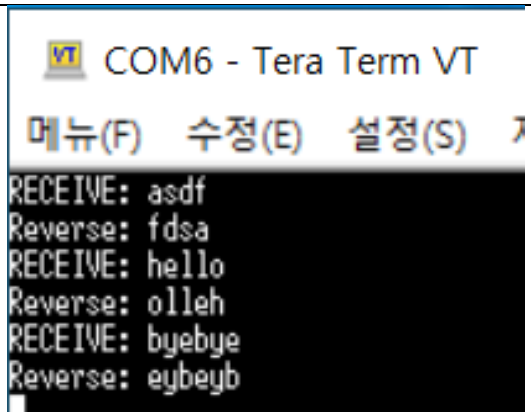


clicked 는 방금 전에 버튼을 눌렀는지를 나타내는 변수이다. clicked 를 이용해 버튼을 꾹 누르고 있을 때, LED 점멸 간격이 계속 변하지 않도록 설정했다. duration 은 LED 점멸 간격을 나타내는 변수로 ms 단위이다. duration = (duration + 2000) % 6000 로 1 초, 3 초, 5 초를 순환하도록 작성했다.

Problem 4

```
#define F_CPU 3333333

#include <avr/io.h>
#include <util/delay.h>

void USART3_init(void);
void USART3_Transmit(unsigned char data);
unsigned char USART3_Receive(void);
void USART3_Transmit_String(unsigned char data[]);
void USART3_Transmit_Reverse();

#define BAUD_RATE 9600
#define USART_BAUD_RATE(BAUD_RATE) ((float) (F_CPU * 64) / (16 * (float)
BAUD_RATE) + 0.5)


void USART3_init(void) {
    PORTB_DIR |= PIN0_bm;
    PORTB_DIR &= ~PIN1_bm;
    USART3_BAUD = (uint16_t) USART_BAUD_RATE(BAUD_RATE);
    USART3_CTRLB |= USART_TXEN_bm | USART_RXEN_bm;
}

void USART3_Transmit(unsigned char data) {
    while (!((USART3_STATUS) &USART_DREIE_bm));
    USART3_TXDATAL = data;
}

unsigned char USART3_Receive(void) {
    while (!((USART3_STATUS) &USART_RXCIF_bm));
    return USART3_RXDATAL;
}

void USART3_Transmit_String(unsigned char data[]) {
    int i = 0;
    while (data[i] != '\0') {
        USART3_Transmit(data[i]);
        i++;
```

```
        _delay_ms(10);
    }
}

void USART3_Transmit_Reverse() {
    int cnt = 0;
    unsigned char buf[20] = {};

    while (1) {
        buf[cnt] = USART3_Receive();
        if (buf[cnt] == '\0' || buf[cnt] == '\n' || buf[cnt] == '\r')
            break;
        cnt++;
    }

    USART3_Transmit_String("RECEIVE: ");
    USART3_Transmit_String(buf);

    USART3_Transmit_String("Reverse: ");
    for (int i = cnt - 1; i >= 0; i--) {
        USART3_Transmit(buf[i]);
    }
    USART3_Transmit_String("\n");
}

int main(void) {
    USART3_init();

    while (1) {
        USART3_Transmit_Reverse();
        _delay_ms(1000);
    }
    return 0;
}
```



위의 코드는 USART3_Transmit_Reverse 를 사용해서 문자열을 받고, 받은 문자열을 거꾸로 만들어서 통신을 받는 형태의 코드이다. buf[cnt] = USART3_Receive();를 사용해

USART3 에서 데이터를 받아 정렬한다. for 구문을 사용해서 USART3_Transmit_Reverse 를 사용해 얻은 데이터들을 역순으로 출력한다. 이 때 사용된 USART3_Transmit 는 USART3 에 송신된 데이터를 모두 출력하는 역할을 한다.

Conclusion

Problem 1

_delay_ms(),_delay_us() 함수는 <util/delay.h> 라이브러리에서 지원하는 함수이다. _delay_ms() 함수의 최대 지연 값은 262.14ms/F_CPU(MHz)이며, _ delay_us() 함수의 최대 지연 값은 768us/F_CPU(MHz)이다. 이 함수들은 CPU 의 클럭 주기를 사용하여 지연 시간을 생성하며, CPU 클럭 주기나 클럭 주파수에 따라 약간의 오차가 발생할 수 있다.

Problem 2

아래의 코드는 DIR 의 설정을 다르게 해서 구현을 했다. 이 방법 말고도 여러 다른 방법들이 있다.

```
#include <avr/io.h>
#include <util/delay.h>

#define PORTA         (uint8_t *)        (0x0400)
#define PORTA_DIR     (unsigned char *)  (0x0400)
#define PORTA_DIRSET  (unsigned char *)  (0x0401)
#define PORTA_DIRCLR  (unsigned char *)  (0x0402)
#define PORTA_OUT     (unsigned char *)  (0x0404)
#define PORTA_OUTSET  (unsigned char *)  (0x0405)
#define PORTA_OUTCLR  (unsigned char *)  (0x0406)
#define PORTA_IN      (unsigned char *)  (0x0408)

int main(void)
{
   *PORTA_DIR = 0x85; //1000 0101 : PA0, PA2, PA7 사용

    while (1)
    {
         *PORTA_OUT |= (1<<7);
        _delay_ms(3000);
         *PORTA_OUT &= ~(1<<7);
        _delay_ms(3000);

         *PORTA_OUT |= (1<<2);
        _delay_ms(3000);
         *PORTA_OUT &= ~(1<<2);
```

```
        _delay_ms(3000);

         *PORTA_OUT |= (1<<0);
        _delay_ms(3000);
           *PORTA_OUT &= ~(1<<0);
        _delay_ms(3000);
    }
}
```

위 코드는 PA0, PA2, PA7 핀을 출력으로 사용하고, 이들 핀을 이용해 LED 를 제어하는 코드이다. 먼저, *PORTA_DIR = 0x85 를 사용해서 PA0, PA2, PA7 핀을 출력으로 사용하도록 설정한다. 그리고, while 문 안에서 PORTA_OUT |= (1<<N); 코드를 이용해서 PA0, PA2, PA7 핀 중에서 하나를 HIGH 로 설정하면, 해당 LED 가 켜진다. 이후에 _delay_ms(3000) 함수를 사용해서 3 초 동안 대기한 후에, *PORTA_OUT &= ~(1<<N); 코드를 이용해서 PA0, PA2, PA7 핀 중에서 HIGH 로 설정된 핀의 출력 값을 LOW 로 변경하면, 해당 LED 가 꺼진다.

Problem 3

3 번 문제는 버튼을 누를 때마다 토글 주기를 1 초, 3 초, 5 초로 순환하도록 구현하는 문제이다. 현재 LED 점멸 상태를 나타내는 별도의 변수를 사용하지 않고, PORTA_OUTTGL 명령을 사용해 LED 를 토글한다.

15.5.8 Output Value Toggle

Name: OUTTGL
Offset: 0x07
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
	OUTTGL[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 7:0 – OUTTGL[7:0] Output Value Toggle
This register can be used instead of a read-modify-write to toggle the output value of individual pins.
Writing a '1' to OUTTGL[n] will toggle the corresponding bit in PORTx.OUT.
Reading this bit field will always return the value of PORTx.OUT.

PORTA_OUT 과 거의 동일한데, 원하는 비트를 1 로 설정하면 그 자리의 값이 반전된다. Output Value Toggle Register 의 주소는 0x0400 + 0x07 = 0x0407 이다.

tick 이 10 씩 증가하기 때문에 while 한 사이클 당 10ms 가 소요되야 한다. tick += 10 이전까지 약 4.7ms 가 소요되기 때문에 _delay_us(5300);를 넣어 10ms 가 되도록 맞춰주었다. tick 간격을 짧게 잡은 이유는 while 한 사이클이 길어지면 버튼의 상태를 체크하는 코드가 더 드물게 호출된다. tick 간격이 길다면 버튼 짧게 눌렀다 땠을 때, PORTA_IN 가 0 이 되는 것을

확인하기 전에 1 로 바뀌어 버려 버튼을 누르지 않은 것으로 인식할 수 있다. 따라서 가능한 `tick` 을 짧게 해 버튼의 상태를 체크하는 코드가 자주 호출되게 만들었다.

Problem 4

`USART` 모듈을 사용하여 데이터를 전송하기 위해서는 `USART_TXEN_bm` 과 `USART_RXEN_bm` 을 1 로 설정하여 `USART` 의 전송과 수신을 활성화해야 합니다. 이후 `USART_BAUD_RATE(BAUD_RATE)` 함수를 사용하여 `USART` 의 전송 속도를 설정해야 하는데, 이 함수는 레지스터에 로드해야 하는 값을 계산합니다. 이 계산에서 0.5 를 더하는 이유는 계산 값에 가장 가까운 정수로 반올림하기 위함입니다. 이렇게 설정된 `USART3` 모듈은 통신 데이터를 주고받는 역할을 한다.