МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ

КАФЕДРА ЕЛЕКТРОННИХ ПРИСТРОЇВ ТА СИСТЕМ

ЗВІТ

ПРО ВИКОНАННЯ РОЗРАХУНКОВО-ГРАФІЧНОЇ РОБОТИ З ДИСЦИПЛІНИ

«ОСНОВИ ЗАСТОСУВАННЯ МІКРОПРОЦЕСОРІВ»

ВАРІАНТ No 30

ВИКОНАВ: студент гр. ДС-01 Івахненко Максим
(Прізвище, Ім’я)

ПРИЙНЯВ: викладач
Коваленко Євген Юрійович 29.03.2024 р.
(Прізвище, Ім’я) (Оцінка, Підпис) (Дата)

КИЇВ

КПІ ім. ІГОРЯ СІКОРСЬКОГО

2024

Завдання: Написати програму, яка буде керувати RGB стрічкою. Зробити кнопки для керування кожним каналом кольору. Зробити можливість загрузки та вигрузки кольору з EEPROM пам’яті.

За основу візьмемо мікроконтроллер STM8S103F3P6

Для керування стрічкою нам знадобиться три канали ШІМ. Лише у TIM2 є 3 канали для генерації ШІМ.
Канал CH1 виведений на PD4. Буде використовуватись для зеленого кольору
Канал CH2 виведений на PD3. Буде використовуватись для червоного кольору
Канал CH3 виведений на PA3. Буде використовуватись для синього кольору

З керування нам потрібні кнопки для зміни яскравості коєжного каналу:
Btn_R+ PD2
Btn_R- PC7
Btn_G+ PC6
Btn_G- PC5
Btn_B+ PC4
Btn_B- PC3

Та кнопки для роботи з EEPROM Пам’яттю
Btn_FLASH PB4
Btn_LOAD PB5

Спочатку налаштуємо порти GPIO:

Table 21. I/O port configuration summary

Mode	DDR bit	CR1 bit	CR2 bit	Function	Pull-up	P-buffer	Diodes	
							to V_{DD}	to V_{SS}
Input	0	0	0	Floating without interrupt	Off	Off	On	On
	0	1	0	Pull-up without interrupt	On			
	0	0	1	Floating with interrupt	Off			
	0	1	1	Pull-up with interrupt	On			
Output	1	0	0	Open drain output	Off	Off		
	1	1	0	Push-pull output		On		
	1	0	1	Open drain output, fast mode		Off		
	1	1	1	Push-pull, fast mode	Off	On		
	1	x	x	True open drain (on specific pins)	Not implemented		Not implemented (1)	

Конфігурація GPIO для каналів PWM TIM2:

```
// TIM2_CH1
// Output - PushPull - 2MHz
PD_DDR |= (1 << 4);
PD_CR1 |= (1 << 4);
PD_CR2 &= ~(1 << 4);
PD_ODR &= ~(1 << 4);

// TIM2_CH2
// Output - PushPull - 2MHz
PD_DDR |= (1 << 3);
PD_CR1 |= (1 << 3);
PD_CR2 &= ~(1 << 3);
PD_ODR &= ~(1 << 3);

// TIM2_CH3
// Output - PushPull - 2MHz
PA_DDR |= (1 << 3);
PA_CR1 |= (1 << 3);
PA_CR2 &= ~(1 << 3);
PA_ODR &= ~(1 << 3);
```

Конфігурація GPIO для кнопок (усі кнопки підключені до землі з внутрішньою підтяжкою до + та вимкненими зовнішніми перериваннями):

```
PD_DDR &= ~(1 << 2); // Input
PD_CR1 |= (1 << 2); // Pull-Up
PD_CR2 &= ~(1 << 2); // Interrupts disabled

PC_DDR &= ~(1 << 7);
PC_CR1 |= (1 << 7);
PC_CR2 &= ~(1 << 7);

PC_DDR &= ~(1 << 6);
PC_CR1 |= (1 << 6);
PC_CR2 &= ~(1 << 6);

PC_DDR &= ~(1 << 5);
PC_CR1 |= (1 << 5);
PC_CR2 &= ~(1 << 5);

PC_DDR &= ~(1 << 4);
PC_CR1 |= (1 << 4);
PC_CR2 &= ~(1 << 4);

PC_DDR &= ~(1 << 3);
PC_CR1 |= (1 << 3);
PC_CR2 &= ~(1 << 3);


PB_DDR &= ~(1 << 4);
PB_CR1 |= (1 << 4);
PB_CR2 &= ~(1 << 4);

PB_DDR &= ~(1 << 5);
PB_CR1 |= (1 << 5);
PB_CR2 &= ~(1 << 5);

PA_DDR &= ~(1 << 1);
PA_CR1 |= (1 << 1);
PA_CR2 &= ~(1 << 1);

PA_DDR &= ~(1 << 2);
PA_CR1 |= (1 << 2);
PA_CR2 &= ~(1 << 2);
```

Тактуватись мікроконтроллер буде від внутрішнього RC генератора з частотою 2 Мгц. Це джерело тактування у мікроконтроллері обрано за замовчуванням, тому його додатково налаштовувати не потрібно.

Для зручності роботи з станами бітів були зроблені функції типу

```
uint8_t btn_r_plus_is_pressed() {
    return(~PD_IDR & (1 << 2));
}
```

Налаштування TIM2:

```
void tim2_init() {
    // TIM2_ARR = F_Master / F_PWM
    // 2 (MHz) / 125 (Hz) = 16000
    const uint16_t tim2_arrval = 16000;

    TIM2_PSCR = 0x00; // Prescaler = 1
    TIM2_ARRH = tim2_arrval >> 8;
    TIM2_ARRL = tim2_arrval & 0x00FF;

    // TIM2_CH1 init
    TIM2_CCR1H = 0x00;
    TIM2_CCR1L = 0x00;
    TIM2_CCER1 &= ~CC1P; // Active high
    TIM2_CCER1 |= CC1E; // Enable CH1 output
    TIM2_CCMR1 |= (0b110 << 4); //PWM mode 1

    // TIM2_CH2 init
    TIM2_CCR2H = 0x00;
    TIM2_CCR2L = 0x00;
    TIM2_CCER1 &= ~CC2P; // Active high
    TIM2_CCER1 |= CC2E; // Enable CH2 output
    TIM2_CCMR2 |= (0b110 << 4); //PWM mode 1

    // TIM2_CH3 init
    TIM2_CCR3H = 0x00;
    TIM2_CCR3L = 0x00;
    TIM2_CCER2 &= ~CC3P; // Active high
    TIM2_CCER2 |= CC3E; // Enable CH3 output
    TIM2_CCMR3 |= (0b110 << 4); //PWM mode 1

    TIM2_CR1 |= CEN; // Enable TIM2
}
```

Тут ми розрахували значення регістрів таймера для роботи ШІМ з частотою 125 Гц та налаштували 3 канали ШІМ з активним високим рівнем. Регістри захоплення та порівняння нам не потріні, тому ми їх ніяк не налаштовуємо. Дозволяємо роботу кожного каналу а обираємо режим роботи ШІМ та обираємо режим роботи ШІМ згідно таблиці:

Channel configured in output

7	6	5	4	3	2	1	0
Reserved	OC2M[2:0]			OC2PE	Reserved	CC2S[1:0]	
r	rw	rw	rw	rw	r	rw	

Bit 7 Reserved

Bits 6:4 **OC2M[2:0]**: Output compare 2 mode

Bit 3 **OC2PE**: Output compare 2 preload enable

Bit 2 Reserved

Bits 1:0 **CC2S[1:0]**: Capture/compare 2 selection

This bitfield defines the direction of the channel (input/output) as well as the used input.

00: CC2 channel is configured as output

01: CC2 channel is configured as input, IC2 is mapped on TI2FP2

10: CC2 channel is configured as input, IC2 is mapped on TI1FP2

11:CC2 channel is configured as input, IC2 is mapped on TRC. This mode works only if an internal trigger input is selected through the TS bit (TIM5_SMCR register).

Note: CC2S bits are writable only when the channel is off (CC2E = 0 in TIMx_CCER1).

Робота з EEPROM

Для запису яінформації до EEPROM памяті потрібно зняти блокування з неї та дозволити запис.

Робиться це послідовним записом двої чисел в регістр FLASH_DUKR:

Enabling write access to the DATA area

After a device reset, it is possible to disable the DATA area write protection by writing consecutively two values called MASS keys to the FLASH_DUKR register (see *Section 4.8.9: Flash register map and reset values*). These programmed keys are then compared to two hardware key values:

- First hardware key: 0b1010 1110 (0xAE)
- Second hardware key: 0b0101 0110 (0x56)

після чого потрібно дочекатися коли скинеться біт DUL в регістрі FLASH_IAPSR:

```
void eeprom_unlock() {
  if (!(FLASH_IAPSR & 0x02))
  {
    // unlock EEPROM
    FLASH_DUKR = 0xAE;
    FLASH_DUKR = 0x56;
  }
  // wait for acces to write
  while (!(FLASH_IAPSR & DUL));
}
```

Блокування запису робиться скиданням біта DUL

```
void eeprom_lock() {
  FLASH_IAPSR &= ~(DUL);
}
```

Функції запису та читання EEPROM:

```
void eeprom_write(uint16_t mem_cell, uint8_t data) {
    eeprom_unlock();

    uint8_t  *addr;
    addr = (uint8_t *)(EEPROM_FIRST_ADDR + mem_cell); //Initialize  pointer
                                                      //
    __asm sim __endasm; // Disable interrupts
    *addr = data;
    while(EOP != (~FLASH_IAPSR & EOP)); // Wait for writing to complete
    __asm rim __endasm; // Enable interrupts

    eeprom_lock();
}

void eeprom_read(uint16_t mem_cell, uint8_t *data) {
    uint8_t  *addr;
    addr = (uint8_t *)(EEPROM_FIRST_ADDR + mem_cell);

    __asm sim __endasm;
    *data = *addr;
    while(EOP != (~FLASH_IAPSR & EOP));
    __asm rim __endasm;
}
```

Біт EOP встановиться в одиницю коли запис закінчиться.

Робота з кольорами:

Функція запису значень кольорів до регістрів CCR відповідних каналів:

```
void write_color_to_registers(struct Color *color) {
    uint16_t red = normalize_from( &color->r );
    uint16_t green = normalize_from( &color->g );
    uint16_t blue = normalize_from( &color->b );

    TIM2_CCR2H = red >> 8;
    TIM2_CCR2L = red;

    TIM2_CCR1H = green >> 8;
    TIM2_CCR1L = green;

    TIM2_CCR3H = blue >> 8;
    TIM2_CCR3L = blue;
}
```

Оскільки залежність яскравості світлодіодної стрічки від поданої на них напруги є логорифмічною, то нам потрібно вирівняти цю крив експоненціальною функцією:

```
uint16_t normalize_from(uint8_t *val) {
    if(*val == 0) {
        return 0;
    } else {
        float tmp1 = *val;
        float tmp2 = sqrtf(10 * tmp1) / 5;
        return (expf(tmp2));
    }
}
```

Вигрузка і загрузка значень з/у EEPROM:

```
void load_color_from_eeprom(struct Color *color, uint8_t color_cell) {
    uint8_t r, g, b;
    eeprom_read(3*color_cell+0, &r);
    eeprom_read(3*color_cell+1, &g);
    eeprom_read(3*color_cell+2, &b);

    color->r = r;
    color->g = g;
    color->b = b;
}

void write_color_to_eeprom(struct Color *color, uint8_t color_cell) {
    eeprom_write(3*color_cell+0, color->r);
    eeprom_write(3*color_cell+1, color->g);
    eeprom_write(3*color_cell+2, color->b);
}
```

тут ми звертаємся до трбох ячейок памяті EEPROM, в кожну з яких записано значення 8-бітного кольору.

Обробка натиснутих кнопок:

```
void button_hundler(struct Color *color) {
    if(btn_r_plus_is_pressed()) {
        smart_increment(&color->r);
    }

    if(btn_r_minus_is_pressed()) {
        smart_decrement(&color->r);
    }

    if(btn_g_plus_is_pressed()) {
        smart_increment(&color->g);
    }

    if(btn_g_minus_is_pressed()) {
        smart_decrement(&color->g);
    }

    if(btn_b_plus_is_pressed()) {
        smart_increment(&color->b);
    }

    if(btn_b_minus_is_pressed()) {
        smart_decrement(&color->b);
    }

    if(btn_brightness_plus_is_pressed()) {
        smart_increment(&color->r);
        smart_increment(&color->g);
        smart_increment(&color->b);
    }

    if(btn_brightness_minus_is_pressed()) {
        smart_decrement(&color->r);
        smart_decrement(&color->g);
        smart_decrement(&color->b);
    }

    // Problems with reading input status. Button disabled
    //if(btn_flash_is_pressed()) {
    //   load_color_from_eeprom(&rgb, 0);
    //    rgb.r = 100;
    //}

    // Button uses as flash and load button
    if(btn_load_is_pressed()) {
```

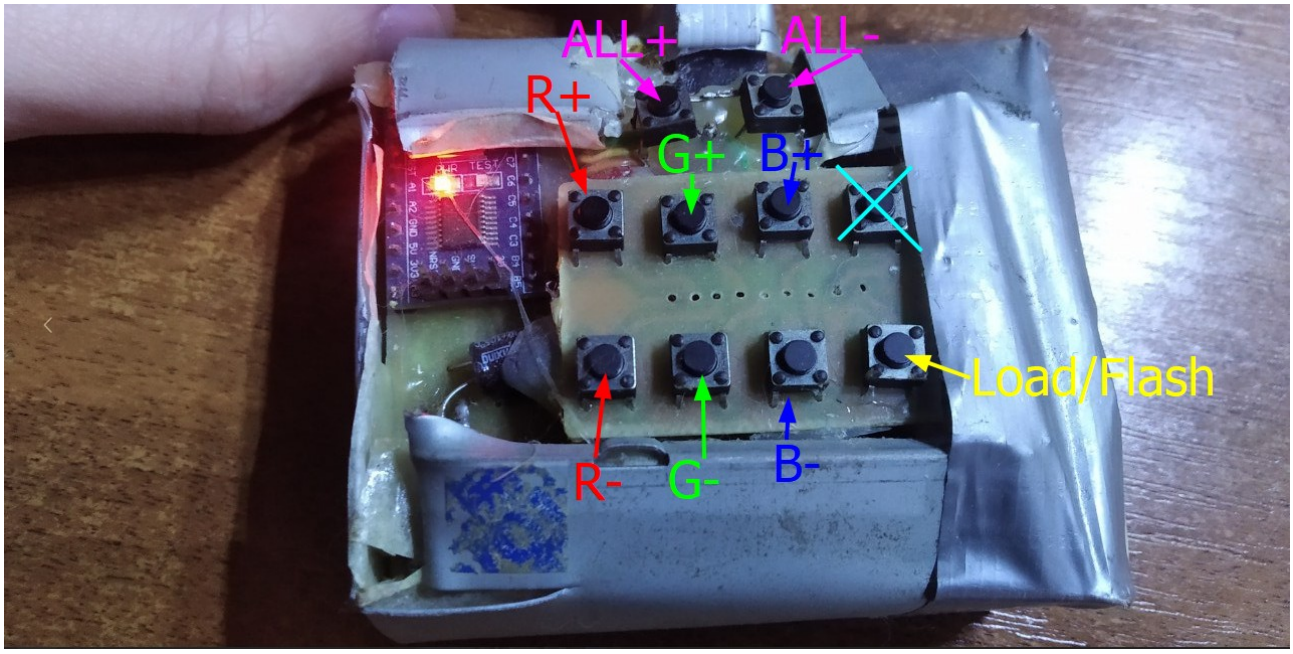
```
    uint8_t counter = 0;
    while(counter < 10 && btn_load_is_pressed()) {
      delay(65535);
      counter += 1;
    }

    struct Color rgb_buf;
    load_color_from_eeprom(&rgb_buf, 0);
    write_color_to_registers(&rgb_buf);
    delay(65535);
    delay(65535);
    delay(65535);

    // Timer with a preview of the color that will be erased in EEPROM to record the new color
    while(counter < 23 && btn_load_is_pressed()) {
      delay(65535);
      delay(65535);
      if (counter % 2 == 0) {
        write_color_to_registers(&rgb_buf);
      }
      else {
        write_color_to_registers(&rgb);
      }
      counter += 1;
    }

    if(counter >= 10 && counter < 23) {
      rgb = rgb_buf;
    }
    else if (counter == 23) {
      write_color_to_eeprom(&rgb, 0);
    }
  }
}
```

Оскільки у мене виникла проблема, через яку я не міг прочитати стан пнопки для прошивання eeprom, а білье кнопок не залишилось, було вирішино зробити одну кнопку для запису кольору та його читання. Це реалізовано за допомогою затримки нажатою кнопки. Якщо кнопку нажати і відразу відпустити то короткочасно покажеться колір який зараз записаний у память EEPROM. Якщо кнопку тримати три секунди то колір вивантажиться з EEPROM, якщо тримати 5 секунд – то колір запишеться до EEPROM.

Головна функція а цикл:

```
struct Color rgb;

int main() {
    __asm sim __endasm; // Disable interrupts

    clk_init();
    gpio_init();
    tim2_init();
    //uart_init();

    __asm rim __endasm; // Enable interrupts

    rgb.r = 0;
    rgb.g = 0;
    rgb.b = 0;

    load_color_from_eeprom(&rgb, 0);

    while(1) {
        button_hundler(&rgb);
        write_color_to_registers(&rgb);
    }
}
```

репозиторій: https://github.com/slplsswkds/fb



Демонстрація зміни яскравості показана у відео brightness.mp4

Демонстрація вивантаження та завантаження кольору до/з EEPROM показана у відео eeprom.mp4