ENGINE

Teaching online electronics, microcontrollers and programming in Higher Education

Programing of embedded systems

9. Bargraf RGB

Lead Partner: Warsaw University of Technology

Authors: Daniel Król

University of Applied Sciences in Tarnow

Erasmus+

GRANT NUMBER: 2020-1-PL01-KA226-HE-095653

Declaration

This report has been prepared in the context of the ENGINE project. Where other published and unpublished source materials have been used, these have been acknowledged.

Copyright

Funding Disclaimer

This project has been funded with support from the European Commission. This report reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Programing of embedded systems

9. Bargraf RGB

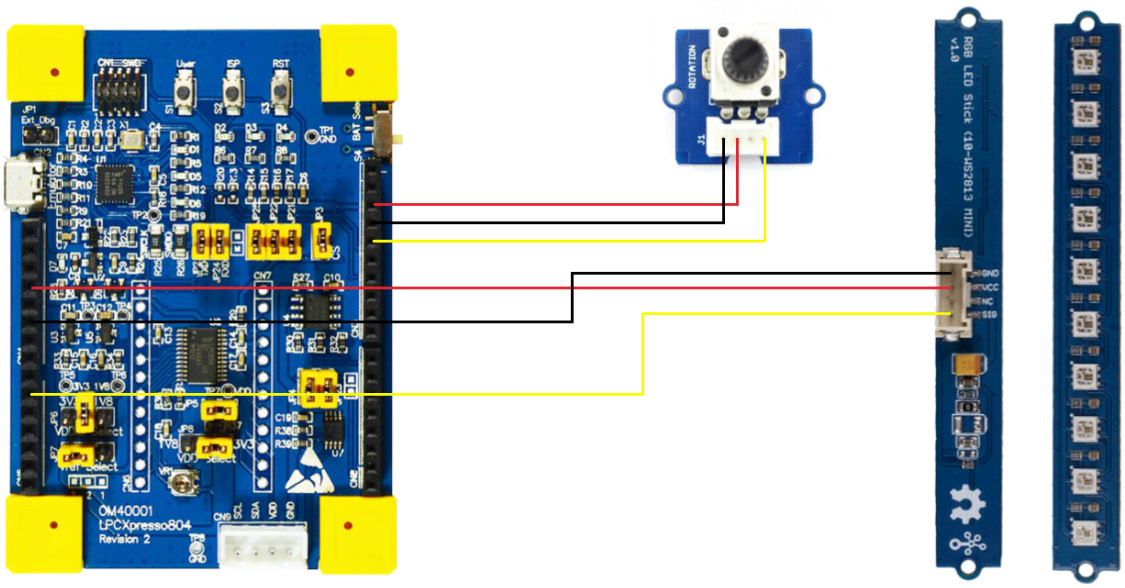
I. Przetwornik A/C

1. Stwórz nowy projekt i nazwij go np. *Lab09*.
2. Skonfiguruj interfejs *SPI* do sterowania diodami *Neopixels* - jak na zajęciach *Lab06*.
3. Skonfiguruj przetwornik A/C (jeden kanał, próbkowanie 10 Hz) - jak na zajęciach *Lab07*.
4. Zmodyfikuj kod głównego pliku projektu:

```
#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "LPC804.h"
#include "fsl_debug_console.h"
#include "fsl_power.h"
#include "oled.h"

#define LEDS 10
#define GET_BIT(k, n)     (k &   (1 << (n)))
#define SET_BIT(k, n)     (k |=  (1 << (n)))
#define CLR_BIT(k, n)     (k &= ~(1 << (n)))

#define CODE_0                                  0b10000
#define CODE_1                                  0b11100

uint32_t colors[LEDS]={0};

static adc_result_info_t gAdcResultInfoStruct;
adc_result_info_t *volatile gAdcResultInfoPtr = &gAdcResultInfoStruct;
volatile uint16_t adcValue = 0;


/* ADC_SEQA_IRQn interrupt handler */
void ADC_ADC_SEQ_A_IRQHANDLER(void) {
          /* Get status flags */
          if (kADC_ConvSeqAInterruptFlag == (kADC_ConvSeqAInterruptFlag & ADC_GetStatusFlags(ADC_PERIPHERAL))) {
                    /* Place your interrupt code here */
                    ADC_GetChannelConversionResult(ADC_PERIPHERAL, 0, gAdcResultInfoPtr);
                    adcValue = gAdcResultInfoStruct.result;
                    /* Clear status flags */
                    ADC_ClearStatusFlags(ADC_PERIPHERAL, kADC_ConvSeqAInterruptFlag);
          }
}

void Neopixels_Send(SPI_Type *base, uint32_t n, uint32_t *value)
{
          uint16_t LED_data=0;

          for(int j=0;j<n;j++) {
                    for(int i=23;i>=0;i--) {
                              LED_data = GET_BIT(value[j], i) ? CODE_1 : CODE_0;

                              while(!(base->STAT & SPI_STAT_TXRDY_MASK));
                              base->TXDAT = LED_data ;
                    }
          }
          // Reset >= 280 us (WS2813)
          LED_data=0;
          for(int j=0;j<225;j++) { // 280 / 1.25
                    while(!(base->STAT & SPI_STAT_TXRDY_MASK));
                    base->TXDAT = LED_data ;
          }
}

void Line(uint16_t v) {

          uint16_t level;
          for(int i=0;i<LEDS;i++) {
                    colors[i] = 0; // black
          }
          level = (LEDS-1)*v/4095;
          colors[level] = 0x00000F; // blue
          Neopixels_Send(SPI0_PERIPHERAL, LEDS, colors);
}
/*
 * @brief   Application entry point.
 */
int main(void) {

          /* Power on ADC. */
          POWER_DisablePD(kPDRUNCFG_PD_ADC0);
          /* Init board hardware. */
          BOARD_InitBootPins();
          BOARD_InitBootClocks();
          BOARD_InitBootPeripherals();
#ifndef BOARD_INIT_DEBUG_CONSOLE_PERIPHERAL
          /* Init FSL debug console. */
          BOARD_InitDebugConsole();
#endif
```

```
        SPI_WriteConfigFlags(SPI0_PERIPHERAL,  kSPI_ReceiveIgnore);

        while(1) {

                Line(adcValue);
        }
        return 0 ;
}
```

5. Podłącz moduł potencjometru oraz moduł *Neopixels* według poniższego schematu:



6. Zbuduj projekt, zaprogramuj układ i sprawdź działanie przykładu.

   Nie przekraczaj wartości 15 na poszczególnych składowych RGB! - aby nie przeciążyć stabilizatora napięcia na płytce z mikrokontrolerem.

II. Zadania

1. Napisz funkcję zapalającą wszystkie diody na kolor zielony do ustalonego poziomu:

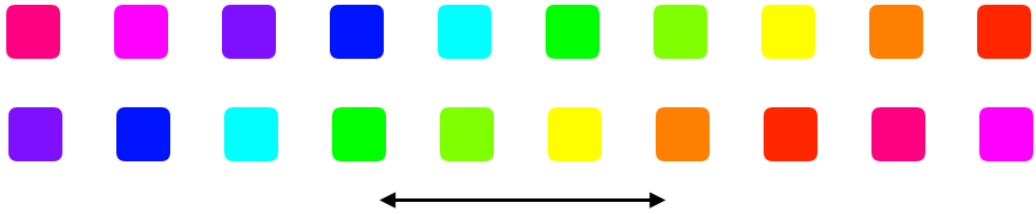
2. Napisz funkcję działającą jak powyżej, ale 4 ostatnie diody mają zapalać się odpowiednio na kolor żółty i czerwony:

3. Dopisz funkcję *hold-peak*, która wyświetla wartość szczytową w kolorze czerwonym. Wartość czsytowa powinna „opadać" znacznie wolniej niż sama linijka świetlna:

4. Napisz funkcję zapalającą wszystkie diody do ustalonego poziomu, na kolor z tablicy LUT:

```
uint32_t colorLUT[LEDS] = {
                    // GRB
                    0x000f00, //red
                    0x080f00, //orange
                    0x0f0f00, //yellow
                    0x0f0800, //chartreuse
                    0x0f0000, //green
                    0x0f000f, //cyan
                    0x00000f, //blue
                    0x00080f, //purple
                    0x000f0f, //violet
                    0x000f08  //magenta

};
```

5. Wpisz funkcję zapalającą wszystkie diody jak powyżej, ale wykonującą rotację kolorów o wartość poziomu:



6. Stwórz własne efekty zależne od wartości zwracanej przez przetwornik A/C. Pamiętaj aby nie przekraczać wartości 15 na poszczególnych składowych RGB!