ENGINE

Teaching online electronics, microcontrollers and programming in Higher Education

Programing of embedded systems

9. Bargraph RGB

Lead Partner: Warsaw University of Technology

Authors: Daniel Krol

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 Where other published and unpublished source materials have been used, these have been acknowledged.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. Bargraph RGB

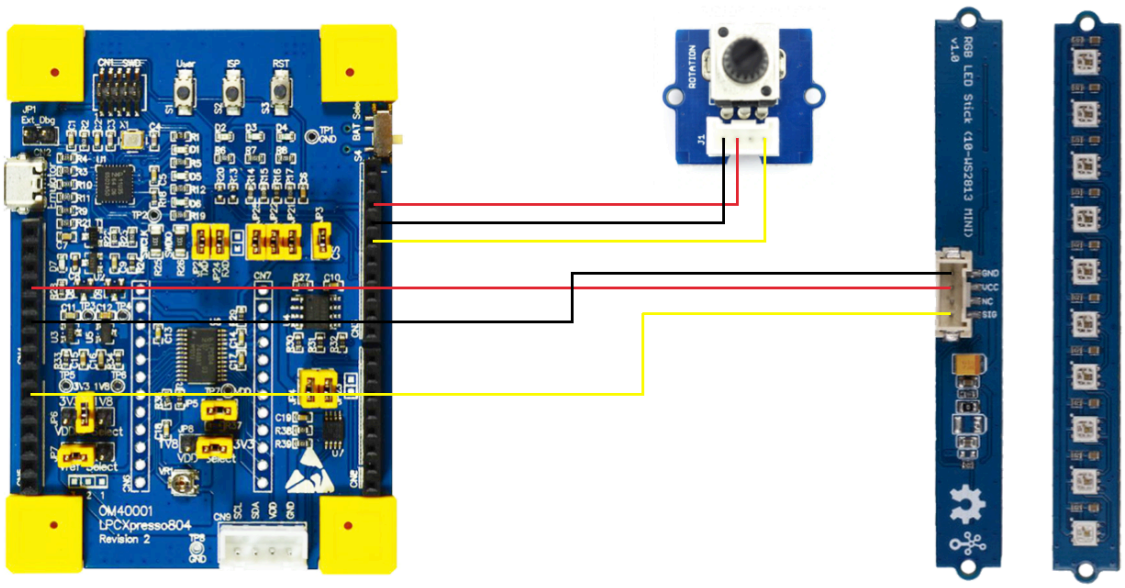
I. A/D Converter

1. Create a new project for the *LPCXpresso804* board and name it eg *Lab09*.
2. Configure the *SPI* interface to control *Neopixels* diodes - as in the *Lab06* manual.
3. Configure the *A/D* converter (one channel, 10 Hz sampling) - as in the *Lab07* manual.
4. Go to the main project file and modify the code as below:

```
#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. Connect the potentiometer and the *Neopixels* modules according to the diagram below:



6. Build a project, program the microcontroller and test the example.

   Do not exceed the value of 15 for individual RGB components! - in order not to overload the voltage stabilizer on the microcontroller board.

II. Exercises

1. Write a function that turns all the LEDs green to a set level:

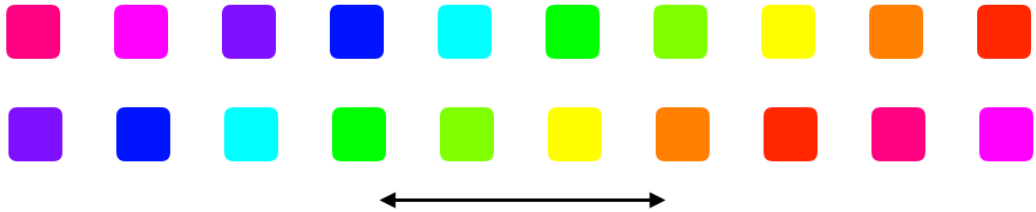
2. Write a function that works as above, but the last 4 LEDs should light up in yellow and red respectively:

3. Add a *hold-peak* function that displays the peak value in red. The peak value should "fall" much slower than the line of light itself:

4. Write a function that lights up all the LEDs to a set level, on a color from the *LUT* table:

```
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. Write a function that lights up all the LEDs as above, but rotates the colors by the level value:



6. Create your own effects depending on the value returned by the A / D converter. Remember not to exceed the value of 15 for each RGB component!