

# ENGINE

Teaching online electronics, microcontrollers and programming in Higher Education

---

## Programing of embedded systems

### 8. Analog Joystick

---

**Lead Partner: Warsaw University of Technology**

**Authors: Daniel Krol**

University of Applied Sciences in Tarnow

# Programing of embedded systems

## 8. Analog Joystick

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 Where other published and unpublished source materials have been used, these have been acknowledged.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

© Copyright 2021 - 2023 the ENGINE Consortium

Warsaw University of Technology (Poland)

International Hellenic University (IHU) (Greece)

European Lab for Educational Technology- EDUMOTIVA (Greece)

University of Padova (Italy)

University of Applied Sciences in Tarnow (Poland)

All rights reserved.



This document is licensed to the public under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.

### Funding Disclaimer

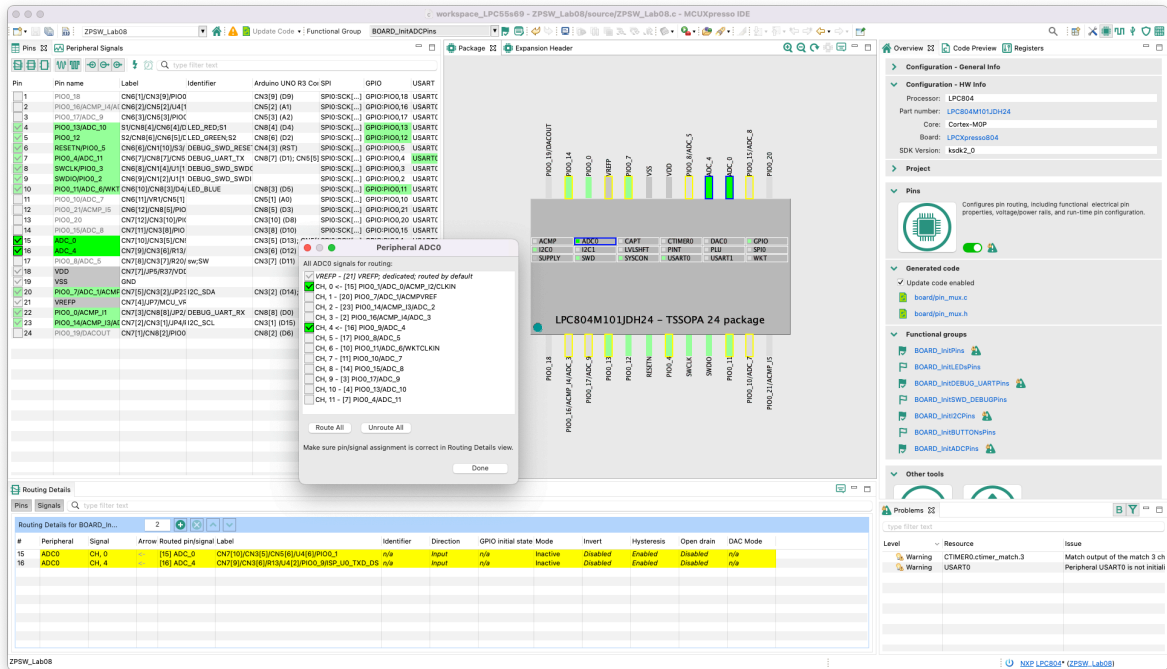
This project has been funded with support from the European Commission. This report reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

# Programming of embedded systems

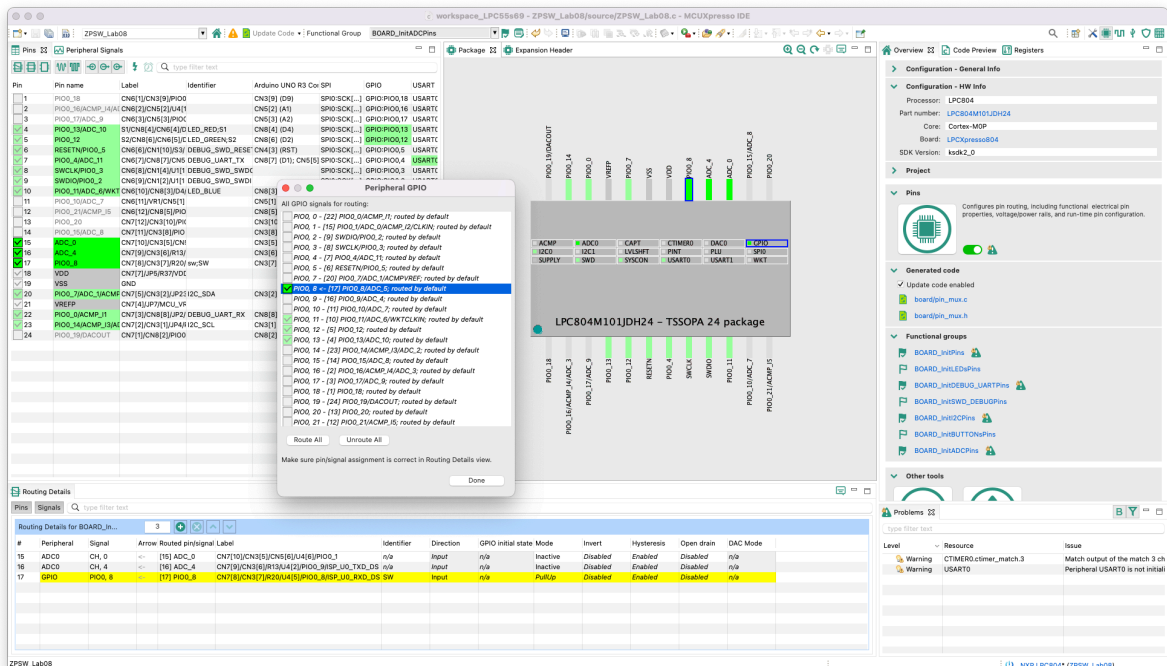
## 8. Analog Joystick

### I. A/D Converter

1. Copy the project from the previous class and name it eg *Lab08*.
2. Go to *Config Tool* -> *Pins* and open the *BOARD\_InitADCPins* preset. Click on the ADC block and to the existing *ADC0* signal (*PIO0\_1* pinout), similarly add the *ADC4* signal (*PIO0\_9* pinout):



3. Add pin *PIO0\_8* as input from *PullUp* and add *SW* identifier:



# Programming of embedded systems

## 8. Analog Joystick

4. Go to the ADC settings and change its configuration by adding an additional channel (CH 4):

**12-bit ADC Controller (ADC)** [Peripheral drivers (Device specific)]

Name: ADC Custom name:

Peripheral: ADC

**General configuration**

**Basic ADC configuration**

Clock mode: System clock - BOARD\_BootClockFRO18M: 15 MHz, BOARD\_BootClockFRO24M: 12 MHz, BOARD\_BootClockFRO30M: 15 MHz

Clock source frequency: 15 MHz (BOARD\_BootClockFRO18M)

Clock divider number: 0

Low power mode:

**Configure threshold settings**

**Threshold values pair 0**

Low value: 0 High value: 0

**Threshold values pair 1**

Low value: 0 High value: 0

**ADC conversion sequence A**

Set high priority for conversion sequence:

Hardware trigger: CTIMERO\_MAT3

Trigger polarity: A positive edge

Synchronization bypassing:

Single step mode:

Interrupt source: Entire sequence

**ADC conversion sequence B**

Set high priority for conversion sequence:

Hardware trigger: Disabled

Trigger polarity: A negative edge

Synchronization bypassing:

Single step mode:

Interrupt source: Each conversion

**Sampled channels** + X

#	Custom name	Channel number	Channel threshold pair	Threshold interrupt mode	Conversion sequence
0		CH, 0 » [15] CN7	Threshold pair 0	Interrupt disabled	Sequence A
1		CH, 4 » [16] CN7	Threshold pair 0	Interrupt disabled	Sequence A

Interrupt sources:  Sequence A interrupt  Sequence B interrupt  Overrun interrupt

**Enable Sequence A interrupt**

Interrupt: ADC\_SEQ\_A\_IRQn

Interrupt request: Enabled in initialization

Enable priority initialization:

Priority: 0

Enable custom handler name:

Interrupt handler name: ADC\_ADC\_SEQ\_A\_IRQHANDLER

Handler template: Copy to clipboard

**Enable Sequence B interrupt**

Interrupt: ADC\_SEQ\_B\_IRQn

Interrupt request: Enabled in initialization

Enable priority initialization:

Priority: 0

Enable custom handler name:

Interrupt handler name: ADC\_ADC\_SEQ\_B\_IRQHANDLER

Handler template: Copy to clipboard

**Enable Threshold compare interrupt**

Interrupt: ADC\_THCMP\_IRQn

Interrupt request: Enabled in initialization

Enable priority initialization:

Priority: 0

Enable custom handler name:

Interrupt handler name: ADC\_ADC\_THCMP\_IRQHANDLER

Handler template: Copy to clipboard

**Enable Overrun error interrupt**

Interrupt: ADC\_OVR\_IRQn

Interrupt request: Enabled in initialization

Enable priority initialization:

Priority: 0

Enable custom handler name:

Interrupt handler name: ADC\_ADC\_OVR\_IRQHANDLER

Handler template: Copy to clipboard

# Programming of embedded systems

## 8. Analog Joystick

5. 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"

static adc_result_info_t gAdcResultInfoStruct;
adc_result_info_t *volatile gAdcResultInfoPtr = &gAdcResultInfoStruct;

char sbuff[32];

volatile uint16_t gAxisX = 0;
volatile uint16_t gAxisY = 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);
        gAxisY = gAdcResultInfoStruct.result;

        ADC_GetChannelConversionResult(ADC_PERIPHERAL, 4, gAdcResultInfoPtr);
        gAxisX = gAdcResultInfoStruct.result;

        /* Clear status flags */
        ADC_ClearStatusFlags(ADC_PERIPHERAL, kADC_ConvSeqAInterruptFlag);
    }
}

/*
 * @brief Application entry point.
 */
int main(void) {
    /* Power on ADC. */
    POWER_DisablePD(kPDRUNCFG_PD_ADC0);
    /* Init board hardware. */
    BOARD_InitBootPins();
    BOARD_InitBootClocks();
    BOARD_InitBootPeripherals();
#ifdef BOARD_INIT_DEBUG_CONSOLE_PERIPHERAL
    /* Init FSL debug console. */
    BOARD_InitDebugConsole();
#endif

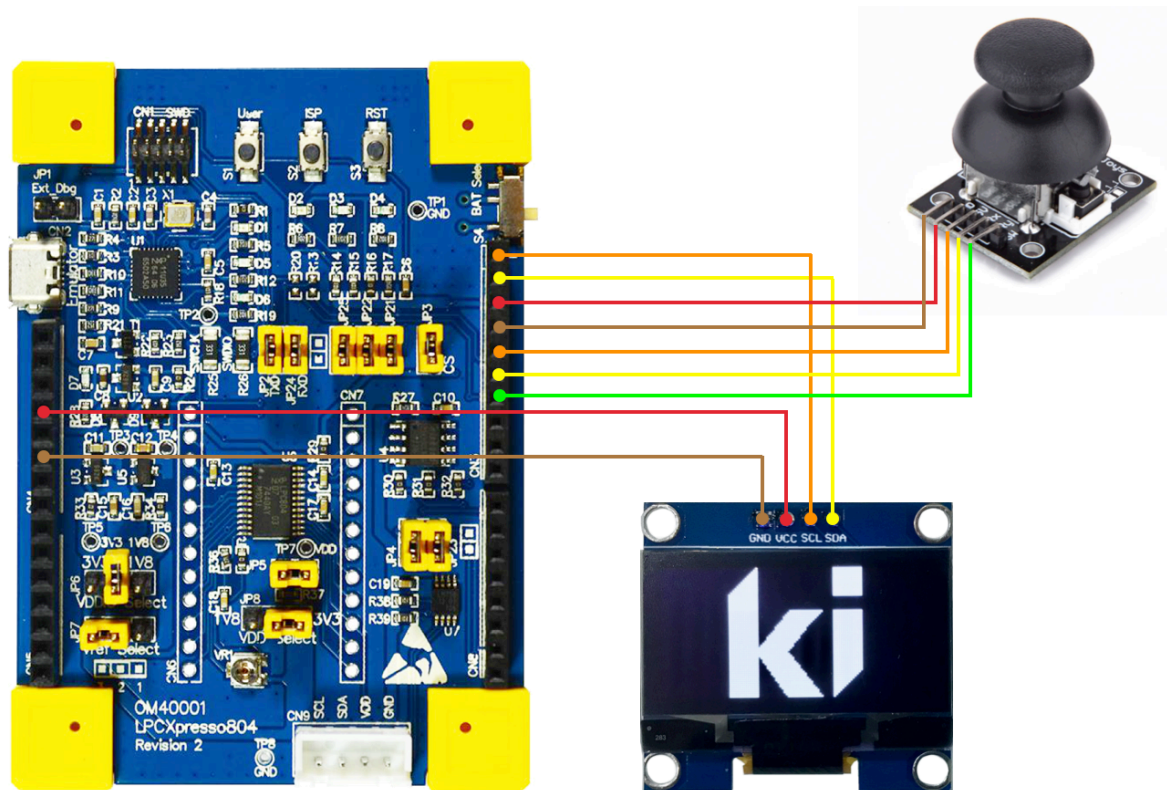
    /* Initialize OLED */
    OLED_Init(I2C0_PERIPHERAL);

    while(1) {
        OLED_Clear_Screen(0);
        sprintf(sbuff, "X: %5d", gAxisX);
        OLED_Puts(0, 0, sbuff);
        sprintf(sbuff, "Y: %5d", gAxisY);
        OLED_Puts(0, 1, sbuff);
        OLED_Refresh_Gram();
    }
    return 0 ;
}
```

# Programming of embedded systems

## 8. Analog Joystick

6. Connect the display and the joystick to the board according to the following diagram:



7. Program the microcontroller and check the example operation.

## II. Button operation

1. Modify your project code by adding Z axis button support:

```
#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"

static adc_result_info_t gAdcResultInfoStruct;
adc_result_info_t *volatile gAdcResultInfoPtr = &gAdcResultInfoStruct;
char sbuff[32];
volatile uint16_t gAxisX = 0;
volatile uint16_t gAxisY = 0;
volatile bool gAxisZ = 0;

/* ADC_SEQ_IRQn interrupt handler */
void ADC_ADC_SEQ_A_IRQHANDLER(void) {
    /* Get status flags */
    if (kADC_ConvSeqInterruptFlag == (kADC_ConvSeqInterruptFlag & ADC_GetStatusFlags(ADC_PERIPHERAL))) {
        /* Place your interrupt code here */
        ADC_GetChannelConversionResult(ADC_PERIPHERAL, 0, gAdcResultInfoPtr);
        gAxisY = gAdcResultInfoStruct.result;

        ADC_GetChannelConversionResult(ADC_PERIPHERAL, 4, gAdcResultInfoPtr);
        gAxisX = gAdcResultInfoStruct.result;

        gAxisZ = GPIO_PinRead(BOARD_INITADCPINS_SW_GPIO,
                              BOARD_INITADCPINS_SW_PORT,
                              BOARD_INITADCPINS_SW_PIN);

        /* Clear status flags */
        ADC_ClearStatusFlags(ADC_PERIPHERAL, kADC_ConvSeqInterruptFlag);
    }
}
```

# Programming of embedded systems

## 8. Analog Joystick

```
    }
}

/*
 * @brief Application entry point.
 */
int main(void) {

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

    /* Initialize OLED */
    OLED_Init(I2C0_PERIPHERAL);

    while(1) {

        OLED_Clear_Screen(0);
        sprintf(sbuff, "X: %5d", gAxisX);
        OLED_Puts(0, 0, sbuff);
        sprintf(sbuff, "Y: %5d", gAxisY);
        OLED_Puts(0, 1, sbuff);
        sprintf(sbuff, "Z: %5d", gAxisZ);
        OLED_Puts(0, 2, sbuff);
        OLED_Refresh_Gram();

    }

    return 0 ;
}
```

2. Build the project in **Release** mode, program the microcontroller and check the example.

### III. Cursor support

1. Modify the project code:

```
#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"

static adc_result_info_t gAdcResultInfoStruct;
adc_result_info_t *volatile gAdcResultInfoPtr = &gAdcResultInfoStruct;
char sbuff[32];
volatile uint16_t gAxisX = 0;
volatile uint16_t gAxisY = 0;
volatile bool gAxisZ = 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);
        gAxisY = gAdcResultInfoStruct.result;

        ADC_GetChannelConversionResult(ADC_PERIPHERAL, 4, gAdcResultInfoPtr);
        gAxisX = gAdcResultInfoStruct.result;

        gAxisZ = GPIO_PinRead(BOARD_INITADCPINS_SW_GPIO,
                              BOARD_INITADCPINS_SW_PORT,
                              BOARD_INITADCPINS_SW_PIN);

        /* Clear status flags */
        ADC_ClearStatusFlags(ADC_PERIPHERAL, kADC_ConvSeqAInterruptFlag);
    }
}

void setCursor(uint8_t x, uint8_t y, uint8_t size) {

    int8_t a, b;

    a=x-size;
    b=x+size;
    if(a<0) {
        a=0;
    }
    OLED_Draw_Line(a, y, b, y);
    a=y-size;
```

# Programming of embedded systems

## 8. Analog Joystick

```
        b=y+size;
        if(a<0) {
            a=0;
        }
        OLED_Draw_Line(x, a, x, b);
    }
}
/*
 * @brief Application entry point.
 */
int main(void) {
    uint8_t cx, cy;

    /* Power on ADC. */
    POWER_DisablePD(kPDRUNCFG_PD_ADC0);
    /* Init board hardware. */
    BOARD_InitBootPins();
    BOARD_InitBootClocks();
    BOARD_InitBootPeripherals();
#ifdef BOARD_INIT_DEBUG_CONSOLE_PERIPHERAL
    /* Init FSL debug console. */
    BOARD_InitDebugConsole();
#endif
    /* Initialize OLED */
    OLED_Init(I2C0_PERIPHERAL);

    while(1) {
        cx = gAxisX/32; // width: 128
        cy = 63-gAxisY/64; // height: 64

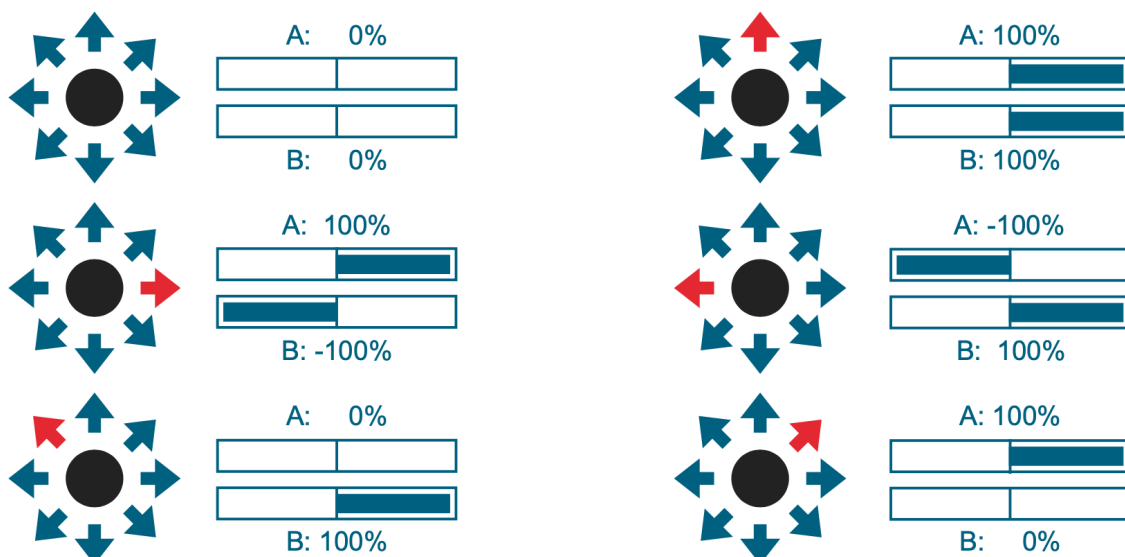
        OLED_Clear_Screen(0);
        sprintf(sbuff, "X:%3d Y:%2d Z:%d", cx, cy, gAxisZ);
        OLED_Puts(0, 0, sbuff);

        setCursor(cx, cy, 5);
        if(!gAxisZ) {
            OLED_Draw_Circle(cx, cy, 8);
        }
        OLED_Refresh_Gram();
    }
    return 0;
}
```

2. Build the project in **Release** mode, program the microcontroller and check the example.

### IV. Exercises

1. Write a *PowerControl* function capable of generating control signals for 2 motors of the tracked vehicle depending on the position of the joystick. The function should present the calculated control in the form of two progress bars or deflection indicators (as in the previous class) and display the power values as a percentage. Examples of joystick settings:





# Programming of embedded systems

## 8. Analog Joystick

In order to display negative values, integer variables with printf functions, sprint etc., you should add the `PRINTF_ADVANCED_ENABLE` constant in the preprocessor settings:

