UNIVERZITET U BEOGRADU • ELEKTROTEHNIČKI FAKULTET •

Projekat iz Sistema u Realnom Vremenu

Redni broj 23

Uroš Stefanović 2020/0019

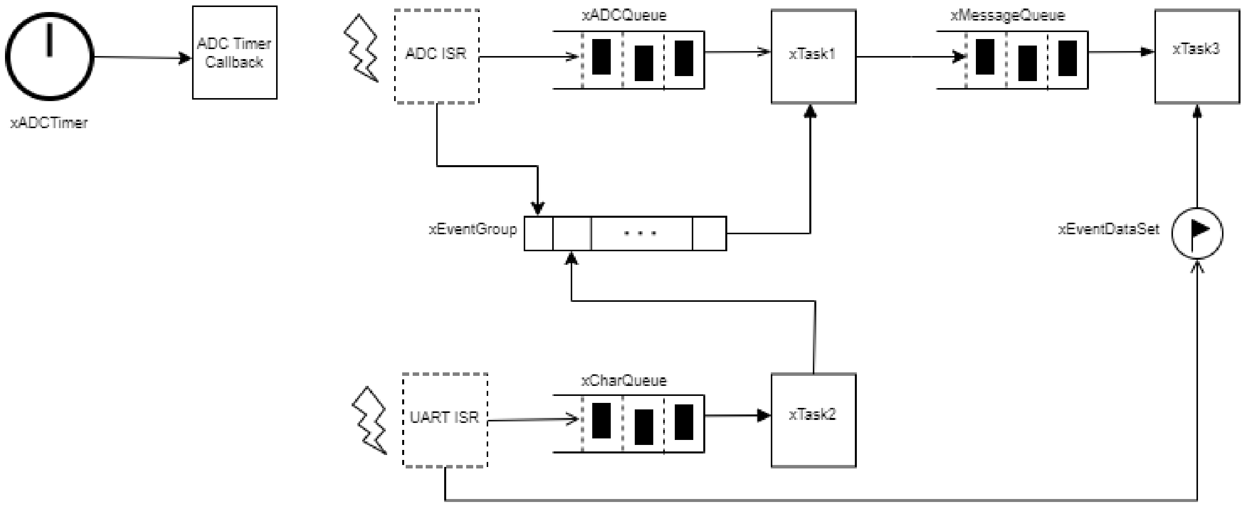
Jun 2024.

Tekst projektnog zadatka (ABCDEF = 010111):

Startuje akvizicija sa kanala A0, A1 na svakih 1000ms pomocu **softverskog tajmera**. Potrebno je implementirati odlozenu obradu prekida (defered interrupt processing) AD konvertora, tako sto se rezultat konverzije u prekidnoj rutini upisuje u red sa porukama (Queue) i obavestava se task xTask1 o prispecu nove poruke putem **grupe dogadaja (EventGroup)**. Poruka treba da sadrzi informaciju o kanalu koji je ocitan i gornjih 9 bita rezultata AD konverzije. Task xTask1 cuva poslednju ocitanu vrednost za svaki kanal.

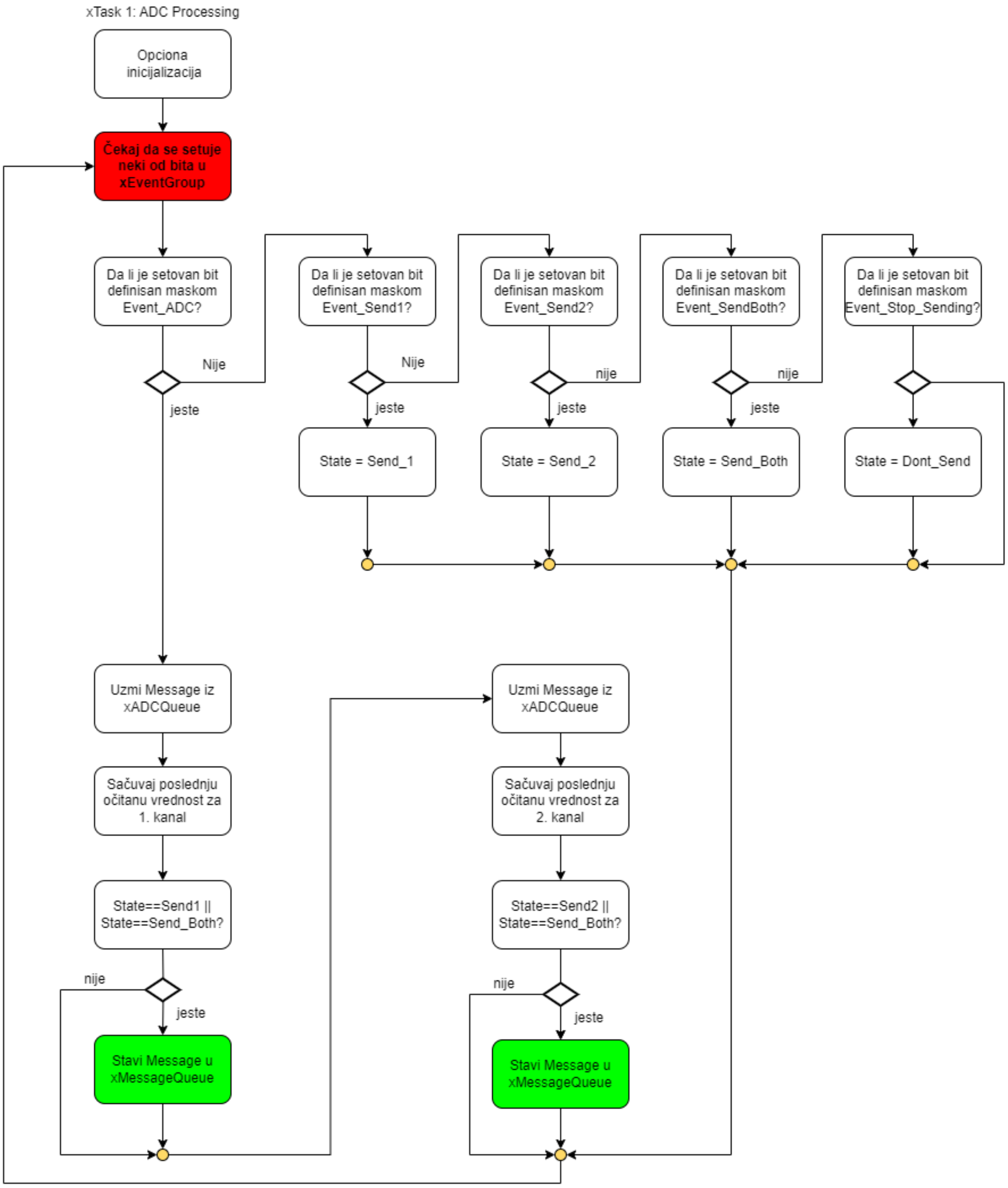
Task xTask2 **implementira odlozenu obradu prekida za UART callback rutinu i na prijem karaktera '1'-'4'** obavestava task xTask1 putem **grupe dogadaja (EventGroup)** o kanalu cije ocitane vrednosti rezultata konverzije treba da salje tasku xTask3. Svaki put kada stigne nova vrednost sa AD konvertora task xTask1 smesta odgovarajuci podatak u red sa porukama na kojem ceka task xTask3. Task xTask3 racuna razliku izmedu uzastopnih vrednosti ocitanog kanala i prikazuje na **UART-u.**

Arhitektura realizovanog softvera:

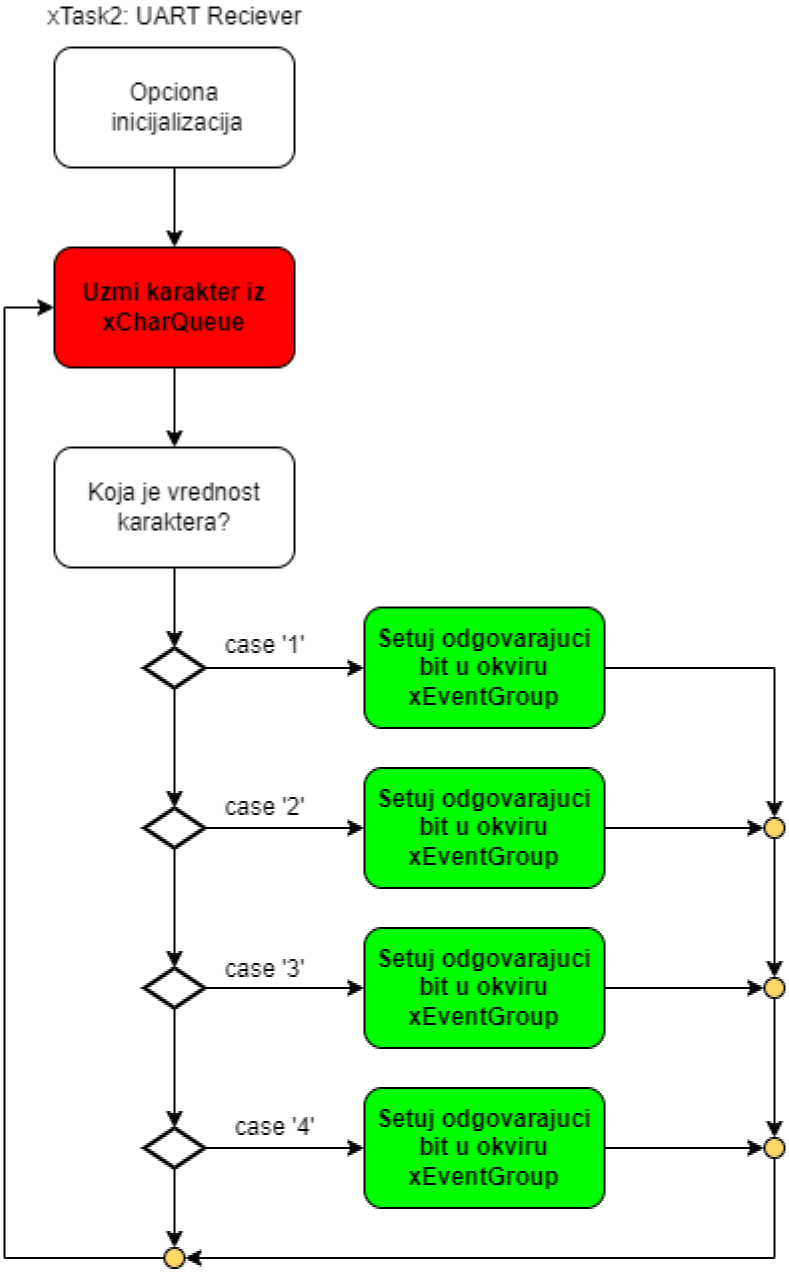


Dijagrami aktivnosti taskova i prekidnih rutina:

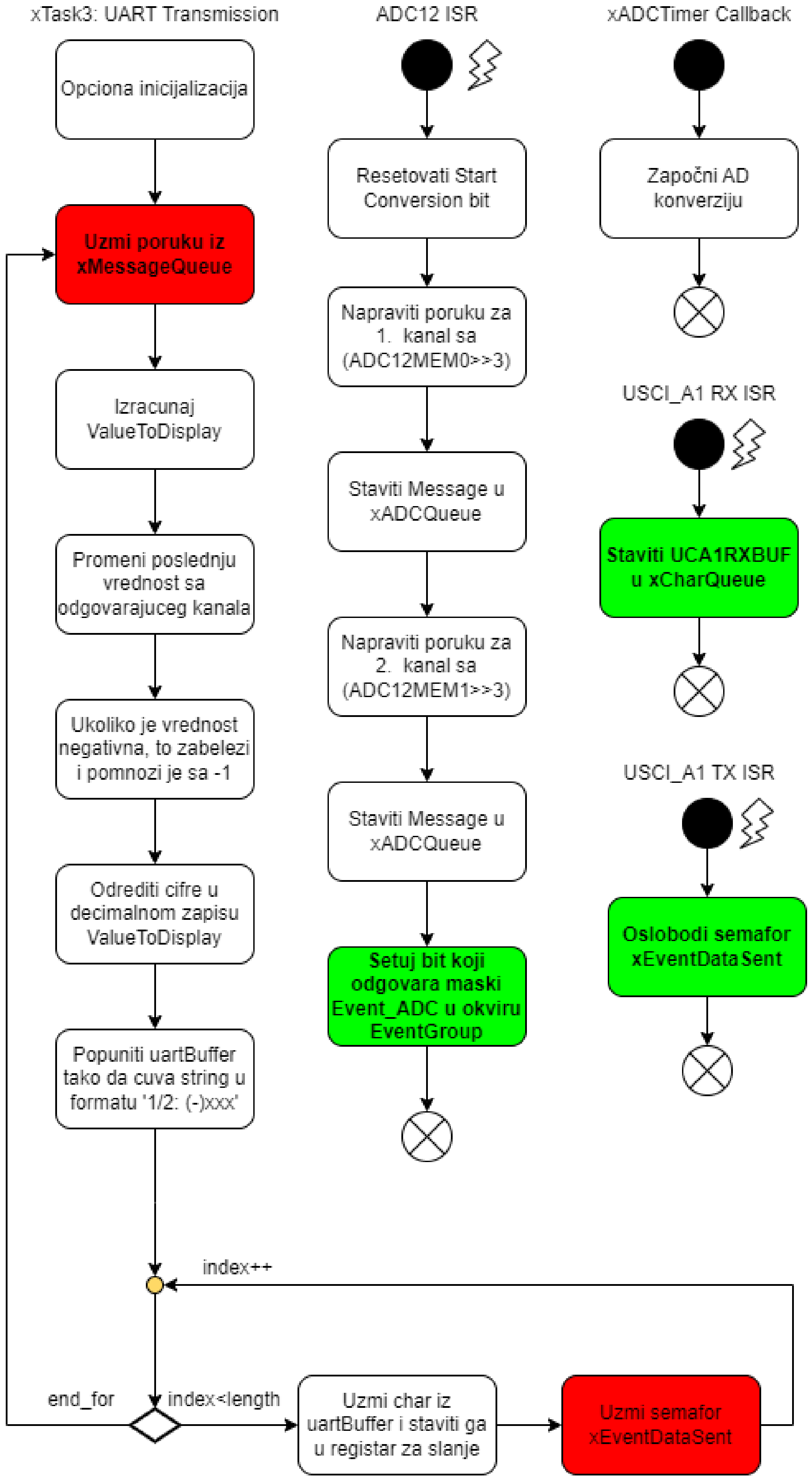
Task1:


xTask 1: ADC Processing
Opciona inicijalizacija
Čekaj da se setuje neki od bita u xEventGroup
Da li je setovan bit definisan maskom Event_ADC?
Nije
jeste
Da li je setovan bit definisan maskom Event_Send1?
Nije
jeste
State = Send_1
Da li je setovan bit definisan maskom Event_Send2?
nije
jeste
State = Send_2
Da li je setovan bit definisan maskom Event_SendBoth?
nije
jeste
State = Send_Both
Da li je setovan bit definisan maskom Event_Stop_Sending?
State = Dont_Send
Uzmi Message iz xADCQueue
Sačuvaj poslednju očitanu vrednost za 1. kanal
State==Send1 || State==Send_Both?
nije
jeste
Stavi Message u xMessageQueue
Uzmi Message iz xADCQueue
Sačuvaj poslednju očitanu vrednost za 2. kanal
State==Send2 || State==Send_Both?
nije
jeste
Stavi Message u xMessageQueue

Task2:


xTask2: UART Reciever
Opciona inicijalizacija
Uzmi karakter iz xCharQueue
Koja je vrednost karaktera?
case '1'
Setuj odgovarajuci bit u okviru xEventGroup
case '2'
Setuj odgovarajuci bit u okviru xEventGroup
case '3'
Setuj odgovarajuci bit u okviru xEventGroup
case '4'
Setuj odgovarajuci bit u okviru xEventGroup

Task3, ADC12 ISR, ADCTimer Callback, USCI_A1 ISR:


xTask3: UART Transmission
Opciona inicijalizacija
Uzmi poruku iz xMessageQueue
Izracunaj ValueToDisplay
Promeni poslednju vrednost sa odgovarajuceg kanala
Ukoliko je vrednost negativna, to zabelezi i pomnozi je sa -1
Odrediti cifre u decimalnom zapisu ValueToDisplay
Popuniti uartBuffer tako da cuva string u formatu '1/2: (-)xxx'
index++
end_for
index<length
Uzmi char iz uartBuffer i staviti ga u registar za slanje
Uzmi semafor xEventDataSent
ADC12 ISR
Resetovati Start Conversion bit
Napraviti poruku za 1. kanal sa (ADC12MEM0>>3)
Staviti Message u xADCQueue
Napraviti poruku za 2. kanal sa (ADC12MEM1>>3)
Staviti Message u xADCQueue
Setuj bit koji odgovara maski Event_ADC u okviru EventGroup
xADCTimer Callback
Započni AD konverziju
USCI_A1 RX ISR
Staviti UCA1RXBUF u xCharQueue
USCI_A1 TX ISR
Oslobodi semafor xEventDataSent

Programski kod:

```c
/********************************************************************************
 * @file main.c
 * @brief Real-Time Embedded Systems Course - ADC Data Display Application
 * @version 1.0
 * @date June, 2024
 *
 * @details
 * This project implements a real-time application that performs Analog-to-Digital
 * (ADC) conversions on two channels every second and transmits the converted
 * values over UART to a PC for display. The user can interact with the system
 * via UART commands to select which ADC channel data to display or to stop
 * the display.
 *
 * @section Functional Overview
 * 1. ADC Sampling:
 *    - Two ADC channels are sampled every second using a software timer.
 *    - The converted ADC values are processed to retain only the upper 9 bits.
 *
 * 2. UART Communication:
 *    - UART is used to receive commands from the user and to transmit ADC values
 *      back to the PC.
 *    - Commands:
 *      - '1': Display values from the first ADC channel.
 *      - '2': Display values from the second ADC channel.
 *      - '3': Display values from both ADC channels.
 *      - '4': Stop displaying values.
 *
 * @section Tasks and Synchronization
 * 1. Task1 (ADC Processing Task):
 *    - Handles ADC conversions and processes the values to retain the upper 9 bits.
 *
 * 2. Task2 (UART Receiving Task):
 *    - Handles UART reception using deferred interrupt processing.
 *    - Queues received commands for processing.
 *
 * 3. Task3 (UART Transmission Task):
 *    - Transmits processed ADC values over UART to the PC.
 *
 * @section Synchronization Mechanisms
 * - Binary Semaphores: Used to synchronize tasks.
 * - Queues: Used to handle UART commands and ADC data.
 * - Event Groups: Used to manage task notifications and events.
 *
 * @section Software Timer
 * - A software timer is configured to trigger ADC conversions every second.
 *
 * @section Implementation Details
 * - The project utilizes FreeRTOS for task management and synchronization.
 * - ADC values are stored in a buffer, and the upper 9 bits are extracted for transmission.
 * - UART communication is implemented with interrupt handling to ensure responsive command processing.
 *
 * @author Uros Stefanovic
 ********************************************************************************/

/* Standard includes. */
#include <stdio.h>
#include <stdlib.h>

/* FreeRTOS includes. */
#include "FreeRTOS.h"
#include "task.h"
#include "semphr.h"
#include "queue.h"
#include "timers.h"
#include "event_groups.h"
#include "semphr.h"

/* Hardware includes. */
#include "msp430.h"

/* User's includes */
#include "../ETF5529_HAL/hal_ETF_5529.h"
```

```
/* Macros for converting between ASCII and binary coding */
#define ASCII2DIGIT(x)      (x - '0')
#define DIGIT2ASCII(x)      (x + '0')
#define ARRAY_LENGTH        9

/** Task priorities */
#define xTASK1_PRIO        ( 1 )
#define xTASK2_PRIO        ( 2 )
#define xTASK3_PRIO        ( 3 )

/* freeRTOS object parameters */
#define QUEUE_LENGTH            10
#define ADC_TIMER_PERIOD        (pdMS_TO_TICKS(1000))

/* Event bit definitions */
#define  mainEVENT_ADC                  0x02    // ADC ISR has sent Task1 a message
#define  mainEVENT_STOP_SENDING         0x04    // Task2 has detected '4' on UART input
#define  mainEVENT_SEND_1               0x08    // Task2 has detected '1' on UART input
#define  mainEVENT_SEND_2               0x10    // Task2 has detected '2' on UART input
#define  mainEVENT_SEND_BOTH            0x20    // Task2 has detected '3' on UART input

/* freeRTOS object handlers */
xQueueHandle        xADCQueue;
xQueueHandle        xCharQueue;
xQueueHandle        xMessageQueue;
EventGroupHandle_t  xEventGroup;
TimerHandle_t       xADCTimer;
xSemaphoreHandle    xEventDataSent;

/**
 * @brief Configure hardware upon boot
 */
static void prvSetupHardware( void )
{
    taskDISABLE_INTERRUPTS();

    /* Disable the watchdog. */
    WDTCTL = WDTPW + WDTHOLD;

    hal430SetSystemClock( configCPU_CLOCK_HZ, configLFXT_CLOCK_HZ );

//    /* Init buttons */
//    P1DIR &= ~0x30;
//    P1REN |= 0x30;
//    P1OUT |= 0x30;

    /* Initialize ADC */
    ADC12CTL0 = ADC12SHT0_2 + ADC12MSC + ADC12ON; // Sampling time, multi-sample conversion, ADC on
    ADC12CTL1 = ADC12SHP + ADC12CONSEQ_1;         // Use sampling timer, single sequence
    ADC12MCTL0 = ADC12INCH_0;                     // A0 ADC input select; Vref=AVcc
    ADC12MCTL1 = ADC12INCH_1 + ADC12EOS;          // A1 ADC input select; Vref=AVcc, end of sequence
    ADC12IE = ADC12IE1;                           // Enable interrupt for ADC12MEM1 (end of sequence)
    ADC12CTL0 |= ADC12ENC;                        // Enable conversions
    P6SEL          |= 0x03;                      // P6.0 and P6.1 ADC option select

    /* Initialize UART */
    P4SEL       |= BIT4+BIT5;                    // P4.4,5 = USCI_AA TXD/RXD
    UCA1CTL1    |= UCSWRST;                      // **Put state machine in reset**
    UCA1CTL1    |= UCSSEL_2;                     // SMCLK
    UCA1BRW      = 1041;                         // 1MHz - Baudrate 9600
    UCA1MCTL    |= UCBRS_6 + UCBRF_0;            // Modulation UCBRSx=1, UCBRFx=0
    UCA1CTL1    &= ~UCSWRST;                     // **Initialize USCI state machine**
    UCA1IE      |= UCRXIE;                       // Enable USCI_A1 RX interrupt
    UCA1IE      |= UCTXIE;                       // Enable USCI_A1 TX interrupt

    /* initialize LEDs */
    vHALInitLED();
}


/**
 * @brief Software timer Callback Function
 *
 *  The timer will run for 1000ms, after which it will start the ADC and sample channel A0 and A1
 */
void    prvADCTimerCallback(TimerHandle_t xTimer){
    // Trigger ADC Conversion
    ADC12CTL0 |= ADC12SC;
}
```

```c
/**
 * @brief Message struct used for communication between tasks
 *
 * The message contains:
 * a value after ADC conversion,
 * the channel from which the value is sampled
 */
struct Message{
    uint8_t channel;
    uint16_t value;
};


/**
 * @brief UART state enum
 *
 * The state is kept in Task1 and used for keeping track of which channel to display over UART
 */
typedef enum{
    SEND_1,
    SEND_2,
    SEND_BOTH,
    DONT_SEND
}state_t;


/**
 * @brief xTask1: ADC Processing Task
 *
 * This task does deffered interrupt processing for ADC.
 * It receives a message from ADC ISR and determines which channels
 * should be passed to Task3 (sent over UART).
 */
static void prvxTask1( void *pvParameters )
{
    EventBits_t eventValue;

    volatile state_t state = DONT_SEND;

    volatile struct Message xMessage;

    volatile uint16_t xFirstChannelLastValue = 0;
    volatile uint16_t xSecondChannelLastValue = 0;

    while(1){

        /* Wait for ADC event or a char sent from UART */
        eventValue = xEventGroupWaitBits(xEventGroup,
                    mainEVENT_ADC | mainEVENT_SEND_1 | mainEVENT_SEND_2 | mainEVENT_SEND_BOTH | mainEVENT_STOP_SENDING,
                    pdTRUE,
                    pdFALSE,
                    portMAX_DELAY);

        /*Check what caused the exit from the blocked state*/
        if(eventValue & mainEVENT_ADC){
            xQueueReceive(xADCQueue, &xMessage, 0); // Non-blocking call

            // Change last received value
            xFirstChannelLastValue = xMessage.value;

            // If in proper state, send the value to task 3
            if(state==SEND_1 || state==SEND_BOTH){
                xQueueSendToBack(xMessageQueue, &xMessage, portMAX_DELAY);
            }

            xQueueReceive(xADCQueue, &xMessage, 0); // Non-blocking call

            // Change last received value
            xSecondChannelLastValue = xMessage.value;

            // If in proper state, send the value to task 3
            if(state==SEND_2 || state==SEND_BOTH){
                xQueueSendToBack(xMessageQueue, &xMessage, portMAX_DELAY);
            }

        }
        if(eventValue & mainEVENT_SEND_1){
            state = SEND_1;
        }
        if(eventValue & mainEVENT_SEND_2){
            state = SEND_2;
        }
        if(eventValue & mainEVENT_SEND_BOTH){
            state = SEND_BOTH;
        }
        if(eventValue & mainEVENT_STOP_SENDING){
            state = DONT_SEND;
        }
    }
}



/**
 * @brief xTask2: UART Receiver Task
 *
 *  This task does deffered interrupt processing for UART.
 *  When the user send over UART a character between '1' and '4',
 *  this task signals Task1 accordingly.
 */
static void prvxTask2( void *pvParameters ){

    volatile char        recChar =   0;

    while(1){
        /*Read char from the queue*/
        xQueueReceive(xCharQueue, &recChar, portMAX_DELAY); // blocking call
        switch(recChar){
        case '1':
            xEventGroupSetBits(xEventGroup, mainEVENT_SEND_1);
            break;
        case '2':
            xEventGroupSetBits(xEventGroup, mainEVENT_SEND_2);
            break;
        case '3':
            xEventGroupSetBits(xEventGroup, mainEVENT_SEND_BOTH);
            break;
        case '4':
            xEventGroupSetBits(xEventGroup, mainEVENT_STOP_SENDING);
            break;
        }
    }
}

/**
 * @brief xTask3: UART Transmission Task
 *
 *  This task sends data over UART.
 */
static void prvxTask3( void *pvParameters ){
    volatile struct Message xMessage;

    // Values from last message, used for finding difference
    volatile uint16_t xFirstChannelLastValue = 0;
    volatile uint16_t xSecondChannelLastValue = 0;

    // Variables used for formating the value for sending
    volatile int xValueToDisplay;
    volatile uint8_t xHundredDigit;
    volatile uint8_t xTenDigit;
    volatile uint8_t xOneDigit;
    volatile bool negative;

    // An array of chars used for UART transmission
    volatile char uartBuffer[ARRAY_LENGTH];
    volatile int index = 0;
    volatile int bufferLength;

    while(1){
        xQueueReceive(xMessageQueue, &xMessage, portMAX_DELAY); // blocking call

        if(xMessage.channel==1){
            xValueToDisplay = xMessage.value - xFirstChannelLastValue;
            xFirstChannelLastValue = xMessage.value;
        }
        if(xMessage.channel==2){
            xValueToDisplay = xMessage.value - xSecondChannelLastValue;
```

```c
            xSecondChannelLastValue = xMessage.value;
        }
        // In case of negative differences
        if(xValueToDisplay>=0){
            negative = false;
        }
        else{
            negative = true;
            xValueToDisplay *= -1;
        }

        // Finding the digits in decimal format
        xHundredDigit = xValueToDisplay/100;
        xValueToDisplay -= xHundredDigit*100;
        xTenDigit = xValueToDisplay/10;
        xOneDigit = xValueToDisplay - xTenDigit*10;

        // Format: '1/2: (-)xxx'
        index = 0;
        uartBuffer[index++] = DIGIT2ASCII(xMessage.channel);
        uartBuffer[index++] = ':';
        uartBuffer[index++] = ' ';
        if(negative){
            uartBuffer[index++] = '-';
        }
        uartBuffer[index++] = DIGIT2ASCII(xHundredDigit);
        uartBuffer[index++] = DIGIT2ASCII(xTenDigit);
        uartBuffer[index++] = DIGIT2ASCII(xOneDigit);
        uartBuffer[index++] = '\n';
        uartBuffer[index++] = '\r';
        bufferLength = index;

        // Sending data
        for(index = 0; index < bufferLength; index++){
            UCA1TXBUF = uartBuffer[index];
            xSemaphoreTake(xEventDataSent, portMAX_DELAY); // blocking call
        }
    }
}



/**
 * @brief main function
 */
void main( void )
{
    /* Configure peripherals */
    prvSetupHardware();

    /* Create tasks */
    xTaskCreate( prvxTask1,                         // task function
                 "ADC Processing Task",             // task name
                 configMINIMAL_STACK_SIZE,          // stack size
                 NULL,                              // no parameter is passed
                 xTASK1_PRIO,                       // priority
                 NULL                               // we don't need handle
               );
    xTaskCreate( prvxTask2,                         // task function
                 "UART Receiver Task",              // task name
                 configMINIMAL_STACK_SIZE,          // stack size
                 NULL,                              // no parameter is passed
                 xTASK2_PRIO,                       // priority
                 NULL                               // we don't need handle
               );
    xTaskCreate( prvxTask3,                         // task function
                 "UART Transmission Task",          // task name
                 configMINIMAL_STACK_SIZE,          // stack size
                 NULL,                              // no parameter is passed
                 xTASK3_PRIO,                       // priority
                 NULL                               // we don't need handle
               );

    /* Create timer */
    xADCTimer = xTimerCreate("ADC timer",
                 ADC_TIMER_PERIOD,
                 pdTRUE,
                 NULL,
                 prvADCTimerCallback);


    // Create other freeRTOS objects
    xEventGroup         = xEventGroupCreate();

    xEventDataSent      =   xSemaphoreCreateBinary();

    xADCQueue             =   xQueueCreate(QUEUE_LENGTH,sizeof(struct Message));
    xCharQueue             =   xQueueCreate(QUEUE_LENGTH,sizeof(char));
    xMessageQueue             =   xQueueCreate(QUEUE_LENGTH,sizeof(struct Message));

    // Start timer
    xTimerStart(xADCTimer, portMAX_DELAY);

    /* Start the scheduler. */
    vTaskStartScheduler();


    /* If all is well then this line will never be reached.  If it is reached
    then it is likely that there was insufficient (FreeRTOS) heap memory space
    to create the idle task.  This may have been trapped by the malloc() failed
    hook function, if one is configured. */
    for( ;; );
}






/**
 * @brief ADC12 ISR
 *
 * When the interrupt happens on the ADC12IFG1,
 * the values from the ADC12MEM0 and ADC12MEM1 registers are formatted
 * into a message structure, after which they are sent to the ADCQueue.
 */
void __attribute__ ( ( interrupt( ADC12_VECTOR  ) ) ) vADC12ISR( void )
{
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;
    uint16_t    temp =0;
    struct Message message;
    switch(__even_in_range(ADC12IV,34))
    {
        case  0: break;                           // Vector  0:  No interrupt
        case  2: break;                           // Vector  2:  ADC overflow
        case  4: break;                           // Vector  4:  ADC timing overflow
        case  6:                                  // Vector  6:  ADC12IFG0
            break;
        case  8:                                  // Vector  8:  ADC12IFG1
            // Reset 'Start Conversion' bit
            ADC12CTL0 &= ~(ADC12SC);

            // Put into a message object
            message.channel = 1;
            message.value = ADC12MEM0 >> 3;
            // Send to Task1
            xQueueSendToBackFromISR(xADCQueue, &message, &xHigherPriorityTaskWoken);

            // Put into a message object
            message.channel = 2;
            message.value = ADC12MEM1 >> 3;
            // Send to Task1
            xQueueSendToBackFromISR(xADCQueue, &message, &xHigherPriorityTaskWoken);

            // Signal xTask1 the ISR has finished
            xEventGroupSetBitsFromISR(xEventGroup, mainEVENT_ADC, &xHigherPriorityTaskWoken);
            break;
        case 10: break;                           // Vector 10:  ADC12IFG2
        case 12: break;                           // Vector 12:  ADC12IFG3
        case 14: break;                           // Vector 14:  ADC12IFG4
        case 16: break;                           // Vector 16:  ADC12IFG5
        case 18: break;                           // Vector 18:  ADC12IFG6
        case 20: break;                           // Vector 20:  ADC12IFG7
        case 22: break;                           // Vector 22:  ADC12IFG8
        case 24: break;                           // Vector 24:  ADC12IFG9
        case 26: break;                           // Vector 26:  ADC12IFG10
        case 28: break;                           // Vector 28:  ADC12IFG11
        case 30: break;                           // Vector 30:  ADC12IFG12
        case 32: break;                           // Vector 32:  ADC12IFG13
        case 34: break;                           // Vector 34:  ADC12IFG14
        default: break;
    }
    /* trigger scheduler if higher priority task is woken */
    portYIELD_FROM_ISR( xHigherPriorityTaskWoken );
}

/**
 * @brief USCI_A1 ISR
 *
 * If a character is received, it is passed to xTask2.
 * When a character was sent, xTask3 is notified.
 */
void __attribute__ ( ( interrupt( USCI_A1_VECTOR  ) ) ) vUARTISR( void )
{
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;

    switch(UCA1IV)
    {
        case 0:break;                             // Vector 0 - no interrupt
        case 2:                                   // Vector 2 - RXIFG
            xQueueSendToBackFromISR(xCharQueue, &UCA1RXBUF, &xHigherPriorityTaskWoken);
        break;
        case 4:                                   // Vector 4 - TXIFG
            xSemaphoreGive(xEventDataSent);
            break;
        default: break;
    }
    /* trigger scheduler if higher priority task is woken */
    portYIELD_FROM_ISR( xHigherPriorityTaskWoken );

}
```