```
/*
 *  Nickolas Schiffer
 *  CMPE 146 S19
 *  Lab: UART: part0
 */

/**
 * @file
 * @brief This is the application entry point.
 */

#include <stdio.h>
#include <LPC17xx.h>
#include "utilities.h"
#include <tasks.hpp>
#include <switches.hpp>
#include <LED_Display.hpp>
#include <UART/UART_0_1_2_3.hpp>


#define LOW  false
#define HIGH true

enum operation {
    add,
    sub,
    mult
};
uint8_t dig1 = 0;
uint8_t dig2 = 0;
operation op = add;


SemaphoreHandle_t rx_sem = NULL;

void vALU(void *pvParameters){
    auto led = LED_Display::getInstance();
    auto uart3 = LabUart();
    uart3.Initialize(LabUart::U3, LabUart::b38400, true, LabUart::f8_bit);
    uint8_t d1, d2, result = 0;
    operation op_recv;
    while(1){
        d1 = (uint8_t)uart3.Receive();
        d2 = (uint8_t)uart3.Receive();
        op_recv = (operation)uart3.Receive();
        printf("ALU recv d1: %u, d2: %u, op: %u\n", d1, d2, op_recv);
        switch(op_recv){
            case add:
                result = d1 + d2;
                break;
            case sub:
                if (d1 < d2){
                    printf("no negative results please\n");
                    result = 0;
                    break;
                }
                result = d1 - d2;
                break;
            case mult:
                result = d1 * d2;
                break;
        }
        led.setNumber(result);
        uart3.Transmit((char)(result / 10));
        uart3.Transmit((char)(result % 10));
        printf("ALU result: %u\n", result);

    }
}

void vControl_Unit(void *pvParameters){
    auto led = LED_Display::getInstance();

    LabUart uart2 = LabUart();
    uart2.Initialize(LabUart::U2, LabUart::b38400, true, LabUart::f8_bit);
    uint8_t result1, result2;
    while(1){
        if (xSemaphoreTake(rx_sem, portMAX_DELAY)){
            printf("started\n");
            uart2.Transmit((char)dig1);
            uart2.Transmit((char)dig2);
            uart2.Transmit((char)op);

            result1 = (uint8_t)uart2.Receive();
            result2 = (uint8_t)uart2.Receive();
            printf("received: %d, %d\n", result1, result2);
            led.setNumber((result1 * 10) + result2);
            printf("CU Result %d\n",(result1 * 10) + result2);
            vTaskDelay(500);
        }

    }
}

```

```
void vInterfaceControl(void *pvParamters){
    auto sw = Switches::getInstance();
    auto led = LED_Display::getInstance();
    led.init();
    led.clear();
    sw.init();
    bool sw_prev_states[4] = {HIGH};
    bool sw_current_states[4] = {HIGH};

    while (1){
        for (int i = 0; i < 4; i++){
            sw_prev_states[i] = sw_current_states[i];
            sw_current_states[i] = sw.getSwitch(i + 1);
            if ((sw_current_states[i] == HIGH) && (sw_prev_states[i] == LOW)){
                switch(i){
                    case 0:
                        dig1 = (dig1 + 1) % 10;
                        printf("dig1: %d\n",dig1);
                        led.setNumber((dig1 * 10) + dig2);
                        break;
                    case 1:
                        dig2 = (dig2 + 1) % 10;
                        printf("dig2: %d\n",dig2);
                        led.setNumber((dig1 * 10) + dig2);
                        break;
                    case 2:
                        switch(op){
                            case add:
                                op = sub;
                                break;
                            case sub:
                                op = mult;
                                break;
                            case mult:
                                op = add;
                                break;
                        }
                        printf("op: %d\n",op);
                        break;
                    case 3:
                        xSemaphoreGive(rx_sem);
                        vTaskDelay(500);
                        break;

                }
            }
        }
    }
}

int main(){

    rx_sem = xSemaphoreCreateBinary();

    scheduler_add_task(new terminalTask(PRIORITY_HIGH));
    xTaskCreate(vInterfaceControl,"InterfaceControl",1000, NULL,PRIORITY_LOW, NULL);
    xTaskCreate(vControl_Unit,"CU",1000, NULL,PRIORITY_LOW, NULL);
    xTaskCreate(vALU,"ALU",1000, NULL,PRIORITY_LOW, NULL);

    scheduler_start();
    return -1;
}

```

```cpp
/*
 * UART_0_1_2_3.hpp
 *
 *  Created on: Mar 9, 2019
 *      Author: nickschiffer (nick@schiffer.us)
 */

#ifndef UART_0_1_2_3_HPP_
#define UART_0_1_2_3_HPP_

#include <LPC17xx.h>
#include <char_dev.hpp>

#define QUEUE_SIZE 100
#define ENQUEUE_TIMEOUT 1
#define DEQUEUE_TIMEOUT portMAX_DELAY


class LabUart// : public CharDev
{
 public:
    enum UART_Device {
        U0,
        U1,
        U2,
        U3
    };

    enum BAUD_Rate {
        b600    = 600,
        b1200   = 1200,
        b2400   = 2400,
        b4800   = 4800,
        b9600   = 9600,
        b14400  = 14400,
        b19200  = 19200,
        b38400  = 38400,
        b56000  = 56000,
        b57600  = 57600,
        b115200 = 115200
    };

    enum Frame_Size {
        f5_bit,
        f6_bit,
        f7_bit,
        f8_bit
    };

    enum Stop_Bit {
        s1_bit,
        s2_bit
    };

    enum Parity_Mode {
        pNone,
        pOdd,
        pEven,
        pForced_1,
        pForced_2
    };

    enum UART_Init_Result{
        Success,
        Invalid_Parity_Config,
        No_Device_Initialized,
        Unspecified_Error
    };

    LabUart(){};
    ~LabUart();

    /*
     * Initializes UARTn interface.
     * @param device is the UART_Device enumeration of the desired device.
     * @param baud_rate is the BAUD_Rate enumeration of the desired BAUD rate.
     * @param rx_interrupt determines whether an interrupt will occur upon RX.
     * @param frame_size is the Frame_Size enumeration of the desired frame size.
     * @param break_control determines whether or not break control will occur.
     * @param parity_enable determines whether parity bits will be generated.
     * @param parity_select is the Parity_Select enumeration for desired parity behavior.
     * @return returns UART_Init_Result::Success if Initialization is successful,
     *     other UART_Init_Result status if Initialization has failed
     */
    int Initialize(UART_Device device = U2, BAUD_Rate baud_rate = b38400,
            bool rx_interrupt_enable = true, Frame_Size frame_size = f8_bit, Stop_Bit stop_bit = s1_bit,
            bool break_control = false, bool parity_enable = false, Parity_Mode parity_mode = pNone);
    /*
```

```
     * Transmits character on initialized device
     * @param c char to be transmitted
     * @return returns 1 if successful, 0 if failed
     */
    int Transmit(char c);

    /*
     * Returns next char in RX FIFO
     * If RX interrupt is not enabled, this will block until a character is available
     * @return oldest char in RX FIFO
     */
    char Receive();






 private:
    UART_Device this_device = U2;
    bool initialized = false;
    bool intr_enabled = false;

    static void uart0_rx_intr();
    static void uart1_rx_intr();
    static void uart2_rx_intr();
    static void uart3_rx_intr();

    static QueueHandle_t u0_rx_queue;
    static QueueHandle_t u1_rx_queue;
    static QueueHandle_t u2_rx_queue;
    static QueueHandle_t u3_rx_queue;


};



#endif /* UART_2_3_HPP_ */

```

```cpp
/*
 * UART_0_1_2_3.cpp
 *
 *  Created on: Mar 9, 2019
 *      Author: nickschiffer (nick@schiffer.us)
 */

#include <UART/UART_0_1_2_3.hpp>

QueueHandle_t LabUart::u0_rx_queue;
QueueHandle_t LabUart::u1_rx_queue;
QueueHandle_t LabUart::u2_rx_queue;
QueueHandle_t LabUart::u3_rx_queue;

LabUart::~LabUart()
{
    //TODO disable and cleanup device
}

int LabUart::Initialize(UART_Device device, BAUD_Rate baud_rate,
        bool rx_interrupt_enable, Frame_Size frame_size, Stop_Bit stop_bit, bool break_control,
        bool parity_enable, Parity_Mode parity_mode)
{

    switch (device){
        case U0:
        case U1:
        case U2:
        case U3:{
            /*
             * Power on UART Device
             */
            switch(device){
                case U0:
                    LPC_SC->PCONP |= (1 << pconp_uart0);
                    break;
                case U1:
                    LPC_SC->PCONP |= (1 << pconp_uart1);
                    break;
                case U2:
                    LPC_SC->PCONP |= (1 << pconp_uart2);
                    break;
                case U3:
                    LPC_SC->PCONP |= (1 << pconp_uart3);
                    break;
            }

            /*
             * Select Peripheral Clock Divider
             * Select 01 for CCLK/1
             */
                switch(device){
                    case U0:
                        LPC_SC->PCLKSEL0 &= ~(1 << 7);
                        LPC_SC->PCLKSEL0 |=  (1 << 6);
                        break;
                    case U1:
                        LPC_SC->PCLKSEL0 &= ~(1 << 9);
                        LPC_SC->PCLKSEL0 |=  (1 << 8);
                        break;
                    case U2:
                        LPC_SC->PCLKSEL1 &= ~(1 << 17);
                        LPC_SC->PCLKSEL1 |=  (1 << 16);
                        break;
                    case U3:
                        LPC_SC->PCLKSEL1 &= ~(1 << 19);
                        LPC_SC->PCLKSEL1 |=  (1 << 18);
                        break;
                }

            /*
             * Select TX and RX pins
             */
                switch(device){
                    case U0:
                        LPC_PINCON->PINSEL0 |=  (1 << 4);
                        LPC_PINCON->PINSEL0 &= ~(1 << 5);
                        LPC_PINCON->PINSEL0 |=  (1 << 6);
                        LPC_PINCON->PINSEL0 &= ~(1 << 7);
                        break;
                    case U1:
                        LPC_PINCON->PINSEL4 |=  (1 << 1);
                        LPC_PINCON->PINSEL4 &= ~(1 << 0);
                        LPC_PINCON->PINSEL4 |=  (1 << 3);
                        LPC_PINCON->PINSEL4 &= ~(1 << 2);
                        break;
                    case U2:
                        LPC_PINCON->PINSEL4 |=  (1 << 17);
                        LPC_PINCON->PINSEL4 &= ~(1 << 16);
                        LPC_PINCON->PINSEL4 |=  (1 << 19);
                        LPC_PINCON->PINSEL4 &= ~(1 << 18);
                        break;
                    case U3:
                        LPC_PINCON->PINSEL9 |=  (0b11 << 24);
                        LPC_PINCON->PINSEL9 |=  (0b11 << 26);
                        break;
                }
            /*
             * Enable pullup on TX pin (not recommended on RX pin)
             */
                switch(device){
                    case U0:
                        LPC_PINCON->PINMODE0 &= ~(0b11 << 4);
                        break;
                    case U1:
                        LPC_PINCON->PINMODE4 &= ~(0b11 << 0);
                        break;
                    case U2:
                        LPC_PINCON->PINMODE4 &= ~(0b11 << 16);
                        break;
                    case U3:
                        LPC_PINCON->PINMODE9 &= ~(0b11 << 24);
                        break;
                }
            /*
             * Select requested frame size
             */
                switch (frame_size){
                    case f5_bit:
                        /*
                         * Set frame size to 5 (00)
                         */
                        switch(device){
                            case U0:
                                LPC_UART0->LCR &= ~(0b11 << 0);
                                break;
                            case U1:
                                LPC_UART1->LCR &= ~(0b11 << 0);
                                break;
                            case U2:
                                LPC_UART2->LCR &= ~(0b11 << 0);
                                break;
                            case U3:
                                LPC_UART3->LCR &= ~(0b11 << 0);
                                break;
                        }
                        break;
                    case f6_bit:
                        /*
                         * Set frame size to 6 (01)
                         */
                        switch(device){
                            case U0:
                                LPC_UART0->LCR &= ~(1 << 1);
                                LPC_UART0->LCR |=  (1 << 0);
                                break;
                            case U1:
                                LPC_UART1->LCR &= ~(1 << 1);
                                LPC_UART1->LCR |=  (1 << 0);
                                break;
                            case U2:
                                LPC_UART2->LCR &= ~(1 << 1);
                                LPC_UART2->LCR |=  (1 << 0);
                                break;
                            case U3:
                                LPC_UART3->LCR &= ~(1 << 1);
                                LPC_UART3->LCR |=  (1 << 0);
                                break;
                        }
                        break;
                    case f7_bit:
                        /*
                         * Set frame size to 7 (10)
                         */
                        switch(device){
                            case U0:
                                LPC_UART0->LCR &= ~(1 << 0);
                                LPC_UART0->LCR |=  (1 << 1);
                                break;
                            case U1:
                                LPC_UART1->LCR &= ~(1 << 0);
                                LPC_UART1->LCR |=  (1 << 1);
                                break;
                            case U2:
                                LPC_UART2->LCR &= ~(1 << 0);
                                LPC_UART2->LCR |=  (1 << 1);
                                break;
                            case U3:
                                LPC_UART3->LCR &= ~(1 << 0);
                                LPC_UART3->LCR |=  (1 << 1);
                                break;
                        }
                        break;
                    case f8_bit:
                        /*
                         * Set frame size to 7 (11)
                         */
```

```
            switch(device){
                case U0:
                    LPC_UART0->LCR |= (0b11 << 0);
                    break;
                case U1:
                    LPC_UART1->LCR |= (0b11 << 0);
                    break;
                case U2:
                    LPC_UART2->LCR |= (0b11 << 0);
                    break;
                case U3:
                    LPC_UART3->LCR |= (0b11 << 0);
                    break;
            }
            break;
        default:
            return Unspecified_Error;
    }
/*
 * Select requested number of stop bits
 */
    switch (stop_bit){
        case s1_bit:
            /*
             * 1 stop bit
             */
            switch(device){
                case U0:
                    LPC_UART0->LCR &= ~(1 << 2);
                    break;
                case U1:
                    LPC_UART1->LCR &= ~(1 << 2);
                    break;
                case U2:
                    LPC_UART2->LCR &= ~(1 << 2);
                    break;
                case U3:
                    LPC_UART3->LCR &= ~(1 << 2);
                    break;
            }
            break;
        case s2_bit:
            /*
             * 2 stop bits
             */
            switch(device){
                case U0:
```

```
                                LPC_UART0->LCR |=  (1 << 2);
                                break;
                            case U1:
                                LPC_UART1->LCR |=  (1 << 2);
                                break;
                            case U2:
                                LPC_UART2->LCR |=  (1 << 2);
                                break;
                            case U3:
                                LPC_UART3->LCR |=  (1 << 2);
                                break;
                        }
                        break;
                    default:
                        return Unspecified_Error;
                }
            /*
             * Enable Parity and parity mode if requested
             */
                if (parity_enable){
                    switch(device){
                        case U0:
                            LPC_UART0->LCR |= (1 << 3);
                            break;
                        case U1:
                            LPC_UART1->LCR |= (1 << 3);
                            break;
                        case U2:
                            LPC_UART2->LCR |= (1 << 3);
                            break;
                        case U3:
                            LPC_UART3->LCR |= (1 << 3);
                            break;
                    }
                    switch (parity_mode){
                        case pOdd:
                            /*
                             * Odd parity. Number of 1s in the transmitted
                             * character and the attached parity bit will be odd.
                             */
                            switch(device){
                                case U0:
                                    LPC_UART0->LCR &= ~(0b11 << 4);
                                    break;
                                case U1:
                                    LPC_UART1->LCR &= ~(0b11 << 4);
                                    break;
                                case U2:
                                    LPC_UART2->LCR &= ~(0b11 << 4);
                                    break;
                                case U3:
                                    LPC_UART3->LCR &= ~(0b11 << 4);
                                    break;
                            }
                            break;
                        case pEven:
                            /*
                             * Even Parity. Number of 1s in the transmitted
                             * character and the attached parity bit will be even.
                             */
                            switch(device){
                                case U0:
                                    LPC_UART0->LCR &= ~(0b11 << 4);
                                    LPC_UART0->LCR |=  (1 << 4);
                                    break;
                                case U1:
                                    LPC_UART1->LCR &= ~(0b11 << 4);
                                    LPC_UART1->LCR |=  (1 << 4);
                                    break;
                                case U2:
                                    LPC_UART2->LCR &= ~(0b11 << 4);
                                    LPC_UART2->LCR |=  (1 << 4);
                                    break;
                                case U3:
                                    LPC_UART3->LCR &= ~(0b11 << 4);
                                    LPC_UART3->LCR |=  (1 << 4);
                                    break;
                            }
                            break;
                        case pForced_1:
                            /*
                             * Forced "1" stick parity.
                             */
                            switch(device){
                                case U0:
                                    LPC_UART0->LCR &= ~(0b11 << 4);
                                    LPC_UART0->LCR |=  (1 << 5);
                                    break;
                                case U1:
                                    LPC_UART1->LCR &= ~(0b11 << 4);
                                    LPC_UART1->LCR |=  (1 << 5);
                                    break;
```

```
                                case U2:
                                    LPC_UART2->LCR &= ~(0b11 << 4);
                                    LPC_UART2->LCR |=  (1 << 5);
                                    break;
                                case U3:
                                    LPC_UART3->LCR &= ~(0b11 << 4);
                                    LPC_UART3->LCR |=  (1 << 5);
                                    break;
                            }
                            break;
                        case pForced_2:
                            /*
                             * Forced "0" stick parity.
                             */
                            switch(device){
                                case U0:
                                    LPC_UART0->LCR |= (0b11 << 4);
                                    break;
                                case U1:
                                    LPC_UART1->LCR |= (0b11 << 4);
                                    break;
                                case U2:
                                    LPC_UART2->LCR |= (0b11 << 4);
                                    break;
                                case U3:
                                    LPC_UART3->LCR |= (0b11 << 4);
                                    break;
                            }
                            break;
                        default:
                            return Invalid_Parity_Config;
                    }
                }
                else {
                    switch(device){
                        case U0:
                            LPC_UART0->LCR &= ~(1 << 3);
                            break;
                        case U1:
                            LPC_UART1->LCR &= ~(1 << 3);
                            break;
                        case U2:
                            LPC_UART2->LCR &= ~(1 << 3);
                            break;
                        case U3:
                            LPC_UART3->LCR &= ~(1 << 3);
                            break;
                    }
                    if (parity_mode != pNone)
                        return Invalid_Parity_Config;
                }
            /*
             * Enable Break Control if Requested
             */
                switch(device){
                    case U0:
                        break_control ? (LPC_UART0->LCR |= (1 << 6)) : (LPC_UART0->LCR &= ~(1 << 6));
                        break;
                    case U1:
                        break_control ? (LPC_UART1->LCR |= (1 << 6)) : (LPC_UART1->LCR &= ~(1 << 6));
                        break;
                    case U2:
                        break_control ? (LPC_UART2->LCR |= (1 << 6)) : (LPC_UART2->LCR &= ~(1 << 6));
                        break;
                    case U3:
                        break_control ? (LPC_UART3->LCR |= (1 << 6)) : (LPC_UART3->LCR &= ~(1 << 6));
                        break;
                }
            /*
             * Set requested baud rate (see baud rate formula)
             */
                uint32_t pclk;
                uint8_t clk_div;
                switch(device){
                    case U0:
                        clk_div = ((LPC_SC->PCLKSEL1 >> 6) & 0x3);
                        break;
                    case U1:
                        clk_div = ((LPC_SC->PCLKSEL0 >> 8) & 0x3);
                        break;
                    case U2:
                        clk_div = ((LPC_SC->PCLKSEL1 >> 16) & 0x3);
                        break;
                    case U3:
                        clk_div = ((LPC_SC->PCLKSEL1 >> 18) & 0x3);
                        break;
                }
                switch (clk_div){
                    case clkdiv_1:
                        pclk = sys_get_cpu_clock();
                        break;
                    case clkdiv_2:
                        pclk = sys_get_cpu_clock() >> 1;
                        break;
                    case clkdiv_4:
                        pclk = sys_get_cpu_clock() >> 2;
                        break;
                    case clkdiv_8:
                        pclk = sys_get_cpu_clock() >> 3;
                        break;
                    default:
                        return Unspecified_Error;
                }
                uint16_t dl = (uint16_t)(pclk / ((uint32_t)baud_rate << 4));
                    /*
                     * Set DLAB bit to 1 to access DLL and DLM registers
                     */
                        switch (device){
                            case U0:
                                LPC_UART0->LCR |= (1 << 7);
                                break;
                            case U1:
                                LPC_UART1->LCR |= (1 << 7);
                                break;
                            case U2:
                                LPC_UART2->LCR |= (1 << 7);
                                break;
                            case U3:
                                LPC_UART3->LCR |= (1 << 7);
                                break;
                            default:
                                return Unspecified_Error;
                        }
                    /*
                     * Set DLM and DLL registers
                     */
                        switch (device){
                            case U0:
                                LPC_UART0->DLL = (dl & 0xFF);
                                LPC_UART0->DLM = ((dl >> 8) & 0xFF);
                                break;
                            case U1:
                                LPC_UART1->DLL = (dl & 0xFF);
                                LPC_UART1->DLM = ((dl >> 8) & 0xFF);
                                break;
                            case U2:
                                LPC_UART2->DLL = (dl & 0xFF);
                        LPC_UART2->DLM = ((dl >> 8) & 0xFF);
                        break;
                      case U3:
                        LPC_UART3->DLL = (dl & 0xFF);
                        LPC_UART3->DLM = ((dl >> 8) & 0xFF);
                        break;
                      default:
                        return Unspecified_Error;
                    }
              /*
               * Enable RX Interrupt if requested
               */
                /*
                 * Enable RX Interrupt (need to set DLAB to 0 first)
                 */
                  switch (device){
                    case U0:
                      LPC_UART0->LCR &= ~(1 << 7);
                      if (rx_interrupt_enable){
                        LPC_UART0->IER |=  (1 << 0);
                        NVIC_EnableIRQ(UART0_IRQn);
                        isr_register(UART0_IRQn, uart0_rx_intr);
                        u0_rx_queue = xQueueCreate(QUEUE_SIZE, sizeof(char));
                        intr_enabled = true;

                      }
                      else
                        LPC_UART0->IER &= ~(1 << 0);
                      break;
                    case U1:
                      LPC_UART1->LCR &= ~(1 << 7);
                      if (rx_interrupt_enable){
                        LPC_UART1->IER |=  (1 << 0);
                        NVIC_EnableIRQ(UART1_IRQn);
                        isr_register(UART1_IRQn, uart1_rx_intr);
                        u1_rx_queue = xQueueCreate(QUEUE_SIZE, sizeof(char));
                        intr_enabled = true;
                      }
                      else
                        LPC_UART1->IER &= ~(1 << 0);
                      break;
                    case U2:
                      LPC_UART2->LCR &= ~(1 << 7);
                      if (rx_interrupt_enable){
                        LPC_UART2->IER |=  (1 << 0);
                        NVIC_EnableIRQ(UART2_IRQn);
                        isr_register(UART2_IRQn, uart2_rx_intr);
                        u2_rx_queue = xQueueCreate(QUEUE_SIZE, sizeof(char));
                        intr_enabled = true;
                      }
                      else
                        LPC_UART2->IER &= ~(1 << 0);
                      break;
                    case U3:
                      LPC_UART3->LCR &= ~(1 << 7);
                      if (rx_interrupt_enable){
                        LPC_UART3->IER |=  (1 << 0);
                        NVIC_EnableIRQ(UART3_IRQn);
                        isr_register(UART3_IRQn, uart3_rx_intr);
                        u3_rx_queue = xQueueCreate(QUEUE_SIZE, sizeof(char));
                        intr_enabled = true;
                      }
                      else
                        LPC_UART3->IER &= ~(1 << 0);
                      break;
                    default:
                      return Unspecified_Error;
                  }
              /*
               * Start FIFO Queues and reset TX & RX Buffers
               */
                switch (device){
                  case U0:
                    LPC_UART0->FCR |= (1 << 0);
                    LPC_UART0->FCR |= ((1 << 1) | (1 << 2));
                    break;
                  case U1:
                    LPC_UART1->FCR |= (1 << 0);
                    LPC_UART1->FCR |= ((1 << 1) | (1 << 2));
                    break;
                  case U2:
                    LPC_UART2->FCR |= (1 << 0);
                    LPC_UART2->FCR |= ((1 << 1) | (1 << 2));
                    break;
                  case U3:
                    LPC_UART3->FCR |= (1 << 0);
                    LPC_UART3->FCR |= ((1 << 1) | (1 << 2));
                    break;
                  default:
                    return Unspecified_Error;
```

```
                }
            /*
             * Enable RX Status Line Interrupts
             */
                switch (device){
                    case U0:
                        LPC_UART0->IER |= (1 << 2);
                        break;
                    case U1:
                        LPC_UART1->IER |= (1 << 2);
                        break;
                    case U2:
                        LPC_UART2->IER |= (1 << 2);
                        break;
                    case U3:
                        LPC_UART3->IER |= (1 << 2);
                        break;
                    default:
                        return Unspecified_Error;
                }
            this_device = device;
            initialized = true;
            return Success;
        }
        default:
            return No_Device_Initialized;
    }
}

int LabUart::Transmit(char c)
{
    switch(this_device){
        case U0:
            while (!((LPC_UART0->LSR >> 5) & 1));
            LPC_UART0->THR = c;
            break;
        case U1:
            while (!((LPC_UART1->LSR >> 5) & 1));
            LPC_UART1->THR = c;
            break;
        case U2:
            while (!((LPC_UART2->LSR >> 5) & 1));
            LPC_UART2->THR = c;
            break;
        case U3:
            while (!((LPC_UART3->LSR >> 5) & 1));
            LPC_UART3->THR = c;
            break;
        default:
            return No_Device_Initialized;

    }
    return Success;
}

char LabUart::Receive()
{
    char c = NULL;
    if (intr_enabled){
        switch(this_device){
            case U0:
                xQueueReceive(u0_rx_queue, &c, DEQUEUE_TIMEOUT);
                break;
            case U1:
                xQueueReceive(u1_rx_queue, &c, DEQUEUE_TIMEOUT);
                break;
            case U2:
                xQueueReceive(u2_rx_queue, &c, DEQUEUE_TIMEOUT);
                break;
            case U3:
                xQueueReceive(u3_rx_queue, &c, DEQUEUE_TIMEOUT);
                break;
        }
    return c;
    }
    else{
        switch(this_device){
            case U0:
                while (!(LPC_UART0->LSR & 1));
                    c = LPC_UART0->RBR;
                break;
            case U1:
                while (!(LPC_UART1->LSR & 1));
                    c = LPC_UART1->RBR;
                break;
            case U2:
                while (!(LPC_UART2->LSR & 1));
                    c = LPC_UART2->RBR;
                break;
            case U3:
                while (!(LPC_UART3->LSR & 1));
                    c = LPC_UART3->RBR;
                break;
        }
    return c;
    }
}

void LabUart::uart0_rx_intr()
{
    char c = LPC_UART0->RBR;
    xQueueSend(u0_rx_queue, &c, ENQUEUE_TIMEOUT);
}

void LabUart::uart1_rx_intr()
{
    char c = LPC_UART1->RBR;
    xQueueSend(u1_rx_queue, &c, ENQUEUE_TIMEOUT);
}

void LabUart::uart2_rx_intr()
{
    char c = LPC_UART2->RBR;
    xQueueSend(u2_rx_queue, &c, ENQUEUE_TIMEOUT);
}

void LabUart::uart3_rx_intr()
{
    char c = LPC_UART3->RBR;
    xQueueSend(u3_rx_queue, &c, ENQUEUE_TIMEOUT);
}

```