Main.cpp

```
LabUART A;
uint8_t received[3], final, received1, MSB, LSB, total;
int i = 0;
int j = 0;
QueueHandle_t my_queue;
void my_uart2_rx_intr(void)
{
    if(LPC_UART2->IIR & (4<<0)){
        received1 = LPC_UART2->RBR;
        xQueueSendFromISR(my_queue, &received1, NULL);
    }
}
uint8_t calculate(uint8_t receivedVal1, uint8_t receivedVal2,
        uint8_t receivedChar){
    switch(receivedChar){
        case(0x2B):
        return final = receivedVal1 + receivedVal2;
        break;
        case(0x2D):
        return final = receivedVal1 - receivedVal2;
        break;
        case(0x2A):
        return final = receivedVal1 * receivedVal2;
        break; }
}
void masterTask(void *p)
{
    A.transfer(5);
    A.transfer(7);
    A.transfer('+');
    while (1) {
        {
            if (xQueueReceive(my_queue, &received1, portMAX_DELAY))
                received[j] = received1;
            printf("received: %x\n",
                    received[j]);
            j++; }
        if(j==2){
            total = received[0]*10  + received[1];
            LD.setNumber(total);
            printf("value: %i\n", total);
        } }
}

int main(void)
{
    A.init_my_uart2(2);
    isr_register(UART2_IRQn, my_uart2_rx_intr);
    my_queue = xQueueCreate(3, sizeof(int));
    xTaskCreate(masterTask, (const char*)"trc", STACK_BYTES(2048), 0, 1, 0);
    //xTaskCreate(slaveTask, (const char*)"trc", STACK_BYTES(2048), 0, 1, 0);

    scheduler_start();
    vTaskStartScheduler();
    return 0;
}
```

UARTLab_basics.cpp

```
/*
 *     SocialLedge.com - Copyright (C) 2013
 *
```

```
 *     This file is part of free software framework for embedded processors.
 *     You can use it and/or distribute it as long as this copyright header
 *     remains unmodified.  The code is free for personal use and requires
 *     permission to use in a commercial product.
 *
 *      THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS,
IMPLIED
 *      OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 *      MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS
SOFTWARE.
 *      I SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL,
OR
 *      CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 *
 *     You can reach the author of this software at :
 *          p r e e t . w i k i @ g m a i l . c o m
 */

/**
 * @file
 * @brief This is the application entry point.
 *          FreeRTOS and stdio printf is pre-configured to use uart0_min.h
before main() enters.
 *          @see L0_LowLevel/lpc_sys.h if you wish to override printf/scanf
functions.
 *
 */
#include "tasks.hpp"
#include "examples/examples.hpp"
#include "FreeRTOS.h"
#include "task.h"
#include "uart0_min.h"
#include "labGPIO_0.hpp"
#include "utilities.h"
#include "io.hpp"
#include "storage.hpp"
#include "printf_lib.h"
#include "queue.h"


/**
 * The main() creates tasks or "threads".  See the documentation of
scheduler_task class at scheduler_task.hpp
 * for details.  There is a very simple example towards the beginning of this
class's declaration.
 *
 * @warning SPI #1 bus usage notes (interfaced to SD & Flash):
 *      - You can read/write files from multiple tasks because it
automatically goes through SPI semaphore.
 *      - If you are going to use the SPI Bus in a FreeRTOS task, you need to
use the API at L4_IO/fat/spi_sem.h
 *
 * @warning SPI #0 usage notes (Nordic wireless)
 *      - This bus is more tricky to use because if FreeRTOS is not running,
the RIT interrupt may use the bus.
 *      - If FreeRTOS is running, then wireless task may use it.
 *        In either case, you should avoid using this bus or interfacing to
external components because
 *        there is no semaphore configured for this bus and it should be used
exclusively by nordic wireless.
 */


QueueHandle_t xQueue1;
```

```
void my_uart3_rx_intr(void)
{
  // TODO: Queue your data and clear UART Rx interrupt
        u0_dbg_printf("inside the interrupt routine\n");
        char test_send = 'D';
        if(LPC_UART3->IIR & (4<<0))
        {
             test_send = LPC_UART3->RBR;
        }
        if(xQueueSendFromISR( xQueue1, &test_send,NULL))
        {
            u0_dbg_printf("data sent to Queue %c\n", test_send);
        }
        vTaskDelay(1000);
        LPC_UART3->LCR &= ~(1 << 7);  // Enable DLAB
        LPC_UART3->IER &= ~(1 << 0); // Reset the Interrupt
        //LPC_UART3->IIR |= (1 << 0); // Reset the Interrupt
};

void init_my_uart3(void)
{
  // Init PINSEL, baud rate, frame size, etc.
    //LPC_SC->PCONP &= ~(1 << 25); //UART3 Power Reset
    LPC_SC->PCONP |= (1 << 25); // UART3 Power Enable
    LPC_SC->PCLKSEL1 &= ~(3 << 18); //CLR CLK
    LPC_SC->PCLKSEL1 |= (1 << 18); // SET CLK
    LPC_UART3->LCR = 0x03; //bits 7:0 -- parity
    LPC_UART3->LCR |= (1 << 7); //DLAB is enabled
    uint16_t dll = 48000000 / (16 * 4800); //calculating the baud
    LPC_UART3->DLL = dll; // setting baud rate
    LPC_UART3->LCR = 3; //DLAB is disable -- set to 8bit transfer
    LPC_PINCON->PINSEL9 |= ((1 << 25) | (1 << 24)); //set bit 25:24 11 for
TXD3
    LPC_PINCON->PINSEL9 |= ((1 << 27) | (1 << 26)); //set bit 27:26 11 for
RXD3
    LPC_PINCON->PINMODE9 &= ~((1 << 13) | (1 << 12)); //set bit 25:24 for 00
pull up -- disable pull-down resistor
    LPC_PINCON->PINMODE9 &= ~((1 << 13) | (1 << 12)); //set bit 27:26 for 00
pull up

  // Init UART Rx interrupt (TX interrupt is optional)
    NVIC_EnableIRQ(UART3_IRQn);
    LPC_UART3->IER |= (1 << 0); //Enable Receive Data Interrupt
    //isr_register(UART3_IRQn, my_uart3_rx_intr);
}

void sender_task(char pVcharacter)
{
    if(!(LPC_UART3->LSR & (1 << 6)))
    {
        LPC_UART3->THR = pVcharacter;
    }
}

void my_task(void *p)
{
    char test_rx = 'B';
    char temp;
    while (1)
    {
        //sender_task('A');
        //sender_task('L');
        //sender_task('I');
```

```
        temp = 'A';
        LPC_UART3->THR = temp;
        u0_dbg_printf("Send first character %c\n", temp);
        LPC_UART3->THR = 'B';
        LPC_UART3->THR = 'C';
        if(xQueueReceive(xQueue1, &test_rx, 1000))
        {
            u0_dbg_printf("Recieved data %c\n", test_rx);
        }
        else
        {
            u0_dbg_printf("Failed to recieve data\n");
        }
    }
}

int main(void) {

    static uint8_t args;
    args = 0;
    TaskHandle_t handler = NULL;
    uint32_t STACK_SIZE = 2048;
    isr_register(UART3_IRQn, my_uart3_rx_intr);
    xQueue1 = xQueueCreate( 10, sizeof( char ) );
    if(xQueue1 == NULL )
       {
           /* Queue was not created and must not be used. */
       }

    init_my_uart3();
    xTaskCreate(my_task, "Transmit and Receive Queue", STACK_SIZE, &args,
PRIORITY_HIGH, &handler );
    //xTaskCreate(my_uart3_rx_intr, "Sender Task", STACK_SIZE, &args,
PRIORITY_MEDIUM, &handler );
    //xTaskCreate(task1, "Nothing task", STACK_SIZE, &args, PRIORITY_MEDIUM,
&handler);


    //xTaskCreate( task2, "task_2", STACK_SIZE, &args, PRIORITY_HIGH,
&handler);
    vTaskStartScheduler();
    //lab8
    //  scheduler_add_task(new consumertask(PRIORITY_MEDIUM));
    //  scheduler_add_task(new producertask(PRIORITY_MEDIUM));
    //  scheduler_add_task(new watchdogtask(PRIORITY_HIGH));
    //

        scheduler_add_task(new terminalTask(PRIORITY_HIGH));

        scheduler_start(); ///< This shouldn't return
        return -1;

}
```

LabUart.cpp

```
#include "uartDriver.hpp"
void LabUART::init_my_uart2(int portValue)
{
    if(portValue == 2){
        LPC_SC->PCONP |= (1<<24); //init uart2 and uart3
        LPC_SC->PCLKSEL1 &= ~(0x3 << 16);
        LPC_SC->PCLKSEL1 |= (1 << 16);
        LPC_UART2->LCR |= (1<<7);
```

```
        LPC_UART2->DLL |= (0x38<<0);
        LPC_UART2->DLM |= (0x01<<0);
        LPC_UART2->LCR |= (3<<0);
        LPC_UART2->LCR &= ~(1<<7);
        LPC_PINCON->PINSEL4 &= ~(0xF<<16);
        LPC_PINCON->PINSEL4 |= (0xA<<16);
        LPC_UART2->FCR |= (1<<0);
        LPC_UART2->IER |= (1<<0);
        NVIC_EnableIRQ(UART2_IRQn);
    }
    else if(portValue == 3){
        LPC_SC->PCONP |= (1<<25);
        LPC_SC->PCLKSEL1 &= ~(0x3 << 18);
        LPC_UART3->LCR |= (1<<7);
        LPC_UART3->DLL |= (0x38<<0);
        LPC_UART3->DLM |= (0x01<<0);
        LPC_UART3->LCR |= (3<<0);
        LPC_UART3->LCR &= ~(1<<7);
        LPC_PINCON->PINSEL0 &= ~(0xF<<0);
        LPC_PINCON->PINSEL0 |= (0xA<<16);
        LPC_PINCON->PINSEL9 &= ~(0xF<<24);
        LPC_PINCON->PINSEL9 |= (0xF<<24);
        LPC_UART3->FCR |= (1<<0);
        LPC_UART3->IER |= (1<<0);
        NVIC_EnableIRQ(UART3_IRQn);
    }
}
void LabUART::transfer(uint8_t value){
    LPC_UART2->THR |= (value<<0);
    while(!(LPC_UART2->LSR & (1 << 6)));
}
LabUART::LabUART(){
}
LabUART::~LabUART(){

}
```

LabUart.hpp

```
#ifndef UARTDRIVER_HPP_
#define UARTDRIVER_HPP_
#include <iostream>
#include <stdint.h>
#include <stdio.h>
#include "FreeRTOS.h"
#include "queue.h"
#include "lpc_isr.h"
#include "handlers.hpp"
#ifndef LABUART_H
#define LABUART_H

class LabUART
{
private:
public:
    uint8_t resultMSB, resultLSB, received;
    char operatorVal;
    void init_my_uart2(int portValue);
    void transfer(uint8_t value);
    LabUART();
    ~LabUART();
};
#endif
```

```
#endif /* UARTDRIVER_HPP_ */
```