```
/*
 * LabUart.hpp
 *
 *  Created on: Mar 9, 2019
 *      Author: chadw
 */

#ifndef LABUART_HPP_
#define LABUART_HPP_
#include <stdio.h>
#include <FreeRTOS.h>
#include "LPC17xx.h"
#include "sys_config.h"
#include <string.h>
#include <utilities.h>
#include <queue.h>

class LabUart
{
 public:

    enum Port {
        uart2,
        uart3
    };

    enum Parity {
        none,
        odd,
        even
    };

    typedef void (*IsrPointer)(void);

    /**
    * Constructor/Deconstructor. No additional code here.
    */
    LabUart();
    ~LabUart();

    /**
    * 1) Initialize UART 2 or 3.
    * 2) Putting port only will default port to 38400bps, 8-N-1 configuration.
    *
    * @param port_arg is the UART port: uart2 or uart3.
    * @param baud_rate sets the baud rate. Default is 38400 bps
    * @param data_length is the length of the data being transmitted/received. Must
be 5, 6, 7 or 8. Default is 8.
    * @param parity_bit chooses the parity bit type. None, odd, even. Default is
none.
    * @param stop_bits chooses quanitity of stop bits. 1 or 2. Default is 1
    * @param receiving queue size. Default is 100.
    */
    bool Initialize(Port port_arg, uint16_t queue_size = 100, uint32_t baud_rate =
38400, uint8_t data_length = 8, Parity parity_bit = none,
                    uint8_t stop_bits = 1);
```

```
    void Transmit(const char *text);

    void Transmit(const char text);

    char Receive();

    bool Available();

    void SetBaudRate(uint32_t baud_rate);

    void InteruptHandler();

    void AttatchHandler(IsrPointer p);

 private:
    Port port;
    LPC_UART_TypeDef* UART_REG;
    QueueHandle_t uart_queue;
    uint16_t qSize;
};



#endif /* LABUART_HPP_ */
```

```
/*
 * LabUart.cpp
 *
 *  Created on: Mar 9, 2019
 *      Author: chadw
 */
#include "LabUart.hpp"

LabUart::LabUart() {}
LabUart::~LabUart() {}

//extern "C"
//{
//    void UART_IRQHandler()
//    {
//        Receive();
//    }
//}

bool LabUart::Initialize(Port port_arg, uint16_t queue_size, uint32_t baud_rate,
uint8_t data_length, Parity parity_bit,
                uint8_t stop_bits)
{
    port = port_arg;

    if(port == uart2)
    {
        UART_REG = LPC_UART2;
    }
    else if (port == uart3)
```

```
{
    UART_REG = LPC_UART3;
}
else
{
    return false;
}

switch (port_arg) {
    case uart2:
        LPC_SC->PCONP |= (1 << 24);         //Power on UART2
        LPC_SC->PCLKSEL1 &= ~(3 << 16);     //Set PCLK to CCLK/1 for UART2
        LPC_SC->PCLKSEL1 |= (1 << 16);
        break;
    case uart3:
        LPC_SC->PCONP |= (1 << 25);         //Power on UART3
        LPC_SC->PCLKSEL1 &= ~(3 << 18);     //Set PCLK to CCLK/1 for UART 3
        LPC_SC->PCLKSEL1 |= (1 << 18);
        break;
}

SetBaudRate(baud_rate);

if(data_length == 5)
{
    UART_REG->LCR &= ~(3 << 0);
}
else if(data_length == 6)
{
    UART_REG->LCR &= ~(3 << 0);
    UART_REG->LCR |= (1 << 0);
}
else if(data_length == 7)
{
    UART_REG->LCR &= ~(3 << 0);
    UART_REG->LCR |= (1 << 1);
}
else if(data_length == 8)
{
    UART_REG->LCR |= (3 << 0);
}
else
{
    return false;
}

if(parity_bit == none)
{
    UART_REG->LCR &= ~(1 << 3);
}
else if (parity_bit == even)
{
    UART_REG->LCR |= (1 << 3);
    UART_REG->LCR &= ~(3 << 4);
    UART_REG->LCR |= (1 << 4);
}
else if (parity_bit == odd)
```

```
    {
        UART_REG->LCR |= (1 << 3);
        UART_REG->LCR &= ~(3 << 4);
    }
    else
    {
        return false;
    }

    if(stop_bits == 1)
    {
        UART_REG->LCR &= ~(1 << 2);
    }
    else if (stop_bits == 2)
    {
        UART_REG->LCR |= (1 << 2);
    }
    else
    {
        return false;
    }

    UART_REG->FCR = 0x01;

    if(port == uart2)
    {
        LPC_PINCON->PINSEL4 &= ~(3 << 16); //Setup Pins
        LPC_PINCON->PINSEL4 |= (1 << 17);
        LPC_PINCON->PINSEL4 &= ~(3 << 18);
        LPC_PINCON->PINSEL4 |= (1 << 19);
        LPC_PINCON->PINMODE4 &= ~(3 << 16);
        LPC_PINCON->PINMODE4 |= (1 << 17);
        LPC_PINCON->PINMODE4 &= ~(3 << 18);
        LPC_PINCON->PINMODE4 |= (1 << 19);
        NVIC_EnableIRQ(UART2_IRQn);
    }
    else if(port == uart3)
    {
      LPC_PINCON->PINSEL9 |= (3 << 24); //Setup Pins
      LPC_PINCON->PINSEL9 |= (3 << 26);
      LPC_PINCON->PINMODE9 &= ~(3 << 24);
      LPC_PINCON->PINMODE9 |= (1 << 25);
      LPC_PINCON->PINMODE9 &= ~(3 << 26);
      LPC_PINCON->PINMODE9 |= (1 << 27);
      NVIC_EnableIRQ(UART3_IRQn);
    }

    UART_REG->LCR &= ~(1 << 7);        //Clear DLAB
    UART_REG->IER |= (1 << 0);         //Enable Receive Interupt

    uart_queue = xQueueCreate(queue_size, sizeof(char));

    return true;
}

void LabUart::Transmit(const char *text)
{
```

```
    for(int i = 0; i < (int)strlen(text); i++)
    {
    UART_REG->THR = text[i];
    delay_ms(1);
    }
}

void LabUart::Transmit(const char text)
{
    UART_REG->THR = text;
    delay_ms(1);
}

char LabUart::Receive()
{
    uint8_t r;

    xQueueReceive(uart_queue, &r, 0);
    delay_ms(2);
    return r;
}

bool LabUart::Available()
{
    uint8_t b;
    if(xQueuePeek(uart_queue, &b, 0))
    {
        return false;
    }
    else
    {
        return true;
    }
}

void LabUart::SetBaudRate(uint32_t baud_rate)
{
    unsigned int pclk = sys_get_cpu_clock();
    uint16_t divisor_latch = pclk / 16 / baud_rate;

    UART_REG->LCR |= (1 << 7);                          //Set DLAB
    UART_REG->DLM = (divisor_latch >> 8) & 0x00FF;     //Set Baud rate Divisor Latch
MSB
    UART_REG->DLL = (divisor_latch) & 0x00FF;          //Set Baud rate Divisor Latch
LSB
    UART_REG->LCR &= ~(1 << 7);                         //Clear DLAB
}

void LabUart::InteruptHandler()
{
    uint8_t recv_byte;
    if( ! (UART_REG->LSR & (1 << 2)) || (UART_REG->LSR & (1 << 3)) || (UART_REG->LSR
& (1 << 4)) )
    {
         recv_byte = UART_REG->RBR;
    }
```

```
    xQueueSend(uart_queue, &recv_byte, 0);
}

void LabUart::AttatchHandler(IsrPointer p)
{
    if(port == uart2)
    {
        isr_register(UART2_IRQn, p);
    }
    else if(port == uart3)
    {
        isr_register(UART3_IRQn, p);
    }

}
```

```
/**Main.cpp**/

#include "tasks.hpp"
#include "LPC17xx.h"
#include <stdio.h>
#include <utilities.h>
#include <queue.h>
#include <LabUart.hpp>
#include <string.h>
#include <io.hpp>

LabUart my_uart2;
LabUart my_uart3;

void isrUart2(void)
{
    my_uart2.InteruptHandler();
}

void isrUart3(void)
{
    my_uart3.InteruptHandler();
}

void math_recv(void *p)
{
    while(1)
    {
        char digit1, digit2, op;
        char answer = 0;
        my_uart2.Initialize(LabUart::uart2, 20);
        my_uart2.AttatchHandler(isrUart2);

        while(1)
```

```
        {
            if(!my_uart2.Available())
            {
                digit1 = my_uart2.Receive() - 48;
                digit2 = my_uart2.Receive() - 48;
                op = my_uart2.Receive();
                break;
            }
        }

        if(op == 43)
        {
            answer = digit1 + digit2;
        }
        else if(op == 45)
        {
            answer = digit1 - digit2;
        }
        else if(op == 42)
        {
            answer = digit1 * digit2;
        }

        my_uart2.Transmit(answer);
        LD.setNumber(answer);

        vTaskDelay(10);
    }
}

void math_send(void *p)
{
    char receive;
    my_uart2.Initialize(LabUart::uart2, 20);
    my_uart2.AttatchHandler(isrUart2);

    //send digits/operator
    my_uart2.Transmit('6');
    my_uart2.Transmit('5');
    my_uart2.Transmit('*');

    //Receive Result
    while(1)
    {
        if(!my_uart2.Available())
        {
            receive = my_uart2.Receive();
            LD.setNumber(receive - 48);
        }
    }
}
/**
```

```
 * 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.
 */
int main(void)
{

   /// This "stack" memory is enough for each task to run properly (512 * 32-
bit) = 2Kbytes stack
        const uint32_t STACK_SIZE_WORDS = 512;
        xTaskCreate(math_send, "math", STACK_SIZE_WORDS, NULL, PRIORITY_LOW,
NULL);
```