Lab: UART

Objective

- To learn how to communicate between two devices using UART.
- **Reinforce** interrupts by setting up an interrupt on receive (Part 2)
- **Reinforce** RTOS queues
- It is required to **finish** `Part 0` **and** `Part 1` prior to your lab's start time

Assignment

This assignment will require a partner except for Part 0 and Part 1 . The overall idea is to interface two boards using your UART driver. It may be best to test a single UART driver using loopback (tie your own RX and TX wires together) in order to ensure that your driver is functional before trying to put two boards together.

Part 0: Implement UART driver

- Before you start the assignment, please read **Chapter 18: UART0/2/3** in your LPC User manual (`UM10562.pdf`). You can skip the sections related to FIFO, interrupts or the DMA.
  - From LPC User manual to understand the different registers involved in writing a UART driver
  - Refer `Table 84` from LPC User manual for pin configuration
- From the schematics pdf(`Schematics-RevE.2.pdf`), identify the pins numbers which can do UART2 (U2_TXD/U2_RXD) and UART3(U3_TXD/U3_RXD) and make a note of it because you will be needing them for pin function configuration

Implement `uart_lab.h` and `uart_lab.c` :

```
#pragma once
#include <stdint.h>
#include <stdbool.h>
typedef enum {
  UART_2,
```

```
  UART_3,
} uart_number_e;
void uart_lab__init(uart_number_e uart, uint32_t peripheral_clock, uint32_t baud_rate) {
  // Refer to LPC User manual and setup the register bits correctly
  // The first page of the UART chapter has good instructions
  // a) Power on Peripheral
  // b) Setup DLL, DLM, FDR, LCR registers
}
// Read the byte from RBR and actually save it to the pointer
bool uart_lab__polled_get(uart_number_e uart, char *input_byte) {
  // a) Check LSR for Receive Data Ready
  // b) Copy data from RBR register to input_byte
}
bool uart_lab__polled_put(uart_number_e uart, char output_byte) {
  // a) Check LSR for Transmit Hold Register Empty
  // b) Copy output_byte to THR register}
```

The divider equation in the LPC user manual is a bit confusing, please reference the code below to figure out how to set the dividers to achieve your baud rate.

```
    /* Baud rate equation from LPC user manual:
     * Baud = PCLK / (16 * (DLM*256 + DLL) * (1 + DIVADD/DIVMUL))
     *
     * What if we eliminate some unknowns to simplify?
     * Baud = PCLK / (16 * (DLM*256 + DLL) * (1 + 0/1))
     * Baud = PCLK / (16 * (DLM*256 + DLL)
     *
     * | DLM | DLL | is nothing but 16-bit number
     * DLM multiplied by 256 is just (DLM << 8)
     *
     * The equation is actually:
     * Baud = PCLK / 16 * (divider_16_bit)
     */
    const uint16_t divider_16_bit = 96*1000*1000 / (16 * baud_rate);
    LPC_UART2->DLM = (divider_16_bit >> 8) & 0xFF;
    LPC_UART2->DLL = (divider_16_bit >> 0) & 0xFF;
```

Part 1: Loopback test of **UART** driver

1. Choose either UART2 or UART3. Connect TX with RX pin.
2. Implement two tasks which reads whatever you just write and test if everything works successfully.
3. Note that you will use the "polled" version of the driver that will consume the CPU while waiting for data to be received. Once you get this working, the next part will utilize interrupt driven approach.

```
#include "FreeRTOS.h"
#include "task.h"
#include "uart_lab.h"
void uart_read_task(void *p) {
  while (1) {
    //TODO: Use uart_lab__polled_get() function and printf the received value
    vTaskDelay(500);
  }
}
void uart_write_task(void *p) {
  while (1) {
    //TODO: Use uart_lab__polled_put() function and send a value
```

```
    vTaskDelay(500);
  }
}
void main(void) {
  //TODO: Use uart_lab__init() function and initialize UART2 or UART3 (your choice)
  //TODO: Pin Configure IO pins to perform UART2/UART3 function

  xTaskCreate(uart_read_task, ...);
  xTaskCreate(uart_write_task, ...);
  vTaskStartScheduler();}
```

Part 2: Receive with Interrupts

Instead of polling for data to be received, you will extend your UART driver to trigger an interrupt when there is data available to be read.

```
// file: uart_lab.c
// TODO: Implement the header file for exposing public functions (non static)
// The idea is that you will use interrupts to input data to FreeRTOS queue
// Then, instead of polling, your tasks can sleep on data that we can read from the queue
#include "FreeRTOS.h"
#include "queue.h"
// Private queue handle of our uart_lab.c
static QueueHandle_t your_uart_rx_queue;
// Private function of our uart_lab.c
static void your_receive_interrupt(void) {
  // TODO: Read the IIR register to figure out why you got interrupted

  // TODO: Based on IIR status, read the LSR register to confirm if there is data to be read

  // TODO: Based on LSR status, read the RBR register and input the data to the RX Queue
  const char byte = UART->RBR;
  xQueueSendFromISR(your_uart_rx_queue, &byte, NULL);
}
```

```
// Public function to enable UART interrupt
// TODO Declare this at the header file
void uart__enable_receive_interrupt(uart_number_e uart_number) {
  // TODO: Use lpc_peripherals.h to attach your interrupt
  lpc_peripheral__enable_interrupt(..., your_receive_interrupt);
  // TODO: Enable UART receive interrupt by reading the LPC User manual
  // Hint: Read about the IER register

  // TODO: Create your RX queue
  your_uart_rx_queue = xQueueCreate(...);
}
// Public function to get a char from the queue (this function should work without modification)
// TODO: Declare this at the header file
bool uart_lab__get_char_from_queue(char *input_byte, uint32_t timeout) {
  return xQueueReceive(your_uart_rx_queue, input_byte, timeout);}
```

Part 3: Interface two SJ2 boards with UART

- After all the above parts are completed and successfully tested, you are now ready to establish communication between 2 SJ2 boards.
- Assign one board as **Sender** and another as **Receiver**. Connect Tx pin of Sender to Rx pin of Receiver and Rx pin of Sender to Tx pin of Receiver. Do not forget the common ground.
- Have one board send a random number, one char at a time over UART
  - See reference code below; most of the code is setup for you at `board_1_sender_task()`
- Have the other board re-assemble this number
  - You will have to figure out some of the logic yourself

```
#include <stdlib.h>

// This task is done for you, but you should understand what this code is doing
void board_1_sender_task(void *p) {
  char number_as_string[16] = { 0 };

   while (true) {
     const int number = rand();
     sprintf(number_as_string, "%i", number);
```

```
    // Send one char at a time to the other board including terminating NULL char
    for (int i = 0; i <= strlen(number_as_string); i++) {
      uart_lab__polled_put(number_as_string[i]);
      printf("Sent: %c\n", number_as_string[i]);
    }

    printf("Sent: %i over UART to the other board\n", number);
    vTaskDelay(3000);
  }
}
void board_2_receiver_task(void *p) {
  char number_as_string[16] = { 0 };
  int counter = 0;
  while (true) {
    char byte = 0;
    uart_lab__get_char_from_queue(&byte, portMAX_DELAY);
    printf("Received: %c\n", byte);

    // This is the last char, so print the number
    if ('\0' == byte) {
      number_as_string[counter] = '\0';
      counter = 0;
      printf("Received this number from the other board: %s\n", number_as_string);
    }
    // We have not yet received the NULL '\0' char, so buffer the data
    else {
      // TODO: Store data to number_as_string[] array one char at a time
      // Hint: Use counter as an index, and increment it as long as we do not reach max value of 16
    }
  }}
```

? Make sure to connect the ground pins of both the boards together. Otherwise you will see scrambled characters.

Conclusion

- Submit all relevant files and files used (`uart_lab.h` and `uart_lab.c`)
  - Do not submit dead or commented code
  - Do not submit code you did not write
- Turn in any the screenshots of terminal output
- Logic Analyzer Screenshots: Waveform of UART transmission (or receive) between two boards
  - Make sure your logic analyzer is configured to decode UART protocol
  - Whole window screenshot with the **Decoded Protocols** (lower right hand side of window) clearly legible.

Revision #26
Created 6 years ago by sree harsha
Updated 3 years ago by Preet Kang