uart_lab.h

```c
//
// Created by nick_ on 10/3/2020.
//
#pragma once

#include <stdbool.h>
#include <stdint.h>
#include <stdlib.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);

// Read the byte from RBR and actually save it to the pointer
bool uart_lab__polled_get(uart_number_e uart, char *input_byte);

bool uart_lab__polled_put(uart_number_e uart, char output_byte);

void uart__enable_receive_interrupt(uart_number_e uart_number);

bool uart_lab__get_char_from_queue(char *input_byte, uint32_t timeout);
```

uart_lab.c

```c
//
// Created by nick_ on 10/3/2020.
//
#include "FreeRTOS.h"

#include "lpc40xx.h"
#include "lpc_peripherals.h"
#include "queue.h"
#include "uart_lab.h"
#include <stdio.h>

static const LPC_UART_TypeDef *uart_memory_map[] = {LPC_UART2, LPC_UART3};

static LPC_UART_TypeDef *uart__get_struct(uint8_t uart_port) { return
(LPC_UART_TypeDef *)uart_memory_map[uart_port]; }
```

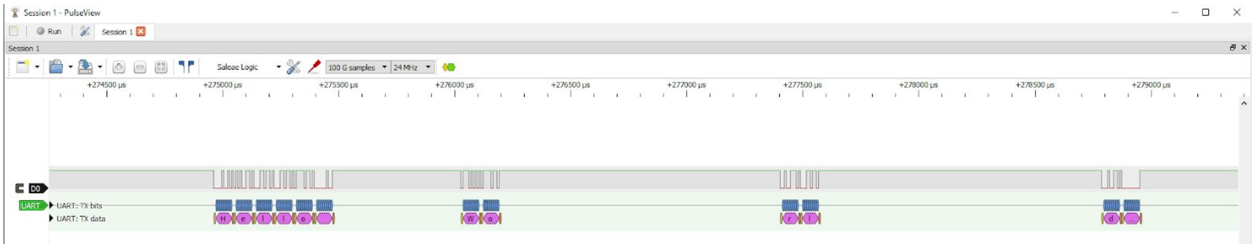
```
static QueueHandle_t your_uart_rx_queue;

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
 LPC_SC->PCONP |= (1 << ((uint32_t)uart + 24)); // register 24 and 25
 // b) Setup DLL, DLM, FDR, LCR registers
 const uint16_t divider_16_bit = 96 * 1000 * 1000 / (16 * baud_rate);
 uart__get_struct(uart)->LCR = (0x1 << 7); // DLAB
 uart__get_struct(uart)->DLM = (divider_16_bit >> 8) & 0xFF;
 uart__get_struct(uart)->DLL = (divider_16_bit >> 0) & 0xFF;
 uart__get_struct(uart)->FDR = (0x1 << 4); // not needed? Default
 uart__get_struct(uart)->FCR = 0x1;        // enable fifo
 uart__get_struct(uart)->LCR = 0x3;        // 8bit transfer

 // IOCON
 if (uart) {               // uart_3
   LPC_IOCON->P4_28 = 0x2; // tx
   LPC_IOCON->P4_29 = 0x2; // rx
 } else {                  // uart_2
   LPC_IOCON->P0_10 = 0x1; // tx
   LPC_IOCON->P0_11 = 0x1; // rx
 }
}

// 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
 bool status = uart__get_struct(uart)->LSR & (0x1 << 0); // RDR register
 // b) Copy data from RBR register to input_byte
 if (status) {
   *input_byte = (char)(uart__get_struct(uart)->RBR & 0xFF);
   //    fprintf(stderr, "\nReading value: %c\n", *input_byte);
 }
 return status;
}

bool uart_lab__polled_put(uart_number_e uart, char output_byte) {
 // a) Check LSR for Transmit Hold Register Empty
 bool status = uart__get_struct(uart)->LSR & (0x1 << 5); // THRE register
 // b) Copy output_byte to THR register
 if (status) {
   uart__get_struct(uart)->THR |= output_byte;
   //    fprintf(stderr, "writing value: %c\n", output_byte);
```

```c
 }
 return status;
}

// Private function of our uart_lab.c
static void your_receive_interrupt_uart3(void) {
 // TODO: Read the IIR register to figure out why you got interrupted
 uint8_t uart_port = 1;
 if (!uart__get_struct(uart_port)->IIR & 0x1) {
   return;
 }
 //  fprintf(stderr, "\ninterrupt on uart_%d\n", uart_port + 2);
 // 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
 char byte = 0;
 while (!uart_lab__polled_get(uart_port, &byte)) {
   ;
 }
 //  fprintf(stderr, "\nReading value: %c\n", byte);
 xQueueSendFromISR(your_uart_rx_queue, &byte, NULL);
}

static void your_receive_interrupt_uart2(void) {
 // TODO: Read the IIR register to figure out why you got interrupted
 uint8_t uart_port = 0;
 if (!uart__get_struct(uart_port)->IIR & 0x1) {
   return;
 }
 //  fprintf(stderr, "\ninterrupt on uart_%d\n", uart_port + 2);
 // 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
 char byte = 0;
 while (!uart_lab__polled_get(uart_port, &byte)) {
   ;
 }
 //  fprintf(stderr, "\nReading value: %c\n", byte);
 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) {
 uart__get_struct(uart_number)->IER = 0x1; // enable interrupt
 if (uart_number) {                        // uart_3
   lpc_peripheral__enable_interrupt(LPC_PERIPHERAL__UART3,
your_receive_interrupt_uart3, "uart3");
 } else { // uart_2
   lpc_peripheral__enable_interrupt(LPC_PERIPHERAL__UART2,
your_receive_interrupt_uart2, "uart2");
 }
 your_uart_rx_queue = xQueueCreate(16, sizeof(char));
}

// 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);
}
```

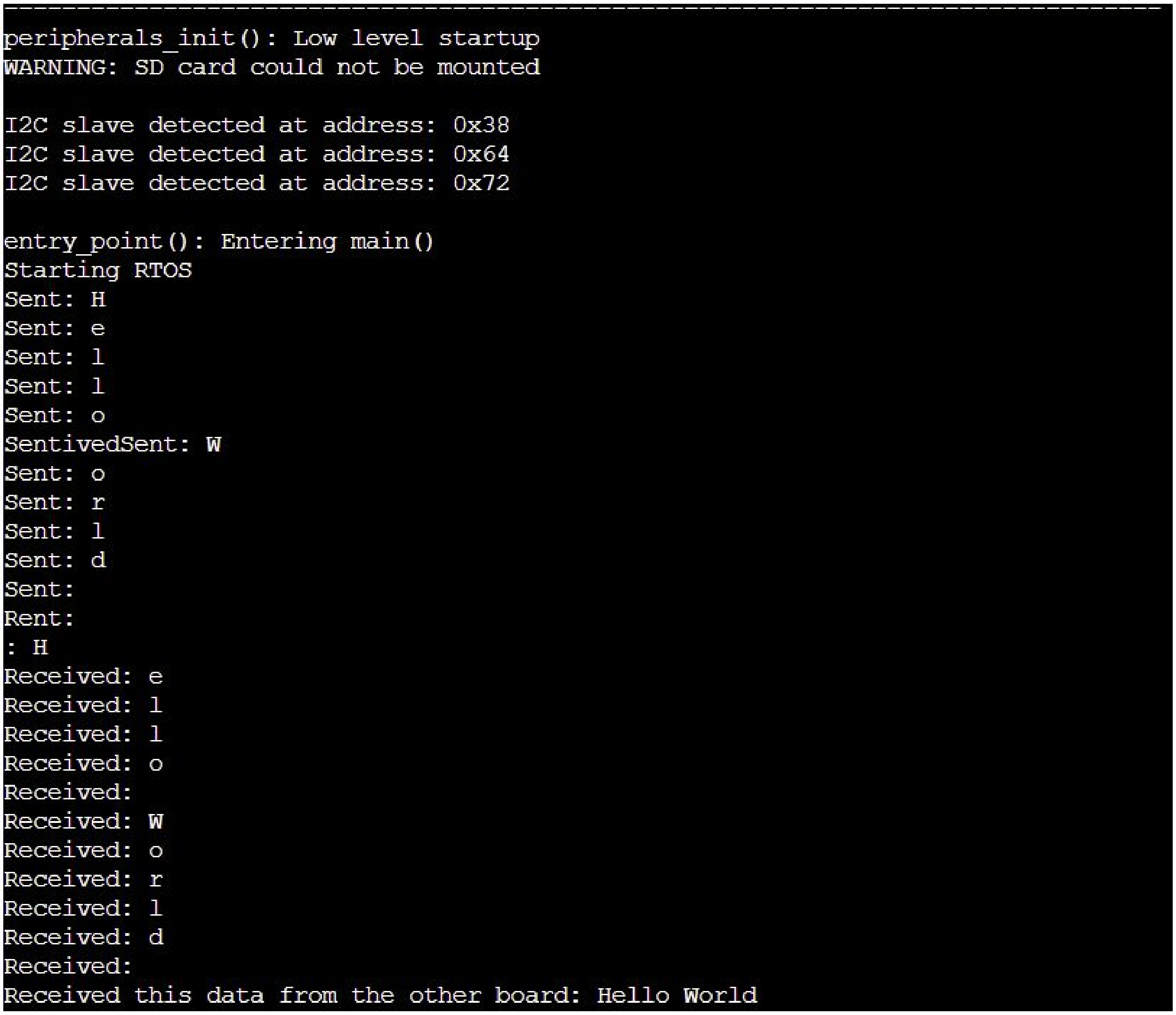
main.c

```
#if Lab6
void uart_read_task(void *p);
void uart_write_task(void *p);
void board_1_sender_task(void *p);
void board_2_receiver_task(void *p);
#endif

int main(void) { // main function for project
 puts("Starting RTOS");

#if outOfTheBox
 create_blinky_tasks();
 create_uart_task();
#else

 // TODO: Use uart_lab__init() function and initialize UART2 or UART3 (your choice)
 // TODO: Pin Configure IO pins to perform UART2/UART3 function
 uart_lab__init(UART_2, 96, 115200);
 uart__enable_receive_interrupt(UART_2);
//  uart_lab__init(UART_3, 96, 115200);
//  uart__enable_receive_interrupt(UART_3);

 xTaskCreate(board_2_receiver_task, /*description*/ "uart_task", /*stack depth*/
4096 / sizeof(void *),
             /*parameter*/ (void *)1,
             /*priority*/ 1, /*optional handle*/ NULL);
 xTaskCreate(board_1_sender_task, /*description*/ "uart_task2", /*stack depth*/
4096 / sizeof(void *),
             /*parameter*/ (void *)1,
             /*priority*/ 2, /*optional handle*/ NULL);
#endif
 vTaskStartScheduler(); // This function never returns unless RTOS scheduler runs
out of memory and fails
 return 0;
}

#if Lab6
void uart_read_task(void *p) {
 while (1) {
   // TODO: Use uart_lab__polled_get() function and printf the received value
   char testValue = 0;
   uart_lab__polled_get(UART_2, &testValue);
   fprintf(stderr, "%c", testValue);
   vTaskDelay(500);
 }
}

void uart_write_task(void *p) {
 char testString[] = "Hello world\n\n";
 int strlength = sizeof(testString) / sizeof(char);
 int i = 0;
 fprintf(stderr, "%s", testString);
 while (1) {
   // TODO: Use uart_lab__polled_put() function and send a value
   uart_lab__polled_put(UART_3, testString[i % strlength]);
   ++i;
   vTaskDelay(500);
 }
}
// 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[] = "Hello World";

 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(UART_3, number_as_string[i]);
//      uart_lab__polled_put(UART_2, 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 data 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
      number_as_string[counter] = byte;
      ++counter;
    }
  }
}
#endif
```



Pulse view Data Analyzer capture



```
-------------------------------------------------------------------------------
peripherals_init(): Low level startup
WARNING: SD card could not be mounted

I2C slave detected at address: 0x38
I2C slave detected at address: 0x64
I2C slave detected at address: 0x72

entry_point(): Entering main()
Starting RTOS
Sent: H
Sent: e
Sent: l
Sent: l
Sent: o
SentivedSent: W
Sent: o
Sent: r
Sent: l
Sent: d
Sent:
Rent:
: H
Received: e
Received: l
Received: l
Received: o
Received:
Received: W
Received: o
Received: r
Received: l
Received: d
Received:
Received this data from the other board: Hello World
```

Loop back from uart3 Tx to uart2 Rx