Main.c

```
#if Lab7
static QueueHandle_t switch_queue;

typedef enum { switch__off, switch__on } switch_e;

void producer(void *p);
void consumer(void *p);
#endif

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

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

 xTaskCreate(producer, /*description*/ "producer", /*stack depth*/ 4096 / sizeof(void *),
             /*parameter*/ (void *)1,
             /*priority*/ 1, /*optional handle*/ NULL);
 xTaskCreate(consumer, /*description*/ "consumer", /*stack depth*/ 4096 / sizeof(void *),
             /*parameter*/ (void *)1,
             /*priority*/ 2, /*optional handle*/ NULL);

 // TODO Queue handle is not valid until you create it
 switch_queue =
     xQueueCreate(1, sizeof(switch_e)); // Choose depth of item being our enum (1 should be
okay for this example)

#endif
 vTaskStartScheduler(); // This function never returns unless RTOS scheduler runs out of
memory and fails
 return 0;
}

#if Lab7
// TODO: Create this task at PRIORITY_LOW
void producer(void *p) {
 // SW3 p0.29
 LPC_IOCON->P0_29 &= ~(0x3 << 3);
 LPC_IOCON->P0_29 |= (1 << 3);  // pull down
 LPC_GPIO0->DIR &= ~(1 << 29); // set as input

 while (1) {
   // This xQueueSend() will internally switch context to "consumer" task because it is
higher priority than this
   // "producer" task Then, when the consumer task sleeps, we will resume out of
xQueueSend()and go over to the next
   // line
```

```
   // TODO: Get some input value from your board
   const switch_e switch_value = (LPC_GPIO0->PIN & (0x1 << 29)) >> 29; //value of switch

   // TODO: Print a message before xQueueSend()
   // Note: Use printf() and not fprintf(stderr, ...) because stderr is a polling printf
   printf("sending switch_value: %d\n", switch_value);
   xQueueSend(switch_queue, &switch_value, 0);
   // TODO: Print a message after xQueueSend()
   printf("Returned to producer task\n");

   vTaskDelay(1000);
 }
}

// TODO: Create this task at PRIORITY_HIGH
void consumer(void *p) {
 switch_e switch_value;
 while (1) {
   // TODO: Print a message before xQueueReceive()
   printf("waiting to consume a value from the queue\n");
   xQueueReceive(switch_queue, &switch_value, portMAX_DELAY);
   // TODO: Print a message after xQueueReceive()
   printf("received switch_value: %d\n", switch_value);
 }
}
#endif
```

- Use higher priority for `producer task`, and note down the order of the print-outs
    - Completed the producer task fully before starting the consumer task. This means that when I print out “ Returned to producer task” it is false in this case, because the consumer task was never completed.

```
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
sending switch_value: 0
Returned to producer task
waiting to consume a value from the queue
received switch_value: 0
waiting to consume a value from the queue
sending switch_value: 1
Returned to producer task
received switch_value: 1
waiting to consume a value from the queue
sending switch_value: 1
Returned to producer task
received switch_value: 1
waiting to consume a value from the queue
sending switch_value: 0
Returned to producer task
received switch_value: 0
waiting to consume a value from the queue
sending switch_value: 0
Returned to producer task
received switch_value: 0
waiting to consume a value from the queue
sending switch_value: 1
Returned to producer task
received switch_value: 1
waiting to consume a value from the queue
sending switch_value: 1
Returned to producer task
received switch_value: 1
waiting to consume a value from the queue
sending switch_value: 0
Returned to producer task
received switch_value: 0
waiting to consume a value from the queue
```

-

- Use higher priority for `consumer task`, and note down the order of the print-outs
    - The Consumer task begins and then is put to sleep while waiting for a value to be placed in the queue. Once the producer adds a value to the queue, the consumer task takes over to consume the value and once the consumer task is completed the producer task is able to finish.

```
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
waiting to consume a value from the queue
sending switch_value: 0
received switch_ucer taskwaiting to consume a value from the queue
Raiting to consume a valuesending switch_value: 0
received switch_value: 0
waiting to consume a value from the queue
Raiting to consume a valuesending switch_value: 1
received switch_value: 1
weturned to producer task
 from the queue
Returned to producer task
sending switch_value: 1
received switch_value: 1
weturned to producer task
 from the queue
Returned to producer task
sending switch_value: 0
received switch_value: 0
weturned to producer task
 from the queue
Returned to producer task
sending switch_value: 0
received switch_value: 0
weturned to producer task
 from the queue
Returned to producer task
sending switch_value: 1
received switch_value: 1
weturned to producer task
 from the queue
Returned to producer task
sending switch_value: 0
received switch_value: 0
weturned to producer task
 from the queue
Returned to producer task
sending switch_value: 0
received switch_value: 0
weturned to producer task
 from the queue
Returned to producer task
```

- Use same priority level for both tasks, and note down the order of the print-outs
    - In this case, it appears that the two tasks begin by using round robin, but the delay in the producer task is long enough to prevent the two tasks from overlapping. From that point on it runs as if the producer has higher priority.

```
-----------------------------------------------
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
waiting to consume a value from the queue
sending switch_value: 0
Returned to prodvalue: 0

received switch_value: 0
waiting to consume a value from the queue
sending switch_value: 0
Returned to producer task
received switch_value: 0
waiting to consume a value from the queue
sending switch_value: 0
Returned to producer task
received switch_value: 0
waiting to consume a value from the queue
sending switch_value: 1
Returned to producer task
received switch_value: 1
waiting to consume a value from the queue
sending switch_value: 1
Returned to producer task
received switch_value: 1
waiting to consume a value from the queue
sending switch_value: 0
Returned to producer task
received switch_value: 0
waiting to consume a value from the queue
sending switch_value: 0
Returned to producer task
received switch_value: 0
waiting to consume a value from the queue
sending switch_value: 0
Returned to producer task
received switch_value: 0
waiting to consume a value from the queue
sending switch_value: 1
Returned to producer task
received switch_value: 1
waiting to consume a value from the queue
```

- What is the purpose of the block time during `xQueueReceive()`?
    - The block time is the maximum amount of time the task will be blocked if the queue is empty when xQueueReceive is called. The task can be unblocked sooner than the designated wait time if an item is added to the queue while waiting.
- What if you use ZERO block time during `xQueueReceive()`?
    - If the blocking time is set to 0, xQueueRecieve will return immediately. Returning true if there was an item on the queue and false if the queue was empty.