Main Function

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

#if outOfTheBox
 create_blinky_tasks();
 create_uart_task();
#else
 // LPC_IOCON->P0_30 //pull down resistor set bits 3 and 4 to 01
 adc_to_pwm_task_queue = xQueueCreate(1, sizeof(int));

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

#endif

 vTaskStartScheduler(); // This function never returns unless RTOS scheduler runs
out of memory and fails

 return 0;
}
```

Pwm Task

```
void pwm_task(void *p) {
 pwm1__init_single_edge(1000);

 // Locate a GPIO pin that a PWM channel will control
 // NOTE You can use gpio__construct_with_function() API from gpio.h

 gpio_s pin = gpio__construct_with_function(GPIO__PORT_2, /*Pin*/ 0,
GPIO__FUNCTION_1);

 // We only need to set PWM configuration once, and the HW will drive
 // the GPIO at 1000Hz, and control set its duty cycle to 50%
 pwm1__set_duty_cycle(PWM1__2_0, 50);

 // Continue to vary the duty cycle in the loop
 uint8_t percent = 0;
```

```
 int adc_reading = 0;
 while (1) {

   if (xQueueReceive(adc_to_pwm_task_queue, &adc_reading, 100)) {
     percent = (adc_reading * 100) / 0xfff;
     pwm1__set_duty_cycle(PWM1__2_0, percent);
   }
   //    vTaskDelay(100);
 }
}
```

ADC Task

```
void adc_task(void *p) {
 adc__initialize();

 // TODO This is the function you need to add to adc.h
 // You can configure burst mode for just the channel you are using
 adc__enable_burst_mode();

 // Configure a pin, such as P1.31 with FUNC 011 to route this pin as ADC channel 5
 // You can use gpio__construct_with_function() API from gpio.h
 gpio_s pin = gpio__construct_with_function(GPIO__PORT_0, /*Pin*/ 25,
GPIO__FUNCTION_1);
 LPC_IOCON->P0_25 &= ~(1 << 7);

 int adc_reading = 0; // Note that this 'adc_reading' is not the same variable as
the one from adc_task
 while (1) {
   // Get the ADC reading using a new routine you created to read an ADC burst
reading
   const uint16_t adc_value =
adc__get_channel_reading_with_burst_mode(ADC__CHANNEL_2);
   float adc_voltage = (float)adc_value / 4095 * 3.3;
   fprintf(stderr, "ADC Voltage: %f\n", adc_voltage);
   // Implement code to send potentiometer value on the queue
   // a) read ADC input to 'int adc_reading'
   adc_reading = adc_value;
   // b) Send to queue: xQueueSend(adc_to_pwm_task_queue, &adc_reading, 0);
   xQueueSend(adc_to_pwm_task_queue, &adc_reading, 0);
   vTaskDelay(100);
 }
}
```

Added prints to PWM1.c

```c
void pwm1__set_duty_cycle(pwm1_channel_e pwm1_channel, float duty_cycle_in_percent)
{
 const uint32_t mr0_reg_val = LPC_PWM1->MR0;
 const uint32_t match_reg_value = (mr0_reg_val * duty_cycle_in_percent) / 100;

 switch (pwm1_channel) {
 case PWM1__2_0:
   LPC_PWM1->MR1 = match_reg_value;
   fprintf(stderr, "MR0, and MR1\n");
   break;
 case PWM1__2_1:
   LPC_PWM1->MR2 = match_reg_value;
   fprintf(stderr, "MR0, and MR2\n");
   break;
 case PWM1__2_2:
   LPC_PWM1->MR3 = match_reg_value;
   fprintf(stderr, "MR0, and MR3\n");
   break;
 case PWM1__2_4:
   LPC_PWM1->MR5 = match_reg_value;
   fprintf(stderr, "MR0, and MR4\n");
   break;
 case PWM1__2_5:
   LPC_PWM1->MR6 = match_reg_value;
   fprintf(stderr, "MR0, and MR5\n");
   break;
 default:
   break;
 }

 LPC_PWM1->LER |= (1 << (pwm1_channel + 1)); ///< Enable Latch Register
}
```

New functions in adc.h

```c
void adc__enable_burst_mode(void);

uint16_t adc__get_channel_reading_with_burst_mode(adc_channel_e channel_num);
```

New functions in adc.c

```c
void adc__enable_burst_mode(void) {
 LPC_ADC->CR |= (1 << 16); // enable busts mode
```

```
 LPC_ADC->CR &= ~(0xff);  // clear all ADC Selections
 LPC_ADC->CR |= (1 << 2); // Select ADC 2

 LPC_ADC->CR &= ~(0x111 << 24); // set the start bits to 000
}

uint16_t adc__get_channel_reading_with_burst_mode(adc_channel_e channel_num) {
 uint16_t result = 0;
 const uint16_t twelve_bits = 0x0FFF;
 const uint32_t channel_masks = 0x7;
 const uint32_t adc_conversion_complete = (1 << 31);
 const uint32_t adc_overRun = (1 << 30);

 if ((ADC__CHANNEL_2 == channel_num) || (ADC__CHANNEL_4 == channel_num) ||
(ADC__CHANNEL_5 == channel_num)) {

   while (!(LPC_ADC->GDR & adc_conversion_complete)) { // Wait till conversion is
complete
     ;
   }

   result = (LPC_ADC->GDR >> 4) & twelve_bits; // 12bits - B15:B4
   //    fprintf(stderr, "Result: %d Channel: %d\n", result, ((LPC_ADC->GDR >> 24)
& channel_masks));
 }
 return result;
}
```