Nick Scheele

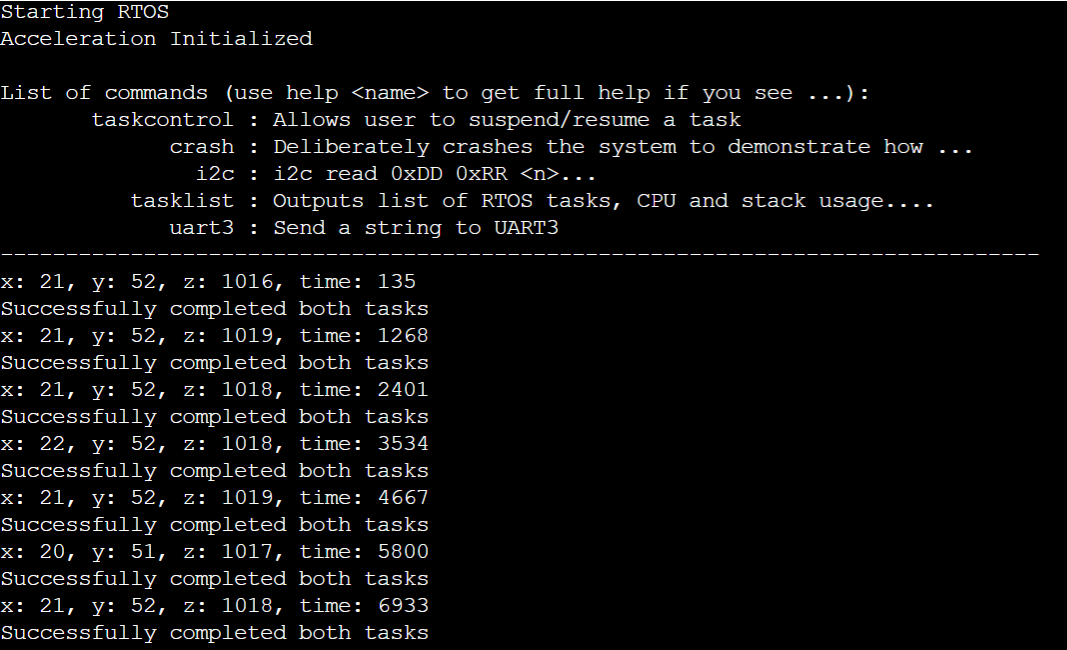
Prabjyot Obhi

Vivek Naveen

Watchdog Lab

Part 0 Code, Screenshot, and Explanation:

In part 0 of the lab, we had to create a producer and consumer task in which the data sent over the queue is the average value of 100 accelerometer readings. A function to get the average value was created, then, it was sent via the RTOS queue to be received by the consumer. Since the consumer has a portMAX\_DELAY in the xQueueReceive, it will wait until the queue has a value to be received. Furthermore, the values were then printed to a file on an SD Card. The SD Card files have been submitted separately.



|  |
| --- |
| main.c |
| #include "acceleration.h"  #include "event\_groups.h"  #include "ff.h"  #include <string.h>  int file\_count = 0;  void write\_file\_using\_fatfs\_pi(acceleration\_\_axis\_data\_s receive\_value) {  delay\_\_ms(2);  // file\_count++;  const char \*filename = "file\_plot.txt";  FIL file; // File handle  UINT bytes\_written = 0;  FRESULT result = f\_open(&file, filename, (FA\_WRITE | FA\_OPEN\_APPEND));  if (FR\_OK == result) {  char string[64];  // sprintf(string, "Value,%i\n", 123);  sprintf(string, "x: %d, y: %d, z: %d\n", receive\_value.x, receive\_value.y, receive\_value.z);  // fprintf(stderr, "printed\n");  if (FR\_OK == f\_write(&file, string, strlen(string), &bytes\_written)) {  } else {  printf("ERROR: Failed to write data to file\n");  }  f\_close(&file);  } else {  printf("ERROR: Failed to open: %s\n", filename);  }  }  static QueueHandle\_t watchdog\_queue;  static EventGroupHandle\_t xCreateGroup;  acceleration\_\_axis\_data\_s get\_send\_value() {  acceleration\_\_axis\_data\_s avg;  int xVal = 0;  int yVal = 0;  int zVal = 0;  for (int i = 0; i < 100; i++) {  xVal += acceleration\_\_get\_data().x;  yVal += acceleration\_\_get\_data().y;  zVal += acceleration\_\_get\_data().z;  delay\_\_ms(1);  }  avg.x = xVal / 100;  avg.y = yVal / 100;  avg.z = zVal / 100;  return avg;  }  void producer\_of\_sensor(void \*p) {  // fprintf(stderr, "producer\n");  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  acceleration\_\_axis\_data\_s send\_value = get\_send\_value();  xQueueSend(watchdog\_queue, &send\_value, 0);  xEventGroupSetBits(xCreateGroup, (1 << 1));  vTaskDelay(1000);  }  }  // TODO: Create this task at PRIORITY\_HIGH  void consumer\_of\_sensor(void \*p) {  acceleration\_\_axis\_data\_s receive\_value;  while (1) {  if (xQueueReceive(watchdog\_queue, &receive\_value, portMAX\_DELAY)) {  uint32\_t time = xTaskGetTickCount();  fprintf(stderr, "x: %d, y: %d, z: %d, time: %d\n", receive\_value.x, receive\_value.y, receive\_value.z, time);  write\_file\_using\_fatfs\_pi(receive\_value);  }  xEventGroupSetBits(xCreateGroup, (1 << 2));  }  } |

Part 1 Code and Explanation:

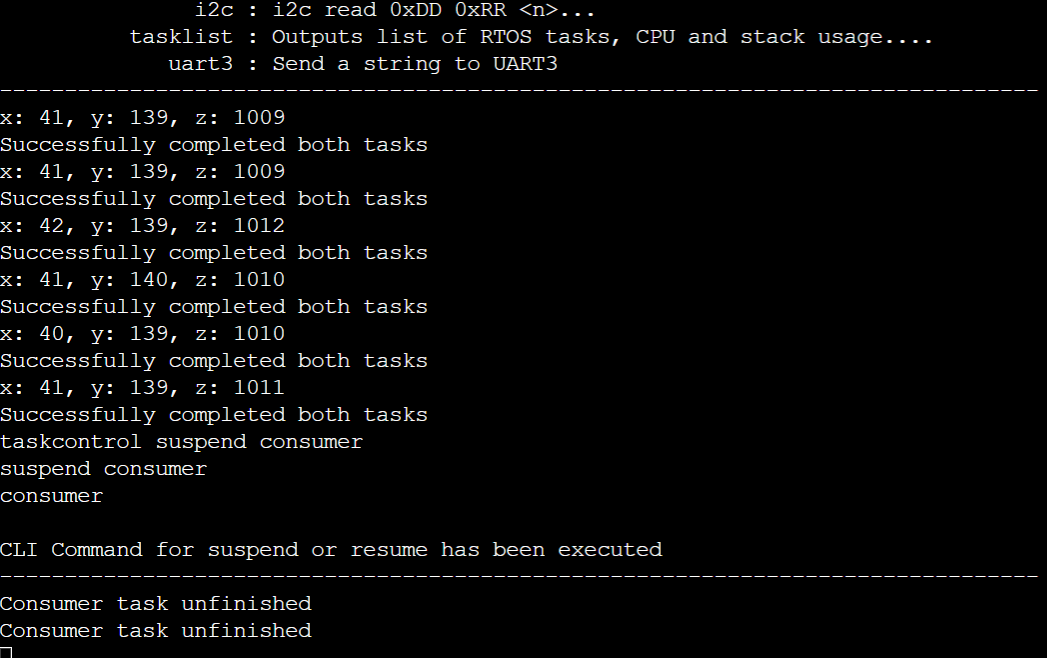
In part 1 of the lab, the software watchdog was implemented. Here, we use the Event Group API given to us by FreeRTOS to set bit 1 when the producer task finishes the while loop and set bit 2 when the consumer finishes the while loop. This way, we are able to check if both tasks finished, hence the “watchdog” watches over and monitors our application. Then, the file was printed to when either task did not finish with an error message. The error file is submitted separately.

|  |
| --- |
| main.c |
| #include "acceleration.h"  #include "event\_groups.h"  #include "ff.h"  #include <string.h>  void write\_file\_using\_fatfs\_pi\_error(int err) {  delay\_\_ms(2);  // file\_count++;  const char \*filename = "error.txt";  FIL file; // File handle  UINT bytes\_written = 0;  FRESULT result = f\_open(&file, filename, (FA\_WRITE | FA\_OPEN\_APPEND));  if (FR\_OK == result) {  char string[64];  // sprintf(string, "Value,%i\n", 123);  // sprintf(string, "x: %d, y: %d, z: %d\n", receive\_value.x, receive\_value.y, receive\_value.z);  if (err == 1) {  sprintf(string, "Both tasks didn't finish\n");  } else if (err == 2) {  sprintf(string, "Consumer task didn't finish\n");  }  // fprintf(stderr, "printed\n");  if (FR\_OK == f\_write(&file, string, strlen(string), &bytes\_written)) {  } else {  printf("ERROR: Failed to write data to file\n");  }  f\_close(&file);  } else {  printf("ERROR: Failed to open: %s\n", filename);  }  }  void watchdog\_task(void \*params) {  while (1) {  vTaskDelay(1000);  EventBits\_t check = xEventGroupWaitBits(xCreateGroup, (1 << 1) | (1 << 2), pdTRUE, pdFALSE, 200);  if (((check & (1 << 1)) != 0) && ((check & (1 << 2)) != 0)) {  fprintf(stderr, "Successfully completed both tasks\n");  } else {  // fprintf(stderr, "Producer task unfinished\n");  int err;  if (check & (1 << 1)) {  fprintf(stderr, "Consumer task unfinished\n");  err = 1;  write\_file\_using\_fatfs\_pi\_error(err);  } else if (check & (1 << 2)) {  fprintf(stderr, "Producer task unfinished\n");  err = 2;  write\_file\_using\_fatfs\_pi\_error(err);  } else {  fprintf(stderr, "Both tasks didnt finish\n");  }  }  }  } |

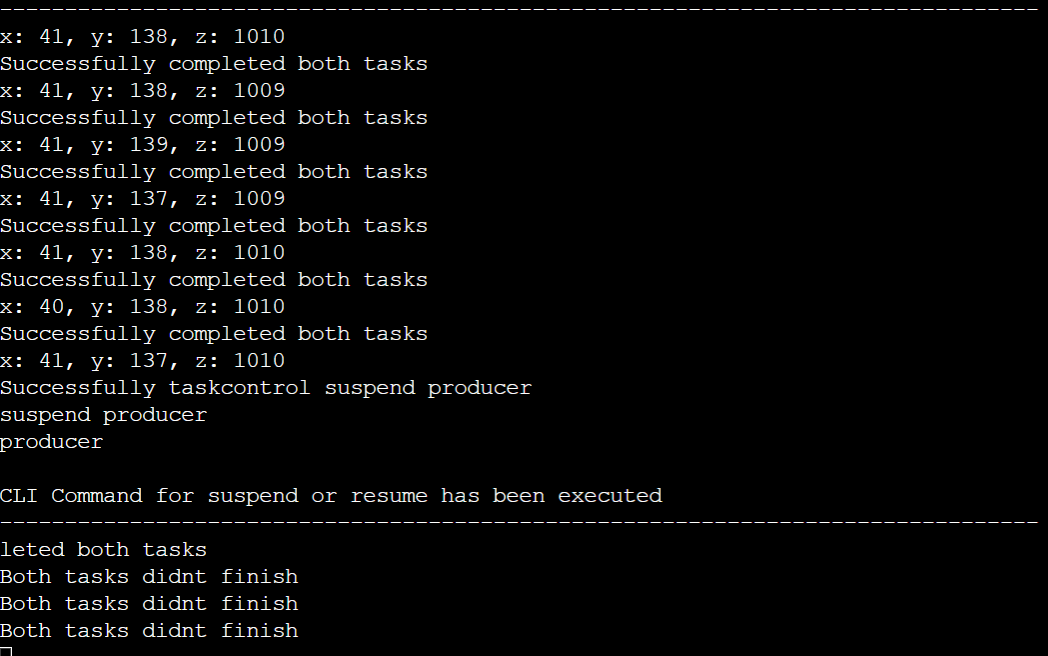
Part 2 Code, Screenshots, and Explanations:

In part 2 of the lab, we used CLI commands that were written to suspend and resume tasks. These CLI commands were reused from last lab. When a consumer task is suspended, we expect for the bit at the end of the loop not to be set, and when this happens, we expect our error message to display on both the sd card as well as the telemetry output. When the consumer task is suspended, we get the message that the consumer task did not finish. Then, when the producer task is suspended, we get an error message that both tasks did not complete. This is because when the producer task is suspended, the consumer task will sleep forever as it has a portMAX\_DELAY. Below these screenshots, we see the accelerometer values plotted on a line graph. The x-axis for this graph is the time, which found using the xTaskGetTickCount API in FreeRTOS. We can see from the plot that the accelerometers x-value is in a sinusoidal wave form, and this is because the board was flipped by the horizontal axis when taking data. The y and z values are relatively stable, and this is because the board was only flipped on the horizontal axis. The full source code is shown below.

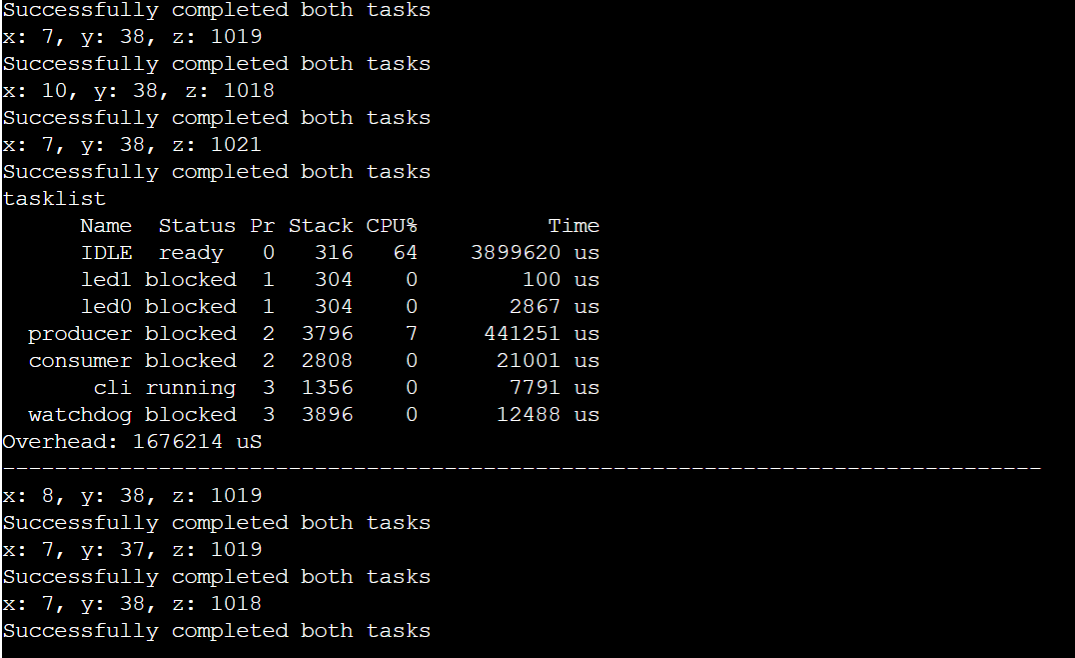
|  |
| --- |
| main.c |
| #include <stdio.h>  #include "FreeRTOS.h"  #include "task.h"  #include "board\_io.h"  #include "common\_macros.h"  #include "periodic\_scheduler.h"  #include "queue.h"  #include "sj2\_cli.h"  #include "acceleration.h"  #include "event\_groups.h"  #include "ff.h"  #include <string.h>  int file\_count = 0;  void write\_file\_using\_fatfs\_pi(acceleration\_\_axis\_data\_s receive\_value) {  delay\_\_ms(2);  // file\_count++;  const char \*filename = "file\_plot.txt";  FIL file; // File handle  UINT bytes\_written = 0;  FRESULT result = f\_open(&file, filename, (FA\_WRITE | FA\_OPEN\_APPEND));  if (FR\_OK == result) {  char string[64];  // sprintf(string, "Value,%i\n", 123);  sprintf(string, "x: %d, y: %d, z: %d\n", receive\_value.x, receive\_value.y, receive\_value.z);  // fprintf(stderr, "printed\n");  if (FR\_OK == f\_write(&file, string, strlen(string), &bytes\_written)) {  } else {  printf("ERROR: Failed to write data to file\n");  }  f\_close(&file);  } else {  printf("ERROR: Failed to open: %s\n", filename);  }  }  void write\_file\_using\_fatfs\_pi\_error(int err) {  delay\_\_ms(2);  // file\_count++;  const char \*filename = "error.txt";  FIL file; // File handle  UINT bytes\_written = 0;  FRESULT result = f\_open(&file, filename, (FA\_WRITE | FA\_OPEN\_APPEND));  if (FR\_OK == result) {  char string[64];  // sprintf(string, "Value,%i\n", 123);  // sprintf(string, "x: %d, y: %d, z: %d\n", receive\_value.x, receive\_value.y, receive\_value.z);  if (err == 1) {  sprintf(string, "Both tasks didn't finish\n");  } else if (err == 2) {  sprintf(string, "Consumer task didn't finish\n");  }  // fprintf(stderr, "printed\n");  if (FR\_OK == f\_write(&file, string, strlen(string), &bytes\_written)) {  } else {  printf("ERROR: Failed to write data to file\n");  }  f\_close(&file);  } else {  printf("ERROR: Failed to open: %s\n", filename);  }  }  static QueueHandle\_t watchdog\_queue;  static EventGroupHandle\_t xCreateGroup;  acceleration\_\_axis\_data\_s get\_send\_value() {  acceleration\_\_axis\_data\_s avg;  int xVal = 0;  int yVal = 0;  int zVal = 0;  for (int i = 0; i < 100; i++) {  xVal += acceleration\_\_get\_data().x;  yVal += acceleration\_\_get\_data().y;  zVal += acceleration\_\_get\_data().z;  delay\_\_ms(1);  }  avg.x = xVal / 100;  avg.y = yVal / 100;  avg.z = zVal / 100;  return avg;  }  void producer\_of\_sensor(void \*p) {  // fprintf(stderr, "producer\n");  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  acceleration\_\_axis\_data\_s send\_value = get\_send\_value();  // TODO: Print a message before xQueueSend()  // Note: Use printf() and not fprintf(stderr, ...) because stderr is a polling printf  // fprintf(stderr, "before xqueuesend\n");  // printf("send");  xQueueSend(watchdog\_queue, &send\_value, 0);  // TODO: Print a message after xQueueSend()  // fprintf(stderr, "after xqueuesend\n");  // fprintf(stderr, "------------------producer done--------------");  xEventGroupSetBits(xCreateGroup, (1 << 1));  vTaskDelay(1000);  }  }  // TODO: Create this task at PRIORITY\_HIGH  void consumer\_of\_sensor(void \*p) {  acceleration\_\_axis\_data\_s receive\_value;  while (1) {  // TODO: Print a message before xQueueReceive()  // fprintf(stderr, "before xqueuereceive\n");  if (xQueueReceive(watchdog\_queue, &receive\_value, portMAX\_DELAY)) {  uint32\_t time = xTaskGetTickCount();  fprintf(stderr, "x: %d, y: %d, z: %d, time: %d\n", receive\_value.x, receive\_value.y, receive\_value.z, time);  write\_file\_using\_fatfs\_pi(receive\_value);  }  xEventGroupSetBits(xCreateGroup, (1 << 2));  }  }  void watchdog\_task(void \*params) {  while (1) {  vTaskDelay(1000);  // if (xEventGroupWaitBits(xCreateGroup, (1 << 1), pdTRUE, pdFALSE, 100) &&  // xEventGroupWaitBits(xCreateGroup, (1 << 2), pdTRUE, pdFALSE, 100)) {  // fprintf(stderr, "Successfully completed both tasks\n");  // } else {  // if (!xEventGroupWaitBits(xCreateGroup, (1 << 1), pdTRUE, pdFALSE, 0)) {  // fprintf(stderr, "Producer task unfinished\n");  // }  // if (!xEventGroupWaitBits(xCreateGroup, (1 << 2), pdTRUE, pdFALSE, 0)) {  // fprintf(stderr, "Consumer task unfinished\n");  // }  // }  // if (xEventGroupWaitBits(xCreateGroup, (1 << 1) | (1 << 2), pdTRUE, pdFALSE, 200)) {  // fprintf(stderr, "Successfully completed both tasks\n");  // } else {  // // fprintf(stderr, "Producer task unfinished\n");  // if (xEventGroupWaitBits(xCreateGroup, (1 << 1), pdTRUE, pdFALSE, 0)) {  // fprintf(stderr, "Consumer task unfinished\n");  // } else if (xEventGroupWaitBits(xCreateGroup, (1 << 2), pdTRUE, pdFALSE, 0)) {  // fprintf(stderr, "Producer task unfinished\n");  // } else {  // fprintf(stderr, "Both tasks didnt finish\n");  // }  // }  EventBits\_t check = xEventGroupWaitBits(xCreateGroup, (1 << 1) | (1 << 2), pdTRUE, pdFALSE, 200);  if (((check & (1 << 1)) != 0) && ((check & (1 << 2)) != 0)) {  fprintf(stderr, "Successfully completed both tasks\n");  } else {  // fprintf(stderr, "Producer task unfinished\n");  int err;  if (check & (1 << 1)) {  fprintf(stderr, "Consumer task unfinished\n");  err = 1;  write\_file\_using\_fatfs\_pi\_error(err);  } else if (check & (1 << 2)) {  fprintf(stderr, "Producer task unfinished\n");  err = 2;  write\_file\_using\_fatfs\_pi\_error(err);  } else {  fprintf(stderr, "Both tasks didnt finish\n");  }  }  }  }  void watchdog\_main() {  if (acceleration\_\_init()) {  fprintf(stderr, "Acceleration Initialized\n");  }  TaskHandle\_t prod\_watchdog;  TaskHandle\_t cons\_watchdog;  TaskHandle\_t watchdog;  xTaskCreate(producer\_of\_sensor, "producer", 1024, NULL, 2, &prod\_watchdog);  xTaskCreate(consumer\_of\_sensor, "consumer", 1024, NULL, 2, &cons\_watchdog);  xTaskCreate(watchdog\_task, "watchdog", 1024, NULL, 3, &watchdog);  // TODO Queue handle is not valid until you create it  watchdog\_queue = xQueueCreate(  1, sizeof(acceleration\_\_axis\_data\_s)); // Choose depth of item being our enum (1 should be okay for this example  xCreateGroup = xEventGroupCreate();  }  int main(void) {  create\_blinky\_tasks();  create\_uart\_task();  // If you have the ESP32 wifi module soldered on the board, you can try uncommenting this code  // See esp32/README.md for more details  // uart3\_init(); // Also include: uart3\_init.h  // xTaskCreate(esp32\_tcp\_hello\_world\_task, "uart3", 1000, NULL, PRIORITY\_LOW, NULL); // Include esp32\_task.h  puts("Starting RTOS");  // producer\_consumer\_assignment();  watchdog\_main();  vTaskStartScheduler(); // This function never returns unless RTOS scheduler runs out of memory and fails  return 0;  } |

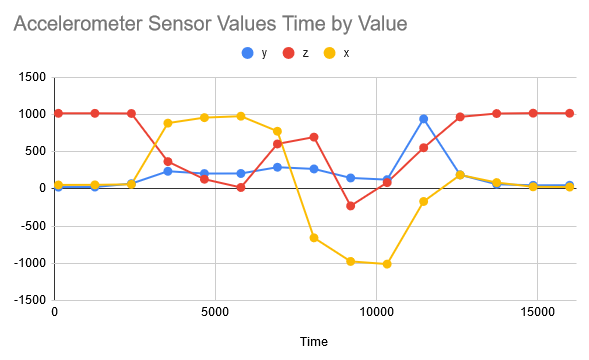


Consumer task is suspended



Producer task is suspended





Plotted values for accelerometer