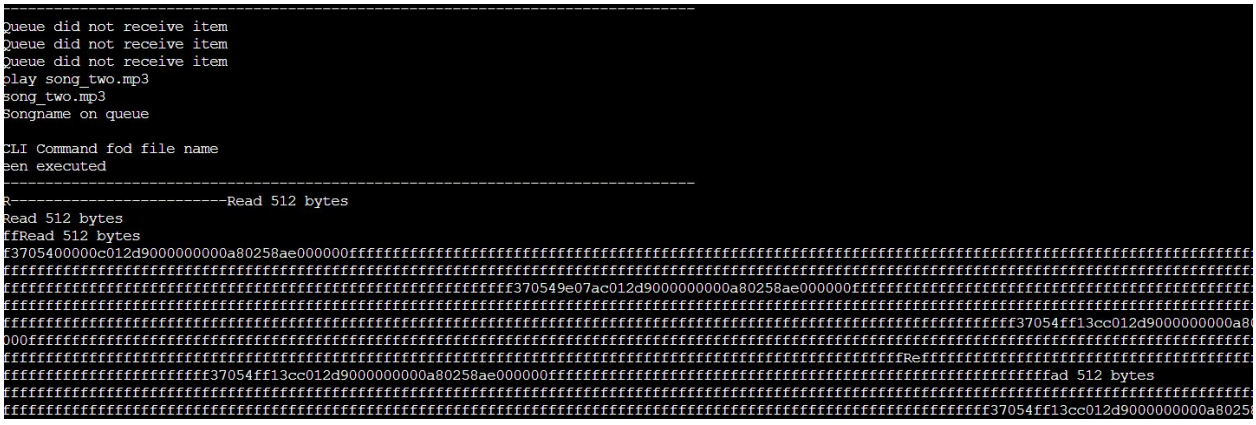
Nick Scheele
Prabjyot Obhi
Vivek Naveen

Milestone 1 MP3

Code, Explanations, and Screenshots:

For milestone 1, we built the structure to read mp3 file type data from the SD card. A simple mp3 file was placed on the mp3 card and then the bytes were read and printed on telemetry. The design uses 2 RTOS queues, and a CLI command sends the song name to one queue. The queue uses the extern keyword in order to be recognized in other files such as where the CLI is processed. Then, the song name, or the file name, is used to read the data. That is read 512 bytes at a time and sent over another queue to be printed on the screen. The queue size is 2 in order to reduce the waiting time between the data read and the print. The file data is printed in hex values shown below. There are two tasks that are used here. One reads the data from the MP3, and the other (player task), prints the data. The FATFS library is used to read the file from the SD card. The song name is also limited to 32 bytes.



CLI command with song name and hex values printed

Hex values read from MP3 file

main.c

```
extern QueueHandle_t song_name_queue;
static QueueHandle_t mp3_file_queue;

typedef char songname_t[32];
typedef char song_data_t[512];

static void read_file(const char *filename) {
  puts("Read and stored file name");
  FIL file;
  UINT bytes_written = 0;
  FRESULT result = f_open(&file, filename, (FA_READ | FA_OPEN_EXISTING));
  if (FR_OK == result) {
          song_data_t buffer = {};
          UINT bytes_to_read = 512;
          UINT bytes_done_reading = 1;
          while (bytes_done_reading > 0) {
          FRESULT rd = f_read(&file, buffer, bytes_to_read, &bytes_done_reading);
          xQueueSend(mp3_file_queue, buffer, portMAX_DELAY);
          if (FR_OK == rd) {
          printf("Read %d bytes\n", bytes_done_reading);
          }
          }

          f_close(&file); // not sure when to close file
  } else {
          puts("Unavailable song");
  }
}
static void mp3_file_reader_task(void *p) {
  songname_t s_name = {};
  while (1) {
          if (xQueueReceive(song_name_queue, &s_name, 3000)) {
          read_file(s_name);
          } else {
          puts("Queue did not receive item");
          }
  }
}

static void mp3_decoder_send_block(song_data_t s_data) {
  for (size_t index = 0; index < sizeof(song_data_t); index++) {
          vTaskDelay(3);
          putchar(s_data[index]);
  }
}

static void print_hex(song_data_t s_data) {
  for (size_t index = 0; index < sizeof(song_data_t); index++) {
          vTaskDelay(1);
          printf("%02x", s_data[index]);
```

```
  }
}

static void mp3_data_player_task(void *p) {
  song_data_t s_data = {};
  while (1) {
          memset(&s_data[0], 0, sizeof(song_data_t));
          if (xQueueReceive(mp3_file_queue, &s_data[0], portMAX_DELAY)) {
          // mp3_decoder_send_block(s_data);
          print_hex(s_data);
          }
  }
}
void milestone_1_main() {
  song_name_queue = xQueueCreate(1, sizeof(songname_t));
  mp3_file_queue = xQueueCreate(2, sizeof(song_data_t));
  // TaskHandle_t get_name;
  xTaskCreate(mp3_file_reader_task, "reader", 512, NULL, PRIORITY_MEDIUM, NULL);
  xTaskCreate(mp3_data_player_task, "player", 512, NULL, PRIORITY_HIGH, NULL);
  // xTaskCreate(get_song_name_task, "Get Song Name", 1, NULL, PRIORITY_MEDIUM,
&get_name);
}

int main(void) {
  create_blinky_tasks();
  create_uart_task();
  milestone_1_main();
  puts("Starting RTOS");

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

  return 0;
}
```

handlers_general.c

```
QueueHandle_t song_name_queue;
typedef char songname_t[32];

app_cli_status_e cli__play(app_cli__argument_t argument, sl_string_t
user_input_minus_command_name,
                  app_cli__print_string_function cli_output) {
  // sl_string is a powerful string library, and you can utilize the sl_string.h API to parse
parameters of a command

  // Sample code to output data back to the CLI
  sl_string_t s = user_input_minus_command_name; // Re-use a string to save memory
```

```
  char sch = 's';
  printf("\n");
  songname_t s_name = {0};
  sl_string__copy_to(s, s_name, sizeof(s_name) - 1);
  if (xQueueSend(song_name_queue, &s_name, 0)) {
          puts("Songname on queue");
  } else {
          puts("Songname failed to queue");
  }
  printf("\n");
  sl_string__printf(s, "CLI Command for play has been executed\n");
  cli_output(NULL, s);

  return APP_CLI_STATUS__SUCCESS;
}
```