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

```
2971296c5084e9995ffead3f7322d0c98cal10199353243fffffff26cdc779143c440c4d12fa66eafffffffff6a0596c340fffffdfab0586c39fff0c4b078cdffffdafcbde1f60f7bbfff16792c5963730491fffeefff37074da184603288cc265009388f5e56a599c000a33f65e717888002e0703fffffba8c462f36632855221a20ffb2cf64918349029e9971bb15440d70ab5920eadfb2f00280ce2f49a951ad504ae01ad3c02b8539367b05504df1fa18a443226d96d2e6129e441345a25ebaf465773d59fd7188bf18f6f7f3d77881abeffc53ffad5fd753e3fbd3eb74ce717faf3d3ff8aef39de3713713e6fbfbd623e22409be337de77e1e2f7bfdffbc6b1129bcde44f3d6f5b66d1ff370748818160762dbcc3c608c98b6c5ff896000f1ab4d5ce6dba6b3fd33bf4f5f8d533f34f7bdf7acd71abe35Read 512 bytes
ef7afff17bd713fbfaf89ad6359cef37ce2f2496bcb58cbacc5cead7dd2205fb026143aaec75100000024ca89ad106d8175f350be5c8198ecc7cc6c46ea32a0511a08f0e4390c155eabef079
e9c35730b870016ea3dad4960000059a5080d8d842a8190da9da76e70f1585d8012925ce2e6095b50bab7362218af990b3d3a44be913a88302330fff370743610e49d672acc3c0000008b8bace
038c600166d8a335bea37ff1bc4583f5b8f4f6a38479adf15be18bdf5f7d7610163f76611b402fb6917ec50fe308898002b1603f0d897030d320a0ca8fbd655a94b9e918880059994169d1a9
1f87dc2286ed12b98800c35ad2322d4004955555536f252884ac696228f83fa91a99303c657aa47640de28d4cd8b1fe3f08d45b4ffb5595C63e5fa6d669ea350055910009415242e3d4288e3
ea55f6b94da5683054a80997fff37074190d39ded71fcc39401888916e173c2100071ee73d1566e5a910da830ed15729435061553deb898137475db3d3d74adf2ad5283a88bb1e9857f9456c71
014222552ceb5b4cea62084da68ab95f2b56af2b7ccfadbcb4a31cedbe637b4cff4528b7e8868ae2004029426608ca2118333826a55136fabd6b3e56f93e94a4820d7f9e9196f65ab7110019
14422552ceb5b4cea62084da68ab95f2b56af2b7ccfadbcb4a31cedbe637b4cff4528b7e8868b8204749345357115571dc6466919e0e2603e28468e81341312f3f6b692233d2e9736c35
caf325a9754fc43bb0ee703550b2bb3b747750e2f0b79774f590f5a7dc4e4df66698a5d8bd0217b83487366351557110c194666919e0e2603e28468e81341312f3f6b692233d2e9736c35
caf325a9754fc43bb0ee703550b2bb3b747750e2f0b79774f590f5a7dc4e4df66698a5d8bd0217b83487366351557110c194666176e2920396286488159788b18f99dffb331863aa
394c2b9e823672a94aff64364b2ed7ced6b892
```

Hex values read from MP3 file

```
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 ts 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

```
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;
}
```