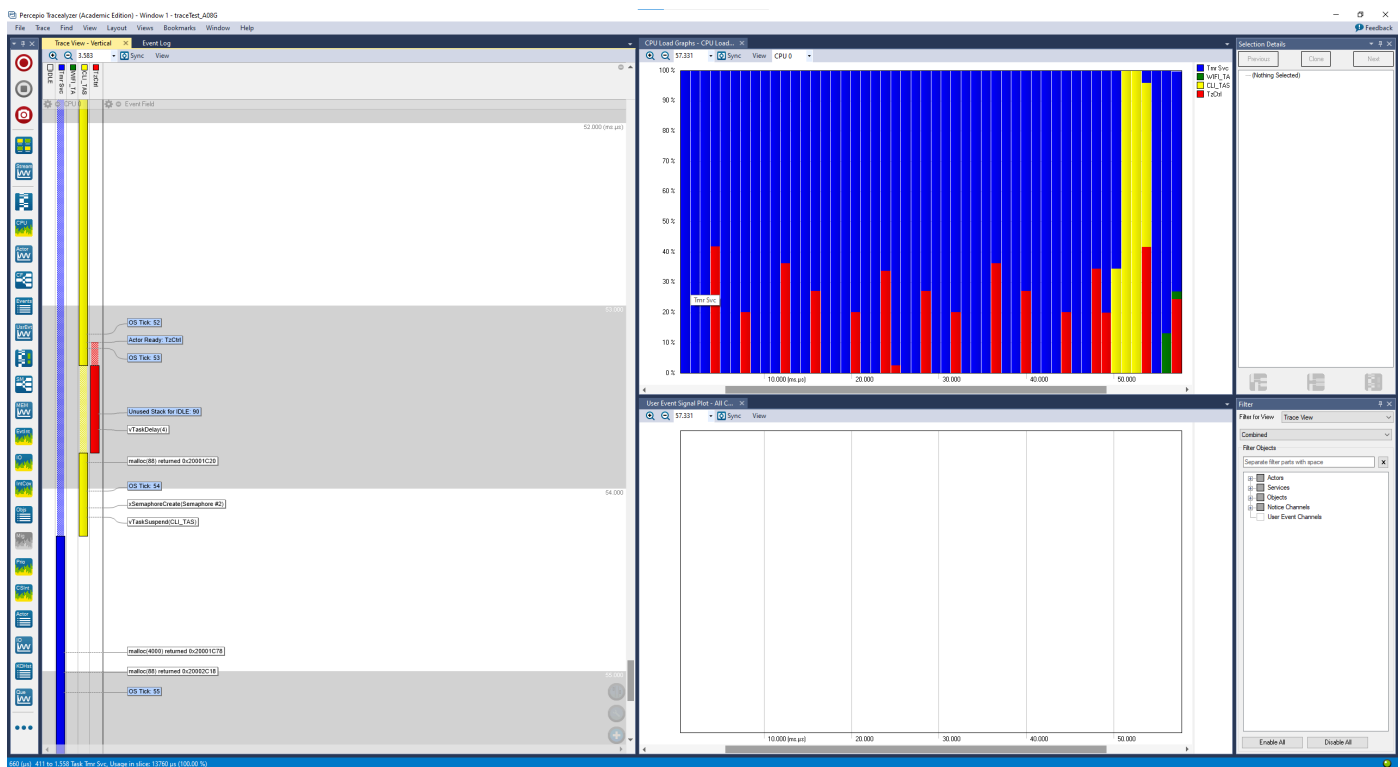


- 1. Using Percepio
- 2. Capture SD Card Comms
- 3. Bootloader Design
- 4. Bootloader Implementation
- 5. CRC checks
- Video Implementation Link
- <https://drive.google.com/file/d/1PkBrn4j8JweuwwwCsMoRXW76o6jn-LaF/view?usp=sharing>

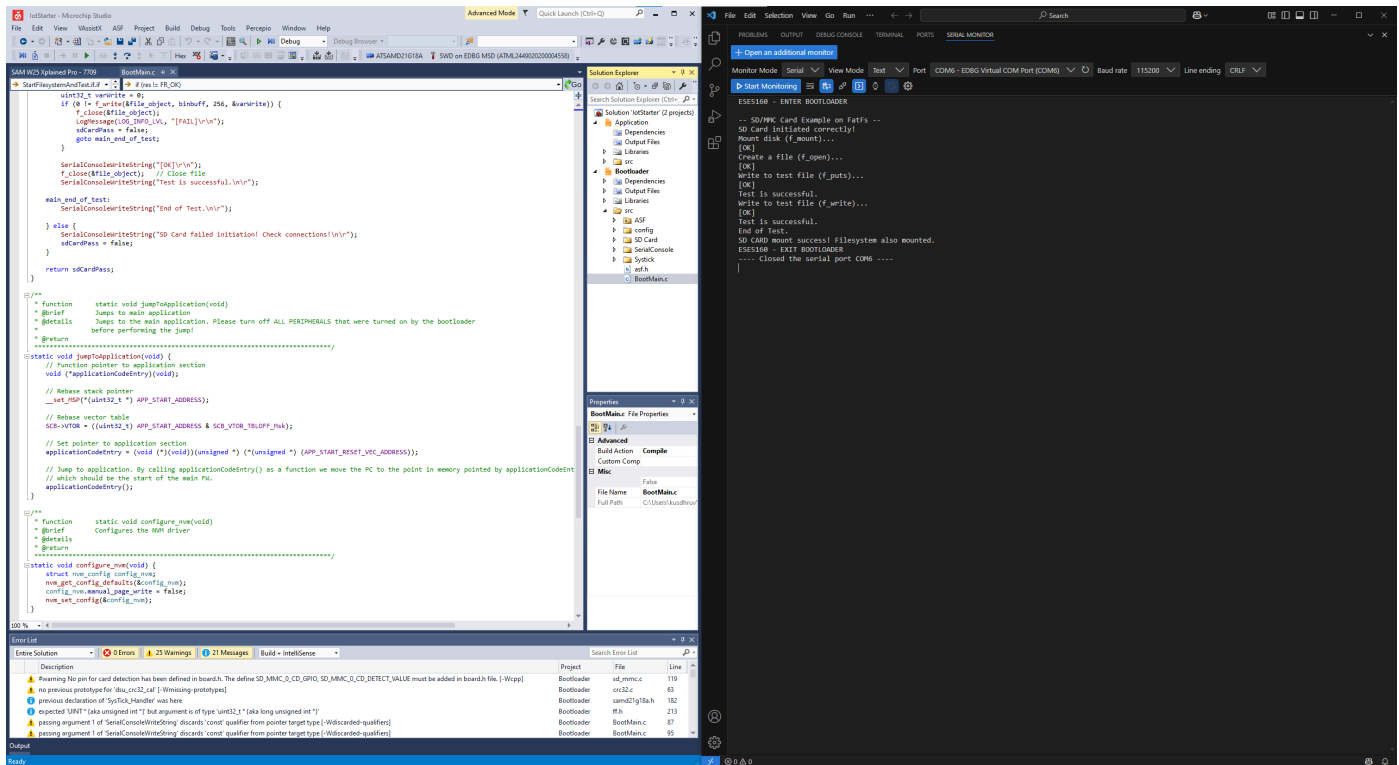
1. Using Percepio

Submit a screenshot of your Percepio trace in your README file.

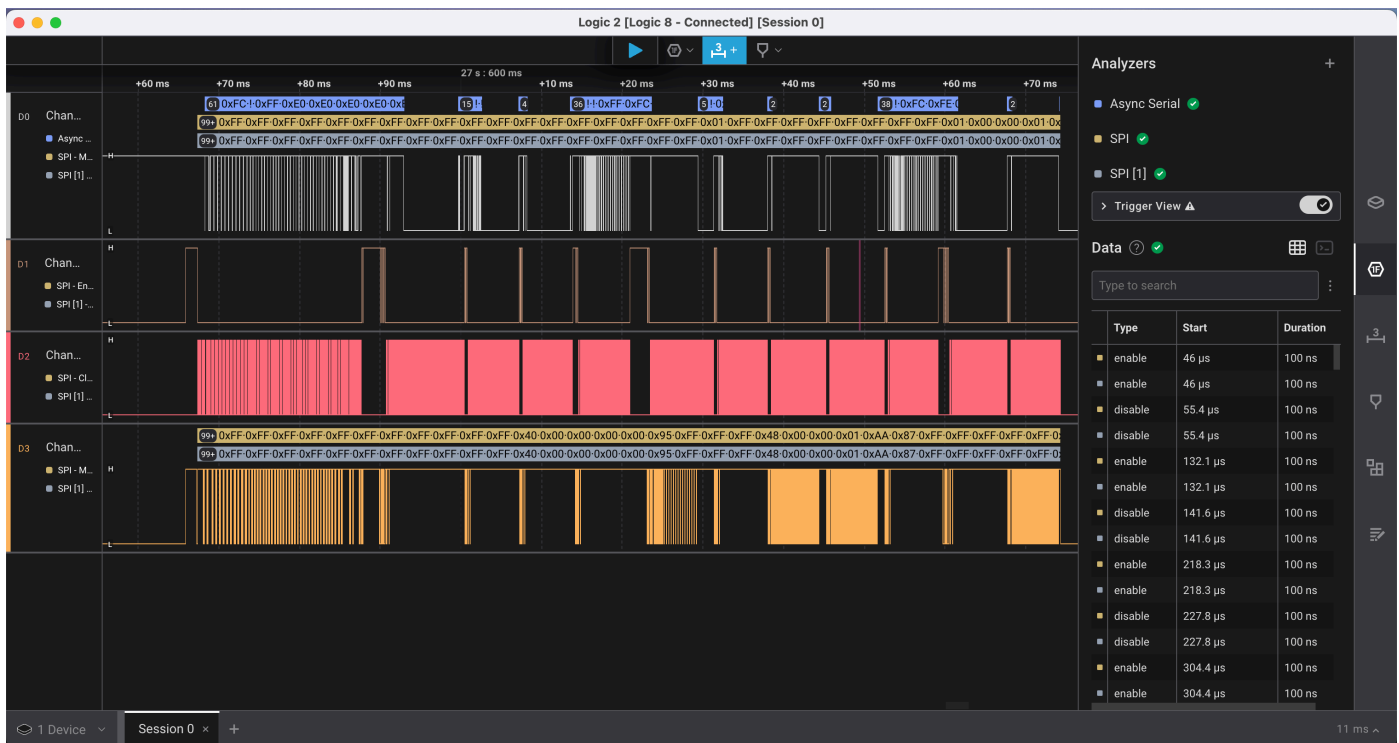


2. Capture SD Card Comms

Submit a photo of your hardware connections between the SAMW25 Xplained dev board, the SD card module, and the logic analyzer.



Submit a screenshot of the decoded message, as well as a *small* capture file of the SD card communication.



3. Bootloader Design

Update your README.md describing how your bootloader will work.

MCU Reset and Bootloader Entry: When the microcontroller resets, the hardware automatically loads the starting values, its stack pointer and the reset vector. Since the

bootloader is stored at the very beginning of the memory (address 0x0), the processor immediately starts running the bootloader code.

Checking the Boot Status Flag: The bootloader first looks for a special flag which is stored in non-volatile memory or set via an external trigger like a button press, to decide if there's a new firmware update waiting. This flag helps determine whether the device should update its code or simply run the existing application.

Decision: Update or Validate Firmware

If an update is needed: The bootloader proceeds with the update process. It begins by downloading the new firmware from an external source (like an SD card). Because the MCU has limited RAM, it reads this firmware in small chunks rather than all at once.

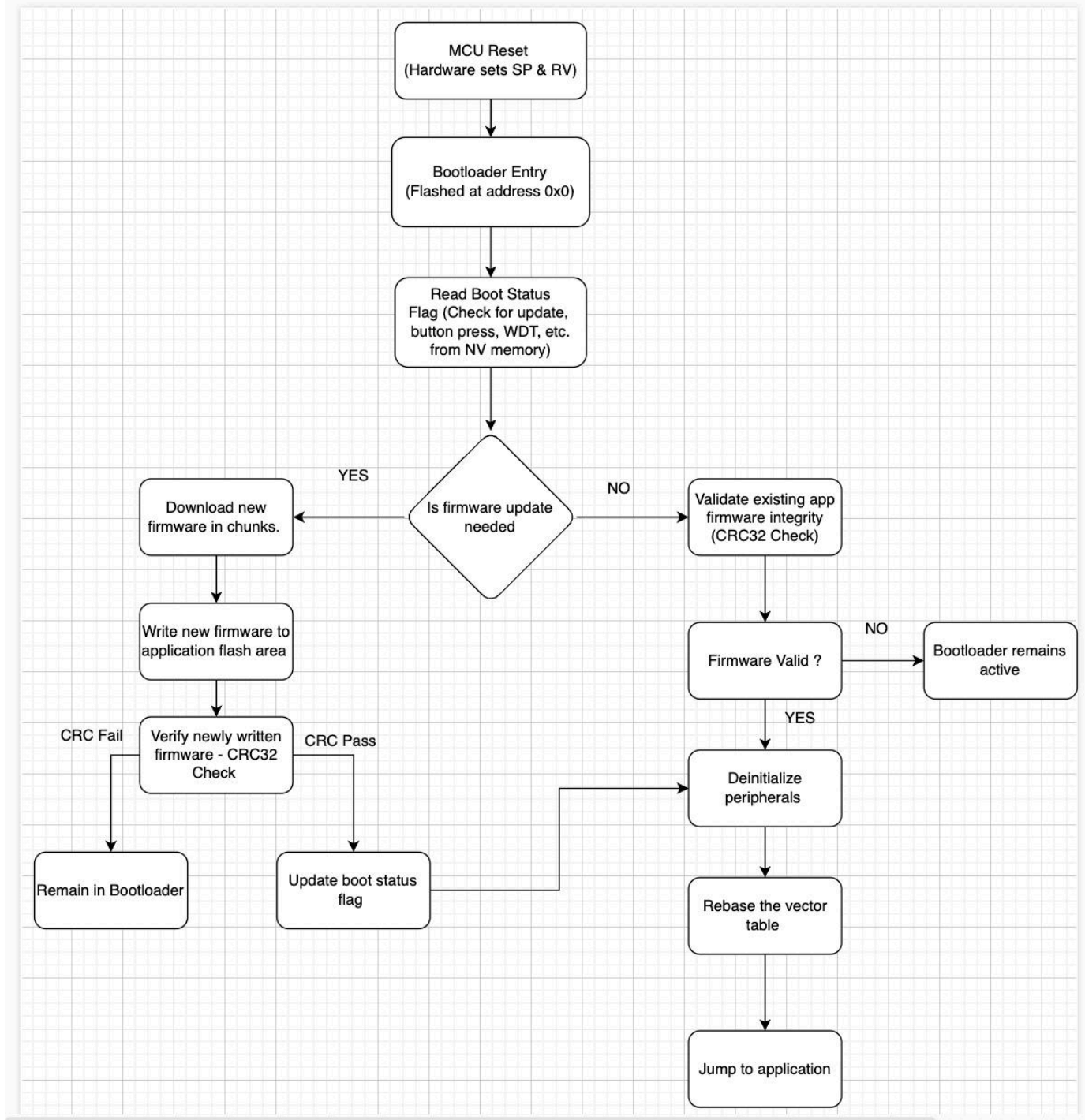
If no update is needed: The bootloader validates the existing application firmware by performing a CRC32 checksum to ensure the code hasn't been corrupted.

Handling the Firmware Update Process In the update process, once the new firmware is downloaded in chunks, the bootloader writes it into the flash memory region reserved for the application. This region is set to start at a specific offset (for example, 0x12000), which prevents it from overwriting the bootloader itself. After writing, another CRC32 check is performed on the new firmware to confirm its integrity. If everything checks out, the bootloader updates the status flag to mark the new firmware as valid.

Preparing for Application Launch Whether you're running an update or using the already-valid application, the next steps are similar. The bootloader first deinitializes any peripherals (like sensors, communication modules, etc.) that were activated during the bootloader phase. This ensures the application starts in a clean, predictable state.

Rebasing the Vector Table and Jumping to the Application Finally, the bootloader resets the system's vector table so that it points to the application's reset handler in the new memory area (starting at 0x12000). Then, it transfers control from the bootloader to the application, allowing the device to start executing its main functions.

Include a flow chart of your bootloader implementation in your repository README.



4. Bootloader Implementation

Added the updated code to the GitHub

5. CRC checks

Submit your well-commented firmware to the GitHub classroom assignment.

Updated Code in Gitub

```
// STEP 1: FLAG CHECK – insert here!
char flag_file[16] = "";
char bin_file[16] = "";
uint32_t expected_crc = 0xFFFFFFFF;

resA = f_open(&file_object, "0:TestA.bin", FA_READ);

// Check for flag
if (f_open(&file_object, "0:FlagA.txt", FA_READ) == FR_OK) {
    strcpy(bin_file, "0:TestA.bin");
    expected_crc = 0x1EF640AF;
    f_close(&file_object);
    f_unlink("0:FlagA.txt"); // Delete after reading
} else if (f_open(&file_object, "0:FlagB.txt", FA_READ) == FR_OK) {
    strcpy(bin_file, "0:TestB.bin");
    expected_crc = 0xFFECCCF0;
    f_close(&file_object);
    f_unlink("0:FlagB.txt");
} else {
    SerialConsoleWriteString("No flag found. Booting current app...\r\n");
    goto EXIT_BOOTLOADER;
}
```

```

// STEP 2: LOAD .BIN FILE INTO FLASH
FIL firmware_file;
UINT bytes_read;
uint8_t buffer[64]; // Flash write block = 64 bytes
uint32_t flash_address = APP_START_ADDRESS;

// Open the firmware binary file from SD
res = f_open(&firmware_file, bin_file, FA_READ);
if (res != FR_OK) {
    SerialConsoleWriteString("ERROR: Couldn't open .bin file from SD card.\r\n");
    goto EXIT_BOOTLOADER;
}

// Read from SD and write to flash
while (1) {
    res = f_read(&firmware_file, buffer, sizeof(buffer), &bytes_read);
    if (res != FR_OK || bytes_read == 0) break;

    // Erase 256-byte row every 4 writes (64 * 4 = 256)
    if ((flash_address % 256) == 0) {
        nvm_erase_row(flash_address);
    }

    // Write 64 bytes to flash
    nvm_write_buffer(flash_address, buffer, bytes_read);
    flash_address += bytes_read;
}

// f_close(&firmware_file);
// SerialConsoleWriteString("Firmware loaded into flash.\r\n");

#define APP_FLASH_SIZE 0x8000 // Adjust if your app is smaller/larger than 32KB

// Debug Change Test
// *****/
// Before closing, get actual .bin size from firmware_file
uint32_t firmware_size = f_size(&firmware_file);
f_close(&firmware_file);
SerialConsoleWriteString("Firmware loaded into flash.\r\n");
// *****/

```

```
// STEP 3: VERIFY CRC
```

```
uint32_t raw_crc = 0xFFFFFFFF;  
dsu_crc32_cal(APP_START_ADDRESS, firmware_size, &raw_crc);  
uint32_t calculated_crc = ~raw_crc;
```

```
char crc_log[64];  
sprintf(crc_log, "Calculated CRC: 0x%08lX | Expected: 0x%08lX\r\n", calculated_crc, expected_crc);  
SerialConsoleWriteString(crc_log);
```

```
if (calculated_crc != expected_crc) {  
    SerialConsoleWriteString("CRC MISMATCH! Halting boot.\r\n");  
    while (1); // Stop here if corrupted  
} else {  
    SerialConsoleWriteString("CRC Verified! Firmware is good.\r\n");  
}
```

```
/*END SIMPLE SD CARD MOUNTING AND TEST!*/
```

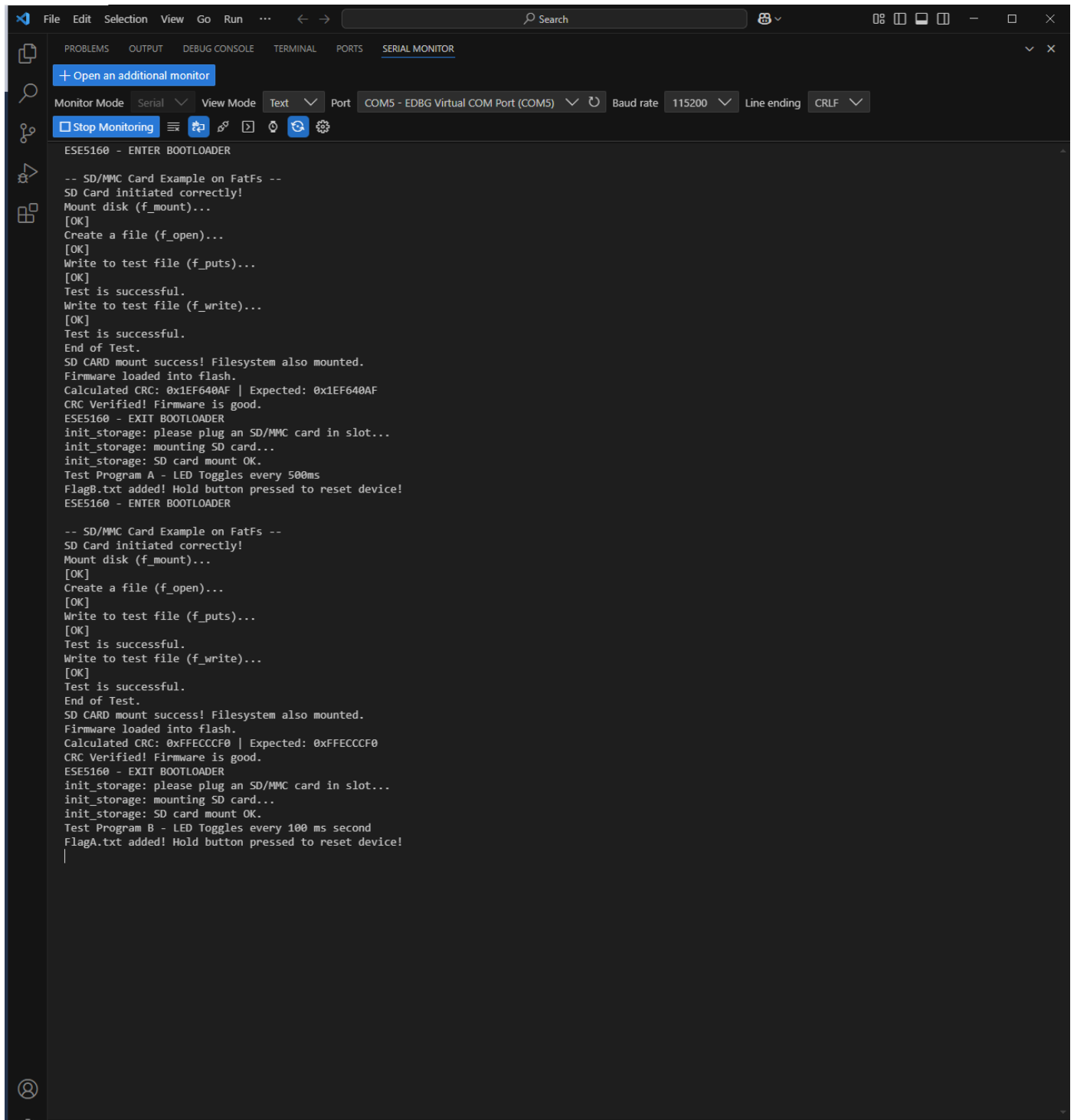
```
/*3.) STARTS BOOTLOADER HERE!*/
```

```
// Students - this is your mission!
```

```
/* END BOOTLOADER HERE!*/
```

```
EXIT_BOOTLOADER:
```

Serial Monitor Implementation Proof



The screenshot shows the Espressif IDE Serial Monitor window. The top bar includes tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, and SERIAL MONITOR. The SERIAL MONITOR tab is active, displaying a blue button to '+ Open an additional monitor'. Below this, the 'Monitor Mode' is set to 'Serial', 'View Mode' to 'Text', and the 'Port' is 'COM5 - EDBG Virtual COM Port (COM5)'. The 'Baud rate' is '115200' and 'Line ending' is 'CRLF'. A 'Stop Monitoring' button is visible. The main area shows the serial output for 'ESE5160 - ENTER BOOTLOADER'. The log includes messages about SD/MMC card initialization, file operations (f_open, f_puts, f_write), CRC verification, and firmware loading. It also shows the device entering the bootloader and then exiting to run 'Test Program A' and 'Test Program B'. The log ends with 'FlagB.txt added! Hold button pressed to reset device!' and 'ESE5160 - ENTER BOOTLOADER'.

```
ESE5160 - ENTER BOOTLOADER

-- SD/MMC Card Example on FatFs --
SD Card initiated correctly!
Mount disk (f_mount)...
[OK]
Create a file (f_open)...
[OK]
Write to test file (f_puts)...
[OK]
Test is successful.
Write to test file (f_write)...
[OK]
Test is successful.
End of Test.
SD CARD mount success! Filesystem also mounted.
Firmware loaded into flash.
Calculated CRC: 0x1EF640AF | Expected: 0x1EF640AF
CRC Verified! Firmware is good.
ESE5160 - EXIT BOOTLOADER
init_storage: please plug an SD/MMC card in slot...
init_storage: mounting SD card...
init_storage: SD card mount OK.
Test Program A - LED Toggles every 500ms
FlagB.txt added! Hold button pressed to reset device!
ESE5160 - ENTER BOOTLOADER

-- SD/MMC Card Example on FatFs --
SD Card initiated correctly!
Mount disk (f_mount)...
[OK]
Create a file (f_open)...
[OK]
Write to test file (f_puts)...
[OK]
Test is successful.
Write to test file (f_write)...
[OK]
Test is successful.
End of Test.
SD CARD mount success! Filesystem also mounted.
Firmware loaded into flash.
Calculated CRC: 0xFFECCCF0 | Expected: 0xFFECCCF0
CRC Verified! Firmware is good.
ESE5160 - EXIT BOOTLOADER
init_storage: please plug an SD/MMC card in slot...
init_storage: mounting SD card...
init_storage: SD card mount OK.
Test Program B - LED Toggles every 100 ms second
FlagA.txt added! Hold button pressed to reset device!
|
```

Video Implementation Link

<https://drive.google.com/file/d/1PkBrn4j8JweuwwvCsMoRXW76o6jn-LaF/view?usp=sharing>