PROJECT 2: FSM-BASED µSD CARD READER

OVERVIEW

For this project you will modify a program for your Freedom board which writes a data block on a µSD card and then reads it back to verify the data was written correctly. Start with the demonstration code (derived from ulibSD) and modify it as needed.

REQUIREMENTS

You will convert three functions into finite state machines: SD_Init, SD_Read and SD_Write. These functions are called by the test_write function in main.c.

MAXIMUM STATE TIMES

Modify so that no state can take more time to execute than the values shown in the table below:

Function	ECE 492-039	ECE 492-039 10% Extra Credit	ECE 592-066/603	ECE 592-066/603 10% Extra Credit
SD_Init	2.5 ms	1.0 ms	1.0 ms	500 µs
SD_Read	2.0 ms	1.0 ms	250 µs	100 µs
SD_Write	2.0 ms	1.0 ms	250 µs	100 µs

DEBUG SIGNALS

Create the debug signals shown in the table below. These will be used in debugging, your report, and grading. Refer to the appendix for an example of the signals in action.

Signal Name	Shield Signal	Logic Analyzer Channel	Notes
SPI CLK	CLK	0	SPI Clock
SPI DO	DO	1	MISO (master in, slave out)
SPI DI	DI	2	MOSI (master out, slave in)
SPI CS	CS	3	SPI Chip Select (active low)
SD_Read	Debug 2	4	1 whenever SD_Read function is executing, 0 otherwise
SD_Write	Debug 3	5	1 whenever SD_Write function is executing, 0 otherwise
SD_Init	Debug 5	6	1 whenever SD_Init function is executing, 0 otherwise
Scheduler	Debug 7	7	1 whenever main function is executing, 0 otherwise

PERIPHERALS USED

- Use the SPI interface to communicate with the µSD controller.
- Use GPIO peripherals for outputs to indicate the current system activity by asserting debug signals on Port B.
- The supplied code uses a timer (LPTMR0) to measure time for delays.

FURTHER INFORMATION

- Embedded Systems Fundamentals, Chapters 3 and 8 (SPI).
- FRDM-KL25Z Reference Manual.
- SD Specifications: Part 1 Physical Layer Simplified Specification, Version 6.00, April 10, 2017, SD Card Association Technical Committee. In project's Documents folder.

RECOMMENDED PROCEDURE

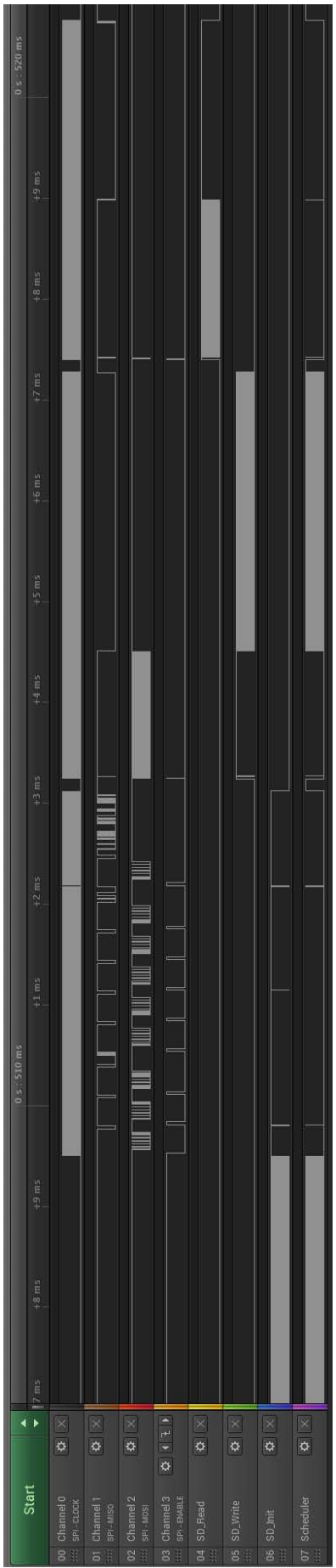
You are recommended to follow this procedure for each task root function (SD_Read, SD_Write, SD_Init):

1. Timing analysis
    a. Use a logic analyzer to determine the SPI communication rate.
    b. Raise the SPI bit rate by modifying the function **SPI_Freq_High** in spi_io.c. The faster you can run the SPI link, the fewer states you'll need in your FSMs. Refer to Chapter 8 of the textbook and the SPI chapter of the KL25 Subfamily Reference Manual for details.
    c. Add twiddle output bits to understand the timing of the three SD functions. Some debug outputs have been defined for you already in debug.h, but you'll need to add code in your SD functions to set, clear or toggle the bits (see next item).
    d. Use a logic analyzer to determine function timing information. Trigger the logic analyzer on the rising edge of the function's twiddle bit. You may need use single run (capture) mode to trigger only on the first rising edge. See the example timeline screenshot in the appendix.
        i. When does the function start (set the bit)?
        ii. When does it finish (clear the bit)?
        iii. When is it blocking in a waiting loop (toggle the bit twice)?
    e. Mark the diagram to identify types of time segments:
        i. Duration dependent on the SPI bit rate
        ii. Duration dependent on SD card controller response or activity
        iii. None of the above
2. Code structure analysis
    a. Obtain a CFG for each root function. The CFGs for SD_Init and SD_Write have been provided for you as PDFs in the project's Documents directory. They were created using a commercial code analysis tool (CrystalFLOW for C, by SGV Software Automation Research Corp.). You will need to draw a CFG for SD_Read.
    b. Examine the CFG and mark the loops with unknown or long durations.
3. Code transformation
    a. The CFG can be thought of as an FSM with all the code contained in a single state. Split this state into multiple states which meet the timing goals (e.g. no more than X ms in any state).
    b. Convert the function's code to implement the FSM you designed above. Modify the input and output data mechanisms (function parameters and return value) as described in the **C to FSM** lecture notes and as discussed in class. Note that you will need to modify the calling function to repeatedly call an FSM function until its return status is idle.
    c. Verify the FSM works properly.

DELIVERABLES

- Zipped archive of project directory submitted via Moodle. Reduce the archive size by cleaning the project targets in MDK-ARM before zipping the directory (Project -> Clean Targets, or Shift-F7).
- Demonstration with TAs after the submission deadline has passed.
- Project report. Use provided project 2 template/rubric.

APPENDIX: EXAMPLE OF DEBUG SIGNALS



Example of logic analyzer output showing debug signals at end of SD_Init, then SD_Write and then SD_Read. Note this code has been converted to FSMs, returning control to calling function frequently.