Lab 8, Week 2
Deliverable Specifications

ECE341, University of Idaho

Spring 2015

Prelab

The prelab should consist of the typical goal, a brief background, and the plan (with CFDs and DFDs). The plan for this lab should focus on developing the algorithm for the arbitrary-length write and read functions. Please note that **simply looping a single byte read or a single byte write will not suffice for the arbitrary length read and write functions**. In addition to the simple cases (writing data within a single page, for example), the algorithm should handle various edge cases that could occur when writing any amount of data to any memory location. Some examples of cases your algorithm should handle are listed below. Keep in mind this is simply a small subset of possible cases your algorithm should handle.

- What if the starting address is at the last byte in a page and the data is two bytes long?
- What happens if the data has a length of zero? Or the pointer to the data is `NULL`?
- What happens if the data size exceeds two pages?
- What happens when the data length is less than a page length, but crosses a page boundary?
- What happens if the memory address is invalid?

The algorithm developed and described in the prelab should be supported with diagrams and pseudo-code.

Demonstration

All EEPROM and I2C related code must reside in two files: `I2C_EEPROM_LIB.h` and `I2C_EEPROM_LIB.c`. The interface for the library (your EEPROM driver) is specified below:

- `void init_EEPROM()`
  - Initialize the I2C bus and the EEPROM such that after this function is called, the user of your library can immediately begin using the read and write functions
- `int eeprom_read(int mem_addr, char *i2cData, int len)`
  - Read `len` bytes from the EEPROM starting from EEPROM memory address `mem_addr` into the buffer pointed to by `i2cdata`.
  - Return zero on success or a non-zero value if there is an error. Examples of errors might include an invalid memory address passed into the function, a `NULL` pointer for `i2cData`, an invalid `len` argument, or an I2C bus error. Each error you handle should return a different non-zero value.
  - The function will not return until either `len` bytes were read into `i2cData` or there was error.

- `int eeprom_write( int mem_addr, char *i2cData, int len)`
  - Write `len` bytes into the EEPROM starting at memory address `mem_addr` from the buffer `i2cData`.
  - The function should not return until either the write is complete or an error has occurred[1]. See the `eeprom_read` specification for examples of errors.

In addition to the function interface definition, the EEPROM I2C device address and the page length should be `#define`d in the header (`I2C_EEPROM_LIB.h`) so the user of your driver library could easily use a (slightly) different EEPROM.

To test your library, create a new `main` source file, include your `I2C_EEPROM_LIB.h` file, and use the `eeprom_write` and `eeprom_read` functions to write data to the memory, then read the data from the memory into a different buffer, and compare the two arrays and verify they are the same. Use the LCD to display the result of the test.

The demonstration grade will be determined by how well your driver works when extensively tested with a test suite written by the TA. The remainder of the grade will be determined from the quality of your test set to test your driver. The point allocation for the demonstration grade follows:

75	Driver performance and functionality in test suite.
25	Quality of your test suite to test your driver.

Report

Introduction

The introduction should contain the goal of the lab, followed by the necessary background information to give the reader sufficient context to understand the remainder of the report. Try to keep the background information under a page.

Implementation

The implementation section should start by describing the three library functions (including the actual source code for the functions), and include the top level CFDs and DFDs for each function. After the library functions have been described, any support functions used should be described. An example of a support or helper function might include a function to poll the EEPROM to see if the write is complete. In addition to the description for the support functions, include the code for them as well as sub-diagrams.

Testing and Verification

This section should describe how you tested your library functions:

- Include oscilloscope screen captures that show a single byte write and a single byte read. Annotate the captures and identify each part of the I2C waveform (START and STOP conditions, the device address, R/!W bit, ACK, ect.). Also indicate the purpose for each byte on the waveform (control byte, address high, address low, data, ect). Finally, indicate on the waveform which device (master, slave, both or none) is "driving" the SDA line. Feel free to use multiple copies of the same waveform to maintain clarity.

- Identify cases that need to be tested. Some examples might include invalid parameters and data lengths around page boundaries.

[1]hint: use a function like `int wait_i2c_xfer(int SlaveAddress)` to poll the device until it isn't busy anymore before returning from the function

- For each case identified, describe the test you used to show that your library works correctly. Show the code that tests the library, but only show the parts that change between tests and make a reference to the code that is common between tests (such as the array comparison or initialization, for example).
- Determine the read and write speed for the EEPROM. Copy the following table into your report and record each measurement. Describe and show how you made the measurements (for example, if you used the oscilloscope then include the captures showing the measurement; if you used a timer, include the source code showing how and where it was used). Start each operation at the beginning of a page.

Length(Bytes)	Write time	Read time	Write Data Rate	Read Data Rate
1				
32				
63				
64				
65				
127				
128				
129				
1024				
8096				
16,384				
32768				

- Using the data from the above table:
  - Include a plot of the data.
  - What is the average data rate when writing to the EEPROM? What is the average data rate when reading from the EEPROM?
  - Develop an expression[2] for the approximate time required to write x bytes, $T_{write}(x)$.
  - Develop an expression for the approximate time required to read x bytes, $T_{read}(x)$.

Conclusion

In the conclusion of the report, address the following:

- On any bidirectional bus, there is a risk that two (or more) devices may try to place different voltage levels on the bus at the same time. When interfacing with the LCD, this problem was overcome with tristate outputs. How does I2C overcome this problem? In other words, how does I2C handle bus contention (where two devices attempt to drive the same wire (the SDA line, for example) to different voltage levels at the same time)? What are the consequences of this design? Why wouldn't a tristate solution (like the LCD interface) work for I2C?

[2]Don't use a simple linear regression. Rather, determine the number of pages that need to be written, and determine how long it takes to write a page to the memory. Use that information as a starting point for your expression.