Learning Objectives

This assignment continues a sequence of projects where students will progressively build upon embedded C programming skills to produce a working Industrial Internet of Things (IIoT) sensor endpoint.  When completed, the sensor will be capable of communicating with a management controller over Ethernet using a RESTful interface, measuring temperature readings at a programmable interval of time, and providing rudimentary error logging capabilities.  This assignment focuses on providing the necessary system support for non-volatile storage of vital product data (VPD), configuration information, and system event logging. Each of these functions make use of the ATMEGA 328P on-chip EEPROM.

For this project, the student will create library function code to access the EEPROM as well as higher-level functions to support vital product data and configuration setting storage.  EEPROM writes will be buffered in a 64-byte buffer that is written back to the EEPROM using an interrupt service routine.  Configuration data will be cached so that reads and writes to the data may continue regardless of the EEPROM hardware state.  More information about the functionality of this project can be found in the Project 2 Design Specification.

At the end of this assignment, you will be able to:

1. Use the Visual Studio IDE to create “classes” for non-volatile data structures and eeprom functions.
2. Implement a simple write buffer design pattern in C.
3. Implement a simple write cache in C.
4. Read and write data from an eeprom using an interface described in a datasheet.
5. Locate data structures within a memory device using diagnostic dump information.
6. Identify the size of program and data space used by the project.

Helpful Hints

These helpful hints may be useful when completing the project:

1. Review the class notes, data sheet and design guide before starting.
2. Make fullest use of the simulator to debug your code before testing it on actual hardware.
3. Perform unit testing on each class before integrating them with others.
4. Make private class-scope variables static.  This prevents the names from being linked to other same-named variables in other C files.  Declare public class variables as external in the associated header file, and declare them in the .C file.
5. Use #pragmas and a helper function to ensure proper timing when writing the EEPROM (see class notes).
6. Use the debugger with breakpoints to make sure that the EEPROM interrupt service routine is functioning properly - in particular, when it is done writing the buffer, it should disable interrupts so that it does not continue writing the device.
7. Place the structure definition for config and vpd in the header files for these classes.

Instructor Provided Headers and Libraries

Please use the following library and headers for this project.  Make sure that you update your project's Makefile to include the static library and the updated macro definition for YOURNAME.

* [delay.h](https://asu.instructure.com/courses/193939/files/89244457/download?wrap=1)[Download delay.h](https://asu.instructure.com/courses/193939/files/89244457/download?download_frd=1)
* [led.h](https://asu.instructure.com/courses/193939/files/89244543/download?wrap=1)[Download led.h](https://asu.instructure.com/courses/193939/files/89244543/download?download_frd=1)
* [log.h](https://asu.instructure.com/courses/193939/files/89244499/download?wrap=1)[Download log.h](https://asu.instructure.com/courses/193939/files/89244499/download?download_frd=1)
* [rtc.h](https://asu.instructure.com/courses/193939/files/89244432/download?wrap=1)[Download rtc.h](https://asu.instructure.com/courses/193939/files/89244432/download?download_frd=1)
* [timer1.h](https://asu.instructure.com/courses/193939/files/89244562/download?wrap=1)[Download timer1.h](https://asu.instructure.com/courses/193939/files/89244562/download?download_frd=1)
* [uart.h](https://asu.instructure.com/courses/193939/files/89244504/download?wrap=1)[Download uart.h](https://asu.instructure.com/courses/193939/files/89244504/download?download_frd=1)
* [util.h](https://asu.instructure.com/courses/193939/files/89244431/download?wrap=1)[Download util.h](https://asu.instructure.com/courses/193939/files/89244431/download?download_frd=1)
* [lib\_proj2.a](https://asu.instructure.com/courses/193939/files/89244391/download?wrap=1)[Download lib\_proj2.a](https://asu.instructure.com/courses/193939/files/89244391/download?download_frd=1)

Utility Programs

The following program can be loaded into your ATMEGA328p development board in order to erase the EEPROM (in case you program it with values you do not want).

* [erase\_eeprom.hex](https://asu.instructure.com/courses/193939/files/89244475/download?wrap=1)[Download erase\_eeprom.hex](https://asu.instructure.com/courses/193939/files/89244475/download?download_frd=1)

Design Specification

The following design specification contains detailed architectural definition for this project. Please refer to this when generating your code.

[Project 2 Design Specification.pdf](https://asu.instructure.com/courses/193939/files/89244487/download?wrap=1)[Download Project 2 Design Specification.pdf](https://asu.instructure.com/courses/193939/files/89244487/download?download_frd=1)

Project Description and Requirements

In order to get full credit for this project, the student has to create a C language program for the Atmega328P microcontroller that consists of five program files (config.c, eeprom.c, main.c, util.c,and vpd.c) and their related .h files.  Main.c provides the main program entry point and implements a control loop design pattern.  eeprom.c contains code to support reading and writing of the hardware eeprom device.  Config.c and vpd.c both implement higer-level data structures that access the eeprom “class”.  Util.c provides generic utility functions used by the other “classes”.

**THE EEPROM ONLY SUPPORTS 100,000 WRITE CYCLES.  CARE SHOULD BE TAKEN DURING THIS ASSIGNMENT NOT TO CONTINUOUSLY WRITE TO THE EEPROM WITHIN A LOOP AS THIS WILL QUICKLY WEAR IT OUT. PLEASE DEBUG USING THE SIMULATOR TO MAKE SURE THAT YOU DON'T CONTINUOUSLY WRITE THE DEVICE.**

**The requirements for main.c:**

1. main.c may only include header files provided for this project on Canvas or those created by the student for this project.

2. main.c implements a single function: int main().

3. In the initialization stage of the program, main() performs the following

* Initializes uart, config, led, log, rtc, vpd (in this order) with their respective init functions. (The rtc will initialize timer1, delay will initialize timer 0 when used).
* Initialize the led blink pattern to “--- -.- “ (OK in Morse Code);
* Initialize the RTC date/time to “01/01/2019 00:00:00

4. Following initialization, write the following information to the console, each on its own line:

* “SER 486 Project 2 – ***yourname***\n\r” (where ***yourname*** is replaced with your name).
* model
* manufacturer
* token

5. Prior to the main loop, perform the following tasks

* Set config.use\_static\_ip to 1 and set the config modified state.
* Clear the event log, then add three records to it with values of 0xaa, 0xbb and 0xcc.

6. Within the cyclic executive loop of main():

* Update the led blink finite state machine on each iteration through the loop.
* Update the log every iteration through the loop – this will write back any modified log entries if the EEPROM is not busy (at most one log entry per iteration).
* Update config every iteration through the loop. This will write back the config data if it has been modified and the EEPROM is not busy.

7. Add the following code after other functions within the loop. (Note that “dumped” will need to be declared and initialized to 0 prior entering the loop.

If ((!eeprom\_isbusy()) && (!dumped)) {  
      dump\_eeprom(0,0x100);  
      dumped=1;  
}

This code will dump the contents of the eeprom once and only once at the first time the eeprom write buffer is empty.

8. main.c must be properly commented with:

* Comments at the start of the file that include description of file, your name, course number and assignment number
* Comments at the start of each function.
* Comments within the code
* Comments must be accurate.

**The requirements for config.c and config.h:**

1. Do not include any header files in config.c other than eeprom.h, util.h, and config.h.

2. Implement the config class as defined in the design specification

* Correct member data
* Correct member functions
* Correct behavior

3. config.c and config.h must be properly commented with:

* Comments at the start of the file that include description of file, your name, course number and assignment number.
* Comments at the start of each function and within the code.
* Comments must be accurate.

**The requirements for eeprom.c and eeprom.h:**

1. The only header file that may be included in eeprom.c is “eeprom.h”.

2. Implement the eeprom class as defined in the design specification

* Correct member data
* Correct member functions
* Correct behavior

3. eeprom.c and eeprom.h must be properly commented with:

* Comments at the start of the file that include description of file, your name, course number and assignment number.
* Comments at the start of each function and within the code.
* Comments must be accurate.

**The requirements for util.c and util.h:**

1. util.h – use the file provided by the instructor without modification.

2. Include no header files in util.c

3. Implement the student-provided utility functions as defined in the design specification

* Correct parameters and return values
* Correct behavior

4. util.c must be properly commented with:

* Comments at the start of the file that include description of file, your name, course number and assignment number.
* Comments at the start of each function and within the code.
* Comments must be accurate.

**The requirements for vpd.c and vpd.h:**

1. Do not include any header files in vpd.c other than eeprom.h, util.h and vpd.h;

2. Implement the vpd class as defined in the design specification

* Correct member data
* Correct member functions
* Correct behavior

3. vpd.c and vpd.h must be properly commented with:

* Comments at the start of the file that include description of file, your name, course number and assignment number.
* Comments at the start of each function and within the code.
* Comments must be accurate.

**General Requirements:**

1. Build and link the code using the –Os (optimize for size) compiler setting and the lib\_proj2.a static library file. No other libraries should be included.

2. The project runs avr-size as part of the post-build process. This will allow you to determine the size of your code footprint.

* The amount of flash memory used is the sum of the TEXT and DATA numbers reported.
* The amount of static RAM used is the sum of DATA and BSS.

3. Please calculate the amount of flash and static RAM used and report the following (with calculations).  Do not share these calculations with anyone else, as this is equivalent to sharing test answers.

* Amount of Flash memory used =
* Amount of Static RAM used =
* Percentage of Flash memory used =
* Percentage of Static RAM used =

4. From your program output, identify the following information from the EEPROM dump and include your results with your memory calculations (above).  Do not share these answers with anyone else as this is equivalent to sharing test answers.

* EEPROM address of the data checksum for vpd
* EEPROM address of the data checksum for config
* Value of checksum for config
* EEPROM address of checksum for second log record
* Value of checksum for second log address

Screen Shot