CS 341 – Lab 3
EEPROM

In this lab you will write a checksum function to see if our EEPROM memory has been corrupted. The starter code can be found on GitHub under Lab 3. Today you will have to implement the initialize() and checksum() functions.

Some Background Info

Configuration data may be information about the product such as manufacturer's ID, part number, and even the customer service phone number or web/email address. The device may be programmed to display this information upon user request. Configuration data may also be calibration information used to properly calculate the values read from or written to analog ports or other data needed for the product to operate that may differ from device to device. Configuration data must be preserved through power outages and/or processor resets, but it usually not part of the compiled program code because it may need to have different values for different individual devices. It may be stored in Electrically Erasable PROM (EEPROM).

The contents of an EEPROM are preserved across power outages so your code can save data while your device is running. After it is powered off and back on again and/or reset, your code can retrieve it during your initialization sequence. However, you can never tell if your code is executing on an embedded system where your code has previously run and left valid configuration data in the EEPROM or not. So the format of your configuration data in the EEPROM must have some unique characteristics that your initialization code can use to verify its validity. If it appears to be valid, your code can go ahead and use it for configuring this new run. If it appears to be invalid, your initialization code should not rely on the contents of the EEPROM. It should either initialize it to default values or go through an interaction with the operator or another system (e.g. a network server) to obtain and store new configuration data in EEPROM.

A checksum is generated by operating on the contents of the configuration data with an algorithm that produces a checksum value that includes all values in the range of memory being protected. That value is then stored in a specific location in the EEPROM along with the configuration data. Again, the number of bits in the checksum should be large enough to avoid accidental matches and the algorithm should not typically produce a checksum value that will match when run on an uninitialized EEPROM containing all zeros or all ones.

Initialize()

initialize() will overwrite memory addresses 0 through 7 with a string of your choice, and recalculate the checksum and store it at address 8

1. familiarize yourself with each function and the logical flow in setup
2. Making use of the EEPROM.write() function, write a string into memory addresses 0 through 7
3. Call checksum() and store the value in address 8

checksum()

We need to perform a calculation on addresses 0 through 7 and return the value. To do this, we will make use of bitwise exclusive or (^) and bitwise not (~).

1. Cumulatively xor each address with one another
2. Return the compliment of this value

This algorithm is not perfect. The strings “hello” and “ehllo” would have the same checksum. If you have time, think about how you could tweak this algorithm so that it is no longer sensitive to element swaps.

Writing the Lab Report

Since this is the first lab that required you to write code, please remember to copy and paste your code or attach a .ino file. The lab template can be found on GitHub. Email to whitney.hamnett001@umb.edu, and CC all group members so that my grade and reply reach them as well. The report is due by the start of your next lab meeting.