ECE315 – Lab #5 Introduction to Web Services on an Embedded System

Lab Dates, Report Dates and Demo Due Dates:

Please login to Moodle at eclass.srv.ualberta.ca to access all relevant lab dates. The lab schedule section contains the important dates for this semester.

Objectives:

- To investigate the implementation of web services on a ColdFire MCF54415.
- To gain experience with the GET and POST methods of HTTP for passing data from the client to the webserver and back.
- To create a web form that will be used to interface with the hardware introduced in labs one through four.
- To learn about introductory concepts in data sanitization of user input and potential security issues.

Equipment, Parts, and Provided Software:

No new hardware is introduced in this lab. You'll be integrating web services running on the board with the existing hardware from lab one through four.

Documentation:

NNDK Programming Manual Chapters 8, 9, and 10 contain information on web server functions, interactive web forms and dynamic web content.

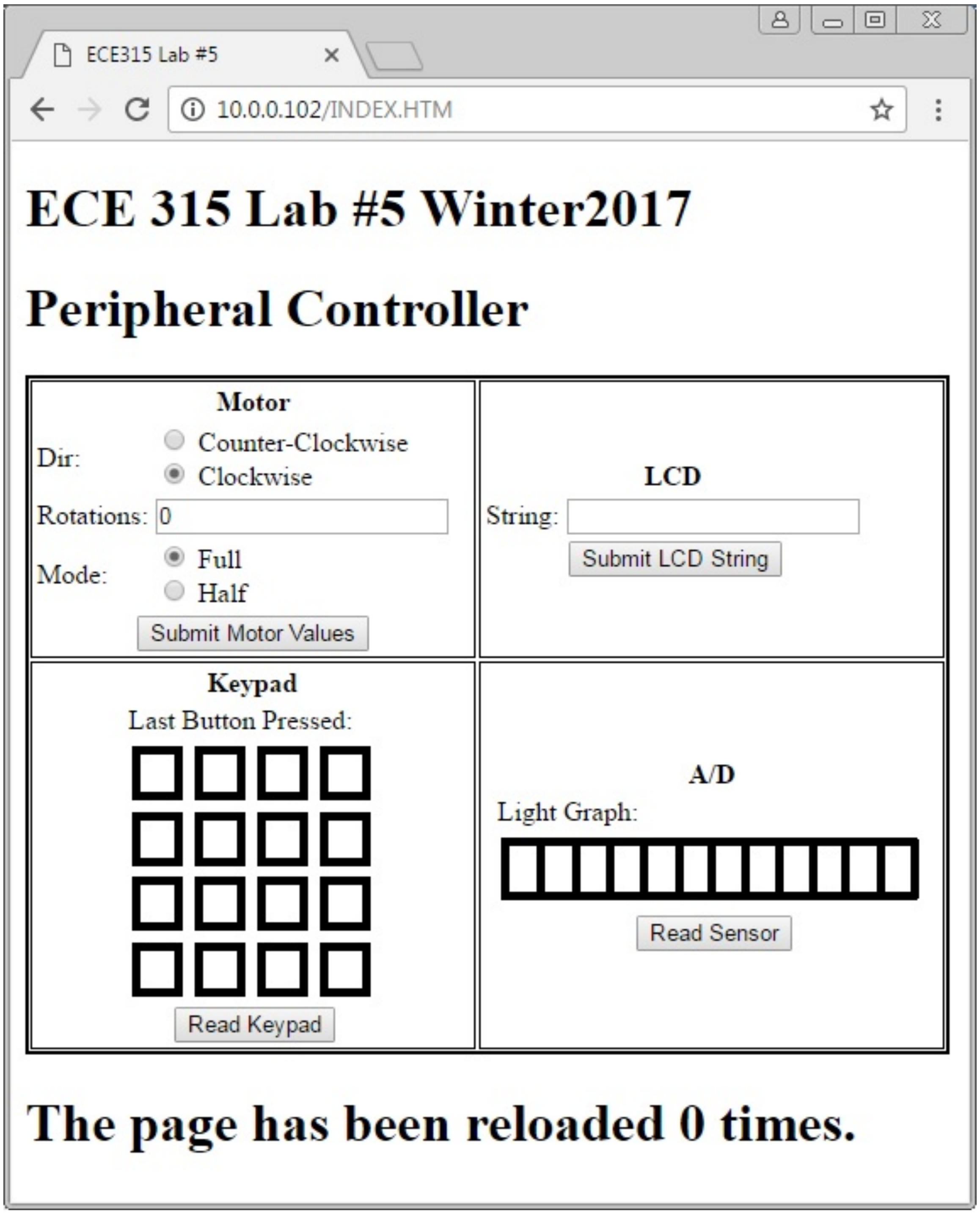
www.w3schools.com/HTML/HTML_forms.asp has more than enough information to complete the form portion of this lab. The screenshot below was also produced with the help of https://www.w3schools.com/css/default.asp.

Standard C Library Reference contains information on many string functions that you may choose to use. Remember that HTTP servers handle lots of data in ASCII text form. So string operations are a very important part of dealing with web server requests.

Introduction:

Web Forms

Your goal in this lab is to create a form that is similar to the following:



ECE 315 Lab #5 Winter2017

Peripheral Controller

Motor Dir: ○ Counter-Clockwise ● Clockwise Rotations: 0 Mode: ● Full ○ Half Submit Motor Values	**LCD** String: Submit LCD String
Keypad Last Button Pressed: Read Keypad	**A/D** Light Graph: Read Sensor

The page has been reloaded 0 times.

Figure 1: Sample Web Form.

In this lab, we’ll be working with forms, POST methods and a special HTML tag provided by NetBurner. The FUNCTIONCALL tag is available to dynamically call a function in the project to execute code.

A sample FUNCTIONCALL implementation is provided in the code package. Think of the code being executed as code that assembles the HTML page text that is returned to the client. Note that web servers are heavy senders and receivers of ASCII text.

The client (PC) – server (NetBurner) – client (PC) flow is the following:

1. The client submits a GET request to the HTTP server for the 10.0.0.102/index.htm file.
2. The server builds the index.htm file by assembling the fixed HTML text and executing the dynamically generated code contained in all the FUNCTIONCALL tags and corresponding functions.
3. The assembled file is transmitted to the client and displayed by the client-side web browser.
4. The user types data into the form on the client side and clicks on the submit button.
5. This user action generates a POST request to the HTTP server.
6. Our custom POST request handler is called by the HTTP server to handle the user-generated POST request.
7. We parse the user information passed to our post request handler and store it into the FormData class.
8. A new HTML file needs to be returned to the user. We want our updated index.htm file to be returned to the user so we redirect to index.htm to tell the HTTP server to send that file.
9. Before the file is sent to the user, the server again assembles the fixed HTML code and the dynamically generated FUNCTIONCALL code.

Repeated clicks of the buttons repeat the process starting at step 4 and proceeding through step 9. Reloading the index.htm file only generates repeated GET requests so our POST request handler will not be called.

Motor Subsection and User Motor Input Validation

In the motor subsection, you'll be using the Stepper class that you created in lab #4 to control the operations of the motor. Radio buttons control the direction and mode of the motor. Rotations will be entered using the text field.

If the rotations field contains invalid data, the motor should not be moved when the "Submit Motor Values" is pressed. If any of the fields are invalid the motor should not be moved and the Rotations: text field should contain feedback to the user on the failure. You may also choose to use javascript client-side to limit user input.

LCD Subsection and User LCD Input Validation

In the LCD section, you'll be using the LCD class that you used in lab #1 to display a string entered in the form on the LCD.

If the user inputs information that the LCD cannot display, the user must be notified. Several methods are acceptable. You may choose to enter information directly in the text field to notify the user. You may also choose to use javascript client-side to limit user input. Please limit the string size as well to handle possible buffer overflow issues. Limit the number of characters for the LCD to a maximum of 48 characters.

Keypad Subsection

In the Keypad section, you'll be using the features of the Keypad class that you added to in lab #2 to query the Keypad about which button is currently being pressed. Simply reloading the page should not change the values in the keypad subsection. Querying the keypad class for a new value should only occur when the "Read Keypad" button is clicked. A page reload should show the last button press that was detected but not query the keypad for a new press.

A/D Subsection

In the A/D section, you'll be using the AD class and the light sensor that you interfaced in lab #3 to update the light meter in the table cell. The screen shot above shows the values when the sensor is completely dark. Reloading the page should not change the values displayed. Querying the AD class for a new value should only occur when the "Read Sensor" button is clicked.

Test Information

Please consult eClass for some sample test cases.

Prelab:

No additional wiring is required for this lab. You may wish to consult Chapters 8, 9 and 10 of the NNDK Programming Manual for information about web server functions, interactive forms and dynamic web content.

Exercise 1:

Goal: In this exercise, the goal is to execute a dynamic function using the FUNCTIONCALL tag in the HTML/index.htm file of the project. Every time the index.htm page is reloaded, the counter is updated to indicate the number of times it has been requested.

1. Get the lab5.zip package from the course eClass site. Import the files into your project as you normally do. You'll need the html directory for this lab. Make sure to import it too.
2. Load the provided webpage from 10.0.0.102. Click on any of the buttons to send the form information to the NetBurner HTTPD task. Note what happens when you click on the buttons or reload the page. The counter should be dynamically updated and displayed to the webpage.
3. View the page source from within the browser and see that the FUNCTIONCALL tags have been removed and replaced with the actual string that the function writes to the file descriptor called sock.
4. Feel free to play with this code to help you understand the generation of the webpage that is returned to the client.

Exercise 2:

Goal: In this exercise, you will fill in the HTML table shell provided to you in the html/ index.htm file. The appearance of the table should approximately match the screen shot above.

Steps:

1. Modify the html/index.htm file to include the <!—FUNCTIONCALL function_name--> tags for each location on the form that requires dynamic content.
2. Create the C functions that will create the dynamic content in the form whenever the page is requested. Each of the functions must use the same prototype:

   void function_name (int sock, PCSTR url)

3. Several graphics files have been included in the code package. Feel free to create your own. Should you include images created by others you must provide the source in your report and in the source code. A link is sufficient.
4. Please include a screenshot of your form in your report.

Exercise 3:

Goal: In this exercise, you'll need to parse the data from the URL passed to you in the custom POST request handler called MyDoPost. You will also need to do the string conversions to ensure that the numbers inserted by the user are converted from ASCII text to data for validation and storage. Once the data has been parsed and verified server-side the request should be fulfilled. Validating client-side input using Javascript is optional. Validating user input server-side is mandatory.

Steps:

1. Read the examples in the formcode.cpp file to learn how to call ExtractPostData to help you with the parsing of the fields that are passed to you from the client-side form POST request.
2. Once the data has been parsed and the data extracted from the POST request, you must validate and store the data in the FormData class.
3. You may modify the FormData class in any way you want to fulfill the requirements of the lab. The provided shell is merely a starting point.
4. The FormData class must be considered as a global read/write accessible class and must therefore be protected from simultaneous access through the use of semaphores. You have been provided with FormData::Lock and FormData::Unlock methods to manage the sharing of this data. All sets and gets must be done under lock.
5. The following specifications for the form data apply:
    - Non-integer data and zero in the “Rotations:” field should be considered invalid. Discard the complete entry if any of the data is invalid. This will simplify the code.
    - Leading and trailing whitespace should not make an otherwise valid field entry invalid.
    - Rotations will be arbitrarily limited to 1-10000 for this lab.
    - Limit the number of characters in the “Rotations:” text field to 5.
    - Non-ASCII data in the “String:” field should be considered invalid. The LCD class can only display ASCII characters from 0x20 (space) to 0x7F (right arrow). Discard the complete entry if any of the data is invalid. This will simplify the code.
    - Limit the number of characters in the “String:” text field to 48.
6. The buttons on the form control the four peripherals. Reloading the page should not result in any changes to the peripheral status displayed on the webpage. Reloading the page should also not move the motor nor change the string displayed on the LCD.

Report Requirements:

- Your typed report is to briefly describe your design solution for exercise two and three.
- Please insert a screen capture of your form into the report body. Please provide a reference to the graphics in your code if you did not create the graphic yourself. If you used the provided graphics no reference is required.
- Be sure to provide neatly formatted and well-commented code for all exercises.

- Briefly describe your test cases for verifying that each of your solutions worked properly. Describe which case was tested, as well as the anticipated and actual results for each case. Place this data in a table. **Do not write out your test cases. Please tabulate them.** Non-tabulated test cases will not receive full credit on the reports.

Consult the report writing guidelines if you have any questions regarding the report format. The report writing guidelines are available on the eClass page under Lab Reference Materials

Marking Scheme:

Lab #5 is worth 20% of the final lab mark. Please view the lab 5 marking sheet to ensure that you have completed all of the requirements of the lab. The marking sheet also contains a limited test suite in the demo section. An extensive list of test cases is also provided in eClass.

Report and Source Code Submission:

Reports and code should be submitted via eClass by section. Please note that late reports and demos will not be accepted.