Microprocessor-Based Systems (E155)

Lab 6: Internet of Things

Learning Objectives

By the end of this lab you will have…

- Designed and built a simple IoT device
- Written C libraries to implement the SPI functionality of the ATSAM4S4B
- Interfaced with a temperature sensor module over an SPI link
- Interfaced the ATSAM4S4B with the ESP8266 module over a UART link
- Written a simple HTML page to control and display data from the peripherals connected to your µMudd board

Requirements

Build an internet accessible device to control an onboard LED and measure ambient temperature. Use an ESP8266 with the provided webserver code to host the webpage, use the onboard µMudd GPIO and SPI peripherals to toggle an LED and to read temperature from a provided sensor chip. An end-user must be able to turn the LED ON and OFF and view the current temperature from the webpage. For extra credit, implement the ADC functionality of the MCU to read from an analog sensor such as a phototransistor.

At heart, this lab asks you to consult the datasheet to learn how to directly control the SPI memory mapped peripheral on the µMudd ARM processor, so refrain from consulting the Lab 7 SAM4S4B.h or any other C device drivers for the SAM4S4B.

ESP8266 Web Server

Broadly speaking, everything that you see on the internet is the product of one computer presenting text to another. The text is often formatted in a special, internet-specific, way that includes information about how to display it which is referred to as *hypertext*. (Forgive the early internet engineers this indulgence; I'm sure it sounded really cool at the time.) Hypertext is specified using a compact programming language called hypertext markup language or HTML. It is transferred over the internet based on a predefined set of agreements between all computers which is referred to as the hypertext transfer protocol or HTTP. The latter most of these acronyms should be familiar: whenever you type http:// into a web browser you are informing your computer that you are attempting to retrieve hypertext from the address that follows.

There are two common tools that interact with HTTP: the web browser, which lives on a receiving computer, sends internet requests, and renders the received hypertext; and the web server, which listens for requests from the internet and sends out hypertext in response.

Implementing an HTTP web server on the ARM microcontroller is a non-trivial task. Instead, you will be using an ESP8266, a small WiFi development board which incorporates a TCP/IP stack as well as onboard WiFi and an integrated antenna. You are provided an Arduino language program which hosts an HTTP web server with an HTML page generated by the µMudd.

ESP8266-µMudd Interface

Download the Lab 6 support files from the class web page. Unzip SAM4S4B_lab6 and put all the files in your lab6_xx directory.

ESP8266 development boards are available from the E155 supply closet and should be preprogrammed with the webserver code. The SSID for the WiFi access point is listed on each chip. If after supplying power to the chip and waiting for it to initialize you do not see the expected WiFi network appear, you may need to reprogram the chip with the provided code. Instructions for doing so are provided in Appendix A.

The µMudd must supply a webpage to the ESP8266, and must interpret any web browser requests from the ESP8266. The devices interface through a 125000 baud serial connection, hosted on pins PA9 (URXD0) and PA10 (ITXD0) of the ATSAM4S4B and the hardware UART TX and RX lines of the ESP8266. The protocol is as follows:

1) When the ESP8266 updates the webpage from the µMudd, it sends the most recent request from the client, within '/REQ:' … '\n'. For example, a user accessing the page http://<server_address>/ledon would result in the request '/REQ:ledon\n' being sent to the microcontroller. A user accessing the root webpage of the server, http://<server_address>/ would result in the request '/REQ:\n'
2) The µMudd then transmits the entire web page to the ESP8266. The ESP8266 expects a webpage encoded as an HTML file. Therefore the webpage must start with '<!DOCTYPE html><html>' and end with '</html>'. The ESP will wait for either </html> or 200 ms from the last byte sent over serial before terminating the HTTP request and forwarding the content to the web browser.

The ESP8266 will create a WiFi access point named whatever SSID is labeled on the board. Connect to this WiFi network, and then go to http://192.168.4.1/. Beware that the ESP is slow and it may sometimes take a few attempts to connect.

µMudd Hardware and the Internet of Things

The last component of this lab is to write a program that parses a request from the ESP8266, toggle the LED state as necessary, read from the SPI temperature sensor, and use this data to generate a webpage that is transmitted to the ESP8266. We recommend ensuring that the FPGA code on your

board does not interfere with any of the pins which you may want to use for your sensors or else you will experience undefined behavior.

You will need to write an HTML webpage that displays dynamic temperature data as well as creating requests to change the state of the LED. There are many ways to do this, but we suggest the following resources for information on HTML formatting and interactive elements:
http://www.w3schools.com/html/default.asp
http://www.w3schools.com/html/html_forms.asp

The final product of this lab is a simple member of an emerging class of devices called the Internet of Things. Proponents of these devices argue that everything—from your washing machine to your car to giant factories—should be connected to the internet so that the shared data can be used to optimize and improve societal functions. Internet-controlled lighting, and internet-accessible sensors are two promising domains for the field, and are exemplified in this lab.

Extra Credit: ADC Implementation

Up to one point of extra credit can be earned by writing a library to interface with the memory mapped ADC peripheral provided on the ATSAM4S4B to read and display data from an analog sensor on the webpage. One possible simple analog sensor is an ambient light sensor based on an LPT2023 phototransistor. A sample schematic is shown below. We recommend reading the LPT2023 datasheet before starting to build this design.

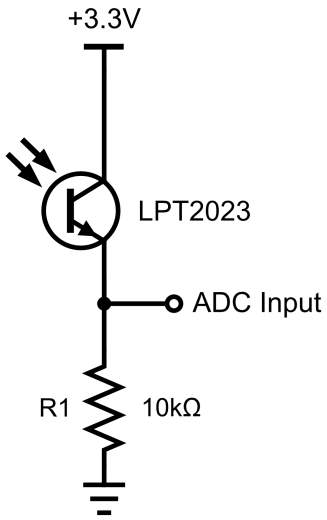

Figure 1: LPT2023 phototransistor example circuit

Credits

This lab was original developed in 2015 by Alex Alves ’16, and redesigned for the µMudd Mark 5.1 by Erik Meike ’21, Kaveh Pezeshki ’21, and Christopher Ferrarin ’20.

Appendix A: Configuring the ESP8266

The ESP8266 chips should be preprogrammed with the webserver code. The SSID is written on the chip. If it hasn't already been programmed, this Appendix details how to program the chip with the webserver code. Do not modify the code in the .ino file except to match the SSID with that listed on the label as the boards will be used by your classmates.

To download the web server onto the ESP8266, you will need to install the following software (as of 8/30/2019) if you are working on your own computer from home. The software should already be installed on the Digital Lab computers.

1) The Arduino IDE[1]
2) Board support for the ESP8266. This requires adding the ESP8266 board manager website[2] to the Arduino Board Manager within the Arduino IDE (File > Preferences), and then adding the ESP8266 board package within (Tools > Boards > Boards Manager).
3) Optional: The ESP8266 Exception Decoder[3]. ESP8266 crashes are notoriously difficult to debug, and this tool vastly simplifies the debugging process if you plan to alter the provided program. Note that this is not a required component of the lab.

The programming process is as follows:

1) Download ESPE155WebServer.ino from the class web page.
2) Open ESPE155WebServer.ino Arduino program with the Arduino IDE
3) Change the WiFi SSID at the top of the file to the name listed on the board.
4) Select the correct serial port (Tools > Port). If you are unsure of the current serial port, you can:
    a. On Windows, open Device Manager, and under the Ports drop-down look for 'Silicon Labs CP210x USB to UART Bridge'
    b. On macOS or Linux, enter 'dmesg | grep tty' in your favorite terminal emulator. This will return a list of all serial port activity.
5) Select the correct board (Tools). We suggest the following settings:
    a. Board: "NodeMCU 1.0 (ESP-12E Module)"
    b. Upload Speed: "921600"
    c. CPU Frequency: "80 MHz"
    d. Flash Size: "4M (1M SPIFFS)"
    e. Debug port: "Serial"
    f. Debug Level: "HTTP_SERVER"
    g. IwIP Variant: "v2 Lower Memory"
    h. VTables: "Flash"
    i. Erase Flash: "Only Sketch"
6) Compile and upload the script with the "Upload" button. Please note that the ESP will not program properly unless if the Tx and Rx pins are unconnected.

---

[1] https://www.arduino.cc/en/Main/Software

[2] http://arduino.esp8266.com/stable/package_esp8266com_index.json

[3] https://github.com/me-no-dev/EspExceptionDecoder