E155 Final Project Report μMudd Mark V.1 Bringup, Redesign and Lab 6 Revision

Christopher Ferrarin and Kaveh Pezeshki

14 December 2018

Abstract

The current ENGR155 development platforms, an Altera Cyclone IV on the μMudd Mark IV and Raspberry Pi 3, present an unbalanced system. The Raspberry Pi 3 is fast enough to invalidate the need for the Cyclone IV in all situations aside from those requiring precise timing, many I/O pins, or other FPGA-specific features. Our project involves bringup of a μMudd redesign that incorporates an ARM MCU, creation of peripheral drivers for the new ARM MCU, and a redesign of Lab 6: Internet of Things. We believe that the new μMudd will solve the imbalance between the MCU and FPGA, and allow for final projects more representative of real-world embedded development work.

1 Introduction

ENGR155 currently uses a Raspberry Pi 3 SoC (System on Chip) and Cyclone IV FPGA as a combined embedded system. Students are expected to work with both devices through the course's labs, and to create a final project that meaningfully implements both the SoC and FPGA. However, this system is unbalanced. The Raspberry Pi 3 uses a modern quad-core SoC, implementing ARM Cortex A-series cores running at 1.2 GHz. It is a device optimized for general computing rather than embedded tasks. The Cyclone IV FPGA, implemented on the in-house μMudd development board, is extremely slow in comparison, useful only for timing-critical or I/O-heavy tasks. This projects aims to solve the SoC-FPGA imbalance by bringing up the next-generation μMudd Mark V that implements an onboard microcontroller, and reworking critical labs.

1.1 Technical Issues

There were three blocking technical issues that prevented the adoption of the μMudd Mark V in the course. We aimed to solve these three issues in our project:

1. The μMudd Mark V, as provided at the start of the project, had a nonfunctional MCU. We needed to identify the blocking bugs in the new PCB layout and MCU implementation, and implement fixes to these issues in the schematics and layout for the board.

2. The ARM MCU implemented on the μMudd Mark V uses a different peripheral set and memory map compared to the Broadcom SoC on the Raspberry Pi 3. We needed to write an equivalent to easyPIO.h, the peripheral driver header file for the Raspberry Pi 3, for the new ARM MCU.

3. Lab 6: Internet of Things, requires an HTTP web server implemented on the Raspberry Pi 3. It is infeasible for students to write an HTTP web server from first principles on a bare ARM MCU, so we needed to redesign Lab 6 to better fit the new μMudd while retaining internet access, wireless communication, or another component of the general IoT device.

1.2 Board Design

Last year, a team of students designed a new μMudd board to rebalance the MCU (microcontroller) or SoC and FPGA platform. They could not upgrade the FPGA, as the current Cyclone IV is the highest-end FPGA available in a hand-solderable form factor as required by the μMudd. Instead, they moved from the Raspberry Pi 3 SoC to an Atmel SAM4S series MCU on the μMudd, which implements a single Cortex M4 core, benchmarking near 38 times slower than the Raspberry Pi 3 in CoreMark [1][2], and a peripheral set much more well-suited to mixed-signal embedded applications. This combination of MCU and FPGA is much more well-balanced, and allows for meaningful use of the FPGA as a compute accelerator. The student team also implemented new features to the board, including an external test board which checks that the MCU and FPGA are operational before assembly in Lab

1. Experience from ENGR085 will carry to this new board, as Keil can be used for device programming.

Upon starting this project, we were presented with a nonfunctional version of the next-generation ENGR155 board: the μMudd Mark V, as well as a set of schematics implementing proposed fixes to the board. While design flaws in the board prevented the MCU from executing code, the board incorporated all of the main features expected from the μMudd Mark V. The combined FPGA / MCU system is illustrated in Figure 1.2.

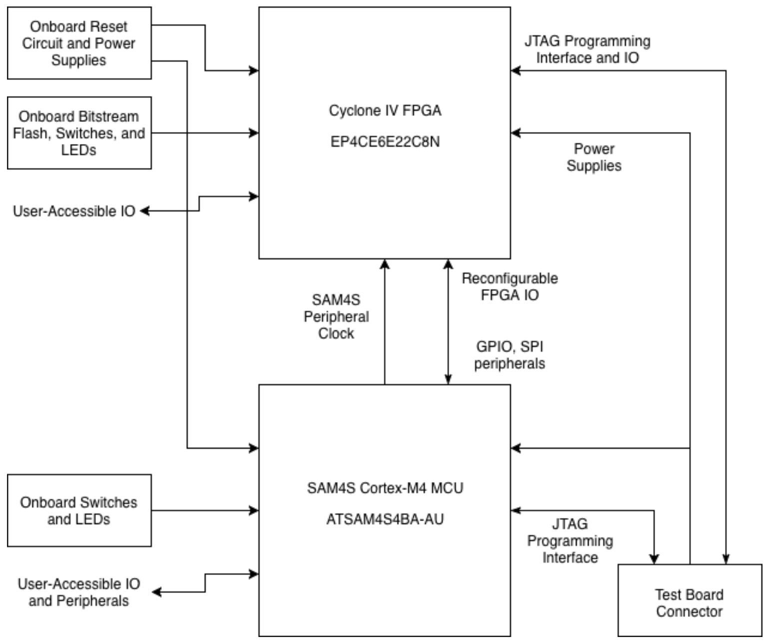


Figure 1: Block Diagram of the Redesigned μMudd

1.3 Lab 6 Design

While most of the labs implementing the Raspberry Pi 3 could be easily altered to instead use the SAM4S microcontroller, Lab 6 demanded a redesign. Currently, Lab 6 tasks students with developing a system which provides control over LEDs on the Raspberry Pi 3 GPIO and fetches current light intensity as measured by a phototransistor and SPI ADC. This information and control capability must be accessible via a webpage hosted on the system.

The Raspberry Pi 3, with its Linux (Raspbian) operating system, had a simple means of hosting an HTML web server in the form of Apache. On the other hand, our microcontroller did not have an operating system at all, and so we sought to retain the learning goals and basic framework of the current Lab 6 while modifying it to be compatible with the SAM4S. Our proposed implementation used an ESP8266 and a WiFi-enabled microcontroller, to host an HTTP webpage as in the current version of Lab 6, as well as adding an SPI pressure and temperature sensor. The block diagram in Figure 1.3 illustrates this redesign of Lab 6.

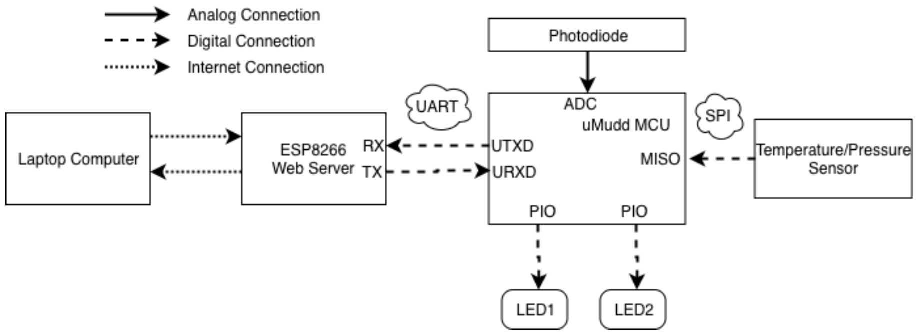


Figure 2: Block Diagram of the Lab 6 Proposal

1.4 Device Driver Design

In the ENGR155 labs, a core skill students gain is the ability to read the datasheet of a new microcontroller and build up basic functionality for a peripheral. Still, in later labs and in the final project, it is convenient to provide a device driver so that more complicated, higher-level designs can be implemented without the need to spend additional time on low-level peripheral work. With the replacement of the Raspberry Pi 3 with a microcontroller came the need to develop a new device driver for the SAM4S. While many exist due to the prevalence of the ARM Cortex-M series, we sought to develop one simple enough for students to understand while robust enough to handle most of the functionality students might use in their final project. This device driver would also be instrumental in our board bringup efforts to help verify functionality of various peripherals and components of the microcontroller as well as in our Lab 6 redesign to create a working prototype.

2 Implementation and Debug Details

2.1 Board Design

Our initial tasks in the project were to determine the cause of the MCU failures in the μMudd Mark V and implement fixes to these bugs in the schematic and layout of the board.

2.1.1 μMudd Mark V Debug

After initial testing of the μMudd Mark V, we noticed unstable MCU behavior. We were only occasionally able to program the SAM4S, and in the cases where our programmer reported a successful program flash, we were unable to enter the debugger, and the MCU would not execute the program outside of the debugger.

Further investigation revealed that, in the cases where the programmer indicated a successful program flash, the microcontroller program memory remained initialized to all 1s. This can be observed in Figure 3.

We realized that this issue was caused by two bugs in the current PCB design: a wiring error with the flash erase pin on the microcontroller, and potential instability due to to the

```
?main:
__cmain:
      0x400040: 0xffff 0xffff  MRC2      p15, #7, PC, C15, ...
      0x400044: 0xffff 0xffff  MRC2      p15, #7, PC, C15, ...
      0x400048: 0xffff 0xffff  MRC2      p15, #7, PC, C15, ...
_call_main:
      0x40004c: 0xffff 0xffff  MRC2      p15, #7, PC, C15, ...
      0x400050: 0xffff 0xffff  MRC2      p15, #7, PC, C15, ...
      0x400054: 0xffff 0xffff  MRC2      p15, #7, PC, C15, ...
      0x400058: 0xffff         DC16      65535             ...
_main:
      0x40005a: 0xffff 0xffff  MRC2      p15, #7, PC, C15, ...
__low_level_init:
      0x40005e: 0xffff 0xffff  MRC2      p15, #7, PC, C15, ...
```

Figure 3: The SAM4S main function program memory after a program flash

current microcontroller power supply.

The largest problem with the current μMudd design lies in the MCU ERASE pin, which re-initializes the onboard flash and resets the processor. The ERASE pin can also serve as general-purpose I/O after configuration [3].

On boot, ERASE must be held low to prevent flash erase and re-initialization of the processor. On the current μMudd, ERASE was tied to a general I/O pin on the Cyclone IV FPGA. The connection can be seen in the schematic in Figure 4.

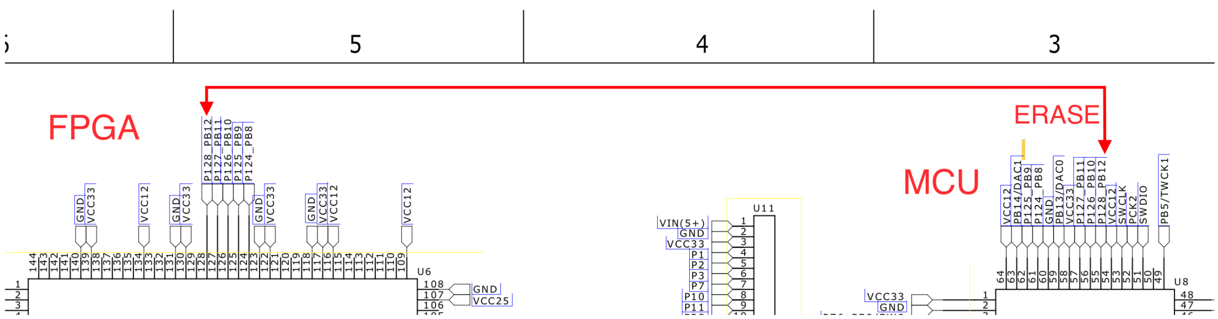


Figure 4: The marked connection ties ERASE on the MCU to pin 128 on the FPGA

The ERASE pin contains a 100-kΩ pull-down resistor [3].An unconfigured Cyclone IV I/O pin contains a 25-kΩ pull-up resistor [4]. This creates a voltage divider circuit as shown in Figure 5. This provides a predicted voltage of 2.64V on the MCU ERASE pin, close to the 2.86V we observed. This is a high logic level which prevented FPGA programming. After correcting this issue by manually breaking the ERASE pin trace, we were able to program and execute code on the μMudd.

The second issue with the microcontroller implementation lies in its power supply configuration. The MCU requires a 3.3V and 1.2V power supply. It can be powered via one 3.3V supply, and use an internal regulator to generate 1.2V, or it can be powered with an external 3.3V and a 1.2V supply. On the μMudd Mark V, discrete 3.3V and 1.2V regulators power the FPGA and MCU. This dual-regulator design of the current board can introduce microcontroller boot issues if timing is not correct, with proper timing illustrated in Figure 6.

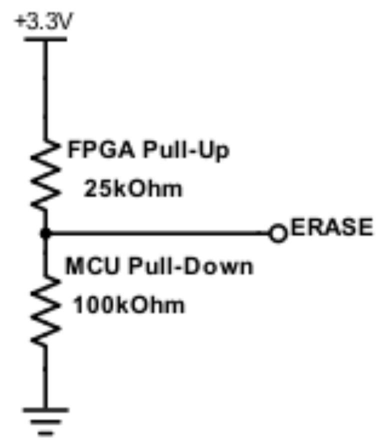


Figure 5: Voltage divider producing a high logic level on the MCU ERASE pin

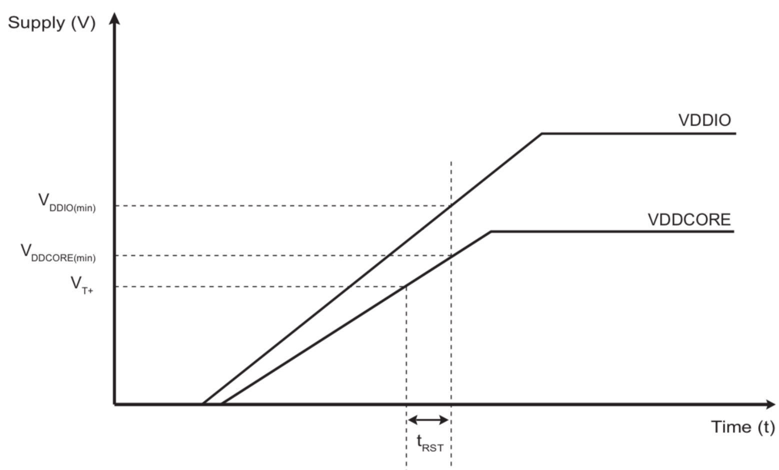


Figure 6: Timing requirements for the 1.2V (VDDCORE) and 3.3V (VDDIO) supplies[1]

We believe that these potential timing errors can cause system instability, as we observed an unresponsive MCU after boot that could only be solved with a full erase and reset.

2.1.2 μMudd Mark V Redesign

We therefore began a PCB redesign to solve these issue. We incorporated the following changes:

1. We tied the ERASE line to a separate pushbutton on the PCB. The FPGA can no longer erase the MCU flash.

2. We moved MCU core and PLL power supplies to the onboard regulator, with an associated RLC filtering circuit as suggested by the manufacturer [5].

3. We re-routed the JTAG interface to a smaller connector compatible with the common J-Link EDU Mini ARM programmer [6].

4. We re-positioned components on the board to allow for easier assembly, as well as for easier button and LED access.

5. We added via shielding on the MCU to FPGA clock line.

6. We propagated the above changes to the Lab 1 test board.

The μMudd and test board PCBs pass all design tool and manufacturer DFM and DRC checks, and are ready for manufacture at Advanced Circuits. Schematics, layout, and a BoM for the μMudd and test board are available in appendices A and B.

2.2 Lab 6 Design

The current version of Lab 6 relies on an HTTP webpage hosted by Apache on the Raspberry Pi 3. As the new ENGR155 embedded system no longer runs Linux, we do not have access to common web hosting tools. However, we wanted to maintain a similar feature list to the current Lab 6. In the revised lab, students will have to implement the following:

1. A voltage divider circuit to generate voltage proportional to ambient light intensity with a LPT2023 phototransistor

2. SPI device drivers and support for the BMP280 pressure and temperature sensor

3. GPIO device drivers to control the state of an two LEDs

4. A basic web page that provides access to current pressure, humidity, and light intensity, as well as LED on/off buttons

Students will be provided with code examples for interfacing with the ADC peripheral on the microcontroller, for UART communication, for controlling the state of a single LED, and for hosting a basic HTTP webpage with dynamic data.

This retains the lab objectives of learning basic IoT and web design and of gaining more experience implementing peripherals and sensors from their datasheets. A rewritten lab manual for Lab 6 is available in Appendix 3.

2.2.1 HTTP Webpage Hosting

It is a non-trivial task for the μMudd board alone to host an HTTP webpage. This would require implementation of a TCP/IP stack, of the HTTP server protocol, and the physical addition of an interface able to connect the μMudd to a router. Instead of attempting this implementation, we decided to use an external off-the-shelf platform capable of TCP/IP networking and WiFi.

In particular, we decided to use an ESP8266 [7], implemented on a NodeMCU ESP-12E [8], as a replacement for the Apache web server. The ESP8266 is a microcontroller platform that implements WiFi and a TCP/IP stack on an RTOS (real-time operating system). This allows the ESP8266 to host or connect to WiFi networks, as well as to act as a client or server in HTTP transactions.

We did not want to require students to implement the webpage hosting backend, as we believed this would increase lab workload to an unacceptable level. The ESP8266 therefore serves as a 'black box' peripheral to the μMudd, accessible over UART. The student has the ability to reprogram the ESP8266, and will be provided with reference code in which they can specify network connections and IP addresses, as well as providing additional configurability. The ESP8266 also prints debug and status messages on a serial connection over a MicroUSB connector.

On boot, the ESP8266 connects to a network predefined by the user, hosts a WiFi access point, and starts a HTTP server. It then requests a webpage to display from the μMudd. Having received a webpage, the server is now ready to handle a HTTP client.

After the user interacts with or refreshes a webpage, the ESP8266 needs to return the client request to the μMudd. A request - response transaction between the ESP8266 and μMudd can take upwards of a half-second for a webpage of any reasonable size. This delay led to hangs and crashes in the client browser. We therefore implemented a HTTP server hosting and webpage request - responses with the μMudd as semi-independent processes. A new webpage will be requested from the μMudd only under the following conditions:

1. The user has reloaded or interacted with the webpage since the webpage was last reloaded from the μMudd

2. At least 10 seconds have passed since the last request-response communication from the μMudd

The ESP8266-μMudd protocol is as follows:

1. When requesting a webpage from the μMudd, the ESP will transmit an abbreviated URL of the last webpage requested by an HTTP client. For example, if the ESP8266 had an IP address of 192.168.1.2, and the user requested the page http://192.168.1.2/ledon, the ESP8266 would transmit the string "<ledon>".

2. The μMudd is then responsible for parsing this abbreviated URL, performing any required actions, and then returning an HTML webpage. The ESP8266 will accept a webpage only if it begins with the string "<!DOCTYPE html><html>" and ends with the string "</html>"

This restricts users to webpage interaction based on redirects, and leads to a time lag between user action on the webpage and the μMudd response. However, this behavior is superficially sufficient for Lab 6. Code for the ESP8266 'black box' peripheral is available in appendix 4.

2.2.2 Demo Code

We have designed a demo webpage to provide students with an example of an IoT device. The demo implements the following features:

1. A status LED on pin PA18 which illuminates when a webpage is transmitted over UART

2. A user-controllable LED on pin PA17

3. Reading from ADC channel 2 on pin AD2

4. An ESP8266 interface which hosts a webpage which displays the voltage on AD2, and provides buttons to turn the user-controllable LED on and off

We have demonstrated this lab on the Olimex SAM3-P256 development board as a temporary replacement for the revised μMudd Mark V. While the SAM3-P256 implements a SAM3S-series MCU rather than a SAM4S-series MCU, the two chips share identical base instruction sets, peripheral sets, and peripheral memory maps. The schematic for our demo is available in Figure 2.2.2, and code for the demo is available in appendix 5.

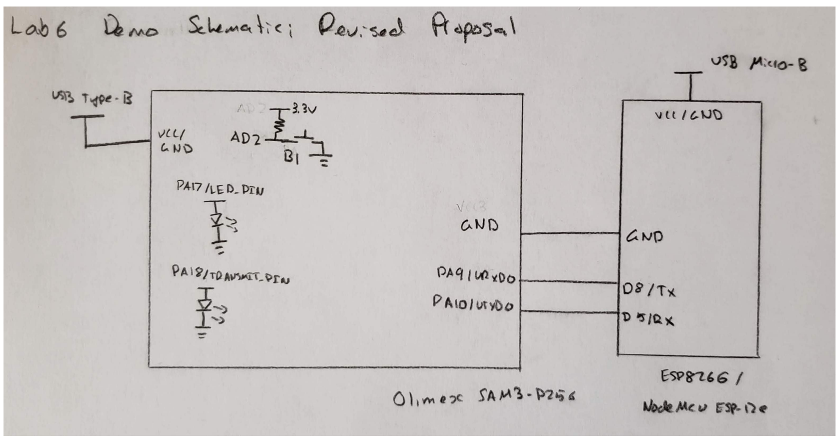


Figure 7: Lab 6 Demo Schematic. Note that the circuits implemented in the box to the left are pre-existing peripherals on the Olimex SAM3-P256

2.3 Device Driver Design

The SAM4S device driver provides minimal working support for the following peripherals:

- PMC (Power Management Controller): For clock multiplexing to peripherals and controlling programmable clocks.

- PIO (Parallel Input/Output Controller): For peripheral function pin multiplexing and reading and writing digital values from pins. Both ports (PIOA and PIOB) are supported.

- TC (Timer Counter): For system delays and counting and triggering at various clock speeds. Both channels (TC0 and TC1) are supported. Since this peripheral will mostly be used to generate delays, we have provided code that allows students to ignore any other functionality of the Timer Counter in order to simply delay the system, while other low-level code exists if they desire this additional functionality.

- SPI (Serial Peripheral Interface): For serial communication with external devices that support SPI.
- UART (Universal Asynchronous Receiver-Transmitter): For serial communication with external devices that support UART.
- PWM (Pulse Width Modulation Controller): For generating square waves of various frequencies and duty cycles. All four channels (PWM0 - PWM3) are supported.
- ADC (Analog-to-Digital Converter): For reading analog voltages.
- RTC (Real Time Clock): For automatic tracking of the time and date.

We believe that this set of peripherals will allow students to tackle the majority of tasks they might encounter in later labs and in the final project, while not being so large as to be unwieldy and confusing.

One guiding principle behind our device driver design was organization. We achieved this in three main ways:

1. We split up the device driver into multiple, independent header files.
2. We used naming conventions for registers, functions, and definitions that were consistent across peripherals.
3. We defined all registers and bits using a hierarchy of named structs.

Dividing the over 1500 lines of code for the device driver into multiple header files has a number of advantages over the previous single easyPIO.h file that was used previously. First, it allows students to more quickly find what they are looking for. For example, if a student wishes to find what inputs the pioPinMode() function can take, rather than searching through a large block of code, they can simply open the SAM4S4B_pio.h header file and look in the "PIO Functions" block of code. This organizational system also delegates all definitions that the user would be likely to change to the SAM4S4B_sys.h header file, since it can be difficult to look through a large device driver and deduce which definitions a user can change without breaking a fundamental function. Finally, this system is more efficient; although this is unlikely to be a major problem, compiling and storing a large header file in a microcontroller every time it is reprogrammed can be slow and inefficient. This way, students need only include in include preprocessor directives those header files for peripherals they are actively using, cutting down on compile time and memory requirements.

Naming conventions for definitions, registers, and functions alike further helps keep the device driver simple and easy to read. Although this makes the code somewhat more verbose, we decided to begin each definition name, register name, and function name of every peripheral with a prefix of that peripheral's name. For example, for the PIO peripheral, a definition might be named PIO_PA13, a register might be named PIO_PER, and a function might be named pioInit().

The final change may well be the most significant. In ENGR155, students are taught how to interact with special function registers on the lowest level possible: creating pointers to specific words in memory and then writing or reading bits or fields using bitwise "and"

and "or" assign operators and bit shifts. While this provides students a fundamental understanding of the effect of their code on hardware, such code is almost impossible to read without the help of a datasheet, and even then can take a copious amount of time. To improve this aspect of the device driver, we defined every bit, register, and register block with the following hierarchy:

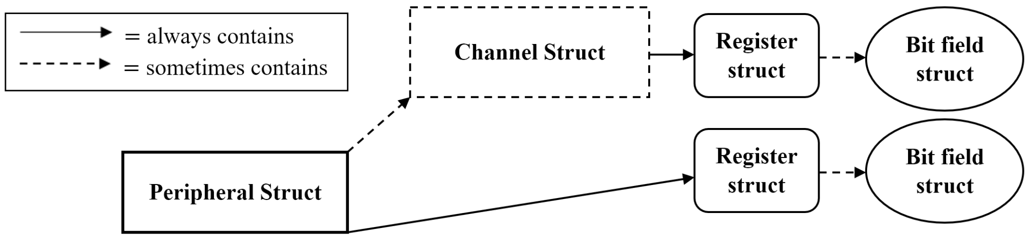


Figure 8: Hierarchy of structs used to organize register definitions in the SAM4S device driver.

In addition, we defined a pointer to every peripheral struct, meaning that students do not need to interact with addresses of words or bit locations within words at all; rather, they interact with the names of registers and bits, which is much more intuitive and readable. Finally, using dot and dereference operators to move down the struct hierarchy results in overall cleaner code than direct low-level operations on words.

Code for the peripheral drivers is available in Appendix F.

3 Results and Discussion

Our proposal deliverables were, given verbatim from our proposal:

1. Identify blocking bugs in the μMudd which completely prevent MCU programming and operation

2. Determining and implementing a solution to the above bugs. This solution may take several forms, as described below (not restated here)

3. Reworking Lab 6 with instructor guidance to fit the new μMudd MCU

We have successfully completed these tasks. In response to the first two tasks, we have created a revised set of PCBs which solve the MCU programming and program execution issues, as well as providing general quality-of-life improvements. In response to the third task, we have created a revised Lab 6 that retains all IoT elements while providing a meaningful role for the μMudd.

Our initial stretch goals of testing labs 4,5, and 7 transitioned into creation of extensive peripheral drivers for the μMudd. We believe that this is a more valuable product than purely testing the other microcontroller-related labs, as it provides a framework that vastly simplifies any further lab testing or development.

Despite this progress, the new μMudd and Lab 6 are not yet adequately polished for release into the next session of ENGR155. We would in particular like to improve our ESP8266 web server to eliminate the 10 second maximum delay between client interaction with the webpage and microcontroller response. We plan to implement and test all other labs, rewrite lab manuals, and thoroughly test the revised μMudd Mark V over the upcoming semester.

4 Budget and BOM

Item Name	Item Description	Vendor	Item Cost
SAM3-P256	SAM3S Development Board	Olimex	$31.09
Ailavi JTAG	JTAG-SWD Adapter	Amazon	$8.99
GikFun EK1199	JTAG-SWD Adapter	Amazon	$7.66
J-Link EDU Mini	JTAG Debugger	Adafruit	$19.95
ESP8266	Web Server Module	Amazon	$5.49
Total Budget			$422.19

References

[1] Benchmark 2758: Broadcom BCM2837 / Raspberry Pi 3. (2018). Retrieved December 13, 2018, from https://www.eembc.org/benchmark/reports/benchreport.php?suite=COREbench_scores=2758

[2] Atmel's SAM4S Clinches Highest CoreMark/MHz scores. (2013, June 27). Retrieved December 13, 2018, from https://atmelcorporation.wordpress.com/2013/06/24/atmels-sam4s-clinches-highest-coremarkmhz-scores/

[3] SAM4S Series: SMART ARM-Based Flash MCU Datasheet. Atmel Corporation, San Jose, CA, 2015.

[4] Cyclone IV Device Handbook, Volume 1 [PDF]. (2016, December). San Jose, CA: Altera Corporation.

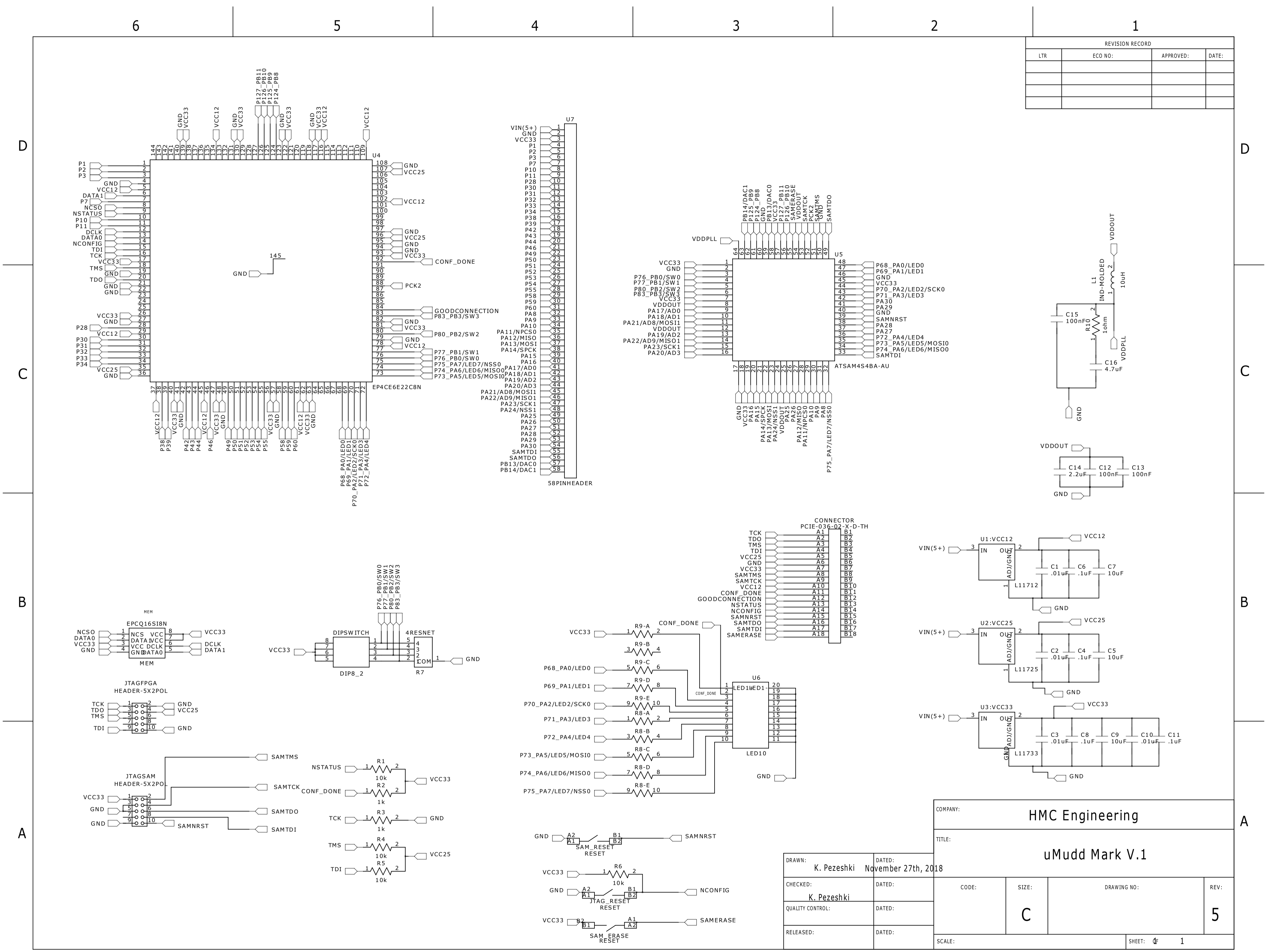
[5] Application Note AT03463: SAM4S Schematic Checklist [PDF]. (2015). San Jose, CA: Atmel Corporation.

[6] J-Link EDU Mini. (2018). Retrieved December 13, 2018, from https://www.segger.com/products/debug-probes/j-link/models/j-link-edu-mini/

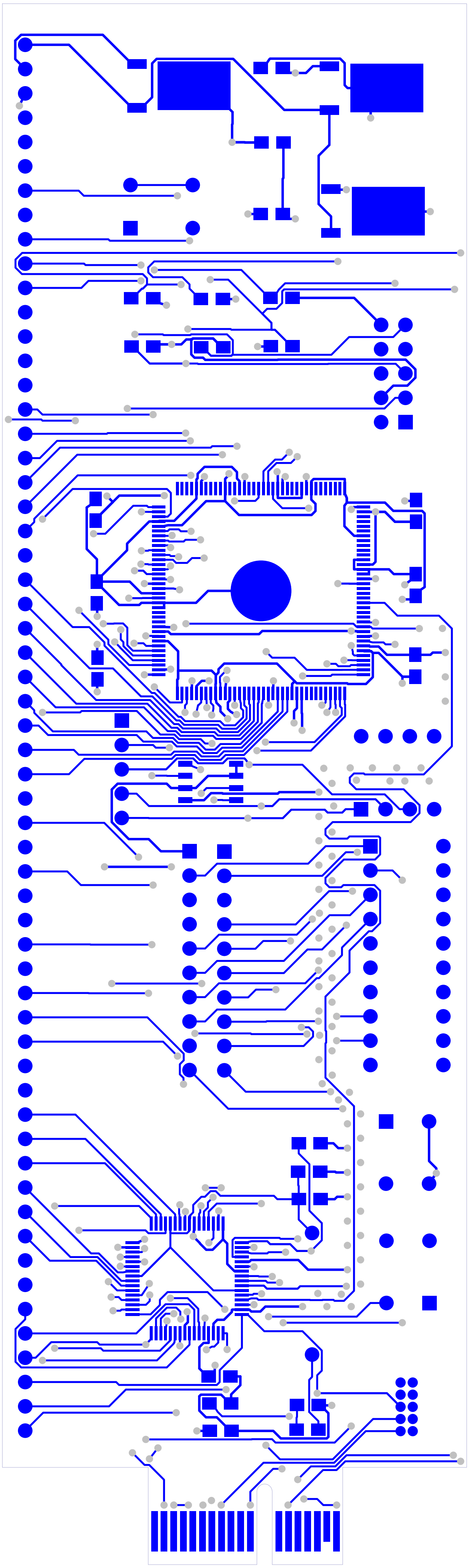
[7] ESP8266. (2018). Retrieved December 13, 2018, from https://www.espressif.com/products/hardware/esp8266ex/overview/

[8] NodeMCU Documentation. (2018, September). Retrieved December 13, 2018, from https://nodemcu.readthedocs.io/en/master/

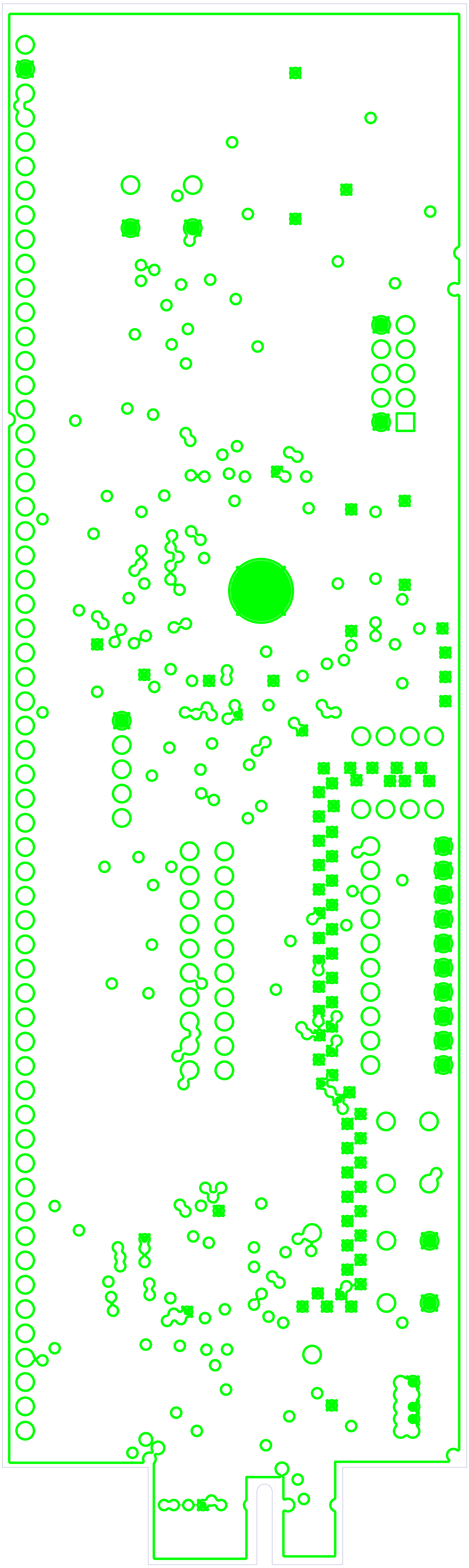
Appendix A: μMudd Schematic and Layout


REVISION RECORD
LTR
ECO NO:
APPROVED:
DATE:
COMPANY:
HMC Engineering
TITLE:
uMudd Mark V.1
DRAWN:
K. Pezeshki
DATED:
November 27th, 2018
CHECKED:
K. Pezeshki
QUALITY CONTROL:
RELEASED:
CODE:
SIZE:
C
DRAWING NO:
REV:
5
SCALE:
SHEET: 1 OF 1

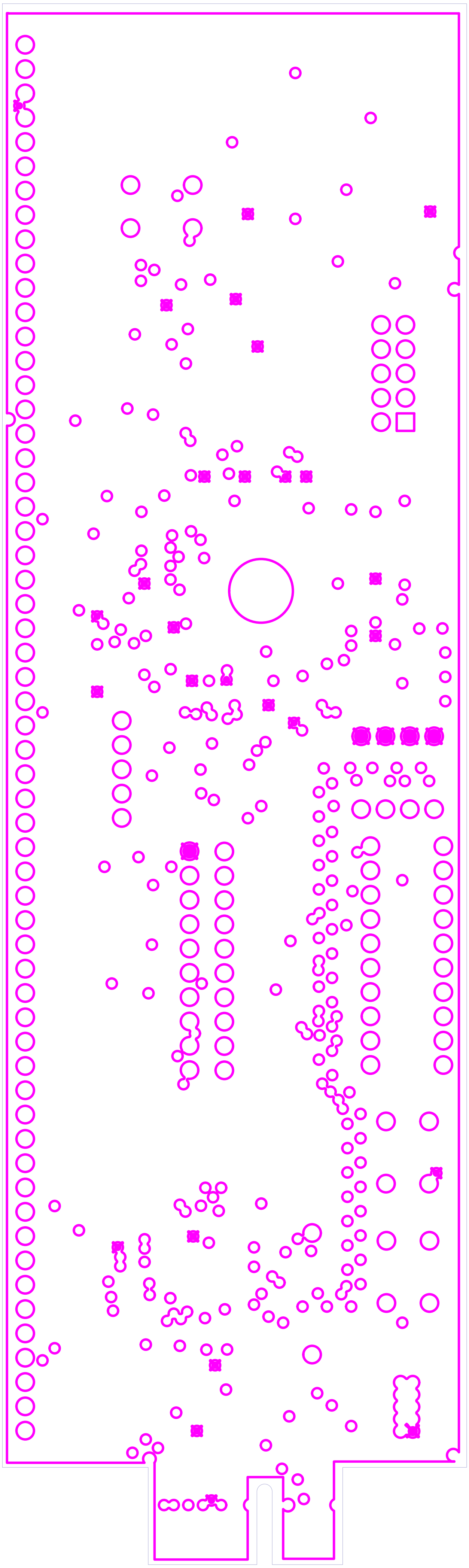
Top Copper



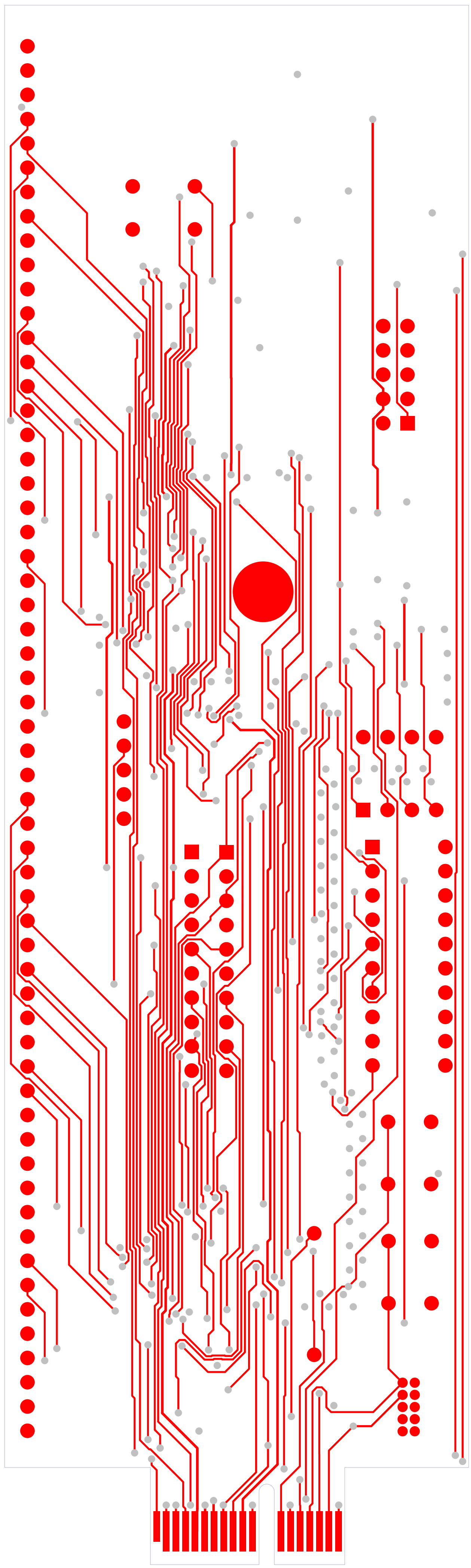
Ground Plane



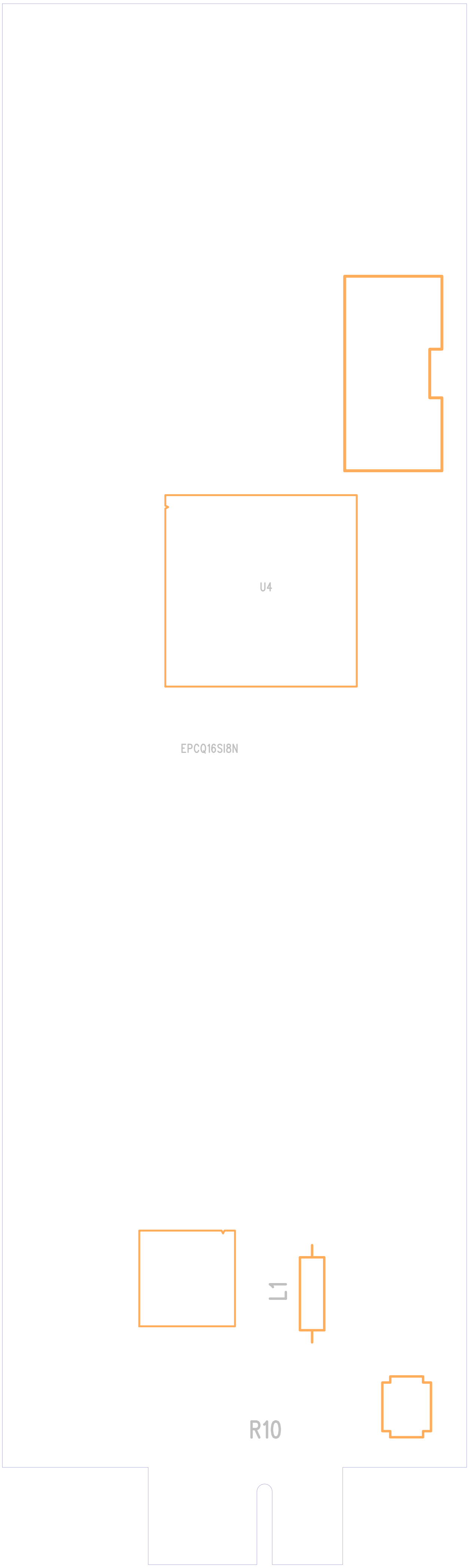
Power Plane



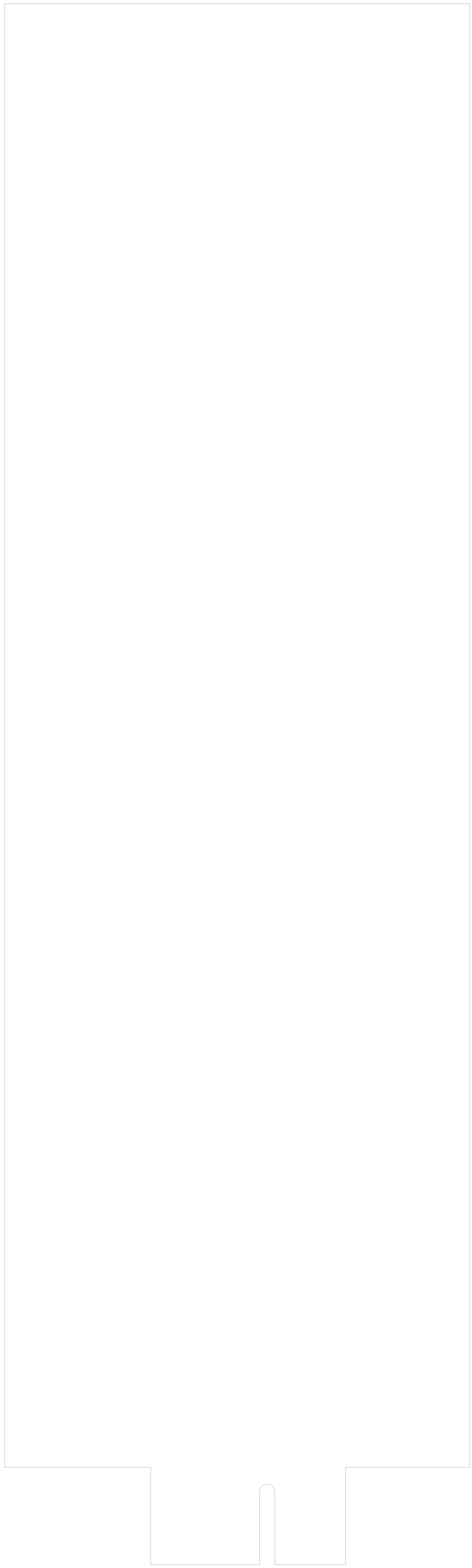
Bottom Copper



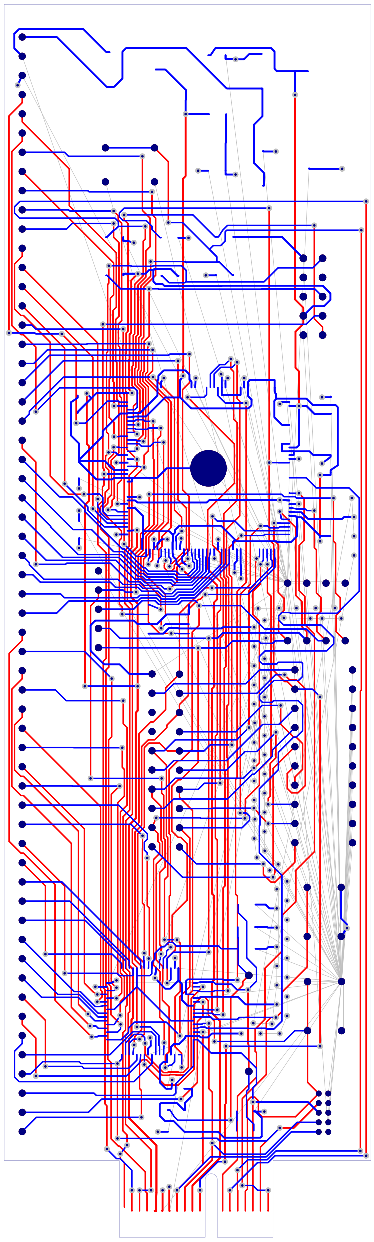
Assembly Top



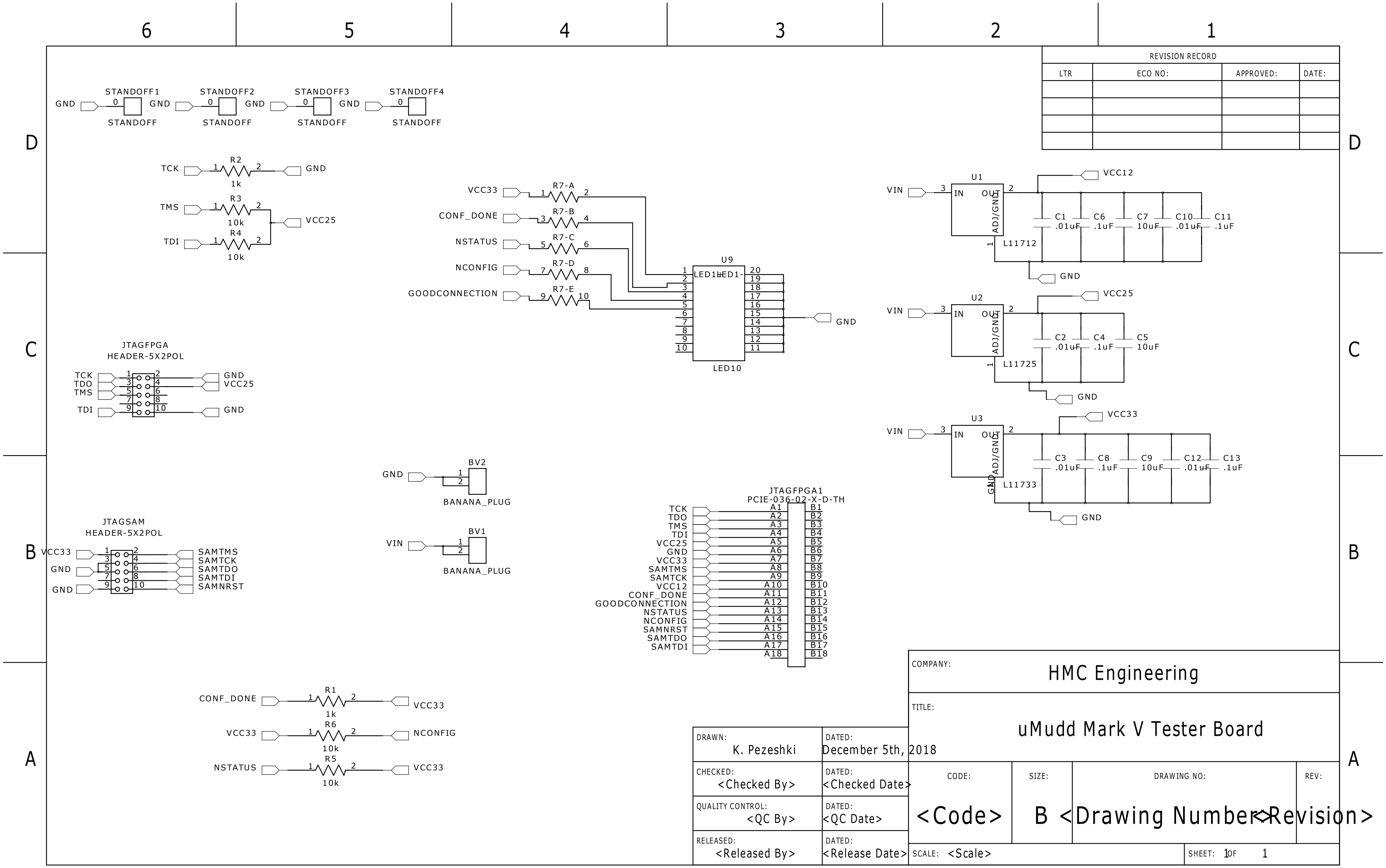
Assembly Bottom



All Traces

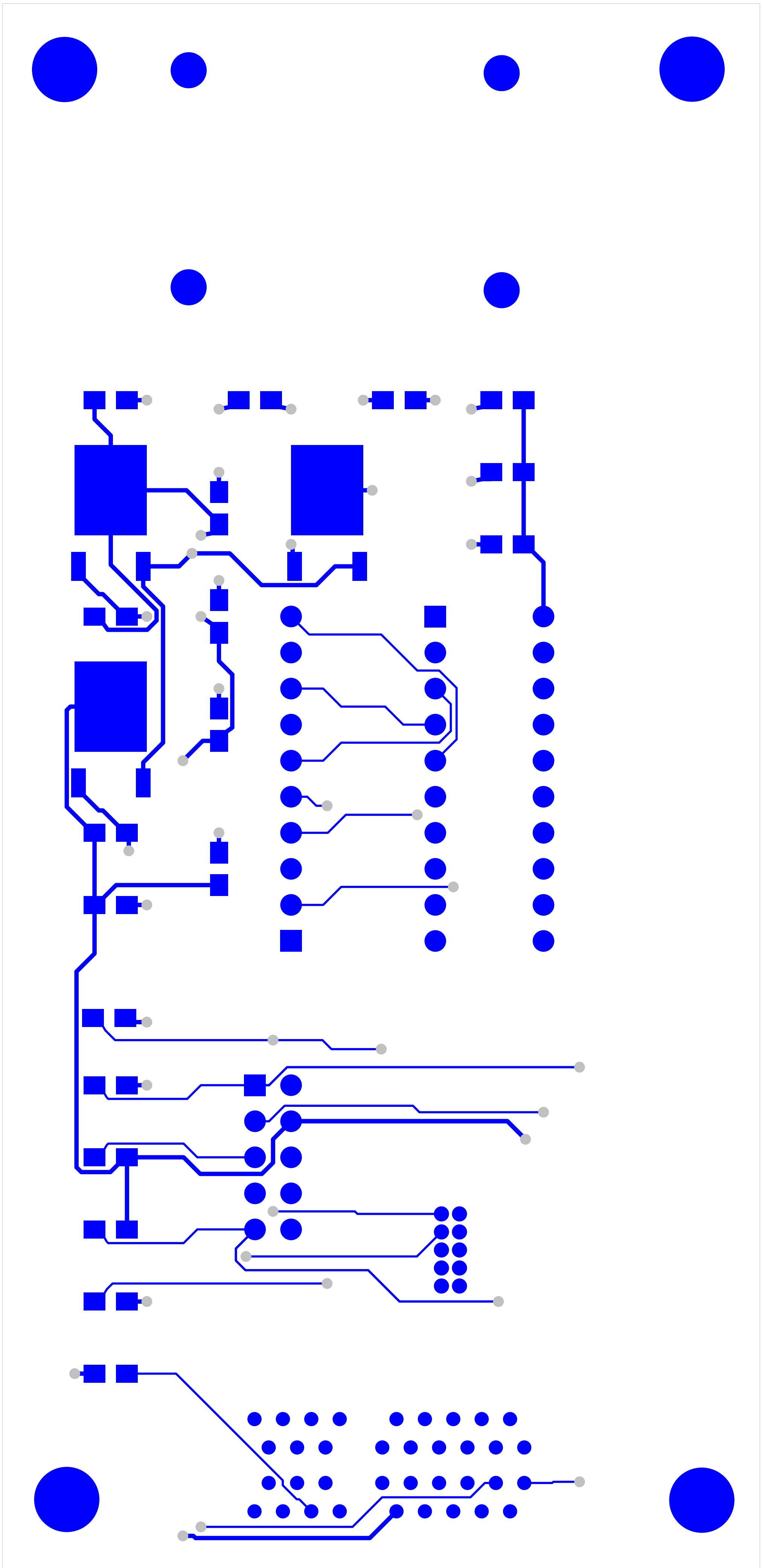


Appendix B: Test Board Schematic and Layout

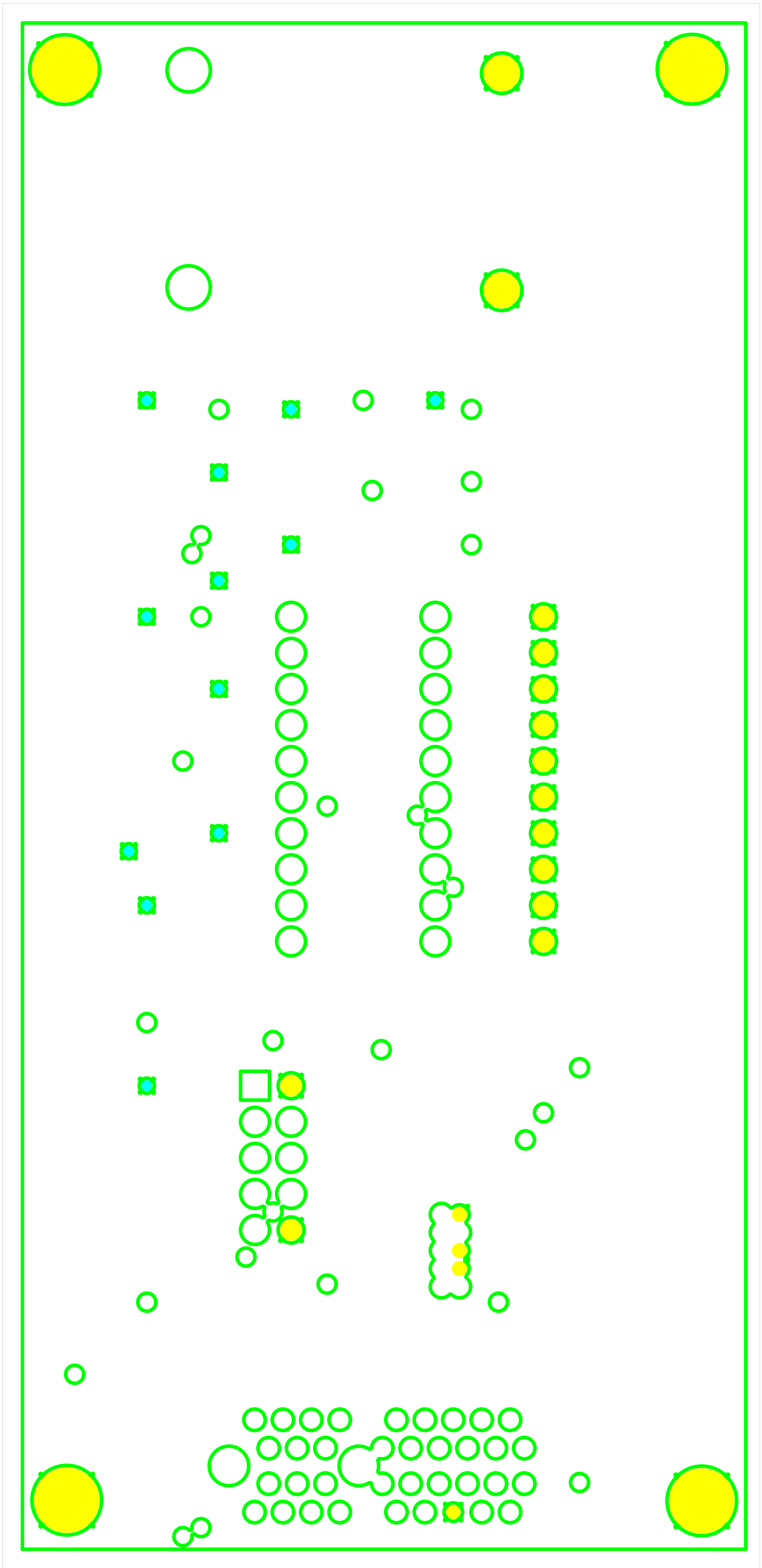


REVISION RECORD
LTR
ECO NO:
APPROVED:
DATE:
STANDOFF1
STANDOFF2
STANDOFF3
STANDOFF4
STANDOFF
GND
TCK
TMS
TDI
TDO
R2
1k
R3
10k
R4
10k
VCC25
VCC33
CONF_DONE
NSTATUS
NCONFIG
GOODCONNECTION
R7-A
R7-B
R7-C
R7-D
R7-E
U9
LED1+
LED1-
LED10
VIN
U1
U2
U3
IN
OUT
ADJ/GND
L11712
L11725
L11733
VCC12
C1 .01uF
C6 .1uF
C7 10uF
C10 .01uF
C11 .1uF
C2 .01uF
C4 .1uF
C5 10uF
C3 .01uF
C8 .1uF
C9 10uF
C12 .01uF
C13 .1uF
JTAGFPGA
HEADER-5X2POL
BV2
BV1
BANANA_PLUG
JTAGFPGA1
PCIE-036-02-X-D-TH
JTAGSAM
SAMTMS
SAMTCK
SAMTDO
SAMTDI
SAMNRST
R1
1k
R6
10k
R5
10k
COMPANY:
HMC Engineering
TITLE:
uMudd Mark V Tester Board
DRAWN:
K. Pezeshki
DATED:
December 5th, 2018
CHECKED:
<Checked By>
<Checked Date>
QUALITY CONTROL:
<QC By>
<QC Date>
RELEASED:
<Released By>
<Release Date>
CODE:
<Code>
SIZE:
B
DRAWING NO:
<Drawing Number>
REV:
<Revision>
SCALE: <Scale>
SHEET: 1 OF 1

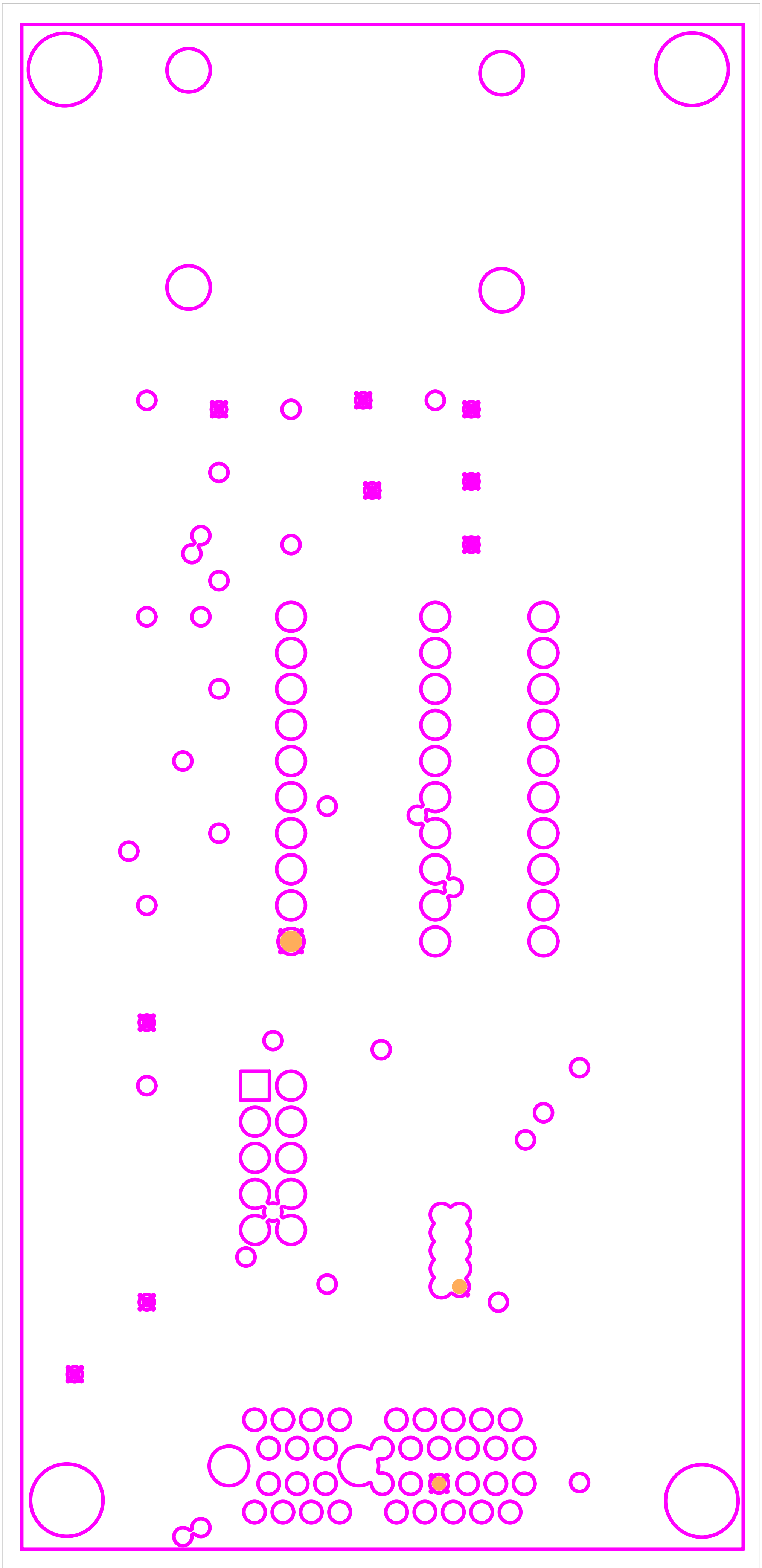
Top Copper



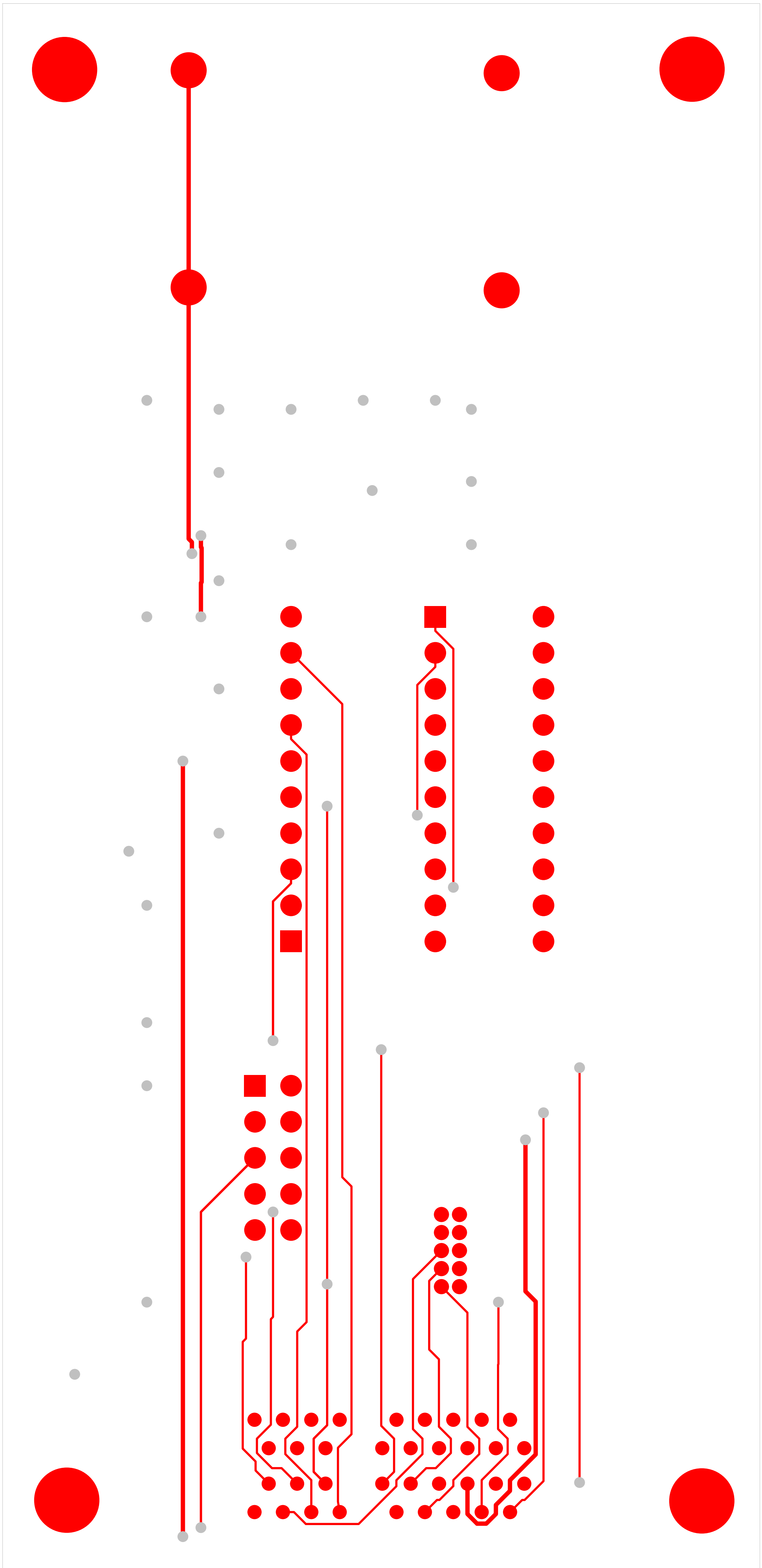
Ground Plane



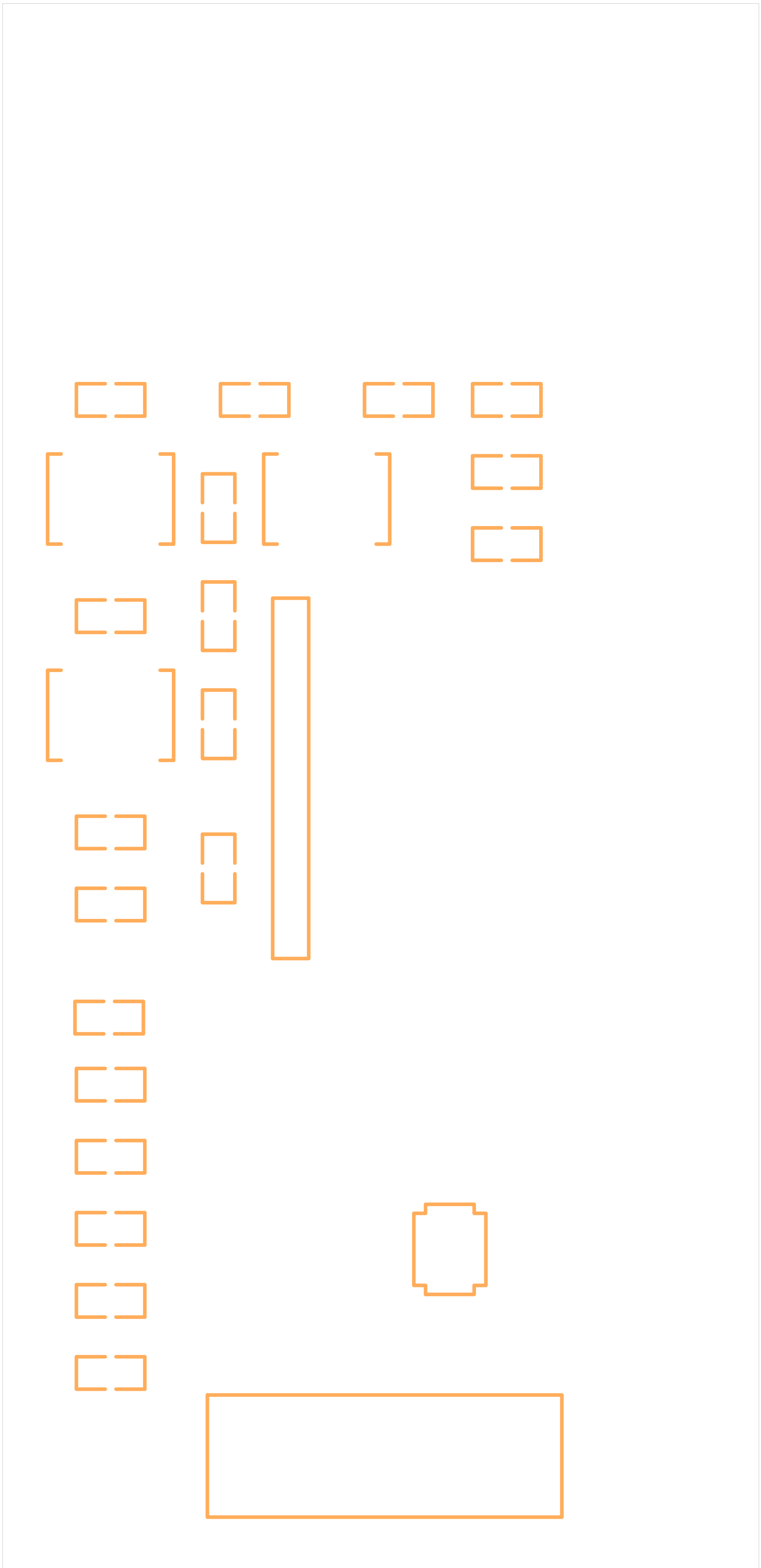
Power Plane



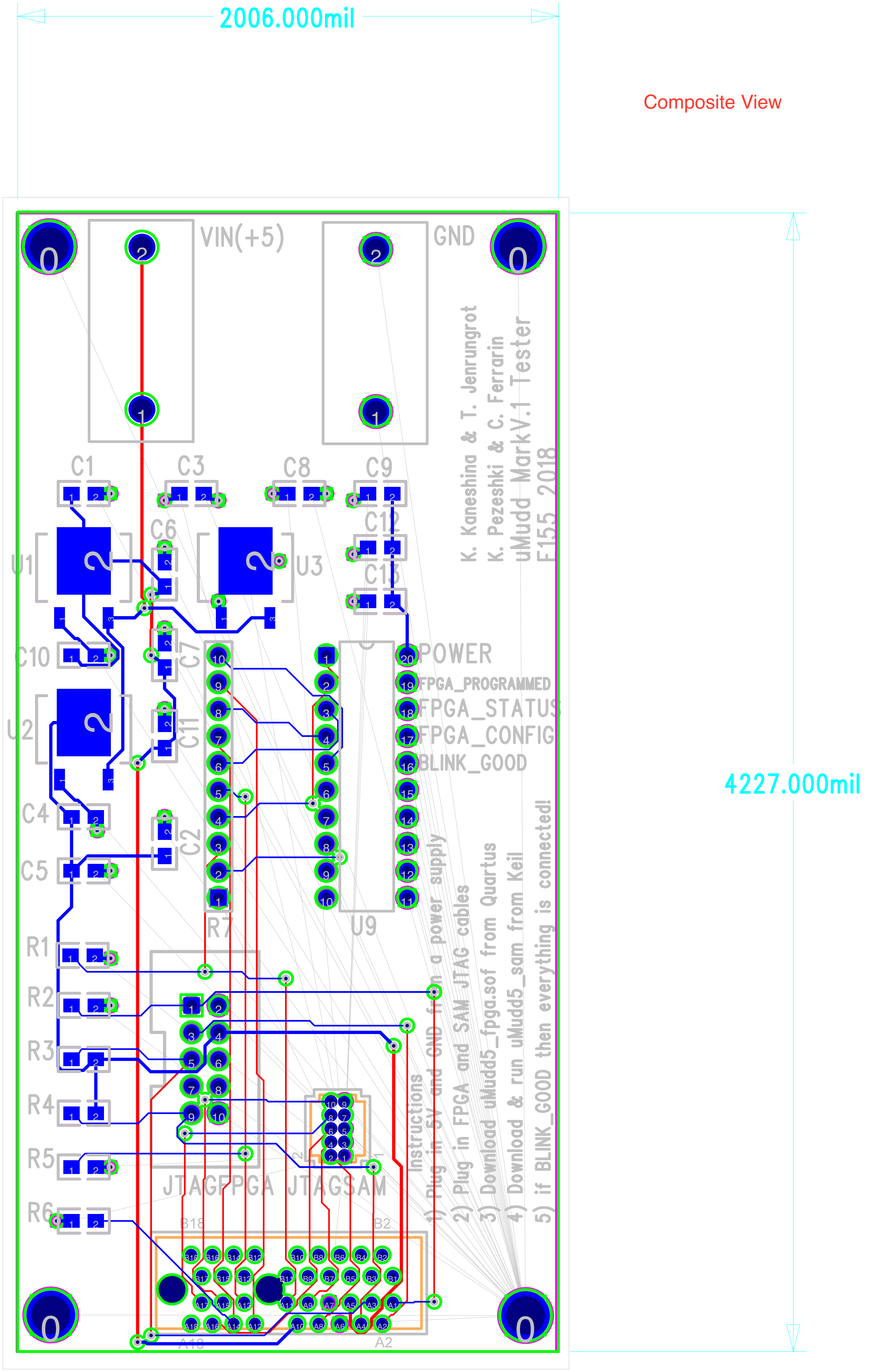
Bottom Copper



Assembly Top



Composite View



Appendix C: Revised Lab 6 Lab Manual

Microprocessor-Based Systems (E155)

D. Harris and M. Spencer **Fall 2018**

Lab 6: Internet of Things

Requirements

Build an internet accessible device to control two LEDs, measure ambient light intensity, and measure three-axis acceleration. Use an ESP8266 with the provided webserver code to host the webpage, and use the onboard µMudd PIO, ADC and SPI peripherals to toggle two LEDs, read light intensity from a phototransistor and to read three-axis acceleration from an SPI LIS3DH accelerometer. An end-user must be able to blink the LEDs, read phototransistor voltage, and read total acceleration from the webpage.

Extra credit is available for improving the website or embedded system in some interesting way. Examples of acceptable improvements include independent control of the LEDs, displaying the current state of the LEDs on the webpage, or implementing another µMudd in a useful way.

At heart, this lab requires you to directly control the memory mapped peripherals on the µMudd ARM processor, so refrain from consulting easySamIO.h or any other C implementations of the SAM4S peripheral set.

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 indulgences, 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.

HTTP is complicated and servicing web service requests takes many steps. Fortunately, the tools necessary to do that are very mature. 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 an 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.

To program the ESP8266, you will need to install (as of 12/11/2018):

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) 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) Open the .ino Arduino program with the Arduino IDE
2) 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.
3) Select the correct board (Tools). We suggest the following settings:
    a. Board: "NodeMCU 1.0 (ESP-12E Module)"
    b. Upload Speed: "115200"
    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"
4) Compile and upload the script with the "Upload" button

On opening the Serial Monitor, you will see a stream of debug messages displaying the current status of the ESP8266.

[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

ESP8266-µMudd Interface:

The ESP8266 program only allows the µMudd to communicate with the outside world. The µMudd must supply a webpage to the ESP8266, and must interpret any client requests to the ESP8266. The devices interface through a 9600 baud serial connection, hosted on pins D8 (TX) and D5 (RX) 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 '<' … '>'. For example, a user accessing the page http://<server_address>/ledon would result in the request '<ledon>' send to the microcontroller. A user accessing the root webpage of the server, http://<server_address>/ would result in the request '<>'
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>'

Note that the ESP8266 does not request a webpage from the µMudd immediately on client request. Instead, it caches a copy of the webpage, and updates the webpage from the µMudd when the server is not processing a client request. At maximum, it will take 10 seconds after the client request to update the webpage, after which the student will have to reload the page in their web browser.

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 LED states as necessary, read from the SPI LIS3DH accelerometer and phototransistor, and use this data to generate a webpage that is transmitted to the ESP8266.

We suggest using the LPT2023 phototransistor to measure ambient light intensity. You will have to design a circuit composed of the LPT2023, a power supply, and a resistor to generate an output voltage that varies with ambient light intensity. We recommend reading the LPT2023 datasheet before starting this design.

You will need to write an HTML webpage that displays dynamic acceleration and voltage data as well as creating requests to change the state of the LEDs. 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.

Credits:

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

Appendix D: ESP8266 ‘Black Box’ Peripheral Code

```
/*
   Kaveh Pezeshki and Christopher Ferrarin
   E155 Lab 6 Generic Webserver

   Hosts a webserver displaying webpages sent over UART, sending back redirect
    URLs to provide a measure of user interaction.

   In more detail:

   1) The webserver starts a 9600 baud serial connection over the hardware UART
    (for debug)
   2) The webserver starts a 9600 baud software serial connection over pins 14
    and 15 (RX, TX)
   3) The webserver connects to a given network, and prints status information
    over the debug UART. By default, this is CINE.
   4) The webserver sends an empty request '<>' to the microcontroller to
    obtain a copy of the webpage
   5) The webserver initializes an HTTP server and a hardware refresh timer
   6) The webserver waits for a request from the client, after which it
    transmits an updated webpage

   steps 4-6 are repeated while the program runs

   In parallel, a hardware timer triggers an interrupt every 10 seconds. This
    will cause a refresh of the webpage from the MCU by
   sending the most recent abbreviated URL to the MCU as "<" + <abbr. url> +
    ">", and waiting for an updated webpage to be returned
   over UART. This only occurs if the following conditions are met:

   1) The hardware timer interrupt has not been handled, occuring when
    refreshWebpage = true
   2) A client has requested data from the webpage after the last refresh of
    the webpage from the MCU. This occurs when webpageUpdated = false

   Abbreviated URL explanation:
   The webserver automatically parses a client request to simplify code on the
    MCU. If the server has example IP address '192.168.1.1', a client request
    may be the URL:
   'http://192.168.1.1/webpage'
   The server will return everything after the IP address and slash. For
    example:
   'http://192.168.1.1/webpage' => '<webpage/>'

   We expect these abbreviated URLs to be under 10 characters
*/

//Importing required libraries
#include <ESP8266WiFi.h>
#include <SoftwareSerial.h>

//defining start and end HTML tags
```

```
const String htmlStart = "<!DOCTYPE html><html>";
const String htmlEnd = "</html>";

//Defining network information
const char* networkName = "CINE"; //set this to the selected network SSID
const char* password    = NULL;   //set this to a non-null value if selected
 network requires authentication
String      ip;                   //stores the current IP. Set in the setup
 function

//Defining the web server and HTTP request variables
WiFiServer server(80);           //The server is accessible over port 80
String     request;              //Stores the client HTTP request
String     parsedRequest = "<>"; //Stores a simplified version of the HTTP
 request to transmit to the MCU
String     currentLine;          //Stores a semi-parsed version of the HTTP
 request
String     webpage = "WAITING FOR DATA";  //The current webpage, updated by the
 MCU

//Defining the softwareSerial interface
SoftwareSerial mcuSerial(14, 15);

extern "C" {
#include "user_interface.h"
}

//defining the webpage refresh timer
os_timer_t refreshTimer;
bool refreshWebpage = false;
bool webpageUpdated = true;

//Timer callback. This function will run when the timer reaches the set webpage
 refresh time
void timerCallback(void *pArg) {
  refreshWebpage = true;
}

//Setup code. Runs once on program execution before loop code
void setup() {
  //starting the debug and MCU serial connections
  Serial.begin(115200);
  mcuSerial.begin(9600);
  Serial.println("Set up serial connections");

  //connecting to WiFi network
  Serial.print("Connecting to network ");
  Serial.println(networkName);
  Serial.print("With password ");
  Serial.println(password);
  WiFi.begin(networkName, password);
```

```
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Attempting connection...");
  }
  //connected to the network. Printing status information
  Serial.print("Connected to WiFi network with IP: ");
  Serial.println(WiFi.localIP());
  ip = ipToString(WiFi.localIP());
  Serial.print("AP IP: ");
  Serial.println(WiFi.softAPIP());

  //starting server
  Serial.println("Starting server");
  server.begin();

  //fetching a new webpage
  receiveWebPage("<>");

  //defining the webpage reload timer
  os_timer_setfn(&refreshTimer, timerCallback, NULL);
  os_timer_arm(&refreshTimer, 10000, true); //fetches a new webpage every 10
   seconds
}

//Main program. Runs repeatedly after setup code
void loop() {

  //we update the webpage every 9 seconds to prevent timeout errors
  if (refreshWebpage & !webpageUpdated) {
    refreshWebpage = false;
    webpageUpdated = true;
    webpage = receiveWebPage(parsedRequest);
  }

  //Wait for a new connection
  WiFiClient webClient = server.available();
  //If a client has connected, we wait for a request
  if (webClient) {
    webpageUpdated = false;
    currentLine = "";
    Serial.println("\nClient Connected");
    while (webClient.connected()) {
      //Reading available bytes from the client if available
      if (webClient.available()) {
        char byteIn = webClient.read();
        request += byteIn;

        //if the line is only a line feed, we have reached the end of the
         client request and will therefore send a response
        //This requires sending a request for a new webpage to the MCU
        if (byteIn == '\n') {
```

```
          if (currentLine.length() == 0) {
            //transmitting the response
            //transmitting HTTP header and content type
            Serial.println("Transmitting webpage");
            webClient.println("HTTP/1.1 200 OK");
            webClient.println("Content-type:text/html");
            webClient.println("Connection: close");
            webClient.println();
            //transmitting the full webpage
            webClient.println(webpage);
            //transmitting an extra newline to catch transmission termination
             errors
            //webClient.println();
            break; //disconnect from the client by breaking from while loop
          }
          else {
            currentLine = "";
          }
        }
        else if (byteIn != '\r') {
          currentLine += byteIn;
        }
      }
    }
    //ending the transaction
    if (parseRequest(request) != "<>") {
      parsedRequest = parseRequest(request);
    }
    request = "";
    webClient.stop();
    Serial.println("Client disconnected");
  }
}

//Converts an IP address to a String
String ipToString(IPAddress address)
{
  return String(address[0]) + "." +
         String(address[1]) + "." +
         String(address[2]) + "." +
         String(address[3]);
}

//Parses an input http request as specified in "Abbreviated URL Explanation"
String parseRequest(String request) {
  Serial.print("////START Received Request: ");
  Serial.println(request);
  Serial.println("////END Received Request: ");

  //favicon is a common icon formatting scheme that tends
  if (request.indexOf("favicon.ico") != -1) return "<>";
```

```
  int getLocation = request.indexOf("GET /");
  int httpLocation = request.indexOf(" HTTP");

  String parsedRequest = "<" + request.substring(getLocation + 5, httpLocation)
   + ">";

  Serial.print("Reduced URL: '");
  Serial.print(parsedRequest);
  Serial.println("'");
  return parsedRequest;
}

int substringInString(String haystack, String needle) {
  for (int i = 0; i < haystack.length()-needle.length(); i++) {
    bool foundsubString = true;
    for(int j = 0; j < needle.length(); j++) {
      if( haystack[i+j] != needle[j]) {
        foundsubString = false;
      }
    }
    if (foundsubString) {return true;}
  }
  return false;
}

//Sends parsedRequest over UART to the MCU and waits for a complete webpage to
 be returned
String receiveWebPage(String parsedRequestIn) {
  Serial.println("////START Received Web Page");
  webpage = ""; //clear webpage in preparation for new webpage to be
   transmitted
  bool webpageReceived = false;
  bool startReceived = false;
  bool endReceived = false;

  //transmitting the parsed request from the client
  Serial.print("Transmitting:");
  Serial.println(parsedRequestIn);
  mcuSerial.print(parsedRequestIn);

  //wait until the entire webpage has been received
  while (!webpageReceived) {
    ESP.wdtFeed(); //resetting watchdog timer
    //checking for new serial data, adding to website
    while (mcuSerial.available()) {
      char newData = mcuSerial.read();
      webpage.concat(newData);;
    startReceived = (webpage.indexOf(htmlStart) != -1);
    endReceived = (webpage.indexOf(htmlEnd) != -1);
    webpageReceived = startReceived && endReceived;
```

```
    }
  }
  Serial.println("Received webpage");
  Serial.println(webpage);
  Serial.println("////END Received Web Page");
  return webpage;
}
```

Appendix E: Lab 6 Demo Code

```
/*
 * lab6Demo.c
 *
 * Created: 12/9/2018 7:21:32 PM
 * Author : Kaveh Pezeshki and Christopher Ferrarin
 *
 * MCU backend for Lab 6. Generates webpages as requested by the ESP8266, and
  interfaces with the onboard PIO and ADC peripherals to allow user control of
  an LED and to display the voltage on ADC channel 2.
 *
 * The LED on PA17 is controlled by the webpage, active low
 * The LED on PA18 is active when the webpage is transmitted to the ESP8266
  from the MCU
 *
 * In more detail:
 * 1) The MCU initializes peripherals
 * 2) The MCU loads any incoming bytes from UART into a buffer, and scans for
  the sequence '< ... >' Held within the angle brackets is the request from the
  microcontroller. Steps 3-5 are executed if a request is detected
 * 3) The MCU turns on PA17 if 'on' is within the request, and turns off PA17
  if 'off' is within the request
 * 4) The MCU reads the CH2 ADC voltage
 * 5) The MCU generates and sends the webpage to the ESP8266 over UART
 */

#include "easySamIO.h"   //peripheral header file
#include <string.h>       //string operations for parsing incoming requests
#include <stdio.h>        //float -> string conversion for ADC
#define LED_PIN 17        //LED controlled by webpage. Active low
#define TRANSMIT_PIN 18   //LED indicated webpage transmission. Active low
#define VOLTAGE_CHARS_TO_TRANSMIT 5 //The number of characters in the string
 representation of the CH2 voltage to transmit as a part of the webpage


//The webpage is separated into a 'start' and 'end' section. These sandwich the
 CH2 voltage, which is inserted between the two webpage arrays. The webpage is
 raw HTML
//Note: as " terminates the string, we escape the " with \". This is
 interpreted as the raw character '"' rather than as an escape character
const char* webpageStart = "<!DOCTYPE html><html>\n    <head>\n
 <title>E155 Web Server Demo Webpage</title>\n            </head>\n    <body>\n
 <h1>E155 Web Server Demo Webpage</h1>\n        <p>Current Microcontroller ADC:
 </p>\n        ";
//ADC CH2 voltage is printed between these
const char* webpageEnd   = "\n        <p>LED Control:</p>\n        <form
 action=\"on\">\n            <input type=\"submit\" value=\"Turn the LED on!
 \" />\n        </form>\n        <form action=\"off\">\n                <input
 type=\"submit\" value=\"Turn the LED off!\" />\n        </form>\n    </
 body>\n</html>\n";
```

```
//as we do not dynamically calculate webpage size for the constant start and
 end arrays, it is given as constant
const int webpageStartChars = 215;
const int webpageEndChars = 264;


void transmitWebpage() {
    digitalWrite(TRANSMIT_PIN, LOW);
    //first transmitting the initial section of the webpage
    for (int charCount = 0; charCount < webpageStartChars; charCount++) {
        uartTx(webpageStart[charCount]);
        if (webpageStart[charCount] == '\n') {uartTx('\r');} //some
         interpreters want a carriage return and line feed. Adding a carriage
         return if line feed is detected
    }

    //reading the ADC
    float ch2Voltage;
    char ch2VoltageStr[VOLTAGE_CHARS_TO_TRANSMIT];
    ch2Voltage = adcRead(CH2);
    //converting ADC voltage as a float to a string
    snprintf(ch2VoltageStr, VOLTAGE_CHARS_TO_TRANSMIT, "%f", ch2Voltage);
    //Transmitting the voltage string to the webpage. Note: snprintf transmits
     a null terminator as its last character. This breaks many string parsing
     functions, so we do not transmit it.
    for (int charCount = 0; charCount < VOLTAGE_CHARS_TO_TRANSMIT-1; charCount+
     +) {
        uartTx(ch2VoltageStr[charCount]);
    }
    //finally transmitting the final section of the webpage
    for (int charCount = 0; charCount < webpageEndChars; charCount++) {
        uartTx(webpageEnd[charCount]);
        if (webpageEnd[charCount] == '\n') {uartTx('\r');}
    }
    digitalWrite(TRANSMIT_PIN, HIGH);
}


int main(void) {
    /* Initialize the SAM system */
    samInit(); //peripheral initialization. See easySamIO.h
    pinMode(LED_PIN, OUTPUT); //LED_PIN is controlled by the webpage
    digitalWrite(LED_PIN, HIGH);
    pinMode(TRANSMIT_PIN, OUTPUT); //TRANSMIT_PIN is active when a webpage is
     being transmitted
    digitalWrite(TRANSMIT_PIN, HIGH);
    uartInit(4, 25); //initializing the UART with no parity, 9600 baud
    adcInit(ADC_BITS_12); //initializing the ADC with a precision of 12 bits
    adcChannelInit(CH2, ADC_GAIN_X1, ADC_OFFSET_OFF); //initializing channel 2
     of the ADC with no gain or offset
```

```
int currentRxState = 0;                 //1 when there is an unread byte in
 the UART RX register, 0 otherwise
char character;                         //character read by the UART
char request[14] = "               ";  //14-character buffer to store the
 webpage request
int requestFound = 0;                   //0 only if '<...>' not in the
 request
int currentRequestChar = 0;             //The number of valid characters in
 request
const char requestStart = 0x3c;         //hex representation of character
 '<'
const char requestEnd = 0x3e;           //hex representation of character
 '>'

transmitWebpage();                      //on boot, transmit the webpage.
 This mitigates any potential request - response timing errors on system
 initialization



while (1)
{
    //checking whether there is a byte to read in the UART RX buffer
    currentRxState = UART_REGS->UART_SR.RXRDY;
    //process if there is an unread byte
    if (currentRxState == 1) {
        if(currentRequestChar == 14) {
            //if buffer is filled, wrap to start
            currentRequestChar = 0;
        }
        //read in unread character
        character = uartRx();
        //add character to buffer
        request[currentRequestChar] = character;
        //searching for substring
        int startInString = strchr(request, requestStart); //0 only if '<'
         not in the request
        int endInString   = strchr(request, requestEnd); //0 only if '>'
         not in the request
        //if a request start character is past the start of the buffer,
         empty the buffer with the start character in position 0
        if (startInString != 0 && currentRequestChar >= 2 && request[0] !=
         requestStart) { //SHOULD THIS BE >=1??
            request[0] = '<';
            for (int i = 1; i < 14; i++) {
                request[i] = ' ';
            }
            currentRequestChar = 0;
        }
        //if the buffer contains both the start and end characters, then
         the request is loaded. Process the request and return a webpage
```

```
            if (startInString != 0 && endInString != 0) {
                int ledOnInString = strstr(request, "on");    //Turn LED on if
                 'on' is in the request
                int ledOffInString = strstr(request, "off");  //Turn LED on if
                 'off' is in the request
                if(ledOnInString != 0) {
                    digitalWrite(LED_PIN, LOW);
                }
                if(ledOffInString != 0) {
                    digitalWrite(LED_PIN, HIGH);
                }
                //the request has been processed, and is therefore cleared
                for (int i = 0; i < 14; i++) {
                    request[i] = ' ';
                }
                //finally, transmitting the webpage
                transmitWebpage();
            }
            currentRequestChar += 1; //preparing to write in the next buffer
             index
        }

    }
}
```

Appendix F: Peripheral Driver Code

```
/* SAM4S4B.h
 *
 * cferrarin@g.hmc.edu
 * kpezeshki@g.hmc.edu
 * 12/11/2018
 *
 * Top-level device driver for the SAM4S4B microcontroller.
 *
 * It is recommended to read the SAM4SB datasheet to understand the peripherals
  in this device
 * driver:
 * http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-11100-32-bit%20Cortex-
  M4-Microcontroller-SAM4S_Datasheet.pdf
 *
 * This device driver provides minimal working support for the following
  peripherals:
 *   -- PMC  (Power Management Controller):
 *      -- For clock multiplexing to peripherals and controlling programmable
  clocks.
 *   -- PIO  (Parallel Input/Output Controller):
 *      -- For peripheral function pin multiplexing and reading and writing
  digital values from pins.
 *   -- TC   (Timer Counter):
 *      -- For system delays and counting and triggering at various clock
  speeds.
 *   -- SPI  (Serial Peripheral Interface):
 *      -- For serial communication with external devices that support SPI.
 *   -- UART (Universal Asynchronous Receiver-Transmitter):
 *      -- For serial communication with external devices that support UART.
 *   -- PWM  (Pulse Width Modulation Controller):
 *      -- For generating square waves of various frequencies and duty cycles.
 *   -- ADC  (Analog-to-Digital Converter):
 *      -- For reading analog voltages.
 *   -- RTC  (Real Time Clock):
 *      -- For automatic tracking of the time and date.
 *
 * Registers in this file are organized into structs in the following chain:
 *   -- Peripheral Struct (e.g. PCM, SPI) (in the case of PIO and TC, there are
  multiple)
 *      -- Channel Struct (e.g. TC_CH[k], PWM_CH[k]) (not always defined)
 *         -- Bit Field Struct (of type <Peripheral>_<Register>_bits struct)
  (not always defined)
 *         -- Register Struct (of type uint32_t struct)
 * The following are examples of how to access members of these structs:
 *   -- Access a register of a peripheral with no channels:
 *      <Peripheral Struct>-><Register Struct>
 *      Example: PIOA->PIO_PER
 *   -- Access a register of a peripheral with channels:
 *      <Peripheral Struct>->><Channel Struct[<Channel Number>]>.<Register
  Struct>
 *      Example: TC->TC_CH[2].TC_CV
```

```
 *   -- Access a bit of a peripheral with no channels:
 *      <Peripheral Struct>-><Bit Field Struct>.<Bit Name>
 *      Example: PMC->PMC_SCER.PCK2
 *   -- Access a bit of a peripheral with channels:
 *      <Peripheral Struct>-><Channel Struct[<Channel Number>]>.<Bit Field
  Struct>.<Bit Name>
 *      Example: TC->TC_CH.TC_CCR.CLKEN
 *
 * The main clock for peripherals is rated at 4 MHz but utilizes an RC
  oscillator, which is cheap
 * and consumes little power but can be inaccurate. As such, it is necessary to
  verify the clock's
 * frequency. This can be done by running the FPGA clock with samInit():
 *    #include "SAM4S4B.h"
 *    int main() {
 *        samInit();
 *    }
 * Observe pin PIO_PA31 and record its frequency. This will be MCK_FREQ divided
  by four, so multiply
 * the value by 4 and record this accurate MCK frequency in the #define
  directive in SAM4S4B_sys.h.
 *
 * Start your main.c file with the following lines:
 *    #include "SAM4S4B.h"
 *    int main() {
 *        samInit();
 *        // Your code goes here
 *    }
 * Remember to intialize each peripheral with its init function before using it
  (although PIO is
 * intialized automatically through samInit()), and enjoy!
 */

#ifndef SAM4S4B_H
#define SAM4S4B_H

#include "SAM4S4B_sys.h"
#include "SAM4S4B_pmc.h"
#include "SAM4S4B_pio.h"
#include "SAM4S4B_tc.h"
#include "SAM4S4B_spi.h"
#include "SAM4S4B_uart.h"
#include "SAM4S4B_pwm.h"
#include "SAM4S4B_adc.h"
#include "SAM4S4B_rtc.h"

////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// Top-Level Functions
////////////////////////////////////////////////////////////////////////////////
 /////////////////////
```

```
// Sets up the clock for the FPGA at 1 MHz
void samInit() {
    pioInit();
    pioPinMode(PIO_PA31, PIO_PERIPH_B);
    pmcPCK2Init();
}


#endif
```

```c
/* SAM4S4B_sys.h
 *
 * cferrarin@g.hmc.edu
 * kpezeshki@g.hmc.edu
 * 12/11/2018
 *
 * Contains top-level system definitions for the SAM4S4B microcontroller.
 */

#ifndef SAM4S4B_SYS_H
#define SAM4S4B_SYS_H

#define ADC_VREF 3.3 // Voltage supplied at the Vref pin (in V)
#define MCK_FREQ 3578000 // Frequency of Master Clock (in Hz)

#endif
```

```
/* SAM4S4B_uart.h
 *
 * cferrarin@g.hmc.edu
 * kpezeshki@g.hmc.edu
 * 12/11/2018
 *
 * Contains base address locations, register structs, definitions, and
  functions for the UART
 * (Universal Asynchronous Receiver-Transmitter) peripheral of the SAM4S4B
  microcontroller. */

#ifndef SAM4S4B_UART_H
#define SAM4S4B_UART_H

#include <stdint.h>
#include "SAM4S4B_pio.h"

////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// UART Base Address Definitions
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

#define UART0_BASE (0x400E0600U) // UART0 Base Address


////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// UART Registers
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

// Bit field struct for the UART_CR register
typedef struct {
    volatile uint32_t        : 2;
    volatile uint32_t RSTRX  : 1;
    volatile uint32_t RSTTX  : 1;
    volatile uint32_t RXEN   : 1;
    volatile uint32_t RXDIS  : 1;
    volatile uint32_t TXEN   : 1;
    volatile uint32_t TXDIS  : 1;
    volatile uint32_t RSTSTA : 1;
    volatile uint32_t        : 23;
} UART_CR_bits;

// Bit field struct for the UART_MR register
typedef struct {
    volatile uint32_t        : 9;
    volatile uint32_t PAR    : 3;
    volatile uint32_t        : 2;
    volatile uint32_t CHMODE : 2;
```

```
    volatile uint32_t        : 16;
} UART_MR_bits;

// Bit field struct for the UART_SR register
typedef struct {
    volatile uint32_t RXRDY   : 1;
    volatile uint32_t TXRDY   : 1;
    volatile uint32_t         : 1;
    volatile uint32_t ENDRX   : 1;
    volatile uint32_t ENDTX   : 1;
    volatile uint32_t OVRE    : 1;
    volatile uint32_t FRAME   : 1;
    volatile uint32_t PARE    : 1;
    volatile uint32_t         : 1;
    volatile uint32_t TXEMPTY : 1;
    volatile uint32_t         : 1;
    volatile uint32_t TXBUFE  : 1;
    volatile uint32_t RXBUFE  : 1;
    volatile uint32_t         : 19;
} UART_SR_bits;

// Peripheral struct for the UART peripheral
typedef struct {
    volatile UART_CR_bits UART_CR;       // (Uart Offset: 0x0000) Control
     Register
    volatile UART_MR_bits UART_MR;       // (Uart Offset: 0x0004) Mode Register
    volatile uint32_t     UART_IER;      // (Uart Offset: 0x0008) Interrupt
     Enable Register
    volatile uint32_t     UART_IDR;      // (Uart Offset: 0x000C) Interrupt
     Disable Register
    volatile uint32_t     UART_IMR;      // (Uart Offset: 0x0010) Interrupt
     Mask Register
    volatile UART_SR_bits UART_SR;       // (Uart Offset: 0x0014) Status
     Register
    volatile uint32_t     UART_RHR;      // (Uart Offset: 0x0018) Receive
     Holding Register
    volatile uint32_t     UART_THR;      // (Uart Offset: 0x001C) Transmit
     Holding Register
    volatile uint32_t     UART_BRGR;     // (Uart Offset: 0x0020) Baud Rate
     Generator Register
    volatile uint32_t     Reserved1[55];
    volatile uint32_t     UART_RPR;      // (Uart Offset: 0x100) Receive
     Pointer Register
    volatile uint32_t     UART_RCR;      // (Uart Offset: 0x104) Receive
     Counter Register
    volatile uint32_t     UART_TPR;      // (Uart Offset: 0x108) Transmit
     Pointer Register
    volatile uint32_t     UART_TCR;      // (Uart Offset: 0x10C) Transmit
     Counter Register
    volatile uint32_t     UART_RNPR;     // (Uart Offset: 0x110) Receive Next
     Pointer Register
```

```
    volatile uint32_t     UART_RNCR;     // (Uart Offset: 0x114) Receive Next
     Counter Register
    volatile uint32_t     UART_TNPR;     // (Uart Offset: 0x118) Transmit Next
     Pointer Register
    volatile uint32_t     UART_TNCR;     // (Uart Offset: 0x11C) Transmit Next
     Counter Register
    volatile uint32_t     UART_PTCR;     // (Uart Offset: 0x120) Transfer
     Control Register
    volatile uint32_t     UART_PTSR;     // (Uart Offset: 0x124) Transfer
     Status Register
} Uart;

// Pointer to a Uart-sized chunk of memory at the UART peripheral
#define UART ((Uart*) UART0_BASE)


////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// UART Definitions
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

// Values which the PAR bits in the UART_MR register can take on
#define UART_MR_PAR_EVEN  0 // Even parity
#define UART_MR_PAR_ODD   1 // Odd parity
#define UART_MR_PAR_SPACE 2 // Parity forced to 0
#define UART_MR_PAR_MARK  3 // Parity forced to 1
#define UART_MR_PAR_NO    4 // No parity

// The specific PIO pins and peripheral function which UART uses, set in
 uartInit()
#define UART_URXD0_PIN PIO_PA9
#define UART_ITXD0_PIN PIO_PA10
#define UART_FUNC      PIO_PERIPH_A

// (UART does not have write protection.)


////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// UART Functions
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

/* Enables the UART peripheral and initializes its parity and baut rate.
 *    -- parity:  A UART parity ID, e.g. UART_MR_PAR_SPACE
 *    -- CD: a 16-bit unsigned integer which determines the baud rate as
  follows:
 *       Baud Rate = MCK_FREQ/(16*CD)
 * Note that pin PA9 is used as receive and pin PA10 is used as transmit.
  pioInit() must be called
```

```
 * first. */
void uartInit(uint32_t parity, uint16_t CD) {
    pmcEnablePeriph(PMC_ID_UART0);

    pioPinMode(UART_URXD0_PIN, UART_FUNC); // Set URXD0 pin mode
    pioPinMode(UART_ITXD0_PIN, UART_FUNC); // Set ITXD0 pin mode

    UART->UART_CR.TXEN = 1; // Enable transmitter
    UART->UART_CR.RXEN = 1; // Enable receiver

    UART->UART_MR.PAR = parity; // Set parity
    UART->UART_BRGR   = CD; // Set baud rate divisor
}

/* Transmits a character (1 byte) over UART
 *    -- data: the character to send over UART */
void uartTx(char data) {
    while (!(UART->UART_SR.TXRDY)); // Wait until previous data has been
     transmitted
    UART->UART_THR = data; // Write data into holding register for transmit
}

/* Receives a character (1 byte) over UART
 *    -- return: the character received over UART */
char uartRx() {
    while (!(UART->UART_SR.RXRDY)); // Wait until data has been received
    return (char) UART->UART_RHR; // Return received data in holding register
}


#endif
```

```c
/* SAM4S4B_tc.h
 *
 * cferrarin@g.hmc.edu
 * kpezeshki@g.hmc.edu
 * 12/11/2018
 *
 * Contains base address locations, register structs, definitions, and
  functions for the TC (Timer
 * Counter) peripheral of the SAM4S4B microcontroller. */

#ifndef SAM4S4B_TC_H
#define SAM4S4B_TC_H

#include <stdint.h>
#include "SAM4S4B_sys.h"

////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// TC Base Address Definitions
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

#define TC0_BASE   (0x40010000U) // TC0 Base Address
#define TC1_BASE   (0x40014000U) // TC1 Base Address


////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// TC Registers
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

// Bit field struct for the TC_CCR register
typedef struct {
    volatile uint32_t CLKEN   : 1;
    volatile uint32_t CLKDIS  : 1;
    volatile uint32_t SWTRG   : 1;
    volatile uint32_t         : 29;
} TC_CCR_bits;

// Bit field struct for the TC_CMR register
typedef struct {
    volatile uint32_t TCCLKS  : 3;
    volatile uint32_t CLKI    : 1;
    volatile uint32_t BURST   : 2;
    volatile uint32_t CPCSTOP : 1;
    volatile uint32_t CPCDIS  : 1;
    volatile uint32_t EEVTEDG : 2;
    volatile uint32_t EEVT    : 2;
    volatile uint32_t ENETRG  : 1;
    volatile uint32_t WAVESEL : 2;
```

```
    volatile uint32_t WAVE    : 1;
    volatile uint32_t ACPA    : 2;
    volatile uint32_t ACPC    : 2;
    volatile uint32_t AEEVT   : 2;
    volatile uint32_t ASWTRG  : 2;
    volatile uint32_t BCPB    : 2;
    volatile uint32_t BCPC    : 2;
    volatile uint32_t BEEVT   : 2;
    volatile uint32_t BSWTRG  : 2;
} TC_CMR_bits;

// Bit field struct for the TC_SR register
typedef struct {
    volatile uint32_t COVFS   : 1;
    volatile uint32_t LOVRS   : 1;
    volatile uint32_t CPAS    : 1;
    volatile uint32_t CPBS    : 1;
    volatile uint32_t CPCS    : 1;
    volatile uint32_t LDRAS   : 1;
    volatile uint32_t LDRBS   : 1;
    volatile uint32_t ETRGS   : 1;
    volatile uint32_t         : 8;
    volatile uint32_t CLKSTA  : 1;
    volatile uint32_t MTIOA   : 1;
    volatile uint32_t MTIOB   : 1;
    volatile uint32_t         : 13;
} TC_SR_bits;

// Channel struct for each of the 3 TC channels
typedef struct {
    volatile TC_CCR_bits TC_CCR;        // (TcChannel Offset: 0x0) Channel
     Control Register
    volatile TC_CMR_bits TC_CMR;        // (TcChannel Offset: 0x4) Channel Mode
     Register
    volatile uint32_t    TC_SMMR;       // (TcChannel Offset: 0x8) Stepper
     Motor Mode Register
    volatile uint32_t    Reserved1[1];
    volatile uint32_t    TC_CV;         // (TcChannel Offset: 0x10) Counter
     Value
    volatile uint32_t    TC_RA;         // (TcChannel Offset: 0x14) Register A
    volatile uint32_t    TC_RB;         // (TcChannel Offset: 0x18) Register B
    volatile uint32_t    TC_RC;         // (TcChannel Offset: 0x1C) Register C
    volatile TC_SR_bits  TC_SR;         // (TcChannel Offset: 0x20) Status
     Register
    volatile uint32_t    TC_IER;        // (TcChannel Offset: 0x24) Interrupt
     Enable Register
    volatile uint32_t    TC_IDR;        // (TcChannel Offset: 0x28) Interrupt
     Disable Register
    volatile uint32_t    TC_IMR;        // (TcChannel Offset: 0x2C) Interrupt
     Mask Register
    volatile uint32_t    Reserved2[4];
```

```
} TcCh;

#define TC_CH_NUMBER 3 // Number of TC channels
// Peripheral struct for a TC peripheral (either TC0 or TC1)
typedef struct {
    TcCh              TC_CH[TC_CH_NUMBER]; // (Tc Offset: 0x0) channel = 0 .. 2
    volatile uint32_t TC_BCR;              // (Tc Offset: 0xC0) Block Control
     Register
    volatile uint32_t TC_BMR;              // (Tc Offset: 0xC4) Block Mode
     Register
    volatile uint32_t TC_QIER;             // (Tc Offset: 0xC8) QDEC Interrupt
     Enable Register
    volatile uint32_t TC_QIDR;             // (Tc Offset: 0xCC) QDEC Interrupt
     Disable Register
    volatile uint32_t TC_QIMR;             // (Tc Offset: 0xD0) QDEC Interrupt
     Mask Register
    volatile uint32_t TC_QISR;             // (Tc Offset: 0xD4) QDEC Interrupt
     Status Register
    volatile uint32_t TC_FMR;              // (Tc Offset: 0xD8) Fault Mode
     Register
    volatile uint32_t Reserved1[2];
    volatile uint32_t TC_WPMR;             // (Tc Offset: 0xE4) Write Protect
     Mode Register
} Tc;

// Pointers to Tc-sized chunks of memory at each TC peripheral
#define TC0 ((Tc*) TC0_BASE)
#define TC1 ((Tc*) TC1_BASE)


////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// TC Definitions
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

// Clock speeds for the 5 TC clocks used in generating delays
#define TC_CLK1_SPEED (MCK_FREQ / 2)
#define TC_CLK2_SPEED (MCK_FREQ / 8)
#define TC_CLK3_SPEED (MCK_FREQ / 32)
#define TC_CLK4_SPEED (MCK_FREQ / 128)
#define TC_CLK5_SPEED 32000

// Values which TCCLKS bits can take on in TC_CMR
#define TC_CLK1_ID 0
#define TC_CLK2_ID 1
#define TC_CLK3_ID 2
#define TC_CLK4_ID 3
#define TC_CLK5_ID 4

// Arbitrary block IDs used to easily find a channel's block
```

```
#define TC_BLOCK0_ID 0
#define TC_BLOCK1_ID 1

// Values which "channelID" can take on in several functions
#define TC_CH0_ID 0
#define TC_CH1_ID 1
#define TC_CH2_ID 2
#define TC_CH3_ID 3
#define TC_CH4_ID 4
#define TC_CH5_ID 5

// Values which the WAVESEL bits can take on in TC_CMR
#define TC_MODE_UP        0 // The counter increases then resets low once it
 caps out
#define TC_MODE_UPDOWN    1 // The counter increases then decreases once it
 caps out
#define TC_MODE_UP_RC     2 // The counter increases then resets low when an RC
 match occurs
#define TC_MODE_UPDOWN_RC 3 // The counter increases then decreases when an RC
 match occurs

// Writing any other value in this field aborts the write operation of the WPEN
 bit.
// Always reads as 0.
#define TC_WPMR_WPKEY_PASSWD (0x54494Du << 8)


////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// TC Functions
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

/* Initializes the TC peripheral by enabling the Master Clock to TC0 and TC1.
 */
void tcInit() {
    pmcEnablePeriph(PMC_ID_TC0);
    pmcEnablePeriph(PMC_ID_TC1);
}

/* Returns the TC block ID that corresponds to a given channel.
 *    -- channelID: a TC channel ID, e.g. TC_CH3_ID
 *    -- return: a TC block ID, e.g. TC_BLOCK1_ID */
int tcChannelToBlock(int channelID) {
    return channelID / 3;
}

/* Returns a pointer to the given block's base address.
 *    -- block: a TC block ID, e.g. TC_BLOCK1_ID
 *    -- return: a pointer to a Tc-sized block of memory at the block "block"
  */
```

```
Tc* tcBlockToBlockBase(int block) {
    return (block ? TC1 : TC0);
}

/* Given a channel, returns a pointer to the corresponding block's base
 address.
 *    -- channelID: a TC channel ID, e.g. TC_CH3_ID
 *    -- return: a pointer to a Tc-sized block of memory at the block "block"
  */
Tc* tcChannelToBlockBase(int channelID) {
    return tcBlockToBlockBase(tcChannelToBlock(channelID));
}

/* Enables a TC channel and configures it with the desired clock and mode.
 *    -- channelID: a TC channel ID, e.g. TC_CH3_ID
 *    -- clock: a TC clock ID, e.g. TC_CLK3_ID
 *    -- mode: a TC mode ID, e.g. TC_MODE_UP_RC */
void tcChannelInit(int channelID, uint32_t clock, uint32_t mode) {
    Tc* block = tcChannelToBlockBase(channelID);
    int chInd = channelID % TC_CH_NUMBER;
    block->TC_CH[chInd].TC_CCR.CLKEN   = 1;     // Enable clock
    block->TC_CH[chInd].TC_CMR.TCCLKS  = clock; // Set clock to desired clock
    block->TC_CH[chInd].TC_CMR.WAVE    = 1;     // Waveform mode
    block->TC_CH[chInd].TC_CMR.WAVESEL = mode;  // Set counting mode to desired
     mode
}

/* Configures TC Channel 0 to perform delays using the fastest clock and RC
 compares. */
void tcDelayInit() {
    tcChannelInit(TC_CH0_ID, TC_CLK1_ID, TC_MODE_UP_RC);
}

/* Reads the current value of the counter of a given channel.
 *    -- channel ID: a TC channel ID, e.g. TC_CH3_ID
 *    -- return: the value (32-bit unsigned integer) in channel "channelID"'s
  counter */
uint32_t tcReadChannel(int channelID) {
    Tc* block = tcChannelToBlockBase(channelID);
    int chInd = channelID % TC_CH_NUMBER;
    return block->TC_CH[chInd].TC_CV;
}

/* Resets the counter of a given channel to zero, at which point it continues
 counting.
 *    -- channel ID: a TC channel ID, e.g. TC_CH3_ID */
void tcResetChannel(int channelID) {
    Tc* block = tcChannelToBlockBase(channelID);
    int chInd = channelID % TC_CH_NUMBER;
    block->TC_CH[chInd].TC_CCR.SWTRG = 1;
}
```

```
/* Sets the value of the RC compare register for a given channel, relevant to
 certain TC modes.
 *    -- channel ID: a TC channel ID, e.g. TC_CH3_ID
 *    -- val: the value (32-bit unsigned integer) to write to the RC register
  */
void tcSetRC_compare(int channelID, uint32_t val) {
    Tc* block = tcChannelToBlockBase(channelID);
    int chInd = channelID % TC_CH_NUMBER;
    block->TC_CH[chInd].TC_RC = val;
}

/* Checks whether an RC match has occurred since the last call to
 tcCheckRC_compare().
 *    -- channel ID: a TC channel ID, e.g. TC_CH3_IC
 *    -- return: 1 if an RC match has occurred since the last read; 0 if it
  hasn't */
int tcCheckRC_compare(int channelID) {
    Tc* block = tcChannelToBlockBase(channelID);
    int chInd = channelID % TC_CH_NUMBER;
    return block->TC_CH[chInd].TC_SR.CPCS;
}

/* Delays the system by a specified number of microseconds
 *    -- duration: the number of microseconds to delay
 * Note: This works up to (2^16 - 1 = 65535) us. Using the fastest available
  clock, TC_CLCK1_ID,
 * we achieve a resolution of 0.5 us. Also note that the doesn't use the above
  functions to optimize
 * speed; ideally, it would be written in assembly language for further
  optimization. Requires that
 * tcDelayInit() be called previously. Has not been tested rigorously for
  accuracy. */
void tcDelayMicros(uint32_t duration) {
    Tc* block = tcChannelToBlockBase(TC_CH0_ID);
    int chInd = TC_CH0_ID % TC_CH_NUMBER;
    block->TC_CH[chInd].TC_CCR.SWTRG = 1; // Reset counter
    block->TC_CH[chInd].TC_RC = duration * (TC_CLK1_SPEED / 1e6); // Set
     compare value
    while(!(block->TC_CH[chInd].TC_SR.CPCS)); // Wait until an RC Compare has
     occurred
}

/* Delays the system by a specified number of milliseconds
 *    -- duration: the number of milliseconds to delay
 * Note: The dependence on a "for" loop makes this code less efficient than
  tcDelayMicros(), and
 * so should be avoided for durations shorter than 65 milliseconds, in which
  case tcDelayMicros()
 * is the better option. Requires that tcDelayInit() be called previously. Has
  not been tested
```

```
 * rigorously for accuracy. */
void tcDelayMillis(int duration) {
    for (int i = 0; i < duration; i++) {
        tcDelayMicros(1000);
    }
}

/* Delays the system by a specified number of seconds
 *    -- duration: the number of seconds to delay
 * Note: the dependence on nested "for" loops and function calls makes this
  code extremely
 * inefficient, and so should be avoided for durations shorter than a minute.
  Requries that
 * tcDelayInit be called previously. Has not been tested rigorously for
  accuracy. */
void tcDelaySeconds(int duration) {
    for (int i = 0; i < duration; i++) {
        tcDelayMillis(1000);
    }
}


#endif
```

```c
/* SAM4S4B_spi.h
 *
 * cferrarin@g.hmc.edu
 * kpezeshki@g.hmc.edu
 * 12/11/2018
 *
 * Contains base address locations, register structs, definitions, and
  functions for the SPI (Serial
 * Peripheral Interface) peripheral of the SAM4S4B microcontroller. */

#ifndef SAM4S4B_SPI_H
#define SAM4S4B_SPI_H

#include <stdint.h>

/////////////////////////////////////////////////////////////////////////////
 //////////////////////
// SPI Base Address Definitions
/////////////////////////////////////////////////////////////////////////////
 //////////////////////

#define SPI_BASE   (0x40008000U) // SPI Base Address


/////////////////////////////////////////////////////////////////////////////
 //////////////////////
// SPI Registers
/////////////////////////////////////////////////////////////////////////////
 //////////////////////

// Bit field struct for the SPI_CR register
typedef struct {
    volatile uint32_t SPIEN    : 1;
    volatile uint32_t SPIDIS   : 1;
    volatile uint32_t          : 5;
    volatile uint32_t SWRST    : 1;
    volatile uint32_t          : 16;
    volatile uint32_t LASTXFER : 1;
    volatile uint32_t          : 7;
} SPI_CR_bits;

// Bit field struct for the SPI_MR register
typedef struct {
    volatile uint32_t MSTR    : 1;
    volatile uint32_t PS      : 1;
    volatile uint32_t PCSDEC  : 1;
    volatile uint32_t         : 1;
    volatile uint32_t MODFDIS : 1;
    volatile uint32_t WDRBT   : 1;
    volatile uint32_t         : 1;
    volatile uint32_t LLB     : 1;
```

```c
    volatile uint32_t         : 8;
    volatile uint32_t PCS     : 4;
    volatile uint32_t         : 4;
    volatile uint32_t DLYBCS  : 8;
} SPI_MR_bits;

// Bit field struct for the SPI_RDR register
typedef struct {
    volatile uint32_t RD  : 16;
    volatile uint32_t PCS : 4;
    volatile uint32_t     : 12;
} SPI_RDR_bits;

// Bit field struct for the SPI_TDR register
typedef struct {
    volatile uint32_t TD       : 16;
    volatile uint32_t PCS      : 4;
    volatile uint32_t          : 4;
    volatile uint32_t LASTXFER : 1;
    volatile uint32_t          : 7;
} SPI_TDR_bits;

// Bit field struct for the SPI_SR register
typedef struct {
    volatile uint32_t RDRF    : 1;
    volatile uint32_t TDRE    : 1;
    volatile uint32_t MODF    : 1;
    volatile uint32_t OVRES   : 1;
    volatile uint32_t ENDRX   : 1;
    volatile uint32_t ENDTX   : 1;
    volatile uint32_t RXBUFF  : 1;
    volatile uint32_t TXBUFE  : 1;
    volatile uint32_t NSSR    : 1;
    volatile uint32_t TXEMPTY : 1;
    volatile uint32_t UNDES   : 1;
    volatile uint32_t         : 5;
    volatile uint32_t SPIENS  : 1;
    volatile uint32_t         : 15;
} SPI_SR_bits;

// Bit field struct for the SPI_CSR register
typedef struct {
    volatile uint32_t CPOL   : 1;
    volatile uint32_t NCPHA  : 1;
    volatile uint32_t CSNAAT : 1;
    volatile uint32_t CSAAT  : 1;
    volatile uint32_t BITS   : 4;
    volatile uint32_t SCBR   : 8;
    volatile uint32_t DLYBS  : 8;
    volatile uint32_t DLYBCT : 8;
} SPI_CSR_bits;
```

```
// Peripheral struct for the SPI peripheral
typedef struct {
    volatile SPI_CR_bits  SPI_CR;        // (Spi Offset: 0x00) Control Register
    volatile SPI_MR_bits  SPI_MR;        // (Spi Offset: 0x04) Mode Register
    volatile SPI_RDR_bits SPI_RDR;       // (Spi Offset: 0x08) Receive Data
     Register
    volatile SPI_TDR_bits SPI_TDR;       // (Spi Offset: 0x0C) Transmit Data
     Register
    volatile SPI_SR_bits  SPI_SR;        // (Spi Offset: 0x10) Status Register
    volatile uint32_t     SPI_IER;       // (Spi Offset: 0x14) Interrupt Enable
     Register
    volatile uint32_t     SPI_IDR;       // (Spi Offset: 0x18) Interrupt
     Disable Register
    volatile uint32_t     SPI_IMR;       // (Spi Offset: 0x1C) Interrupt Mask
     Register
    volatile uint32_t     Reserved1[4];
    volatile SPI_CSR_bits SPI_CSR0;      // (Spi Offset: 0x30) Chip Select
     Register 0
    volatile SPI_CSR_bits SPI_CSR1;      // (Spi Offset: 0x30) Chip Select
     Register 1
    volatile SPI_CSR_bits SPI_CSR2;      // (Spi Offset: 0x30) Chip Select
     Register 2
    volatile SPI_CSR_bits SPI_CSR3;      // (Spi Offset: 0x30) Chip Select
     Register 3
    volatile uint32_t     Reserved2[41];
    volatile uint32_t     SPI_WPMR;      // (Spi Offset: 0xE4) Write Protection
     Control Register
    volatile uint32_t     SPI_WPSR;      // (Spi Offset: 0xE8) Write Protection
     Status Register
    volatile uint32_t     Reserved3[5];
    volatile uint32_t     SPI_RPR;       // (Spi Offset: 0x100) Receive Pointer
     Register
    volatile uint32_t     SPI_RCR;       // (Spi Offset: 0x104) Receive Counter
     Register
    volatile uint32_t     SPI_TPR;       // (Spi Offset: 0x108) Transmit
     Pointer Register
    volatile uint32_t     SPI_TCR;       // (Spi Offset: 0x10C) Transmit
     Counter Register
    volatile uint32_t     SPI_RNPR;      // (Spi Offset: 0x110) Receive Next
     Pointer Register
    volatile uint32_t     SPI_RNCR;      // (Spi Offset: 0x114) Receive Next
     Counter Register
    volatile uint32_t     SPI_TNPR;      // (Spi Offset: 0x118) Transmit Next
     Pointer Register
    volatile uint32_t     SPI_TNCR;      // (Spi Offset: 0x11C) Transmit Next
     Counter Register
    volatile uint32_t     SPI_PTCR;      // (Spi Offset: 0x120) Transfer
     Control Register
    volatile uint32_t     SPI_PTSR;      // (Spi Offset: 0x124) Transfer Status
     Register
```

```
} Spi;

// Pointer to an Spi-sized chunk of memory at the SPI peripheral
#define SPI ((Spi*) SPI_BASE)


////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// SPI Definitions
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

// Writing any other value in this field aborts the write operation of the WPEN
 bit.
// Always reads as 0.
#define SPI_WPMR_WPKEY_PASSWD (0x535049u << 8)


////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// SPI Functions
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

/* Enables the SPI peripheral and intializes its clock speed (baud rate),
 polarity, and phase. */
void spiInit(uint32_t clkdivide, uint32_t cpol, uint32_t ncpha) {
    pmcEnablePeriph(PMC_ID_SPI);
    /*Initializes the SPI interface for Chip Select line 0

    clkdivide (0x01 to 0xFF). The SPI clk will be the master clock / clkdivide
    cpol: clock polarity (0: inactive state is logic level 0, 1: inactive state
     is logic level 1)
    ncpha: clock phase (0: data changed on leading edge of clk and captured on
     next edge, 1: data captured on leading edge of clk and changed on next
     edge)
    Please see p585-p586 for cpol/ncpha timing diagrams

    This implements only: (p601/p610)
        1) SPI Master Mode
        2) Fixed Peripheral Select
        3) Mode Fault Detection Enabled
        4) Local Loopback Disabled
        5) 8 Bits Per Transfer
    Please read the SPI User Interface section of the datasheet for more
     advanced configuration features
    */

    //Initially assigning SPI pins (PA11-PA14) to peripheral A (SPI). Pin
     mapping given in p38-p39
    pioPinMode(PIO_PA11, PIO_PERIPH_A);
```

```
    pioPinMode(PIO_PA12, PIO_PERIPH_A);
    pioPinMode(PIO_PA13, PIO_PERIPH_A);
    pioPinMode(PIO_PA14, PIO_PERIPH_A);

    //next setting the SPI control register (p600). Set to 1 to enable SPI
    SPI->SPI_CR.SPIEN = 1;

    //next setting the SPI mode register (p601) with the following:
    //master mode
    //fixed peripheral select
    //chip select lines directly connected to peripheral device
    //mode fault detection enabled
    //WDRBT disabled
    //LLB disabled
    //PCS = 0000 (Peripheral 0 selected), means NPCS[3:0] = 1110
    SPI->SPI_MR.MSTR = 1;

    //next setting the chip select register for peripheral 0 (p610)
    //ignoring delays
    SPI->SPI_CSR0.SCBR = (cpol<<0) | (ncpha<<1) | (clkdivide << 16);
}

char spiSendReceive(char send) {
    //Sends one byte over SPI and returns the received character
    SPI->SPI_TDR.TD = send;
    //Wait until Receive Data Register Full (RDRF, bit 0) and TXEMPTY (bit )
    while (!(SPI->SPI_SR.RDRF) || (SPI->SPI_SR.TXEMPTY));
    //After these status bits have gone high, the transaction is complete
    return (char) (SPI->SPI_RDR.RD);
}

short spiSendReceive16(uint16_t send) {
    //sends one 16-bit short over SPI and returns the received short
    short rec;
    rec = spiSendReceive((send & 0xFF00) >> 8); // send data MSB first
    rec = (rec << 8) | spiSendReceive(send & 0xFF);
    return rec;
}


#endif
```

```c
/* SAM4S4B_rtc.h
 *
 * cferrarin@g.hmc.edu
 * kpezeshki@g.hmc.edu
 * 12/11/2018
 *
 * Contains base address locations, register structs, definitions, and
  functions for the RTC (Real-
 * Time Clock) peripheral of the SAM4S4B microcontroller. */

#ifndef SAM4S4B_RTC_H
#define SAM4S4B_RTC_H

#include <stdint.h>

////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// RTC Base Address Definitions
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

#define RTC_BASE   (0x400E1460U) // RTC Base Address


////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// RTC Registers
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

// Bit field struct for the RTC_CR register
typedef struct {
    volatile uint32_t UPDTIM   : 1;
    volatile uint32_t UPDCAL   : 1;
    volatile uint32_t          : 6;
    volatile uint32_t TIMEVSEL : 2;
    volatile uint32_t          : 6;
    volatile uint32_t CALEVSEL : 2;
    volatile uint32_t          : 14;
} RTC_CR_bits;

// Bit field struct for the RTC_MR register
typedef struct {
    volatile uint32_t HRMOD : 1;
    volatile uint32_t       : 31;
} RTC_MR_bits;

// Bit field struct for the RTC_TIMR register
typedef struct {
    volatile uint32_t SEC  : 7;
    volatile uint32_t      : 1;
```

```c
    volatile uint32_t MIN  : 7;
    volatile uint32_t      : 1;
    volatile uint32_t HOUR : 6;
    volatile uint32_t AMPM : 1;
    volatile uint32_t      : 9;
} RTC_TIMR_bits;

// Bit field struct for the RTC_CALR register
typedef struct {
    volatile uint32_t CENT  : 7;
    volatile uint32_t       : 1;
    volatile uint32_t YEAR  : 8;
    volatile uint32_t MONTH : 5;
    volatile uint32_t DAY   : 3;
    volatile uint32_t DATE  : 6;
    volatile uint32_t       : 2;
} RTC_CALR_bits;

// Bit field struct for the RTC_SR register
typedef struct {
    volatile uint32_t ACKUPD : 1;
    volatile uint32_t ALARM  : 1;
    volatile uint32_t SEC    : 1;
    volatile uint32_t TIMEV  : 1;
    volatile uint32_t CALEV  : 1;
    volatile uint32_t        : 27;
} RTC_SR_bits;

// Bit field struct for the RTC_SCCR register
typedef struct {
    volatile uint32_t ACKCLR : 1;
    volatile uint32_t ALRCLR : 1;
    volatile uint32_t SECCLR : 1;
    volatile uint32_t TIMCLR : 1;
    volatile uint32_t CALCLR : 1;
    volatile uint32_t        : 27;
} RTC_SCCR_bits;

// Peripheral struct for the RTC peripheral
typedef struct {
    volatile RTC_CR_bits   RTC_CR;     // (Rtc Offset: 0x00) Control Register
    volatile RTC_MR_bits   RTC_MR;     // (Rtc Offset: 0x04) Mode Register
    volatile RTC_TIMR_bits RTC_TIMR;   // (Rtc Offset: 0x08) Time Register
    volatile RTC_CALR_bits RTC_CALR;   // (Rtc Offset: 0x0C) Calendar Register
    volatile uint32_t      RTC_TIMALR; // (Rtc Offset: 0x10) Time Alarm
     Register
    volatile uint32_t      RTC_CALALR; // (Rtc Offset: 0x14) Calendar Alarm
     Register
    volatile RTC_SR_bits   RTC_SR;     // (Rtc Offset: 0x18) Status Register
    volatile RTC_SCCR_bits RTC_SCCR;   // (Rtc Offset: 0x1C) Status Clear
     Command Register
```

```
    volatile uint32_t      RTC_IER;    // (Rtc Offset: 0x20) Interrupt Enable
     Register
    volatile uint32_t      RTC_IDR;    // (Rtc Offset: 0x24) Interrupt Disable
     Register
    volatile uint32_t      RTC_IMR;    // (Rtc Offset: 0x28) Interrupt Mask
     Register
    volatile uint32_t      RTC_VER;    // (Rtc Offset: 0x2C) Valid Entry
     Register
} Rtc;

// Pointer to an Rtc-sized chunk of memory at the RTC peripheral
#define RTC ((Rtc*) RTC_BASE)


////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// RTC Definitions
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

// Values which the HRMOD bit in the RTC_SR register can take on
#define RTC_MR_HRMOD_24HR 0 // 24-hour mode (not supported by this driver)
#define RTC_MR_HRMOD_12HR 1 // 12-hour mode (used exclusively by this driver)

// (RTC does not have write protection).


////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// RTC Functions
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

/* Initializes the RTC peripheral's mode to 12-hour mode (24-hour mode is
 default).
   Note: there is no need to enable the clock with PMC with this peripheral. */
void rtcInit() {
    RTC->RTC_MR.HRMOD = RTC_MR_HRMOD_12HR; // Selects 12-hour mode
}

/* Updates the current time (seconds, minutes, hour, AM/PM) according to user
 values.
 *    -- sec: The current number of seconds in the current minute (0-59, BCD)
 *    -- min: The current number of minutes in the current hour (0-59, BCD)
 *    -- hour: The current hour in the current half of the day (1012, BCD)
 *    -- ampm: the current half of the day (AM = 0, PM = 1) */
void rtcUpdateTime(uint32_t sec, uint32_t min, uint32_t hour, uint32_t ampm) {
    while ((!RTC->RTC_SR.SEC));
    RTC->RTC_CR.UPDTIM = 1;
    while (!(RTC->RTC_SR.ACKUPD));
    RTC->RTC_SCCR.ACKCLR = 1;
```

```
    RTC->RTC_TIMR.SEC = sec;
    RTC->RTC_TIMR.MIN = min;
    RTC->RTC_TIMR.HOUR = hour;
    RTC->RTC_TIMR.AMPM = ampm;

    RTC->RTC_CR.UPDTIM = 0;
    RTC->RTC_SR.SEC = 0;
}

/* Updates the current date (date, day, month, year, century) according to user
 values.
 *    -- cent: The current century (19-20, BCD)
 *    -- year: The current year in the current century (0-99, BCD)
 *    -- month: The current month in the current year (1-12, BCD)
 *    -- day: The current day in the current week (1-7, BCD, arbitrary coding)
 *    -- date: The current day in the current month (1-31, BCD) */
void rtcUpdateDate(uint32_t cent, uint32_t year, uint32_t month,
    uint32_t day, uint32_t date) {
    while ((!RTC->RTC_SR.SEC));
    RTC->RTC_CR.UPDCAL = 1;
    while (!(RTC->RTC_SR.ACKUPD));
    RTC->RTC_SCCR.ACKCLR = 1;

    RTC->RTC_CALR.CENT = cent;
    RTC->RTC_CALR.YEAR = year;
    RTC->RTC_CALR.MONTH = month;
    RTC->RTC_CALR.DAY = day;
    RTC->RTC_CALR.DATE = date;

    RTC->RTC_CR.UPDCAL = 0;
    RTC->RTC_SR.SEC = 0;
}

/* Reads the current second in the current minute.
 *    -- return: the current second, in decimal */
int rtcReadSec() {
    int units = (RTC->RTC_TIMR.SEC) & 0xF;
    int tens = (RTC->RTC_TIMR.SEC) >> 4;
    return 10*tens + units;
}

/* Reads the current minute in the current hour.
 *    -- return: the current minute, in decimal */
int rtcReadMin() {
    int units = (RTC->RTC_TIMR.MIN) & 0xF;
    int tens = (RTC->RTC_TIMR.MIN) >> 4;
    return 10*tens + units;
}

/* Reads the current hour in the current half of the day.
```

```
 *    -- return: the current hour, in decimal */
int rtcReadHour() {
    int units = (RTC->RTC_TIMR.HOUR) & 0xF;
    int tens = (RTC->RTC_TIMR.HOUR) >> 4;
    return 10*tens + units;
}

/* Reads the current half of the day (AM or PM).
 *    -- return: the current half of the day, in decimal */
int rtcReadAmPm() {
    return RTC->RTC_TIMR.AMPM;
}

/* Reads the current century.
 *    -- return: the current century, in decimal */
int rtcReadCent() {
    int units = (RTC->RTC_CALR.CENT) & 0xF;
    int tens = (RTC->RTC_CALR.CENT) >> 4;
    return 10*tens + units;
}

/* Reads the current year in the current century.
 *    -- return: the current year, in decimal */
int rtcReadYear() {
    int units = (RTC->RTC_CALR.YEAR) & 0xF;
    int tens = (RTC->RTC_CALR.YEAR) >> 4;
    return 10*tens + units;
}

/* Reads the current month in the current year.
 *    -- return: the current month, in decimal */
int rtcReadMonth() {
    int units = (RTC->RTC_CALR.MONTH) & 0xF;
    int tens = (RTC->RTC_CALR.MONTH) >> 4;
    return 10*tens + units;
}

/* Reads the current day in the current week.
 *    -- return: the current day, in decimal (arbitrary coding) */
int rtcReadDay() {
    return RTC->RTC_CALR.DAY;
}

/* Reads the current day in the current month.
 *    -- return: the current day, in decimal */
int rtcReadDate() {
    int units = (RTC->RTC_CALR.DATE) & 0xF;
    int tens = (RTC->RTC_CALR.DATE) >> 4;
    return 10*tens + units;
}
```

```
#endif
```

```
/* SAM4S4B_pwm.h
 *
 * cferrarin@g.hmc.edu
 * kpezeshki@g.hmc.edu
 * 12/11/2018
 *
 * Contains base address locations, register structs, definitions, and
  functions for the PWM
 * (Pulse Width Modulation Controller) peripheral of the SAM4S4B
  microcontroller. */

#ifndef SAM4S4B_PWM_H
#define SAM4S4B_PWM_H

#include <stdint.h>
#include "SAM4S4B_sys.h"
#include "SAM4S4B_pio.h"

////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// PWM Base Address Definitions
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

#define PWM_BASE (0x40020000U) // PWM Base Address


////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// PWM Registers
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

// Bit field struct for the PWM_CLK register
typedef struct {
    volatile uint32_t DIVA : 8;
    volatile uint32_t PREA : 4;
    volatile uint32_t      : 4;
    volatile uint32_t DIVB : 8;
    volatile uint32_t PREB : 4;
    volatile uint32_t      : 4;
} PWM_CLK_bits;

// Bit field struct for the PWM_CMR register
typedef struct {
    volatile uint32_t CPRE : 4;
    volatile uint32_t      : 4;
    volatile uint32_t CALG : 1;
    volatile uint32_t CPOL : 1;
    volatile uint32_t CES  : 1;
    volatile uint32_t      : 5;
```

```
    volatile uint32_t DTE  : 1;
    volatile uint32_t DTHI : 1;
    volatile uint32_t DTLI : 1;
    volatile uint32_t      : 13;
} PWM_CMR_bits;

// Channel struct for each of the PWM peripheral's 4 channels
typedef struct {
    volatile PWM_CMR_bits PWM_CMR;      // (PwmCh_num Offset: 0x0) PWM Channel
     Mode Register
    volatile uint32_t     PWM_CDTY;     // (PwmCh_num Offset: 0x4) PWM Channel
     Duty Cycle Register
    volatile uint32_t     PWM_CDTYUPD;  // (PwmCh_num Offset: 0x8) PWM Channel
     Duty Cycle Update Register
    volatile uint32_t     PWM_CPRD;     // (PwmCh_num Offset: 0xC) PWM Channel
     Period Register
    volatile uint32_t     PWM_CPRDUPD;  // (PwmCh_num Offset: 0x10) PWM
     Channel Period Update Register
    volatile uint32_t     PWM_CCNT;     // (PwmCh_num Offset: 0x14) PWM
     Channel Counter Register
    volatile uint32_t     PWM_DT;       // (PwmCh_num Offset: 0x18) PWM
     Channel Dead Time Register
    volatile uint32_t     PWM_DTUPD;    // (PwmCh_num Offset: 0x1C) PWM
     Channel Dead Time Update Register
} PwmCh;

// Channel struct for each of the PWM peripheral's 8 comparison options
typedef struct {
    volatile uint32_t PWM_CMPV;      // (PwmCmp Offset: 0x0) PWM Comparison x
     Value Register
    volatile uint32_t PWM_CMPVUPD;   // (PwmCmp Offset: 0x4) PWM Comparison x
     Value Update Register
    volatile uint32_t PWM_CMPM;      // (PwmCmp Offset: 0x8) PWM Comparison x
     Mode Register
    volatile uint32_t PWM_CMPMUPD;   // (PwmCmp Offset: 0xC) PWM Comparison x
     Mode Update Register
} PwmCmp;

#define PWM_CMP_NUMBER 8
#define PWM_CH_NUMBER 4
// Peripheral struct for the PWM peripheral
typedef struct {
    volatile PWM_CLK_bits PWM_CLK;      // (Pwm Offset: 0x00) PWM Clock
     Register
    volatile uint32_t   PWM_ENA;       // (Pwm Offset: 0x04) PWM Enable
     Register
    volatile uint32_t   PWM_DIS;       // (Pwm Offset: 0x08) PWM Disable
     Register
    volatile uint32_t   PWM_SR;        // (Pwm Offset: 0x0C) PWM Status
     Register
```

```
volatile uint32_t   PWM_IER1;      // (Pwm Offset: 0x10) PWM Interrupt
 Enable Register 1
volatile uint32_t   PWM_IDR1;      // (Pwm Offset: 0x14) PWM Interrupt
 Disable Register 1
volatile uint32_t   PWM_IMR1;      // (Pwm Offset: 0x18) PWM Interrupt Mask
 Register 1
volatile uint32_t   PWM_ISR1;      // (Pwm Offset: 0x1C) PWM Interrupt
 Status Register 1
volatile uint32_t   PWM_SCM;       // (Pwm Offset: 0x20) PWM Sync Channels
 Mode Register
volatile uint32_t   Reserved1[1];
volatile uint32_t   PWM_SCUC;      // (Pwm Offset: 0x28) PWM Sync Channels
 Update Control Register
volatile uint32_t   PWM_SCUP;      // (Pwm Offset: 0x2C) PWM Sync Channels
 Update Period Register
volatile uint32_t   PWM_SCUPUPD;   // (Pwm Offset: 0x30) PWM Sync Channels
 Update Period Update Register
volatile uint32_t   PWM_IER2;      // (Pwm Offset: 0x34) PWM Interrupt
 Enable Register 2
volatile uint32_t   PWM_IDR2;      // (Pwm Offset: 0x38) PWM Interrupt
 Disable Register 2
volatile uint32_t   PWM_IMR2;      // (Pwm Offset: 0x3C) PWM Interrupt Mask
 Register 2
volatile uint32_t   PWM_ISR2;      // (Pwm Offset: 0x40) PWM Interrupt
 Status Register 2
volatile uint32_t   PWM_OOV;       // (Pwm Offset: 0x44) PWM Output
 Override Value Register
volatile uint32_t   PWM_OS;        // (Pwm Offset: 0x48) PWM Output
 Selection Register
volatile uint32_t   PWM_OSS;       // (Pwm Offset: 0x4C) PWM Output
 Selection Set Register
volatile uint32_t   PWM_OSC;       // (Pwm Offset: 0x50) PWM Output
 Selection Clear Register
volatile uint32_t   PWM_OSSUPD;    // (Pwm Offset: 0x54) PWM Output
 Selection Set Update Register
volatile uint32_t   PWM_OSCUPD;    // (Pwm Offset: 0x58) PWM Output
 Selection Clear Update Register
volatile uint32_t   PWM_FMR;       // (Pwm Offset: 0x5C) PWM Fault Mode
 Register
volatile uint32_t   PWM_FSR;       // (Pwm Offset: 0x60) PWM Fault Status
 Register
volatile uint32_t   PWM_FCR;       // (Pwm Offset: 0x64) PWM Fault Clear
 Register
volatile uint32_t   PWM_FPV;       // (Pwm Offset: 0x68) PWM Fault
 Protection Value Register
volatile uint32_t   PWM_FPE;       // (Pwm Offset: 0x6C) PWM Fault
 Protection Enable Register
volatile uint32_t   Reserved2[3];
volatile uint32_t   PWM_ELMR[2];   // (Pwm Offset: 0x7C) PWM Event Line 0
 Mode Register
volatile uint32_t   Reserved3[11];
```

```
    volatile uint32_t   PWM_SMMR;      // (Pwm Offset: 0xB0) PWM Stepper Motor
     Mode Register
    volatile uint32_t   Reserved4[12];
    volatile uint32_t   PWM_WPCR;      // (Pwm Offset: 0xE4) PWM Write Protect
     Control Register
    volatile uint32_t   PWM_WPSR;      // (Pwm Offset: 0xE8) PWM Write Protect
     Status Register
    volatile uint32_t   Reserved5[7];
    volatile uint32_t   PWM_TPR;       // (Pwm Offset: 0x108) Transmit Pointer
     Register
    volatile uint32_t   PWM_TCR;       // (Pwm Offset: 0x10C) Transmit Counter
     Register
    volatile uint32_t   Reserved6[2];
    volatile uint32_t   PWM_TNPR;      // (Pwm Offset: 0x118) Transmit Next
     Pointer Register
    volatile uint32_t   PWM_TNCR;      // (Pwm Offset: 0x11C) Transmit Next
     Counter Register
    volatile uint32_t   PWM_PTCR;      // (Pwm Offset: 0x120) Transfer Control
     Register
    volatile uint32_t   PWM_PTSR;      // (Pwm Offset: 0x124) Transfer Status
     Register
    volatile uint32_t   Reserved7[2];
    volatile PwmCmp     PWM_CMP[PWM_CMP_NUMBER]; // (Pwm Offset: 0x130) 0 .. 7
    volatile uint32_t   Reserved8[20];
    volatile PwmCh      PWM_CH[PWM_CH_NUMBER]; // (Pwm Offset: 0x200) ch = 0 ..
     3
} Pwm;

// Pointer to a Pwm-sized chunk of memory at the PWM peripheral
#define PWM ((Pwm*) PWM_BASE)


////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// PWM Definitions
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

// Constants relating to clock speed tuning in pwmInit()
#define PWM_CLK_PRE_MAX 11
#define PWM_CLK_DIV_MAX 255

// The specific PIO pins and peripheral function which PWM uses, set in
 pwmInit()
#define PWM_CH0_PIN PIO_PA11
#define PWM_CH1_PIN PIO_PA12
#define PWM_CH2_PIN PIO_PA13
#define PWM_CH3_PIN PIO_PA14
#define PWM_FUNC    PIO_PERIPH_B

// Values which "channelID" can take on in several functions
```

```
#define PWM_CH0 0
#define PWM_CH1 1
#define PWM_CH2 2
#define PWM_CH3 3

// Values which the CPOL bit in the PWM_CMR register can take on
#define PWM_CMR_CPOL_LOW  0 // Output waveform starts at a low level
#define PWM_CMR_CPOL_HIGH 1 // Output waveform starts at a high level

// Values which the CPRE bits in the PWM_CMR register can take on
#define PWM_CMR_CPRE_MCK     0
#define PWM_CMR_CPRE_MCK2    1
#define PWM_CMR_CPRE_MCK4    2
#define PWM_CMR_CPRE_MCK8    3
#define PWM_CMR_CPRE_MCK16   4
#define PWM_CMR_CPRE_MCK32   5
#define PWM_CMR_CPRE_MCK64   6
#define PWM_CMR_CPRE_MCK128  7
#define PWM_CMR_CPRE_MCK256  8
#define PWM_CMR_CPRE_MCK512  9
#define PWM_CMR_CPRE_MCK1024 10
#define PWM_CMR_CPRE_CLKA    11
#define PWM_CMR_CPRE_CLKB    12

// Writing any other value in this field aborts the write operation of the WPEN
 bit.
// Always reads as 0.
#define PWM_WPCR_WPKEY_PASSWD (0x50574DU << 8)


////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// PWM Functions
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

/* Enables the PWM peripheral and initializes its frequency, period, and duty
 cycle.
 * Requires pioInit().
 *    -- freq: the desired frequency of the PWM clock in Hz
 *    -- period: the desired frequency of the PWM waveform in number of clock
  periods
 *    -- dutyCycle: the desired duty cycle of the PWM waveform in number of
  waveform periods
 * Note: the actual frequency of the PWM waveform is given by freq / period,
  where
 * 0 < period < (2^16 = 65536). The higher the period,the more resolution for
  the duty cycle,
 * which is given by Duty Cycle = dutyCycle / period, where 0 < period < (2^16
  = 65536). Note that
```

```
 * 15.319 Hz <= freq <= 4 MHz based on allowable clock divisions. The alignment
  defaults to
 * left-aligned, and so is not set. Requires pioInit(). */
void pwmInit(int channelID, int freq, uint16_t period, uint16_t dutyCycle) {
    pmcEnablePeriph(PMC_ID_PWM);

    switch (channelID) {
        case PWM_CH0: pioPinMode(PWM_CH0_PIN, PWM_FUNC); break;
        case PWM_CH1: pioPinMode(PWM_CH1_PIN, PWM_FUNC); break;
        case PWM_CH2: pioPinMode(PWM_CH2_PIN, PWM_FUNC); break;
        case PWM_CH3: pioPinMode(PWM_CH3_PIN, PWM_FUNC); break;
    }

    PWM->PWM_DIS |= (1 << channelID); // Disables PWM while setting values

    uint32_t preScl[PWM_CLK_PRE_MAX] =
        {1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024};
    uint32_t preSclIndex = 0;
    uint32_t linDiv;

    // Finds prescaler and linear divider values
    while (preSclIndex < PWM_CLK_PRE_MAX) {
        linDiv = MCK_FREQ / preScl[preSclIndex] / freq;
        if (linDiv <= PWM_CLK_DIV_MAX) break;
        preSclIndex++;
    }

    // Sets the clock if a configuration can be found. Otherwise, disables the
     clock.
    if (preSclIndex < PWM_CLK_PRE_MAX) {
        PWM->PWM_CLK.PREA = preSclIndex;
        PWM->PWM_CLK.DIVA = linDiv;
    } else {
        PWM->PWM_CLK.DIVA = 0;
    }

    PWM->PWM_CH[channelID].PWM_CMR.CPRE = PWM_CMR_CPRE_CLKA; // Set clock speed
    PWM->PWM_CH[channelID].PWM_CMR.CPOL = PWM_CMR_CPOL_HIGH; // Set waveform
     polarity

    PWM->PWM_CH[channelID].PWM_CPRD = period; // Set period
    PWM->PWM_CH[channelID].PWM_CDTY = dutyCycle; // Set duty cycle

    PWM->PWM_ENA |= (1 << channelID); // Enable PWM after setting values
}


#endif
```

```
/* SAM4S4B_pmc.h
 *
 * cferrarin@g.hmc.edu
 * kpezeshki@g.hmc.edu
 * 12/11/2018
 *
 * Contains base address locations, register structs, definitions, and
  functions for the PMC
 * peripheral (Power Management Controller) of the SAM4S4B microcontroller. */

#ifndef SAM4S4B_PMC_H
#define SAM4S4B_PMC_H

#include <stdint.h>

////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// PMC Base Address Definitions
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

#define PMC_BASE   (0x400E0400U) // PMC Base Address


////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// PMC Registers
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

// Bit field struct for the PMC_SCER register
typedef struct {
    volatile uint32_t      : 7;
    volatile uint32_t UDP  : 1;
    volatile uint32_t PCK0 : 1;
    volatile uint32_t PCK1 : 1;
    volatile uint32_t PCK2 : 1;
    volatile uint32_t      : 21;
} PMC_SCER_bits;

// Bit field struct for the PMC_MCFR register
typedef struct {
    volatile uint32_t MAINF    : 16;
    volatile uint32_t MAINFRDY : 1;
    volatile uint32_t          : 15;
} PMC_MCFR_bits;

// Bit field struct fo the PMC_PCK register
typedef struct {
    volatile uint32_t CSS  : 3;
    volatile uint32_t      : 1;
```

```
    volatile uint32_t PRES : 3;
    volatile uint32_t      : 25;
} PMC_PCK_bits;

// Peripheral struct for a PMC peripheral
typedef struct {
    volatile PMC_SCER_bits PMC_SCER;       // (Pmc Offset: 0x0000) System Clock
     Enable Register
    volatile uint32_t      PMC_SCDR;       // (Pmc Offset: 0x0004) System Clock
     Disable Register
    volatile uint32_t      PMC_SCSR;       // (Pmc Offset: 0x0008) System Clock
     Status Register
    volatile uint32_t      Reserved1[1];
    volatile uint32_t      PMC_PCER0;      // (Pmc Offset: 0x0010) Peripheral
     Clock Enable Register 0
    volatile uint32_t      PMC_PCDR0;      // (Pmc Offset: 0x0014) Peripheral
     Clock Disable Register 0
    volatile uint32_t      PMC_PCSR0;      // (Pmc Offset: 0x0018) Peripheral
     Clock Status Register 0
    volatile uint32_t      Reserved2[1];
    volatile uint32_t      CKGR_MOR;       // (Pmc Offset: 0x0020) Main
     Oscillator Register
    volatile PMC_MCFR_bits CKGR_MCFR;      // (Pmc Offset: 0x0024) Main Clock
     Frequency Register
    volatile uint32_t      CKGR_PLLAR;     // (Pmc Offset: 0x0028) PLLA Register
    volatile uint32_t      CKGR_PLLBR;     // (Pmc Offset: 0x002C) PLLB Register
    volatile uint32_t      PMC_MCKR;       // (Pmc Offset: 0x0030) Master Clock
     Register
    volatile uint32_t      Reserved3[1];
    volatile uint32_t      PMC_USB;        // (Pmc Offset: 0x0038) USB Clock
     Register
    volatile uint32_t      Reserved4[1];
    volatile PMC_PCK_bits  PMC_PCK[3];     // (Pmc Offset: 0x0040) Programmable
     Clock 0 Register
    volatile uint32_t      Reserved5[5];
    volatile uint32_t      PMC_IER;        // (Pmc Offset: 0x0060) Interrupt
     Enable Register
    volatile uint32_t      PMC_IDR;        // (Pmc Offset: 0x0064) Interrupt
     Disable Register
    volatile uint32_t      PMC_SR;         // (Pmc Offset: 0x0068) Status
     Register
    volatile uint32_t      PMC_IMR;        // (Pmc Offset: 0x006C) Interrupt
     Mask Register
    volatile uint32_t      PMC_FSMR;       // (Pmc Offset: 0x0070) Fast Startup
     Mode Register
    volatile uint32_t      PMC_FSPR;       // (Pmc Offset: 0x0074) Fast Startup
     Polarity Register
    volatile uint32_t      PMC_FOCR;       // (Pmc Offset: 0x0078) Fault Output
     Clear Register
    volatile uint32_t      Reserved6[26];
```

```
    volatile uint32_t      PMC_WPMR;      // (Pmc Offset: 0x00E4) Write Protect
     Mode Register
    volatile uint32_t      PMC_WPSR;      // (Pmc Offset: 0x00E8) Write Protect
     Status Register
    volatile uint32_t      Reserved7[5];
    volatile uint32_t      PMC_PCER1;     // (Pmc Offset: 0x0100) Peripheral
     Clock Enable Register 1
    volatile uint32_t      PMC_PCDR1;     // (Pmc Offset: 0x0104) Peripheral
     Clock Disable Register 1
    volatile uint32_t      PMC_PCSR1;     // (Pmc Offset: 0x0108) Peripheral
     Clock Status Register 1
    volatile uint32_t      Reserved8[1];
    volatile uint32_t      PMC_OCR;       // (Pmc Offset: 0x0110) Oscillator
     Calibration Register
} Pmc;

// Pointer to a Pmc-sized chunk of memory at the PMC peripheral
#define PMC ((Pmc *) PMC_BASE)


////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// PMC Definitions
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

// Values which "periph" can take on in pmcEnablePeriph()
#define PMC_ID_SUPC   ( 0) // Supply Controller (SUPC)
#define PMC_ID_RSTC   ( 1) // Reset Controller (RSTC)
#define PMC_ID_RTC    ( 2) // Real Time Clock (RTC)
#define PMC_ID_RTT    ( 3) // Real Time Timer (RTT)
#define PMC_ID_WDT    ( 4) // Watchdog Timer (WDT)
#define PMC_ID_PMC    ( 5) // Power Management Controller (PMC)
#define PMC_ID_EFC    ( 6) // Enhanced Embedded Flash Controller (EFC)
#define PMC_ID_UART0  ( 8) // UART 0 (UART0)
#define PMC_ID_UART1  ( 9) // UART 1 (UART1)
#define PMC_ID_PIOA   (11) // Parallel I/O Controller A (PIOA)
#define PMC_ID_PIOB   (12) // Parallel I/O Controller B (PIOB)
#define PMC_ID_USART0 (14) // USART 0 (USART0)
#define PMC_ID_USART1 (15) // USART 1 (USART1)
#define PMC_ID_HSMCI  (18) // Multimedia Card Interface (HSMCI)
#define PMC_ID_TWI0   (19) // Two Wire Interface 0 (TWI0)
#define PMC_ID_TWI1   (20) // Two Wire Interface 1 (TWI1)
#define PMC_ID_SPI    (21) // Serial Peripheral Interface (SPI)
#define PMC_ID_SSC    (22) // Synchronous Serial Controler (SSC)
#define PMC_ID_TC0    (23) // Timer/Counter 0 (TC0)
#define PMC_ID_TC1    (24) // Timer/Counter 1 (TC1)
#define PMC_ID_TC2    (25) // Timer/Counter 2 (TC2)
#define PMC_ID_ADC    (29) // Analog To Digital Converter (ADC)
#define PMC_ID_DACC   (30) // Digital To Analog Converter (DACC)
#define PMC_ID_PWM    (31) // Pulse Width Modulation (PWM)
```

```
#define PMC_ID_CRCCU  (32) // CRC Calculation Unit (CRCCU)
#define PMC_ID_ACC    (33) // Analog Comparator (ACC)
#define PMC_ID_UDP    (34) // USB Device Port (UDP)

// Values which the CSS bits in PMC_PCK[k] can take on; clock IDs
#define PMC_PCK_CSS_SLOW_CLK 0
#define PMC_PCK_CSS_MAIN_CLK 1
#define PMC_PCK_CSS_PLLA_CLK 2
#define PMC_PCK_CSS_PLLB_CLK 3
#define PMC_PCK_CSS_MCK      4

// Values which the PRES bits in PMC_PCK[k] can take on; clock division factors
#define PMC_PCK_PRES_CLK1  0
#define PMC_PCK_PRES_CLK2  1
#define PMC_PCK_PRES_CLK4  2
#define PMC_PCK_PRES_CLK8  3
#define PMC_PCK_PRES_CLK16 4
#define PMC_PCK_PRES_CLK32 5
#define PMC_PCK_PRES_CLK63 6

// Writing any other value in this field aborts the write operation of the WPEN
 bit.
// Always reads as 0.
#define PMC_WPMR_WPKEY_PASSWD (0x504D43U << 8)


////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// PMC Functions
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

/* Routes Master Clock to the desired peripheral, thereby enabling it.
 *    -- periphID: a PMC peripheral ID to enable, e.g. PMC_ID_PIOA */
void pmcEnablePeriph(int periphID) {
    PMC->PMC_PCER0 = 1 << periphID;
}

/* Returns the number of Main Clock cycles within 16 Slow Clock periods.
 *    -- return: the number of Main Clock cycles in 16 Slow Clock periods.
 * This is useful for calibrating the Main Clock (which the peripherals
   indirectly use) if using
 * a reliable crystal oscillator for the slow clock. Returns 0 if the master
   clock is disabled or
 * if the read value is invalid. The RC oscillator which the Slow Clock uses by
   default runs at
 * about 32 kHz, but this can vary fairly substantially between boards */
int pmcCheckMasterClk() {
    int valid = PMC->CKGR_MCFR.MAINFRDY; // Check if reported value is valid
    return valid ? PMC->CKGR_MCFR.MAINF : 0;
}
```

```c
/* Initializes the programmable clock PCK2 for the FPGA to run at 1 MHz */
void pmcPCK2Init() {
    PMC->PMC_PCK[2].CSS = PMC_PCK_CSS_MCK; // Set clock source to master clock
    PMC->PMC_PCK[2].PRES = PMC_PCK_PRES_CLK4; // Divide clock by 4 for 1 MHz
    PMC->PMC_SCER.PCK2 = 1; // Enable programmable clock 0
}


#endif
```

```
/* SAM4S4B_pio.h
 *
 * cferrarin@g.hmc.edu
 * kpezeshki@g.hmc.edu
 * 12/11/2018
 *
 * Contains base address locations, register structs, definitions, and
  functions for the PIO
 * (Parallel Input/Output Controller) peripheral of the SAM4S4B
  microcontroller. */

#ifndef SAM4S4B_PIO_H
#define SAM4S4B_PIO_H

#include <stdint.h>

////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// PIO Base Address Definitions
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

#define PIOA_BASE  (0x400E0E00U) // PIOA Base Address
#define PIOB_BASE  (0x400E1000U) // PIOB Base Address


////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// PIO Registers
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

// Peripheral struct for a PIO peripheral (either PIOA or PIOB)
typedef struct {
    volatile uint32_t PIO_PER;        // (Pio Offset: 0x0000) PIO Enable
     Register
    volatile uint32_t PIO_PDR;        // (Pio Offset: 0x0004) PIO Disable
     Register
    volatile uint32_t PIO_PSR;        // (Pio Offset: 0x0008) PIO Status
     Register
    volatile uint32_t Reserved1[1];
    volatile uint32_t PIO_OER;        // (Pio Offset: 0x0010) Output Enable
     Register
    volatile uint32_t PIO_ODR;        // (Pio Offset: 0x0014) Output Disable
     Register
    volatile uint32_t PIO_OSR;        // (Pio Offset: 0x0018) Output Status
     Register
    volatile uint32_t Reserved2[1];
    volatile uint32_t PIO_IFER;       // (Pio Offset: 0x0020) Glitch Input
     Filter Enable Register
```

```
volatile uint32_t PIO_IFDR;       // (Pio Offset: 0x0024) Glitch Input
 Filter Disable Register
volatile uint32_t PIO_IFSR;       // (Pio Offset: 0x0028) Glitch Input
 Filter Status Register
volatile uint32_t Reserved3[1];
volatile uint32_t PIO_SODR;       // (Pio Offset: 0x0030) Set Output Data
 Register
volatile uint32_t PIO_CODR;       // (Pio Offset: 0x0034) Clear Output Data
 Register
volatile uint32_t PIO_ODSR;       // (Pio Offset: 0x0038) Output Data
 Status Register
volatile uint32_t PIO_PDSR;       // (Pio Offset: 0x003C) Pin Data Status
 Register
volatile uint32_t PIO_IER;        // (Pio Offset: 0x0040) Interrupt Enable
 Register
volatile uint32_t PIO_IDR;        // (Pio Offset: 0x0044) Interrupt Disable
 Register
volatile uint32_t PIO_IMR;        // (Pio Offset: 0x0048) Interrupt Mask
 Register
volatile uint32_t PIO_ISR;        // (Pio Offset: 0x004C) Interrupt Status
 Register
volatile uint32_t PIO_MDER;       // (Pio Offset: 0x0050) Multi-driver
 Enable Register
volatile uint32_t PIO_MDDR;       // (Pio Offset: 0x0054) Multi-driver
 Disable Register
volatile uint32_t PIO_MDSR;       // (Pio Offset: 0x0058) Multi-driver
 Status Register
volatile uint32_t Reserved4[1];
volatile uint32_t PIO_PUDR;       // (Pio Offset: 0x0060) Pull-up Disable
 Register
volatile uint32_t PIO_PUER;       // (Pio Offset: 0x0064) Pull-up Enable
 Register
volatile uint32_t PIO_PUSR;       // (Pio Offset: 0x0068) Pad Pull-up
 Status Register
volatile uint32_t Reserved5[1];
volatile uint32_t PIO_ABCDSR1;  // (Pio Offset: 0x0070) Peripheral Select
 Register 1
volatile uint32_t PIO_ABCDSR2;  // (Pio Offset: 0x0074) Peripheral Select
 Register 2
volatile uint32_t Reserved6[2];
volatile uint32_t PIO_IFSCDR;     // (Pio Offset: 0x0080) Input Filter Slow
 Clock Disable Register
volatile uint32_t PIO_IFSCER;     // (Pio Offset: 0x0084) Input Filter Slow
 Clock Enable Register
volatile uint32_t PIO_IFSCSR;     // (Pio Offset: 0x0088) Input Filter Slow
 Clock Status Register
volatile uint32_t PIO_SCDR;       // (Pio Offset: 0x008C) Slow Clock
 Divider Debouncing Register
volatile uint32_t PIO_PPDDR;      // (Pio Offset: 0x0090) Pad Pull-down
 Disable Register
```

```
volatile uint32_t PIO_PPDER;      // (Pio Offset: 0x0094) Pad Pull-down
 Enable Register
volatile uint32_t PIO_PPDSR;      // (Pio Offset: 0x0098) Pad Pull-down
 Status Register
volatile uint32_t Reserved7[1];
volatile uint32_t PIO_OWER;       // (Pio Offset: 0x00A0) Output Write
 Enable
volatile uint32_t PIO_OWDR;       // (Pio Offset: 0x00A4) Output Write
 Disable
volatile uint32_t PIO_OWSR;       // (Pio Offset: 0x00A8) Output Write
 Status Register
volatile uint32_t Reserved8[1];
volatile uint32_t PIO_AIMER;      // (Pio Offset: 0x00B0) Additional
 Interrupt Modes Enable Register
volatile uint32_t PIO_AIMDR;      // (Pio Offset: 0x00B4) Additional
 Interrupt Modes Disables Register
volatile uint32_t PIO_AIMMR;      // (Pio Offset: 0x00B8) Additional
 Interrupt Modes Mask Register
volatile uint32_t Reserved9[1];
volatile uint32_t PIO_ESR;        // (Pio Offset: 0x00C0) Edge Select
 Register
volatile uint32_t PIO_LSR;        // (Pio Offset: 0x00C4) Level Select
 Register
volatile uint32_t PIO_ELSR;       // (Pio Offset: 0x00C8) Edge/Level Status
 Register
volatile uint32_t Reserved10[1];
volatile uint32_t PIO_FELLSR;     // (Pio Offset: 0x00D0) Falling Edge/Low
 Level Select Register
volatile uint32_t PIO_REHLSR;     // (Pio Offset: 0x00D4) Rising Edge/ High
 Level Select Register
volatile uint32_t PIO_FRLHSR;     // (Pio Offset: 0x00D8) Fall/Rise - Low/
 High Status Register
volatile uint32_t Reserved11[1];
volatile uint32_t PIO_LOCKSR;     // (Pio Offset: 0x00E0) Lock Status
volatile uint32_t PIO_WPMR;       // (Pio Offset: 0x00E4) Write Protect
 Mode Register
volatile uint32_t PIO_WPSR;       // (Pio Offset: 0x00E8) Write Protect
 Status Register
volatile uint32_t Reserved12[5];
volatile uint32_t PIO_SCHMITT;    // (Pio Offset: 0x0100) Schmitt Trigger
 Register
volatile uint32_t Reserved13[19];
volatile uint32_t PIO_PCMR;       // (Pio Offset: 0x150) Parallel Capture
 Mode Register
volatile uint32_t PIO_PCIER;      // (Pio Offset: 0x154) Parallel Capture
 Interrupt Enable Register
volatile uint32_t PIO_PCIDR;      // (Pio Offset: 0x158) Parallel Capture
 Interrupt Disable Register
volatile uint32_t PIO_PCIMR;      // (Pio Offset: 0x15C) Parallel Capture
 Interrupt Mask Register
```

```
    volatile uint32_t PIO_PCISR;      // (Pio Offset: 0x160) Parallel Capture
     Interrupt Status Register
    volatile uint32_t PIO_PCRHR;      // (Pio Offset: 0x164) Parallel Capture
     Reception Holding Register
    volatile uint32_t PIO_RPR;        // (Pio Offset: 0x168) Receive Pointer
     Register
    volatile uint32_t PIO_RCR;        // (Pio Offset: 0x16C) Receive Counter
     Register
    volatile uint32_t Reserved14[2];
    volatile uint32_t PIO_RNPR;       // (Pio Offset: 0x178) Receive Next
     Pointer Register
    volatile uint32_t PIO_RNCR;       // (Pio Offset: 0x17C) Receive Next
     Counter Register
    volatile uint32_t Reserved15[2];
    volatile uint32_t PIO_PTCR;       // (Pio Offset: 0x188) Transfer Control
     Register
    volatile uint32_t PIO_PTSR;       // (Pio Offset: 0x18C) Transfer Status
     Register
} Pio;

// Pointers to Pio-sized chunks of memory at each PIO peripheral
#define PIOA ((Pio*) PIOA_BASE)
#define PIOB ((Pio*) PIOB_BASE)


////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// PIO Definitions
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

// Values which "val" can take on in pioWritePin()
#define PIO_LOW  0 // Value to write a pin low (0 V)
#define PIO_HIGH 1 // Value to write a pin high (3.3 V)

// Arbitrary port IDs used to easily find a pin's port
#define PIO_PORT_ID_A 0 // Arbitrary ID for PIO Port A
#define PIO_PORT_ID_B 1 // Arbitrary ID for PIO Port B

// Values which "function" can take on in pioPinMode()
#define PIO_INPUT     0 // Arbitrary ID for an input I/O line
#define PIO_OUTPUT    1 // Arbitrary ID for an output I/O line
#define PIO_PERIPH_A  2 // Arbitrary ID for peripheral function A
#define PIO_PERIPH_B  3 // Arbitrary ID for peripheral function B
#define PIO_PERIPH_C  4 // Arbitrary ID for peripheral function C
#define PIO_PERIPH_D  5 // Arbitrary ID for peripheral function D
#define PIO_PULL_DOWN 6 // Arbitrary ID for a pull-down resistor
#define PIO_FLOATING  7 // Arbitrary ID for neither a pull-up nor a pull-down
 resistor
```

```
// Pin definitions for every PIO pin, which "pin" can take on in several
 functions
#define PIO_PA0  0
#define PIO_PA1  1
#define PIO_PA2  2
#define PIO_PA3  3
#define PIO_PA4  4
#define PIO_PA5  5
#define PIO_PA6  6
#define PIO_PA7  7
#define PIO_PA8  8
#define PIO_PA9  9
#define PIO_PA10 10
#define PIO_PA11 11
#define PIO_PA12 12
#define PIO_PA13 13
#define PIO_PA14 14
#define PIO_PA15 15
#define PIO_PA16 16
#define PIO_PA17 17
#define PIO_PA18 18
#define PIO_PA19 19
#define PIO_PA20 20
#define PIO_PA21 21
#define PIO_PA22 22
#define PIO_PA23 23
#define PIO_PA24 24
#define PIO_PA25 25
#define PIO_PA26 26
#define PIO_PA27 27
#define PIO_PA28 28
#define PIO_PA29 29
#define PIO_PA30 30
#define PIO_PA31 31
#define PIO_PB0  32
#define PIO_PB1  33
#define PIO_PB2  34
#define PIO_PB3  35
#define PIO_PB4  36
#define PIO_PB5  37
#define PIO_PB6  38
#define PIO_PB7  39
#define PIO_PB8  40
#define PIO_PB9  41
#define PIO_PB10 42
#define PIO_PB11 43
#define PIO_PB12 44
#define PIO_PB13 45
#define PIO_PB14 46
```

```
// Writing any other value in this field aborts the write operation of the WPEN
 bit.
// Always reads as 0.
#define PIO_WPMR_WPKEY_PASSWD (0x50494Fu << 8)


////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// PIO Functions
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

/* Initializes the PIO peripheral by enabling the Master Clock to PIOA and
 PIOB. */
void pioInit() {
    pmcEnablePeriph(PMC_ID_PIOA);
    pmcEnablePeriph(PMC_ID_PIOB);
}

/* Returns the port ID that corresponds to a given pin.
 *    -- pin: a PIO pin ID, e.g. PIO_PA3
 *    -- return: a PIO port ID, e.g. PIO_PORT_ID_A */
int pioPinToPort(int pin) {
    return pin >> 5;
}

/* Returns a pointer to the given port's base address.
 *    -- port: a PIO port ID, e.g. PIO_PORT_ID_A
 *    -- return: a pointer to a Pio-sized block of memory at the port "port" */
Pio* pioPortToBase(int port) {
    return port ? PIOB : PIOA;
}

/* Given a pin, returns a pointer to the corresponding port's base address.
 *    -- pin: a PIO pin ID, e.g. PIO_PA3
 *    -- return: a pointer to a Pio-sized block of memory at the pin's port */
Pio* pioPinToBase(int pin) {
    return pioPortToBase(pioPinToPort(pin));
}

/* Sets a function (either I/O behavior, peripheral, or setting) of a pin.
 *    -- pin: a PIO pin ID, e.g. PIO_PA3
 *    -- function: a PIO function ID, e.g. PIO_PERIPH_C. While I/O functions
  (PIO_INPUT, PIO_OUTPUT)
 *       and peripherals (PIO_PERIPH_A - PIO_PERIPH_D) are mutually exclusive,
  settings
 *       (PIO_PULL_DOWN, PIO_FLOATING), can always be altered.
 * Note: upon reset, pins are configured as input I/O lines (as opposed to
  peripheral functions),
 * the peripheral defaults to PIO_PERIPH_A, the pull-up resistor is enabled,
  and the pull-down
```

```
 * resistor is disabled. All other optional pin functions, which are not
  provided in this driver,
 * are disabled upon reset. Note also that pin PA31 is used for the FPGA clock,
  and should not be
 * altered */
void pioPinMode(int pin, int function) {
    Pio* port = pioPinToBase(pin);
    int offset = pin % 32;

    switch (function) {
        case PIO_INPUT:
            break; // Do nothing, since this is default behavior
        case PIO_OUTPUT:
            port->PIO_OER     |=  (1 << offset); // Configures an I/O line as
             an output
            break;
        case PIO_PERIPH_A:
            port->PIO_PDR     |=  (1 << offset); // Sets a pin to be
             peripheral-controlled
            port->PIO_ABCDSR1 &= ~(1 << offset); // Sets the peripheral which
             controls a pin
            port->PIO_ABCDSR2 &= ~(1 << offset); // Sets the peripheral which
             controls a pin
            break;
        case PIO_PERIPH_B:
            port->PIO_PDR     |=  (1 << offset); // Sets a pin to be
             peripheral-controlled
            port->PIO_ABCDSR1 |=  (1 << offset); // Sets the peripheral which
             controls a pin
            port->PIO_ABCDSR2 &= ~(1 << offset); // Sets the peripheral which
             controls a pin
            break;
        case PIO_PERIPH_C:
            port->PIO_PDR     |=  (1 << offset); // Sets a pin to be
             peripheral-controlled
            port->PIO_ABCDSR1 &= ~(1 << offset); // Sets the peripheral which
             controls a pin
            port->PIO_ABCDSR2 |=  (1 << offset); // Sets the peripheral which
             controls a pin
            break;
        case PIO_PERIPH_D:
            port->PIO_PDR     |=  (1 << offset); // Sets a pin to be
             peripheral-controlled
            port->PIO_ABCDSR1 |=  (1 << offset); // Sets the peripheral which
             controls a pin
            port->PIO_ABCDSR2 |=  (1 << offset); // Sets the peripheral which
             controls a pin
            break;
        case PIO_PULL_DOWN:
            port->PIO_PUDR    |=  (1 << offset); // Disables the pull-up
             resistor
```

```
            port->PIO_PPDER   |=  (1 << offset); // Enables the pull-down
             resistor
        case PIO_FLOATING:
            port->PIO_PUDR    |=  (1 << offset); // Disables the pull-down
             resistor
    }
}

/* Reads the digital voltage on a pin configured as an input I/O line.
 *    -- pin: a PIO pin ID, e.g. PIO_PA3
 *    -- return: a PIO value ID, either PIO_HIGH or PIO_LOW */
int pioReadPin(int pin) {
    Pio* port = pioPinToBase(pin);
    int offset = pin % 32;
    return ((port->PIO_PDSR) >> offset) & 1;
}

/* Writes a digital voltage to a pin configured as an output I/O line.
 *    -- pin: a PIO pin ID, e.g. PIO_PA3
 *    -- val: a PIO value ID, either PIO_HIGH or PIO_LOW */
void pioWritePin(int pin, int val) {
    Pio* port = pioPinToBase(pin);
    int offset = pin % 32;
    if (val) {
        port->PIO_SODR |= (1 << offset);
    } else {
        port->PIO_CODR |= (1 << offset);
    }
}

/* Switches the digital voltage on a pin configured as in output I/O line
 *    -- pin: a PIO pin ID, e.g. PIO_PA3 */
void pioTogglePin(int pin) {
    int currentVal = pioReadPin(pin);
    pioWritePin(pin, !currentVal);
}


#endif
```

```
/* SAM4S4B_adc.h
 *
 * cferrarin@g.hmc.edu
 * kpezeshki@g.hmc.edu
 * 12/11/2018
 *
 * Contains base address locations, register structs, definitions, and
  functions for the ADC
 * (Analog-to-Digital Converter) peripheral of the SAM4S4B microcontroller. */

#ifndef SAM4S4B_ADC_H
#define SAM4S4B_ADC_H

#include <stdint.h>
#include "SAM4S4B_sys.h"
#include "SAM4S4B_pio.h"

////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// ADC Base Address Definitions
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

#define ADC_BASE   (0x40038000U) // ADC Base Address


////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// ADC Registers
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

// Bit field struct for the ADC_CR register
typedef struct {
    volatile uint32_t SWRST : 1;
    volatile uint32_t START : 1;
    volatile uint32_t       : 30;
} ADC_CR_bits;

// Bit field struct for the ADC_MR register
typedef struct {
    volatile uint32_t TRGEN : 1;
    volatile uint32_t TRGSEL : 3;
    volatile uint32_t LOWRES : 1;
    volatile uint32_t SLEEP  : 1;
    volatile uint32_t FWUP   : 1;
    volatile uint32_t FREERUN : 1;
    volatile uint32_t PRESCAL : 8;
    volatile uint32_t STARTUP : 4;
    volatile uint32_t SETTLING : 2;
    volatile uint32_t          : 1;
```

```
    volatile uint32_t ANACH    : 1;
    volatile uint32_t TRACKTIM : 4;
    volatile uint32_t TRANSFER : 2;
    volatile uint32_t          : 1;
    volatile uint32_t USEQ     : 1;
} ADC_MR_bits;

// Bit field struct for the ADC_ACR register
typedef struct {
    volatile uint32_t       : 4;
    volatile uint32_t TSON  : 1;
    volatile uint32_t       : 3;
    volatile uint32_t IBCTL : 2;
    volatile uint32_t       : 22;
} ADC_ACR_bits;

// Peripheral struct for the ADC peripheral
typedef struct {
    volatile ADC_CR_bits  ADC_CR;        // (Adc Offset: 0x00) Control Register
    volatile ADC_MR_bits  ADC_MR;        // (Adc Offset: 0x04) Mode Register
    volatile uint32_t     ADC_SEQR1;     // (Adc Offset: 0x08) Channel Sequence
     Register 1
    volatile uint32_t     ADC_SEQR2;     // (Adc Offset: 0x0C) Channel Sequence
     Register 2
    volatile uint32_t     ADC_CHER;      // (Adc Offset: 0x10) Channel Enable
     Register
    volatile uint32_t     ADC_CHDR;      // (Adc Offset: 0x14) Channel Disable
     Register
    volatile uint32_t     ADC_CHSR;      // (Adc Offset: 0x18) Channel Status
     Register
    volatile uint32_t     Reserved1[1];
    volatile uint32_t     ADC_LCDR;      // (Adc Offset: 0x20) Last Converted
     Data Register
    volatile uint32_t     ADC_IER;       // (Adc Offset: 0x24) Interrupt Enable
     Register
    volatile uint32_t     ADC_IDR;       // (Adc Offset: 0x28) Interrupt
     Disable Register
    volatile uint32_t     ADC_IMR;       // (Adc Offset: 0x2C) Interrupt Mask
     Register
    volatile uint32_t     ADC_ISR;       // (Adc Offset: 0x30) Interrupt Status
     Register
    volatile uint32_t     Reserved2[2];
    volatile uint32_t     ADC_OVER;      // (Adc Offset: 0x3C) Overrun Status
     Register
    volatile uint32_t     ADC_EMR;       // (Adc Offset: 0x40) Extended Mode
     Register
    volatile uint32_t     ADC_CWR;       // (Adc Offset: 0x44) Compare Window
     Register
    volatile uint32_t     ADC_CGR;       // (Adc Offset: 0x48) Channel Gain
     Register
```

```
    volatile uint32_t     ADC_COR;       // (Adc Offset: 0x4C) Channel Offset
     Register
    volatile uint32_t     ADC_CDR[15];   // (Adc Offset: 0x50) Channel Data
     Register
    volatile uint32_t     Reserved3[2];
    volatile ADC_ACR_bits ADC_ACR;       // (Adc Offset: 0x94) Analog Control
     Register
    volatile uint32_t     Reserved4[19];
    volatile uint32_t     ADC_WPMR;      // (Adc Offset: 0xE4) Write Protect
     Mode Register
    volatile uint32_t     ADC_WPSR;      // (Adc Offset: 0xE8) Write Protect
     Status Register
    volatile uint32_t     Reserved5[5];
    volatile uint32_t     ADC_RPR;       // (Adc Offset: 0x100) Receive Pointer
     Register
    volatile uint32_t     ADC_RCR;       // (Adc Offset: 0x104) Receive Counter
     Register
    volatile uint32_t     Reserved6[2];
    volatile uint32_t     ADC_RNPR;      // (Adc Offset: 0x110) Receive Next
     Pointer Register
    volatile uint32_t     ADC_RNCR;      // (Adc Offset: 0x114) Receive Next
     Counter Register
    volatile uint32_t     Reserved7[2];
    volatile uint32_t     ADC_PTCR;      // (Adc Offset: 0x120) Transfer
     Control Register
    volatile uint32_t     ADC_PTSR;      // (Adc Offset: 0x124) Transfer Status
     Register
} Adc;

// Pointer to an Adc-sized chunk of memory at the ADC peripheral
#define ADC ((Adc*) ADC_BASE)


////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// ADC Definitions
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

// Values which "channel" can take on in several functions
#define ADC_CH0  0
#define ADC_CH1  1
#define ADC_CH2  2
#define ADC_CH3  3
#define ADC_CH4  4
#define ADC_CH5  5
#define ADC_CH6  6
#define ADC_CH7  7
#define ADC_CH8  8
#define ADC_CH9  9
#define ADC_CH15 15
```

```
// Values which the LOWRES bit can take on in the ADC_MR register
#define ADC_MR_LOWRES_BITS_12 0 // 12-bit resolution
#define ADC_MR_LOWRES_BITS_10 1 // 10-bit resolution

// Values for each channel's gain in ADC_CGR
#define ADC_CGR_GAIN_X1 0 // Unity gain
#define ADC_CGR_GAIN_X2 2 // Gain times 2
#define ADC_CGR_GAIN_X4 3 // Gain times 4

// Values for each channel's offset in ADC_COR
#define ADC_COR_OFFSET_ON  1 // Centers the analog signal on (G-1)Vrefin/2
 prior to gain
#define ADC_COR_OFFSET_OFF 0 // No offset

// The specific PIO pins and peripheral function which ADC uses, set in
 adcInit()
#define ADC_CH0_PIN PIO_PA17
#define ADC_CH1_PIN PIO_PA18
#define ADC_CH2_PIN PIO_PA19
#define ADC_CH3_PIN PIO_PA20
#define ADC_CH4_PIN PIO_PB0
#define ADC_CH5_PIN PIO_PB1
#define ADC_CH6_PIN PIO_PB2
#define ADC_CH7_PIN PIO_PB3
#define ADC_CH8_PIN PIO_PA21
#define ADC_CH9_PIN PIO_PA22
#define ADC_FUNC    PIO_PERIPH_D

// The maximum value the ADC will record (dependent on the resolution)
#define ADC_DMAX_10 1023 // 2^10 - 1
#define ADC_DMAX_12 4095 // 2^12 - 1

// Writing any other value in this field aborts the write operation of the WPEN
 bit.
// Always reads as 0.
#define ADC_WPMR_WPKEY_PASSWD (0x414443U << 8)


////////////////////////////////////////////////////////////////////////////////
 /////////////////////
// ADC Functions
////////////////////////////////////////////////////////////////////////////////
 /////////////////////

/* Enables the ADC peripheral and initializes its resolution
 *    -- resolution: an ADC resolution ID, e.g. ADC_MR_LOWRES_BITS_10
 * Note: the ADC clock defaults to MCK_FREQ / 2 = 2 MHz; 1 MHz to 20 MHz is
  allowed. */
void adcInit(uint32_t resolution) {
    pmcEnablePeriph(PMC_ID_ADC);
```

```c
    ADC->ADC_MR.LOWRES = resolution; // Set resolution
    ADC->ADC_MR.ANACH = 1; // Allow channels to have independent settings
}

/* Enables an ADC channel and initializes its gain and offset
 *    -- channel: an ADC channel ID, e.g. ADC_CH3
 *    -- gain: an ADC gain ID, e.g. ADC_CGR_GAIN_X2
 *    -- offset: an ADC offset ID, e.g. ADC_COR_OFFSET_ON. Set the offset to 1
  to center the analog
 *       signal on (Gain - 1)*Vref/2 prior to gain */
void adcChannelInit(int channel, int gain, int offset) {
    // Set the channel's PIO pin to perform its ADC function
    switch (channel) {
        case ADC_CH0: pioPinMode(ADC_CH0_PIN, ADC_FUNC); break;
        case ADC_CH1: pioPinMode(ADC_CH1_PIN, ADC_FUNC); break;
        case ADC_CH2: pioPinMode(ADC_CH2_PIN, ADC_FUNC); break;
        case ADC_CH3: pioPinMode(ADC_CH3_PIN, ADC_FUNC); break;
        case ADC_CH4: pioPinMode(ADC_CH4_PIN, ADC_FUNC); break;
        case ADC_CH5: pioPinMode(ADC_CH5_PIN, ADC_FUNC); break;
        case ADC_CH6: pioPinMode(ADC_CH6_PIN, ADC_FUNC); break;
        case ADC_CH7: pioPinMode(ADC_CH7_PIN, ADC_FUNC); break;
        case ADC_CH8: pioPinMode(ADC_CH8_PIN, ADC_FUNC); break;
        case ADC_CH9: pioPinMode(ADC_CH9_PIN, ADC_FUNC); break;
        case ADC_CH15:                                   break;
    }
    ADC->ADC_CHER |= (1 << channel); // Enable the ADC channel

    // Set the gain
    ADC->ADC_CGR |= (gain << (2*channel));
    ADC->ADC_CGR &= ~((~gain & 0b11) << (2*channel));

    // Set the offset
    ADC->ADC_COR |= (offset << channel);
    ADC->ADC_COR &= ~((~offset & 0b1) << channel);
}

/* Reads the analog voltage reported by an ADC channel
 *    -- channel: an ADC channel ID, e.g. ADC_CH3
 *    -- return: the analog voltage represented by the ADC's report (in V)
 * Note: it is important to measure the voltage at the ADC's Vref pin and
  record it in
 * SAM4S4B_sys.h for accurate results. */
float adcRead(int channel) {
    ADC->ADC_CR.START = 1; // Start conversion
    while (!((ADC->ADC_ISR >> channel) & 1)); // Wait for conversion
    int d = ADC->ADC_CDR[channel]; // Received digital value
    int dMax = (ADC->ADC_MR.LOWRES) ? ADC_DMAX_10 : ADC_DMAX_12; // Maximum
     possible value
    return (((float) d) / dMax) * ADC_VREF;
}
```

```
#endif
```