LAB PROTOTYPE BOARD FOR THE RPi PICO 2040

A Design Project Report

Presented to the School of Electrical and Computer Engineering of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering, Electrical and Computer Engineering

Submitted by: Emily Wang

MEng Field Advisor: Hunter Adams

Degree Date: December 2021

Abstract

Master of Engineering Program

School of Electrical and Computer Engineering

Cornell University

Design Project Report

Project Title:

Lab Prototype Board for the RPi Pico 2040

Author:

Emily Wang (jw829)

Abstract:

Designed by Raspberry Pi, RP2040 is a dual-core, ARM Cortex-M0+ processor with powerful internal peripherals. The Raspberry Pi Pico is an affordable and versatile board built using RP2040 that breaks out all its peripheral pins so the chip can be easily programmed and interfaced. The ECE 4760 class is considering building a prototype PCB that will provide a socket for the RPi Pico and provide a digital-to-analog converter, an LCD, an IMU sensor, a port expander, a VGA connector, and headers for connection to student-built circuitry. The new prototype will provide students with a greater variety of hardware opportunities and a better programming experience than the previous prototype using PIC32. In this prototype, we took suggestions from the previous development board and produced a compact 2-layer PCB that carries the RPi Pico and its hardware peripherals. Some firmware changes will also be made to run the existing code on the new PCB.

Contents

1 Executive Summary

The motivation behind the project is to produce a prototype PCB using the RP2040 microcontroller to assist the ECE 4760 lab exercises. ECE 4760 is an upper-level ECE course that uses microcontrollers as components in electronic design and embedded control. Currently, the class uses a PIC32 microcontroller for programming and interfacing with the hardware peripherals, and we want to upgrade the system to use RP2040.

The new prototype board will keep all the original integrated circuits (ICs) such as the digital-to-analog converter (DAC), the port expander, and the voltage regulator. It will also include additional hardware peripherals such as a VGA display port, an inertial measurement unit (IMU) sensor via I2C, and a new TFT display screen that has four user-input buttons. The VGA display port can display anything in 3-bit color so students can design games and display visuals on a computer screen. The IMU sensor allows students to explore the I2C protocol and have interesting data to play with for the final project. The new TFT screen can be used as a game console due to its convenient size and buttons. All the GPIO pins on the RPi Pico are broken out on the prototype board for usability and testing.

The programming environment is set up on a Windows machine, and some software programs are written to create a demo which exercises the board peripherals. Specifically, the DAC channels are configured via the Serial Peripheral Interface (SPI) interface. This is done by configuring a timer interrupt at 40KHz on the Pico; whenever the timer overflows, a callback function runs and outputs a sine wave to the DAC. The port expander is also configured by translating the existing functions on the PIC32 using the RPi Pico libraries.

2 Project Background

The current prototype board for the ECE 4760 class includes a port expander, a two-channel digital-to-analog converter (DAC), a TFT screen, a programming header-plug, and a 5 V power supply. The prototype board is intended to be used for the lab exercises and the final project, and it can support a variety of hardware peripherals such as analog/digital sensors, PWM outputs, and animation displays.

The current prototype board uses a PIC32 microcontroller which has a 40 MHz clock rate and a 32-bit RISC CPU. It has two I2C and two SPI modules, and the existing prototype uses both SPI modules for its DAC, TFT, and port expander. It also supports UART serial communication where students can communicate with the PIC32 via a Python interface. The PIC32 is programmed via an ICSP header that is broken out on the prototype board.

While the current prototype is sufficient for ECE 4760 lab exercises, we want to consider using a new processor called the RP2040 and redesigning a prototype to expand the hardware/software potentials for student projects. The RP2040 has its own development board called the RPi Pico which allows for the microcontroller to be easily programmed by dragging and dropping a file. The Pico also has a rich peripheral set that is well-documented, including SPI, I2C, and eight programmable I/O state machines for custom peripheral support, allowing students to explore complicated projects using the extensive resources available.

3 Design Overview

The prototype PCB shall be designed around the RPi Pico and host several hardware peripherals such as a DAC, a port expander, a TFT screen, and a VGA display port. Design consideration shall keep in mind the hardware requirements to provide the most usability and versatility for student projects.

The design shall also consider providing both hardware and software learnings for students taking the ECE4760 class. Students should be able to bring up a bare protoboard and program it without prior experience in PCB bring-up and C programming.

3.1 Prototype PCB Component Selection

Subsystem	Component
MCU	RPi Pico with RP2040
IMU Sensor	LSM9DS1; Accelerometer/Gyro/Magnetometer
VGA Display	L77HDE15SD1CH4F; 3-bit VGA output
TFT Screen	240x135 LCD display using SPI
Port Expander	MCP23S17-E_SP; 16-bit input/Output Expander with SPI interface
Digital-to-Analog Converter	MCP4822-E/P; 2-channel DAC

Table 1. Prototype PCB components

3.2 PCB Block Diagram

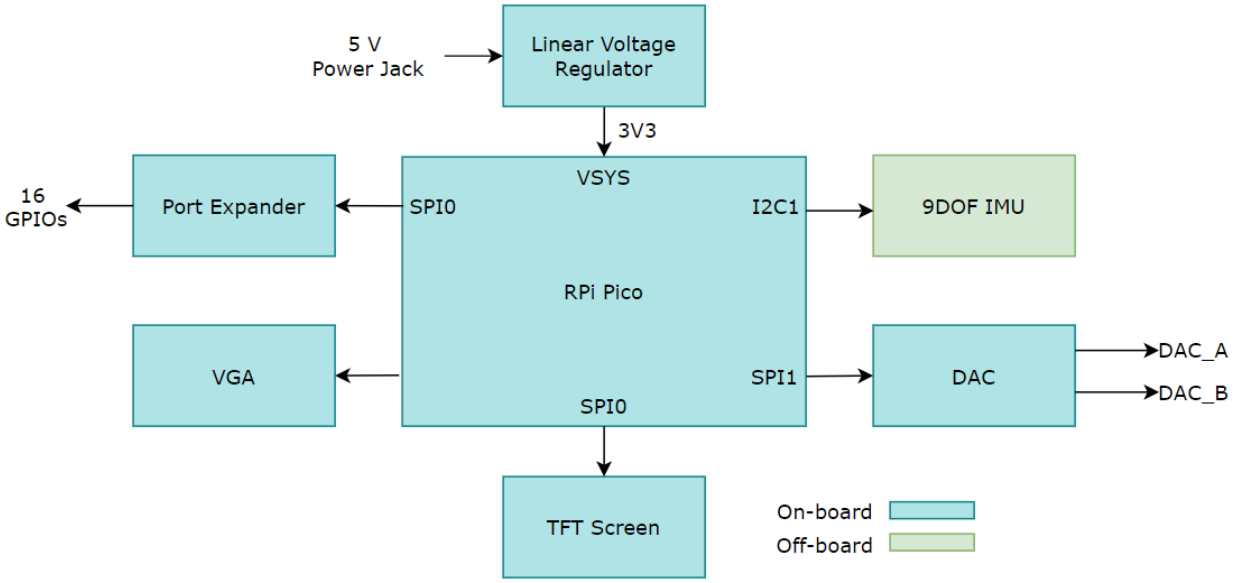


Figure 1. Block Diagram

3.3 PCB Physical Layout

The physical layout of the PCB shall give plenty of space for each component so students can solder by hand. The PCB is designed to have two layers where both the top and bottom are signal layers, and the grounds are connected via ground pours. The size of the PCB is 116.8 mm x 79.5 mm. The layout design considers the locations of ports and breakout headers and makes sure they are easily accessible.

All components on the PCB are through-hole components for easy assembly. All components are on the top layer with clear silkscreen labeled. Both the TFT screen and the RPi Pico can be snapped in via breakout headers; all critical pins on the RPi Pico and the port expander can be accessed via the breakout headers. The off-board IMU sensor can be easily attached/detached via the stemma header. Several jumpers are added for GPIO versatility such that the jumper can be disconnected for other GPIO purposes.

The prototype PCB is constructed with a 2-layer stack-up with a traditional through via design. We used the 2-layer prototype service at OSHPARK as our PCB manufacturer, and their standard material specification and stackup are shown below. No high-speed signals or critical paths need to be considered in this design.

3.3.1 PCB Component Placement

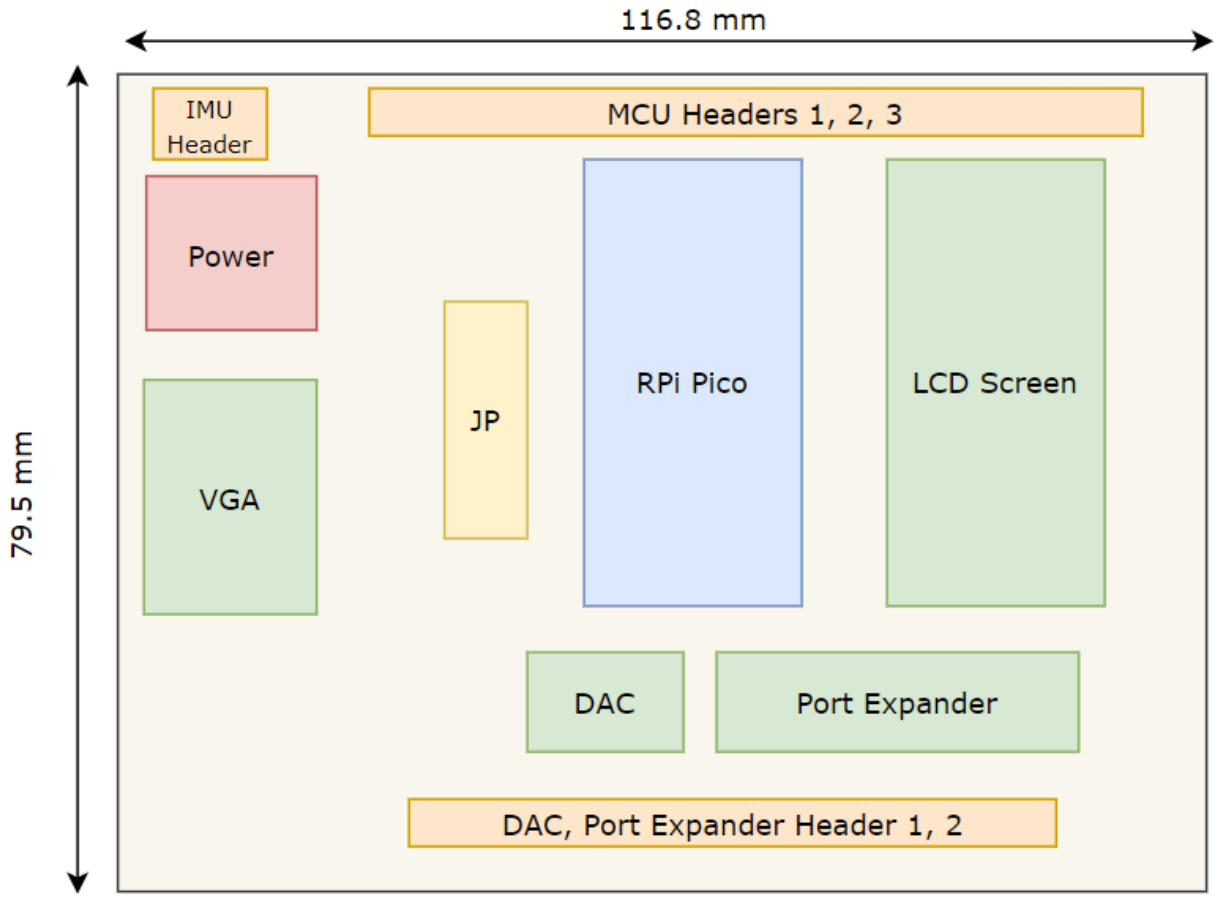


Figure 2. Layout Placement

3.3.2 PCB Stackup

Thickness	Layer	Tolerance
1mil (0.0254mm)	silkscreen	+/-0.2mil (0.00508mm)
1mil (0.0254mm)	solder resist	+/-0.2mil (0.00508mm)
1.4 mil (0.0356mm)	1 oz copper	
60 mil (1.5240mm)	core	+/-6mil (0.1524mm)
1.4 mil (0.0356mm)	1 oz copper	
1mil (0.0254mm)	solder resist	+/-0.2mil (0.00508mm)
1mil (0.0254mm)	silkscreen	+/-0.2mil (0.00508mm)

Figure 3. PCB Stackup

3.3.3 PCB Material Specification

Spec	Value	
Substrate	175Tg FR4	Kingboard KB6167F Datasheet
Board Thickness	63mil (1.6mm) nominal	
Dielectric	4.5 at 10Mhz	
Soldermask Color	Purple	Mask Datasheet
Minimum soldermask web	4 mil (0.1016mm)	
Maximum soldermask alignment	3mil (0.0762mm)	Covers retraction, expansion, and shift
Silkscreen minimum line width	5 mil (0.127mm) (recommended minimum) 3 mil (0.0762mm) (short lines, text, graphics)	Silkscreen Datasheet
Maximum board size	16in (406.4mm) by 22in (558.8mm)	
Minimum board size	0.25in (6.35mm) by 0.25in (6.35mm)	

Figure 4. PCB Material Spec

4 Electrical Specification

4.1 MCU

Designed by Raspberry Pi, RP2040 is a dual-core, ARM Cortex-M0+ processor with powerful internal peripherals. It is a low-cost, high-performance microcontroller device with 30 multifunction GPIO pins, a 4-channel ADC with an internal temperature sensor, a DMA channel to offload repetitive data transfer tasks, and two PLLs to provide a fixed 48MHz clock and a flexible system clock up to 133MHz.

The RP2040 has its own development board called the RPi Pico which is affordable and well-documented. The Pico exposes 26 multi-function GPIO pins from the RP2040 via breakout headers, making it extremely user-friendly for users of all skill levels. To program RP2040 using the Pico, one can simply drag and drop a file via the microUSB interface.

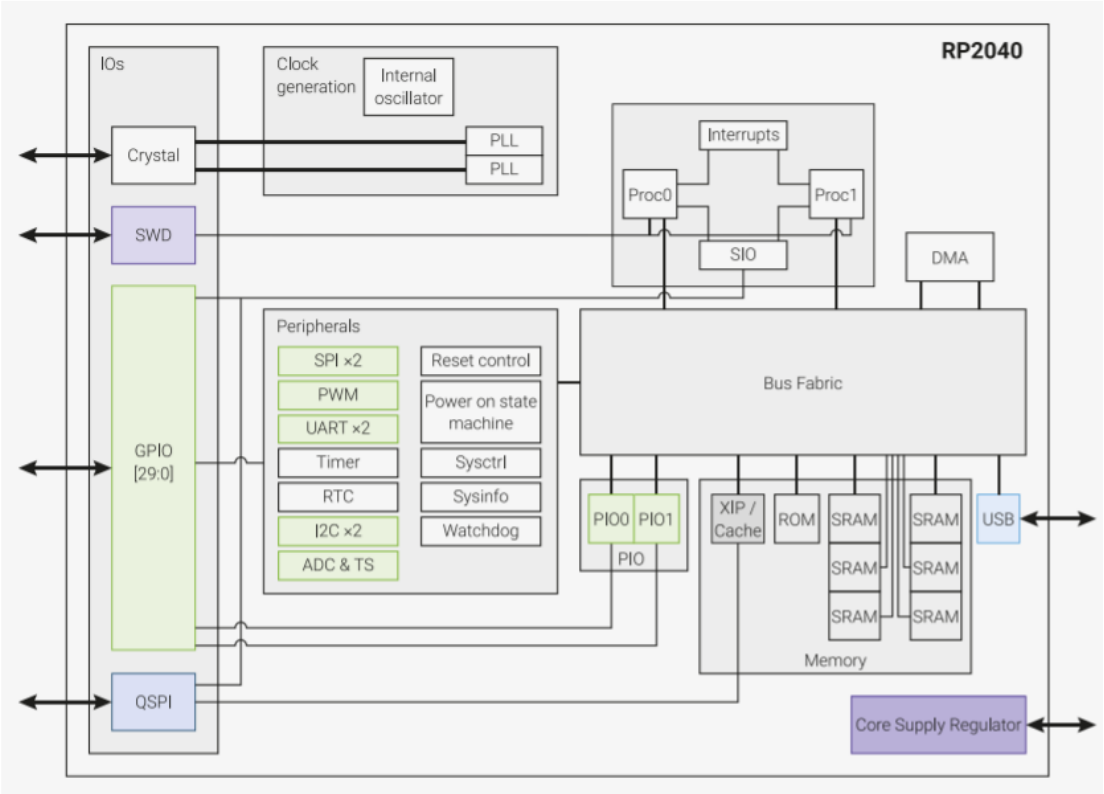


Figure 5. RP2040 Internal Block Diagram

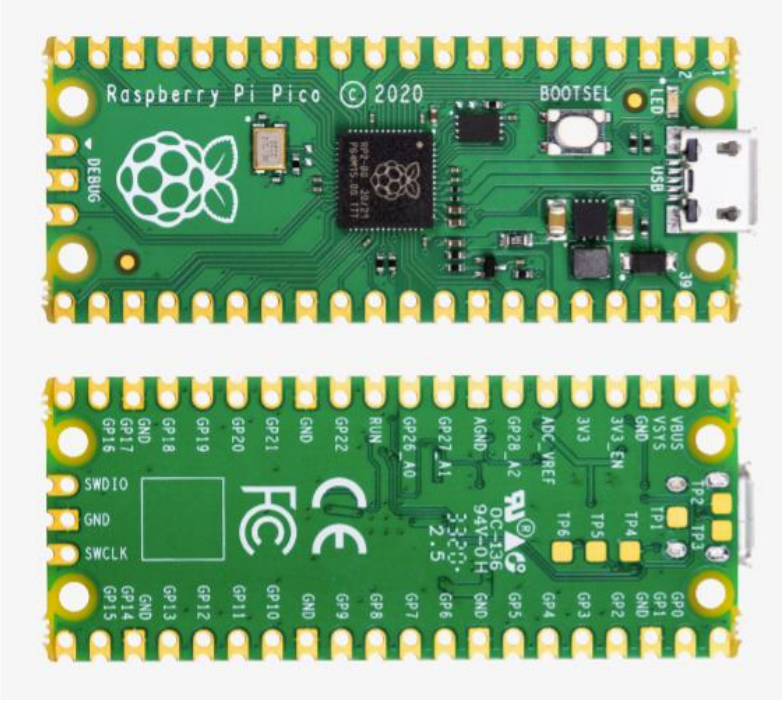


Figure 6. RPi Pico

4.1.1 MCU Power

The RP2040 has its own internal voltage regulator to supply the core voltage, and the RPi Pico has an onboard buck/boost switching regulator to generate 3.3V from the VSYS pin. The VSYS pin is the main system input voltage pin, which can vary in the allowed range 1.8 V to 5.5 V. The onboard regulator generates 3.3V to supply RP2040 and its I/O, and it has two modes: PFM mode and PWM mode. The default PFM mode has the best efficiency overall, and the PWM mode improves voltage ripples but at the cost of much worse efficiency at light loads.

The Pico is also compatible with using an external battery or power regulator via the VSYS pin. On the prototype PCB, a 5 V power supply is stepped down to 3.3 V using an external LDO, which then the voltage is used to supply the Pico and the rest of the onboard peripherals. Both the 5 V from the power supply and 3.3 V from the LDO are broken out as a header for potential use in student projects.

The linear voltage regulator used on the prototype board is the MCP 1702-3302E/TO by Microchip Technology. It has a 250 mA output current and a maximum dropout voltage of 0.725 V. LDO is the appropriate choice because it is cheaper and smaller than a switching regulator, and efficiency is not the priority on this protoboard.

Below shows the Pico power consumption when the board is using a VGA, a SD Card, and an Audio board.

Pico Board	**Average VBUS Current @ 5V (mA)**			**Maximum VBUS Current @5V (mA)**		
	Temperature (°C)			**Temperature (°C)**		
	-25	25	85	-25	25	85
#1	84.8	83.4	87.3	90.9	87.8	93.1
#2	87.5	89.9	89.8	93.4	94.1	94.0
#3	84.4	86.2	86.9	90.6	92.9	91.3
Mean	**85.6**	**86.5**	**88.0**	**91.6**	**91.6**	**92.8**

Figure 7. RPi Pico power consumption

4.1.2 MCU Serial Interface

The MCU supports UART, SPI, and I2C interfaces with two channels each. On the prototype PCB, the I2C protocol is used to communicate with the IMU sensor. The I2C interface can be used to transfer and/or receive data on a 2-wire bus network, and the RP2040 can operate as both master and slave. There are two pins for the I2C:

- SCL, clock output in master mode, input in slave mode
- SDA, data input/output pin

The I2C interface requires external pull-up resistors to work properly. In this case, the IMU sensor module has 10K pullup resistors on the board, but additional pullup resistors are needed if the I2C is used for other purposes.

Both SPI channels are used on the Pico. One of the SPI channels is used for the LCD, and the other SPI channel is shared between the DAC and the port expander. Each SPI channel has four pins:

- SCLK, clock output in master mode, input in slave mode
- CS_N, active low chip select
- TX, transmit data or MOSI
- RX, receive data of MISO

The SPI_0 channel is shared between the port expander and the DAC. The DAC does not connect to the RX line because it is a single-way transmission from the Pico to the DAC. Software manipulation is needed so the two devices can share the SPI_0 channel. The TFT screen uses the SPI_1 channel, and it doesn't use the RX line either due to its single-way transmission.

4.1.3 MCU Debug Interface

The RPi Pico uses Serial Wire Debug (SWD) which is a standard interface on Cortex-M-based microcontrollers. It can be used to reset the board, load code into flash, and set the code running. The RPi Pico exposes the RP2040 SW interface on three pins: GND, SWDIO, SWCLK. The host can use the SWD port to access RP2040 at any time without the need to manually reset the board.

4.1.4 MCU Pinout

Pin Number	GPIO Number	Microcontroller Function	Peripheral
1	GPIO0	UART0 TX/GPIO	UART
2	GPIO1	UART0 RX/GPIO	UART
3	N/A	GND	
4	GPIO2	SPI0 SCK/GPIO	Port expander SCK
5	GPIO3	SPI0 TX/GPIO	Port expander SI
6	GPIO4	SPI0 RX/GPIO	Port expander SO
7	GPIO5	SPI0 CSn/GPIO	Port expander CS
8	N/A	GND	
9	GPIO6	GPIO	Port expander INTA
10	GPIO7	GPIO	Port expander INTB
11	GPIO8	GPIO	PWM1
12	GPIO9	SPI1 CSn/GPIO	DAC CS
13	N/A	GND	
14	GPIO10	SPI1 CLK/GPIO	DAC CLK
15	GPIO11	SPI1 TX/GPIO	DAC TX
16	GPIO12	GPIO	VGA HSYNC
17	GPIO13	GPIO	VGA VSYNC
18	N/A	GND	
19	GPIO14	GPIO	VGA R
20	GPIO15	GPIO	VGA G
21	GPIO16	GPIO	VGA B
22	GPIO17	SPI0 CSn/GPIO	TFT CS
23	N/A	GND	
24	GPIO18	SPI0 SCK/GPIO	TFT SCK
25	GPIO19	SPI0 TX/GPIO	TFT TX
26	GPIO20	GPIO	TFT D/C
27	GPIO21	GPIO	TFT BL
28	N/A	GND	
29	GPIO22	GPIO	TFT RESET
30	N/A	RUN	
31	GPIO26	I2C SDA/GPIO	IMU SDA

32	GPIO27	I2C SDA/GPIO	IMU SCL
33	N/A	GND	
34	GPIO28	GPIO	PWM2
35	N/A	ADC_VREF	
36	N/A	3V3(OUT)	
37	N/A	3V3_EN	
38	N/A	GND	
39	N/A	VSYS	LDO output
40	N/A	VBUS	

Table 2. Pico Pin Mapping on Prototype PCB

4.2 Peripherals and I/O

4.2.1 Digital-to-Analog Converter

The DAC used on the prototype PCB is the MCP2822. It has a 12-bit resolution with two output channels, and it uses the SPI protocol to communicate with the RPi Pico. The DAC allows students to work with audio samples and analog outputs, and it can also be used as an excellent debugging tool via the oscilloscope.

The supply voltage range for the DAC is from 2.7 V to 5.5 V. On the prototype PCB, it uses 3.3 V from the output of the onboard LDO as supply. The current at the output pins is around 25 mA and the current at the input pins is around 2 mA; the input current is under the recommended I/O load of 300 mA on the RPi Pico.

The DAC package is shown in the figure below. The LDAC_N transfers the input latch registers to the DAC registers when low. It can also be tied low if transfer on the rising edge or CS_N is desired. In our design, the ladder option is used.

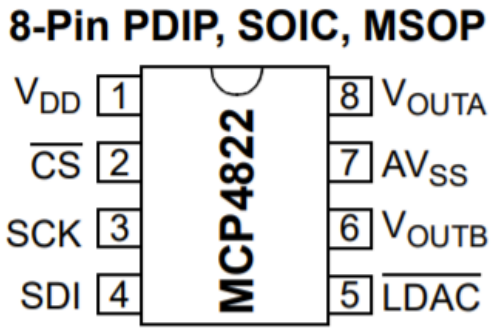


Figure 8. DAC Pin Mapping

The write command of the DAC is initiated by driving the CS pin low. This is followed by clocking the four configuration bits and the 12 data bits into the SDI pin on the rising edge of the SCK. The end of a transaction is signified by raising the CS pin. This causes the data to latch into the DAC input registers. All DAC writes are 16-bit words with the most significant four bits as config bits and the rest as data bits. The detailed description of the 16 bits is listed as below:

- Bit 15: DAC_A or DAC_B select bit; 1 = DAC_B, 0 = DAC_A
- Bit 14: don't care
- Bit 13: output gain select bit: 1 = 1x, 0 = 2x

- Bit 12: output power-down control bit; 1 = output power-down control bit, 0 = output is high-impedance
- Bit 11-0: DAC data bits

The DAC is configured at 40 kHz where it is called every 25 uS on the Pico's interrupt timer. The following function is used to construct a repeating timer that calls the callback function at the specified interval in uS. The repeated delay is set <0 so the delay is between the starting time of each callback.

```
add_repeating_timer_us (int64_t delay_us,
                        repeating_timer_callback_t callback,
                        void* user_data,
                        repeating_timer_t * out)
```

However, setting the delay to 25 uS (40 kHz) presents an assertion error shown in the figure below. The same error is present in Hunter's code. The DAC displays successfully when the delay is increased to 75 uS (13.33 kHz), and the suspicion is that the timer interrupt cannot achieve 40 kHz of speed. This problem can be solved by using Direct Memory Access (DMA).

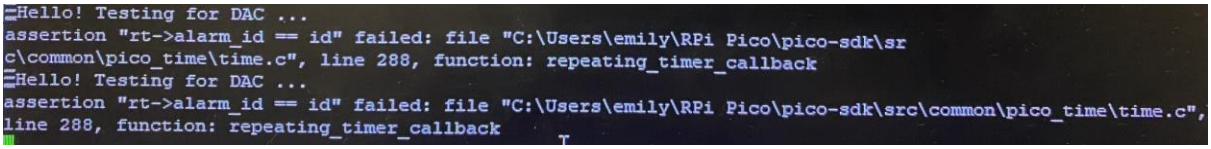


Figure 9. Assertion Error using > 13.33kHz during interrupt

When the timer interrupt is running at 13.33 kHz, a 400 Hz sine wave can be successfully sent over to both channels.

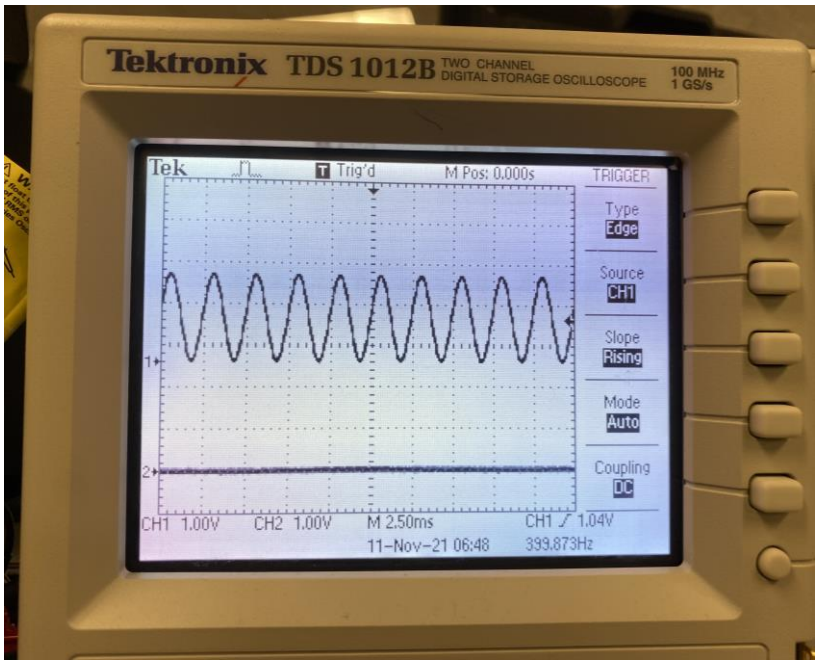


Figure 10. DAC output at 400Hz using 13.33kHz interrupt

4.2.2 Port Expander

The port expander is the MCP23S17 that features a 16-bit remote bidirectional I/O port. It has a high-speed SPI interface that has a maximum speed of 10 MHz, and the I/O pins can be configured as active-high, active-low, or open-drain ports using INTA and INTB pins. It operates from 1.8 V to 5.5 V.

The pin package for the port expander is shown in the figure below. The RESET_N pin is connected to the RUN pin on the RPi Pico, so it can reset whenever the Pico resets. The INTA and INTB are both connected to GPIO pins on the Pico. A0, A1, and A2 are all hardware address pins; they are externally biased to GND so that the address is set to 000.

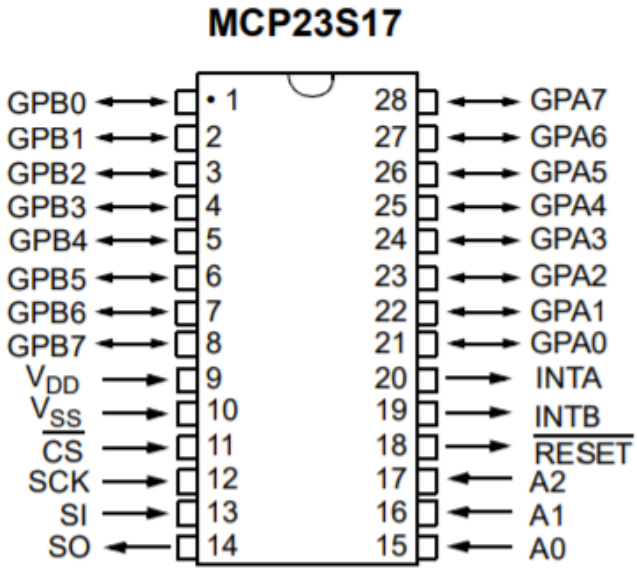


Figure 11. Port Expander Pin mapping

The MCP23S17 consists of multiple 8-bit configuration registers for input, output, and polarity selection. In a transaction, both the SPI read and write start by lowering the CS pin. The write command (hardware address with R/W bit cleared) is then clocked into the device for write, and the read command (hardware address with R/W bit set) is clocked into the device for read. Following the command, the register address is sent in for both reading and writing, and for SPI reading the data is read out to the buffer in the next cycle. The opcode format is shown in the figure below:

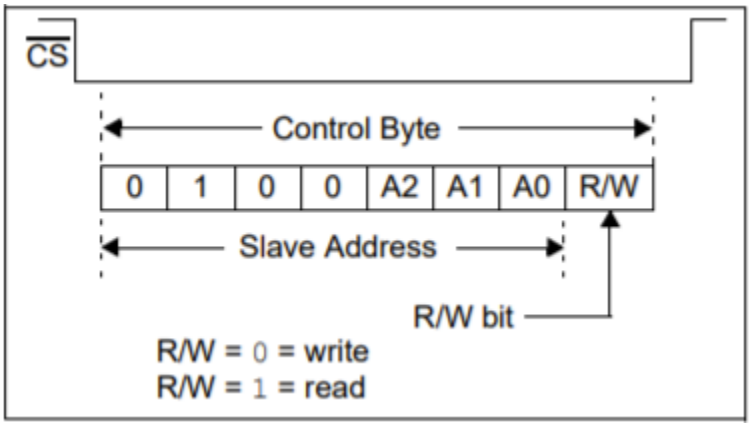


Figure 12. Port Expander Op Code Format

Before using the port expander, the IOCON register that contains several bits for configuring the device needs to be set up. The following explains each bit and their function:

- Bit 7: BANK bit; 1 = register associated with each port are separated into different banks, 0 = registers all use the same bank
- Bit 6: MIRROR bit; 1 = INT pins are internally connected, 0 = INTA and INTB associate with different ports
- Bit 5: SEQOP bit; 1 = sequential operation disabled, 0 = sequential operation enabled
- Bit 4: DISSLW pin; 1 = slew rate disabled, 0 = slew rate enabled
- Bit 3: HAEN pin; 1 = hardware address enabled, 0 = hardware address disabled
- Bit 2: ODR pin; 1 = open-drain output, 0 = active driver output (use INPOL)
- Bit 1: INPOL pin; 1 = active-high for I/O, 0 = active-low for I/O
- Bit 0: unimplemented

To read and write from/to the port expander, multiple helper functions are written for register configurations so students can simply call those functions. For more details, see the firmware section.

4.2.3 VGA

The VGA port used is L77HDE15SD1CH4F, and the design references [1]. The VGA monitor requires HSYNC and VSYNC signals for horizontal and vertical blanking timing. They can go on any GPIO, as long as they are next to each other. Three bits are used for R, G, B coloring. A design constraint it has is that the VGA PIO software requires R, G, B signals to be on contiguous (consecutive numerical order) GPIOs on the RPi Pico, with the sequence being Red first, then Green, then Blue.

4.2.4 TFT Screen

The TFT display is designed specifically for the RPi Pico using the SPI bus. It has 240x135 resolution and four key buttons for any gaming application. It has the same dimension as the RPi Pico and can be plugged directly into the Pico via the breakout headers. On our protoboard, they are placed next to each other for ease of wiring.

4.2.5 Power Regulator

The LDO on the prototype PCB is MCP1702 from Microchip Technology. It takes in 5 V from the power supply and regulates it to 3.3 V stable system voltage. It has a low dropout voltage at around 0.725 V. The power supply can be switched on/off via an SPDT slide switch, and it uses a diode to prevent backward current flow. The LDO implements both input and output decoupling capacitors as the design recommended.

4.2.6 Breakout Headers and Jumpers

The prototype PCB breaks out all the pins from the RPi Pico and the port expander so students can access them easily. Additionally, the DAC outputs, GND, and power pins are also broken out. These breakout headers are placed at the edge of the board for convenient access.

Additionally, jumpers are used to increase GPIO versatility. Some pins (VGA pins, for instance) are only used for certain applications, and they can be disconnected via jumpers to free up additional GPIO pins on the Pico.

5 Firmware

The current development board has a functional set of firmware using the PIC32, and some parts need to be updated when transferring the code over to the RP2040. The changes are minimal since all the hardware peripherals have been kept the same on the new prototype board.

5.1 SPI

I worked on the firmware for two modules: the DAC and the Port Expander. Both modules use the SPI interface to communicate with the Pico, and conveniently Pico has a very nicely documented SPI library [2]. To use the library, one must include the header files in the C file as well as the CMakeList as shown below:

```
#include <stdio.h>
#include <math.h>
#include "pico/stdlib.h"
#include "hardware/spi.h"
```

Libraries needed SPI, C file

```
# Pull in our pico_stdlib which pulls in commonly used features
target_link_libraries(port_expander pico_stdlib hardware_spi hardware_base)
```

Configuration needed in the CMakeList, CMakeLists

As mentioned in the previous section, the Serial Peripheral Interface is a master or slave interface for synchronous serial communication with a peripheral device. On the prototype PCB, the Pico always acts as a master device where the rest of the SPI peripherals are slave devices.

Each SPI controller can be connected to a number of GPIO pins (TX, RX, SCK, CS). The SPI pins need to be defined along with their corresponding SPI port. The following configures SPI channel 1 with baud rate set to 10MHz and data bits set to 8 bits. For both the port expander and the DAC, the CS pin is required to be active-low so it is initialized as high during setup.

```
//SPI configuration (SPI1)
#define PIN_CS   9
#define PIN_SCK  10
#define PIN_MOSI 11
#define PIN_MISO 12
#define SPI_PORT spi1
//set SPI (either in a setup function or main)
// Initialize SPI channel (channel, baud rate set to 10MHz)
spi_init(SPI_PORT, 10000000) ;
//Format (channel, data bits per transfer, polarity, phase, order)
spi_set_format(SPI_PORT, 8, 0, 0, 0);
// Map SPI signals to GPIO ports
gpio_set_function(PIN_SCK, GPIO_FUNC_SPI);
gpio_set_function(PIN_MOSI, GPIO_FUNC_SPI);
gpio_set_function(PIN_MISO, GPIO_FUNC_SPI);
gpio_set_function(PIN_CS, GPIO_FUNC_SPI);

//initialize CS pin high
```

```
gpio_init(PIN_CS);
gpio_set_dir(PIN_CS, GPIO_OUT);
gpio_put(PIN_CS, 1);
```

The 8-bit SPI transaction can be done using the following functions. Another set of functions is available for 16-bit SPI transactions (see [2]). Note that the write function will automatically discard any data received back (junk), so junk data doesn't have to be taken out of the receiver manually after each transaction. Similarly, the read function sends in a repeated_tx_data until data is read in from RX.

```
int spi_write_blocking(spi_ins_t *spi, const uint8_t *src, size_t len)

int spi_read_blocking(spi_inst_t *spi, uint8_t repeated_tx_data,
uint8_t *dst, size_t len)
```

The implementation detail for the DAC and port expander are described below.

5.2 DAC

First, the DAC parameters are defined at the top of the program for configuration.

```
//DAC parameters
// A-channel, 1x, active
#define DAC_config_chan_A 0b0011000000000000
// B-channel, 1x, active
#define DAC_config_chan_B 0b1011000000000000
```

A DDS sine table is implemented to send to the output of the DAC [3]. In this application, we are outputting a 400 Hz sine wave using 256 data bits for each period. The sampling frequency is defined at 13.33 kHz due to the software limitation mentioned in section 4.2.1.

In the main function, the SPI channel is set up with a 20 MHz baud rate and 16-bit data transfer. Since the DAC is a single-way device, only the SCK, CS, and MOSI(TX) lines are configured. The main function then calculates the sine table. Lastly, a timer is constructed, and a repeating timer callback function is used to calculate the DAC data every 75 uS.

```
    //create a timer
    struct repeating_timer timer;
    //add a timer at 40MHz, and call back to the DAC function
    //timer in us, the callback function it goes to, user data sent, pointer to store the repeating timer info
    add_repeating_timer_us(-75, repeating_timer_callback_core_0, NULL, &timer);
```

In the callback function, a single data point from the sine table is extracted and is then sent over to the configured SPI port.

```
// DAC callback function.  The function outputs a sine wave on the DAC_B channel
bool repeating_timer_callback_core_0(struct repeating_timer *t){
    // DDS phase and sine table lookup
    phase_accum_main_0 += phase_incr_main_0;
    DAC_data_0 = (DAC_config_chan_A | ((sin_table[phase_accum_main_0>>24] + 2048) & 0xffff))  ;
    //printf("DAC data is %d \n", DAC_data_0);
```

```
    // SPI write (no spinlock b/c of SPI buffer)
    spi_write16_blocking(SPI_PORT, &DAC_data_0, 1) ;
    return true;
}
```

5.3 Port Expander

The port expander implemented multiple helper functions so students can call each one based on the desired configuration. The three main functions that need to be modified for RP2040 are initPE(), readPE(), and writePE().

In initPE(), the SPI channel is set up with a baud rate of 10MHz and 8-bit data transfer. All four SPI lines are configured, and the CS pin is initialized as high. The IOCON register is configured using the writePE() function where all the bits are cleared except for the sequential operation bit.

```
// initialize PE using SPI
void initPE(){
    //set SPI
    // Initialize SPI channel (channel, baud rate set to 10MHz)
    spi_init(SPI_PORT, 10000000) ;
    // Format (channel, data bits per transfer, polarity, phase, order)
    spi_set_format(SPI_PORT, 8, 0, 0, 0);
    // Map SPI signals to GPIO ports
    gpio_set_function(PIN_SCK, GPIO_FUNC_SPI);
    gpio_set_function(PIN_MOSI, GPIO_FUNC_SPI);
    gpio_set_function(PIN_MISO, GPIO_FUNC_SPI);
    gpio_set_function(PIN_CS, GPIO_FUNC_SPI);

    //initialize CS pin high
    gpio_init(PIN_CS);
    gpio_set_dir(PIN_CS, GPIO_OUT);
    gpio_put(PIN_CS, 1);


    //set IOCON register
    while(!spi_is_writable(SPI_PORT));
    writePE(IOCON,( CLEAR_BANK   | CLEAR_MIRROR | SET_SEQOP |
                CLEAR_DISSLW | CLEAR_HAEN   | CLEAR_ODR |
                CLEAR_INTPOL ));
}
```

In writePE(), the function takes in a register address and the data that needs to be written. The write function first writes the opcode with the write bit enabled; it then sends the register address and data. Since this is a sequential operation, all messages are put in a buffer and can be transmitted together. The CS line is pulled low before the transaction and is returned to high after the transaction.

```
// write data to the port expander pins using reg_addr
static void writePE(uint8_t reg_addr, uint8_t data){
```

```
    uint8_t buf[3];
    buf[0] = PE_OPCODE_HEADER | WRITE; //remove read bit
    buf[1] = reg_addr;
    buf[2] = data;
    //clear CS
    cs_select();

    //send SPI opcode and write
    spi_write_blocking(SPI_PORT, buf, 3);
    while(spi_is_busy(SPI_PORT)); // wait until job is done

    //CS high
    cs_deselect();
    printf("writing: \n");
    printf("op code is %d\n", buf[0]);
    printf("reg addr is %d\n", buf[1]);
    printf("written data is %d\n", buf[2]);
}
```

The readPE() function is similar to writePE() except it returns the data read from the register. The function takes in a register address, and it send the opcode with read bit enabled and the register address to the Pico. The spi_read function is used to retrieve the desired register value.

```
static uint8_t readPE(uint8_t reg_addr){
    uint8_t buf[2];
    buf[0] = PE_OPCODE_HEADER | READ; //remove read bit
    buf[1] = reg_addr;
    uint8_t out = 0;
    //clear CS
    cs_select();

    // OPCODE and HW Address (Should always be 0b0100000), LSB as read
    spi_write_blocking(SPI_PORT, buf, 2);
    while(spi_is_busy(SPI_PORT)); // wait until job is done

    // read from the reg_addr
    spi_read_blocking(SPI_PORT, 0, &out, 1);  //sending in junk
    while(spi_is_busy(SPI_PORT)); // wait until job is done
    //CS high
    cs_deselect();
    printf("reading: \n");
    printf("op code is %d\n", buf[0]); //1000001 binary
    printf("reg addr is %d\n", buf[1]); //send reg_addr
    printf("out is %d\n", out); //out is 0
    return out;
}
```

Other functions are directly copied over from the original port expander code [4].

To test the code, I set all the pins on Port A as output pins and toggled them in the while(1) loop. I probed each GPIO output with an oscilloscope and see if the write function was working properly. After some debugging, the oscilloscope reading showed CS line communicating and the GPIO output toggle between 0 and 1. As shown in the figure below, the CS pin is high unless a transaction happens.

```
int main(){
   stdio_init_all();
   printf("Testing Port Expander\n");
   initPE();

   mPortASetPinsOut(0b11111111);    //Set portA as output
   mPortBSetPinsIn(0b11111111);    //Set portA as input
   mPortBEnablePullUp(0b11111111);    //enable portB as all high's

   while(1){
      printf("*****************************setting A0****************************** \n");
      setBits(GPIOA, 0b11111111); //set pin A0
      sleep_ms(1000);
      printf("*****************************reading portB****************************** \n");
      portB_val = readPE(GPIOB); //should be all high
      printf("portB val is %d\n", portB_val);
      sleep_ms(1000);
      printf("*****************************clearing A0****************************** \n");
      clearBits(GPIOA, 0b11111111); //clear pin A0

      sleep_ms(1000);
   }
}
```

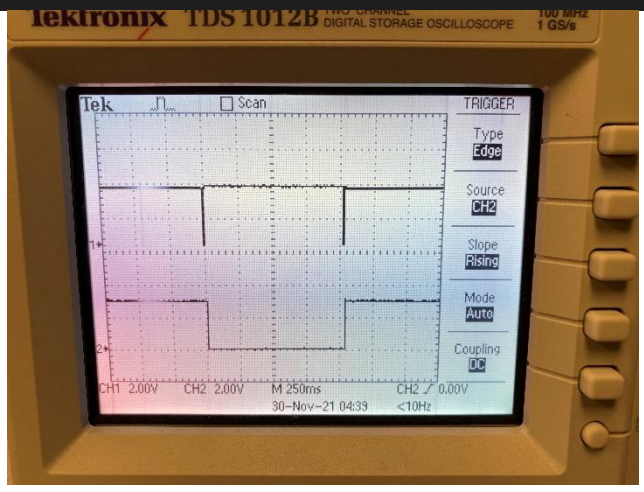


Figure 13. Port Expander working with oscilloscope reading. CH1 is CS, CH2 is A0 output that's toggling.

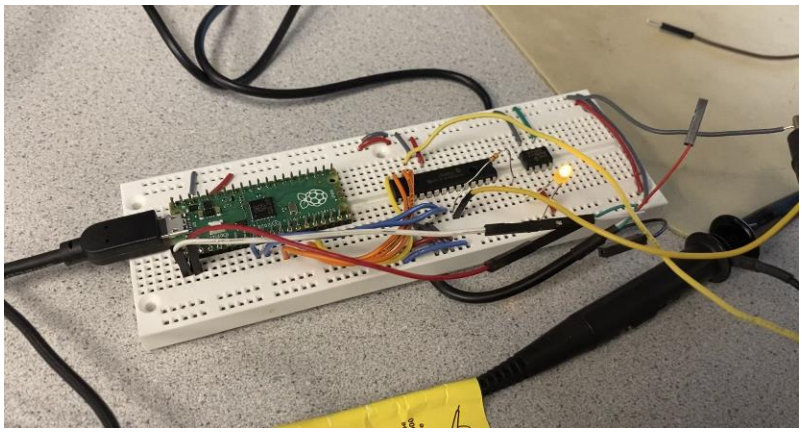


Figure 14. Hardware setup to test the port expander

During my debugging process, I realized that the original port expander setup on SPI channel 0 wasn't working. There were no activities on any of the lines. I then switched to SPI channel 1 which was working previously for the DAC device, and the write function began to work. I believe that there were some configurations missing for SPI channel 0. Further investigation is needed.

I wasn't able to get the read function to work on the port expander. I set up Port B as input ports and enabled pull-up on all the pins, and they were indeed probed as high. However, when I read GPIOB using the read function, the register value appears to be 0. I debugged this by first putting the RX line to high and examining the register value. The register value was 0xff which means the SPI was working correctly. Further investigation is needed on the port expander read function.

6 Future work

Currently, the prototype board is still under manufacturing. If it comes back in time before the semester ends, I would like to assemble the board and perform testing. Additionally, the port expander read function needs to be further investigated because it cannot read any register values. Additional firmware also needs to be written for other peripherals such as the VGA display and the TFT screen.

7 Conclusion

Overall, I was able to reach my goal of designing the prototype PCB and writing demo code for several hardware peripherals. However, I wanted to assemble and test the PCB before I leave, but that wasn't achieved due to time constraints.

Despite the project being only one semester long, I gained great technical and personal insights. I was able to experience a full product design cycle from brainstorming and setting requirements, to designing and programming the hardware. I learned to appreciate the art of product management and how crucial it is to prepare and research early on so each stage can transition smoothly. I also gained knowledge I firmware development. I have always been a hands-on learner, but writing firmware for the hardware that I worked on gave me a more in-depth understanding of the project. Lastly, doing individual work has been very refreshing and rewarding because I get to appreciate my own work and develop my ways of problem-solving. As an engineer, I've spent most of my time working in a group or under direct guidance, and having to solve challenging problems on my own for this project has made me more independent and confident in future studies.

8 References

[1] https://datasheets.raspberrypi.com/rp2040/hardware-design-with-rp2040.pdf

[2] https://datasheets.raspberrypi.com/pico/raspberry-pi-pico-c-sdk.pdf

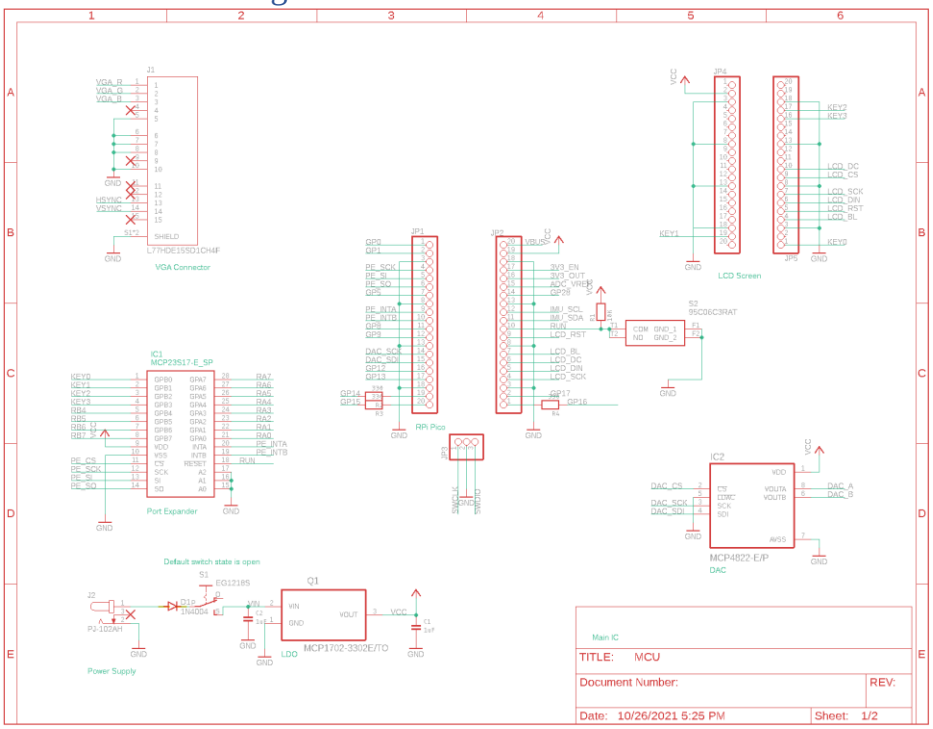
[3] https://vha3.github.io/Pico/Multi/MultiCore.html

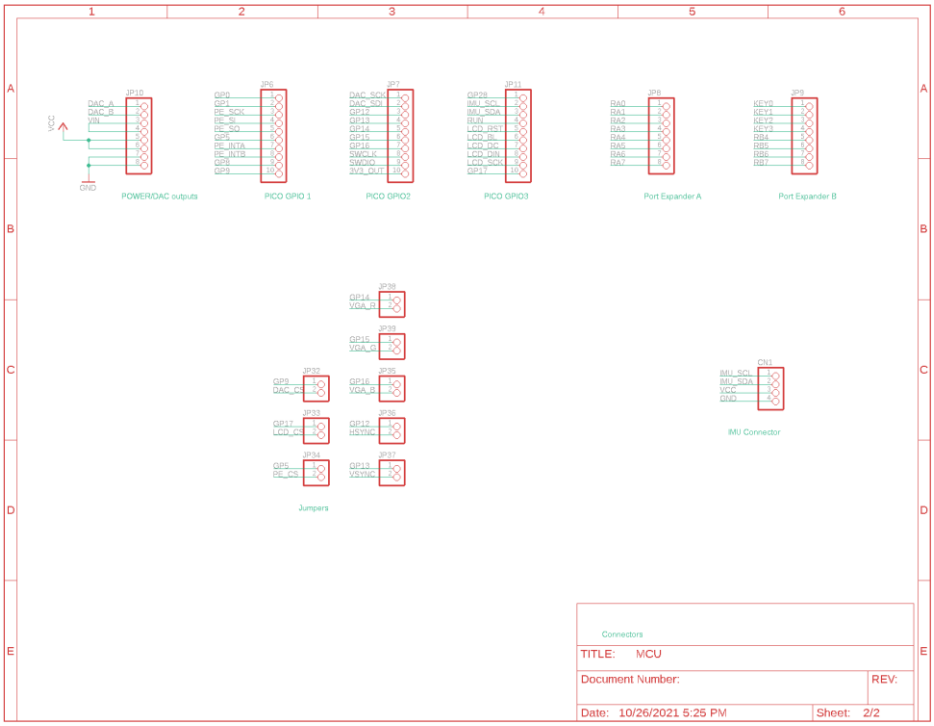
[4] https://people.ece.cornell.edu/land/courses/ece4760/PIC32/index_port_expander.html

[5] https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf

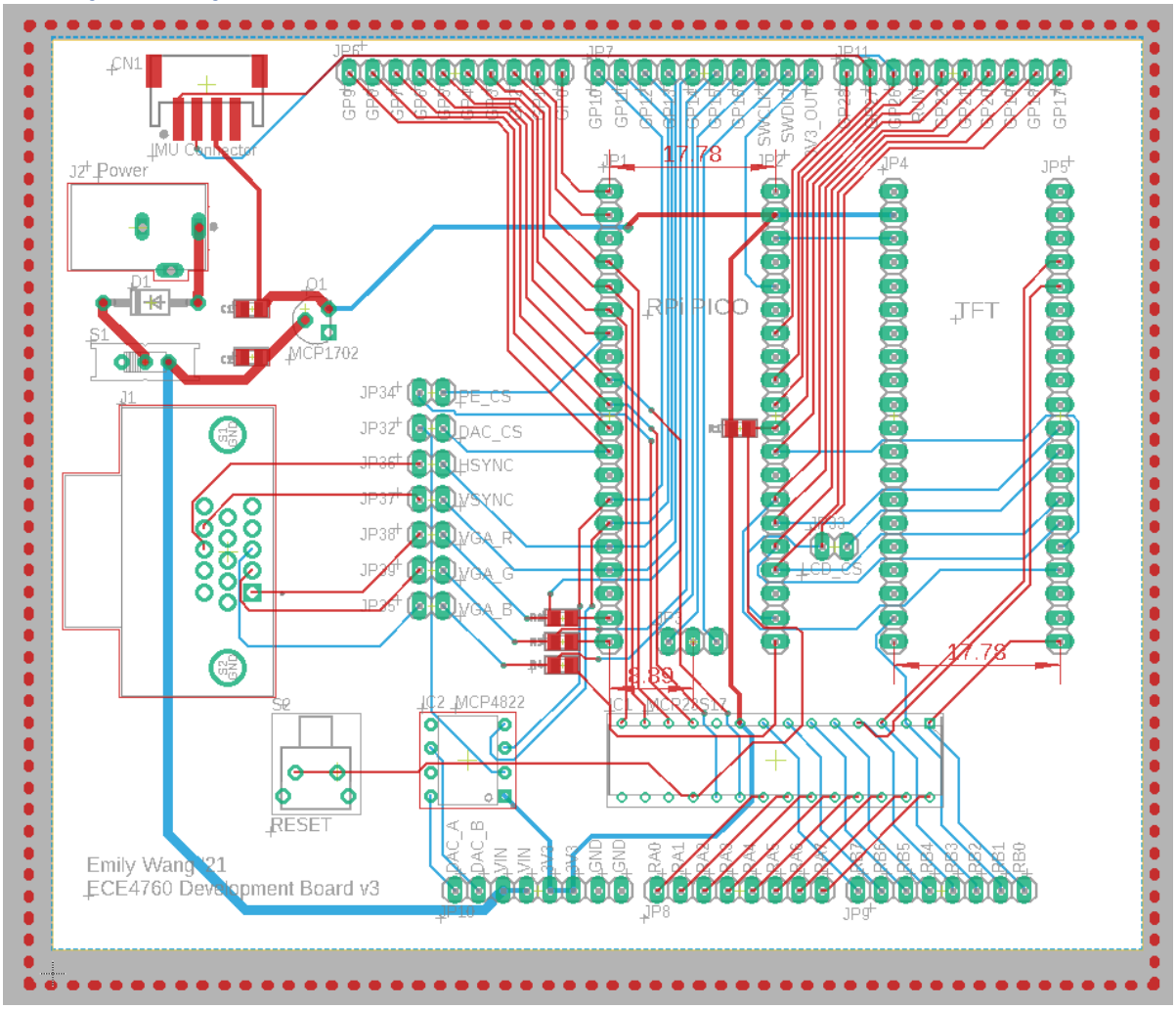
9 Appendix

9.1 Schematic Design





9.2 Physical Layout



9.3 Bill of Materials

Part	Parts Number	Description	Digikey Link	Quantity per board	Total Units to order	Total Price
C1,C2	CC0805JKX7R7BB105	Ceramic Cap, 0805, 1uF	https://www.digikey.com/en/products	2	20	3.98
CN1		JST 4Pin Connector	https://www.adafruit.com/product/43	1	10	3.5
D1	1N4004	Input diode	https://www.digikey.com/en/products	1	25	2.48
IC1	MCP23S17-E_SP	Port Expander	https://www.mouser.com/ProductDet	1	5	8.15
IC2	MCP4822-E/P	DAC	https://www.mouser.com/ProductDet	1	5	19.65
J1	L77HDE15SD1CH4F	VGA Port	https://www.mouser.com/ProductDet	1	5	8.3
J2	PJ-102AH	Power Jack	https://www.digikey.com/en/products	1	5	3.8
Q1	MCP1702-3302E/TO	Linear Regulator	https://www.digikey.com/en/products	1	5	2.9
R1	RMCF0805JT10K0	Resistor, 0805, 10K	https://www.digikey.com/en/products	1	10	0.2
R2, R3, R4	RMCF0805JT330R	Resistor, 0805, 330	https://www.digikey.com/en/products	3	20	0.4
S1	EG1218S	Input Power Switch	https://www.digikey.com/en/products	1	5	3.8
S2	95C06C3RAT	Reset switch	https://www.mouser.com/ProductDet	1	5	2.35
Others						
IMU Sensor	ICM 20948	9 DOF	https://www.adafruit.com/product/45	1	2	29.9
PH to SH cable		Converting PCB PH to IMU SH	https://www.adafruit.com/product/44	1	2	1.9
					Total	91.31

[BOM Source]

9.4 Acronym Table

ACRONYM	DEFINITION
ADC	Analog-to-digital converter
DAC	Digital-to-analog converter
DDS	Direct Digital Synthesis
DMA	Direct Memory Access
GPIO	General Purpose Input Output
I2C	Inter-Integrated Circuit; a communication protocol
IC	Integrated Circuits
IMU	Inertial Measurement Unit
LCD	Liquid Crystal Display
LDO	Low-drop power regulator
MISO	Master In Slave Out
MOSI	Master Out Slave In
PCB	Printed Circuit Board
PLL	Phase Lock Loop
PWM	Pulse Width Modulation
RX	Receive X
SPI	Serial Peripheral Interface; a communication protocol
TFT	Thin Film Transistor
TX	Transmit X
VGA	Video Graphic Array