Piston 2 User Guide

Last Updated: 01/10/2022

Hardware Overview

The required hardware components to use the Piston 2 motherboard are a 9V DC power supply, a 5V DC power supply, a programable waveform generator (clock supply), AD5686R Eval Board to supply select voltage references, a Teensy 4.1 microcontroller, Cerebro 2 chip on board (CoB) packages, and miscellaneous cables.

Hardware Setup

To begin, connect the 9V power supply to the Piston motherboard (before inserting the package) via the banana plug receptacles on the top right of the PCB. Now verify all on-board voltage references, circled in blue in Figure 2, are configured to the correct value as listed in Table 1 and the 2-pin jumpers are connected. Leave the jumpers for the off-board voltage references, circled in yellow, disconnected as these will be supplied by the AD5686R DAC evaluation board for better ReRAM programming. However, the on-board reference ICs can be used if needed during debugging or when utilizing more basic ReRAM programming techniques. The unlabeled voltage reference test points (those closest to the jumpers) are connected to the output on the onboard voltage reference IC for a given net. The labeled voltage reference test points are directly connected to the package socket. These two nets are connected when the 2-pin jumper is installed. The same applies for the power supply test points (circled in orange in Figure 2) on the right side of the board. The 1.2V and 3.3V nets supply power for the digital circuitry in the chip. The 3.3VA test point supplies a separate 3.3V supply to the on-board current references. These supplies should be set to the appropriate voltage as listed in Table 1 before configuring the current references.

Net	Voltage (V)	Net	Voltage (V)
V_ADC_T	0.700	VDDL	1.200
VDDWL	3.300	V_DAC_T	2.900
V_DAC_B	0.500	V_READ	1.200
VCM	0.900	V_RESET	3.000
V_SET	3.000	V_FORM	3.000
VDDH	3.300	1.2V	1.200
3.3V	3.300	3.3VA	3.300

Table 1: Voltage source values.

Next configure the current sources. To set these without a chip installed a test structure exists for each current source to set the desired output for each net. To measure the current, remove the 2-pin jumper from the given source and connect a current meter with microamp precision between the middle test point and the test structure test point (circled in dashed pink in Figure 2), with the test structure being on the low potential side. Like the voltage references the labeled test point connects directly to the socket and the unlabeled middle test point is the current source IC output. Configure each current reference to the value specified in Table 2. After configuring a current reference, install the 2-pin jumper connecting the two pins on the header closest to the labeled package test point.

Net	Current (µA)
I_TIA_T#	8.0
I_TIA_B#	8.0
I_DAC_#	10.0

Table 2: Current source values.

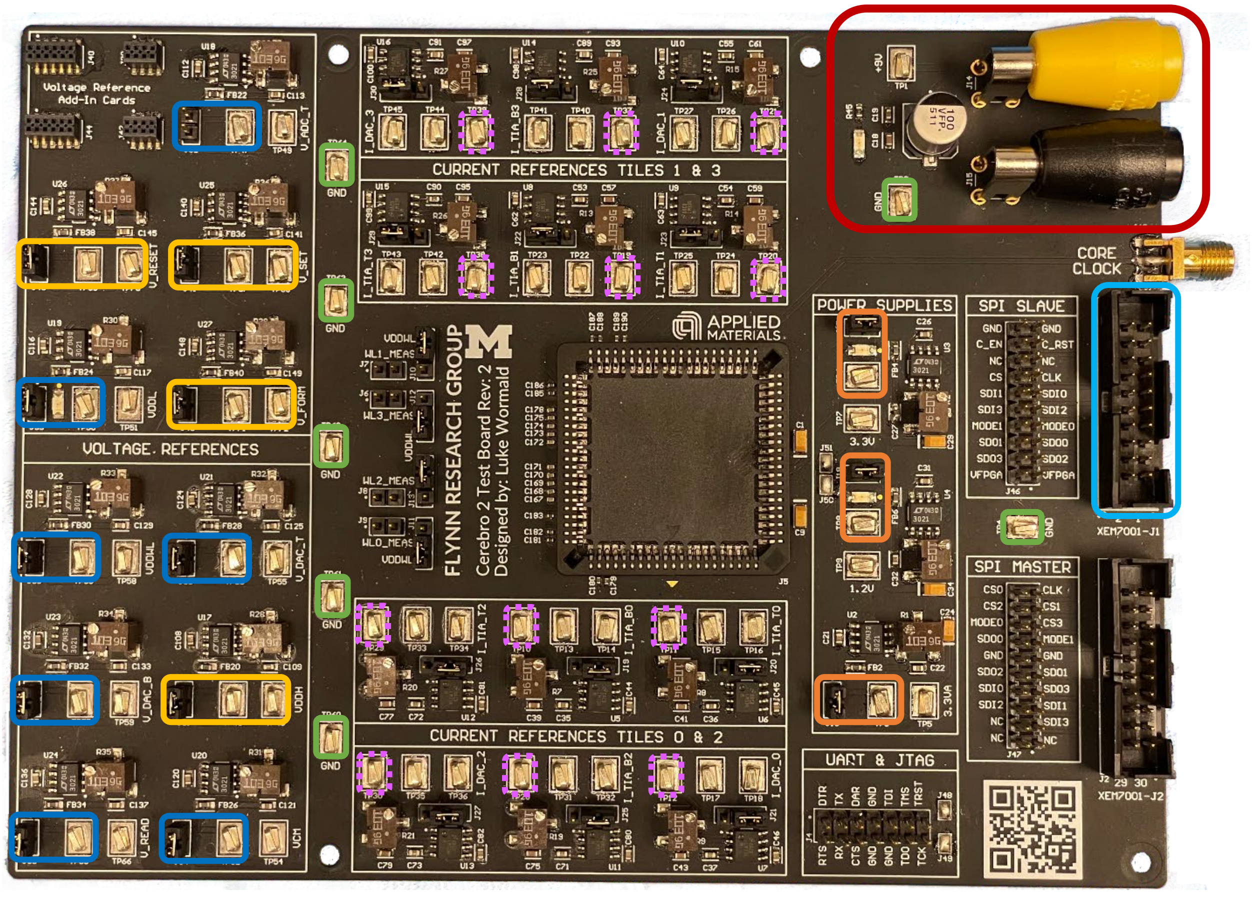


Figure 1: Piston 2 motherboard callouts.

9V Power Supply Connectors	🟥
On-Board Power Supplies	🟧
On-Board Voltage References	🟦
Off-Board Voltage References	🟨
Current Source Test Structures	🟪
Ground Test Points	🟩
Teensy Ribbon Cable Connector	🟦

Table 3: Figure 2 PCB callout color codes.

After configuring the voltage and current supplies, the Cerebro 2 CoB package can be installed in the Piston 2 motherboard. To install a package, align the gold triangles on the package and the motherboard and insert the chip into the socket. Then press firmly on opposite corners of the package until it is fully inserted into the socket. To remove a package, use a PLCC-84 extraction tool, being extra careful as the substrate is thinner that a typical PLCC package and may not exit the socket cleanly.

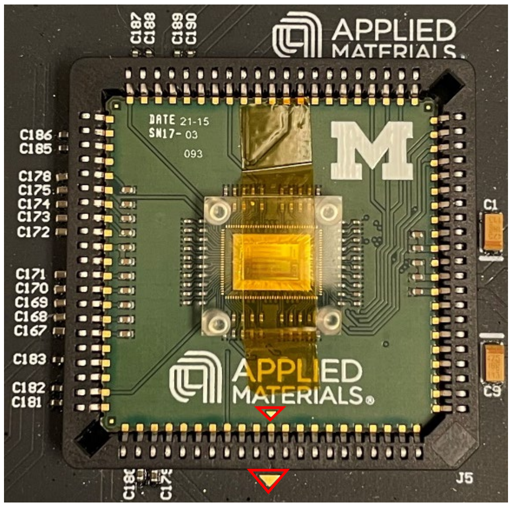


Figure 2: Cerebro 2 CoB package inserted into the motherboard. (Alignment arrows outlined in red.)

Now set up the Teensy microcontroller and external DAC evaluation board. To begin with insert the Teensy into the provided breakout board as shown in Figure 3. This breakout PCB has a keyed 20-pin ribbon cable connector (circled in blue in Figure 3) for easy connection to the Piston motherboard and a breakout header to connect to the external DAC (circled in yellow in Figure 3).

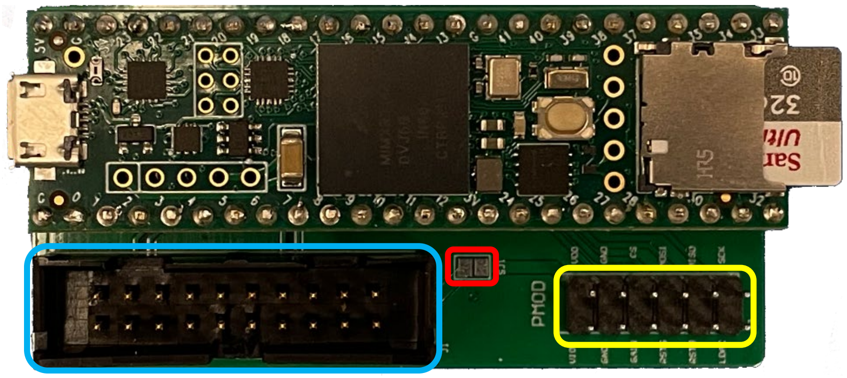


Figure 3: Teensy 4.1 installed into its breakout board.

First connect the Teensy breakout board header to the matching one on the AD5686R DAC evaluation board (circled in yellow in Figure 4). These headers are labeled PMOD and are a one-to-one connection. The DAC evaluation board requires 5V power. This can be supplied by a discrete DC power supply via the screw terminal connector circled in red in Figure 4 or via the Teensy PMOD from the 5V USB connection by closing the solder jumper on the Teensy breakout board circled in red in Figure 3. Be sure to configure the power selection jumper on the DAC evaluation board per its data sheet. Configure all other jumpers on the evaluation board as shown in Figure 4. Then connect the 20-pin ribbon cable between the Teensy breakout board and the Piston motherboard. Both connectors are keyed to ensure proper connection.

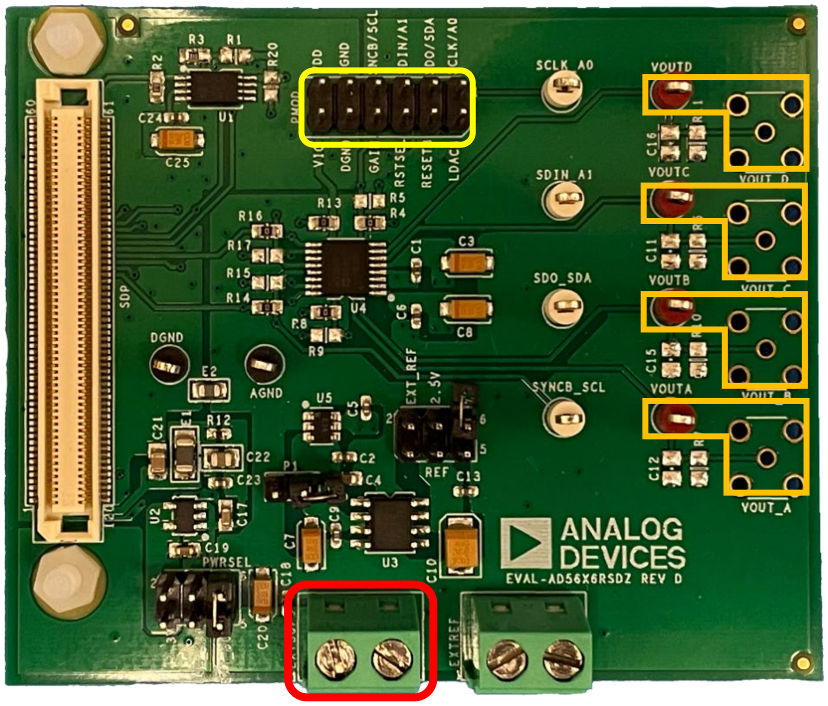


Figure 4: AD5686R evaluation board.

The four voltage references produced by the DAC can be connected to the Piston motherboard by clipping a wire to the test points outlined in amber in Figure 4 and connecting it to the bottom pin of the two pin jumpers or labeled test points connected to socket circled in amber in Figure 1. For proper use of the provided example code, ensure that each DAC channel (A, B, C, and D) are connected to the correct net as outlined in Table 4.

Net	DAC
V_SET	A
V_RESET	B
V_FORM	C
VDDH	D

Table 4: External DAC connections.

Finally, connect the Teensy to the host computer with a micro-USB cable and connect the clock to the Piston motherboard via the SMA connector edge mounted on the right of the board. The clock should be configured as a high impedance, 3.3V peak to peak sinewave with a 1.65V DC offset. The clock can run up to 64 MHz. Running the clock closer to 7 MHz is recommended during ReRAM programming. The end result should look similar to Figure 5.

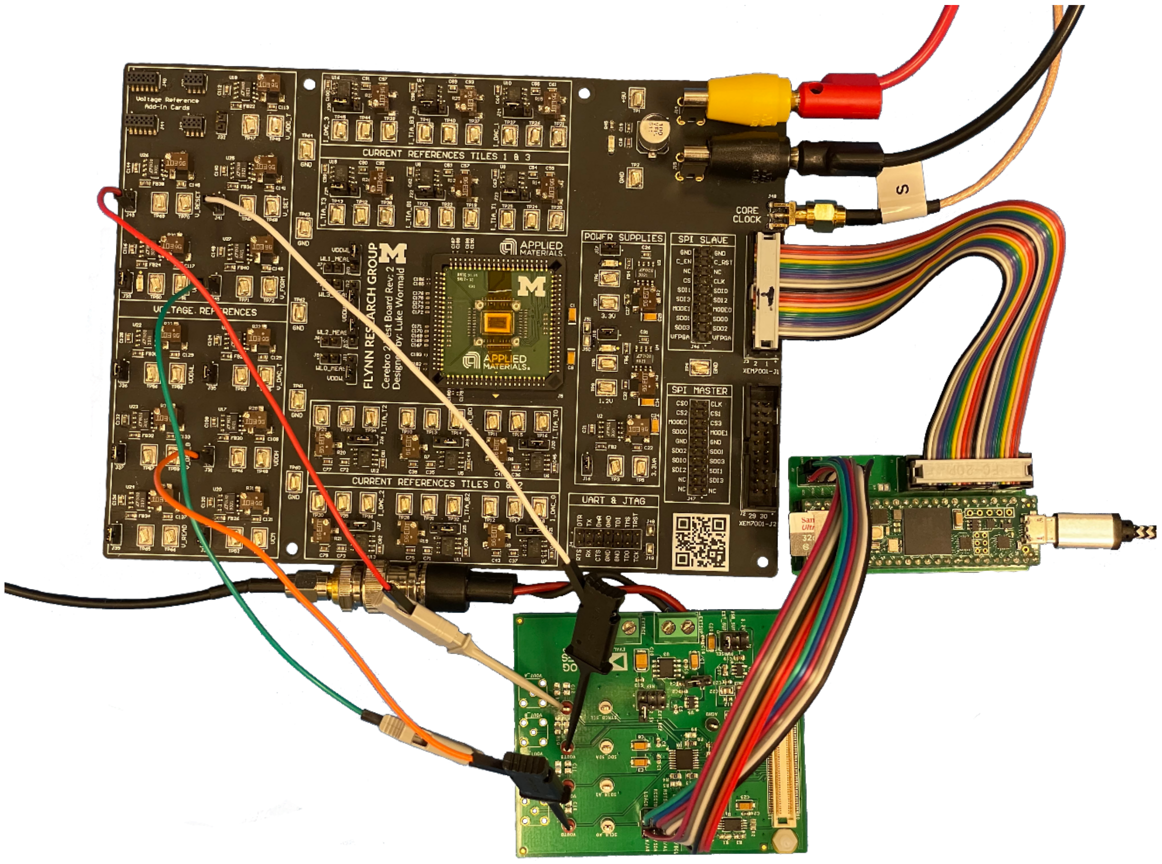


Figure 5: Complete test setup with external DAC and Teensy.

Software Setup

The IDE required to use the provided code repository is Visual Studio Code (VS Code) with the PlatformIO extension installed. VS Code is platform agnostic and can run on Windows, Linux, or MacOS on either x86 or ARM processors. First install VS Code from here, https://code.visualstudio.com/#alt-downloads, with the version appropriate for your operating system and hardware.

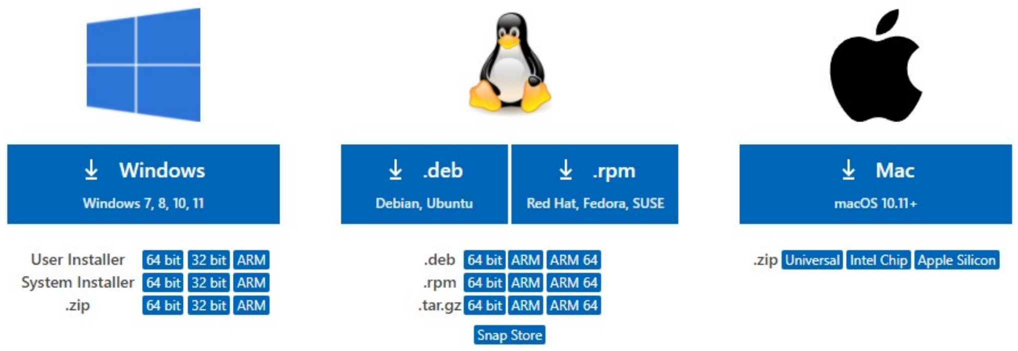


Figure 6: Visual Studio Code installer download options.

Now setup the VS Code installation with the required extensions. The official Microsoft extension should be used for C/C++. The specific extensions are circled in green in Figure 7. To install extensions, click the block stack icon on the left side toolbar (outlined in blue in Figure 7) and search for the extension name. Click the blue install button on the bottom right of the extension you wish to install. Certain extensions will take longer than others and will typically provide status updates in the bottom toolbar.

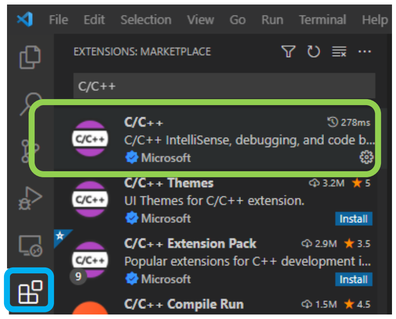


Figure 7: Required official VS Code extensions.

PlatformIO IDE is a third-party extension for VS Code that can be used to program many microcontrollers, FPGAs, and other embedded platforms. This package is shown below and once installed will add a new icon to the left side toolbar (circled in orange in Figure 8).

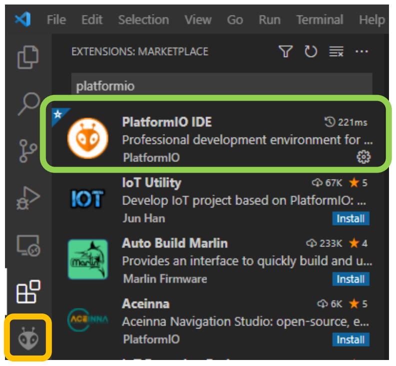
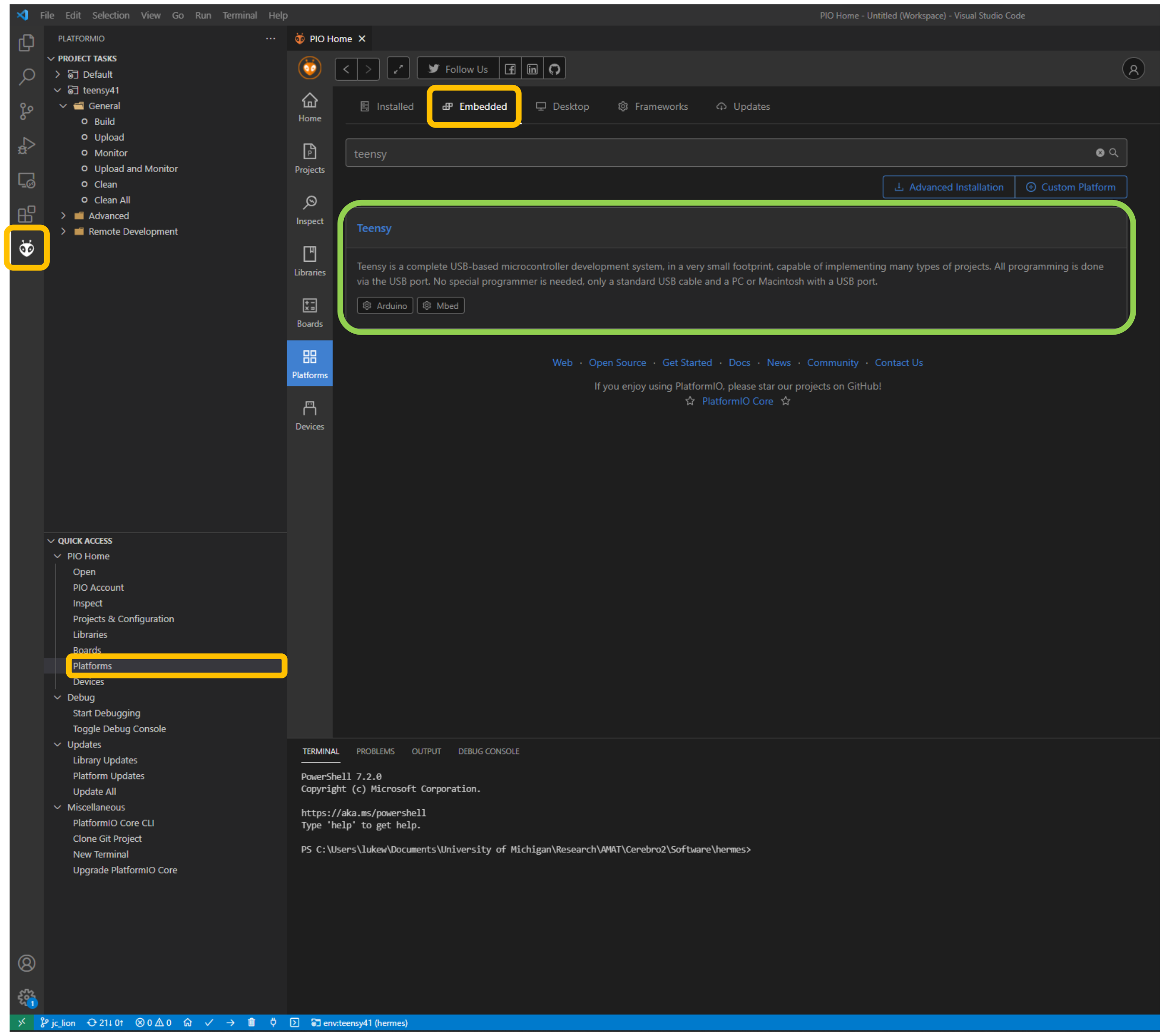


Figure 8: PlatformIO extension and toolbar icon.

Other useful extensions *vscode-icons* and GitLens. During extension installation it may be necessary for VS Code to restart. After installing the PlatformIO IDE extension the Teensy platform must be installed. One easy way to do this is to set up a new project for the Teensy 4.1 on the PlatformIO home screen and this will install all necessary prerequisites. To manually install, click on the PlatformIO icon, then Platforms under the PIO Home dropdown menu. Select embedded and search for the Teensy platform. Click on the blue Teensy link to go to the installation page and install the newest version. This will also automatically install the Arduino framework.



Teensy is a complete USB-based microcontroller development system, in a very small footprint, capable of implementing many types of projects. All programming is done via the USB port. No special programmer is needed, only a standard USB cable and a PC or Macintosh with a USB port.

If you enjoy using PlatformIO, please star our projects on GitHub!

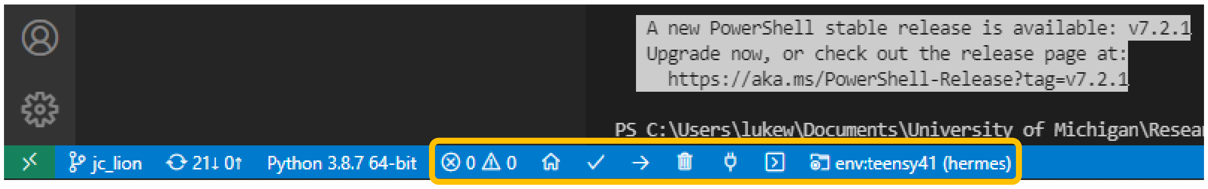
```
PowerShell 7.2.0
Copyright (c) Microsoft Corporation.

https://aka.ms/powershell
Type 'help' to get help.

PS C:\Users\lukew\Documents\University of Michigan\Research\AMAT\Cerebro2\Software\hermes>
```

Figure 9: PlatformIO platform installation page.

To open the software got to *File -> Open Folder* and select the project folder (the root folder containing *src, lib,* etc.). Once opened PlatformIO should automatically recognize the project and populate the toolbar circled in amber in Figure 10. The icons from left to right are Errors, Warnings, PIO Home, Compile and Verify, Upload, Clean, Serial Monitor, Terminal, and Platform Environment. For additional information regarding installation and setup please see the respective websites for each piece of software.



```
A new PowerShell stable release is available: v7.2.1
Upgrade now, or check out the release page at:
  https://aka.ms/PowerShell-Release?tag=v7.2.1

PS C:\Users\lukew\Documents\University of Michigan\Resea
```

Figure 10: VS Code extension toolbar.

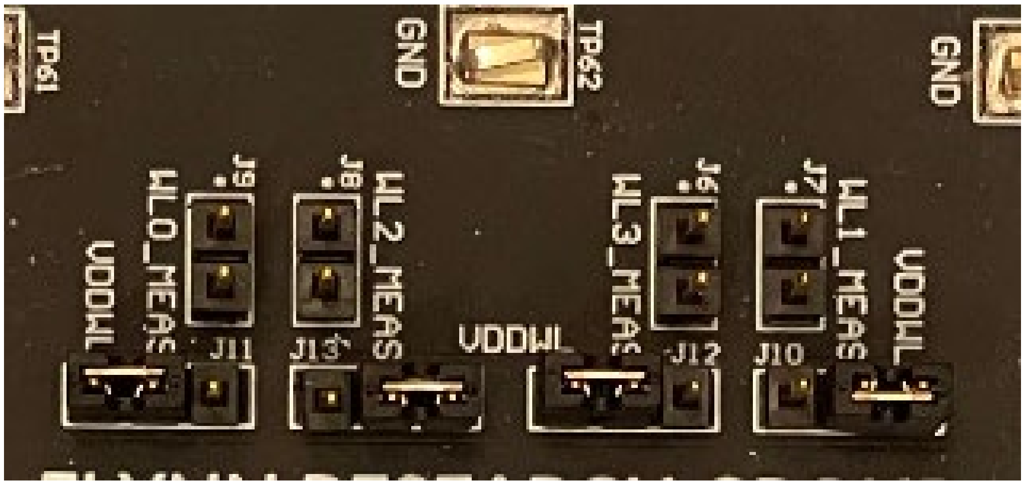
The Teensy should be flashed with the PlatformIO project, the main file of which is *main.cpp* found in the *src* folder. The code repository can be found in the shared documentation folder accessed by scanning the QR code on the motherboard. The *main.cpp* file currently contains example code. See the included libraries for various programming algorithms, bitcell functions, and MAC operation functions.

Note: On Linux Teensy framework installation require additional udev rules found here:

https://www.pjrc.com/teensy/td_download.html

Taking Measurements

To measure the current of any of the voltage supplies, disconnect the jumper and place a current meter in series between the two pin or the 2 SMD test point. When performing power measurements be sure to measure the voltage on the low side of the current meter to account for voltage drop inside the meter. The same measurement techniques can be used for the current sources, taking care to measure the current going into the chip and not into the test structure. The wordline voltage (VDDWL) headers can be used to measure the voltage on each tile's wordline supplied by the internal DAC. To do this the 3-pin jumper for the given tile must be aligned with the measurement header (away from the VDDWL label). Then in the software enable both the wordline DAC and wordline VDD registers. Figure 11 shows the typical configuration for these headers during normal operation where VDDWL is connected to each wordline.



Software Usage

The CerebroFunctions.cpp function descriptions are as follows. The low-level register functions are omitted but documented via comments in the code files.

Function	**Description**
startRead(uint32_t Tile)	Trigger the state machine for the given tile to read the column outputs.
startWrite(uint32_t tile)	Trigger the state machine for the given tile to apply write pulses based upon the preconfigured write registers.
setGateDAC(uint32_t tile, uint16_t dac_code)	Set the wordline DAC for a given tile to a given 16 bit code translating to a voltage between V_DAC_T and V_DAC_B.
setDAC(uint32_t tile, uint8_t row, uint8_t dac_code)	Set the input DAC (pulse DAC) for a given row in a given tile to an 8-bit pulse code (0-255.)
setDACs(uint32_t tile, uint32_t row, int num_rows, uint8_t buffer[])	Set the input DAC (pulse DAC) for multiple rows from the given row up to the given row plus the number of rows with the values for each row stored in the buffer.
readADC(uint32_t tile, uint8_t col)	Read the output ADC for a given column.
readADCs(uint32_t tile, uint8_t buffer[])	Read out all 32 ADCs (all even or all odd columns depending on configuration).

The CerebroConfig.cpp function descriptions are as follows.

Function	**Description**
configIdle(uint32_t Tile)	Place the given tile in the idle configuration.
configAddress(uint32_t tile, uint8_t row, uint8_t col)	Set the address registers for a specific bitcell. (Used for configSingleRead and configWrite)
configSingleRead(uint32_t tile, uint8_t row, uint8_t col)	Configure the state machine to read a single bitcell at the given row and column position in the given tile.
configWrite(uint32_t tile, uint8_t row, uint8_t col, uint8_t mode, uint16_t dac_code, uint8_t cycles)	Configure the state machine to perform a write operation (form, set, or reset) on a single bitcell at the given row and column position in the given tile.
configMAC(uint32_t tile, int mode)	Configure the state machine to perform a MAC operation on the even or odd columns.

The CerebroBitcell.cpp function descriptions are as follows.

Function	**Description**
read(uint32_t tile, uint8_t row, uint8_t col, uint8_t pulses, uint8_t offsets[])	Read the stored value of a single bitcell. This function performs all operation configurations and state machine triggers, calling configSingleRead, setDAC, startRead, configIdle, and readADC.
write(uint32_t tile, uint8_t row, uint8_t col, uint8_t mode, uint16_t dac_code, uint8_t cycles)	Perform a write operation on a single bitcell. This function performs all operation configurations and state machine triggers, calling configWrite, startWrite, and configIdle.
set(uint32_t tile, uint8_t row, uint8_t col, float vDL)	Perform a set operation on a single bitcell. This function performs all operation configurations and state machine

	triggers, calling configWrite, startWrite, and configIdle. This also configures the AD5686R to provide a V_SET voltage.
rst(uint32_t tile, uint8_t row, uint8_t col, float vBL)	Perform a reset operation on a single bitcell. This function performs all operation configurations and state machine triggers, calling configWrite, startWrite, and configIdle. This also configures the AD5686R to provide a V_RESET voltage.
write1(int write_mode, uint32_t tile, uint8_t row, uint8_t col, float vset_start, float vset_step, uint8_t set_stop, float vrst_start, float vrst_step, uint8_t rst_stop, uint8_t offsets[], int dataCount, uint8_t buffer[])	Programming loop to write a value to a cell. For additional details please see comments in Cerebrobitcell.cpp.

CerebroCrossbar.cpp contains functions to perform MAC operations across an entire tile and function to perform a matrix multiplication using MACs and stitching the even and odd results together. CerebroAlgo.cpp contains various programming algorithms for use with filamentary ReRAM cells. These programming algorithms are provided as is and may require tuning to function. CerebroArray.cpp contains function utilizing nested for loops to perform read, set, reset, and form operations on a 2D array of cells instead of just a single cell. The AD5686R.cpp file contains functions for using the AD5686R external DAC evaluation board over SPI. Please see the comments in the both the AD5686R.cpp and AD5686R.h files for instructions. A demo is also provided in the main.cpp file. Please se the code comments for further details pertaining the the usage of the functions described in the above tables and in this paragraph.