PrairieLearn Pi Pico TM4C123GH6PM Emulator

Senior Design May 2024 - 33

Mitch Hudson
Tyler Weberski
Chris Costa
Andrew Winters
Carter Murawski
Matt Graham

Table of Contents

Introduction

This document outlines how to use and configure the TM4C123GH6PM emulator created by the sdmay24-33 team. This will include the setup of the emulator from the sdmay24-33 repository, how to set up the emulator from scratch, and how to use the automated script. This will also include the next steps that may be taken to implement the emulator into PrairieLearn further, as well as a walkthrough of the repository structure. This emulator is intended to be used to compile C code specific to the microcontroller. This emulator's intended functionality is intended to be used to compare student-written code to randomly generated correct answers, as well as for generating code for PrairieLearn interactive questions.

Repository Structure

Overview

The sdmay24-33 repository contains an emulator directory. This document focuses on the emulator directory structure only. Please see other documentation for the structure of the sdmay24-33 repository in its entirety. The emulator directory is broken into the three subdirectories of pico-sdk, rp2040js, and build-hex-files.

emulator

The emulator directory contains the three subdirectories of pico-sdk, rp2040js, and build-hex-files. This directory also contains the *emulator.sh* script and the *npm_output.txt* text file. The *emulator.sh* script will be discussed in the “Using the Automated Script Configuration” section below. The *npm_output.txt* text file is used to store the outputs of the *emulator.sh* script.

pico-sdk

The pico-sdk directory holds the SDK information for the original Raspberry Pi Pico emulator. This directory is necessary for the running of the emulator.

rp2040js

The rp2040js directory holds the original JavaScript emulator of the Raspberry Pi Pico. This directory is where complied C files are run, and their results are determined.

build-hex-files

The build-hex-files directory is used as the custom build location for C files. This directory contains the *build-hex-files.c* file, which is modified to contain the C code that is to be executed on the emulator.

Using the Automated Script Configuration

Overview

This section illustrates how to use the *emulator.sh* existing script to run the emulator. This script will employ the steps seen in “Configure the Existing Repository Structure” and output to a text file. This script can be modified to meet the specific needs of its application. This script also contains comments to help explain the functionality in depth.

Functionality

The functionality of the *emulator.sh* script is as follows: replace any username occurrences with the current user account, configure path variables for the SDK, install any needed packages, build a hex file from the modified C file in build-hex-files, copy the hex file to the correct location in rp2040js, configure the environment, run the hex file on the emulator, and store the output in the *npm_output.txt* text file.

Running the Script

From your Linux environment’s home directory, run the following Git clone command:

```
$ https://git.ece.iastate.edu/sd/sdmay24-33.git
```

This will create the directory called sdmay24-33. Change into the nested emulator directory and run the script included within:

```
~$ cd sdmay24-33/emulator/
~/sdmay24-33/emulator$ ./emulator.sh
```

Running this script will output the results to the *npm_output.txt* text file:

```
~/sdmay24-33/emulator$ less npm_output.txt
```

```
Hello, World!
npm_output.txt (END)
```

Configuring the Existing Repository Structure

Overview

This section details the process of configuring the emulator to run within the sdmay24-33 repository with the preexisting modifications. This section is intended to be informative, and the ‘Using the Automated Script Configuration” section is advised to be used in its place.

Configuration

On the Linux environment, open Terminal and navigate to the user’s home directory. Then, run the following Git clone command:

```
$ https://git.ece.iastate.edu/sd/sdmay24-33.git
```

This will create the directory called sdmay24-33. Change into this directory and run the following commands to ensure your system is up to date:

```
~$ cd sdmay24-33/emulator/
~/sdmay24-33/emulator$ sudo apt update
~/sdmay24-33/emulator$ sudo apt install
```

After completing the above step, the configuration of the emulator can begin. To start, run the following installation command:

```
~/sdmay24-33/emulator$ sudo apt install cmake gcc-arm-none-eabi libnewlib-arm-none-eabi build-essential
```

This will install the proper cmake configuration for the emulator. Next, open the following file using nano:

```
~/sdmay24-33/emulator$ sudo nano ~/.bashrc
```

Once inside the file editor, add the following line to the end of the file if it does not already exist. The variable *$USER* automatically fills the current user's username. If the given line exists in the file with another username rather than the user currently logged in, it must be replaced:

```
export PICO_SDK_PATH=/home/$USER/sdmay24-33/emulator/pico-sdk
```

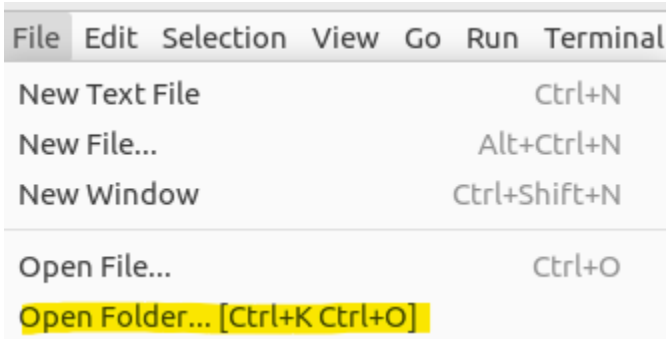
Save the file by pressing Ctrl+X, then pressing Y, then pressing Enter. Leave the Terminal window open for use later in this section.

If Visual Studio Code is not installed, visit https://code.visualstudio.com/download for the most up-to-date version.

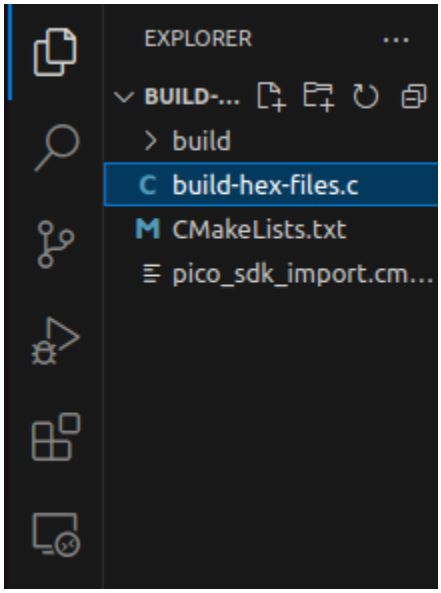
Install the following Visual Studio Code extensions:

```
~/sdmay24-33/emulator$ code --install-extension marus25.cortex-debug
~/sdmay24-33/emulator$ code --install-extension ms-vscode.cmake-tools
~/sdmay24-33/emulator$ code --install-extension ms-vscode.cpptools
```

Now Open Visual Studio Code. Inside Visual Studio Code, Select File, Open Folder, and then select "sdmay24-33/emulator/build-hex-files" in the user's home directory from the filesystem:

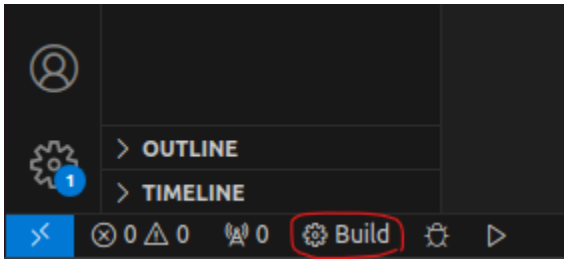


Open the *build-hex-files.c* file from the navigation pane:

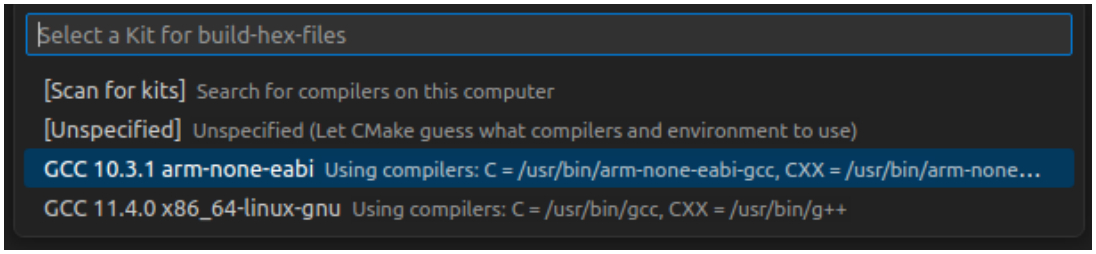


This file can be edited to include any C code that is to be run on the emulator. Note that the line “stdio_init_all();” must be the first line of “main()” for standard input and output to function. Examples of how to use UART functionality are shown in the remainer of the file. For more examples, visit https://github.com/raspberrypi/pico-examples.

After any modifications have been made and saved, select the Build icon at the bottom-left corner of the window:

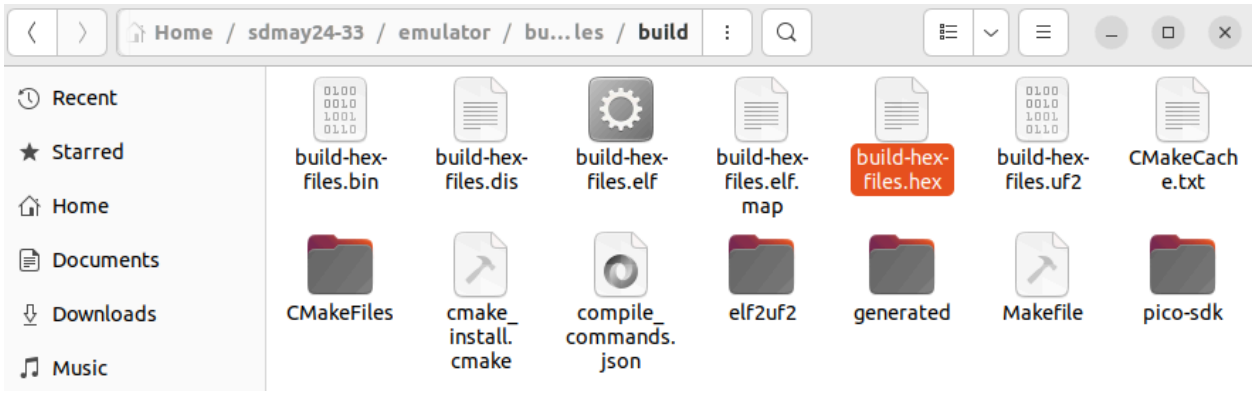


A prompt will appear at the top of the window for “Select a kit for build-hex-files.” Select “GCC 10.3.1 arm-none-eabi” from the dropdown:

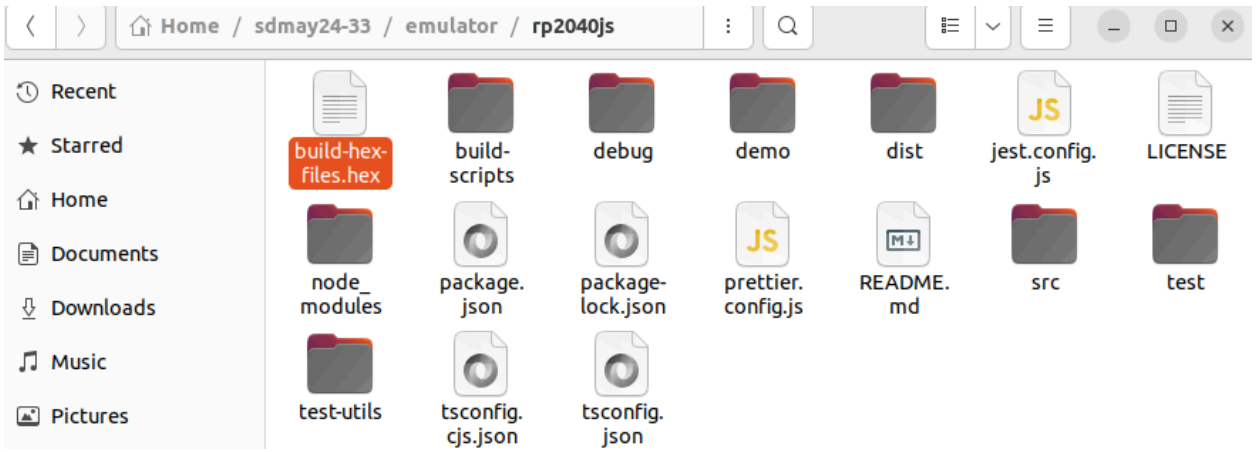


After the completion of the build in the Visual Studio Code, the Visual Studio Code window may be closed.

Open Files and navigate to “sdmay24-33/emulator/build-hex-files/build” in the user’s home directory. Copy the *build-hex-files.hex* file to the clipboard:



Next, navigate to “sdmay24-33/emulator/rp2040js” in the user’s home directory. Paste the *build-hex-files.hex* file from the clipboard here:



After the completion of the above step, the Files window may be closed.

Open the Terminal window used earlier in this section and change into the “sdmay23-33/emulator/rp2040js” directory:

```
~/sdmay24-33/emulator$ cd rp2040js/
```

From this directory, run the following installation commands. Note that NodeJS version 20 or later is required:

```
~/sdmay24-33/emulator/rp2040js$ sudo apt install nodejs
~/sdmay24-33/emulator/rp2040js$ sudo apt install npm
```

Once installed, run the following commands to initialize and start the environment:

```
~/sdmay24-33/emulator/rp2040js$ npm install
~/sdmay24-33/emulator/rp2040js$ npm start
```

After running these commands, the output of the C file edited earlier in this section will be displayed in the terminal, assuming the appropriate output statements were included. The program can be stopped by pressing Ctrl+C.

```
[redacted]:~/sdmay24-33/emulator/rp2040js$ npm start

> rp2040js@1.0.1 start
> tsx demo/emulator-run.ts

Hello, World!^C
[redacted]:~/sdmay24-33/emulator/rp2040js$
```

Configuring the Emulator from Scratch

Overview

This section details the process of configuring the emulator from scratch to run within the sdmay24-33 repository without the preexisting modifications. This section is intended to be informative, and the ‘Using the Automated Script Configuration” section is advised to be used in its place. This section uses the pico-examples repository to demonstrate the setup process and provide example code.

Configuration

On the Linux environment, open Terminal and navigate to the user's home directory. Then, run the following Git clone command:

```
$ https://git.ece.iastate.edu/sd/sdmay24-33.git
```

This will create the directory called sdmay24-33. Change into this directory and create a new directory called emulator:

```
~/sdmay24-33$ mkdir emulator
```

Change into the new emulator directory and run the following commands to ensure your system is up to date:

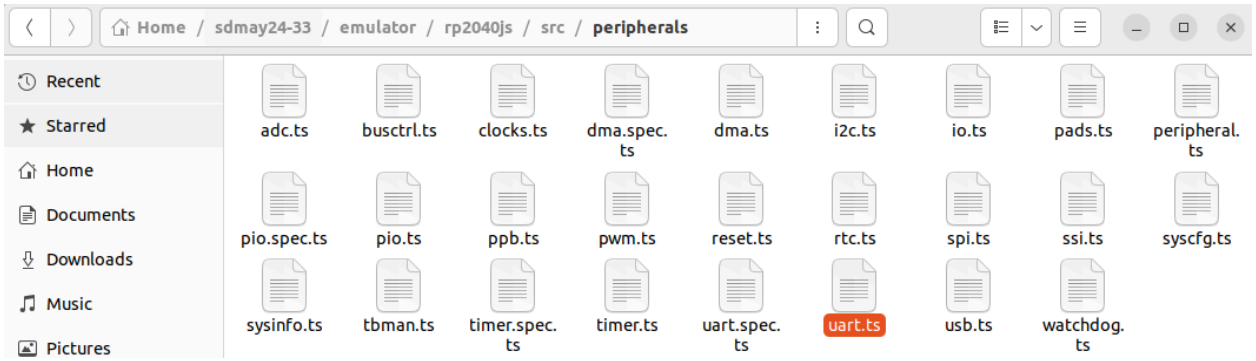
```
~$ cd sdmay24-33/emulator/
~/sdmay24-33/emulator$ sudo apt update
~/sdmay24-33/emulator$ sudo apt install
```

Next, clone the following repositories separately inside the emulator directory:

```
~/sdmay24-33/emulator$ git clone https://github.com/wokwi/rp2040js.git
~/sdmay24-33/emulator$ git clone https://github.com/raspberrypi/pico-sdk.git
~/sdmay24-33/emulator$ git clone https://github.com/raspberrypi/pico-examples.git
```

After completing this step, leave the Terminal window open for use later in this section.

Open Files and navigate to sdmay24-33/emulator/rp2040js/src/peripherals and open the *uart.ts* file. Leave the Files window open for use later in this section:



In the *uart.ts* file, make the following modifications. The comment lines represent the existing code with their modifications placed int the line below each comment:

```
10 //const UARTLCR_H = 0x2c;
11 const UARTLCRH = 0x2c;
12 //const UARTCR = 0x30;
13 const UARTCTL = 0x30;
14 //const UARTIMSC = 0x38;
15 const UARTIM = 0x38;
```

```
146       //case UARTLCR_H:
147       case UARTLCRH:
148         return this.line
149       //case UARTCR:
150       case UARTCTL:
151         return this.ctrl
152       //case UARTIMSC:
153       case UARTIM:
```

```
195       //case UARTLCR_H:
196       case UARTLCRH:
197         this.lineCtrlRegister = value;
198         break;
199
200       //case UARTCR:
201       case UARTCTL:
202         this.ctrlRegister = value;
203         if (this.enabled) {
204           this.rp2040.dma.setDREQ(this.dreq.tx);
205         } else {
206           this.rp2040.dma.clearDREQ(this.dreq.tx);
207         }
208         break;
209
210       //case UARTIMSC:
211       case UARTIM:
212         this.interruptMask = value & 0x7ff;
213         this.checkInterrupts();
214         break;
```

Open the existing Files window and open the *uart.spec.ts* file and make the following modifications:

```
 7 //const OFFSET_UARTLCR_H = 0x2c;
 8 const OFFSET_UARTLCRH = 0x2c;
 9
10 describe('UART', () => {
11   it('should correctly return wordLength based on UARTLCR_H value', () => {
12     const rp2040 = new RP2040();
13     const uart = new RPUART(rp2040, 'UART', 0, { rx: 0, tx: 0 });
14     //uart.writeUint32(OFFSET_UARTLCR_H, 0x70);
15     uart.writeUint32(OFFSET_UARTLCRH, 0x70);
```

The section “Configuring the Existing Repository Structure” shows additional modifications made that will not be required. These modifications include removing excess print lines from files in the pico-sdk and rp2040js repositories as well as checks for Git repository configurations that prevented adding the repositories to the sdmay24-33 repository with the current configuration.

The Files window may now be closed.

Open the Terminal window from earlier in this session, the configuration of the emulator can begin. To start, run the following installation command:

```
~/sdmay24-33/emulator$ sudo apt install cmake gcc-arm-none-eabi libnewlib-arm-none-eabi build-essential
```

This will install the proper cmake configuration for the emulator. Next, open the following file using nano:

```
~/sdmay24-33/emulator$ sudo nano ~/.bashrc
```

Once inside the file editor, add the following line to the end of the file if it does not already exist. The variable *$USER* automatically fills the current user’s username:

```
export PICO_SDK_PATH=/home/$USER/sdmay24-33/emulator/pico-sdk
```

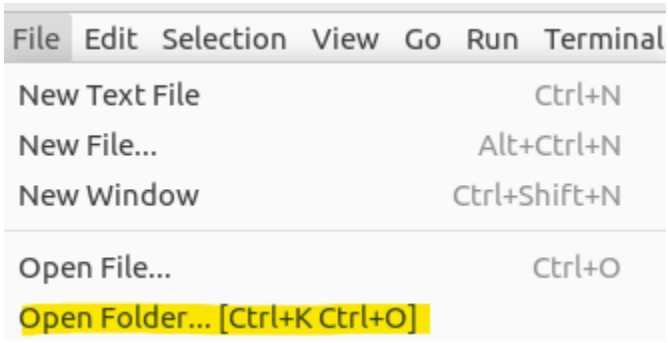
Save the file by pressing Ctrl+X, then pressing Y, then pressing Enter. Leave the Terminal window open for use later in this section.

If Visual Studio Code is not installed, visit https://code.visualstudio.com/download for the most up-to-date version.

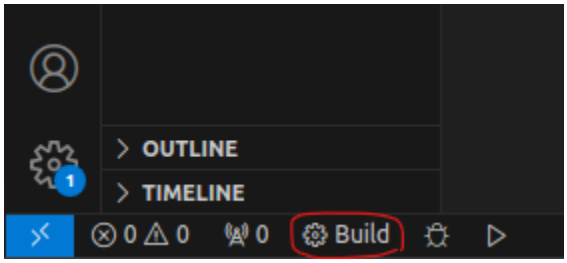
Install the following Visual Studio Code extensions:

```
~/sdmay24-33/emulator$ code --install-extension marus25.cortex-debug
~/sdmay24-33/emulator$ code --install-extension ms-vscode.cmake-tools
~/sdmay24-33/emulator$ code --install-extension ms-vscode.cpptools
```

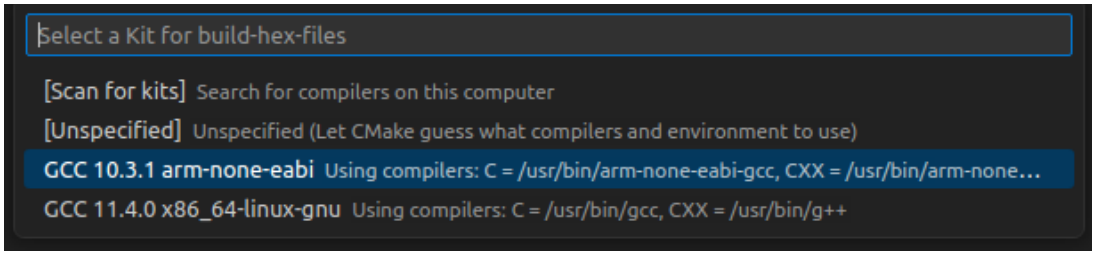
Now Open Visual Studio Code. Inside Visual Studio Code, Select File, Open Folder, and then select “sdmay24-33/emulator/pico-examples” in the user’s home directory from the filesystem:



After opening the directory, select the Build icon at the bottom-left corner of the window:



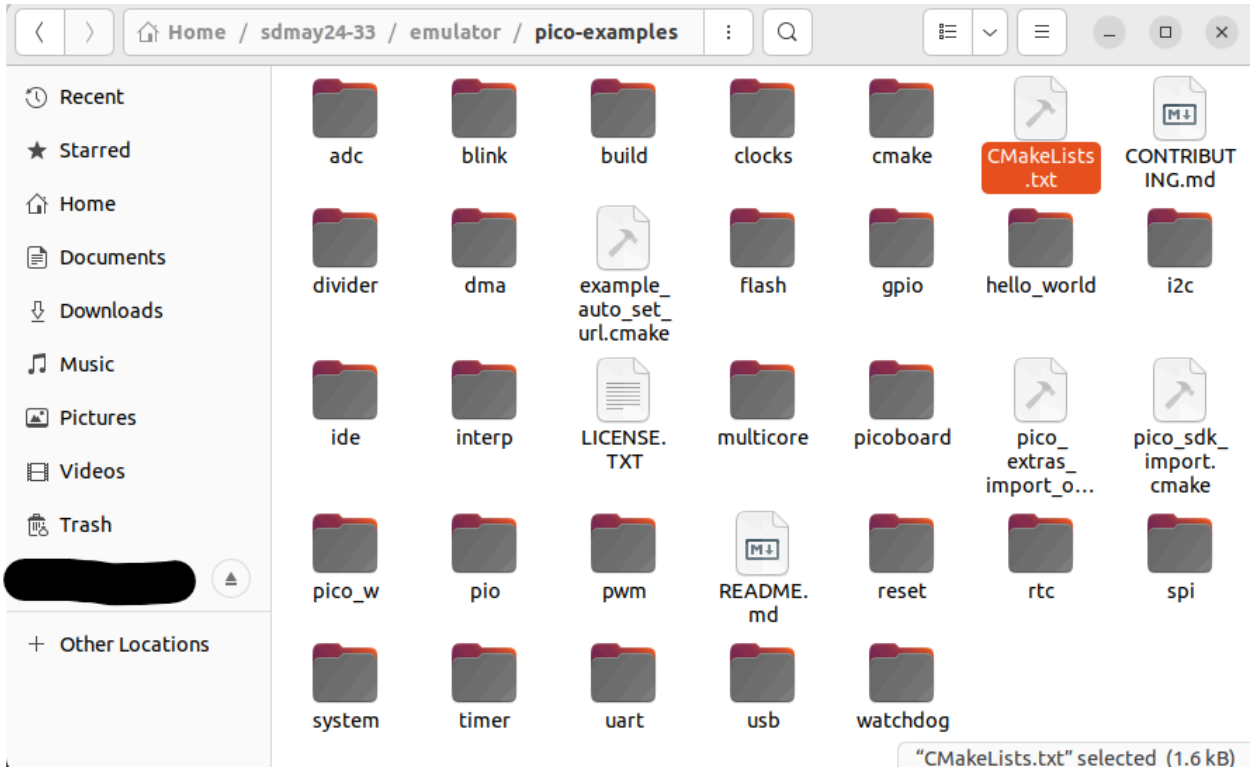
A prompt will appear at the top of the window for “Select a kit for build-hex-files.” Select “GCC 10.3.1 arm-none-eabi” from the dropdown:



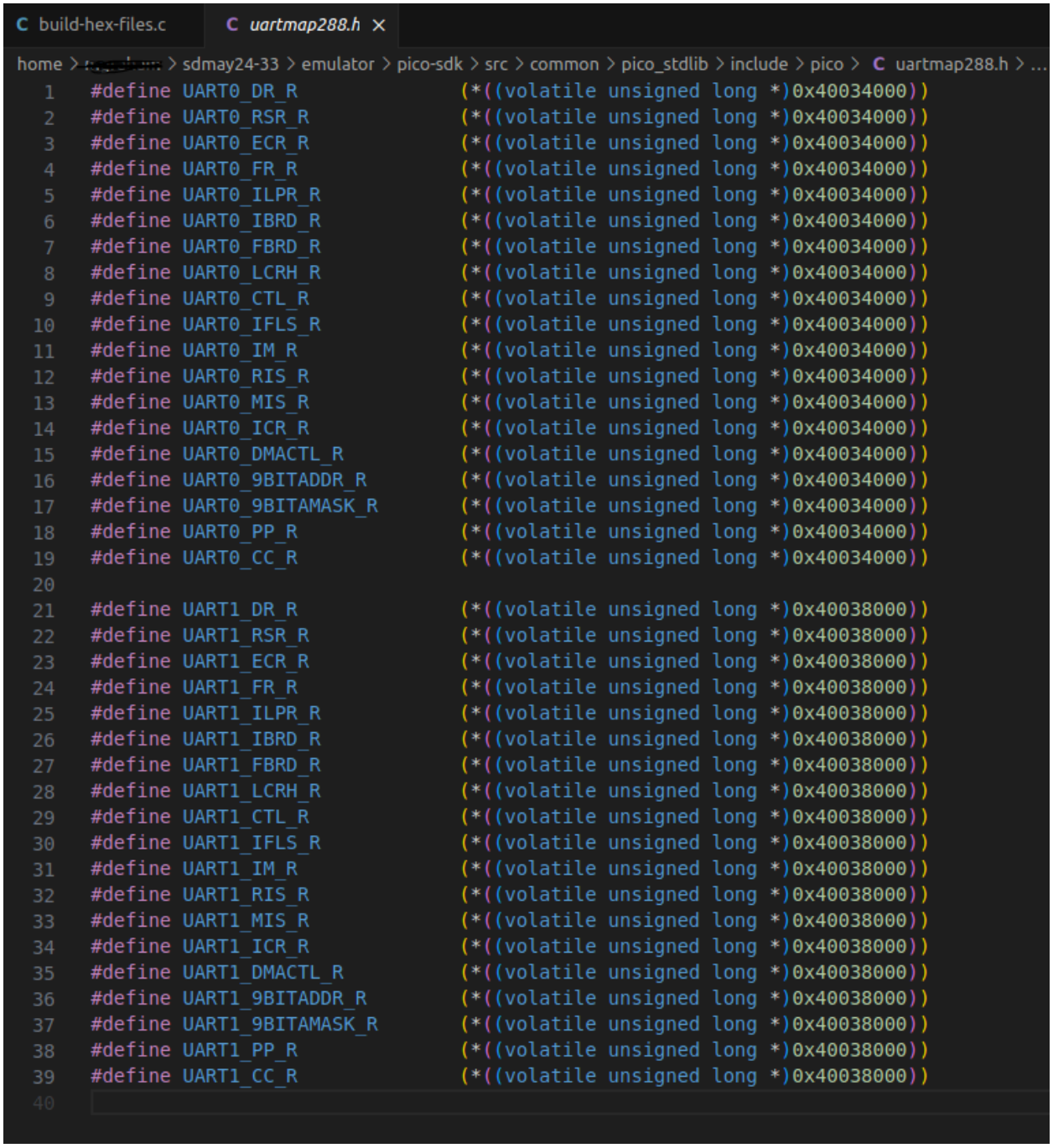
After the completion of the build in the Visual Studio Code, open the Terminal window from earlier in this section and create the following directory:

```
~/sdmay24-33/emulator$ mkdir build-hex-files
```

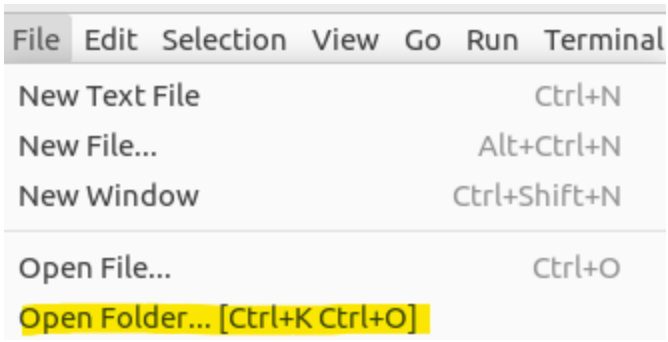
Open Files and navigate to “sdmay24-33/emulator/pico-examples” in the user’s home directory. Copy the *CMakeLists.txt* file and paste it into the “sdmay24-33/emulator/build-hex-files” directory:



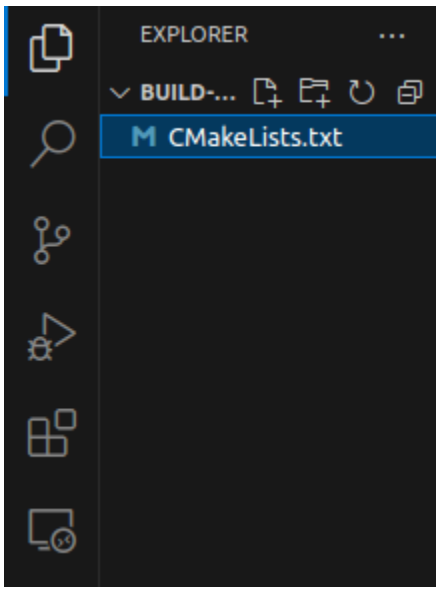
Next, navigate to “sdmay24-33/emulator/pico-sdk/src/common/pico_stdlib/include/pico” and create the file *uartmap288.h*. Paste the following code into the file:



Now Open Visual Studio Code. Inside Visual Studio Code, Select File, Open Folder, and then select “sdmay24-33/emulator/build-hex files” in the user’s home directory from the filesystem:



Open the *CMakeLists.txt* file from the navigation pane:



Modify the *CMakeLists.txt* file to contain only the following lines:

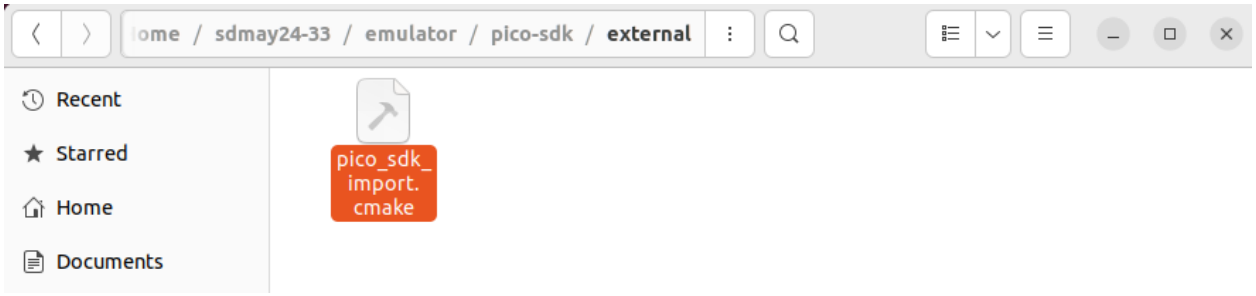
CMakeLists.txt

```
cmake_minimum_required(VERSION 3.12)

# Pull in SDK (must be before project)
include(pico_sdk_import.cmake)

project(build-hex-files C CXX ASM)
set(CMAKE_C_STANDARD 11)
set(CMAKE_CXX_STANDARD 17)

if (PICO_SDK_VERSION_STRING VERSION_LESS "1.3.0")
    message(FATAL_ERROR "Raspberry Pi Pico SDK version 1.3.0 (or later) required. Your version is ${PICO_SDK_VERSION_STRING}")
endif()

# Initialize the SDK
pico_sdk_init()

add_executable(build-hex-files
        build-hex-files.c
        )

# pull in common dependencies
target_link_libraries(build-hex-files pico_stdlib)

# create map/bin/hex/uf2 file etc.
pico_add_extra_outputs(build-hex-files)

```

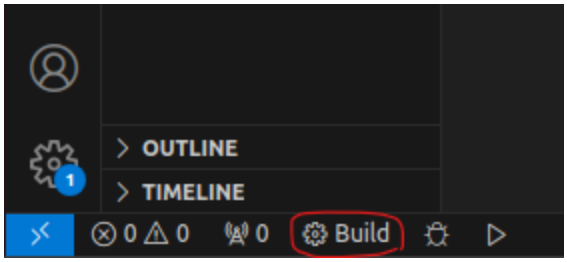
Create a new file called *build-hex-files.c* and modify it only to contain the following lines:

```c
#include <stdio.h>
#include "pico/stdlib.h"
#include "hardware/uart.h"
#include "pico/uartmap288.h"

int main() {

    UART0_DR_R = 'H';
    UART0_DR_R = 'e';
    UART0_DR_R = 'l';
    UART0_DR_R = 'l';
    UART0_DR_R = 'o';
    UART0_DR_R = ',';
    UART0_DR_R = ' ';
    UART0_DR_R = 'W';
    UART0_DR_R = 'o';
    UART0_DR_R = 'r';
    UART0_DR_R = 'l';
    UART0_DR_R = 'd';
    UART0_DR_R = '!';
}
```

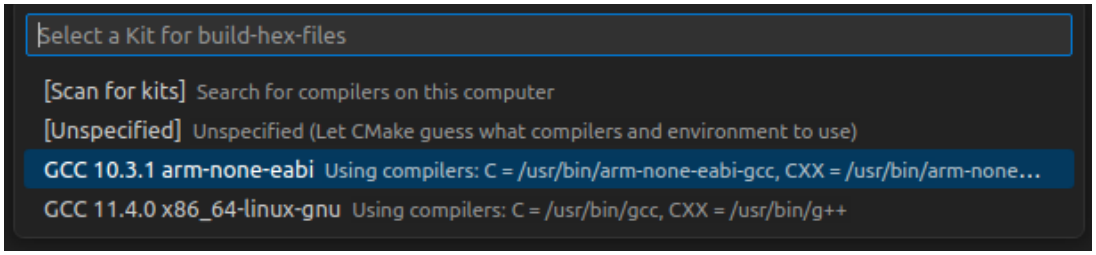
Open Files and navigate to “sdmay24-33/emulator/pico-sdk/external” in the user’s home directory. Copy the *pico_sdk_import.cmake* file and paste it into the “sdmay24-33/emulator/build-hex-files” directory:



Open the existing Visual Studio Code Window and select the Build icon at the bottom-left corner of the window:

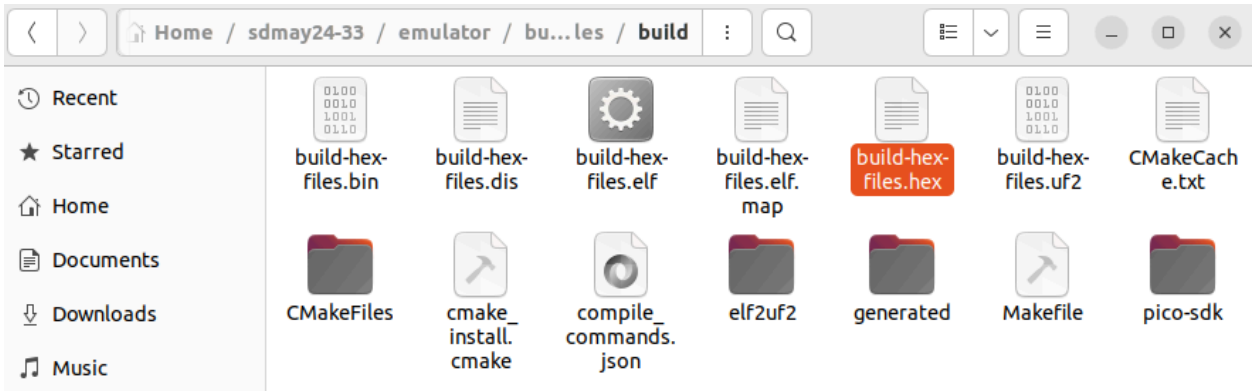


A prompt will appear at the top of the window for “Select a kit for build-hex-files.” Select “GCC 10.3.1 arm-none-eabi” from the dropdown:

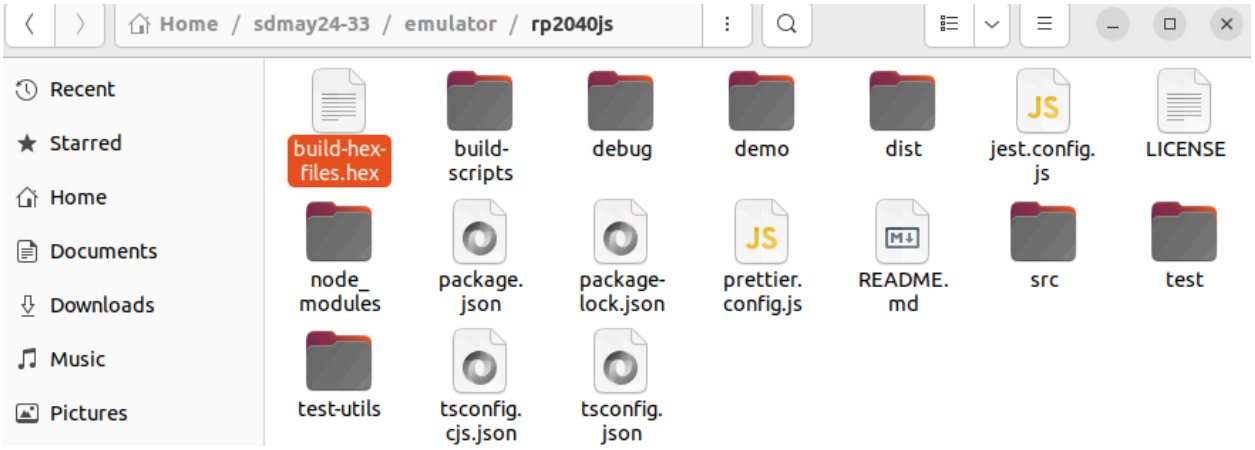


After the completion of the build in the Visual Studio Code, the Visual Studio Code window may be closed.

Open Files and navigate to “sdmay24-33/emulator/build-hex-files/build” in the user’s home directory. Copy the *build-hex-files.hex* file to the clipboard:



Next, navigate to “sdmay24-33/emulator/rp2040js” in the user’s home directory. Paste the *build-hex-files.hex* file from the clipboard here:



After the completion of the above step, the Files window may be closed.

Open the Terminal window used earlier in this section and change into the “sdmay23-33/emulator/rp2040js” directory:

```
~/sdmay24-33/emulator$ cd rp2040js/
```

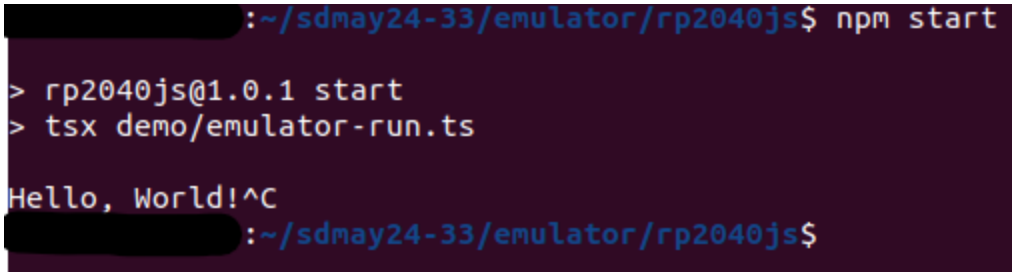
From this directory, run the following installation commands. Note that NodeJS version 20 or later is required:

```
~/sdmay24-33/emulator/rp2040js$ sudo apt install nodejs
~/sdmay24-33/emulator/rp2040js$ sudo apt install npm
```

Once installed, run the following commands to initialize and start the environment:

```
~/sdmay24-33/emulator/rp2040js$ npm install
~/sdmay24-33/emulator/rp2040js$ npm start
```

After running these commands, the output of the C file edited earlier in this section will be displayed in the terminal, assuming the appropriate output statements were included. The program can be stopped by pressing Ctrl+C.



Next Steps for Implementation

Overview

This section describes the recommended next steps to implement the emulator. Specifically, this section refers to the further integration of the emulator into the PrairieLearn environment and its interactive questions. The goal of this emulator is to provide a platform to run microcontroller-specific code. Additionally, the emulator aims to compare the output of preset or randomized microcontroller-specific code to the output given by a student in an interactive PrairieLearn question.

Next Steps

The recommended next steps for implementation are intended to integrate the emulator into the PrairieLearn environment. It is recommended that the emulator has an automated functionality to pass C code generated by an interactive question or a user to the *build-hex-files.c* file and that the outputs be compared and evaluated using the existing functionalities of PrairieLearn. It is also recommended to store the outputs in separate files, using command line arguments to specify this to the script for easier comparison. The emulator should also have its outputs evaluated against the expected outputs of the original TM4C123GH6PM microcontroller, as not all functionality was implemented on this emulator. The base addresses for UART found in the file pico-sdk/src/rp2040/hardware_regs/include/hardware/regs/*addressmap.h* have not been changed to the correct values. When these values were changed, it caused issues and the team was unable to solve these issues before the end of their tenure on the project. Instead, the file *uartmap288.h* was created, as seen earlier, and the values were replaced with the RP2040 values. This should be done for GPIO and all other desired functions of the microcontroller. The skeleton for this code can be found at https://class.ece.iastate.edu/cpre288/resources/docs/REF_tm4c123gh6pm.h. The addresses given should be replaced with the corresponding base addresses.

Resources

General Resources

Pi Pico Emulator: https://docs.wokwi.com/parts/wokwi-pi-pico
Pi Pico SDK Custom Setup:
https://www.youtube.com/watch?v=TutPFyyT3P8&ab_channel=pi3g

Repositories

RP2040JS: https://github.com/wokwi/rp2040js
Pi Pico SDK: https://github.com/raspberrypi/pico-sdk
Pi Pico Examples: https://github.com/raspberrypi/pico-examples

Data Sheets

RP2040: https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf
TM4C123GH6PM:
https://class.ece.iastate.edu/cpre288/resources/docs/Tiva_TM4C123GH6PM_datasheet.pdf