RGBbaby – manual

By Emiel Visser and Joris Bol

HAN RGBbaby

DOCUMENT ORGANISATIONAL DETAILS:

Document type:	
Skills consultant:	Paul van Wegen
Project client:	Johan Korten
Education:	Embedded Systems Engineering
Institute:	School of Engineering and Automotive, Hogeschool Arnhem-Nijmegen
Report date:	9-1-2024
Report version:	v1.0
Project duration:	Semester 3
Education term:	
Students:	2111740 (Emiel Visser), 2108100 (Joris Bol)

Version management

Version	Date	Change	Name
V1.0	11-9-2023	Creation document, preface, C1, C2 & C3.	J. Bol
V1.0	12-11-2023	Added programming environment tab.	E. Visser
V1.0	20-11-2023	Added spectrometer tab.	J. Bol
V1.0	04-01-2024	Added more information in the hardware section.	E. Visser
V1.0	05-01-2024	Added more information in the hardware section.	E. Visser
V1.0	06-01-2024	Added more information in the hardware section.	E. Visser
V1.0	07-01-2024	Added more info in the hardware and debugging section.	E. Visser
V1.0	09-01-2024	Added bibliography, performed spellcheck.	E. Visser

Table of contents

Preface

This document is meant for the successors of this project. In this document, you will find helpful installation guides and other useful instructions. Chapter 1, 2 and 3 are written in tutorial style, containing a step-by-step process of how you can set up your working environment and how to work with the PCBs we made. The following chapter explains our way of working and how you can build upon it. Chapter 5 contains an explanation of our workflow and were and what we documented. This document is strictly meant to help guide you in the setup of your work environment and not as a project guide.

Project summary

RGBbaby is a project that came in to being because of an S3 project. The Wilhelmina kinderziekenhuis (WKZ) is the client and Johan Korten the product owner. J. Korten is developing a basic life support doll for the hospital based on an already existing CPR baby doll. This project together with other projects inside the Health Concept Lab inside the HAN are meant to expand upon the doll and make it an advanced life support doll.

Our project is specifically focusing on the colour changing abilities of the skin. This meant that we had to develop an actuator and a new skin for the baby. There already was some research done on this topic by other students regarding the skin of the baby, sadly we found this research to be insufficient, so we redid most of it ourselves.

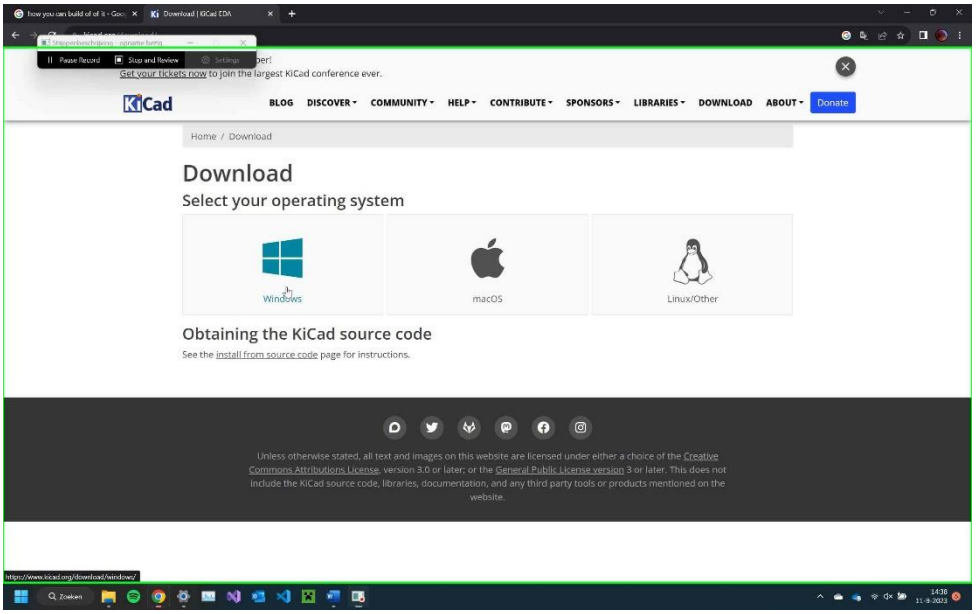
There was no previous research done on the led topic, so this had to be done from the start. Due to research done by another student, there was no certainty on the final form factor of the product, so we had to stay as modular as possible. This has another benefit, being that the WKZ is constantly adding ideas and requirements. This also means that the shape and behaviour of the baby is constantly changing.

The changing of the skin colour is an important aspect of the CPR doll. The skin colour is an important clue for the wellbeing of a baby. If the skin turn slightly blue the nurse must be on high alert because this can be an indication of a lack of oxygen.

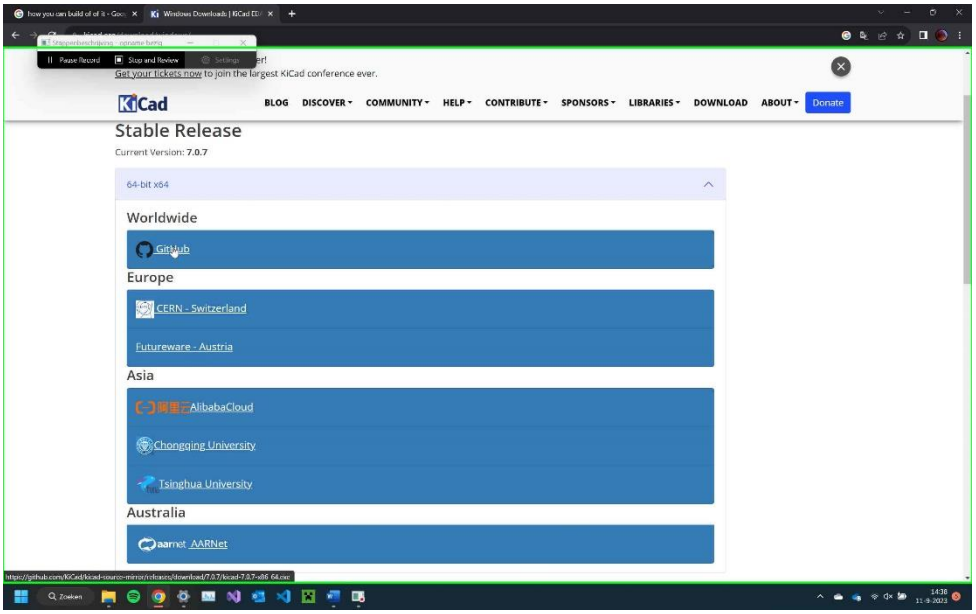
1 – KiCad setup

This chapter is an installation guide for KiCad. If you follow these steps, by the end you should be able to open our KiCad files and edit them.

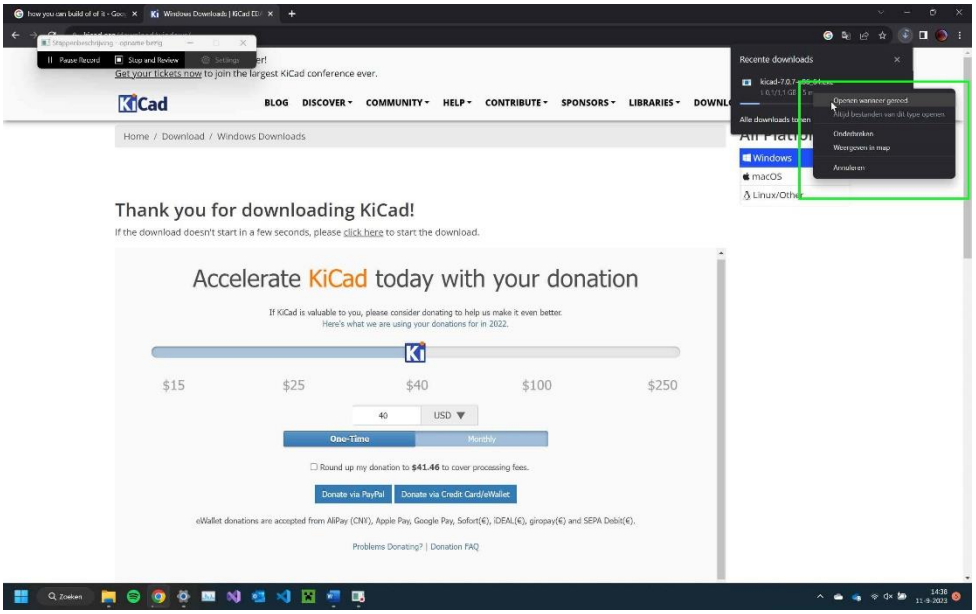
1. Go to: https://www.kicad.org/download/ (Charras, 2023) and select your operating system.



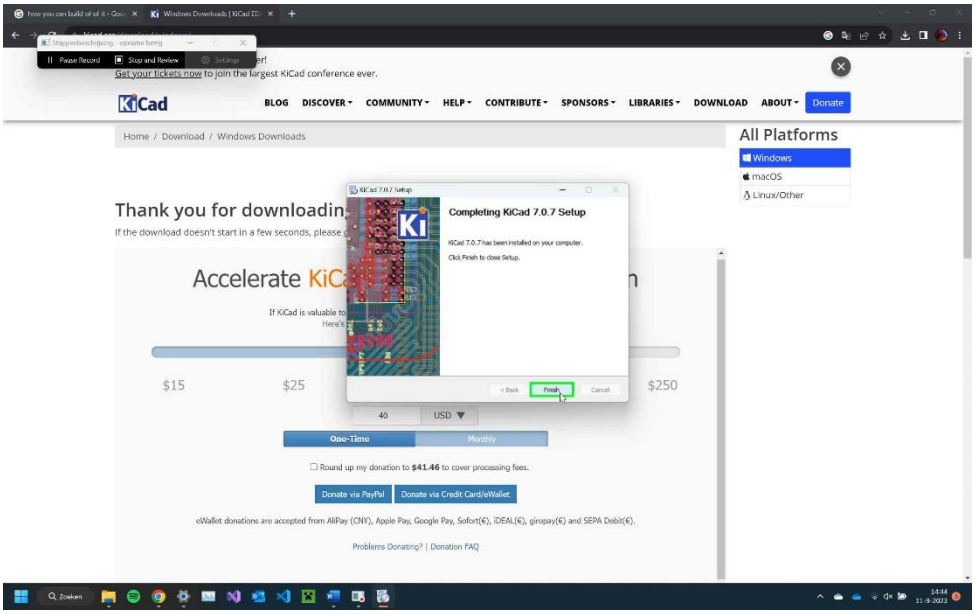
2. Select GitHub.



3. Click on the download button and select open when ready, this might take a while.



4. When downloaded, go through the installer instructions. You don't have to do anything.



5. Congratulations, you now have access to our design files.

2 – Programming environment

For writing the software we can use different programming environments. In this paragraph you will find different integrated development environments (IDE) and programming methods.

2.1 – Visual studio code

For the setup of Visual studio code we kindly refer you to this link: https://robotpatient.github.io/Manikin_Software/General/flash_build_instructions/ (Hogeweij, Flash build instructions, 2023) a detailed description is already published here.

2.2 – Clion

Clion is a modern IDE made by JetBrains. You might know them from one of their other programming IDE's or tools like: IntelliJ, PyCharm, PhpStorm or DataGrip.

In this paragraph we will show you how you can get Clion installed. How to download the required compiler and how you can start your first project.

2.2.1 – Get a JetBrains licence

If you don't have one already go to: https://www.jetbrains.com/shop/eform/students (JetBrains, 2023) Enter the required information. Important: Use your HAN email to apply for a licence.

Follow the instructions given by JetBrains.

2.2.2 – Install Clion

After you have received your licence you now have access to a number of JetBrains products. We will only be using Clion though so head over to the JetBrains website (JetBrains, 2023):
And download Clion. Clion is available for Windows, Linux and MacOS. Run the installer as you would with any regular installer. Follow the steps in the installer.

2.2.3 – Install Arm GNU toolchain

To be able to compile and build your code you will need a platform specific toolchain. Since our microcontrollers have an ARM architecture we will need to install the ARM toolchain. Head over to the following link: https://developer.arm.com/downloads/-/arm-gnu-toolchain-downloads (ARM, 2023)

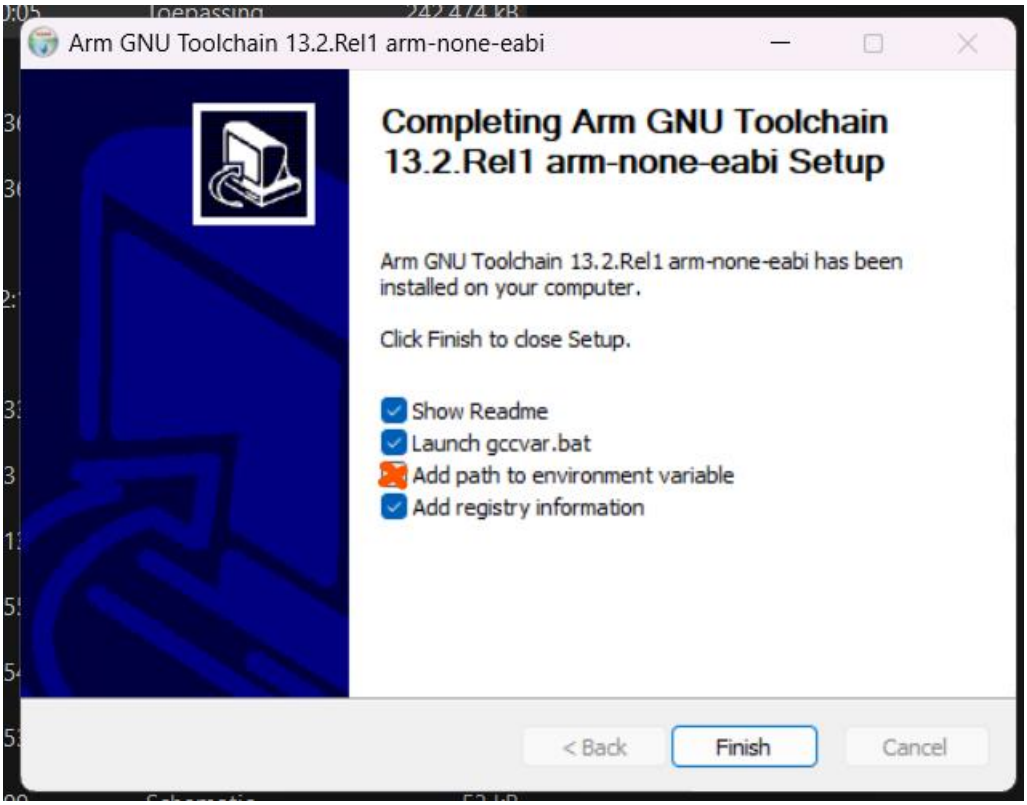
We will need the *AArch32 bare-metal target (arm-none-eabi)* version.

For windows: you will need the *arm-gnu-toolchain-13.2.rel1-mingw-w64-i686-arm-none-eabi.exe* file.

For Linux: you will need the *arm-gnu-toolchain-13.2.rel1-x86_64-arm-none-eabi.tar.xz* file.

For MacOS: you will need the *arm-gnu-toolchain-13.2.rel1-darwin-arm64-arm-none-eabi.tar.xz* file.

Run the installer. Important: make sure to check the *add to path* checkbox!



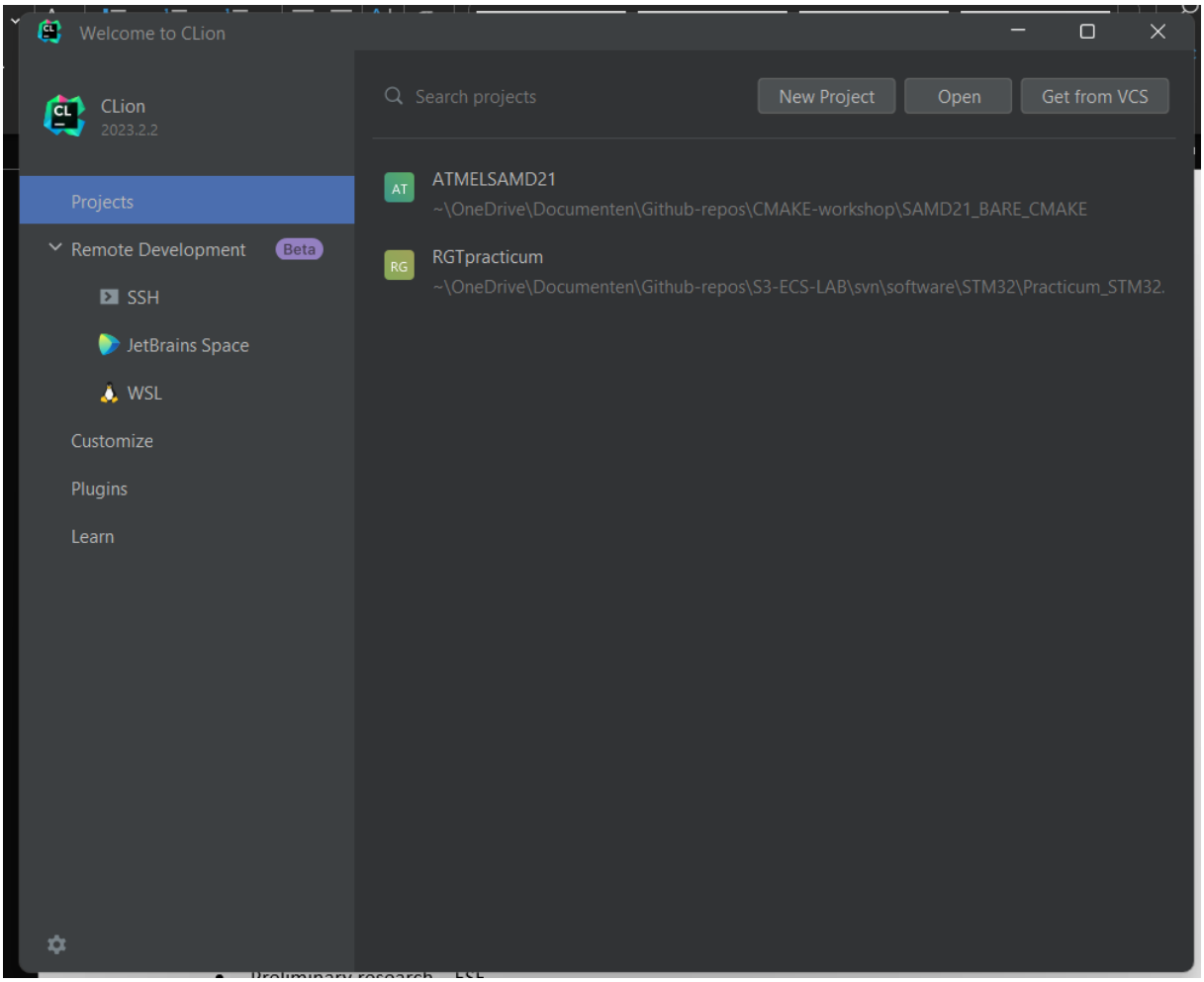
2.2.4 – Cloning git repository

To make it easy for you Victor Hogeweij has created a base project for us. This repository can be cloned here: https://github.com/Hoog-V/SAMD21_BARE_CMAKE (Hogeweij, samd21_bare_cmake, 2023)

If you don't know how to clone a repository you can follow this guide: https://docs.github.com/en/repositories/creating-and-managing-repositories/cloning-a-repository (GitHub, 2023)

2.2.5 – Setting up IDE

Now we can finally startup Clion. When you first start up Clion you will see a window like this:



Go to: *Open -> <browse to location of git repository> -> browse to 'src' -> browse to 'CMakeList.txt' ->* click *'OK'*.

The project is now opened and will be opened when you start up Clion the next time. However we are not done yet.

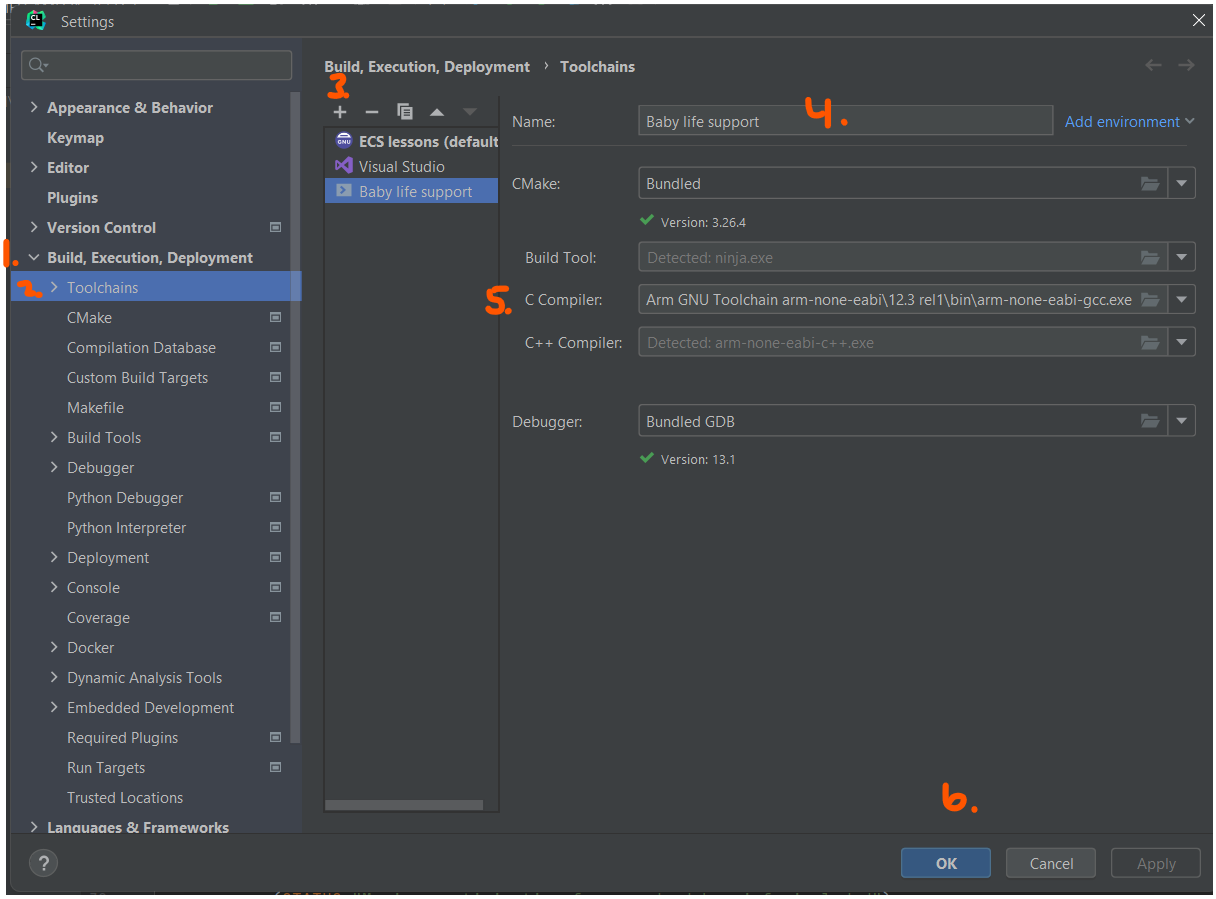
2.2.6 – Setting up toolchain

We have opened a project but we cannot compile or build anything yet because we haven't setup the compiler and debugger.

To do this, go to *file -> settings -> build, execution, deployment -> toolchains*

And add a new toolchain by pressing the +. Give your toolchain a name. After you've done that you must select the path to the GCC compiler we have just installed.

On windows you can find the installer at *C:\Program Files (x86)\Arm GNU Toolchain arm-none-eabi\12.3 rel1\bin\arm-none-eabi-gcc.exe*. Press *'OK'* if you have done this.



2.2.7 – Configuring CMake

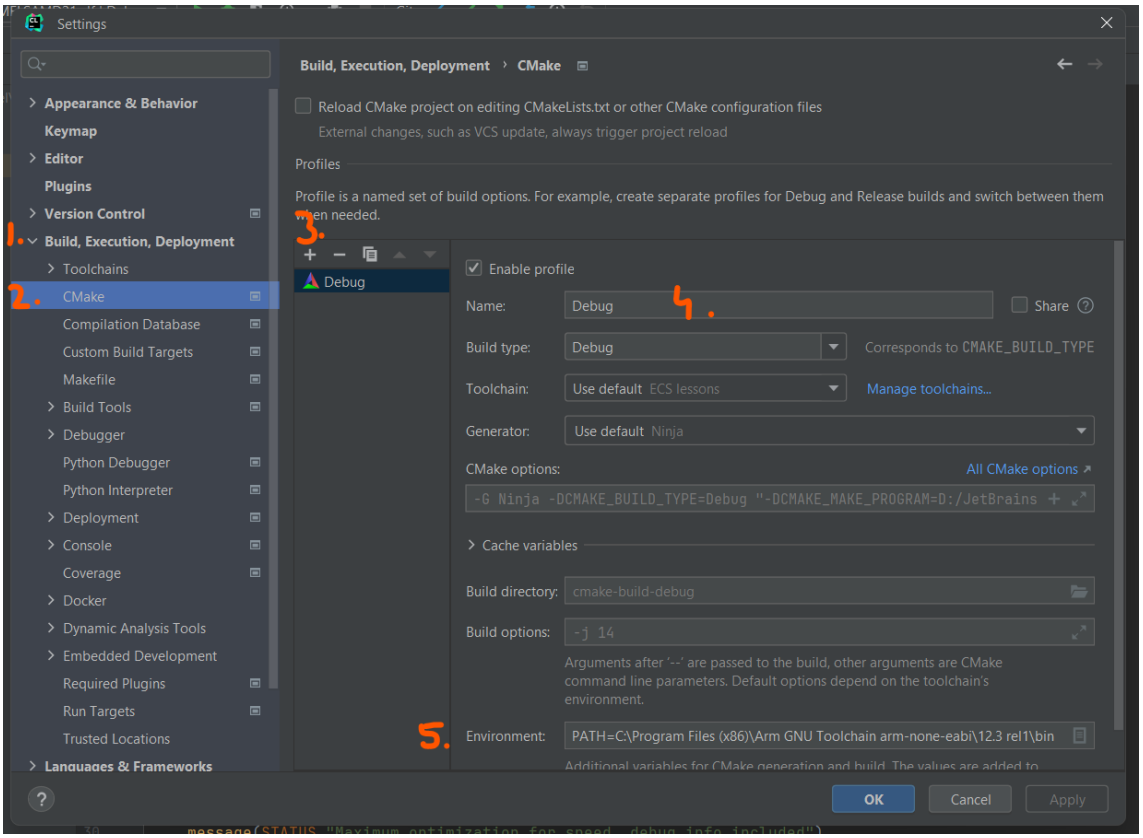
To configure CMake go to *file -> settings -> build, execution, deployment -> CMake*

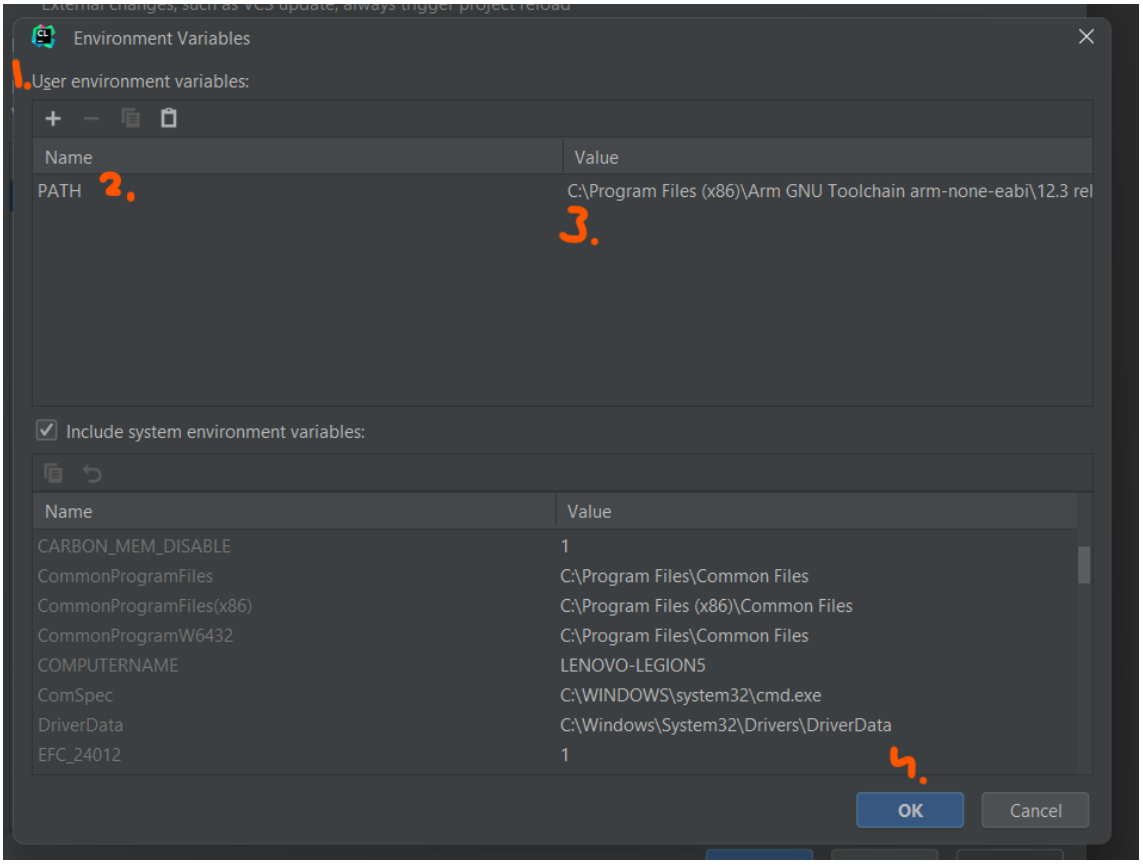
Add a new CMake configuration by pressing the + symbol.
Give your configuration a name.
Press *'OK'*.

Test by pressing Ctrl + F9. If you get errors about a path not being found you might want to try this next step:

Go back to the menu from before. Press the edit button in the *Environment* tab. A new menu will open. Press the + button once more and type PATH in the *Name* column. Add the path to the GCC compiler (see 2.1.6) in the Variable column. Then press OK.

Test by pressing Ctrl + F9. If there are still errors check if you've followed all the aforementioned steps correctly. If you still encounter problems don't be afraid to ask for help!


Settings
Build, Execution, Deployment › CMake
Appearance & Behavior
Keymap
Editor
Plugins
Version Control
Build, Execution, Deployment
Toolchains
CMake
Compilation Database
Custom Build Targets
Makefile
Build Tools
Debugger
Python Debugger
Python Interpreter
Deployment
Console
Coverage
Docker
Dynamic Analysis Tools
Embedded Development
Required Plugins
Run Targets
Trusted Locations
Languages & Frameworks
Reload CMake project on editing CMakeLists.txt or other CMake configuration files
External changes, such as VCS update, always trigger project reload
Profiles
Profile is a named set of build options. For example, create separate profiles for Debug and Release builds and switch between them when needed.
Debug
Enable profile
Name: Debug
Share
Build type: Debug
Corresponds to CMAKE_BUILD_TYPE
Toolchain: Use default ECS lessons
Manage toolchains...
Generator: Use default Ninja
CMake options:
All CMake options
-G Ninja -DCMAKE_BUILD_TYPE=Debug "-DCMAKE_MAKE_PROGRAM=D:/JetBrains
Cache variables
Build directory: cmake-build-debug
Build options: -j 14
Arguments after '--' are passed to the build, other arguments are CMake command line parameters. Default options depend on the toolchain's environment.
Environment: PATH=C:\Program Files (x86)\Arm GNU Toolchain arm-none-eabi\12.3 rel1\bin
OK
Cancel
Apply
1.
2.
3.
4.
5.


Environment Variables
User environment variables:
Name
Value
PATH
C:\Program Files (x86)\Arm GNU Toolchain arm-none-eabi\12.3 rel
Include system environment variables:
Name
Value
CARBON_MEM_DISABLE 1
CommonProgramFiles C:\Program Files\Common Files
CommonProgramFiles(x86) C:\Program Files (x86)\Common Files
CommonProgramW6432 C:\Program Files\Common Files
COMPUTERNAME LENOVO-LEGION5
ComSpec C:\WINDOWS\system32\cmd.exe
DriverData C:\Windows\System32\Drivers\DriverData
EFC_24012 1
OK
Cancel
1.
2.
3.
4.

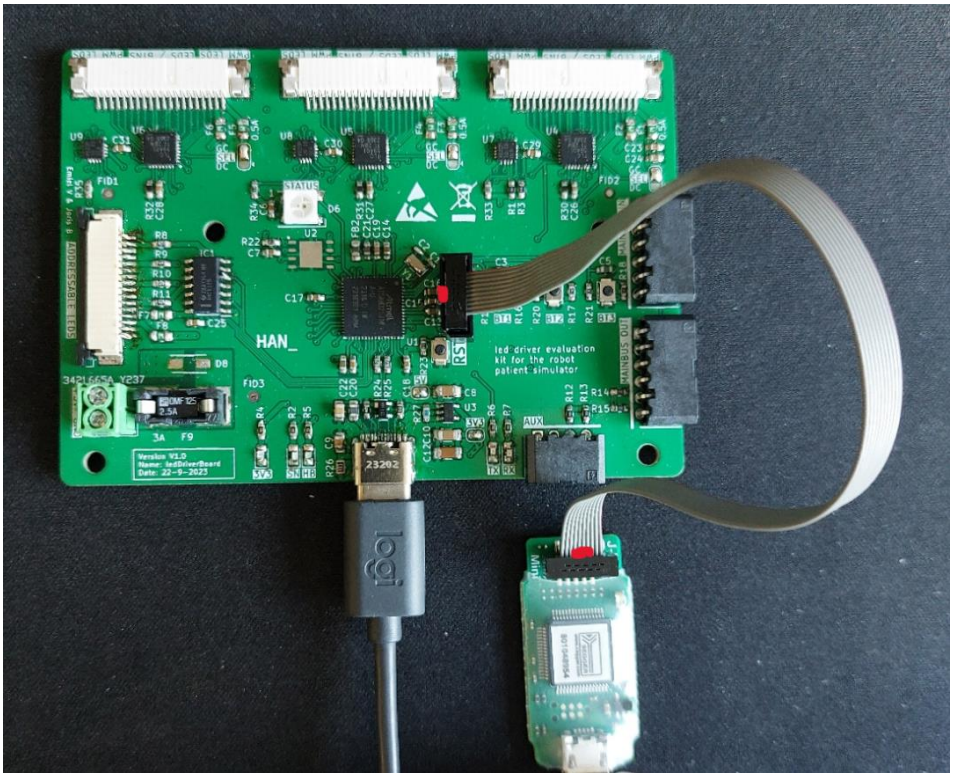
2.3 – Flashing and debugging

Now that we have chosen an environment where we can write and compile our software in we still need to get that program onto our microcontroller board. You can do this via a couple of different ways. Here we'll show you two ways to do this.

2.3.1 – ledDriverBoardV2:

Whether you want to install an Arduino bootloader or want to program the microcontroller using embedded C/C++ you will have to use the following method:

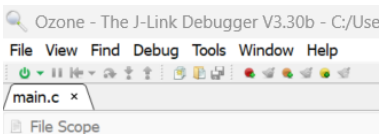
Plug a USB-C cable into the board. Connect the other end of the USB-C cable to a power source like a laptop. After you have done that you can connect the J-LINK EDU mini (Segger, 2024) debugger to the board. The connector has a notch. The red markings in the picture below indicate where the notch should go.



2.3.1.1 – Programming and debugging using Ozone:

You are now ready to upload code to the board via Clion or Visual Studio code using a program called Ozone. In Ozone the program can be started, stopped, halted and debugged.

First you will need a *.jdebug* project. You will find one in the code folders.



The *"ON"* symbol lets you connect (and flash) to a target. After that you can press the symbol next to it to start the program. In the program you will be able to step through code, see register values and set register values. Super handy.

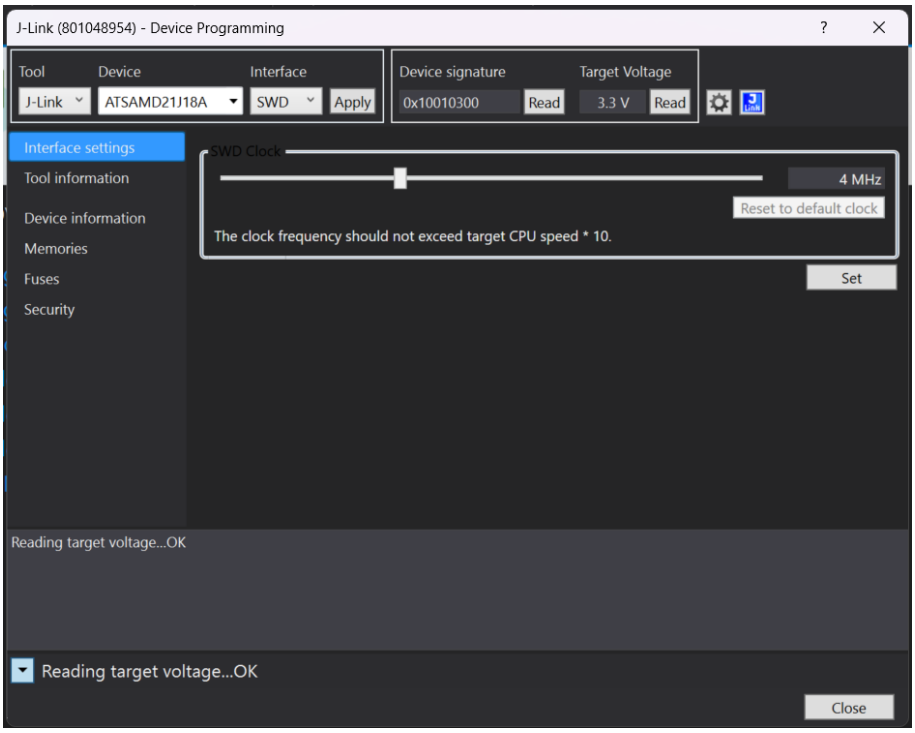
2.3.1.2 – Uploading Arduino bootloader:

To upload the bootloader to the board we will first need to download the bootloader (AtmelUniversityFrance, 2023). After you have done that you can open *Microchip Studio -> Goto Device Programming (Ctrl + Shift + P)*.

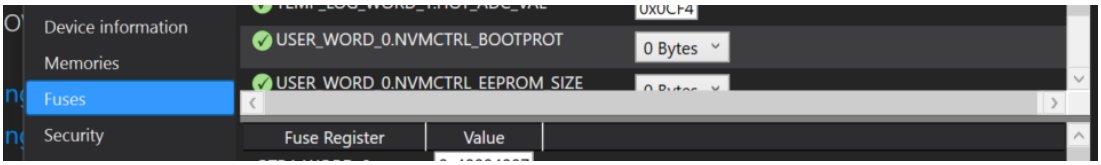
A new window will open:

Tool: J-Link
Device: ATSAMD21J18A
Interface: SWD

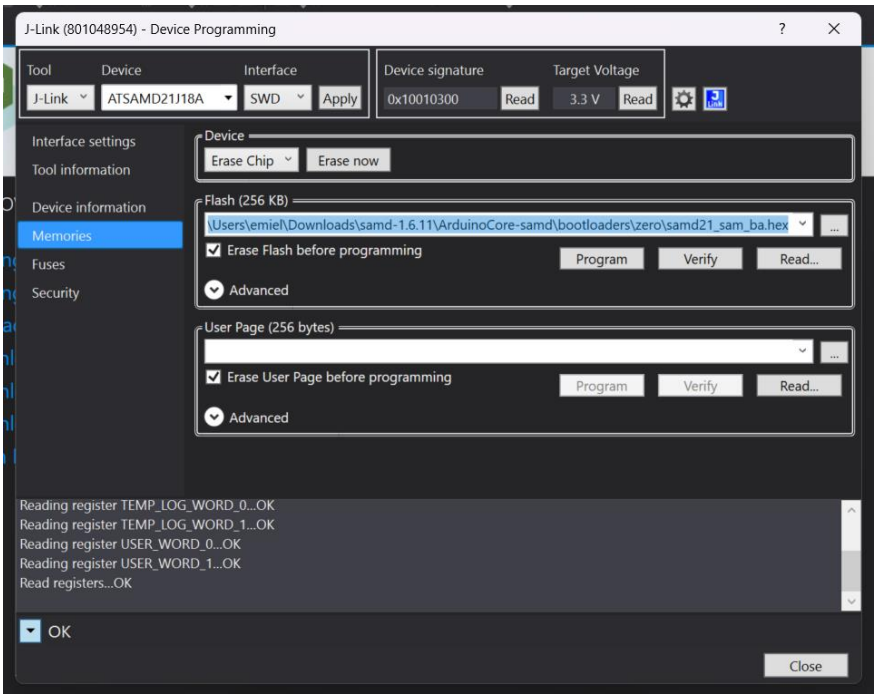
Hit apply. Next to that section you will find the device signature and target voltage buttons. Read both of those.



Go to the fuses tab and scroll down to the *USER_WORD_0.NVMCTRL_BOOTPROT* fuse. Set the fuse to *0 bytes* and hit program. Verify that the fuse has been set correctly.



Now you can go to the Memories tab. In the Flash section of the memories tab you have to select the bootloader you'd like to flash to the board. Hit program and wait for the message saying that the flash was successful.

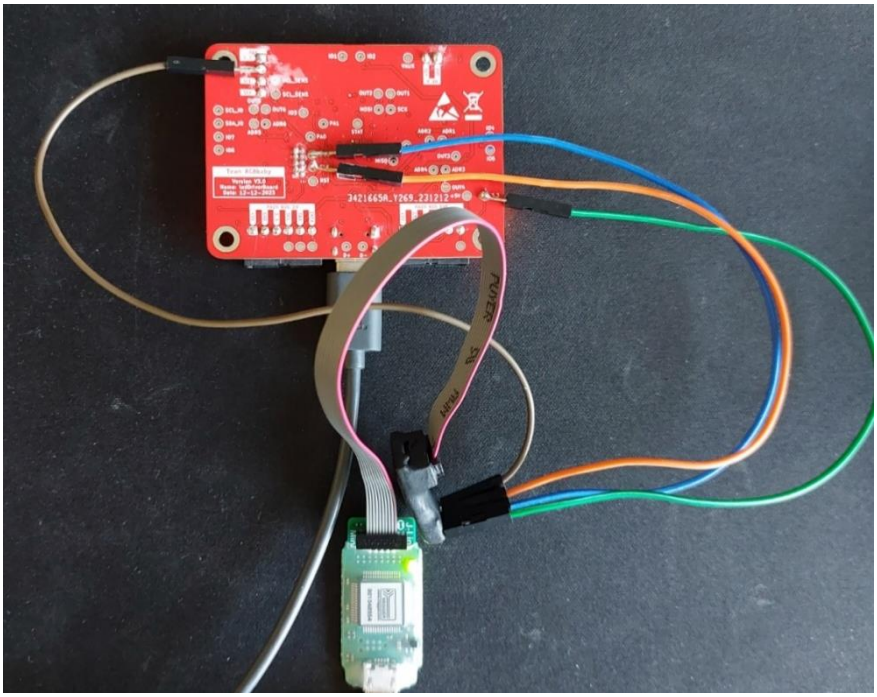


However we are not done yet. We still have to set the *bootprot* fuse back to *8192 bytes* otherwise the bootloader will be overwritten when we flash our code to the board. Don't forget to hit program again after setting the fuse!

2.3.2 – LedDriverBoardV3:

Whether you want to install an Arduino bootloader or want to program the microcontroller using embedded C(++) you will have to use the following method:

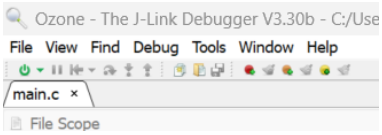
Plug a USB-C cable into the board. Connect the other end of the USB-C cable to a power source like a laptop. After you have done that you can connect the J-LINK EDU mini (Segger, 2024) debugger to the board. Because we didn't connect the debug header correctly you will have to use the solder pads on the back of the board to access the debug interface.



2.3.2.1 – Programming and debugging using Ozone:

You are now ready to upload code to the board via Clion or Visual Studio code using a program called Ozone. In Ozone the program can be started, stopped, halted and debugged.

First you will need a *.jdebug* project. You will find one in the code folders.



The *"ON"* symbol lets you connect (and flash) to a target. After that you can press the symbol next to it to start the program. In the program you will be able to step through code, see register values and set register values. Super handy.

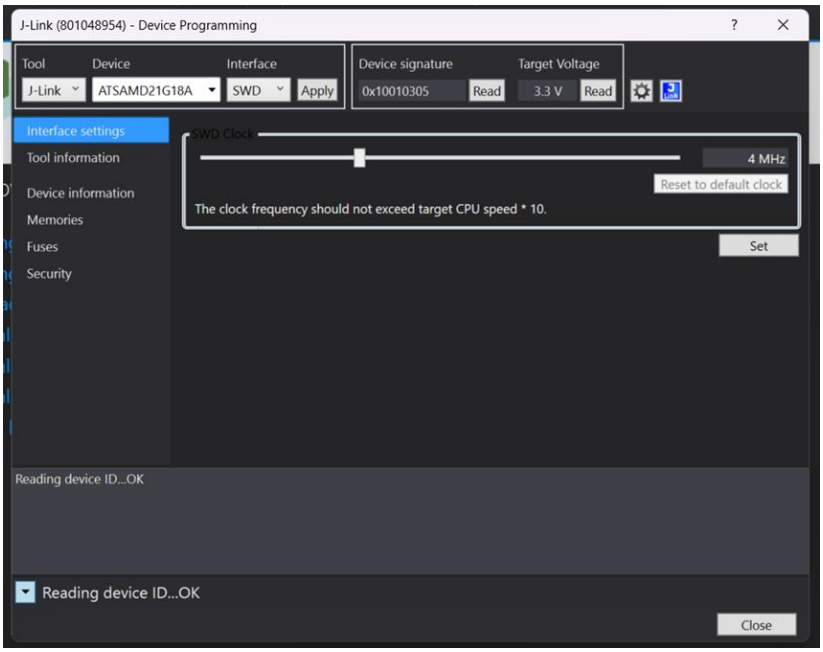
2.3.2.2 – Uploading Arduino bootloader:

To upload the bootloader to the board we will first need to download the bootloader (Adafruit, 2023) After you have done that you can open *Microchip Studio -> Goto Device Programming (Ctrl + Shift + P)*.

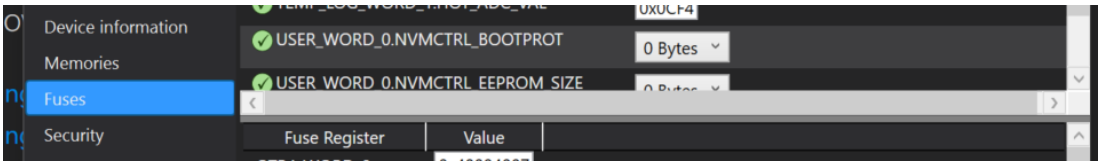
A new window will open:

Tool: J-Link
Device: ATSAMD21JG8A
Interface: SWD

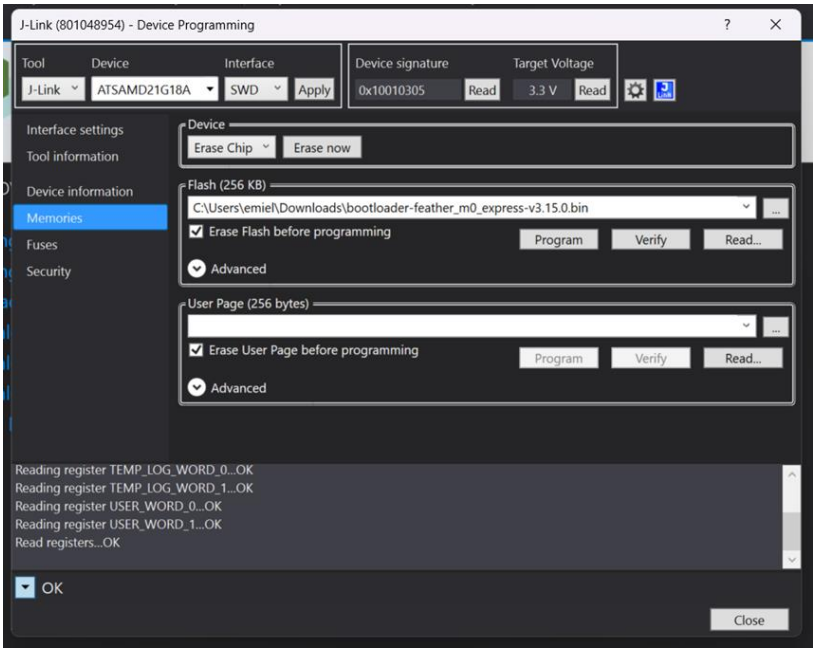
Hit apply. Next to that section you will find the device signature and target voltage buttons. Read both of those.



Go to the fuses tab and scroll down to the *USER_WORD_0.NVMCTRL_BOOTPROT* fuse. Set the fuse to *0 bytes* and hit program. Verify that the fuse has been set correctly.



Now you can go to the Memories tab. In the Flash section of the memories tab you have to select the bootloader you'd like to flash to the board. Hit program and wait for the message saying that the flash was successful.

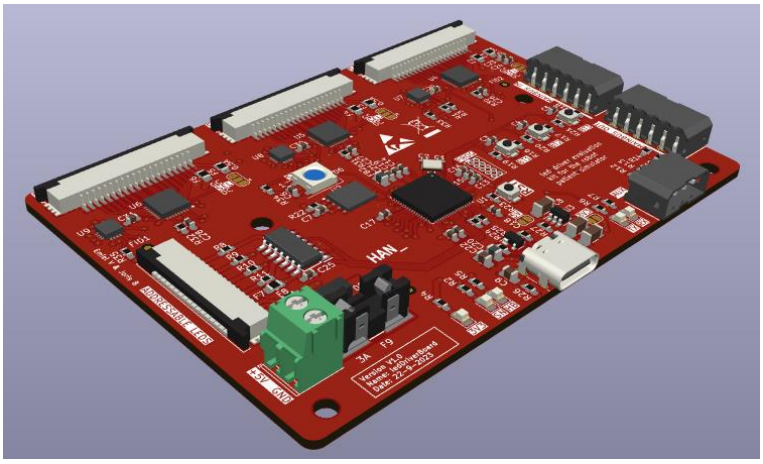


However we are not done yet. We still have to set the *bootprot* fuse back to *8192 bytes* otherwise the bootloader will be overwritten when we flash our code to the board. Don't forget to hit program again after setting the fuse!

3 – Hardware

In this chapter we will describe how you can use the PCBs we have made.

3.1 – LedDriverBoardV2



The second version (V1.0 on the silkscreen, this is a mistake) of our PCB is a test bed for a couple of different methods we wanted to try to drive the LEDs we wanted to test.

3.1.1 – Power:

You can power this board by inserting a USB-C cable in the USB port. The USB port can output *5V@3A max*.

We also included the possibility to use an external power supply (green connector). The supply voltage is *5V max*. The fuse in the fuse holder is a replaceable *2.5A fast blow fuse*.

The default way of powering the system is by using the main communication bus. The connectors for the *main bus* are the two black 6pin connectors on our board.

3.1.2 – First power up:

Assuming that you have installed the bootloader on the chip we can now connect our USB-C cable or debugger to the board. The device should register as a COM port on your windows device. If you're using a debugger you have to manually connect to it using the firmware your debugger uses.

3.1.3 – Programming:

You can use your preferred IDE to program our board. For the test code we have used Arduino. However, we recommend programming on register level due to some limitations with our design. More on that later.

3.1.4 – Inputs:

There are several inputs available on our board. We have broken out three user addressable buttons which can be used for all sorts of applications. The *main bus in* and *main bus out* ports both have an interrupt enabled pin.

3.1.5 – Busses:

We have three I2C busses on out board. The *main system bus*. This bus is used to communicate with other modules. We have an *auxiliary bus*. This bus is used for auxiliary I2C sensors or actuators. And lastly, we have an *IO I2C bus*. This bus is connected to the IO multiplexers. The FRAM-chip is connected via QSPI to the microcontroller. You can use this chip to store data. It has a storage capacity of 64 kB.

Note: Due to time limitations and problems with the hardware of our PCB we have not been able to get these busses to function on this board.

3.1.6 – Outputs:

This board has several output possibilities. Most of them control LEDs. The led drivers are driven with a dedicated communication protocol. You can find more information about this protocol in the manufactures datasheet (Texas Instruments, 2023). The onboard neopixel is connected to the standard neopixel pin on for example a Adafruit Feather M0 Express. The addressable led strip works via the same principle as the onboard neopixel.

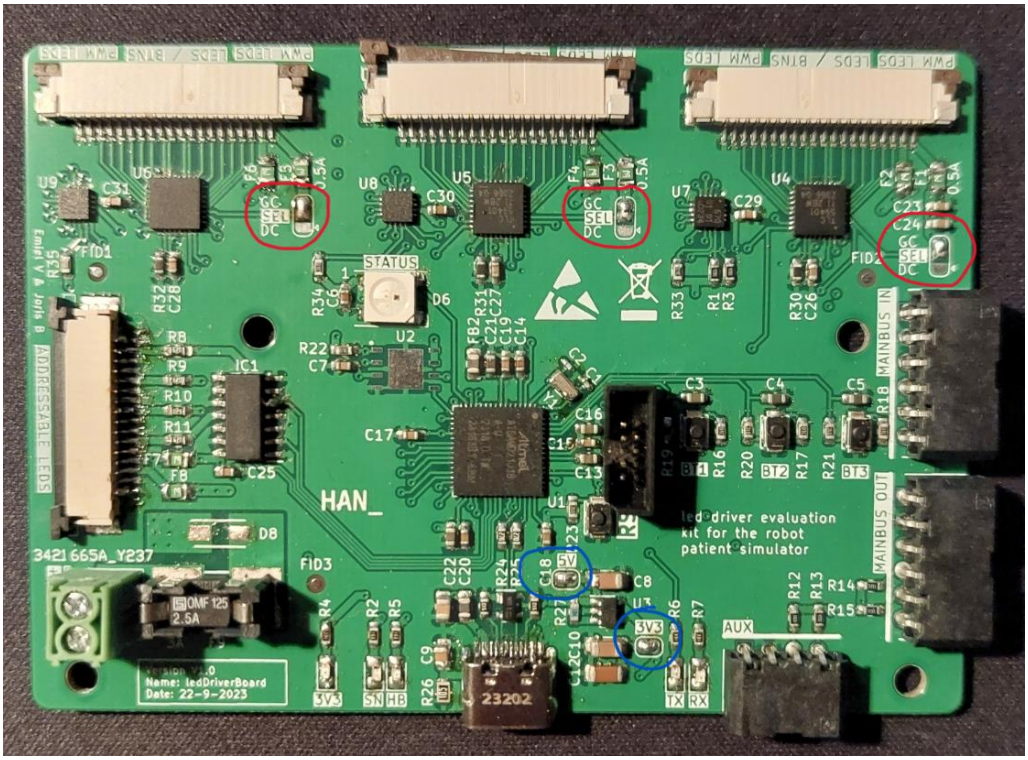
There are several status LEDs on our board. The 3.3V led is not addressable by the user and powers on when the 3.3V LDO (voltage regulator) is supplied with 5V.

3.1.7 – Jumper selections:

There are a bunch of jumpers on this board. In this section we will explain their function. The jumpers marked in red are the mode selection jumpers for the PWM drivers. Currently they have been set to normal mode. The other option is running them in grayscale mode. For more information about these modes. Please refer to the datasheet (Texas Instruments, 2023).

The jumper marked in blue with the 3V3 tag above it connects the LDO (3.3V regulator) to the 3.3V net on the PCB. This jumper must be bridged if no other 3.3V supply is connected. Otherwise the PCB won't do anything.

The other jumper marked in blue with the 5V tag above it connects VBUS (5V from USB port) to the 5V net on the PCB. This jumper can be left open if a 5V supply is attached to the PCB.



3.1.8 – Problems:

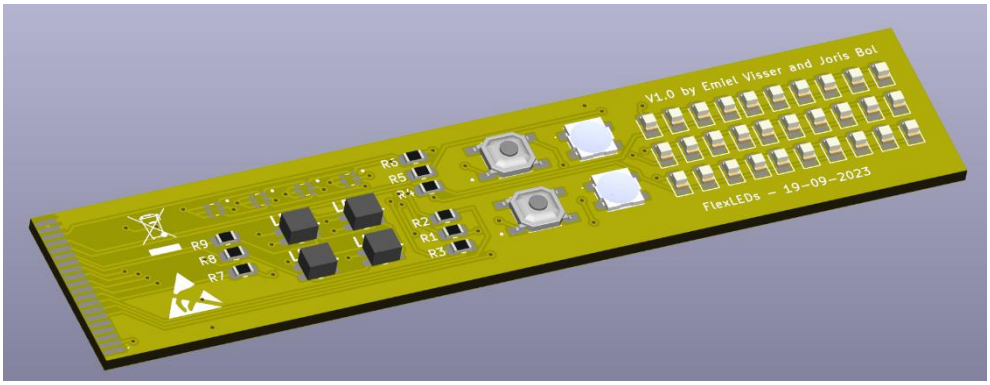
We had a bunch of problems with this board. Mainly finding a suitable Arduino bootloader and libraries for the IO expanders, FRAM chip and led drivers. This is why we recommend programming on register level. We didn't have time for this however so we settled on Arduino for our tests.

3.1.9 – Things to lookout for:

Each led power rail (5V and 3.3V) each have their own 0.5A polyfuse. The maximum allowed current via the external power port is 2.5A. This means this fuse will blow if all led channels are ran at their maximum rating.

The USB and main bus port don't have fuses though. So make sure you don't pull more than 3A through those connectors.

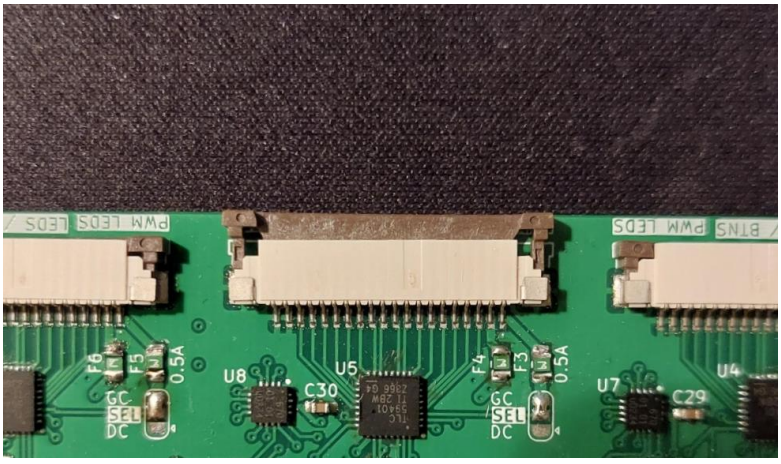
3.2 – FlexledsV1.0



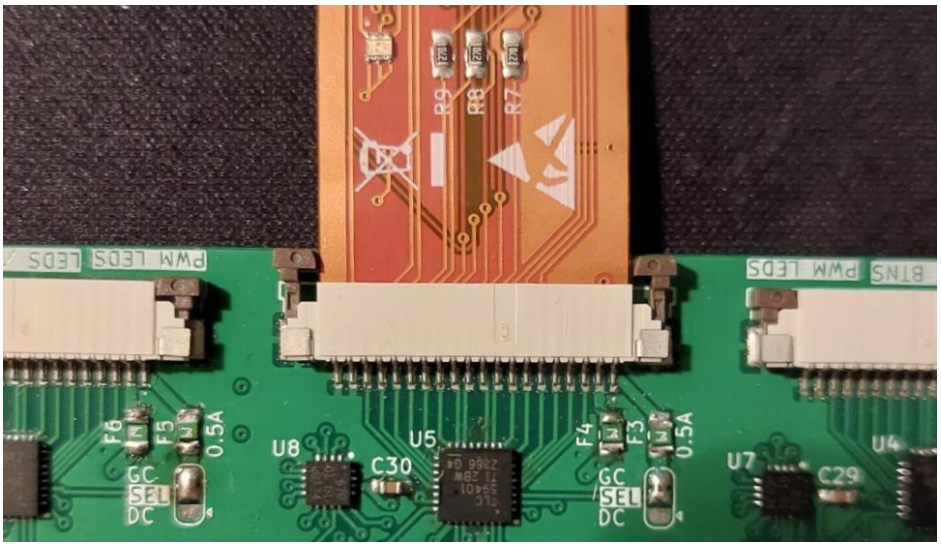
This flexible PCB has a couple of different LEDs we wanted to test on it. This is also the PCB that has two types of buttons on it. This PCB can only be used with the 20pin FPC connector and must be inserted with the components facing up. We have not been able to test this PCB because of the aforementioned issues we ran into with the LedDriverBoardV2.

3.2.1 – Connecting the flex PCB:

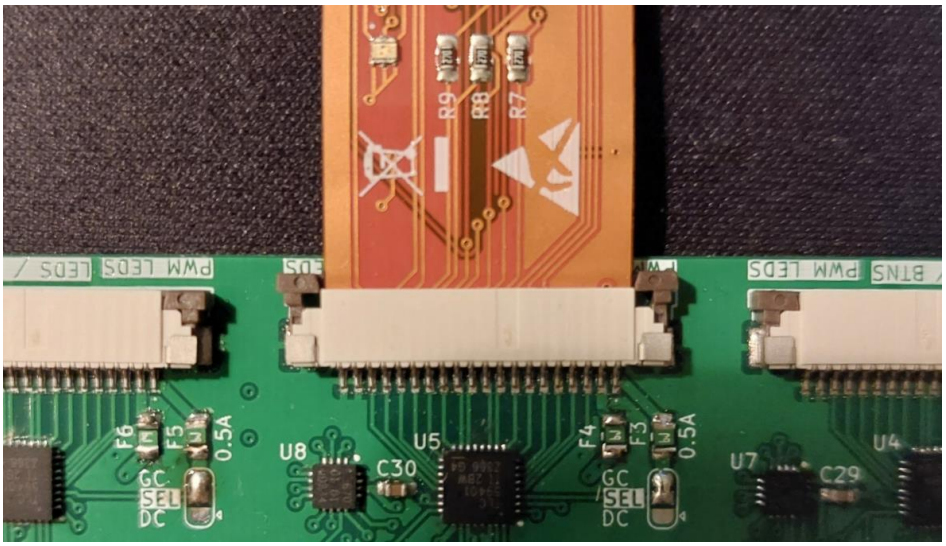
To connect the flex PCB to the main PCB you must first disconnect any power sources connected to the ledDriverBoard. After you have verified that the board is in an unpowered state you can start by pulling the locking tab of one of the 20pin FFC connectors out. The tabs are made out of plastic and break easily if you pull too hard on them. So don't be too rough with them.



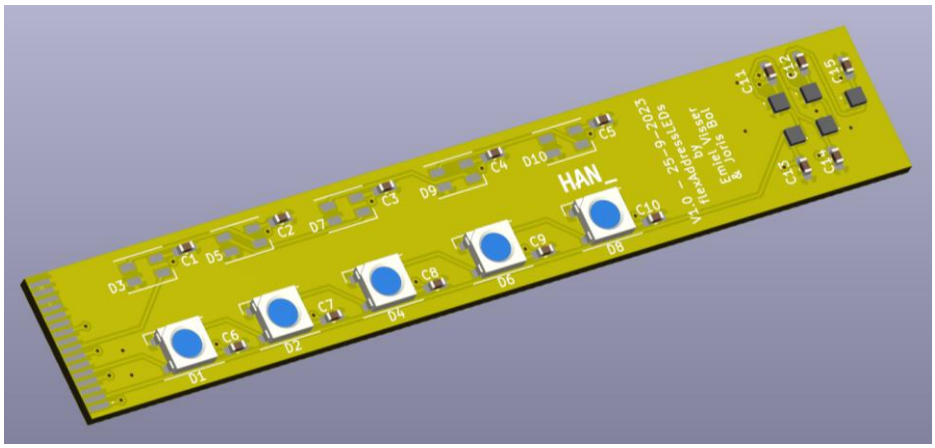
After you have done that you can push the flex PCB, components facing up, in the connector. Make sure the flex PCB is pushed all the way in and is not going in at an angle. This could cause shorts.



Now you can push in the locking tab. It won't lock all the way down because the flex PCB is bit too thick for the connector. Just push it in until the PCB feels snug. It should look something like this:



3.3 – flexAddresLedsV1.0

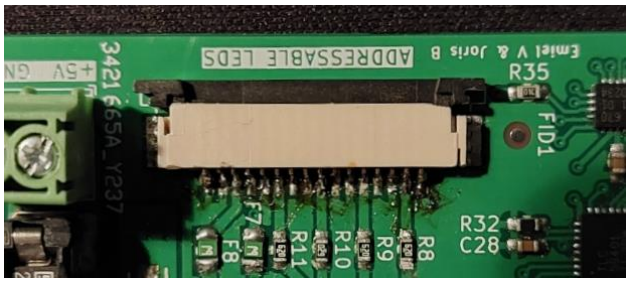


This flexible PCB has the addressable LEDs on it. This PCB has o buttons or other inputs on it in contrary to the flexLedsV1.0 PCB we saw earlier. This flex PCB is only compatible with the 16pin FPC connector. The PCB must be inserted with the components facing up. This board is fully operational.

3.3.1 – Connecting the flex PCB:

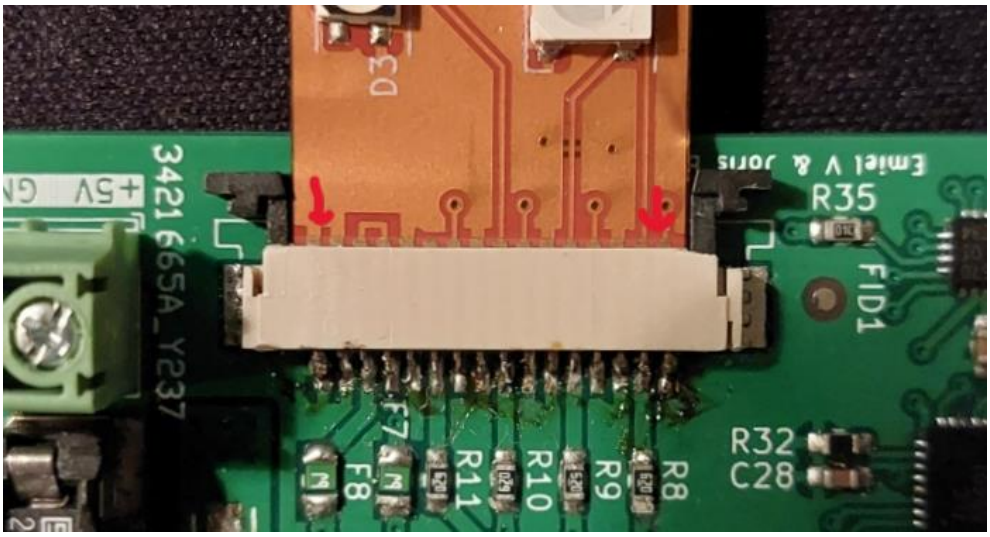
To connect the flex PCB to the main PCB main PCB you must first disconnect any power sources connected to the ledDriverBoard. After you have verified that the board is in an unpowered state you can start by pulling the locking tab of the 16pin FFC connector out.

Note: If you've soldered a new PCB you will have to pull out the plastic tab completely and flip it around 180 degrees. Otherwise the pins in the connector won't make contact with the pads on the flex PCB.

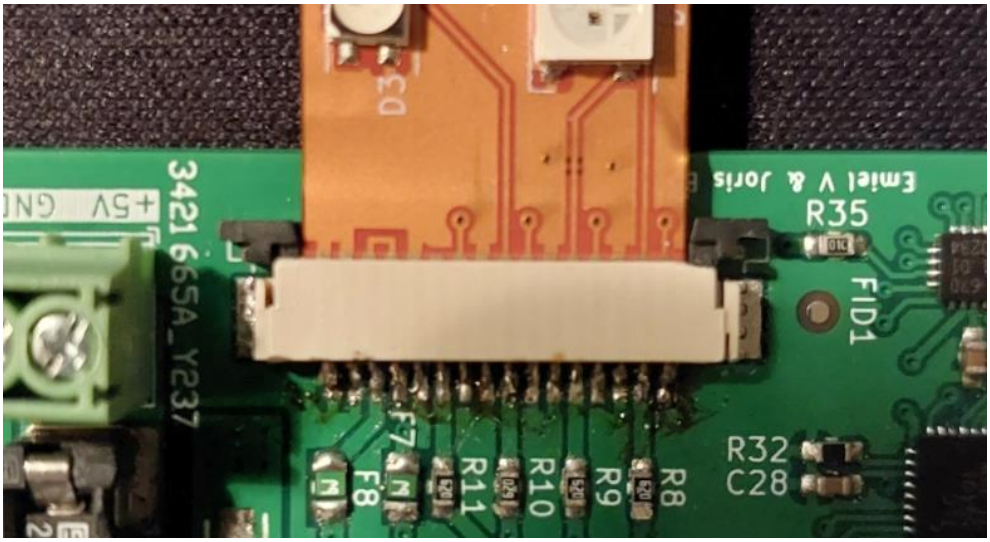


You can now pull the locking tab out carefully. It's easy to pull the tab out completely because the plastic piece locking it in has been broken off.

After you have done that you can push the flex PCB, components facing up, in the connector. Make sure the flex PCB still has a small part of the contact points visible. So don't push it in all the way this time. Make sure the PCB is going in straight and is not going in at an angle. This could cause shorts!

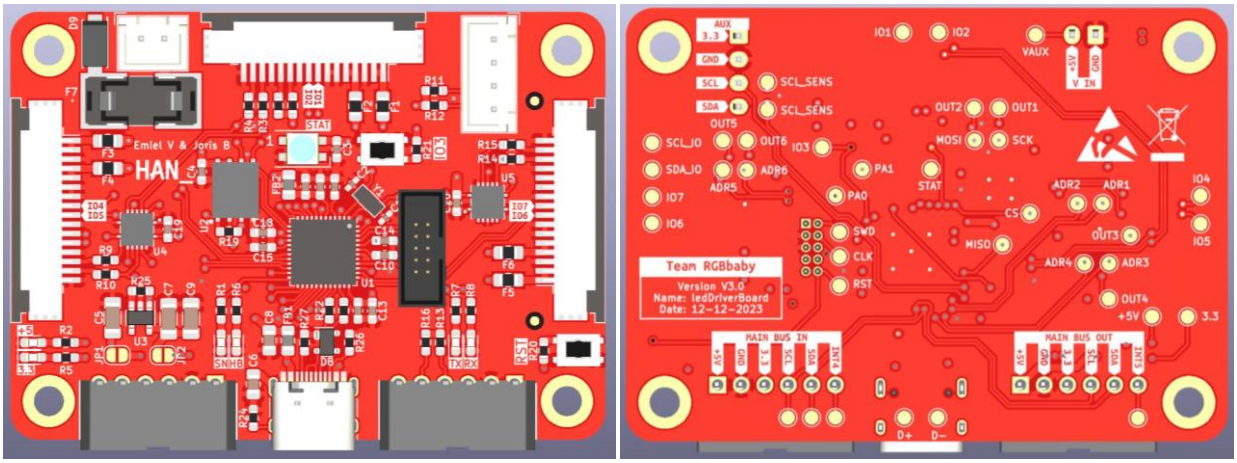


Now you can carefully push in the locking tab. It's easier to do this one side at a time and adjust the flex PCB as needed because the flex PCB tends to go in crooked if you push in both sides of the tab at the same time. Pushing in one side at a time enables you to adjust the orientation of the PCB before it is locked in completely.



3.4 – LedDriverBoardV3

The third version of our PCB is our final version. We decided to drop the PWM drivers and IO expanders and choose to go with addressable LEDs.



3.4.1 – Power:

You can power this board by inserting a USB-C cable in the USB port. The USB port can output *5V@3A max*.

We also included the possibility to use an external power supply (2pin JST connector). The supply voltage is *5V max*. The fuse in the fuse holder is a replaceable *3A fast blow fuse*.

The default way of powering the system is by using the main communication bus. The connectors for the main bus are the two black 6pin connectors.

3.4.2 – First power up:

Assuming that you have installed the bootloader on the chip we can now connect our USB-C cable or debugger to the board. The device should register as a COM port on your windows device. If you're using a debugger you have to manually connect to it using the firmware your debugger uses.

3.4.3 – Programming:

You can use your preferred IDE to program our board. For the test code we have used Arduino. However, we recommend programming on register level because this gives you more flexibility and reliability.

3.4.4 – Inputs:

There are several inputs available on our board. We have broken out one user addressable button which can be used for all sorts of applications. The main bus in and main bus out ports both have an interrupt enabled pin. We also have connected two IO pins to each of the three FFC connectors.

3.4.5 – Busses:

We have three I2C busses on out board. The *main system bus*. This bus is used to communicate with other modules. We have an *auxiliary bus*. This bus is used for auxiliary I2C sensors or actuators. And lastly, we have an *IO I2C bus*. This bus is connected to 16pin FFC connectors and can be used to connect I2C devices that are on the flex PCBs. The FRAM-chip is connected via QSPI to the microcontroller. You can use this chip to store data. It has a storage capacity of 64 kB.

Note: Due to time constraints we have not been able to get these busses to function on this board.

3.4.6 – Outputs:

This board has several output possibilities. Most of them control LEDs. The onboard neopixel is connected to the standard neopixel pin on for example a Adafruit Feather M0 Express. The addressable led strips works via the same principle as the onboard neopixel.

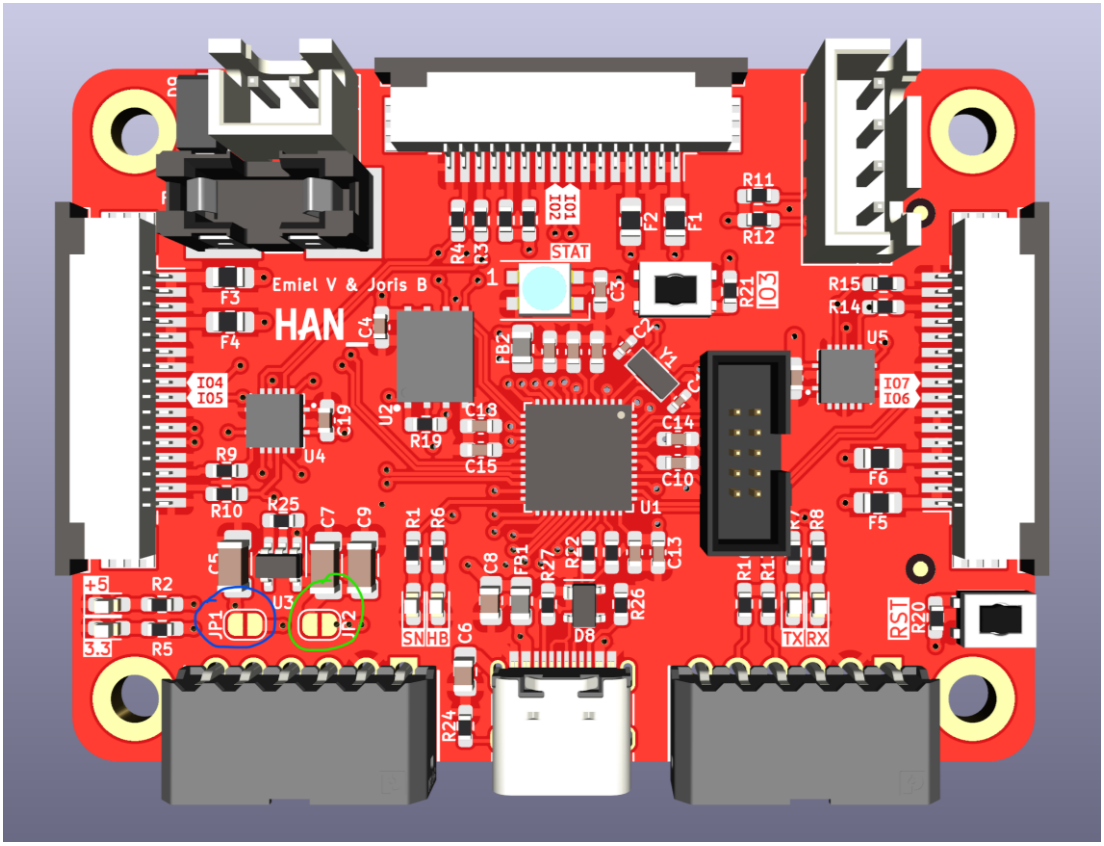
There are several status LEDs on our board. The 5V and 3.3V LEDs are not addressable by the user and power on when 5V is applied and when the 3.3V LDO (voltage regulator) is supplied with 5V.

3.4.7 – Jumper selections:

There are a bunch of jumpers on this board. In this section we will explain their function.

The jumper marked in green connects the LDO (3.3V regulator) to the 3.3V net on the PCB. This jumper must be bridged if no other 3.3V supply is connected. Otherwise the PCB won't do anything.

The other jumper marked in blue connects VBUS (5V from USB port) to the 5V net on the PCB. This jumper can be left open if a 5V supply is attached to the PCB.



3.4.8 – Problems:

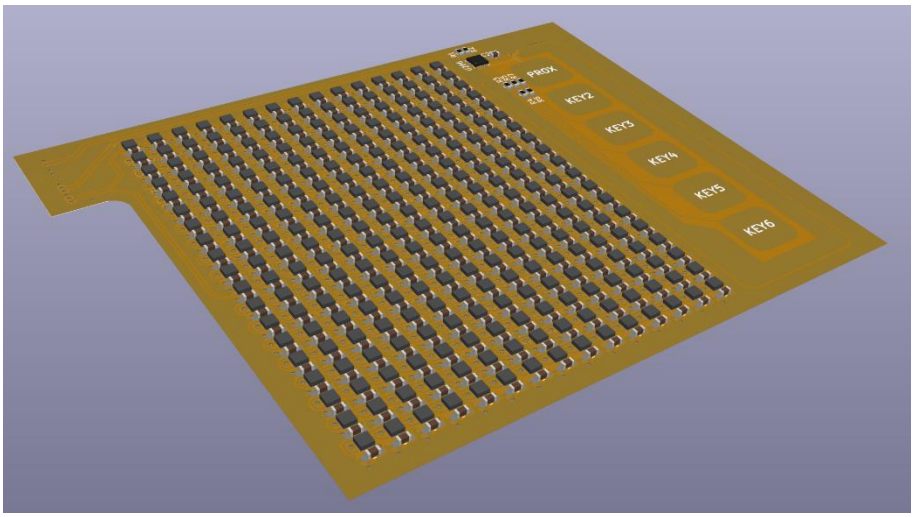
The debug header is incorrectly connected. You must use the 3.3V, GND, SWD and CLK pads on the bottom side of the board.

3.4.9 – Things to lookout for:

Each led power rail (5v and 3.3V) each have their own 1A polyfuse. The maximum allowed current via the external power port is 3A. This means this fuse will blow if all led channels are ran at their maximum rating.

The USB and main bus port don't have fuses though. So make sure you don't pull more than 3A through those connectors.

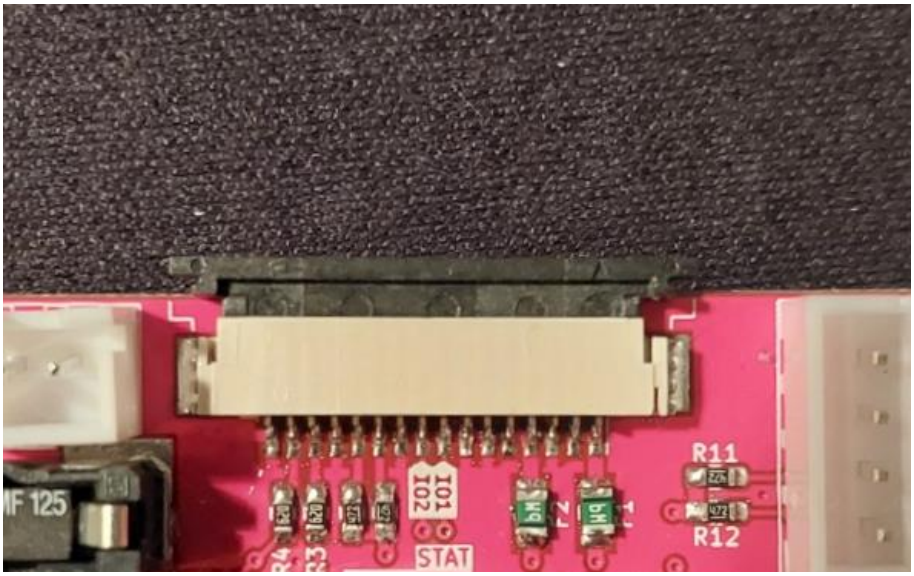
3.5 – flexAddressLedsV2.3



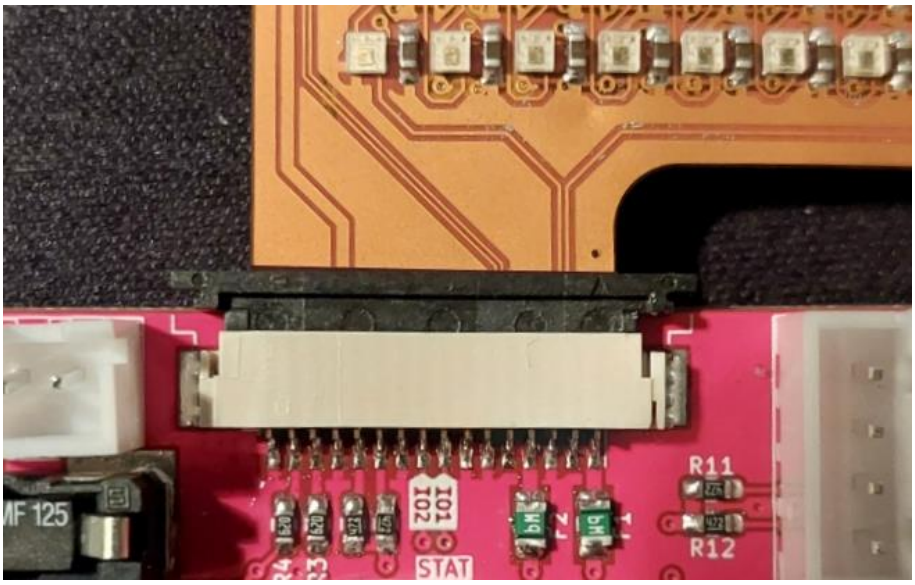
This flexible PCB has a couple of different LEDs we wanted to test on it. This is also the PCB that has two types of buttons on it. This PCB can only be used with the 20pin FPC connector and must be inserted with the components facing up. We have not been able to test this PCB because of the aforementioned issues we ran into with the LedDriverBoardV2.

3.5.1 – Connecting the flex PCB:

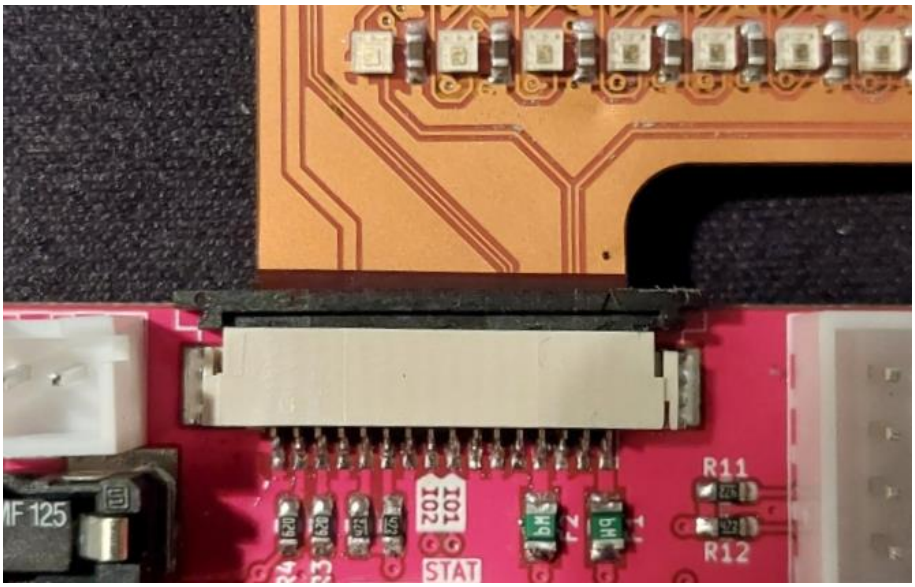
To connect the flex PCB to the main PCB you must first disconnect any power sources connected to the ledDriverBoard. After you have verified that the board is in an unpowered state you can start by pulling the locking tab of one of the 20pin FFC connectors out. The tabs are made out of plastic and break easily if you pull too hard on them. So don't be too rough with them.



After you have done that you can push the flex PCB, components facing up, in the connector. Make sure the flex PCB is pushed all the way in and is not going in at an angle. This could cause shorts.



Now you can push in the locking tab. It won't lock all the way down because the flex PCB is bit too thick for the connector. Just push it in until the PCB feels snug. It should look something like this:



4 – Testing setup

This chapter is about the way we tested certain aspects of our product. This includes unit testing but also the spectrometer setup.

4.1 – PySpectrometer

If you want to recreate the experiments mentioned in the product rapport, you will need the following hardware:

- PicCamera (provided)
- Spectrometer (provided)
- RICOH HF16HL-1b (provided by school)
- Raspberry Pi 4 or higher

This spectrometer project is made by leswright1977. The full documentation can be found at: https://github.com/leswright1977/PySpectrometer2 (Wright, 2023)

Keep in mind that this tutorial is based on the 2022 release.

4.1.1 – Initial setup

You will need a 16GB or more SD-card that fits in your Raspberry Pi. You need to put the latest release of the Raspberry Pi Operating System (OS) on the drive. Its recommend that you use the Raspberry Pi imager to do this. The imager can be downloaded from here that can be downloaded here: https://www.raspberrypi.com/software/ (Raspberrypi foundation, 2023)

After you put the SD-card in the Raspberry Pi, you are now ready to connect the Pi to the internet. This can most easily be done with an ethernet cable.

Make sure your computer is also connected to the same network as your Pi and open the command prompt (CMD). you can now connect to your Raspberry Pi by entering:

```
C:\Users\name> ssh <host name>          (the host name is just the name of your pi)

C:\Users\name> <host name> password: <your password>
```

If everything went correctly, you should see this in your CMD:

```
<host name>@raspberrypi:~ $
```

4.1.2 – Installing the necessary software

We need to install the following software:

- git (this tutorial assumes that you already have git installed)
- python3-opencv

We will start by cloning the repository from GitHub.

```
<host name>@raspberrypi:~ $ cd Documents

<host name>@raspberrypi:~ /Documents $ git clone https://github.com/leswright1977/PySpectrometer2.git
```

Next we will install opencv:

```
<host name>@raspberrypi:~ $ cd

<host name>@raspberrypi:~ $ sudo apt-get install python3-opencv
```

4.1.3 – Running the program

```
<host name>@raspberrypi:~ $ cd Documents/PySpectrometer2/src

<host name>@raspberrypi:~ /Documents/PySpectrometer2/src $ chmod +x PySpectrometer2-Pycam2-v1.0.py

<host name>@raspberrypi:~ /Documents/PySpectrometer2/src $ ./PySpectrometer2-Pycam-v1.0.py
```

To see what the program does, you need to connect a monitor to the pi or use VNC Viewer.

It is important that you connect the camera to your pi first before running the program, otherwise the program will NOT run.

4.1.4 – Hardware

The test setup parts should be provided to you, if not, you can 3D print them yourself with the files we made for you. Assemble the parts and run the program.

5 – Documents and documentation method

This chapter describes where we documented certain topics and how we worked. Our research produced the following documents:

- Plan of Approach.
- Preliminary research – ESE.
- Preliminary research – IPO.
- Product report.
- **This document.**

Plan of approach
This document is one of lesser importance for you, although we do recommend you make one yourself. This document contains the semester planning and other organizational information, specific to us.

Preliminary research – ESE
If you want to find out more about why we made certain choices regarding component selection, this is the place to be. It talks about why we chose the led that we did and why we selected the pressure sensor we use.

Preliminary research – IPO
This is one of the more important and interesting reports regarding the creation of the skin. Noted is the selection of the silicone and pigments used as well as the research regarding the thickness and colour of the skin.

Product report
This is the main report, containing all the technical details and the project process. Important chapters in this report are:

- Functional design:
  You can find to what and goal we worked end what the starting conditions of this project were.
- Technical design:
  Here all the technical requirements can be found, some are mandatory, made by the product owner and some are made by us.
- Realisation:
  In this chapter you can find how we developed the actuator as well as the skin. If you are wondering how we did something or why regarding the actual construction of our product, this is the place to look.

This document
For more information about this document, we kindly refer you to the preface on page 4.

6 – Bibliography

Adafruit. (2023). *v3.15.0.* Retrieved from github: https://github.com/adafruit/uf2-samdx1/releases/download/v3.15.0/bootloader-feather_m0_express-v3.15.0.bin

ARM. (2023). *arm gnu toolchain downloads.* Retrieved from developer.arm.com: https://developer.arm.com/downloads/-/arm-gnu-toolchain-downloads

AtmelUniversityFrance. (2023). *atmel_samd21_xpro_v1.* Retrieved from github: https://github.com/AtmelUniversityFrance/atmel-samd21-xpro-boardmanagermodule/blob/master/module/bootloaders/atmel_samd21_xpro_v1/samd21_sam_ba.hex

Charras, J.-P. (2023). *KiCad download page*. Retrieved from kicad.org: https://www.kicad.org/download/

GitHub. (2023). *cloning a repository.* Retrieved from docs.github.com: https://docs.github.com/en/repositories/creating-and-managing-repositories/cloning-a-repository

Hogeweij, V. (2023). *Flash build instructions.* Retrieved from GitHub: https://robotpatient.github.io/Manikin_Software/General/flash_build_instructions/

Hogeweij, V. (2023). *samd21_bare_cmake.* Retrieved from github: https://github.com/Hoog-V/SAMD21_BARE_CMAKE

JetBrains. (2023). *clion*. Retrieved from jetbrains: https://www.jetbrains.com/clion/download/#section=windows

JetBrains. (2023). *students*. Retrieved from jetbrains: https://www.jetbrains.com/shop/eform/students

Raspberrypi foundation. (2023). *software.* Retrieved from raspberrypi: https://www.raspberrypi.com/software/

Segger. (2024). *j-link-edu-mini*. Retrieved from segger: https://www.segger.com/products/debug-probes/j-link/models/j-link-edu-mini/

Texas Instruments. (2023). *16-channel led driver with dot correction and grayscale pwm control.* Retrieved from TI: https://www.ti.com/lit/ds/symlink/tlc59401.pdf?ts=1704793969761&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FTLC59401

Wright, L. (2023). *PySpectrometerV2*. Retrieved from github: https://github.com/leswright1977/PySpectrometer2