Northwestern CE346
Fall 2025

Lab 0 - Software Setup

Goals

- Get a build environment configured for the future labs and project
- Run C code on the Microbit
- Simple debugging in an embedded context

Equipment

- Computer that you will use for labs
    - If Windows: needs at least 20 GB of space
    - USB ports
- Micro:bit and cables (you can do all of the setup except testing without this)

This lab is required, but not held in person. Every individual student needs to work through it to have a working setup. If you run into problems, please reach out on Piazza or during Office Hours and I will provide help!!

It's always possible that these instructions have bugs, or commands that are out-of-date. **If you encounter any issues, let me know on Piazza and I'll update this document to fix it.**

At the end, you'll submit a copy-paste from terminal to Gradescope so I can see that you've finished these steps. Be sure to submit that before the first lab session!!

Github classroom link: https://classroom.github.com/a/OiF992dU
Submit on Gradescope when finished!

Table of Contents

MacOS Instructions

The good news here is that MacOS natively supports all of the software we need. Since it's not a clean installation though (presumably you've been using your Mac for other programming tasks) some of these steps might fail, or react differently. Power through as best you can and ask questions whenever you need!

To my knowledge this will all work great on both Intel and ARM Macs. I have a personal Intel Macbook that can program these boards (although I installed all the stuff years and years ago). I tested all of these instructions on an ARM M1 Macbook Air.

- Open a terminal window
    - We'll run all of the following `green` commands in the terminal window

- Install MacOS command line developer tools by running `xcode-select --install`
    - You may have to agree to some terms and conditions
    - This might complete immediately if it's already installed. If not, it'll take a few minutes to finish (and it's just *terrible* at estimating the time remaining for some reason)

- Install Homebrew by running: `/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"` (that's all one command)
    - If that's too much to type, you can go here and copy the "Install Homebrew" command, it's the same thing: https://brew.sh/
    - You should also be able to copy-paste from here into terminal. You'll have to use Cmd+Shift+V to paste, or right click and choose "Paste".
    - You'll have to type your password in to run it. Note that it won't show you any feedback when you type, just type the password anyways and hit enter
    - This shouldn't take all that long to install

- Homebrew might say "Run these two commands in your terminal to add Homebrew to your PATH:". If it does, run both of those commands.
    - You can highlight, right click, copy, and then paste it to run it
    - They add things to PATH so you can find executables later. We can always fix after-the-fact if we need to. You'll know things are broken if it can't find the `arm-none-eabi-gcc` executable even after you install it.

- Install our compiler with `brew install gcc-arm-embedded`

- Install our JTAG tools with `brew install open-ocd`

- Install our serial console with `pip3 install pyserial`
    - You might get a warning that “blah blah is not on PATH”. You’ll need to add it to your PATH. If you don’t know how to do this, the following will work:
      ```
      echo "export PATH=\"`python3 -m site --user-base`/bin:\$PATH\"" >> ~/.zshrc
      ```
        - All one command
        - If you’re using Bash, that should either be ~/.bashrc or ~/.profile instead of ~/.zshrc
        - Then you’ll need to close your terminal and open a new one
    - For more modern systems, you’ll get a big complaint about how you should use `pipx` instead. Here are the steps:
        - `brew install pipx`
        - `pipx install pyserial`
        - `pipx ensurepath`

- Create your Github assignment repo
    - There is a github classroom link on the first page of this document. Click it!
    - Pick a group name “really really doesn’t matter for this assignment”
    - Generally, do what github classroom says
    - At the end, it should create a new private repo that you have access to for your code
    - That link might 404. If so, you first have to go to https://github.com/nu-ce346-student and join the organization
    - **Before you clone your repo**: you must also create an SSH key. SSH keys are well-explained here and there are instructions to add the key to Github

- Clone the github classroom repo with
  `git clone <YOUR-REPO-SSH-URL-HERE> nu-microbit-base`
    - That also renames the folder to `nu-microbit-base` as you’ll be using the same repo for the entire quarter
    - Feel free to change it to whatever you like as long as you remember it

- `cd` into the repo and then run `git submodule update --init --recursive`
    - This is the magic command that fixes all git submodule issues
    - This will take a hot minute to run. There’s lots of stuff to download

- `cd` into “software/apps/blink” and run `make`
    - This should compile the code if everything is working!!
    - You’re done! There’s some bonus stuff below though.

- IF COMPILING DOESN'T WORK:
    - It could be an issue with things not being in your PATH. That would result in it not finding certain executables you installed, like `arm-none-eabi-gcc`. Try figuring out where they installed and add them to your PATH.

    - It could be an issue with a Space character in the file path. None of the directories including the code may have a space character (or other special character like plus or colon). They break Makefiles hard. A common issue here is if your username has a space in it. The solution is to move the directory to somewhere without any spaces in the folder names.

- If you have a microbit already, you can run `make flash` and that will actually load the program onto the board. The LED should start blinking

- Also make sure you have some editor installed that you're happy with. You'll use terminal to compile and flash the board, but you don't have to edit files in terminal if you don't want to. VSCode is totally fine, for instance.

Windows Instructions

We don't support native Windows, so you really need to have Linux instead. There are two options here: Windows Subsystem for Linux (WSL) or a virtual machine. Either is totally fine, but I think WSL is probably preferred at this point.

WSL Instructions

For WSL setup, I can't write better instructions that Microsoft already did:

- https://learn.microsoft.com/en-us/windows/wsl/install
    - Just follow that. Only the first part is really necessary

- https://learn.microsoft.com/en-us/windows/wsl/connect-usb
    - You'll need to follow this too to install the `usbipd` service

- You will also need to run commands to connect USB devices to WSL. Unfortunately, you'll have to run these every time you use WSL. It's not too much work though. You can test that once you have a board.
    - Note: You run the `usbipd` commands on **Windows** not in WSL

- Once you've got stuff installed, you should jump to the Linux instructions and do all of that to install stuff for lab. No need for the rest of these Windows instructions.

- Note: when you download the Git repo, you should choose to keep the files for it in Linux, not in Windows (i.e., not in the shared `c/` drive). Having your files shared between Windows and Linux makes accessing them about 100x slower.

Problems with USB Device

- Some students in F25 had issues connecting to USB devices we had to do the following:
    1. Create a file `/etc/udev/rules.d/99-openocd-ce346.rules`
    2. Edit that file (will have to use `sudo` to edit it) and add the following line:
       ```
       SUBSYSTEM=="usb", ATTR{idVendor}=="0d28",
       ATTR{idProduct}=="0204", MODE="0666", GROUP="plugdev"
       ```
    3. Create a file `/etc/wsl.conf`
    4. Edit that file (also needs `sudo`) and add the following two lines:
       ```
       [boot]
       command="service udev start"
       ```
    5. Reboot your computer. Then reattach the USB device. And things should work?? Definitely tell course staff if they don't.
- Branden notes:

    - Enabling systemd might have been sufficient? https://learn.microsoft.com/en-us/windows/wsl/systemd#how-to-enable-systemd
    - https://github.com/dorssel/usbipd-win/discussions/347

VSCode Intellisense

- If you're using WSL, the Intellisense in VSCode might be screwed up. Justin Ansell (F25) put together some instructions on how to fix this. Do these *last* after the Linux setup.
    - Add the "WSL" extension to VSCode
    - Run "wsl" in a Windows terminal where the project folder is located
    - Run "pwd" in the WSL terminal to get the Linux filepath of the project folder, copy that filepath
    - Go to VSCode
    - Press "Ctrl + Shift + P" to run a command
    - Run "WSL: Connect to WSL in New Window"
    - Click "Open Folder"
    - When prompted, paste in the Linux filepath of the folder (DO NOT click "Show Local", that will revert back to the Windows filepath which breaks IntelliSense)

Virtual Machine Instructions

If WSL doesn't work for you, or you want to do a virtual machine instead

This really isn't all that bad. There are a few steps here, but most of it is being careful to select the right options in dialog boxes and then waiting for stuff to install.

Heads up: this does require some storage space. On the order of 20 GB of space.

1. Install a Virtual Machine

A virtual machine (VM) allows you to run an operating system inside a windowed environment while running windows. Basically, it creates a "virtual" computer inside your computer. This has some consequences about speed and security (see CS343), but works great for our needs.

We previously used VirtualBox as our VM software, but it's got some really annoying flaws. I personally use VMWare Workstation, so I'm hoping it works much better for this quarter.

- Install VMWare Workstation. This is annoyingly so much harder than it needs to be.

- First, You can download it:
    - Wow, this is way harder than it should be
    - Go here: https://support.broadcom.com/group/ecx/free-downloads
        - Make an account and verify it and everything
    - Go here again with an account this time: https://support.broadcom.com/group/ecx/free-downloads
    - Click "VMware Workstation Pro" in the bottom right.
    - Click "VMware Workstation Pro 17.0 for Windows"
        - Pick the newest release number
        - Should be 17.6.3 right now
    - You'll need to agree to terms and conditions in the top right and then hit the download "cloud" button
    - Then you'll need to add an address and certify that you're in the US
        - For no particular reason, Tech's address is: 2145 Sheridan Road, Evanston, IL 60208
    - Then you can download the exe file

- Double-click the exe to start the installation

- Follow the installation instructions to install it
    - The defaults are all fine. Just keep hitting "yes" for a while.

- Finally, open it. It'll ask for a license and you should select the "...for Personal Use" radio button and hit Continue.

2. Install Linux on the Virtual Machine

Many flavors of Linux would work just fine, but we're going to use Ubuntu. It's very popular and widely used with lots of support on the internet.

- Download Ubuntu: https://ubuntu.com/download/desktop
    - Download the most recent LTS (Long Term Support) version. At time of writing this was 24.04.2 LTS
    - This is about 6 GB in size, so it's going to take a while to download.
    - Be sure to delete it after you're done with this setup if you're short on space!

- Open VMWare and click the "Create a New Virtual Machine" button on the homepage
    - Choose "Typical"
    - Point the "Installer Disc image file (iso)" at the Ubuntu image you downloaded
        - It should say "This operating system will use Easy Install" which lets you skip a bunch of steps
    - Enter a username and password
        - Your username MUST NOT have a space character in it
        - DO NOT forget this password. You'll need it to install things
    - You can name your machine whatever you like and the default installation directory is probably fine.
        - I use the theming of mythological gods. Professor Dinda's group uses fancy cheeses. My first internship used Decepticons until they ran out, then used Care Bears. Pick a naming scheme that makes you happy.
    - Set "Maximum disk size" to at least 100 GB (I usually pick 200 GB)
        - This isn't necessarily how much space it'll use. Just the maximum. It's totally possible to increase later, but it's a bit of a pain. So, picking a big size now is likely worthwhile.
        - You should keep the disk split into multiple files (the default)
    - That should reach the "Ready to Create Virtual Machine" window. **DON'T HIT ANYTHING YET THOUGH**. Continue to the next step.

- Edit the VM Hardware
    - If you are still on the "Ready to Create Virtual Machine" window, you can hit the "Customize hardware" button. Otherwise you'll need to go to "VM->Settings" in the menubar.
    - For "Memory", pick half of the RAM available on your computer. So if you have 8 GB, pick 4096. Or if you have 16 GB, pick 8192.
    - For "Processors", the default is probably fine, although you could again pick half of your cores.

        - Do select “Virtualize Intel VT-x/EPT or AMD-V/RVI” if you can. It should make the VM run faster. Although it might give you a weird error when you try to boot the VM for the first time if the option isn’t also enabled already in your BIOS. If that’s the case, you can just turn this off again.
    - For “USB Controller”, you’re going to want “USB compatibility” set to “USB 3.1”
    - The rest of the settings should be fine as-is
    - If you’re still on the “Ready to Create Virtual Machine” window, you can now hit finish.

- Let the VM install stuff for a while. Eventually it’ll hit a screen asking you for input that says “Choose your language”
    - The defaults are fine for the first few: Language, Accessibility, Keyboard Layout, Connect to the Internet, Install Ubuntu, Interactive Installation, Default Selection
    - I choose to “Install recommended proprietary software” and check both of those boxes
    - “How do you want to install Ubuntu?” the default is fine.
        - It’ll erase the disk file it created, not your actual disk.
    - “Create your account” You’ll have to enter those details again including password.
        - Remember that your username MUST NOT have a space in it.
        - Uncheck the “Require my password to log in” box. Someone will already need to be logged into your computer to open this anyways, so you don’t need the extra login requirement.
    - “Timezone” automatically worked for me.
    - Hit the “Install” button and let stuff run for a while. This will take probably 10-15 minutes.
    - When it’s done, hit the “Restart Now” green button

- Once it reboots, it should log you in automatically. You’ll have to click through a “Welcome to Ubuntu” dialogue for a bit. Nothing important there.

- To make your life better:
    - First, hit the x button on that yellow pop-up at the bottom of VMware so that goes away forever
    - Second, you can resize the VMWare window, and the Ubuntu desktop will resize as well
    - Third, in VMware View->Customize you can deselect “Library” and “Tabs” to get rid of cruft you don’t need onscreen
    - Fourth, back in Ubuntu, right click the desktop and choose “Display Settings” then “Power” on the right. Set “Screen Blank” to Never.

- Open a terminal in Ubuntu
    - You can click the desktop and hit “Ctrl+Alt+T”
    - Or you can click the Terminal icon on the left bar

- Update Ubuntu
    - Run `sudo apt update`
    - Run `sudo apt upgrade`

- You should now have a “usable” Ubuntu installation. Continue on to the “Linux” instructions to actually get the stuff installed that you need for lab.

Linux Instructions

This part is pretty easy. If you've got Linux you're only a few quick steps away from having a working setup.

- Install various pre-requisites: `sudo apt install build-essential python3-pip python3-serial git vim emacs micro meld screen`

- Install our compiler: `sudo apt install gcc-arm-none-eabi`

- Install our JTAG tools with: `sudo apt install openocd`

- Create your Github assignment repo
    - There is a github classroom link on the first page of this document. Click it!
    - Pick a group name "really really doesn't matter for this assignment"
    - Generally, do what github classroom says
    - At the end, it should create a new private repo that you have access to for your code
    - That link might 404. If so, you first have to go to https://github.com/nu-ce346-student and join the organization
    - **Before you clone your repo**: you must also create an SSH key. SSH keys are well-explained here and there are instructions to add the key to Github
        - Warning: if you're running Windows, make sure you create your key within your Linux VM, not in Windows. At the top of that URL you can pick which OS you want instructions for: choose Linux

- Clone the github classroom repo with `git clone <YOUR-REPO-SSH-URL-HERE> nu-microbit-base`
    - That also renames the folder to `nu-microbit-base` as you'll be using the same repo for the entire quarter
    - Feel free to change it to whatever you like as long as you remember it

- `cd` into the repo and then run `git submodule update --init --recursive`
    - This is the magic command that fixes all git submodule issues
    - This will take a hot minute to run. There's lots of stuff to download

- `cd` into "software/apps/blink" and run `make`
    - This should compile the code if everything is working!!
    - You're done! There's some bonus stuff below though.

- IF COMPILING DOESN'T WORK:
    - It could be an issue with things not being in your PATH. That would result in it not finding certain executables you installed, like `arm-none-eabi-gcc`. Try figuring

out where they installed and add them to your PATH.

    - It could be an issue with a Space character in the file path. None of the directories including the code may have a space character (or other special character like plus or colon). They break Makefiles hard. A common issue here is if your username has a space in it. The solution is to move the directory to somewhere without any spaces in the folder names.

- If you have a microbit already, you can run `make flash` and that will actually load the program onto the board. The LED should start blinking
    - If you're using a VM, before flashing when you first plug in the Microbit, you'll need to go in the VM to "Devices->USB" and check the box next to "Arm BBC micro:bit" to attach it to the VM

- Also make sure you have some editor installed that you're happy with. You'll use terminal to compile and flash the board, but you don't have to edit files in terminal if you don't want to. VSCode is totally fine, for instance.

Gradescope Submission

To prove that you've completed this lab, I'll have you copy-paste the last few lines of output from compiling an application into Gradescope. This is graded on completion: you did it or you didn't. There's not some correct answer (apart from whatever the compiler outputs).

- Go to https://www.gradescope.com/ and find the CE346 course
- The assignment is called "Lab0 Software Setup"
- There's just one question: enter the last few lines of output from compiling the `blink` application. To do this:
  - `cd` into your cloned repo
  - `cd` into `software/apps/blink/`
  - Run `make clean && make`
    - This removes all existing compilation results, and then compiles it
  - You should be able to highlight the last five lines and right-click to copy them from the terminal
  - Paste the last five lines into Gradescope and submit
- That's everything! Good job completing this.