Lab 0 - Personal Lab 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It's always possible that these instructions have bugs, or commands that are out-of-date.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 and I'll update this document to fix it.

There's no submission here to prove you've completed this. Just be sure to do it before we have our first lab session!!

Index:

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`

b

- 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

- Clone the class github repo with `git clone https://github.com/nu-ce346/nu-microbit-base.git`

- `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.

Windows Instructions

We don't support native Windows, so you really need to have Linux instead. You've got two options here: you could install a virtual machine (these instructions cover that) or you could use Windows Subsystem for Linux (WSL) if you're feeling really bold. Instructions for WSL are at the end.

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.

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 here: https://softwareupdate.vmware.com/cds/vmw-desktop/ws/17.6.0/24238078/windows/core/

- Now, you're going to need to open Command prompt and run the following commands (you can use tab-completion instead of typing these!!):
    - `cd Downloads`
    - `tar -xf VMware-workstation-17.6.0-24238078.exe.tar`
    - `VMware-workstation-17.6.0-24238078.exe`
    - That should start the installation dialog

- 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 24.04.1 LTS
    - This is about 5 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`

- Clone the class github repo with `git clone https://github.com/nu-ce346/nu-microbit-base.git`

- `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.

Bonus - WSL Instructions

Courtesy of Joshua Fiest

This is still fairly experimental. You may run into weird issues here. This is not recommended, but you're welcome to try it if you want to! Be sure to read the whole thing first, as there are some updates at the end.

Warning: as these are from Fall 2021, they might increasingly be out-of-date. I haven't tried them personally. If you do try it yourself, let me know how it goes please!

I was able to get the required software to program the Microbit working on my computer using WSL (Windows Subsystem for Linux). This worked for me on Windows 10 21H1. Here's how I did it.

1. Install WSL
    1. Using the search bar on your computer, go to Turn Windows features on or off, then click the check boxes to enable Windows Subsystem for Linux and Hyper-V Platform (If you have Windows Pro, you can also enable Hyper-V Management Tools to make virtual machines, but you don't have to)
    2. In an administrator command prompt, run the command "wsl --update" followed by "wsl --shutdown"
    3. If you already had WSL installed and set up, make sure you are running WSL2 by running the command "wsl -l -v." It should list the installed Linux distros and their versions, make sure the version is 2.
    4. In the Microsoft Store, search for and install Ubuntu
    5. If you are running Windows 10, you will also have to install an XServer like GWSL so that WSL can use a GUI (graphical user interface)
        1. To install GWSL, search for it on the Microsoft Store, then install it. Once it's installed, run it, and click on GWSL Distro Tools -> Display/Audio Auto-Exporting to configure WSL to use GWSL for graphics.
        2. Note: configuring GWSL will prevent WSLg from working properly. If you upgrade to Windows 11, be sure to revert the changes made by GWSL in the Linux installation by opening GWSL in Windows and clicking on GWSL Distro Tools -> More Shells and Options -> ~Clean GWSL Additions. You can then uninstall GWSL since it isn't needed to get WSL graphics to work on Windows 11.
2. Install the 64-bit Windows version of OpenOCD
3. Start Ubuntu by typing it in the search bar and set up your account. It may take a couple of minutes to start the first time.
4. Follow the instructions in the Getting Started lab to set up your Ubuntu installation for programming the Microbit
5. ~~When programming the Microbit, run the J-link Remote Server that was installed with J-link on your windows computer. It should give you an IP address that you can type (or copy and paste) when asked for it while running "make flash." If you are using Windows~~

~~10 with GWSL, make sure to start GWSL before running "make flash" or you won't get the graphical pop-up asking for the IP address.~~

6. ~~Because Ubuntu doesn't have direct access to your USB port, you will need a Windows serial console application instead of miniterm. I used the one built into the Arduino IDE, but PUTTY or something else should work too.~~

~~On Windows 11 21H2, there are two very annoying bugs: the J-link remote server may not show the IP address (you can get it by running "ipconfig" in the Windows command prompt) and if you hit the "Enter" key on your keyboard in the J-link emulator selection screen from Ubuntu, it will continue to think Enter is pressed the next time that window opens, which will prevent you from using it. To get around this, click "Yes" instead of using the Enter key, and if you accidentally hit the Enter key, you can restart Ubuntu by running "wsl --shutdown" in the Windows command prompt.~~

Update 11/9/2021: WSL2 now has compatibility with USB devices. If set up properly, this removes the need for using a separate Windows serial monitor and the J-link remote server. https://devblogs.microsoft.com/commandline/connecting-usb-devices-to-wsl/

For some reason, when using the USB port in WSL2, access will be denied unless you run the command (miniterm or make flash) as sudo.