CS397/497 Wireless Protocols for the Internet of Things
Winter 2021

Lab 1 - Embedded Programming (VirtualBox)

Goals

- Get a build environment configured for the future labs and project
- Run C code on the nRF52840DK
- Simple debugging in an embedded context

Equipment

- Computer that you will use for labs
  - Needs at least 20 GB of space
  - USB ports
- Two nRF52840DK and cables

Lab Steps

1. Create a virtual machine (optional)

The most straightforward approach to getting a build environment configured is to install Ubuntu on a virtual machine. That will be a clean start to add packages too that also won't affect anything on your computer.

You are welcome to not follow this advice. I have a Macbook that I'm capable of programming boards on. I have many friends who use other distributions than Ubuntu. I also knew someone once who claimed he had Windows Subsystem for Linux working for programming boards. Good luck! Skip to step number 2.

Assuming you want to install an Ubuntu virtual machine:
Download the following:

- Virtualbox download link: https://www.virtualbox.org/wiki/Downloads
  - VirtualBox 6.x.xx platform packages (for whatever X is these days)
- Ubuntu download link: https://releases.ubuntu.com/20.04/
  - ubuntu-20.04.x-desktop-amd64.iso

Creating the machine:

- Open Virtualbox. Click "New"
- Type: "Linux", Version: "Ubuntu (64-bit)"
- Memory size: at least 4096, preferably 8192. I usually give a VM half of my machine's memory.
- Hard disk: keep hitting defaults. Make sure it is "Dynamically allocated". Size of virtual disk should be at least 100 GB (not the default of 10 GB, which is crazy small). Since it's dynamically allocated, it'll only actually use what it needs, but resizing the disk later is a huge pain.

- After creating it, you may want to change some settings (the “Settings” gear):
    - General/Advanced/Shared Clipboard: bidirectional
    - System/Processor/Processors: half of what’s available for your system (usually 4)
    - Settings/Display: Video Memory increase to 128 MB and enable 3D Acceleration
- If you’re on MacOS
    - You may have to “Allow” VirtualBox inside “System Preferences”/”Security & Privacy”. This actually required a restart on my Macbook to get working.

Start the virtual machine (the “Start” arrow):

- Start-up disk should be the Ubuntu iso that you downloaded
- Click “Install Ubuntu” once that finally loads.
- Choose “Minimal installation” (unless you want OpenOffice, games, etc.)
- The default “Erase disk and Install Ubuntu” is correct. Click “Install now” (Don’t worry, this will only erase the virtual disk you created for the VM)
- Choose “Continue” on the pop-up warning you about disk sectors
- Choose “Log in automatically” when creating your account to make your life easier
- It’ll take a few minutes to do the installation
- You’ll eventually get to a screen that says “Please remove the installation medium, then press ENTER:”. Just hit enter.
- Click next through a bunch of setup windows.
- You should now have your own Ubuntu machine!

Update and install guest additions (which makes the VM resize and stuff):

- CTRL+ALT+T to open a terminal
- sudo apt update
- sudo apt upgrade
    - This will take a while
    - While it’s going, open Settings/Privacy/Screen Lock and disable/set-to-never everything on that page
- sudo apt install build-essential dkms linux-headers-$(uname -r)
    - Unfortunately you won’t be able to copy-paste yet. That and window resizing is what we’re fixing right now.
- In the virtualbox menubar, go to Devices/Insert Guest Additions…
- A pop-up should say “blah blah Would you like to run it?” Click Run and then type your password
    - That will eventually say “Press Return to close this window…” when it’s done
- sudo reboot
- This should now allow the window to be resized
    - I had problems where the screen would go black when it got too big. I powered off the VM, went to Settings/Display and increased Video Memory to 128 MB and enabled 3D Acceleration, which fixed it. (And I then added those instructions above, so hopefully you won’t have the problem.)

2. Install requirements

If you’re using your own system, a bunch of these will be redundant, but won’t hurt. I’ll show the ubuntu package name and sometimes MacOS instructions. Translate to whatever your own system is.

- sudo apt install build-essential python3 python3-pip git vim emacs
- Install the gcc cross compiler for ARM microcontrollers
    - For the Linux version, you have to do this super manually. The following is one long line of code you can copy-paste into terminal to do it. (Ctrl-Shift-V)

        -
          ```
          cd /tmp  && wget -c https://developer.arm.com/-/media/Files/downloads/gnu-rm/9-2020q2/gcc-arm-none-eabi-9-2020-q2-update-x86_64-linux.tar.bz2 && tar xjf gcc-arm-none-eabi-9-2020-q2-update-x86_64-linux.tar.bz2 && sudo mv gcc-arm-none-eabi-9-2020-q2-update /opt/gcc-arm-none-eabi-9-2020-q2-update && rm gcc-arm-none-eabi-9-2020-q2-update-x86_64-linux.tar.bz2 && sudo ln -s /opt/gcc-arm-none-eabi-9-2020-q2-update/bin/* /usr/local/bin/.
          ```
    - For the MacOS version:
        - `brew tap ArmMbed/homebrew-formulae`
        - `brew install arm-none-eabi-gcc`
    - To check if this works run: arm-none-eabi-gcc --version
      It should autocomplete, work, and return 9.3.1
- Install the Segger JLink tools: https://www.segger.com/downloads/jlink/#J-LinkSoftwareAndDocumentationPack (the “J-Link Software and Documentation Pack” is what you want)
    - For the Linux version:
        - Open that website in your VM. And find “J-Link Software and Documentation pack for Linux, DEB installer, 64-bit”. Click Download. Then accept terms and download software. Choose “Save File” which will put it in ~/Downloads/
        - sudo apt install ~/Downloads/<NAME OF DEB FILE HERE>

3. Download the repository

I’ve put together a git repo with some basic programs that can be loaded on the nRF52840DK. It’s based on nrf52x-base, which is a library I and my colleagues put together at Berkeley for programming our nRF52 based devices.

- The base repo for this class is located at https://github.com/brghena/nu-wirelessiot-base
  - You should probably fork your own copy of the repo
  - That lets you keep your code in version control and still allows you to pull in updates that I make to the original repo
  - Fork button is at the top right
- Clone the repo (or your forked repo) locally and cd into the repo
  - Make sure the directory path where you clone it doesn’t have any parent directories with special characters or spaces. The tools are sometimes brittle to that kind of thing.
- git submodule update --init --recursive
  - This will take a minute to run as it loads everything
  - The nrf52x-base submodule contains all the libraries needed for the nrf52840DK
  - You’ll need to run this command again when things in that submodule change
- Check that code builds
  - cd nu-wirelessiot-base/software/apps/blink/
  - make
    - Should build _build/blink_sdk16_blank.elf
- Check that JLink tools work
  - make flash
  - Should pop up a “J-Link V6.94 Emulation selection” window. Which you should click no to. This is what happens when you try to program a board, but no board is attached.
  - You’ll have to click No like four times. Sorry.
    - Or better, click in the terminal and just “Ctrl-C”

4. Program a board

- Check the nRF52840DK switches
  - “Power” (top-left) should be ON
  - “nRF Power Source” (middle) should be VDD
  - “nRF ONLY|DEFAULT” (bottom right) should be DEFAULT
- Remove the coin cell battery from the nRF52840DK
  - Not just the plastic tab, take the whole battery out
  - You can save it for later if you want to have the board run on batteries
- Plug the board into the computer
  - You plug into the USB on the top of the board (the one on the side is for connecting directly to the microcontroller if you have implemented USB on it)
- Attach the board to the VM

    - In the menubar, click Devices/USB/Segger-JLink (out of your USB devices)
    - If you hover over Devices/USB/ again, it should now have a check mark
    - You'll have to check this button each time you plug in a board. There will be a separate one for each board you have attached to the computer.
- In the blink app
    - make flash
    - It should pop up a window with a loading bar that uploads the code
    - Things like "Downloading file [_build/blink_sdk16_blank.hex]..." and "O.K." are good
    - Things like "J-Link connection not established yet but required for command" and "Connecting to J-Link via USB...FAILED: Failed to open DLL" are bad
    - Also, the board should start blinking LED1 if it works

5. Get some apps working

- There are four good starter apps:
    - blink - blinks LED1
    - button - uses four buttons to control four LEDs
    - printf - periodically prints a message from the board
    - error - demonstrates a hardfault and error messages on the board
- Commands to control them
    - make flash - build code and load it onto the board over JTAG
    - make rtt - open a debug window to view print statements
- Take a look at each of the starter apps and try out modifying board behavior

6. Create a new app

- You will need to make a new folder in apps/ and copy over the Makefile
- Steal some behavior from existing apps and combine them

Lab 1 Submission

1. Demonstrate that you can successfully program a board
    - If you're using an interesting non-standard setup, let me know
2. Demonstrate that you've created a new application