COL788: Advanced Topics in Embedded Computing

Semester I, 2022-2023

Assignment-1: Cross-Compilation

August 22, 2022

Objective

You know what the compilation is! Simply put, it is the process a computer performs to convert a high-level programming language into the machine language that the computer can understand. Cross-compiling involves building a binary on one platform that can run on another platform, e.g. we can use an x86 machine to compile a program for an ARM machine.

In this assignment, you will learn to build executable for a Raspberry Pi board with an ARM processor while working on a Linux system with an x86 processor (if you have any other OS, you can install Linux in the virtual environment).

Background

Cross-Compile Toolchain

A toolchain is a set of distinct software development tools that are linked (or chained) together by specific stages such as GCC, Binutils, and Glibc (a portion of the GNU Toolchain). The cross-compilation involves three machines.

- the build machine on which the toolchain is built (you will be using a ready-made one);
- the host machine on which the toolchain is executed; and
- the target machine for which the toolchain generates code.

The following are two ARM cross-compile toolchains.

- Yocto
  https://docs.yoctoproject.org/
- Buildroot
  https://buildroot.org/

Raspberry Pi Emulator

The best way to learn about an embedded system is to work on it. However, working directly on an unfamiliar system can be daunting. Therefore, in this assignment, you will use an emulator to learn helpful skills before working on a real board later in this course. In this assignment, we will use Quick EMUlator (QEMU) which emulates ARM chipsets, such as the one found in the Raspberry Pi. You can look into the following tutorials to setup QEMU.

- https://www.how2shout.com/linux/how-to-install-qemu-kvm-and-virt-manager-gui-on-ubuntu-20-04-lts/
- https://linuxconfig.org/how-to-run-the-raspberry-pi-os-in-a-virtual-machine-with-qemu-and-kvm

Cyclic Redundancy Check (CRC)

When you send a message from one network node to another, you want to ensure that the message is communicated correctly. Including a Cyclic Redundancy Check (CRC) code along with the message is one way to facilitate this. An M-bit long CRC is calculated using a carefully-selected polynomial of degree M, called the generator polynomial. For instance, for computing a 16-bit CRC, the suggested generator polynomial is $x^{16} + x^{12} + x^5 + 1$. The generator polynomial can also be represented as a bit-string denoting the presence/absence of the different exponents, e.g., 10001000000100001 in the previous example.

Tasks

1. Download and install an ARM cross-compile toolchain for ARM/MIPS32. You can select either Yocto or Buildroot as the toolchain.
2. Install QEMU to run ARM/MIPS32 program.
3. Write a simple program in the C programming language, which computes and validates the 16-bit CRCs for 64-bit messages. You can use the generator polynomial as $x^{16} + x^{12} + x^5 + 1$.
    - In the sender mode, your program should take a set of 8 ASCII characters (say, any letters or numbers) as the input, create the corresponding 64-bit message and then output the 16-bit CRC.
    - In the receiver mode, your program should take two inputs, the message and the CRC, and output whether the message has been received correctly.
    - Your program should contain two test cases where you introduce some bit errors in the message/CRC and the errors are detected.
    - Your program should also have two test cases where you introduce some bit errors in the message/CRC but they are not detected.

Submission

You should submit the following on Gradescope.

- `source_code`: This should contain the code that you cross-compiled.
- `binary_file`: This should contain the cross-compiled binary file.
- `report`: This should be the pdf file containing all the necessary details about your work. For instance, it should explain the steps to build and execute your code. It should have screenshots of terminals to demonstrate that your code works as desired. You should also discuss why some errors are detected and others are not detected in the different test cases used in your code.

Grading

We will schedule a demo session where you can demonstrate the functionality of your code. The final grading rubrics will be published on Gradescope.

Late Submission Policy

The late submission penalty will be 25% for submitting within one day after the submission deadline. It will be 50% for submitting after the first day but before the second day beyond the deadline. We will not accept any submission after two days beyond the submission deadline.