UOIT
CHALLENGE INNOVATE CONNECT

SOFE 4610U:

Internet of Things

Lab #2: Cross Compile and Python GPIO Introduction

Objectives

- Introduce students to cross-compiling on different platforms
- Gain experience and understanding of system environment variables
- Become familiar with compiler techniques

Introduction

A toolchain is a set of programming tools used to produce native code for a specific computer architecture. This includes the compiler, the linker, the C libraries and any utilities used for analyzing the binary formats.

A cross-compiler is a toolchain that produces machine code that runs on a different microprocessor architecture from the one that the code was compiled on. In this lab, we will use the **arm-gcc-Linaro** toolchain to compile code on an X86-64 Linux machine for an ARM target platform single board computer.

The host machine is the computer where the cross-compiler toolchain is run. The target machine is the Raspberry Pi computer, the computer for which we are going to produce code. The Raspberry Pi Version B will be used as the ARM target.

Why cross compiling is important

In theory, a developer who wants to build programs for some device could get the appropriate target hardware (or emulator), and compile natively within that environment. [1] However, our real world is not as ideal as it could be. The target device usually is slower than the host, and also it mostly likely doesn't have as much memory or disk space. In general, cross compiling has given developers speed, capability, availability, flexibility and convenience.

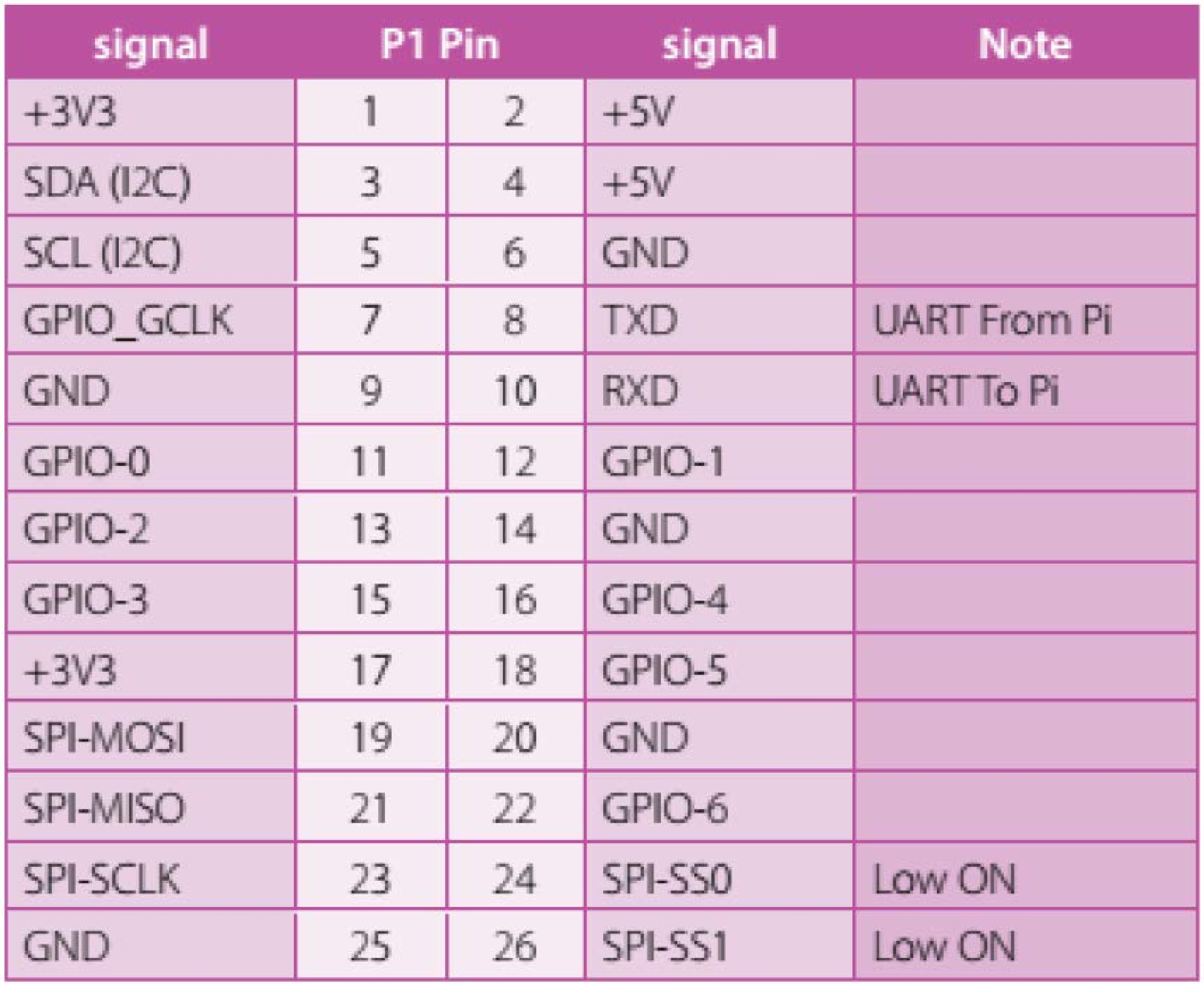
Prelab

Answer the following questions before performing lab #2.

- What microprocessor does the RPi use?
- What is the kernel version of the Ubuntu computer you are using?
- Find the same information about the ARM Linux board you'll be using.

Python GPIO

Python uses RPI.GPIO library to control general purpose input and output. The image given in this lab has built-in python GPIO(general purpose input output) libraries.



signal	P1 Pin		signal	Note
+3V3	1	2	+5V	
SDA (I2C)	3	4	+5V	
SCL (I2C)	5	6	GND	
GPIO_GCLK	7	8	TXD	UART From Pi
GND	9	10	RXD	UART To Pi
GPIO-0	11	12	GPIO-1	
GPIO-2	13	14	GND	
GPIO-3	15	16	GPIO-4	
+3V3	17	18	GPIO-5	
SPI-MOSI	19	20	GND	
SPI-MISO	21	22	GPIO-6	
SPI-SCLK	23	24	SPI-SS0	Low ON
GND	25	26	SPI-SS1	Low ON

Fig.1 Raspberry Pi GPIO Pins

As in the given lab kits, a Sensorian shield is included in the package. Therefore, you can use the shield to control the on-board LED. In this case, GPIO pin 12 is used in this lab.

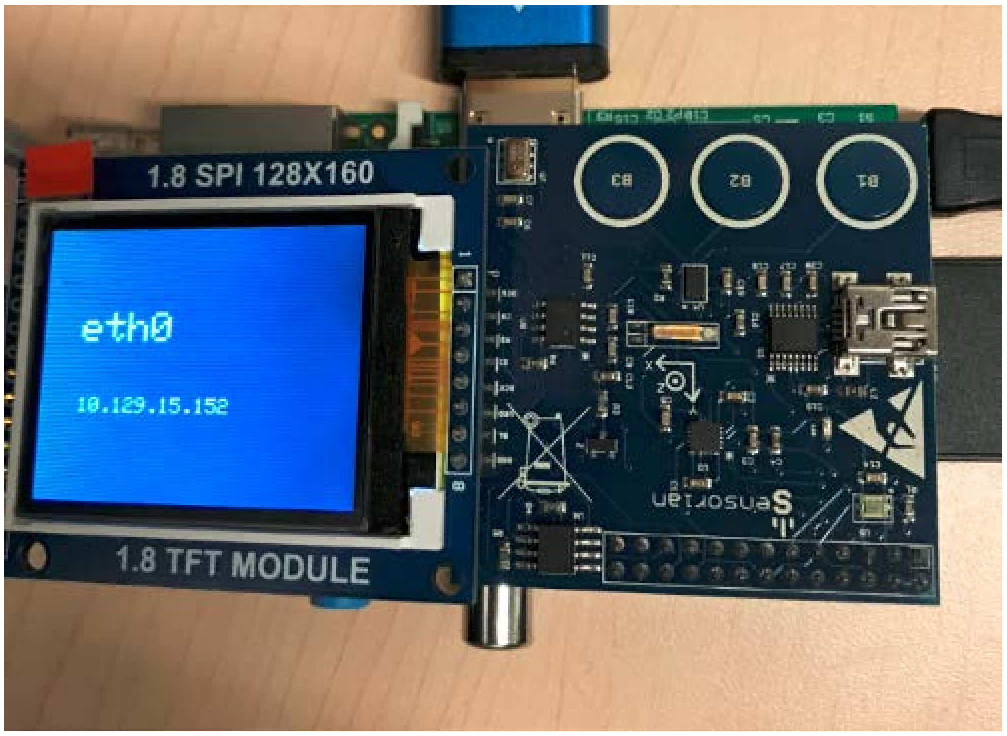


Fig.2 Sensorian Shield with Raspberry Pi

Lab Activity

Lab Tasks

1. Before we are ready to cross-compile any programs a toolchain has to be installed and configured. There are two ways to properly install a toolchain. The first method requires compiling the GCC compiler from scratch. This method is more involved however it allows users to optimize any component of the toolchain. The downside to such a method is that it requires a relatively long time to build the compiler from source, therefore we will skip this method for the simplicity of this lab.

The other method is to use a pre-compiled toolchain and configure it such that one can use it either with a build system or with any IDE. This is the method that we are going to pursue here. Follow the instructions below to set up the cross-compiler on your Ubuntu virtual machine.

Log in your Ubuntu host station or VMWare on Windows host, and create a work folder on the Ubuntu host station named rpi.

Take note of the working directory path since you are going to need it below.

Clone the cross-compiler repo from the Raspberry Pi GitHub account at:
https://github.com/raspberrypi/tools/
by running the command ***git clone https://github.com/raspberrypi/tools/***

2. Unzip the tools folder.
This concludes the installation of the compiler. Under the tools/arm-bcm2708 you'll find 4 different cross-compilers. We will use the 64bit host version of the compiler located under /tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin since the Ubuntu host is a 64-bit architecture.

The compiler is not installed as a program however so we have to let the host OS know about its location. Use `printenv` to find the current PATH variables. Now add the cross-compiler to your path permanently by modifying the environment variables. You will need to edit the `.bash_profile or .bashrc` file and append to it the path of the compiler.

3. Now you can write a HelloWorld program in C and then cross-compile it. Write the program and note down which command you used to compile the program. Try running the program on the host station. What message did you obtain and what does this mean.

The terminal command that you used above works well for simple programs. For multi file programs a build system is needed. On Linux a number of build systems are used such as CMake or Makefiles.

4. Complete the given Makefile so that it uses the cross compiler you configured above to compile the Hello World program that you wrote.

5. Once you have compiled the code, you will have to transfer the executable file to Raspberry Pi. (Note scp or Winscp can be used to transfer files between host and Raspberry Pi)

6. Write the python script to turn on the LED on Sensorian Shield on and off 1 second apart by modifying the given python scripts. (Note, when you run the python script, make sure that you run the code with as root!)

Deliverables

Complete all lab tasks, answer all the questions in lab task. Please note, all lab reports will have title pages, introduction, content of the lab, tasks, and conclusion. In the content of the lab, write the detailed step by step method explaining; how you set *PATH* in the Linux environment, compiling files, and transferring the executable file to the Raspberry Pi.

Submit the Python script with all comments indicating the functionality of all major parts of the code.

Reference

[1] https://landley.net/writing/docs/cross-compiling.html