Lab Work 1 - Tools and Platform

The practical side of this course naturally requires a lot of programming work. We are going to write codes mainly using C programming language. Some assembly will be shown to demonstrate a few examples but students are not required to master them. The recommended development platform is Linux, but the module will show how it can be done on Windows (since this is what most students are familiar with).

Development Board

The development board used in this course will be the Raspberry Pi - a credit-card sized single-board computer developed by the Raspberry Pi Foundation (UK). The board is based on Broadcom's BCM2835 (ARM-based) Applications Processor. Information on the peripherals available on this particular microcontroller can be obtained from the official Raspberry Pi site (here). A local copy for PGT302 students is available here. The Embedded Linux wiki page provides a good documentation for this board (e.g. BCM2835 framebuffer info can be found here).

Introduction

There have been a few versions of the board since it first came out in 2012 (pre-launched the year before). The ones available in our lab are the Model B+ Version 1. Here are some general specifications for this particular board:

- A System on Chip BCM2835 package (with CPU & GPU) stacked on an SDRAM package
- CPU is 700 MHz ARM1176JZF-S core (ARM11 family, ARMv6 instruction set)
- GPU is 250 MHz Broadcom VideoCore IV (supports OpenGL, with video/audio? codec)
- 512MB SDRAM shared by both GPU and CPU
- 40-pin GPIO header connector (27-GPIO pins)
- Powered by mini-USB connector (or GPIO header) requiring up to 600mA (3W)
- 1 ethernet port, 4 USB ports (on-board 5-port USB hub)
- HDMI video/audio output, separate 3.5mm audio jack

Each board optionally comes with a NOOBS (New Out Of the Box Software) micro-SD card. This is basically an OS installer (mostly Linux-based) which can be used to setup the micro-SD card for various applications programming tasks.

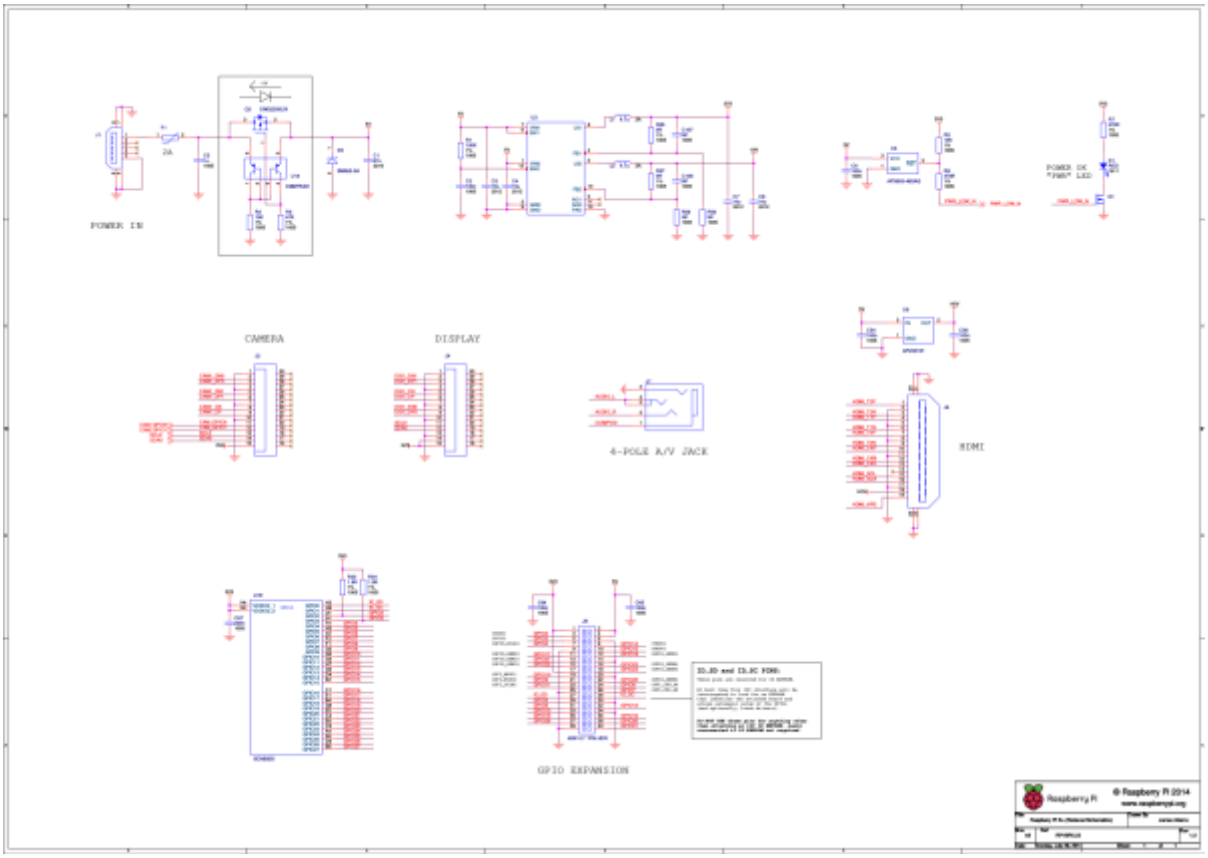
The recommended/official OS is Raspbian (a Debian-based distribution tailored for Raspberry Pi). Other options are considered third-party images, which can be obtained here.

The nice thing about this board is it has an HDMI output (which is becoming a norm for flat panel display unit). Connect a USB keyboard and a USB mouse, you get a PC running Debian 😎

Note: We will only use Linux-based OS towards the end of this course.

DevInfo: Board Schematic

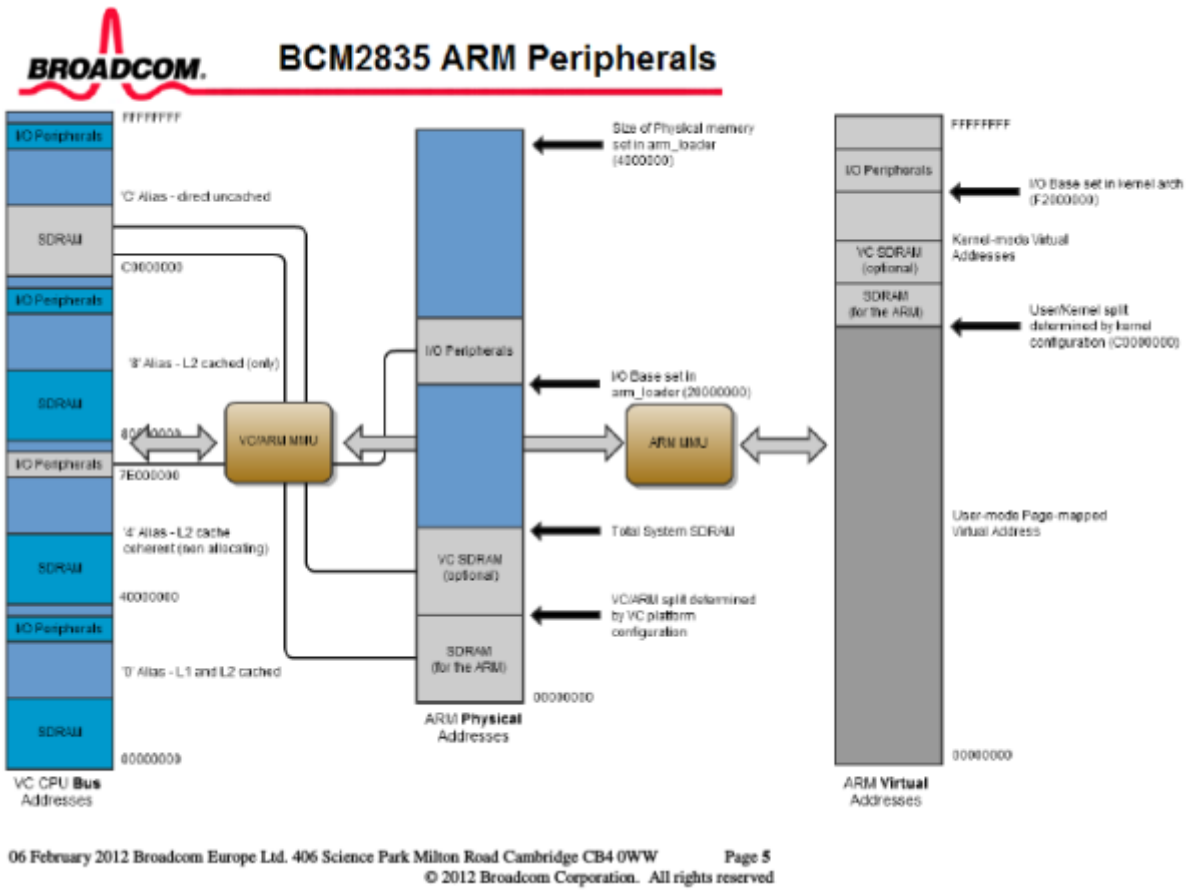
This is the schematic for Raspberry Pi Model B V1.2+.



DevInfo: Peripherals Address Map

The information here is for (somewhat) low-level programming, that is if you are interested in accessing the hardware from your software code. If you are only interested in developing applications for Linux running on the Raspberry Pi, you most probably do not need this.

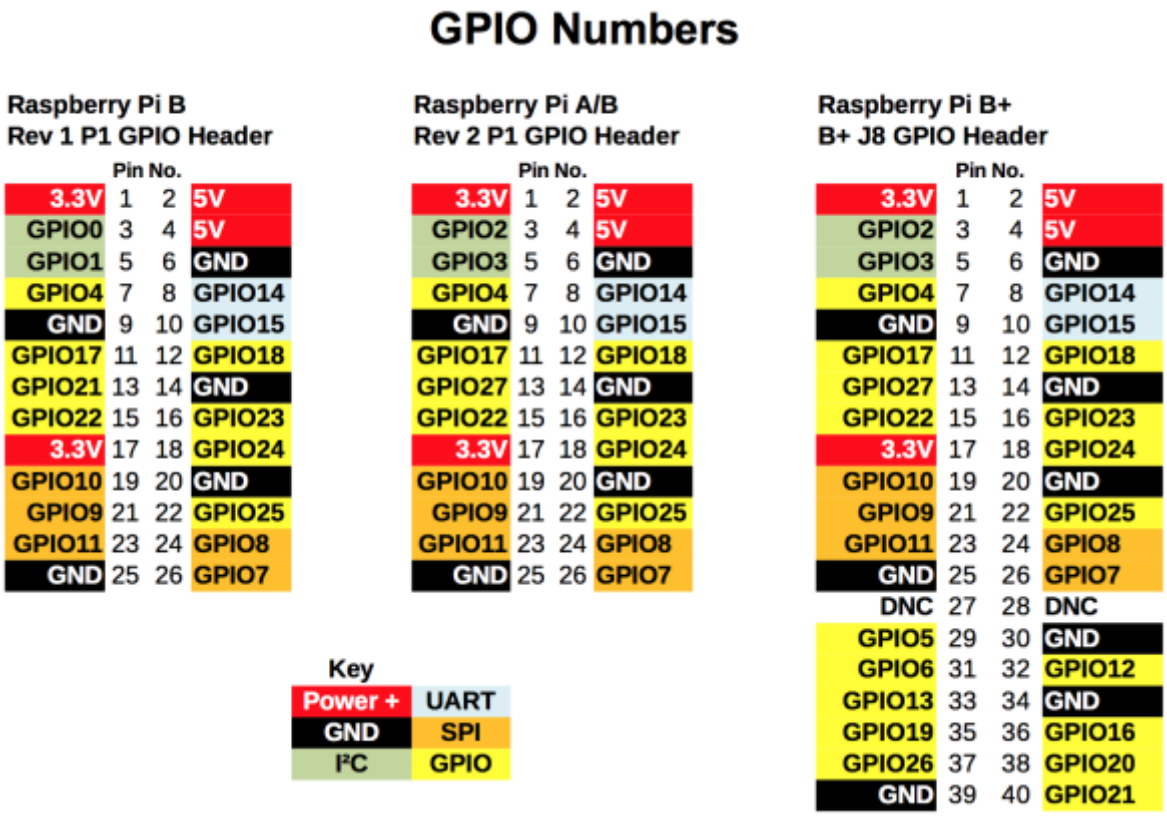
As with any development board, the first things you would like to know before you start programming is how to access (address?) the peripherals. The figure below (taken from BCM2835 ARM Peripherals documentation) shows an overview of the systems peripheral addressing information.



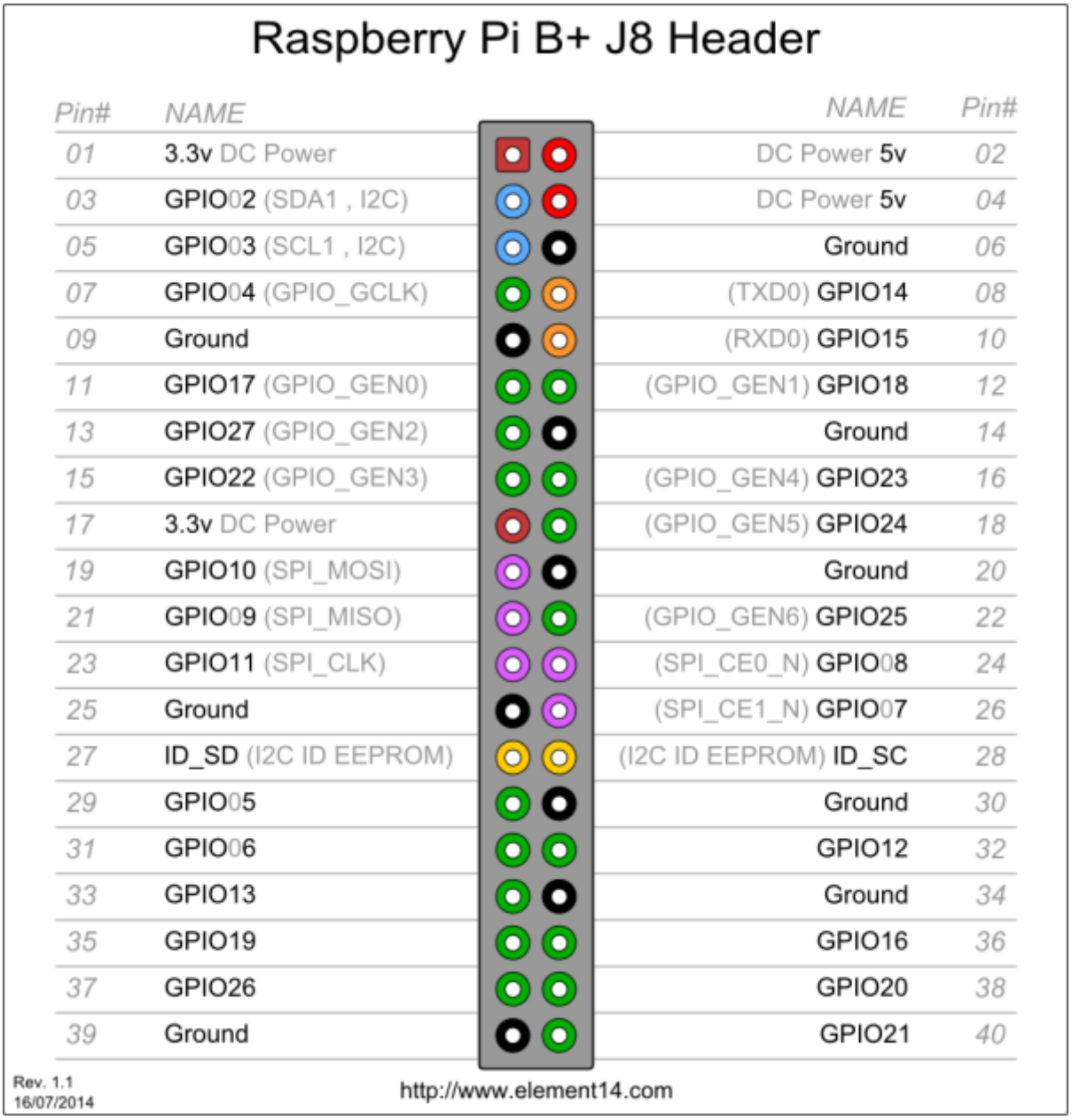
The thing to look for here is the I/O Peripherals (which includes the GPIO pins) and notice that there are three information entries - 0xF2000000 on the ARM Virtual Address, 0x20000000 on the ARM Physical Address (noted as set by arm_loader) and 0x7E000000 on the Video Core CPU Bus Address. You need these information to access the I/O peripherals.

DevInfo: GPIO Header Information

This is the information you need to connect the Pi to various peripherals



You can find the GPIO that you need to use on the respective headers (in our case, it is the 40-pin header).



***Note**: I got these images from the internet. Credits due to the original owner(s).*

Preparing SD Card

***Note**: As mentioned earlier, we will be using bare-metal programming at first, and then, progress towards using Raspbian.*

***Note**: The term SD card mentioned here generally covers/means the microSD card.*

This is a guide to prepare such card from scratch. If you already have the one that comes along with the board (NOOBS card, is it?), this procedure will override and delete existing content. Backup or forever hold your peace! Well, if you cannot do that and you want to have NOOBS back, head on to this page and download the latest NOOBS image.

Bare-metal Programming

Note: *Your instructor will show you how to partition/format the SD card.*

To exclusively work on bare-metal programming (no OS):

- Format the card with at least one FAT32 partition
  - Linux platform - use *fdisk* for this (usually need root... unless your user have permission)
  - Windows platform - use *diskpart* (run as administrator)... OR, maybe *this*? (The link is for portable version.. there is also an installer if that is your cup of coffee 😎)
- Get the required firmware from here - we only need 2 files from boot folder (bootcode.bin and start.elf). For the sake of being 'politically correct', I also copied LICENSE.broadcom file. OR, get my personal copy here (which is what I usually pass to my students on a USB drive).
- Place those 3 files in the root of the previously FAT32-formatted SD card
- Along with your kernel.img code, you should be able to take control of the Pi

A nice config for multiple kernels - this is what I use to have multiple kernel (my1barepi codes) images on my SD card

config-my1barepi.txt

```
# kernel is the alternative kernel filename
# - [Pi 1, Pi Zero, and Compute Module] kernel.img
# - [Pi 2, Pi 3, and Compute Module 3] kernel7.img
# - [Pi4] kernel7l.img.
#kernel=kernel_video_temp.img
#kernel=kernel_sdcard.img
kernel=kernel_pick_one.img
```

Installing Raspbian OS

Update20210907 *The official OS is now known as Raspberry Pi OS (instead of Raspbian). Check out this official page. I will try to update this site A.S.A.P. - but, do not hold your breath* 😬

We will be using Raspbian (the official Linux distribution for Raspberry Pi). This enables us to run web servers and other network-related stuffs.

[201804011654] *Note: I just noticed there is now an option to use Windows10 IOT Core (which is prepared by Microsoft as a third party option), but I will not be using that here. Checkout this page for other options*

- Download system image
  - Official Raspbian (latest)
    - This is a ZIP file containing an image file - e.g. 2018-11-13-raspbian-stretch.zip
  - For PGT302 students, get one with PGT302-specific customization

        - Ask your instructor
- Extract (unzip) the image (file with *.img extension)
    - e.g. 2018-03-13-raspbian-stretch.img
    - currently, at least 8GB SD card is required...
- Write the image to SD card
    - Insert the SD card to your SD card reader
    - Linux Platform - use *dd* for this
        - assuming the device is at /dev/sdb
        - use `dd if=2018-03-13-raspbian-stretch.img of=/dev/sdb` and wait...
    - Windows Platform - I recommend Win32DiskImager (I got the Win32DiskImager-1.0.0-binary.zip file)
        - find (and VERIFY) drive letter for your SD card
        - select image, start write and wait...

Raspbian-ready Plus Bare-metal Programming

To have Raspbian-ready card as well as working on bare-metal programming:

- Follow normal instruction for installing Raspbian OS

Note: *Do the following procedure on your PC.*

To use bare-metal code:

- Rename kernel.img and config.txt to avoid being used
    - e.g. rename kernel.img → kernel-raspbian.img
    - e.g. rename config.txt → config-raspbian.txt
- Make sure there is no config.tx file
- Simply copy your own compiled kernel.img to the FAT32 partition

To get Raspbian back running:

- Get the saved files appropriately named:
    - e.g. copy kernel-raspbian.img → kernel.img
    - e.g. copy config-raspbian.txt → config.txt

Raspberry Pi Zero as USB Client

Pi Zero has a USB On-the-Go (OTG) hub - which, basically means that it can be both host (like USB hub on a PC) AND client (like USB hub on Android or most gadgets these days). So, to setup Pi Zero as a client (this is done on a PC - while preparing the card),

1. Follow normal instruction for installing Raspbian OS
2. Edit `config.txt` (on boot partition) and insert `dtoverlay=dwc2` line
3. Edit `cmdline.txt` and insert `modules-load=dwc2,g_ether` kernel parameter
    - also, insert `g_ether.host_addr=<mac_addr>` to get a fixed MAC address (easier to manage!)
4. Add empty file called `ssh` (same location as `config.txt`) - this will enable ssh

Connect Pi Zero to a PC (allow some time for it to finish booting) and it should appear as ethernet device. To connect to it,

- using network manager:
  - config ipv4 as link-local only
  - config ipv6 as ignore
- use ssh to connect
  - ssh pi@raspberrypi.local
    - default password: raspberry
  - use ssh-copy-id to use key-based auth

To share the internet with the Pi

- find the IP address on usb0

```
$ ifconfig usb0
```

  - we need this later on Pi
- allow IP forwarding on the host

```
echo 1 > /proc/sys/net/ipv4/ip_forward
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

  - to remove this later

```
echo 0 > /proc/sys/net/ipv4/ip_forward
iptables -t nat -D POSTROUTING -o eth0 -j MASQUERADE
```

  - *note for wifi internet, change `eth0` to `wlan0`*
- ssh into pi
- (OPTIONAL) add a name server (e.g. nameserver <ip-add-of-usb0>) to /etc/resolv.conf
  - or, just use 8.8.8.8
- set the default gateway on Pi to host PC's IP

```
# route add default gw <ip-add-of-usb0>
```

Have fun!

Note:*I have tested this using Raspbian Lite on Pi Zero and it worked! (Obviously)*

Advanced Configurations

Using the above setup should get simple projects going without any problems. However, some things on the BCM2835 require special configurations on the GPU side. This can be changed by having a `config.txt` file in the same path as the files above.

Example of the configuration file:

config.txt

```
# --------------
# MEMORY OPTIONS
# --------------

# specify gpu memory allocation
# - min 16, max 192 (256), 448(512), 944 (1024)
# - default 64
#gpu_mem=64

# disables CPU access to GPU L2 cache
# - default 0 (enabled)
#disable_l2cache=1

# --------------
# KERNEL OPTIONS
# --------------

# specify kernel name
# - kernel7.img default for pi2/pi3
# - kernel8.img preferred on pi3 (for 64b mode)
# - common default is kernel.img?
#kernel=kernel.img

# specify startup address for ARM kernel
# - default 32b: 0x8000
# - default 64b: 0x80000
# - kernel_old=1 option overrides to 0?
#kernel_address=0x8000
#kernel_old=1

# ----------------
# ADVANCED OPTIONS
# ----------------

# camera needs start_x.elf firmware
# - or, start_file=start_x.elf, fixup_file=fixup_x.dat
#start_x=1

# prevent red camera led to turn on while camera is active
# - default 0 (enabled)
#disable_camera_led=1
```

To activate an option, simply remove the '#' character for the beginning of the option line (uncomment). More information on `config.txt` can be found here. We can also have conditional filters.

Other custom configuration(s):

config.txt

```
# for 5-inch lcd with touchscreen
max_usb_current=1
hdmi_group=2
hdmi_mode=87
hdmi_cvt 800 480 60 6 0 0 0
dtoverlay=ads7846,cs=1,penirq=25,penirq_pull=2,speed=50000,keep_vref_on
=0,swapxy=0,pmax=255,xohms=150,xmin=200,xmax=3900,ymin=200,ymax=3900
#display_rotate=0
```

General Issues

MicroSD Card Failed Boot

Sometimes, the card simply cannot boot. Use fdisk to check/create 255 heads, 63 sectors and calculate the required cylinders based on

```
disk_size = cylinders * head * sector * sector_size
```

Not sure why this happens... maybe BIOS issue when formatting on PC? Or, maybe my students' laptops are infected with virus?

Raspbian Update Error

- I used the 2017-09-07-raspbian-stretch.img and got an error while trying to update
  - problems seem to be with storage space (98% usage)
  - turns out the partition for root fs was only 4.9GB
- to resize the partition, use fidsk
  - assume card is /dev/sdb (i use USB card reader)
  - `fdisk /dev/sdb`
  - delete partition 2 and recreate (make sure use the same start sector!)
  - save and exit
  - clean the fs `e2fsck -f /dev/sdb2`
  - resize `resize2fs /dev/sdb2`

2026/02/09 22:23

Development Tool

We usually have a more powerful machine on our desktop compared to our target embedded platform. It is therefore more convenient to compile the target embedded software on our desktop (another reason would be simply because our platform is NOT capable of compiling its own program). This process usually requires us to have/build cross-compilers - compiler that runs on a host machine, but produces binary/executable for a target machine. You can read about my experience on cross-compilation here.

A simple script to build a cross-compiler for Pi is available in my1ubuild project. The project is actually a repository of build scripts for various tools. The one we need here is arm-gcc.build.

Note: Video guides are available on YouTube

Linux Platform

You are encouraged (...required, actually) to try to build your own cross-compiler using the script mentioned above. If you have problems, you can always ask your instructor.

Windows Platform

You can either try to build your own cross-compiler on Windows using the script mentioned above, OR you can simply get the one I have already built for you.

Building a Cross-compiler

Note20211027 *Notes on setting up latest MinGW environment is available here.*

We need to have a working compiler. I prefer MinGW (Minimalist GNU for Windows), so that is what I'll use to demonstrate. Download the mingw-get installer from here. At the moment, the latest version is 0.6.2-beta-20131004-1, so I download `mingw-get-0.6.2-mingw32-beta-20131004-1-bin.zip`.

I want my compiler in `C:\Users\Public\Tool\mingw`, so I created that folder and extract the contents into that folder. Open up a command prompt, change path

```
cd C:\Users\Public\Tool\mingw\bin
```

and run the following:

- `mingw-get install “gcc=5.3.*”`
- `mingw-get install “g++=5.3.*”`
- `mingw-get install gmp`
- `mingw-get install mpfr`
- `mingw-get install mpc`
- `mingw-get install msys`
- `mingw-get install msys-wget-bin`

Edit `C:\Users\Public\Tool\mingw\msys\1.0\etc\profile` and comment the last line which is a `cd $HOME` command (insert a # at the beginning of the line).

To make things easier for you, I have written a Windows script to create a link in the Explorer popup menu. Copy this script into `C:\Users\Public\Tool\mingw`. Running this script (*Hint*: double-click) will install (or remove if already installed) a pop-up context menu entry (`MinGW Shell`) whenever you right-click on a folder in Windows Explorer.

Get arm-gcc.build and place it in `C:\Users\Public\Tool\my1ubuild`. Open a MinGW Shell at this location and run

```
TOOL_PATH=/c/users/public/tool sh arm-gcc.build
```

The script will download the required `binutils` and `gcc` tarballs and build them. After a while, you should get your cross compiler at `C:\Users\Public\Tool\xtool-arm`.

Pre-built Cross-compiler

Or, if you just want to get on with your work, here are what you need to download:

- MinGW (MD5:15e5e4d1f4b5a12f5c16926567b629fb) - Minimalist GNU for Windows utilities, which includes msys that provides a make environment. This is a local copy for your convenience. If you want, you can install your own from 'original' source using mingw-get (this is exactly what I used to create that zip file for you to download).
- ARM Cross-Compiler (MD5:876a2fb45779ae726ce07c4f78e63101) - The compiler that you need to compile your bare-metal codes. Either build one on your own as mentioned earlier, or simply use this.

Note that these binaries were built using Windows 7 running in VirtualBox on my Slackware 14.1 installation. But I have tested it on Windows 8 and Windows 10 machines, and I have no problems so far.

Setting-up the cross compiler:

1. Once you have downloaded the files, extract them to *C:\Users\Public\Tool* (so, you should have an *xtool-arm* folder there with a *bin* sub-directory). The location *C:\Users\Public\Tool* is referred to as TOOLPATH in this course), you should have the MinGW compiler in %TOOLPATH%\mingw and the ARM cross-compiler in %TOOLPATH%\xtool-arm.
2. It would be handy to have a shortcut on the Explorer popup menu to open up a MinGW shell in the folder being viewed. Double-click `msys.vbs` (should be in %TOOLPATH%\mingw) to toggle this feature. For the script to actually open up in a particular folder, browse to mingw folder and edit msys\1.0\etc\profile and comment (insert '#' character at the beginning of) the last line (should be `cd $HOME`).
    - if you get a popup message saying `Windows Script Host Access is disabled on this machine`, you will have to use registry editor to fix that. Type `regedit` in a command prompt (you must be ad adminstrator for this... by default, the first account IS). Look for `HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows Script Host\Settings` - and make sure the `Enabled` value is set to 1. (You may also need to change one in `HKEY_LOCAL_MACHINE`)
3. To start working on a project, open your project folder in Windows Explorer and right-click to get the usual popup menu. You should now have a 'MinGW Shell' link. It should open up a terminal (shell) window in that folder. This is where we will do most of the compilation work at the early stages (bare-metal programming).

Code Editor

As mentioned in your programming course, you can use any text editor to write your program (even Windows Notepad!), but it is nice (and useful) to have at least syntax highlighting feature - a 'proper' code editor. For this, I would like to recommend Geany, a GTK-based cross-platform text editor. I use this a lot when coding on Linux. You can download a Windows installer here. It can be installed as

normal user (no admin access required!).

Source Code Management

Most coding projects tend to use source code management (SCM) software, and I recommend git (I have my own notes on this). It is also available on Windows.

Bare-metal Library Code

I have one available - my1barepi.

From:
https://azman.my1matrix.cc/ - **Azman @MY1**

Permanent link:
https://azman.my1matrix.cc/doku.php?id=archive:pgt302lab00&rev=1770522381

Last update: **2026/02/08 03:46**