EC-ENG 231 (Spring 2024)

Beaglebone Programming:

Setup & General Purpose I/O (GPIO)

Fatima Anwar

fanwar@umass.edu

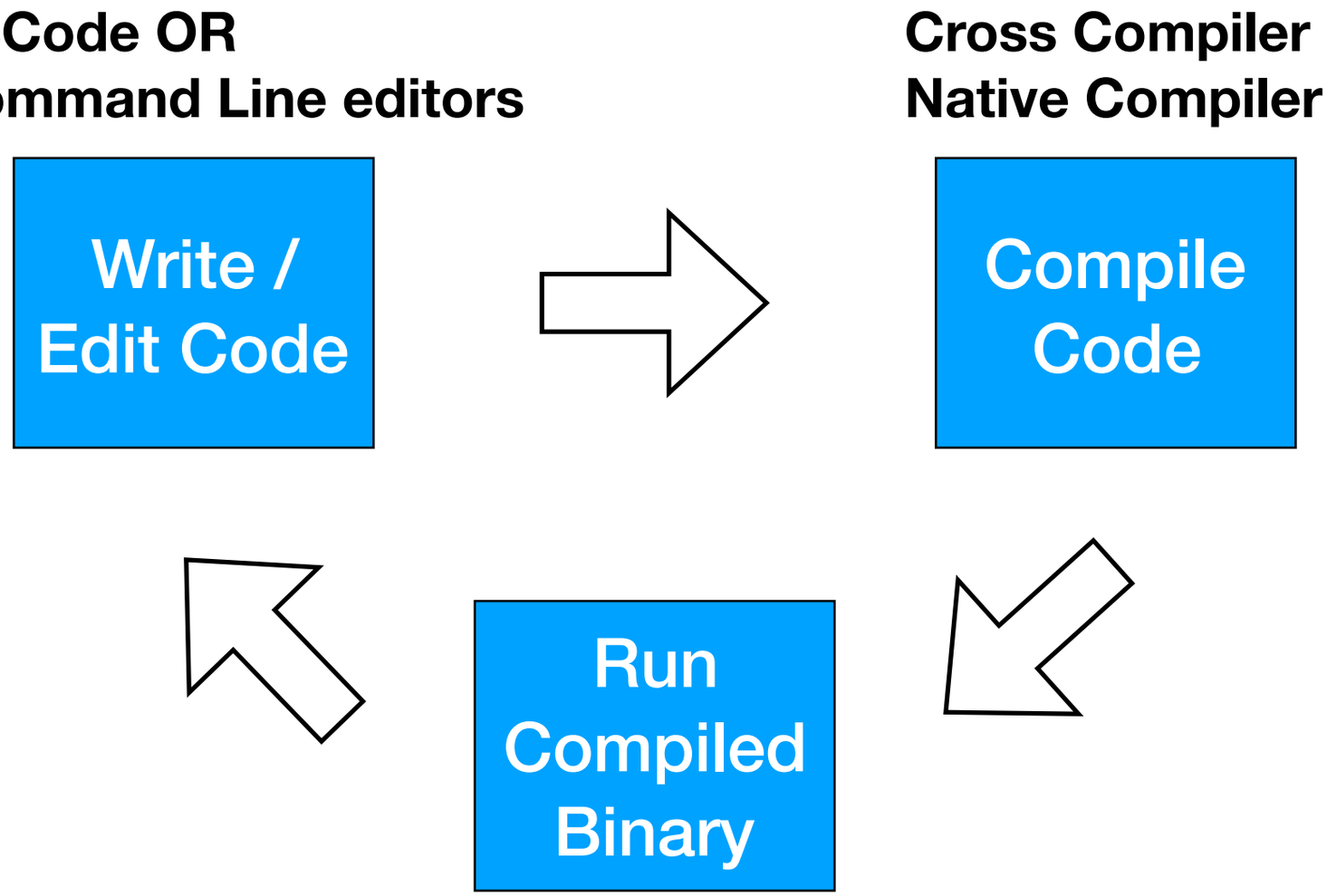
UMass Amherst

First Half Schedule

Date	Topic	Lab Assignment
Feb 7	C Programming	
Feb 12	C Programming	**Lab 1**: C program - Due: Feb 21* (5)
Feb 14	Embedded Linux	
Feb 21	Embedded Linux (cont.) + BBB basics	**Lab 2**: Linux & BBB - Due: Feb 29 * (5)
Feb 22	BBB and GPIO Programming	
Feb 26	GPIO Programming continued	
Feb 28	S/W Life Cycle: interrupts	**Lab 3**: GPIO - Due: Mar 7 * (5)
Mar 4	Multithreading, processes	
Mar 6	Sample code (lets do it together)	**Lab 4**: Interrupt & Threads - Due: Mar 14 * (5)
Mar 11	Clocks and Time	
Mar 13	Event Programming	**Lab 5**: Time & Events - Due: Mar 31 * (10)
Mar 25	Review Day	
Mar 27	Midterm Exam	

** For every day of late submission, 10% of your lab grade will be deducted*

Software Lifecycle

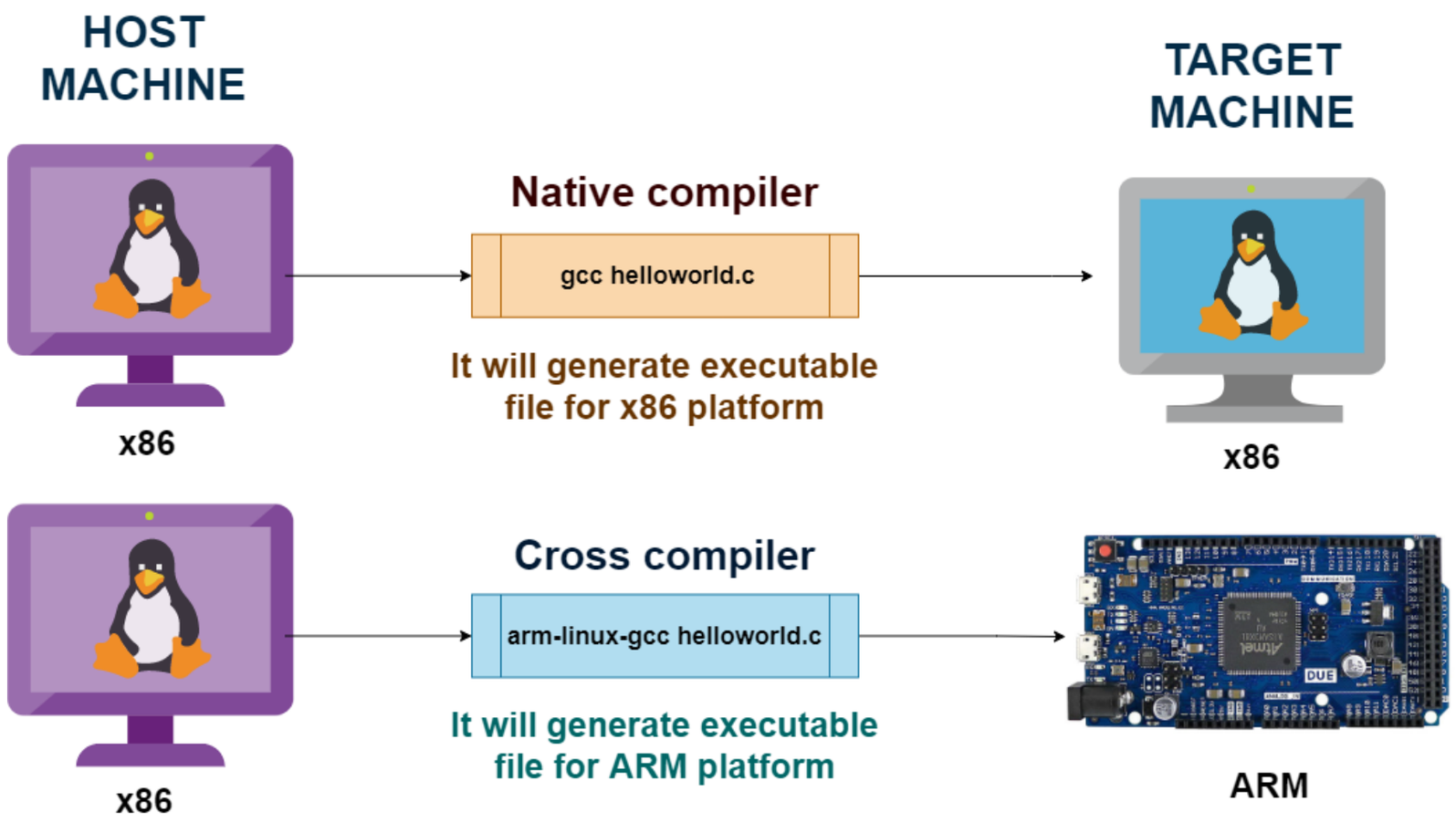


Key Differences in Embedded and Standard Software

	Embedded Software	Standard Sodtware
OS	Full OS,RTOS or no OS	Always an OS
Development / Compilation	Cross-Platformn	Native platformn
Hardware- Agnostic	No	Yes
Complexity	Simple (switch) / complex (car)	Complex

Cross Compilation versus Native Compilation

- Cross-compile is to build a binary on one platform that will run on another platform
- To make a cross-compiler we need a set of development tools that are linked together by specific stages called toolchain

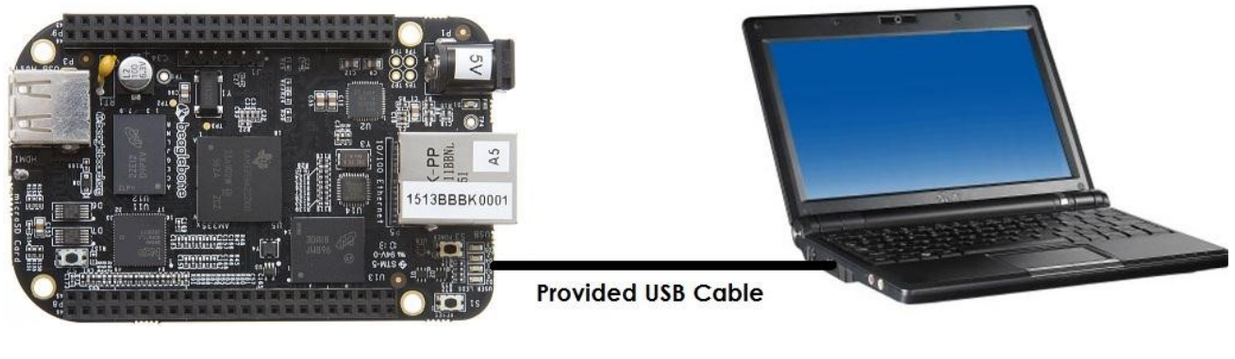


Source: https://www.linuxbaya.com/2020/09/cross-platform-development-toolchain.html

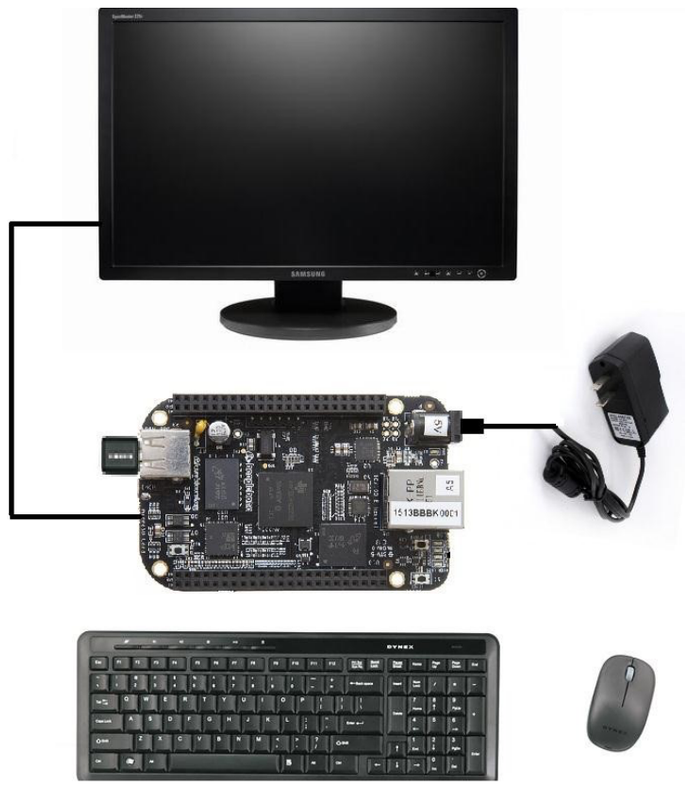
How to Program on Beaglebone Black

Connection Scenarios for Beaglebone Black (BBB)

Tethered to PC



Standalone w/Display and Keyboard/Mouse



After powering your beaglebone, wait for 2 minutes. Once the LED displays a heartbeat pattern, your device has been successfully booted

Connecting through Secure Shell (ssh) - MAC

- Secure Shell (SSH) is a network protocol for secure encrypted communication between network devices
- Use an SSH terminal client to connect to the SSH server that is running on port 22 of your BBB board which allows you to do the following:
  - Log in remotely to the board and execute commands
  - Transfer files to and from the board using the SSH File Transfer Protocol (SFTP)

- Remote Login (MAC / Linux):
  - Open Terminal and run the following command
  - `>> sudo ssh debian@192.168.7.2`
    - 192.168.7.2 is the default IP of the Beaglebone
    - Sometimes IP is 192.168.6.2 for MAC Users

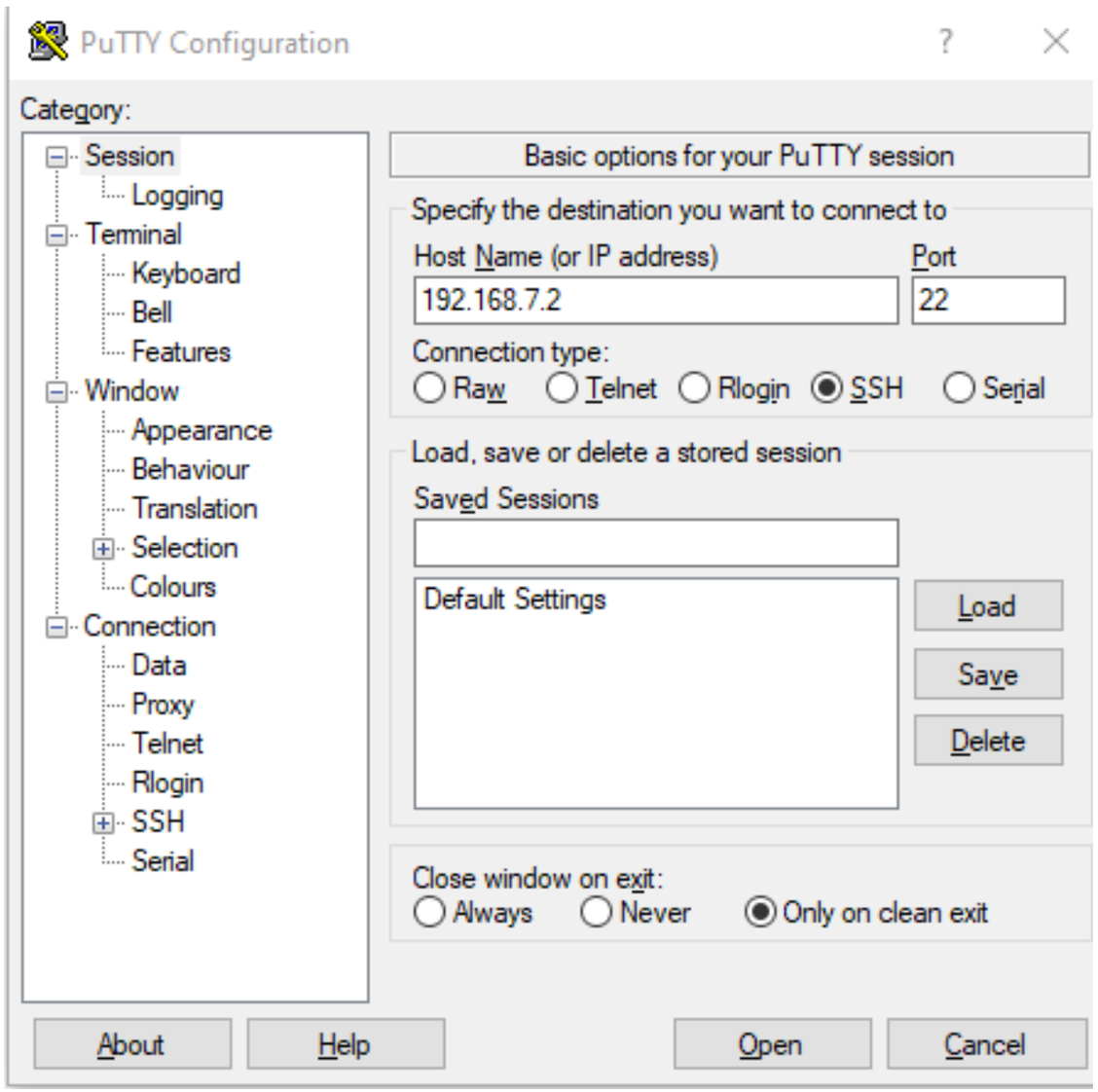
Enter your computer password first (to enable sudo access permission).
Following screen will show up after a successful SSH command

```
Debian GNU/Linux 10 beaglebone ttyGS0

BeagleBoard.org Debian Buster IoT Image 2020-04-06

Support: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian

default username:password is [debian:temppwd]

beaglebone login:
```

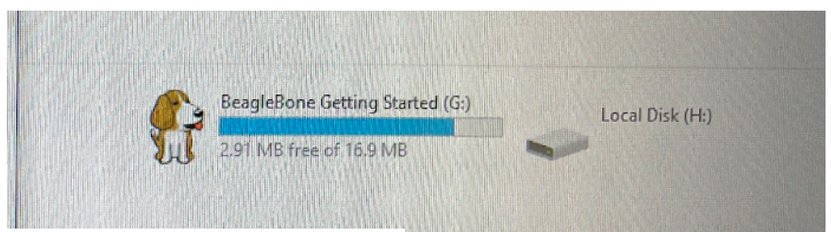
Type username as “debian” and password “temppwd” when asked after a successful connection

Connecting through Secure Shell (ssh) - Windows

- Download and install SSH client, Putty from here: http://www.putty.org/
- Open Putty
- Enter Host name “192.168.7.2”, like the screen on the side,



Enter username “debian” and “temppwd” password



Troubleshoot

Install the BBB compatible drivers on your PC,
From the folder ‘BEAGLEBONE/Drivers/Windows/’. Folders can be located easily in the file explorer.
On Windows 64 bit, install this: http://beagleboard.org/static/Drivers/Windows/BONE_D64.exe
Windows 32 bit, http://beagleboard.org/static/Drivers/Windows/BONE_DRV.exe

Serial Monitor for MAC/Linux

- Create Serial connection, (Mostly for MAC Users)

```
>> ls /dev/tty.*
```

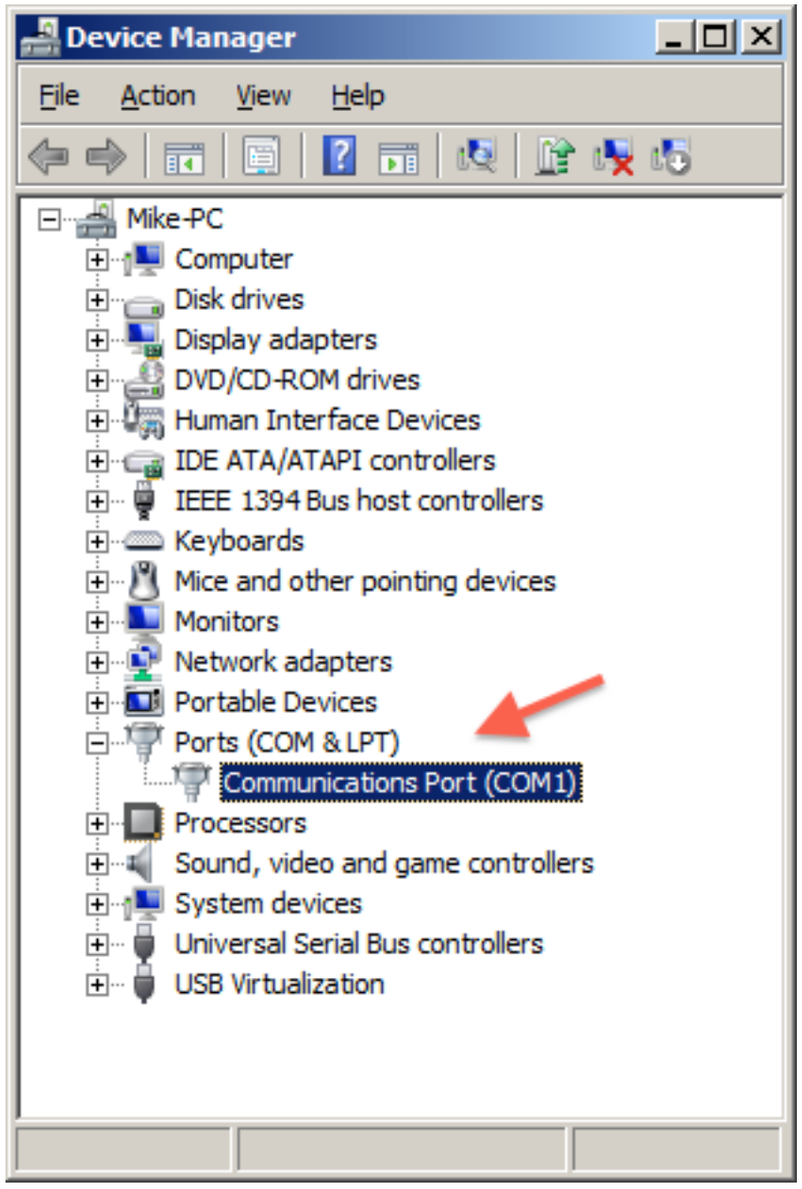
```
>> screen /dev/tty.usbmodem*
```

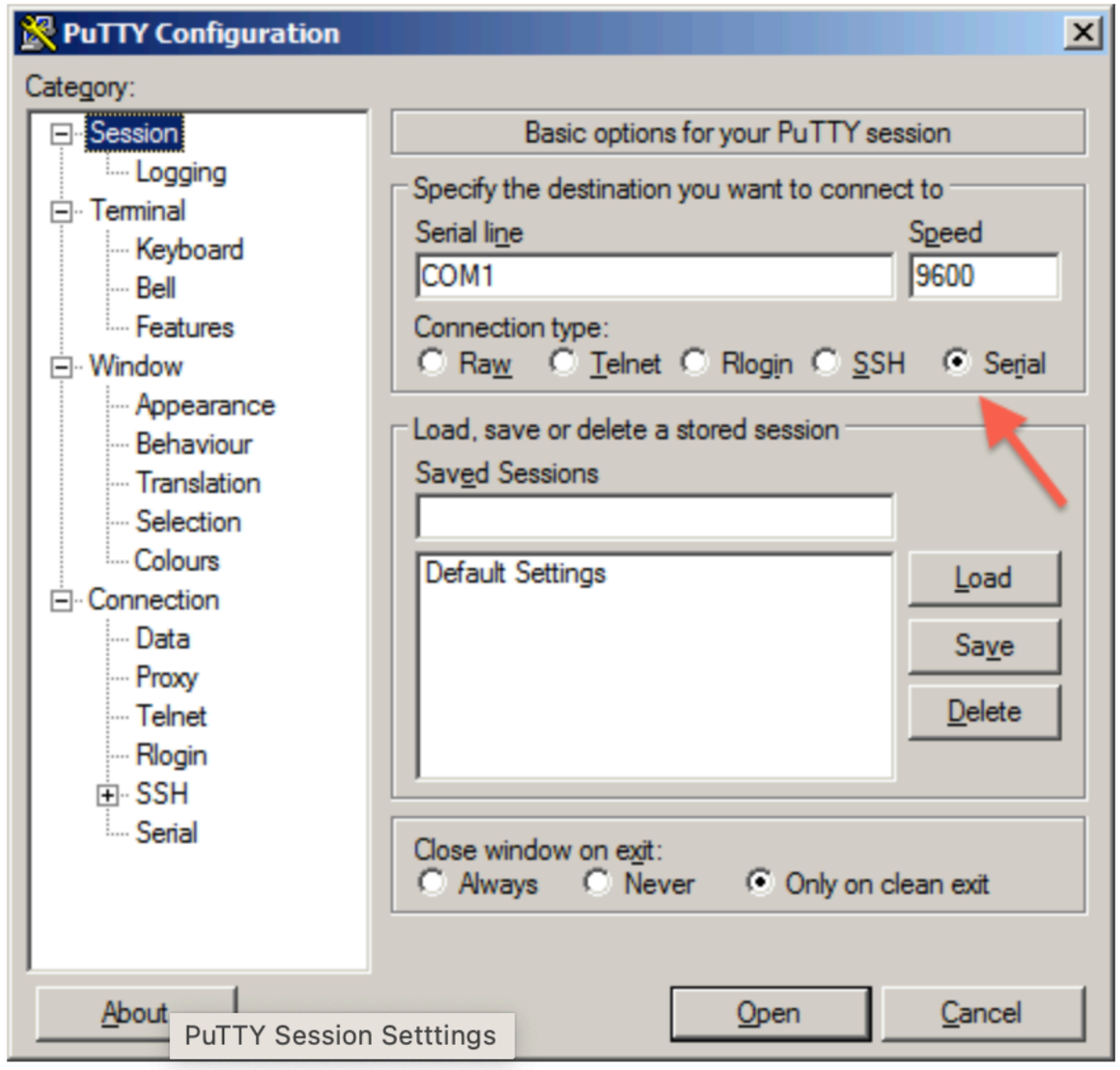
    - Enter username and password
    - Close screen properly, otherwise lingering sessions cause problems

- Useful commands:
    - `screen -ls` (list all process)
    - `ctl-a-k` (kill screen process)

Serial Monitor for Windows

- Navigate to Device manager, Expand Ports (COM & LPT), and Locate COM port
- Open Putty, select Serial and the COM port





Successful Connection to BBB Screen

```
BeagleBoard.org Debian Buster IoT Image 2020-04-06

Support: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian

default username:password is [debian:temppwd]

beaglebone login: debian
Password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
debian@beaglebone:~$
```

Steps for Programming on BBB

- Find out the C compiler version on your BBB,
  - > gcc -v

1. Your development environment,
    1. Use VSCode on PC, Transfer code and Makefile to BBB
    2. Optional: Edit programs on BBB using shell editors such as VIM or NANO

2. Compile code on BBB

3. Run compiled binary and observe the output on BBB

Example: (1) Write C code for Flashing BBB LED

- Use editor of your choice (VSCode) and copy the code in a C file, named "`flash_led.c`"
- You can modify the code by choosing another LED for flashing
- Write a makefile for the C code
- Transfer to BBB and compile with make command
- Run the output binary on BBB and visualize LEDs blinking at the same rate

```c
#include <stdio.h>
#include <unistd.h>

int main(){
    printf("\n LED Flash Start \n");
    FILE *LEDHandle = NULL;
    // directory path to the 4th led's brighness file
    const char *LEDBrightness="/sys/class/leds/beaglebone:green:usr3/brightness";

    // flash the led 10 times
    for(int i=0; i<10; i++){
        // Open the file
        // Read/Write only if the file pointer is not NULL
        // Always check for this condition to avoid Segmentation fault
        if((LEDHandle = fopen(LEDBrightness, "r+")) != NULL){
            // Turn the led on
            fwrite("1", sizeof(char), 1, LEDHandle);
            // Close the led file
            fclose(LEDHandle);
        }
        // sleep for 10^6 microsecond or 1 second
        usleep(1000000);

        // Open the file and check for the same condition as above
        if((LEDHandle = fopen(LEDBrightness, "r+")) != NULL){
            // Turn the led off
            fwrite("0", sizeof(char), 1, LEDHandle);
            // Close the led file
            fclose(LEDHandle);
        }
        // sleep again for 10^6 microsecond or 1 second
        usleep(1000000);
    }
    printf("\n LED Flash End \n");
}
```

Example: (2) Write makefile

- In the working directory, create the "**makefile**" and paste the given contents in it

Replace Spaces by Tabs

```
CC=gcc
CFLAGS=-I.

test.o: flash_led.c
    $(CC) -c -o test.o flash_led.c
test: test.o
    $(CC) -o test test.o
.PHONY: clean
clean:
    rm -f test.o test
```

Make sure to use the correct C file name

- TAKE CARE OF INDENTATION ERRORS

```
vl965-172-31-200-143:BBB_code fatimanwar$ make
makefile:5: *** missing separator.  Stop.
```

Example: (3) Transfer files to BBB

- Use scp command to transfer your C and make files inside your PC working folder to BBB

```
>> scp -r ece231 debian@192.168.6.2:
```

Copy to BBB home

Does recursive copy

- Name of your working folder in PC
- Make sure you are outside of this directory

- If asked “Are you sure you want to continue”, enter “yes”. If asked for the password enter “temppwd”

- Now, on the board terminal, if you type “`ls`” in the root directory, you will see the new folder “`ece231`”

- Now type ‘`ls`’ again to see the contents of the “ece231” folder, you should see the code file and makefile

Example: (4) Compile & Run Output Binary

- Compile the code using **make** command with the target **test**

```
> make test
```

- For cleaning up extra files,

```
> make clean
```

- Execute the program using the command

```
>   ./test
```

- The output would be as follows:

```
debian@beaglebone:~/ece231$ ./test

 LED Flash Start

 LED Flash End
```

- You will observe the LED blinking at the programmed rate

OPTIONAL: Native Programming on BBB

- After logging in to BBB, create a working directory

- Write C code using command line editor such as NANO, VIM

```
> nano flash_led.c
> nano makefile
```

- Copy code and paste inside the editor

- Save the file changes and exit
  - Few options in nano editor:
    - press CTRL+O to save the changes then CTRL+X to close the file

- Compile and run as before

Beaglebone Black GPIOs

Types of Interfaces

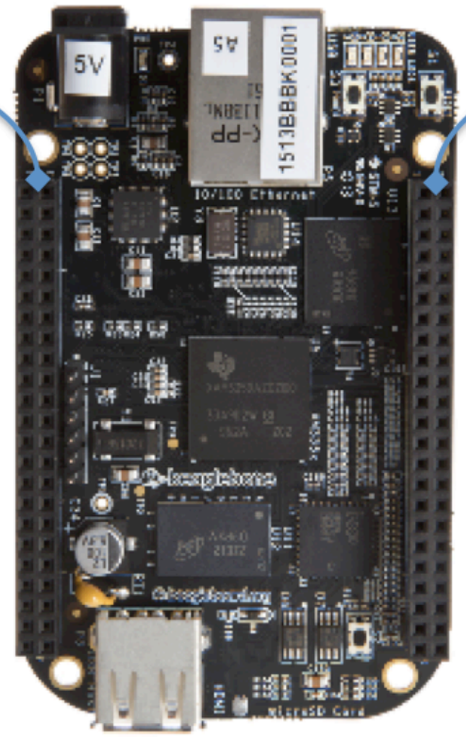
- Interface electronic devices / circuits / modules to BBB in several ways,
  - GPIO - supports versatile devices and circuits
  - Buses (I2C, SPI), UARTs - enables communication with complex modules such as sensors and actuators
  - USB modules - Keyboard and mouse peripheral devices (Linux drivers needed)
  - Wired / Wireless communication - Ethernet, Zigbee, Wi-Fi enables network connectivity

GPIO Header - default configuration

Cape Expansion Headers

P9

DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3
VDD_5V	5	6	VDD_5V
SYS_5V	7	8	SYS_5V
PWR_BUT	9	10	SYS_RESETn
UART4_RXD	11	12	GPIO_60
UART4_TXD	13	14	EHRPWM1A
GPIO_48	15	16	EHRPWM1B
SPI0_CS0	17	18	SPI0_D1
I2C2_SCL	19	20	I2C2_SDA
SPI0_D0	21	22	SPI0_SCLK
GPIO_49	23	24	UART1_TXD
GPIO_117	25	26	UART1_RXD
GPIO_115	27	28	SPI1_CS0
SPI1_D0	29	30	GPIO_112
SPI1_SCLK	31	32	VDD_ADC
AIN4	33	34	GNDA_ADC
AIN6	35	36	AIN5
AIN2	37	38	AIN3
AIN0	39	40	AIN1
GPIO_20	41	42	ECAPPWM0
DGND	43	44	DGND
DGND	45	46	DGND



LEGEND
Power/Ground/Reset
Available Digital
Available PWM
Shared I2C Bus
Reconfigurable Digital
Analog Inputs (1.8V)

P8

DGND	1	2	DGND
MMC1_DAT6	3	4	MMC1_DAT7
MMC1_DAT2	5	6	MMC1_DAT3
GPIO_66	7	8	GPIO_67
GPIO_69	9	10	GPIO_68
GPIO_45	11	12	GPIO_44
EHRPWM2B	13	14	GPIO_26
GPIO_47	15	16	GPIO_46
GPIO_27	17	18	GPIO_65
EHRPWM2A	19	20	MMC1_CMD
MMC1_CLK	21	22	MMC1_DAT5
MMC1_DAT4	23	24	MMC1_DAT1
MMC1_DAT0	25	26	GPIO_61
LCD_VSYNC	27	28	LCD_PCLK
LCD_HSYNC	29	30	LCD_AC_BIAS
LCD_DATA14	31	32	LCD_DATA15
LCD_DATA13	33	34	LCD_DATA11
LCD_DATA12	35	36	LCD_DATA10
LCD_DATA8	37	38	LCD_DATA9
LCD_DATA6	39	40	LCD_DATA7
LCD_DATA4	41	42	LCD_DATA5
LCD_DATA2	43	44	LCD_DATA3
LCD_DATA0	45	46	LCD_DATA1

GPIO Naming Rules for **AM335x**

- AM335x has four chips (0–3) of 32 GPIOs that are numbered 0 to 31
  - Four GPIO chips (0–3)
    - 32 GPIOs on each GPIO chip
    - Example: GPIO1_28 is GPIO 28 out of of 32 (0–31) on the second chip
      - GPIO number corresponding to pin GPIO1_28: (1×32) + 28 = 60
        - Total range is GPIO 0 (i.e., GPIO0_0) to GPIO 127 (i.e., GPIO3_31)
- **Not all AM335x GPIOs are available on the headers of different Beagle boards**
  - Utilized for other functions
  - Not physically connected to the headers

Beaglebone GPIO

65 possible digital I/Os

P9

DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3
VDD_5V	5	6	VDD_5V
SYS_5V	7	8	SYS_5V
PWR_BUT	9	10	SYS_RESETN
GPIO_30	11	12	GPIO_60
GPIO_31	13	14	GPIO_50
GPIO_48	15	16	GPIO_51
GPIO_5	17	18	GPIO_4
I2C2_SCL	19	20	I2C2_SDA
GPIO_3	21	22	GPIO_2
GPIO_49	23	24	GPIO_15
GPIO_117	25	26	GPIO_14
GPIO_115	27	28	GPIO_113
GPIO_111	29	30	GPIO_112
GPIO_110	31	32	VDD_ADC
AIN4	33	34	GNDA_ADC
AIN6	35	36	AIN5
AIN2	37	38	AIN3
AIN0	39	40	AIN1
GPIO_20	41	42	GPIO_7
DGND	43	44	DGND
DGND	45	46	DGND

P8

DGND	1	2	DGND
GPIO_38	3	4	GPIO_39
GPIO_34	5	6	GPIO_35
GPIO_66	7	8	GPIO_67
GPIO_69	9	10	GPIO_68
GPIO_45	11	12	GPIO_44
GPIO_23	13	14	GPIO_26
GPIO_47	15	16	GPIO_46
GPIO_27	17	18	GPIO_65
GPIO_22	19	20	GPIO_63
GPIO_62	21	22	GPIO_37
GPIO_36	23	24	GPIO_33
GPIO_32	25	26	GPIO_61
GPIO_86	27	28	GPIO_88
GPIO_87	29	30	GPIO_89
GPIO_10	31	32	GPIO_11
GPIO_9	33	34	GPIO_81
GPIO_8	35	36	GPIO_80
GPIO_78	37	38	GPIO_79
GPIO_76	39	40	GPIO_77
GPIO_74	41	42	GPIO_75
GPIO_72	43	44	GPIO_73
GPIO_70	45	46	GPIO_71

Source:
https://www.kilobaser.com/blog/2014-07-15-beaglebone-black-gpios

Notes:
https://developer.ridgerun.com/wiki/index.php?title=How_to_use_GPIO_signals

Beaglebone GPIO: Summary

1. The physical pin location, in the form of `PX_Y` (P8_28)
2. The gpio name, in the form of `GPIOX_Y` (GPIO2_24)
3. The gpio number, in the form of `32*X + Y` (88)

   Only the last scheme, the gpio number, is used in software!

Always locate and verify the correctness of pin numbers before programming them

Source: https://vadl.github.io/beagleboneblack/2016/07/29/setting-up-bbb-gpio

Ways to Configure GPIOs

1. GPIO Filesystem

- `/sys/class/gpio/` directory lists the available GPIOs
- The input/output direction of a pin can be set,
  - using the GPIO sysfs entry
  - Examples: **GPIO60** is **GPIO1_28** on **AM335x,** and **pin P9_12** on **beaglebone**

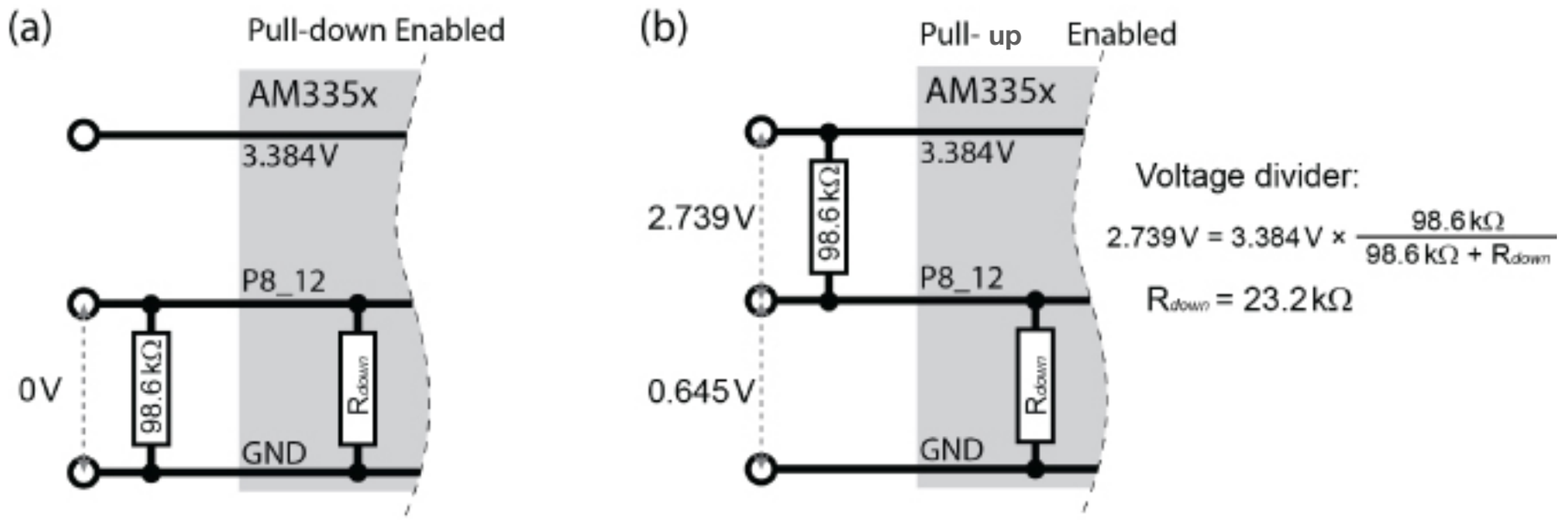
```
debian@ebb:/sys/class/gpio/gpio60$ cat direction
out
debian@ebb:/sys/class/gpio/gpio60$ echo in > direction
debian@ebb:/sys/class/gpio/gpio60$ cat direction
in
```

- **GPIO1_28** being turned on and off

```
debian@ebb:/sys/class/gpio/gpio60$ echo 1 > value
debian@ebb:/sys/class/gpio/gpio60$ echo 0 > value
```

GPIO pull-up pull-down resistors

- Pull-up/down resistors “strongly” tie the i/o to a high/low value using relatively low resistance values
- Beagle boards have “weak” *internal pull-up* and *internal pull-down* resistors that can be configured by setting internal registers



How to program GPIOs

1. Export GPIO to user space
   - Write the GPIO number to the write-only file: **/sys/class/gpio/export**
   - Writing to this file exposes the gpio to the application:

     `echo gpio69 > export` [gpio69 is P8_09]

2. Configure the GPIO direction
   - Write to the file: **/sys/class/gpio/gpio<number>/direction**
   - To configure the GPIO as input (output) write "**in**" ("**out**") to this file (without) quotation marks:

     ```
     echo in > direction
     ```

3. Once configured, read the GPIO value using the file: **/sys/class/gpio/gpio<number>/value**
   - Write to the GPIO using the same file
   - Writing 1 will set the GPIO voltage to high while writing 0 will set the voltage to low value

     ```
     echo 1 > value
     ```

How to program GPIOs: Example

- Configure P8_9 as input, which corresponds to **gpio69**, (verify from GPIO mapping)

```
cd /sys/class/gpio/
```
[Enter GPIO directory]

```
ls
```
[Check if gpio69 is already exposed]

```
echo gpio69 > export
```
[Expose P8_9 to the user application]
[Only if gpio is not exported]

```
cd /sys/class/gpio/gpio69
```
[Enter directory corresponding to P8_9]

```
echo in > direction
```
[Set the gpio as an input pin]

```
cat direction
```
[Prints the value of direction: “in” corresponds..
… to input, and “out” corresponds to output pin]

GPIO toggle using shell script

Copied from textbook, chapter 6

```
debian@ebb:~/exploringbb/chp06/flash_script$ more flash.sh
#!/bin/bash
# Short script to toggle a GPIO pin at the highest frequency
#  possible using Bash - by Derek Molloy
echo "out" > /sys/class/gpio/gpio60/direction
COUNTER=0
while [ $COUNTER -lt 100000 ]; do
    echo 0 > /sys/class/gpio/gpio60/value
    echo 1 > /sys/class/gpio/gpio60/value
    let COUNTER=COUNTER+1
done
debian@ebb:~/exploringbb/chp06/flash_script$ ./flash.sh
```

P9_12

- Output PWM: cycles every 0.54 ms approximately with a frequency of approximately 1.86 kHz
  - Visualized using an oscilloscope / logic analyzer
- Use ‘top’ command to visualize CPU load
- PWM can achieve higher frequencies

2. Config-pin Tool - for Troubleshooting

Usage	Description
`config-pin [-a] <pin> <mode>`	Sets the pin to a mode that is valid for the pin. Use `config-pin -l` to list the valid modes. When the pin is set to mode `gpio` there are additional options.
use instead of `gpio`	**The options for mode `gpio`:**
`in` or `input`	Set the gpio direction to input.
`in+`	Set the gpio direction to input and enable a pull-up resistor on the pin.
`hi-`	Set the gpio direction to input and enable a pull-down resistor on the pin.
`out` or `output`	Set the gpio direction to be an output.
`hi` or `high` or `1`	Set the gpio direction to output and drive the pin high.
`lo` or `low` or `0`	Set the gpio direction to output and drive the pin low.
`config-pin overlay <name>`	Load a named device tree overlay.
`config-pin -l <pin>`	List the valid modes for a pin.
`config-pin -i <pin>`	Show information for a pin.
`config-pin -q <pin>`	Query a pin and report the configuration details.
`config-pin -f [file]`	Read a list of pin configurations from a file.
`config-pin -h`	Display the help text.

How to use Config-pin Tool

Let's try following commands to explore pins and see the direction of the pins and type,

```
GPIO Pin Configurator

Usage: config-pin -c <filename>
       config-pin -l <pin>
       config-pin -q <pin>
       config-pin <pin> <mode>
```

```
> config-pin -q P8_08
> config-pin -l P8_08
> config-pin P8_08 timer
> config-pin -q P8_08
```

Use it as a troubleshooting tool

3. Device Trees (Advanced Concept)

- Tree structure to describe hardware
- Flattened Device Tree (FDT) is a human-readable data structure that describes the hardware on a particular Beagle board using device tree source (DTS) files
- These files describe the properties of the CPU, base memory, GPIOs, PWMs, UARTs, user LEDs, and more
- Compiled into a binary form using a *device tree compiler* (DTC) and are placed on the Linux boot image at,

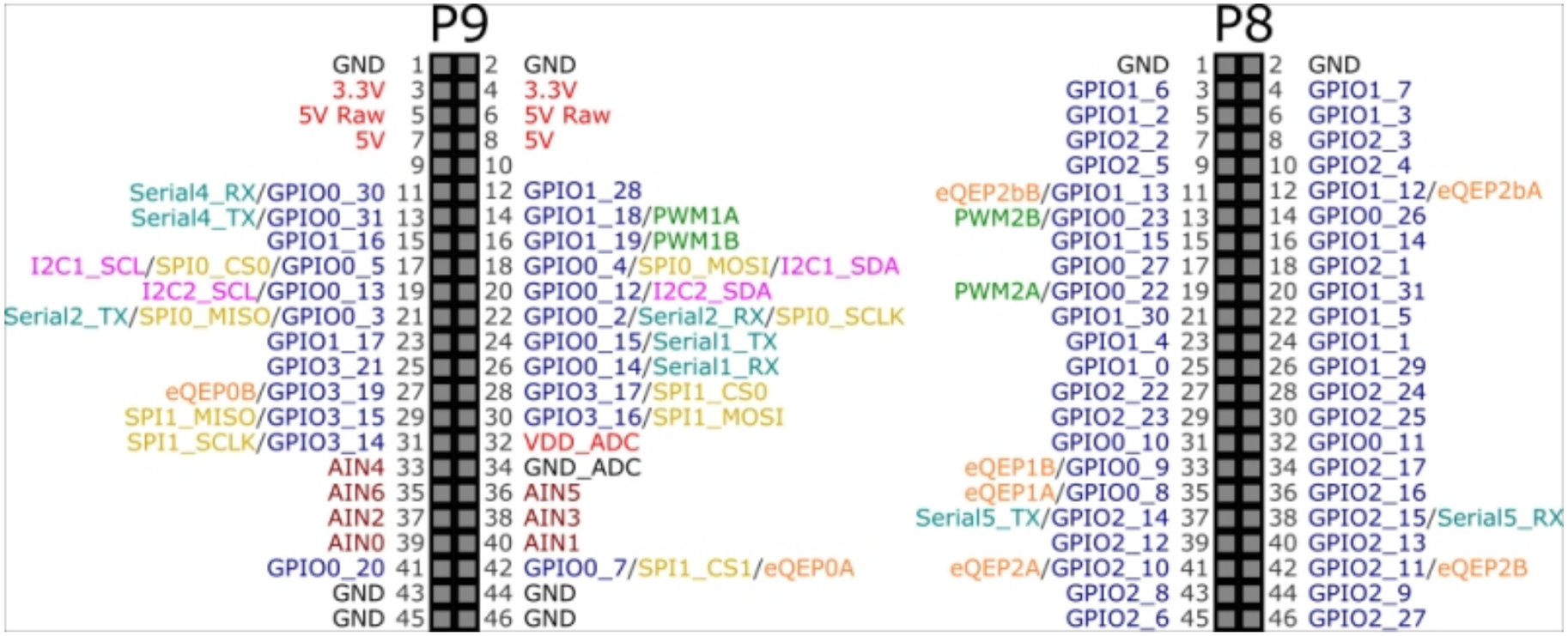
  `/boot/dtbs/`

```
/ {
            compatible = "ti,am33xx";
            interrupt-parent = <&intc>;
            chosen { };
             …
            ocp {
                 compatible = "simple-bus";
                 #address-cells = <1>;
                 #size-cells = <1>;
                 ranges;
                 ti,hwmods = "l3_main";
                 ….
                gpio0: gpio@44e07000 {
                compatible = "ti,omap4-gpio";
                     ti,hwmods = "gpio1";
                     gpio-controller;
                     interrupt-controller;
                     reg = <0x44e07000 0x1000>;
                     interrupts = <96>;
                };
                …
                uart0: serial@44e09000 {
                     compatible = "ti,am3352-uart", "ti,omap3-uart";
                     ti,hwmods = "uart1";
                     clock-frequency = <48000000>;
                     reg = <0x44e09000 0x2000>;
                     interrupts = <72>;
                     status = "disabled";
                     dmas = <&edma 26 0>, <&edma 27 0>;
                     dma-names = "tx", "rx";
                };
```

For information purpose only!

Device Trees based Pin Configuration

- There are seven different modes for each pin, called the *multiplexer mode* (*mmode*)
- Construct a seven-bit value to configure a GPIO pin within Linux using device tree overlays



GPIO Settings				
Bit 6	Bit 5	Bit 4	Bit 3	Bit 2,1,0
Slew Control	Receiver Active	Pullup/Pulldown	Enable Pullup/Pulldown	Mux Mode
0 Fast	0 Disable	0 Pulldown select	0 Enabled	000 Mode 0 to
1 Slow	1 Enable	1 Pullup select	1 Disabled	111 Mode 7

```
e.g. OUTPUT GPIO(mode7) 0x07 pulldown, 0x17 pullup, 0x?f no pullup/down
e.g. INPUT GPIO(mode7) 0x27 pulldown, 0x37 pullup, 0x?f no pullup/down
```

0x37 (0110111) Fast, Input, **Pull-Up**, Enabled, Mux Mode 7

Cape Manager and Device Tree Overlays

- Device tree is loaded when the kernel starts
- It would be a very long and tedious process to modify the device tree, recompile it, and restart the device
- BBB's cape manager allows the modification of parts of the device tree at runtime by loading overlays that enable peripherals or multiplex pins to a specific function
- Device Tree Compiler (DTC) compiles the overlays

Specialized GPIO Programming

1. LED Programming

- LEDs are represented as files at: /sys/class/leds

```
cd /sys/class/leds
ls -l
```

- Navigate to following folder to reach a particular LED and use some commands:

`cd /sys/class/leds/beaglebone\:green\:usrx` [replace x with 0,1,2,3]

- We can find out the current status of the LED by following command:

`more trigger` [Currently trigger is set up as a “heartbeat”]

- We can turn this pattern off by:

`echo none > trigger` [The LED will stop flashing]

- Once the trigger is off, we can turn on the LED using the brightness setting:

`echo 1 > brightness`

- We can turn off the LED using the brightness setting:

`echo 0 > brightness`

- We can set it back to the way it was before we started:

`echo heartbeat > trigger`

2. Pulse Width Modulation (PWM)

- Generation of control signals for motors and certain types of actuators
- Eight PWM outputs on the AM335x,
  - three eHRPWM modules (two outputs each), and two eCAP modules

HARDWARE NAME	HARDWARE ADDRESS	BBB CHIP [3]	CHANNEL	BBB PINS
EHRPWM0	0x48300200	`pwmchip0`	0A	P9_22/P9_31
	0x48300260	`pwmchip0`	0B	P9_21/P9_29
EHRPWM1	0x48302200	`pwmchip2`	1A	P9_14/P8_36
	0x48302260	`pwmchip2`	1B	P9_16/P8_34
EHRPWM2	0x48304200	`pwmchip5`	2A	P8_19/P8_45
	0x48304260	`pwmchip5`	2B	P8_13/P8_46
ECAP0	0x48300100	`pwmchipX`	n/a	P9_42
ECAP2	0x48304100	`pwmchipX`	n/a	P9_28

How to program PWM pins

- Configure PWM device using the following commands:

```
cd /sys/class/pwm/   [Move to PWM directory]
cd pwmchip7          [Enter pwm device corresponding to EHRPWM2]
echo 1 > export      [Configure channel B of EHRPWM2, which corresponds to pin..
                      … P8_13] [* Skip if there is already a folder named “pwm-7:1”]
```

- A new directory “`pwm-7:1`” is created which wasn’t there before
- Optionally check different PWM interfaces
  - “`echo 0 > export`” will let you configure channel A of EHRPWM2 which is pin P8_19 and will create a directory “pwm-7:0”
- Configure the PWM devices and enable it using the following commands:

```
cd pwm-7:1                          [Enter newly created directory to enable PWM and..
                                     .. set it’s period and duty cycle]
echo 1000000000 > period            [Setting PWM period to be 1 second]
echo 250000000 > duty_cycle         [Setting duty cycle to be 0.25 seconds]
echo 1 > enable                     [Enable the PWM]
echo 0 > enable                     [Disables PWM on the pin]
```

PWM Filesystem

- /sys/class/pwm

```
debian@beaglebone:/sys/class/pwm$ ls
pwm-0:0  pwm-1:1  pwm-4:0  pwm-6:0  pwm-7:1   pwmchip1  pwmchip4  pwmchip7
pwm-1:0  pwm-3:0  pwm-4:1  pwm-7:0  pwmchip0  pwmchip3  pwmchip6
```

```
debian@beaglebone:/sys/class/pwm$ cd pwmchip7
debian@beaglebone:/sys/class/pwm/pwmchip7$ ls
device  export  npwm  power  pwm-7:0  pwm-7:1  subsystem  uevent  unexport
debian@beaglebone:/sys/class/pwm/pwmchip7$ cd pwm-7\:1
debian@beaglebone:/sys/class/pwm/pwmchip7/pwm-7:1$ ls -l
total 0
-r--r--r-- 1 root pwm 4096 Apr  7 07:17 capture
lrwxrwxrwx 1 root pwm    0 Apr  7 07:17 device -> ../../pwmchip7
-rw-rw-r-- 1 root pwm 4096 Apr  7 08:03 duty_cycle
-rw-rw-r-- 1 root pwm 4096 Apr  7 08:04 enable
-rw-rw-r-- 1 root pwm 4096 Apr  7 08:03 period
-rw-rw-r-- 1 root pwm 4096 Apr  7 07:17 polarity
drwxrwxr-x 2 root pwm    0 Apr  7 07:17 power
lrwxrwxrwx 1 root pwm    0 Apr  7 07:17 subsystem -> ../../../../../../../../../class/pwm
-rw-rw-r-- 1 root pwm 4096 Apr  7 07:17 uevent
```

1 second

0.25 second

```
debian@beaglebone:/sys/class/pwm/pwmchip7/pwm-7:1$ echo 1000000000 > period
debian@beaglebone:/sys/class/pwm/pwmchip7/pwm-7:1$ echo 250000000 > duty_cycle
debian@beaglebone:/sys/class/pwm/pwmchip7/pwm-7:1$ cat polarity
normal
debian@beaglebone:/sys/class/pwm/pwmchip7/pwm-7:1$ echo 1 > enable
debian@beaglebone:/sys/class/pwm/pwmchip7/pwm-7:1$ echo 0 > enable
```

PWM Filesystem

- Use shell scripting sample codes for gpio operations of interrupting and capturing a pin

PWM Generation using Python

Consider a 3.3 volt signal, which is turning on and off with a frequency of 50 Hz (period of 0.02 seconds, or 20 milliseconds). If during that 20 millisecond period, the signal was “High” for 10 milliseconds, and “Low” for 10 milliseconds, the signal would act like a 1.65 volt analog signal. The output voltage therefore could be considered the rail voltage (3.3 volts) multiplied by the percentage of time the signal is high(duty cycle).

- Create a new file “pwm_test.py”, and copy the following code,

```
import Adafruit_BBIO.PWM as PWM

                    Channel   duty cycle (%)  Frequency(Hz)
PWM.start("P8_13", 25, 1000) # setup PWM
PWM.set_duty_cycle("P8_13", 50) # modify duty cycle
PWM.set_frequency("P8_13", 1) # modify frequency
PWM.stop("P8_13") # Stop PWM
PWM.cleanup() # Clear all channels
```

- Save and compile your python code. Then run it. PWM on the pin can be visualized using a logic analyzer

PWM Testing using Python

- Use logic analyzer to validate pin, PWM frequency and duty cycle
- Alternatively,
  - Hook the PWM pin to a GPIO using a jumper wire
  - Install Adafruit_BBIO.GPIO library on your device and run the following python code,

```
import Adafruit_BBIO.GPIO as GPIO
GPIO.setup("P9_12", GPIO.IN)
n = 0
while(True):
        GPIO.wait_for_edge("P9_12", GPIO.RISING)
        n += 1
        print ("event detected:", n)
```

- PWM will trigger the GPIO pin and “event detected” statement is printed

Reading

- Practice all these concepts/commands on your shells
- Book: Read Chapter 6: Interfacing to Beagleboard Inputs/Outputs
  - Go through only relevant topics