

*EC-ENG 231 (Spring 2024)*

# Introducing Beaglebone Black

Fatima Anwar

[fanwar@umass.edu](mailto:fanwar@umass.edu)

UMass Amherst



# Input & Outputs

# Command Line Inputs

---

- A program can be provided input in two ways:
- First using input arguments with the program executable. Using this method inputs can only be provided at the start of the program execution. This is also referred to as providing command line arguments
- A C program starts its execution with `main()` function, which is often declared with zero input arguments as below:

```
int main() { /* ... */ }
```

- To receive command line arguments, the `main()` function declaration can be modified to have two arguments as below:

```
int main(int argc, char *argv) { /*  
... */ }
```

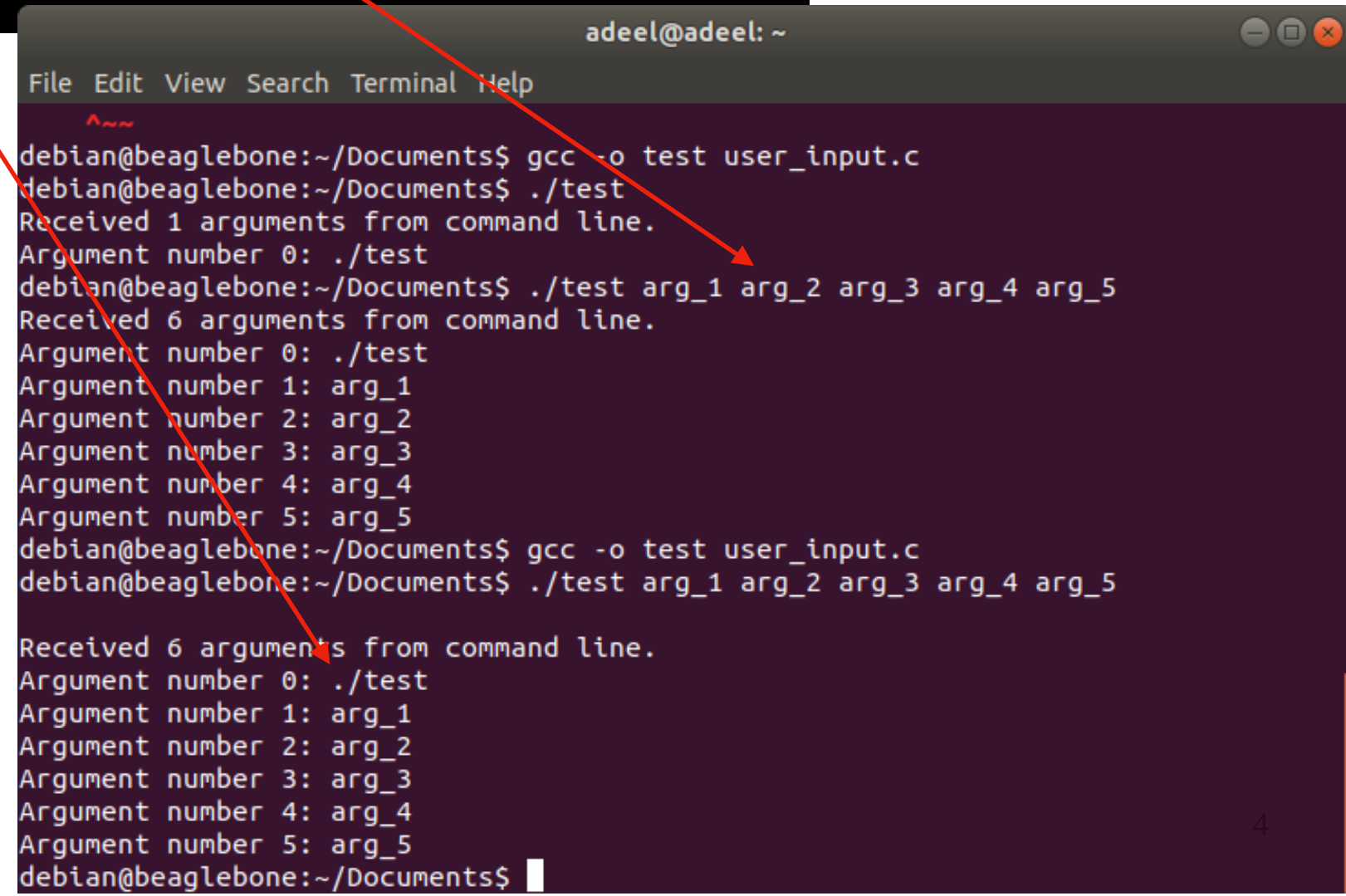
- “**argc**” is an integer containing the total number of arguments provided
- “**argv**” is array of character pointers listing all the arguments



# Command Line Inputs - Example 1

```
#include <stdio.h>

int main(int argc, char *argv[]) { // to receive command line arguments
    printf("\nReceived %d arguments from command line.\n", argc);
    // printing command line arguments
    for(int i=0; i<argc; i++){
        printf("Argument number %d: %s\n", i, argv[i]);
    }
}
```



The terminal window shows the execution of the C program. It first compiles the program with `gcc -o test user_input.c`. Then, it runs the program with `./test`, which outputs "Received 1 arguments from command line." and "Argument number 0: ./test". Next, it runs the program with five additional arguments: `./test arg_1 arg_2 arg_3 arg_4 arg_5`. This results in "Received 6 arguments from command line." and six lines of output, each showing an argument number from 0 to 5 and its corresponding value. Red arrows from the code block above point to the `argc` variable in the first run and the `argv` array in the second run.

```
adeel@adeel: ~
File Edit View Search Terminal Help
^~~
debian@beaglebone:~/Documents$ gcc -o test user_input.c
debian@beaglebone:~/Documents$ ./test
Received 1 arguments from command line.
Argument number 0: ./test
debian@beaglebone:~/Documents$ ./test arg_1 arg_2 arg_3 arg_4 arg_5
Received 6 arguments from command line.
Argument number 0: ./test
Argument number 1: arg_1
Argument number 2: arg_2
Argument number 3: arg_3
Argument number 4: arg_4
Argument number 5: arg_5
debian@beaglebone:~/Documents$ gcc -o test user_input.c
debian@beaglebone:~/Documents$ ./test arg_1 arg_2 arg_3 arg_4 arg_5

Received 6 arguments from command line.
Argument number 0: ./test
Argument number 1: arg_1
Argument number 2: arg_2
Argument number 3: arg_3
Argument number 4: arg_4
Argument number 5: arg_5
debian@beaglebone:~/Documents$
```

# Command Line Inputs

---

- C programs can also receive inputs from inside the `main()` function, using the following functions:
  - **`scanf()`** to receive integers, floating point numbers and characters as input
    - *`int scanf(const char *format, ...)`*
  - **`gets()`** can be used to take character array as input
    - *`int gets(const char *str)`*
  - **`getchar()`** can be used to take a single character input
    - *`int getchar(int char)`*

# Command Line Inputs - Example 2

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

```
int main( ) {  
    int int_value;  
    float float_value;  
    char char_value;  
    char str[100];
```

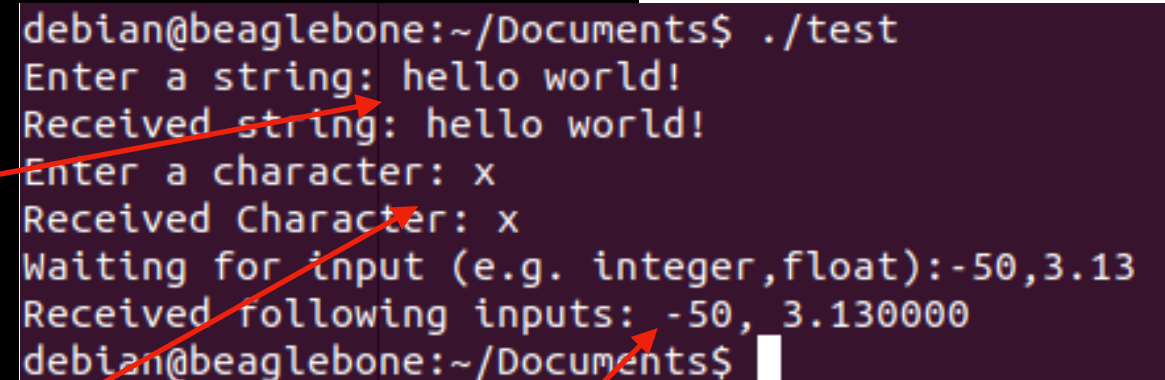
```
    printf("Enter a string: ");  
    gets(str);  
    printf("Received string: %s\n", str);
```

```
    printf("Enter a character: ");  
    char_value = getchar();  
    printf("Received Character: %c\n", char_value);
```

```
    printf("Waiting for input (e.g. integer,float):");
```

```
    // takes comma separated inputs
```

```
    scanf("%d,%f", &int_value, &float_value);  
    printf("Received following inputs: %d, %f\n", int_value, float_value);  
}
```



```
debian@beaglebone:~/Documents$ ./test  
Enter a string: hello world!  
Received string: hello world!  
Enter a character: x  
Received Character: x  
Waiting for input (e.g. integer,float):-50,3.13  
Received following inputs: -50, 3.130000  
debian@beaglebone:~/Documents$
```

The terminal screenshot shows the program's execution. Red arrows point from the code in the left panel to the corresponding output in the terminal. The first arrow points from the `gets(str);` line to the "Received string: hello world!" output. The second arrow points from the `getchar();` line to the "Received Character: x" output. The third arrow points from the `scanf("%d,%f", ...);` line to the "Received following inputs: -50, 3.130000" output.

# Command Line Outputs

---

- Computer programs often need to write out content to the screen. This is the part of user interaction, and is used to either display current program status and accomplish other program objectives
- In C, there are multiple ways to write output to the terminal
- The first one uses **printf** function. It is the most commonly used method to write output to the terminal in C
  - *int printf(const char \*format, ...)*
- There are two other methods used to write output to the terminal in C programs:
  - **puts()** can be used to output character array
    - *int puts(const char \*str)*
  - **putchar()** can be used to output a single character
    - *int putchar(int char)*
- Example codes in next slide demonstrates the use of the above functions

# Command Line Outputs - Examples

```
adeel@adeel: ~  
File Edit View Search Terminal Help  
default username:password is [debian:temppwd]  
  
debian@192.168.7.2's 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.  
Last login: Fri Feb  5 21:59:24 2021 from 192.168.7.1  
debian@beaglebone:~$ cd Documents/  
debian@beaglebone:~/Documents$ gcc -o test user_input.c  
debian@beaglebone:~/Documents$ ./test  
Hello World !  
Integer: -50  
Integer, Floating Point Number: -50, 3.142800  
Integer, Floating Point Number: -50, 3.142800, z  
Integer, Floating Point Number: -50, 3.142800, z, String  
debian@beaglebone:~/Documents$ gcc -o test user_input.c  
debian@beaglebone:~/Documents$ ./test  
Hello World  
!  
debian@beaglebone:~/Documents$
```

```
#include <stdio.h>  
  
int main( ) {  
    char str[100] = "Hello World";  
    // printing a character array  
    puts(str);  
    //printing character  
    putchar('!');  
    putchar('\n');  
}
```

```
int main( ) {  
    // print a string  
    printf("Hello World !\n");  
    //print an integer  
    printf("Integer: %d\n", -50);  
    //print a floating point number  
    printf("Integer, Floating Point Number: %d, %f\n", -50, 3.1428);  
    //print a character  
    printf("Integer, Floating Point Number: %d, %f, %c\n", -50, 3.1428, 'z');  
    //print a string  
    printf("Integer, Floating Point Number: %d, %f, %c, %s\n", -50, 3.1428, 'z', "String");  
}
```



# File Inputs & Outputs (I/O)

---

- So far, we have read all input from the keyboard and written all output to the screen
- Often, we want to be able to read and write information to and from files on disk for several reasons
  - Data Storage
    - when a program terminates, data stored in variables is lost, but files retain information even after the program ends
  - Data Retrieval
    - Files provide a means to retrieve and reuse data
- Applications:
  - Data Management
  - Data Analytics
  - Telemetry
  - Logging
  - Debugging

# File I/O: Open & Close

---

- **Opening files**

- When working with files, you need to declare a pointer of type file
  - `FILE *fptr;`
  - `fptr = fopen("E:\\cprogram\\newprogram.txt", "w");`
- The file must exist or will be created based on the mode specified (e.g., read, write, append)
- File Operations Modes: "r" (read), "w" (write), "a" (append), "r+" (read and write), "w+" (write and read, truncating file), "a+" (append and read)
- **File Pointers:**
  - It is a pointer to a `FILE` structure
  - A cursor that stores the position of the next character to be read or written in a file

- **Closing Files**

- `fclose(fptr);`
- Closes the opened file, freeing up system resources. It's essential to close files after use to avoid resource leaks

# File I/O: Writing to a File

- Writing to files:
  - `fwrite(const void *content, size_t size, size_t nmemb, FILE *fptr)`
  - file version of printf that expects a pointer to the structure FILE
    - `fprintf(fptr, "%d", num)`
    - `fputs(char array[], fptr)`

```
#include <stdio.h>

int main() {

    FILE *filePointer = fopen("example.txt", "w"); // Open "example.txt" in write mode
    int intValue = 42;

    // Write formatted text to the file
    fprintf(filePointer, "%d\n", intValue);
    fprintf(filePointer, "%d\n", intValue);
    fprintf(filePointer, "%d\n", intValue);

    // Example using fputs
    char text[] = "Hello, World!\n";
    fputs(text, filePointer); // Write a string to the file

    // Example using fwrite
    char char_data[] = {'A', 'B', 'C'}; // Char array of A, B, C letters
    fwrite(char_data, sizeof(char), sizeof(char_data), filePointer);

    fclose(filePointer); // Close the file when done using //fclose
    return 0;
}
```

# File I/O: Reading from a File

---

- Allows reading data from a file
- Reading is crucial for retrieving stored information
  - `fscanf()` reads formatted data

```
#include <stdio.h>
int main() {

    // Open "example_int.txt" in read mode
    FILE *filePointer = fopen("example_int.txt", "r");

    // Return if file does not exist
    if(filePointer == NULL){
        printf("File does not exist\n");
        return 0;
    }

    // Example using fscanf
    int intValue;
    // Read an integer from the file
    fscanf(filePointer, "%d", &intValue);
    printf("%d\n", intValue); //print read value

    fscanf(filePointer, "%d", &intValue);
    printf("%d\n", intValue); //print read value

    fscanf(filePointer, "%d", &intValue);
    printf("%d\n", intValue); //print read value

    fclose(filePointer); // Close the file when done
    using fclose
    return 0;
}
```

# File I/O: Reading from a File

- `fgets()` reads a line
- `fread()` reads a specified number of bytes

```
#include <stdio.h>
#include <string.h>

int main() {

    FILE *filePointer = fopen("example_char.txt", "r");

    if(filePointer == NULL){
        printf("File does not exist\n");
        return 0;
    }

    // Example using fgets
    char buffer[100];
    // Read a line of text from the file
    fgets(buffer, sizeof(buffer), filePointer);
    //printf("%s",buffer); //print read value
    puts(buffer);

    // Resetting buffer content
    memset(buffer, '\0', sizeof(buffer));

    // Read byte data from the file
    fread(buffer, sizeof(char), sizeof(buffer), filePointer);
    puts(buffer); //print read value

    // Close the file when done using fclose
    fclose(filePointer);
    return 0;
}
```



# File I/O: Other useful file operations

---

- Moving File Position
  - Function: `fseek()`, `rewind()`
  - Significance: Moves the file position indicator to a specified location. `fseek()` is used for precise positioning, while `rewind()` resets it to the beginning. This is important for random access to different parts of a file
- Checking for End-of-File (EOF)
  - Function: `feof()`
  - Significance: Checks whether the end of the file has been reached. Useful for loop termination when reading a file

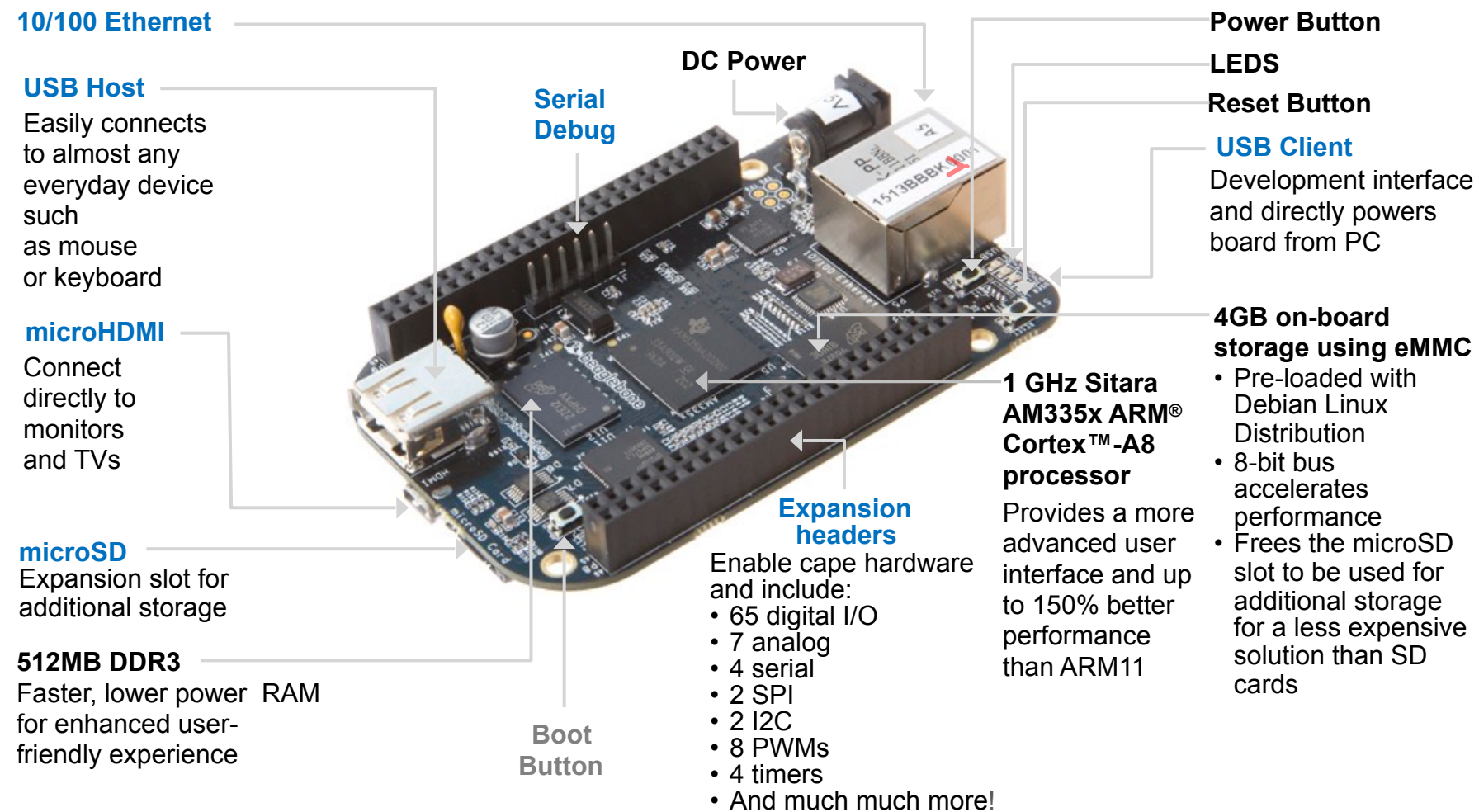
# Reading

---

- Practice all these concepts/commands on your computer and BBB command line
- Book: Read the following topics from Chapter3: Exploring Embedded Linux Systems,
  - Introducing Embedded Linux
  - Advantages Disadvantages of Embedded Linux
  - Is Linux open source and free
  - Kernel Space and User Space
  - Managing Linux Systems
  - System Administration - Linux FileSystem
  - Filesystem Permissions
  - Linux Commands

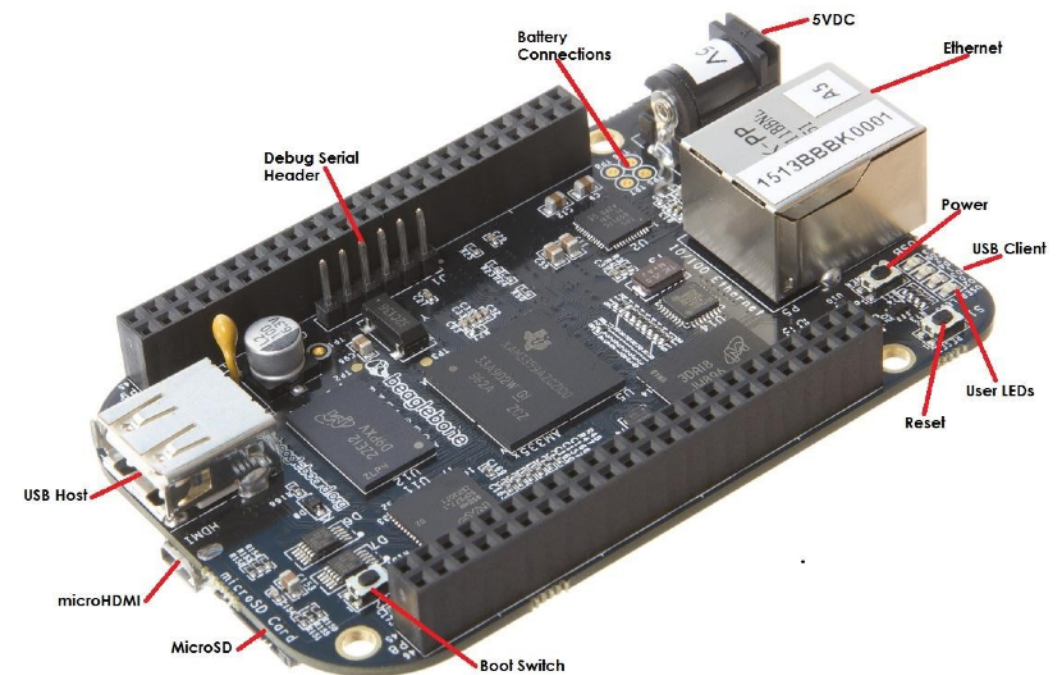
# Overview of Beaglebone Black

# Beaglebone Black Overview



# Connectors & Switches

- **Power Button**
  - alerts the processor to initiate the power down sequence
  - used to power down the board
- **DC Power**
  - accepts 5V power
- **10/100 Ethernet**
  - connection to the LAN
- **USB Client** is a
  - miniUSB connection to a PC
  - also power the board
- **USB Host**
  - Wi-Fi, BLUETOOTH, Keyboard, mouse etc
- **BOOT switch**
  - used to force a boot from the microSD card
- **Reset Button**
  - allows the user to reset the processor
- **microHDMI**
  - display is connected to this port
- **Serial Debug** and **microSD**

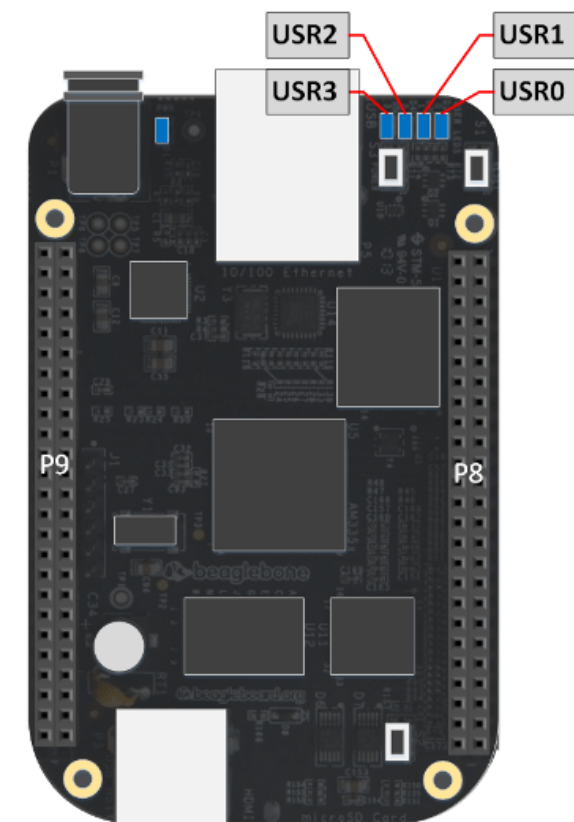




# LEDs

---

- LEDS can be over written and used as desired,
- They do have specific meanings in the image that is shipped with the board once the Linux kernel has booted.
- **USR0** is a heartbeat indicator from the Linux kernel
- **USR1** turns on when the microSD card is being accessed
- **USR2** is an activity indicator. It turns on when the kernel is not in the idle loop
- **USR3** turns on when the onboard eMMC is being accessed

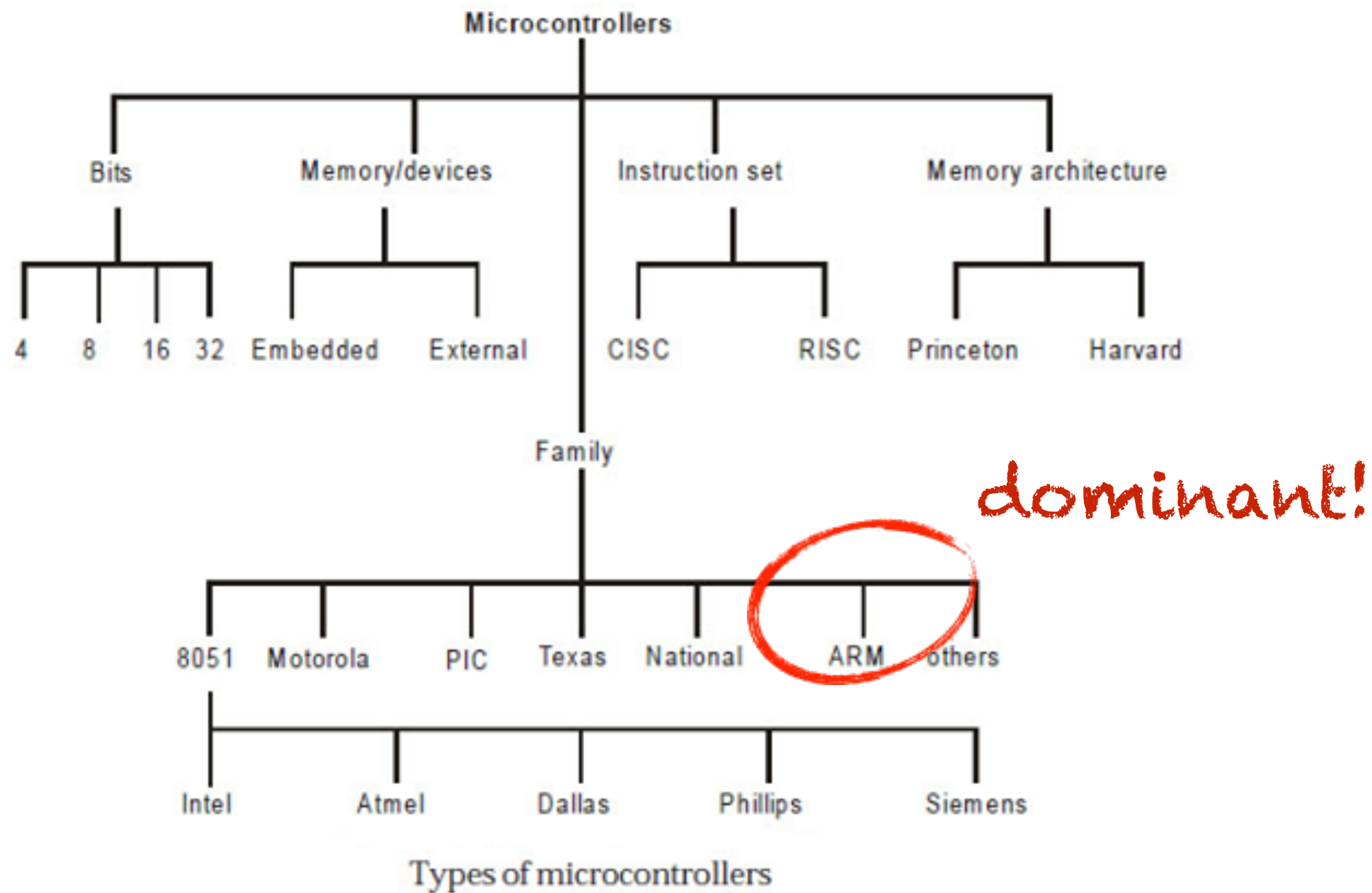


# Beaglebone Black v/s Raspberry Pi

---

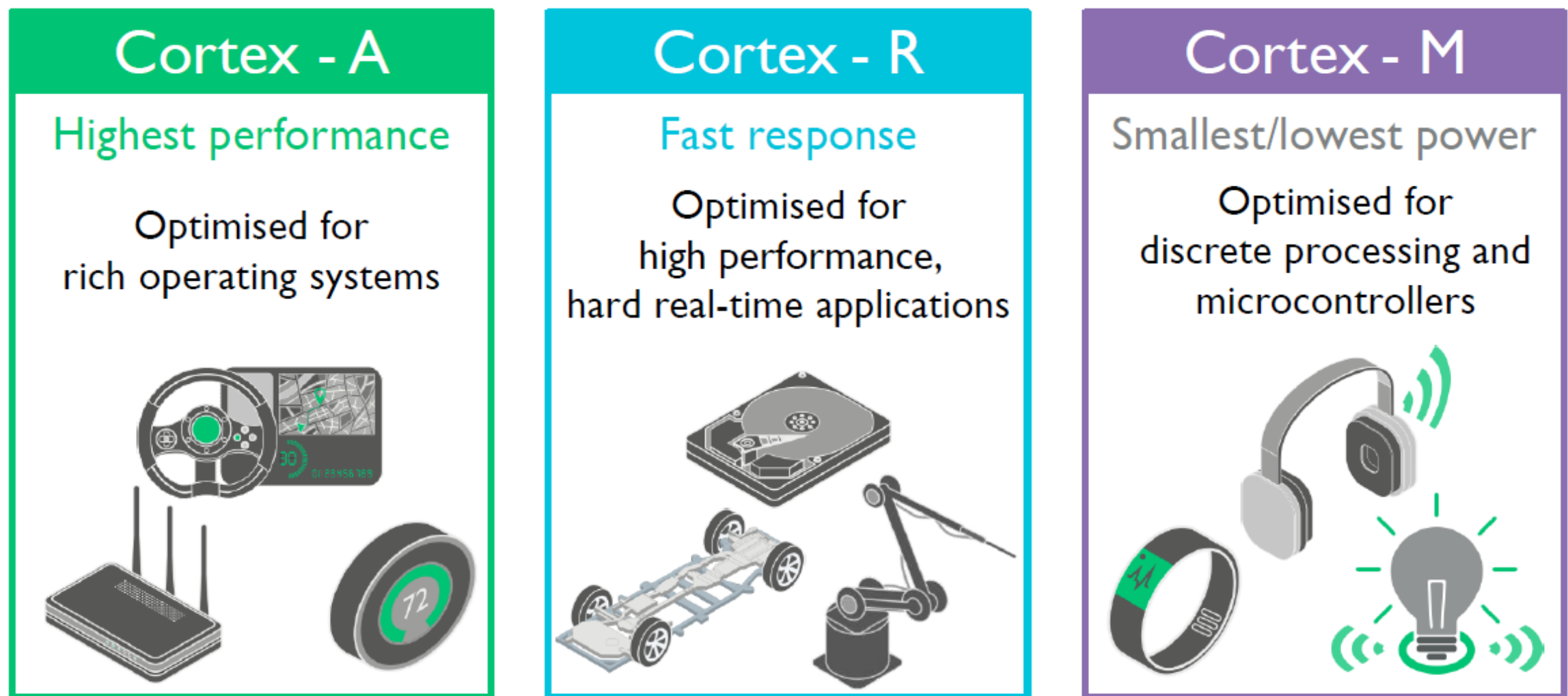
- There are alternatives to BBB such as Raspberry Pi,
  - Closed source for low level details
  - No predictable timing capabilities
  - Hobbyist board
  - Not as diverse GPIOs
- Beaglebone black is,
  - Engineering focused board
  - Extendable hardware with diverse pins
  - More IO capabilities: Timer, PWM, CAN
  - Predictable timing pins and Real Time co-processors

# MCU Specifications Diversity



# ARM Processors

## ARM Architecture: For Diverse Embedded Processing Needs



- Traditionally Chip vendors develop their own CPU Core and license them such as ARM
- Recently, open-source core also available e.g., Berkeley's RISC-V

# Processor

---

- AM335x 1GHz ARM® Cortex-A8
  - 512MB DDR3 RAM
  - 2x PRU 32-bit microcontrollers
  - 4GB 8-bit eMMC on-board flash storage
  - 4KB EEPROM
  - 3D graphics accelerator
  - NEON floating-point accelerator
- Delivers 2000 MIPs at a low cost
- Delivers optional 3D graphics acceleration and key peripherals
- Supports Operating systems: Linux® Android™ Windows Embedded CE QNX ThreadX



# Memory

---

- User-modifiable read-only memory (ROM)
  - Can be erased and reprogrammed repeatedly by applying higher than normal electrical voltage
  - Can be erased and reprogrammed in its entirety, not selectively
  - Limited life time
  - On Beagle board single 4KB EEPROM holds the board information: Board name, Serial number, Revision information
- Embedded Multi-Media Controller (eMMC)
  - A package consisting of both flash memory and a flash memory controller integrated on the same silicon die
  - consists of at least three components: MMC (multimedia card) interface, Flash memory, Flash memory controller
  - Beaglebone has single 4GB embedded MMC (eMMC)

# Boot Modes

---

## ✓ eMMC boot (MMC1):

- Default boot mode
- Fastest boot time
- Enable the board to boot out of the box using the pre-flashed OS image

## ✓ SD boot (MMC0):

- Boot from the microSD slot.
- Used to override kernel on eMMC device
- Used to program the eMMC

## • Serial boot (UART0):

- Boot from serial port
- Allow downloading of direct software
- USB to serial cable is required to use this port

## • USB boot (USB0):

- Supports booting over the USB port

# Bootloader

---

- A piece of code that runs before any operating system is running
- Usually each operating system has a set of bootloaders specific for it
- Contain several ways to boot the OS kernel
- Contain commands for debugging and/or modifying the kernel environment

# Bootloader for Beaglebone: U-Boot

---

- Boots an operating system by reading the kernel and any other required data (e.g. device tree or disk image) into memory
- U-Boot's commands can be used to read or write any arbitrary data
- Using these commands, data can be read from or written to any storage system that U-Boot supports
- Executes the kernel with the appropriate arguments
  - DESCRIPTIONS
  - MACHINE/ARCHITECTURE TYPE
  - COMPRESSION TYPE
  - LOAD ADDRESS
  - CHECKSUMS ETC

# Useful Commands

---

- Determine which version of Linux you are running
  - > `uname -a`
- Use the following command to set a non default user account password
  - > `passwd`
- What time is it,
  - > `date`

COMMAND	DESCRIPTION
<code>more /etc/issue</code>	Returns the Linux distribution you are using.
<code>ps -p \$\$</code>	Returns the shell you are currently using (e.g., bash).
<code>whoami</code>	Returns who you are currently logged in as (e.g., debian).
<code>uptime</code>	Returns how long the system has been running.
<code>top</code>	Lists all the processes and programs executing. Press Ctrl+C to close the view.



# Interfaces & Communication Technologies

---

## Interfaces:

- Six UART
- Two Serial Peripheral interface(SPI)
- Three Inter-Integrated Circuit(I<sup>2</sup>C)
- Two Controller area Network(CAN) bus
- Two Multichannel Audio Serial Port(McASP)
- One USB2.0

## Two port Ethernet Media Access Controller (EMAC)

- Move data between the device and another host connected to the same network
- Synchronous 10/100 Mbit operation

## IEEE1588v2 (Precision time protocol)

- Synchronize clocks throughout a packet-switched network

# Clocks & Timers

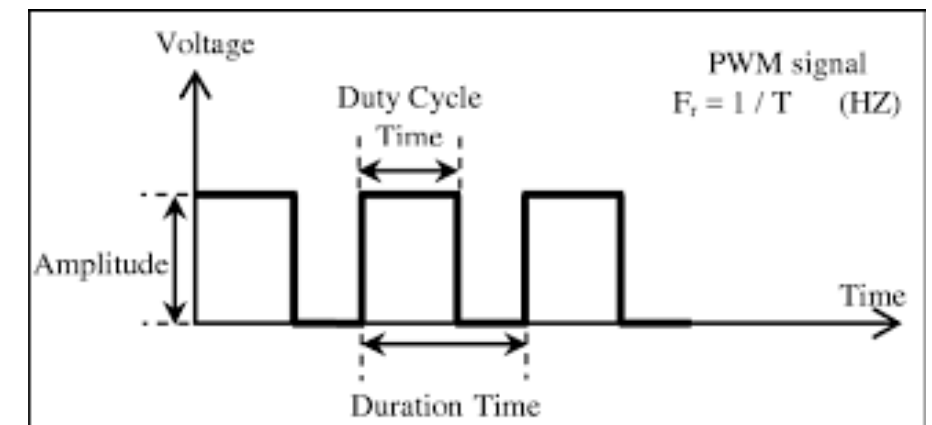
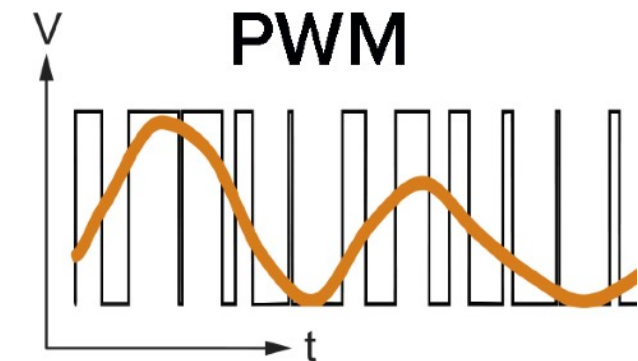
---

- Dual mode timer (DMtimer)\*
  - 32-bit timer: free running upward counter with auto reload capability on overflow
  - DM timer modes
    - Timer mode: time counter
    - Capture mode: allows capturing timer value in register on an input signal
    - Compare mode: allows an interrupt event on a programmable counter matching value
  - There are 4 Timer pins on the board: help in capture and compare mode
- Two clock inputs\*
  - OSC0 oscillator
    - Main oscillator on the device, produces the master high frequency (24-MHz) clock
    - Clocks all non-real-time clock functions from device clock tree
  - OSC1 oscillator
    - Provides a 32.768-KHz reference clock to the real-time clock (RTC)
    - OSC1 is disabled by default after power is applied

\* More details in the datasheet

# Pulse Width Modulation (PWM)

- PWM is a technique for getting analog output with digital pins
- Analog output is simulated on most microcontrollers by creating a fast on/off sequence by controlling the duration of on and off time
- There are 8 PWM pins on BBB
  - There are 3 onboard Enhanced High Resolution PWM (**EHRPWM**) modules, each with two outputs
    - Motor Speed Control
  - There are 2 enhanced capture (eCap) PWM modules
    - Sample rate measurements of audio inputs
    - Speed measurements of rotating machinery sensors)
    - Elapsed time measurements between position sensor pulses



# Internet Connection

---

- Two ways: Ethernet, and USB
- **Internet over USB** (not via wired ethernet)
  - A “private” virtual LAN is created using a single USB cable
  - Your computer must be running to transfer data to/from the internet to the BBB

ADVANTAGES	DISADVANTAGES
Provides a stable network setup for beginners.	Without significant effort, you are limited to a single board per desktop.
When you do not have access to, or control of, network infrastructure hardware, you can still connect the board to the internet.	Network sharing configuration can be difficult, especially on Macintosh desktop computers. Additional configuration must also be performed on the board's Linux OS.
Power is supplied by your desktop machine over USB.	Your desktop machine must be running to transfer data to/from the internet.

# Serial Monitor

---

- Create Serial connection,

> *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)

# File Transfer

---

- SSH File Transfer Protocol (SFTP)
  - A secure file protocol that is used to access, manage, and transfer files over an encrypted SSH transport
  - Works on a client-server model
  - A subsystem of SSH and supports all SSH authentication
- SCP (secure copy) is a command-line utility that allows you to securely copy files and directories between two locations
  - copy a file or directory:
  - From your host pc to root directory of BBB

```
> scp name_of_file.c debian@192.168.7.2:
```
  - Copy an entire folder from your computer to a specific directory in BBB

```
> scp -r name_of_folder1 debian@192.168.7.2:name_of_folder2
```

# How to not destroy your board

---

- It is quite rare to destroy the board but make sure NOT to do the following,
  - Do not shut the board down by pulling out the power jack/USB power. Correctly shut the board down by using a software shutdown (e.g., by pressing the power button once) or by holding the power button for about eight seconds for a “hard” power down
  - The GPIO pins are 3.3V tolerant (most of the ADCs are 1.8V tolerant). Do not connect a circuit that is powered at 5V or you will destroy the board
  - Do not connect circuits that apply power to the expansion header while the board is not powered on
  - Carefully check the pin numbers you are using



# Datasheet Reading

---

- Go through Beaglebone Specifications in the datasheet
- BBB Datasheet,
  - Sitara AM335x ARM Cortex-A8 Technical Reference Manual (TRM): [www.ti.com/product/am3358](http://www.ti.com/product/am3358)

# Reading

---

- Setup your terminals specific to your OS (Windows, MAC, Linux)
- Practice all these concepts/commands on your terminals
- Textbook: Read,
  - Chapter#1
  - Chapter#2: only relevant topics
  - Chapter#5: C overview