



Beaglebone Black booting using TFTP Protocol

Kernel Boot Arguments

**Now, we are ready to boot from the memory!
But since we are using RAM based file system, we
have to tell that to the Linux kernel via the boot
arguments, otherwise Linux will not be having any
idea where exactly the file system resides and the
boot my fail.**

Kernel Boot Arguments

**setenv bootargs console=ttyO0,115200 root=/dev/ram0 rw
initrd=0x88080000**

Annotations:

- Console and baud rate info
- RAMFS device file
- Initramfs address in memory

Kernel Boot Arguments

Use the u-boot environmental variable “bootargs” to send the boot arguments to the linux kernel .

U-boot's Boot From Memory command

`bootm ${kernel_load_address} ${initramfs_load_address} ${dtb_load_address}`

`bootm 0x82000000 0x88080000 0x88000000`

Challenge for you -1

Change the **login name** with a custom name

Format: <your_name>-<board_name>

eg. : kiran-beaglebone

Also change the logo of the file system

Challenge for you -2

Create your own file system using Busybox tool and re test this experiment .

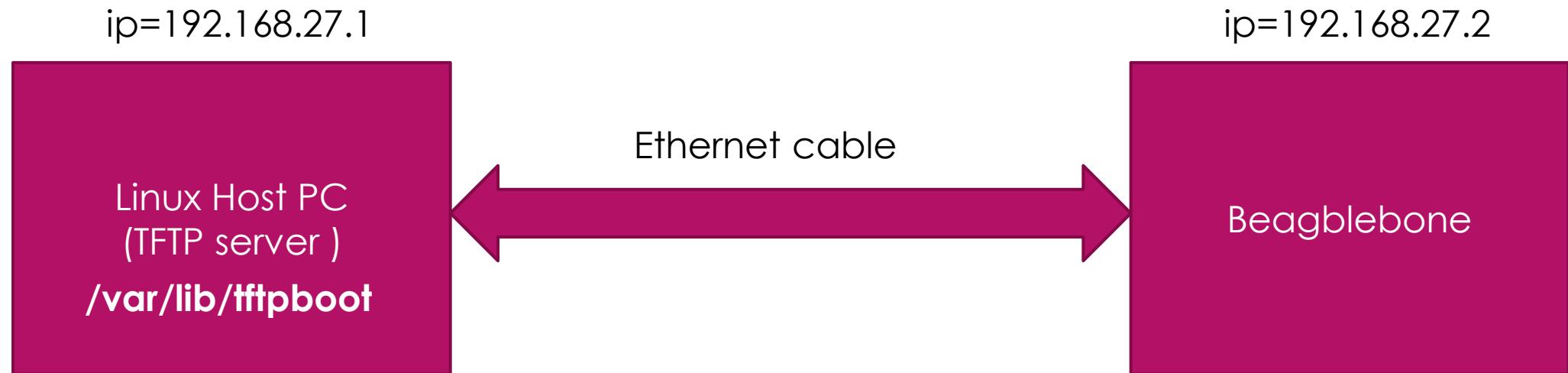
Hint :

- 1) Create file system using busybox
- 2) Convert it to initramfs
- 3) Test the boot with above created initramfs

Next →

Boot via Network (TFTP)

BBB booting using TFTP Protocol



Uboot's TFTP boot command

tftpboot \${load_address} \${file name}

Example:

tftpboot 0x82000000 ulmage

Note: command syntax may change with newer versions of uboot, please run “help tftpboot” on the uboot for detailed explanation

Advantage of using TFTP boot

Faster Transfer of files from HOST PC to development board

During your development if you keep changing your boot binaries or file system, then **tftpboot** procedure will save lots of time related to transfer of those images for files to the board for testing.

You can automate tftpboot using uboot's uEnv.txt, that we will take up next



Next →

Challenges for you to complete

TFTP boot Conclusion

TFTP boot is a very handy way of booting when you need to recompile your kernel repeatedly and inserting removing the SD card from the board is a tedious process.

uEnv.txt from Scratch

**Lets Write and Test the uEnv.txt
from scratch !!**

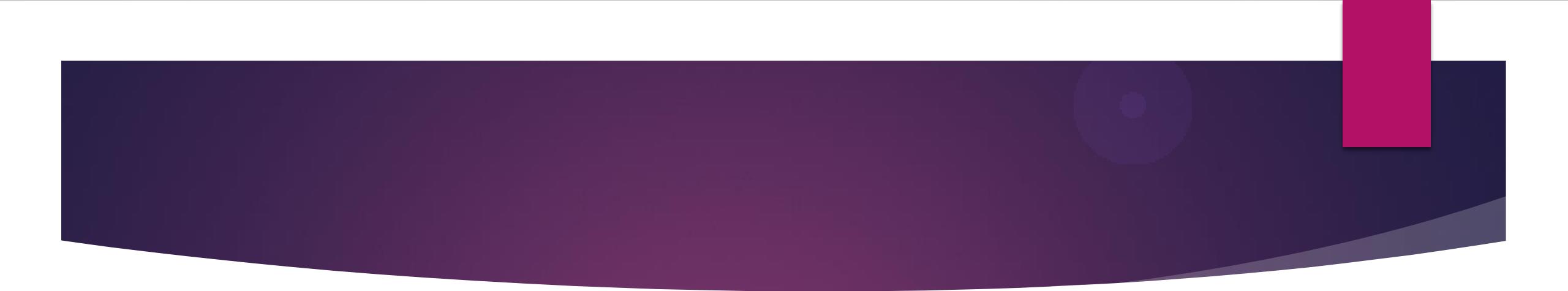
Similar to linux environment variables, u-boot also has set to standard as well as user defined environmental variables which can be used to override or change the behavior of the uboot



uEnv.txt file is nothing but collections of various env variables which are initialized to number of uboot commands and automating the command execution .

U-boot always try to read the uEnv.txt from the boot source, if uEnv.txt not found, it will use the default values of the env variables.

If you don't want u-boot to use default values, enforce new values using uEnv.txt



You can even execute all the commands stored in the env variable in one go !!! Use the command “run”

run name_of_env_variable

‘boot’ command of the u-boot does nothing but running of an env variable called “bootcmd”

So, if you want to change the behavior of “boot”, you have to change commands stored in the env variable “bootcmd”



**Did you know that you can store
multiple u-boot commands as a
value in the env variables and run ???**

U-boot's load from memory device commands

To load a file from **FAT** based file system in to memory use : **fatload**

To load a file from any file system : **load**

Example:

fatload usb 0:1 0x82000000 ulmage

fatload mmc 0:1 0x88000000 initramfs

load mmc 0:1 0x88000000 ulmage

Note: command syntax may change with newer versions of uboot, please run “help <command>” on the uboot for detailed explanation

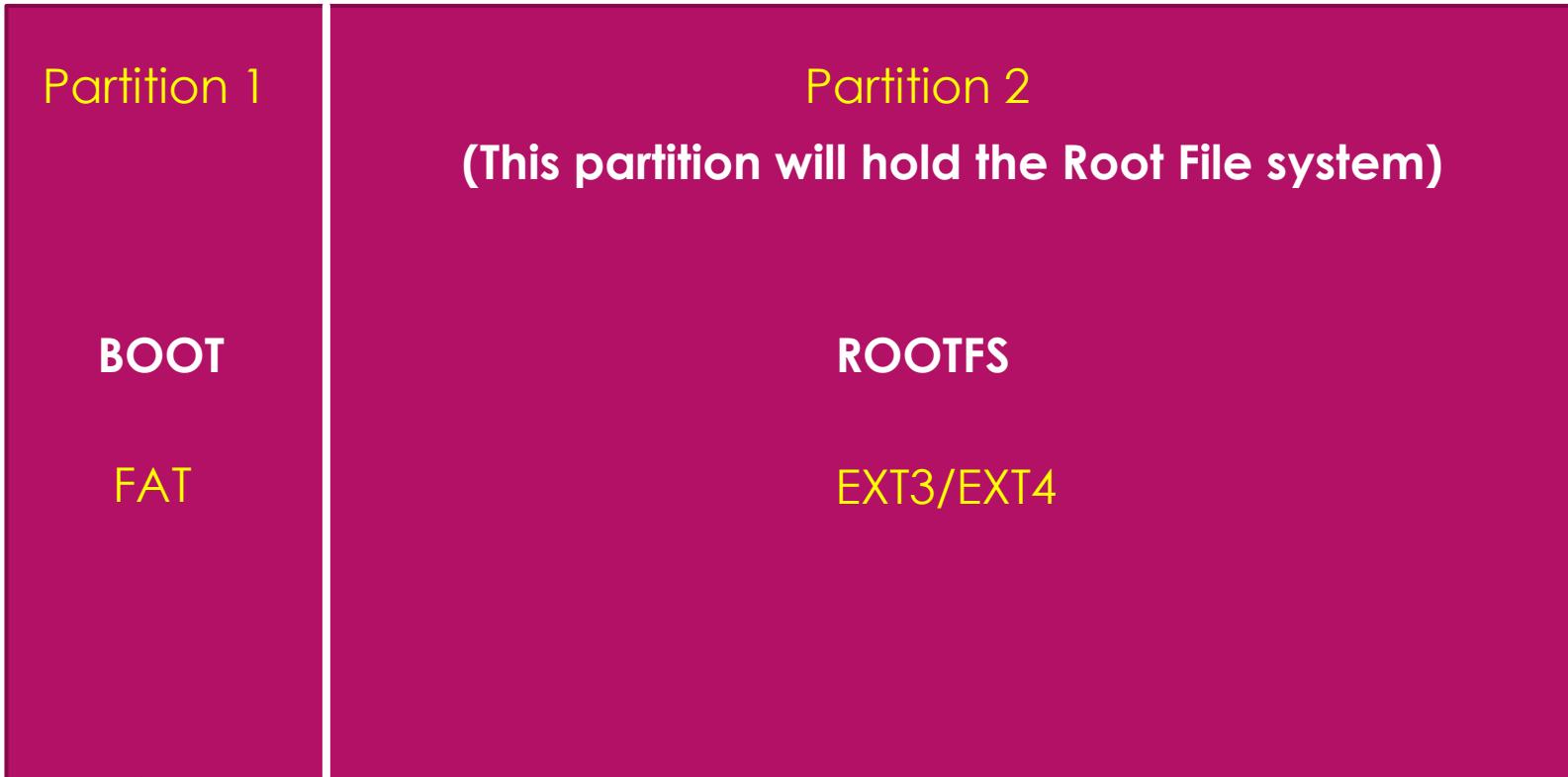
U-boot's load from memory device commands

Lets load the Linux binary image “ulmage” from the second partition of the on-board eMMC memory in to the DDR memory !

U-boot's serial port transfer protocols commands

<i>command</i>	<i>Details</i>
loadx	Send/receive file using xmodem protocol
loady	Send/receive file using ymodem protocol
loadz	Send/receive file using zmodem protocol

μSD card partitions



(This partition holds boot images)

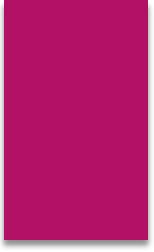
The kernel has no idea, which serial port of the board is used for sending the boot logs

The BBB uses UART0 as the serial debug terminal , which is enumerated as /dev/ttyO0 by the serial driver

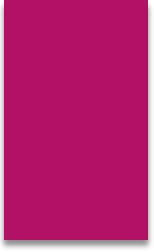
ttyO0 => tty “capital ohh”, zero



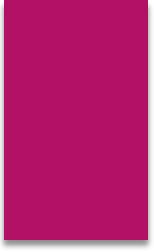
The boot again failed here because, linux has no idea from where exactly it should mount the filesystem. So, you have to send the location and type of file system using “bootargs”



**Lets mount the file system which is present
at the partition 2 of the Micro SD card.**



**Lets transfer this minimal uEnv.txt from host
to board using serial port tranfer protocols
like xmodem or ymodem.**



**Oops !!! This is an older version of u-boot,
the “loadx” command is not supported,
lets try “loady”.**

Oops !!! Again we did something wrong !

Sorry ☺



**The transfer failed , that's because, u-boot's “loady” command has timed out.
So you shouldn't be giving more time
after the execution of “loady” command.**



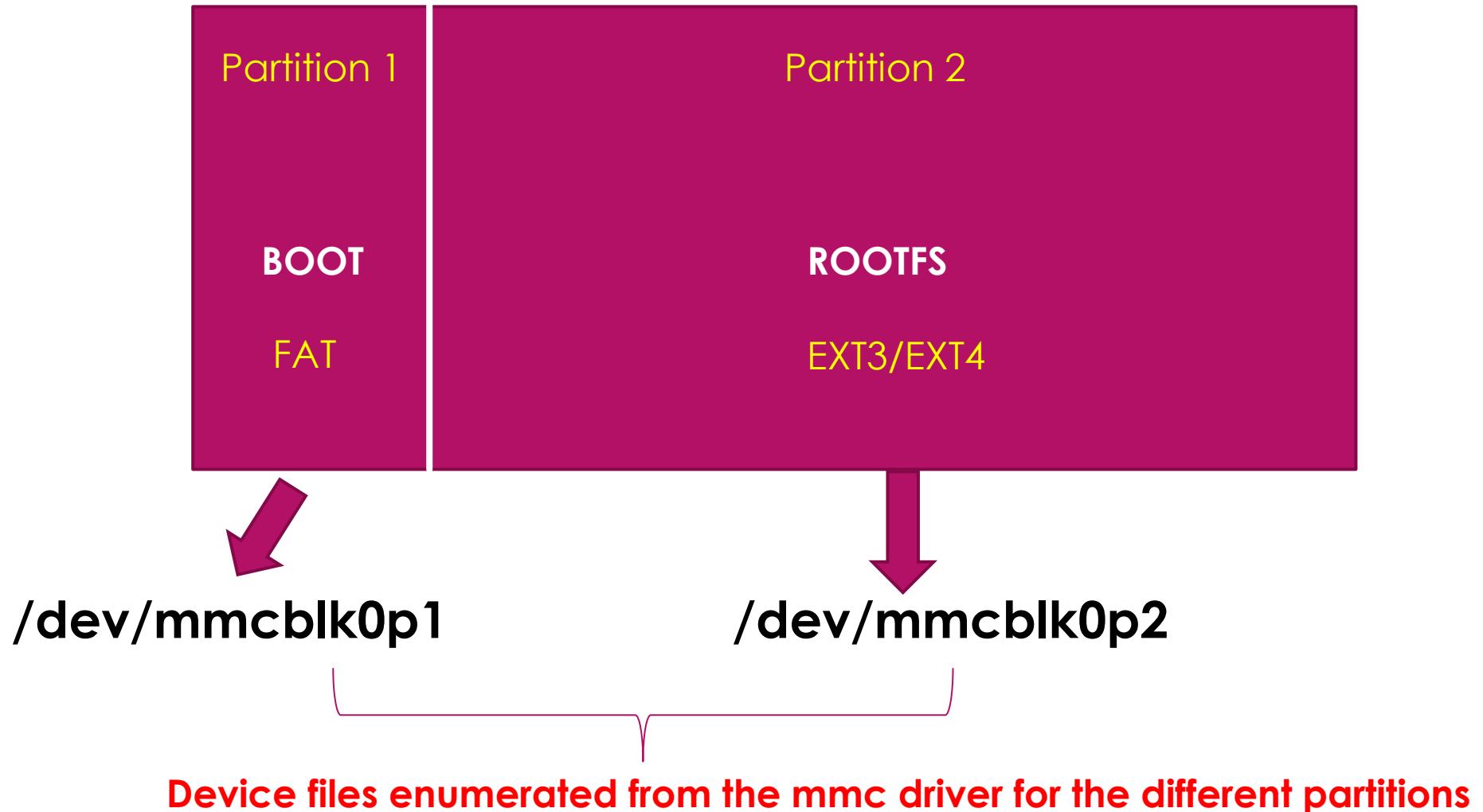
In this case, we loaded the boot images from MMC1 interface(eMMC) and used the file system present on the MMC0(MiroSD card). Challenge for you is to change this uEnv.txt such that boot images are loaded from MMC0(MicroSD).

Default username and password for BBB Debian OS Login

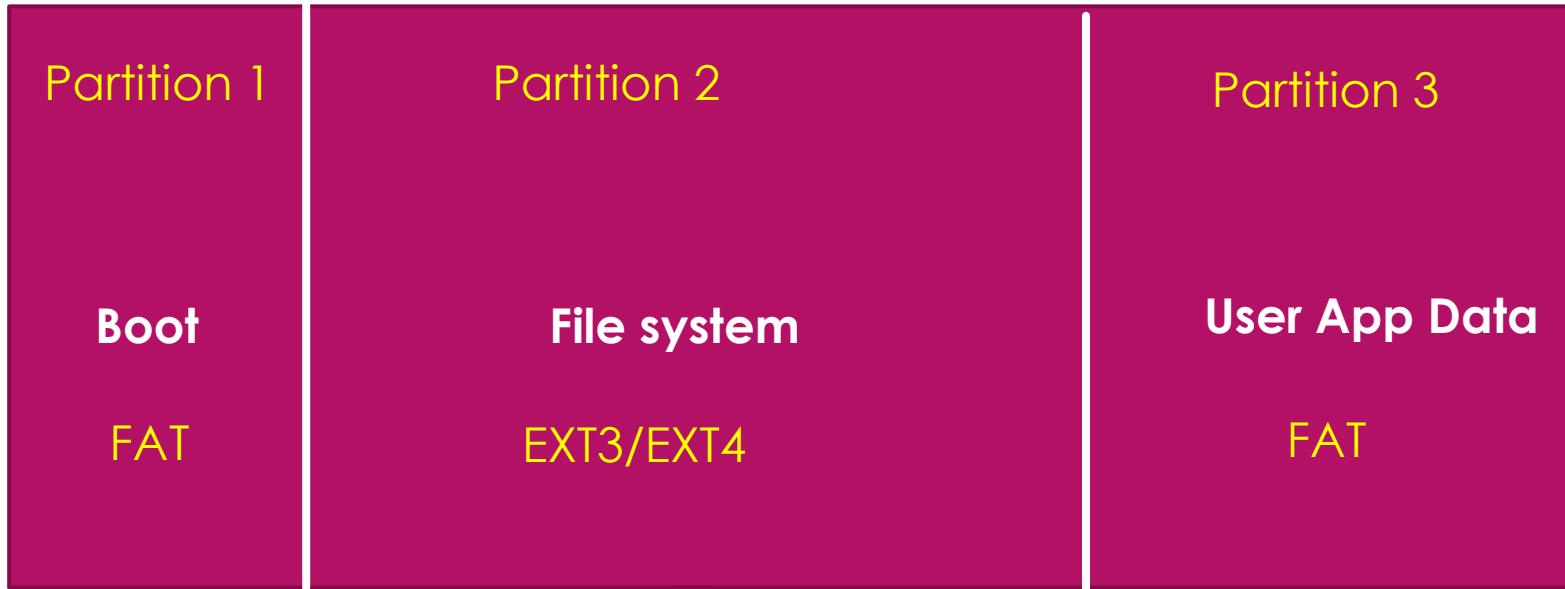
Username : debian

Password : temppwd

MiroSD Card Partitions



eMMC Memory Partitions



Holds boot images

Holds the linux
File system

Carries user data



P4
Mini usb port



Next →

BBB initial setup using UBUNTU Host

Exploring BBB Web Interface



**The BBB Debian OS which is running on
your board already running a web server.
Lets connect to that !!!**

SYSFS, as the name indicates, its nothing but a file system

It is on the fly filesystem and exists on the RAM

It's a kind of window to peek in to the various subsystem of the linux kernel like networking, memory subsystem, bus interfaces, hardware devices and device drivers, etc.

SYSFS

- ▶ **SYSFS, as the name indicates, its nothing but a file system**
- ▶ **It's on the fly file system and exists on the RAM**
- ▶ **It's a kind of window for the user space to peek in to the various subsystem of the Linux kernel like networking, memory subsystem, bus interfaces, hardware devices and device drivers, etc.**

SYSFS

We will write lots of ‘C’ applications to access peripherals via SYSFS, like LCD,EEPROM,I2C sensor, 7segment display,SPI,ADC,etc in this course.

Important note

If your board already running latest version of debian OS image , then you NEED NOT to try this on your board, just watch the video .

check your BBB debian OS version : `lsb_release -da`
Latest release of BBB debian os :
<https://beagleboard.org/latest-images>

User Space

Kernel space details are exposed under /sys for the user applications.

```
debian@beaglebone:/sys$ ls  
block bus class dev devices firmware
```

Kernel Space

Serial devices

Parallel devices

Display subsystem

Networking

USB

Memory

Bus

IO



Flashing BBB's eMMC
using Angstrom eMMC
flasher

Flashing BBB's eMMC using Angstrom eMMC flasher

In the previous video , we flashed the debian image on to the eMMC memory of the beaglebone hardware. So, our board must be containing debian images at this time.

Important Note

The following step will destroy the debian image present on your board eMMC(of course you can again flash it). I suggest, NOT to try angstrom flashing on your board unless there is any specific reason.

Flashing BBB's eMMC using Angstrom eMMC flasher

For windows:

**Use the win32disk manager software to write this image
on to the sd card**

For Ubuntu

```
sudo if=BBB-eMMC-flasher-v2013.06-2013.11.20.img  
of=<your sd card device file name>
```

Linux Device Tree (Flattened Device Tree)

Linux Device Tree Lectures

- 1.What Problem does this device tree solve ?**
- 2.How to generate the Device Tree Binary(DTB) ??**
- 3.Device Tree Source(DTS) File writing syntax**
- 4.Writing DTS File for BBB**

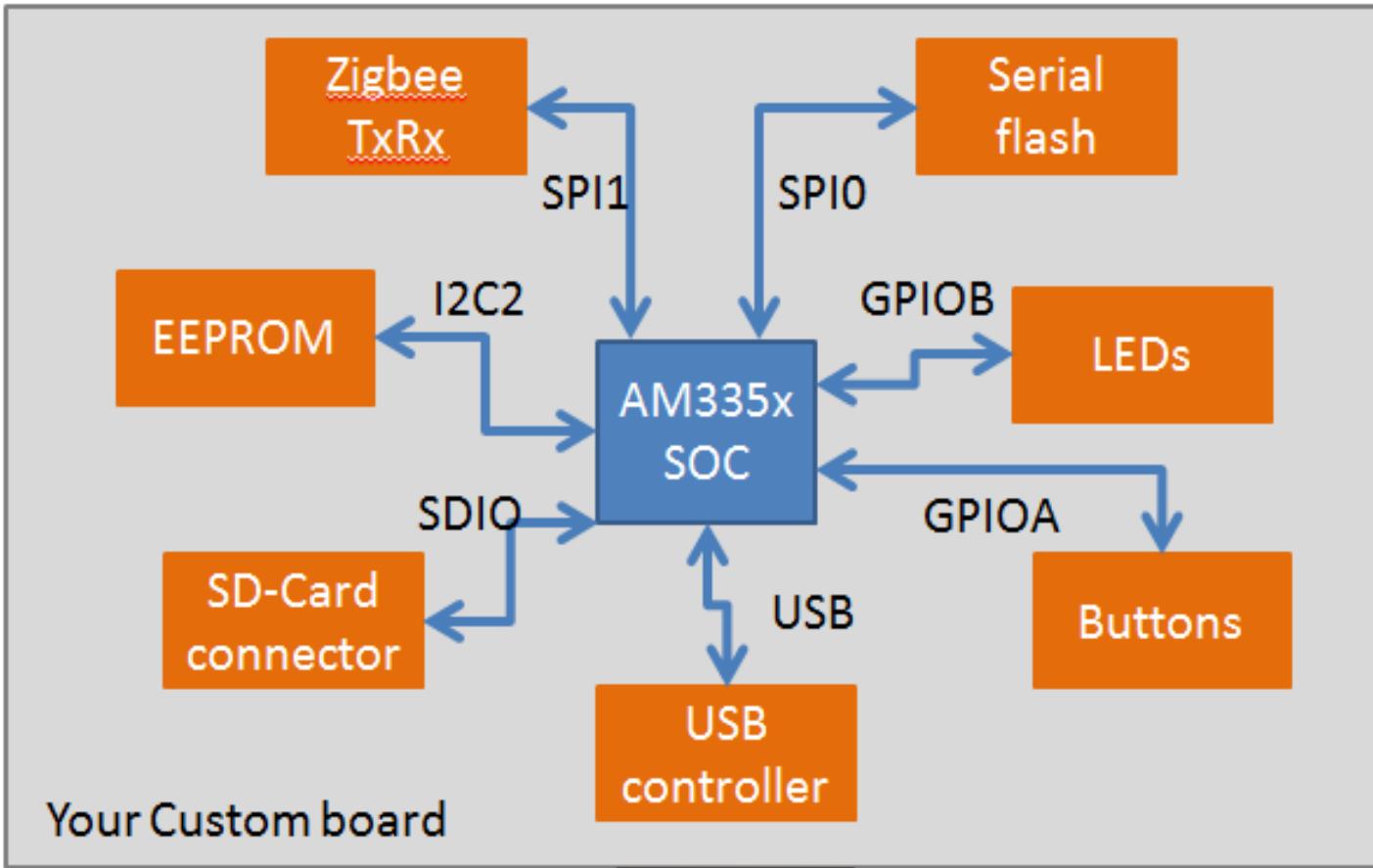


Why Device Tree is introduced ??

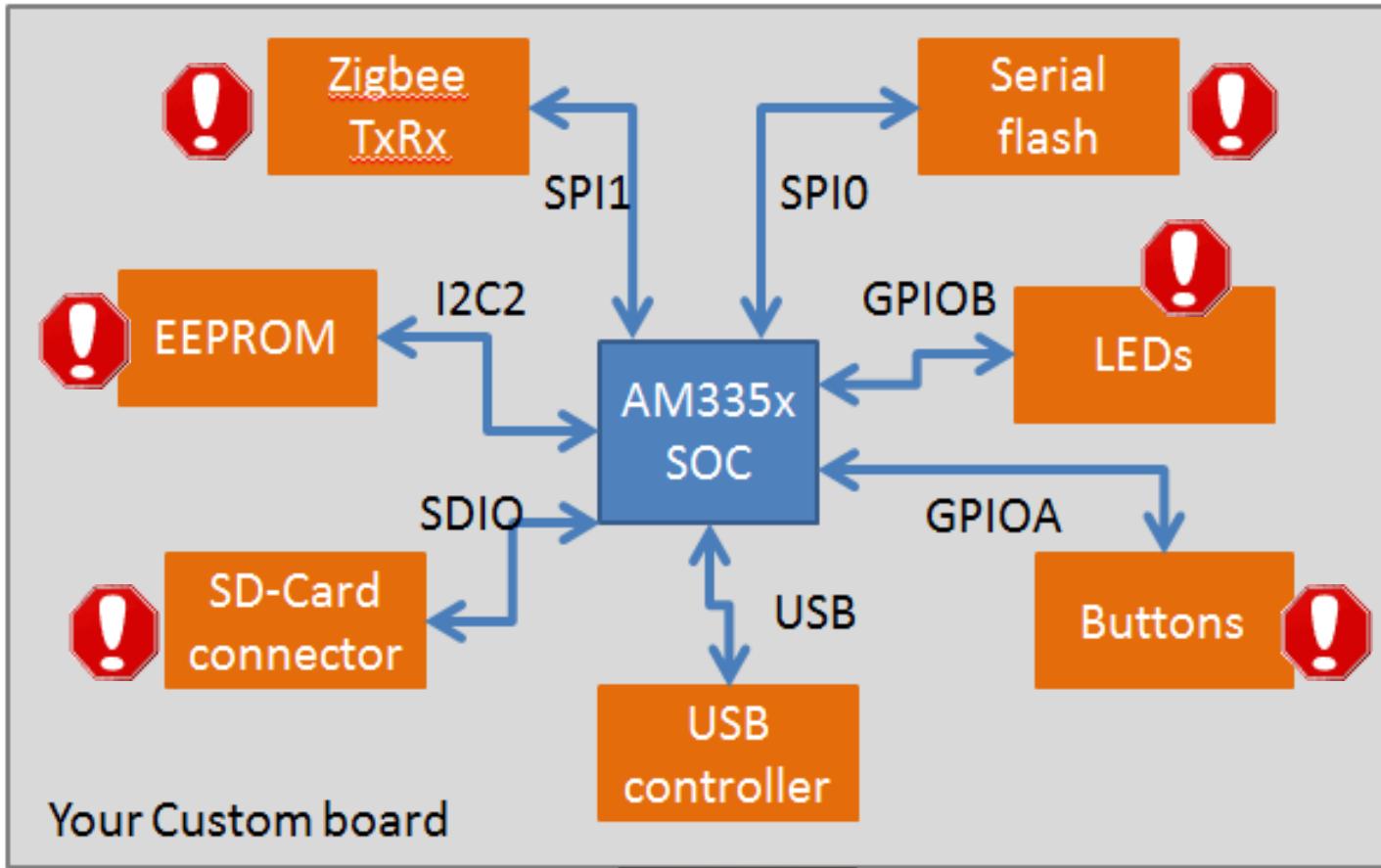
What problem it solves ??



On-Board
Peripherals



External Peripheral



On-Board
Peripherals



Non-Discoverable by itself



Self Discoverable



External Peripheral

Platform devices

The on-board peripherals which connect to SPI,I2C,SDIO,ETHERNET,etc have no capability to announce there existence on the board by themselves to the operating system. These peripherals are also called as “Platform devices”



**How can we make Linux kernel
know about these platform
devices ?**

Zigbee
TxRx

SD-Card
connector

EEPROM

Serial
flash

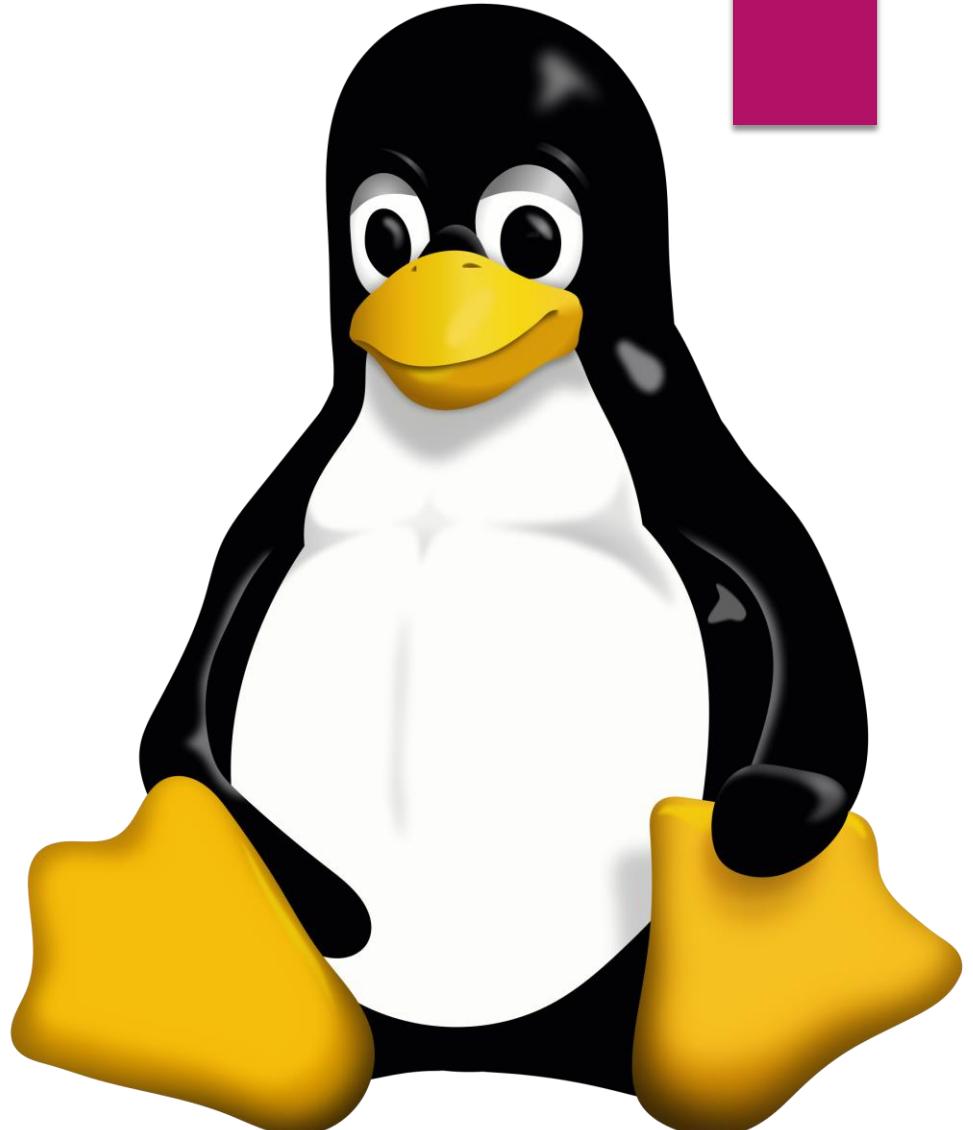
**Non-Self discoverable
peripherals(aka. platform
devices)**

LEDs

Buttons



*These are my platform devices
Please make a list !!!!*



```
static void my_board_init(void)
{
    /* Serial */
    my_board_add_device_serial(&serial_data);

    /* SPI */
    my_board_add_device_spi(&zigbee_data);
    my_board_add_device_spi(&serialflash_data);

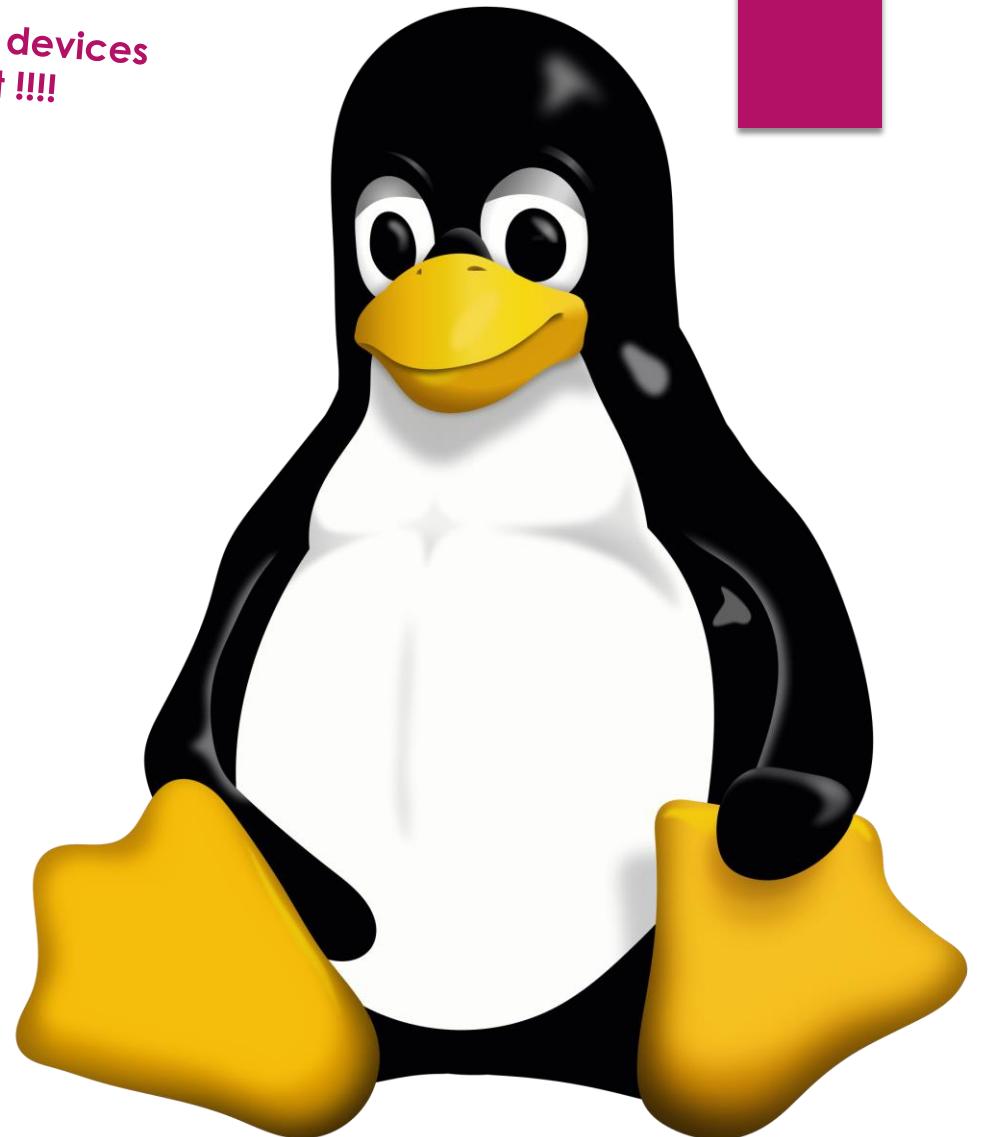
    /* Ethernet */
    my_board_add_device_eth(&eth_data);

    /* I2C */
    my_board_add_device_i2c(&eeprom_data);

    /* LEDs */
    my_board_add_device_gpios(&leds_data);
    ...
}
```

My-board-config.c

These are my platform devices
Please make a list !!!!



```
static void my_board_init(void)
{
    /* Serial */
    my_board_add_device_serial(&serial_data);

    /* SPI */
    my_board_add_device_spi(&zigbee_data);
    my_board_add_device_spi(&serialflash_data);

    /* Ethernet */
    my_board_add_device_eth(&eth_data);

    /* I2C */
    my_board_add_device_i2c(&eeprom_data);

    /* LEDs */
    my_board_add_device_gpios(&leds_data);

    .
    .
    .
}
```

Platform data

my-board-config-file.c
Contains the function called
“**my_board_init**”,
which manually adds all the platform
devices to the linux subsystem along
with the corresponding platform data

zigbee100	at256eprom	gpioleds
sdcard001	lcd	

**Platform
device names**

Platform device data base of linux
(added by the board file)

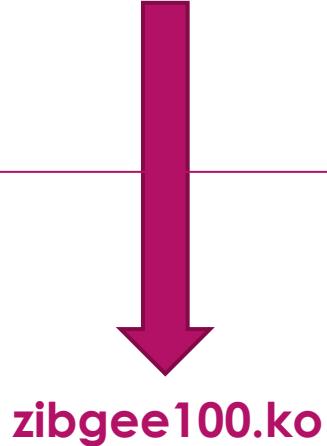
zigbee100.ko
at256eprom.ko
gpioleds.ko
sdcard001.ko
lcd.ko

**Corresponding platform drivers to handle
those devices**

insmod zibgee100.ko

User Space

Kernel Space



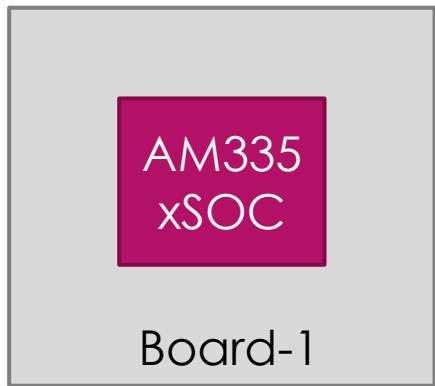
zigbee100	at256eprom	gpioleds
sdcard001	Lcd	

**Platform device data base of linux
(added by the board file)**

```
Probe_fun_of_zigbee100_driver(void *data)
{
```

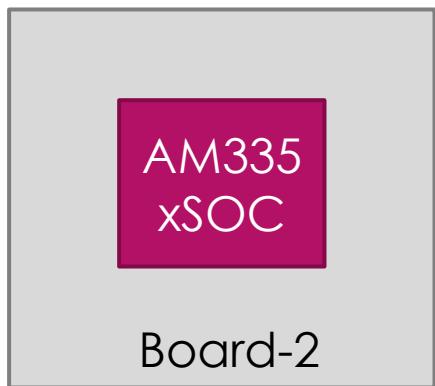
```
//here initialize the device
```

```
}
```



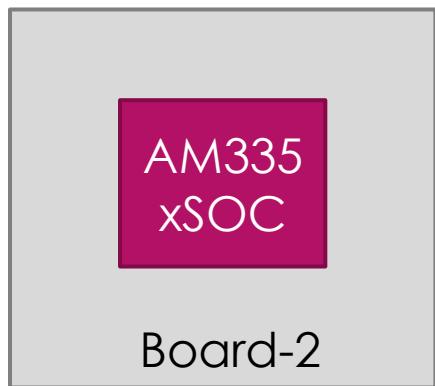
board1-config.c

ulmage-Board-1



board2-config.c

ulmage-Board-1



board3-config.c

ulmage-Board-1

Why Device Tree is introduced ??

The Linux community wanted to cut off the dependencies of platform device enumeration from the Linux kernel , that is, hard coding of platform device specific details in to the Linux kernel.



Board1.dts

```
/ {
    compatible = "ti,am335x-bone", "ti,am33xx";
    [...]
    ocp {
        uart0: serial@44e09000 {
            compatible = "ti,omap3-uart";
            reg = <0x44e09000 0x2000>;
            interrupts = <72>;
            pinctrl-names = "default";
            pinctrl-0 = <&uart0_pins>;
            status = "okay";
        };
    };
}
```



Board1.dtb



Board2.dts

```
/ {
    compatible = "ti,am335x-bone", "ti,am33xx";
    [...]
    ocp {
        uart0: serial@44e09000 {
            compatible = "ti,omap3-uart";
            reg = <0x44e09000 0x2000>;
            interrupts = <72>;
            pinctrl-names = "default";
            pinctrl-0 = <&uart0_pins>;
            status = "okay";
        };
    };
}
```



Board2.dtb



Board3.dts

```
/ {
    compatible = "ti,am335x-bone", "ti,am33xx";
    [...]
    ocp {
        uart0: serial@44e09000 {
            compatible = "ti,omap3-uart";
            reg = <0x44e09000 0x2000>;
            interrupts = <72>;
            pinctrl-names = "default";
            pinctrl-0 = <&uart0_pins>;
            status = "okay";
        };
    };
}
```



Board3.dtb

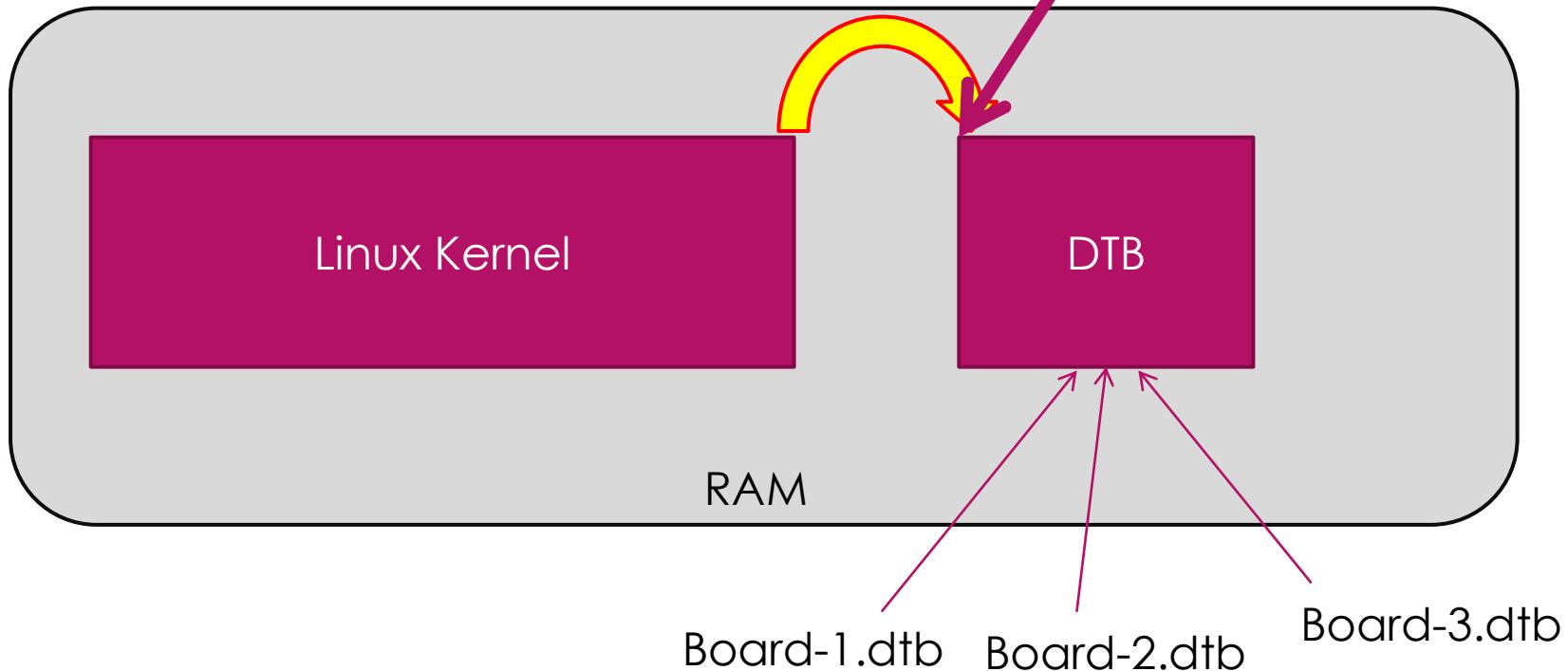
Device Tree Source File (DTS)

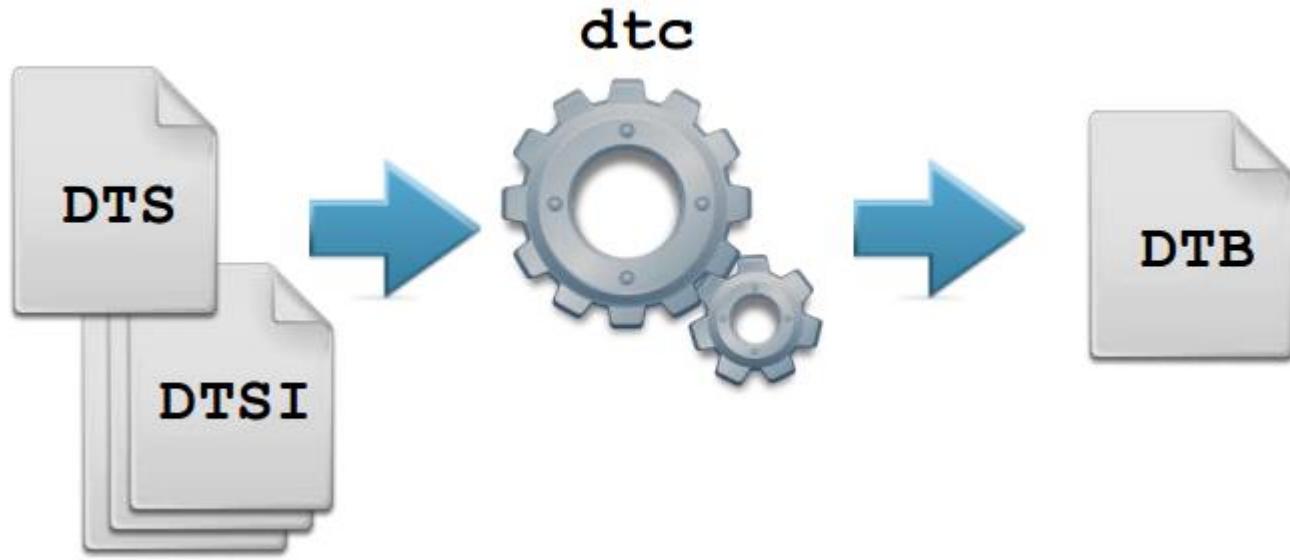
Instead of adding hard coded hardware details in to the linux kernel board file, every board vendors has to come up with a file called DTS.

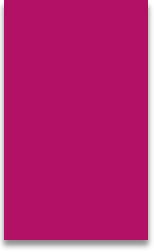
This file actually consists of all the details related to the board written using some pre defined syntaxes. So you can say that this file consists of data structures which describe all the required peripherals of the board.

**During booting linux must
know the address at which
DTB is present in the RAM**

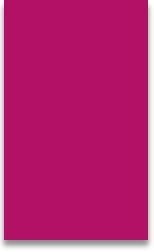
Load address of DTB in RAM





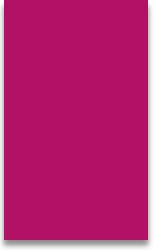


USB device has the inbuilt intelligence to send its details to the operating system.(that means USB supports dynamic discoverability)

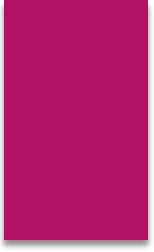


Connect your BBB to PC now via the mini USB cable.

No need to connect Ethernet cable

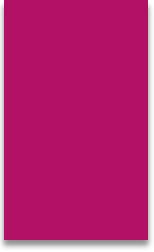


When a driver for a particular platform peripheral is loaded, the Linux calls the "probe" function of the driver if there is any match in its platform device database. In the "Probe" function of the driver, you can do device initializations.



Remember “busybox” binary in this case is statically linked and showing total size of 1.35MB.

Later when we generate “busybox” with dynamic linking, the size must be < 1.35MB.



Remove the SD card
Connect SD card to BBB
Open Minicom
Boot from SD card

Note:

**DO NOT do these steps if you have already
flashed the eMMC in the last lecture(Windows
based).**

This lecture is only for UBUNTU users

Flashing Completed.....

Remove power from the board

Take out your SD card from the board

Power up the board back again

Flashing Completed..... (board is in power down mode)

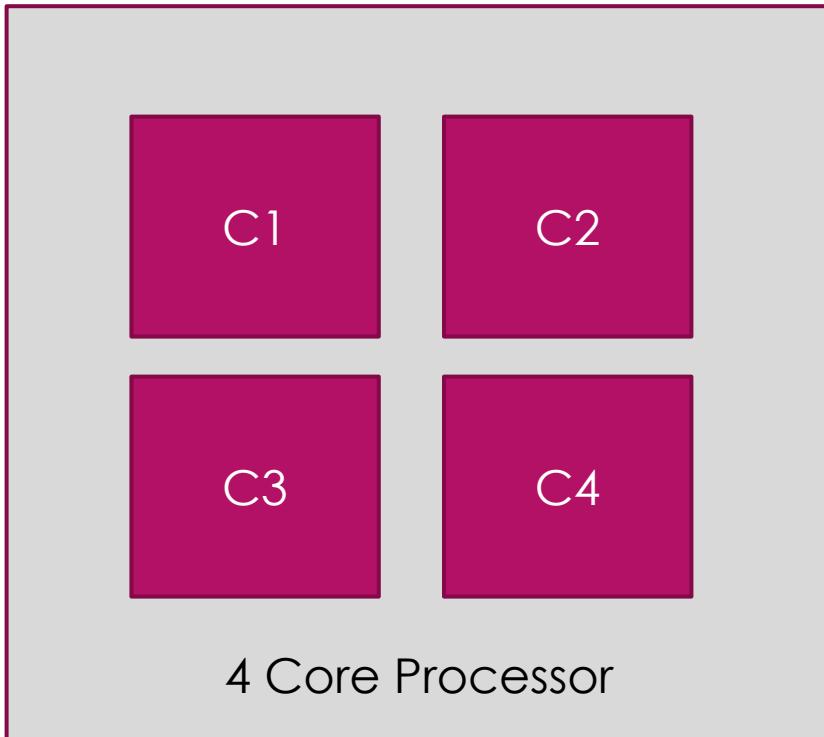
**1) Take out your SD card from the board
2) Press and hold the S3 (power) button for few(~5) seconds.**

OR

2) You can also plug out the mini usb cable from PC and plug in back ..

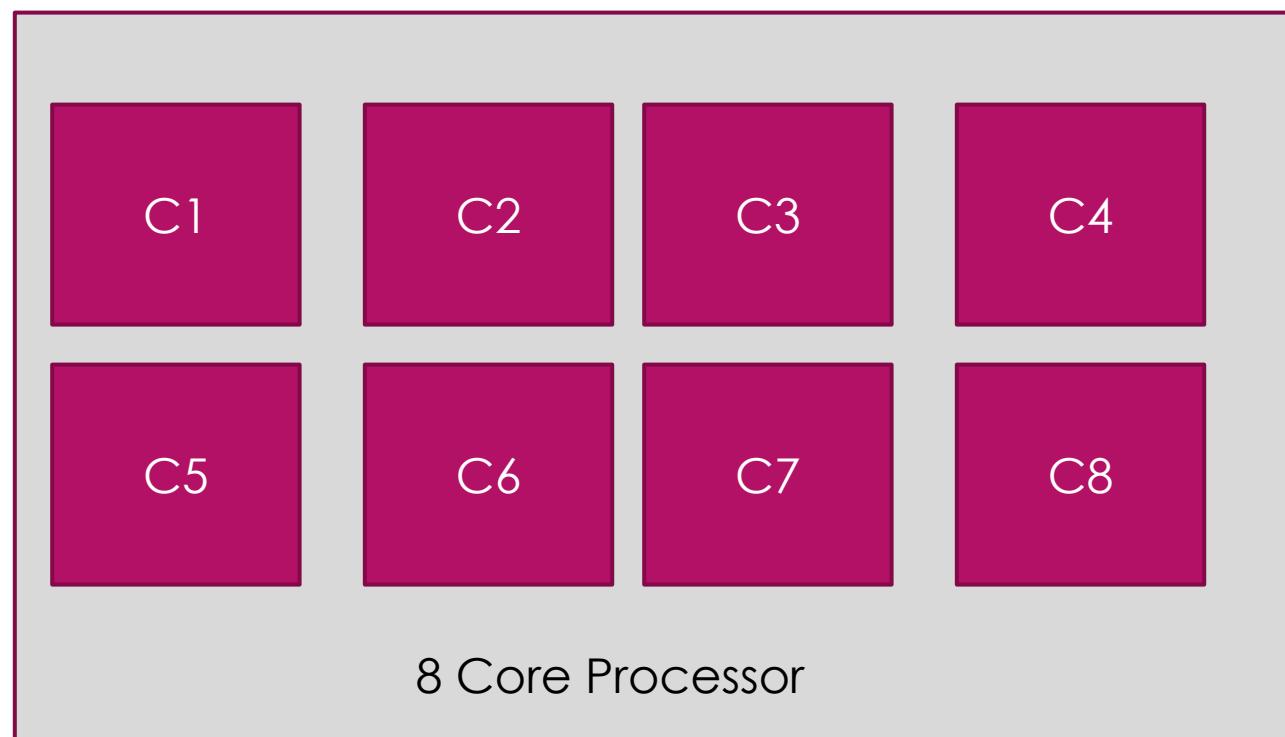
-j4

**4 Threads will be spawned for compilation
1 per core**



-j8

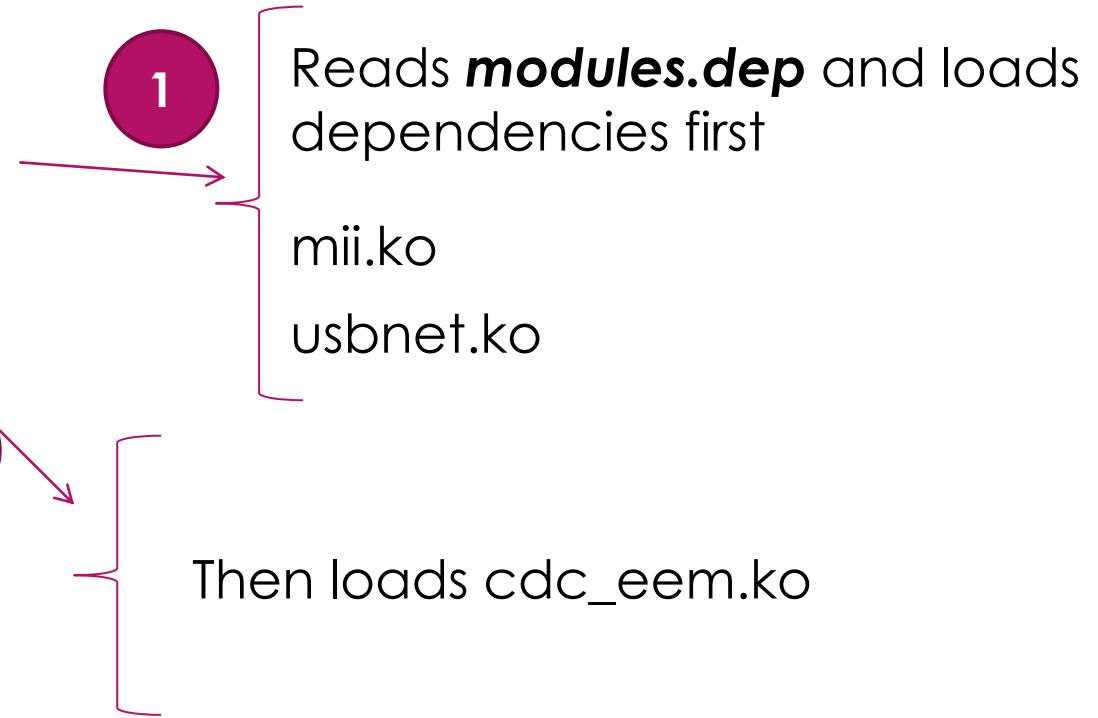
**8 Threads will be spawned for compilation
1 per core**



Next→

Linux Kernel Compilation

modprobe cdc_eem.ko



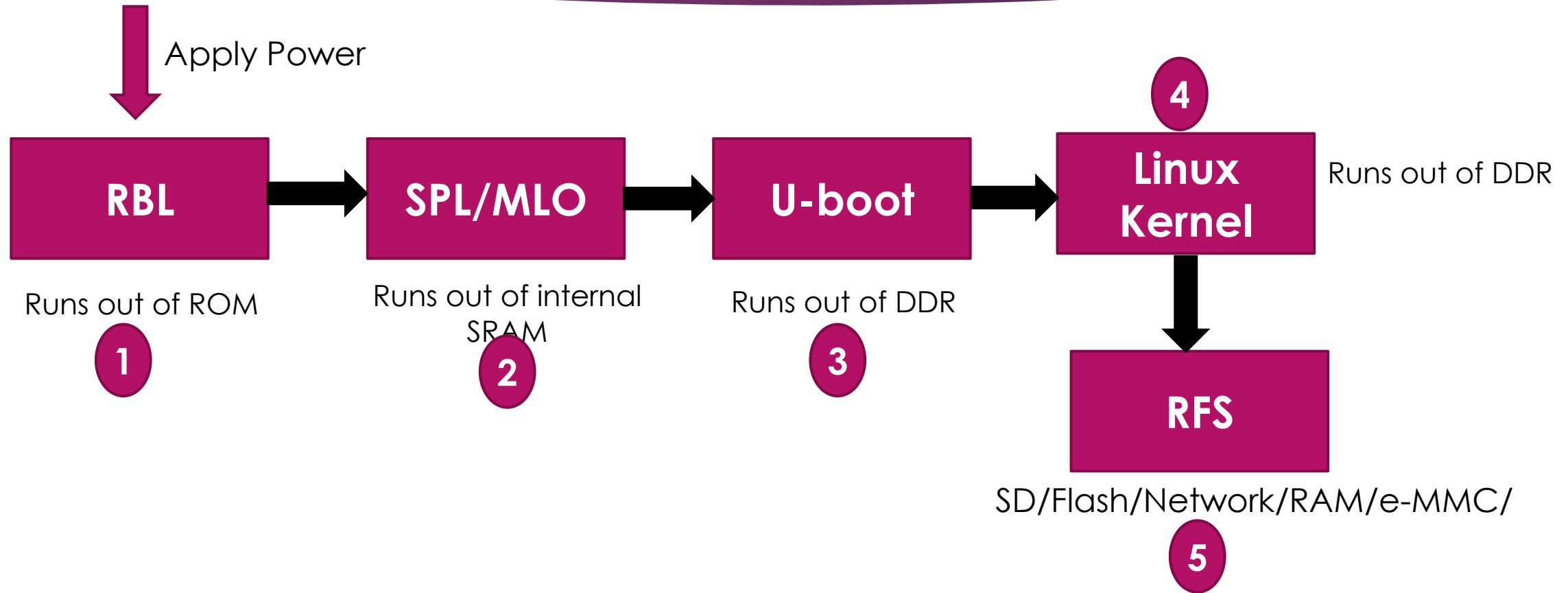
Next→

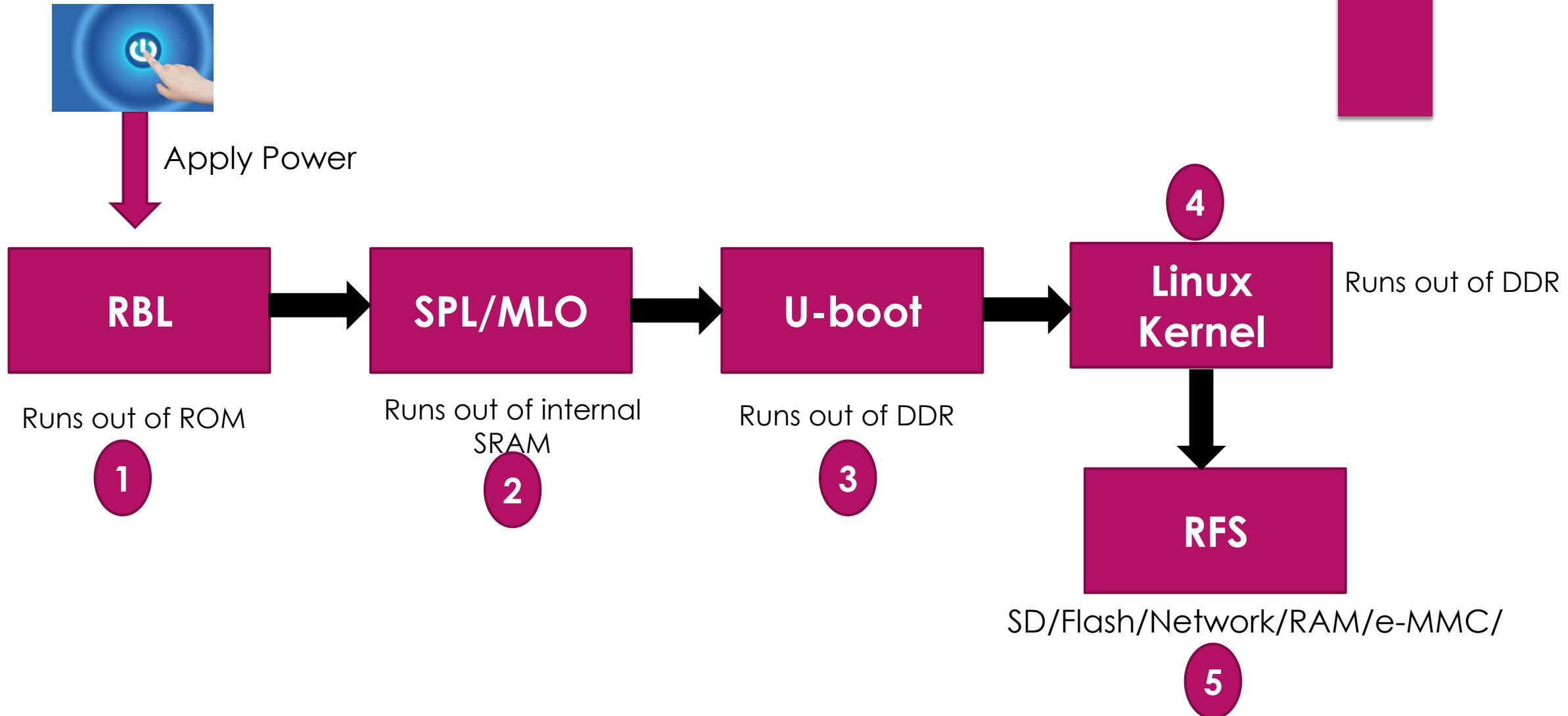
BBB eMMC flashing using Angstrom Linux

For the Successful boot of BBB, we need

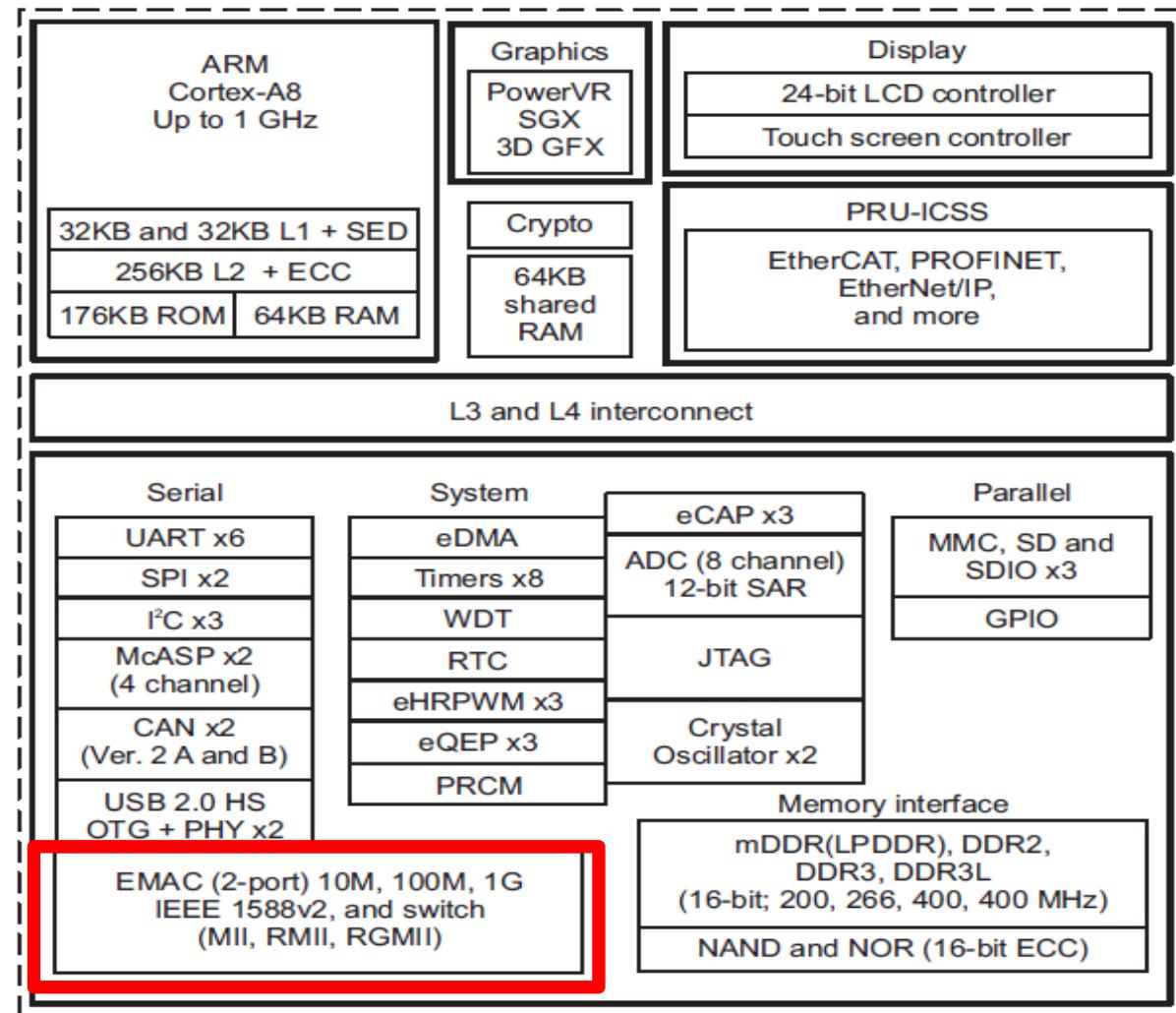
- 1.RBL**
- 2.SPL or MLO**
- 3.U-boot(or any similar bootloader)**
- 4.Linux Kernel**
- 5.Root File System(RFS)**

For the Successful boot of BBB, we need



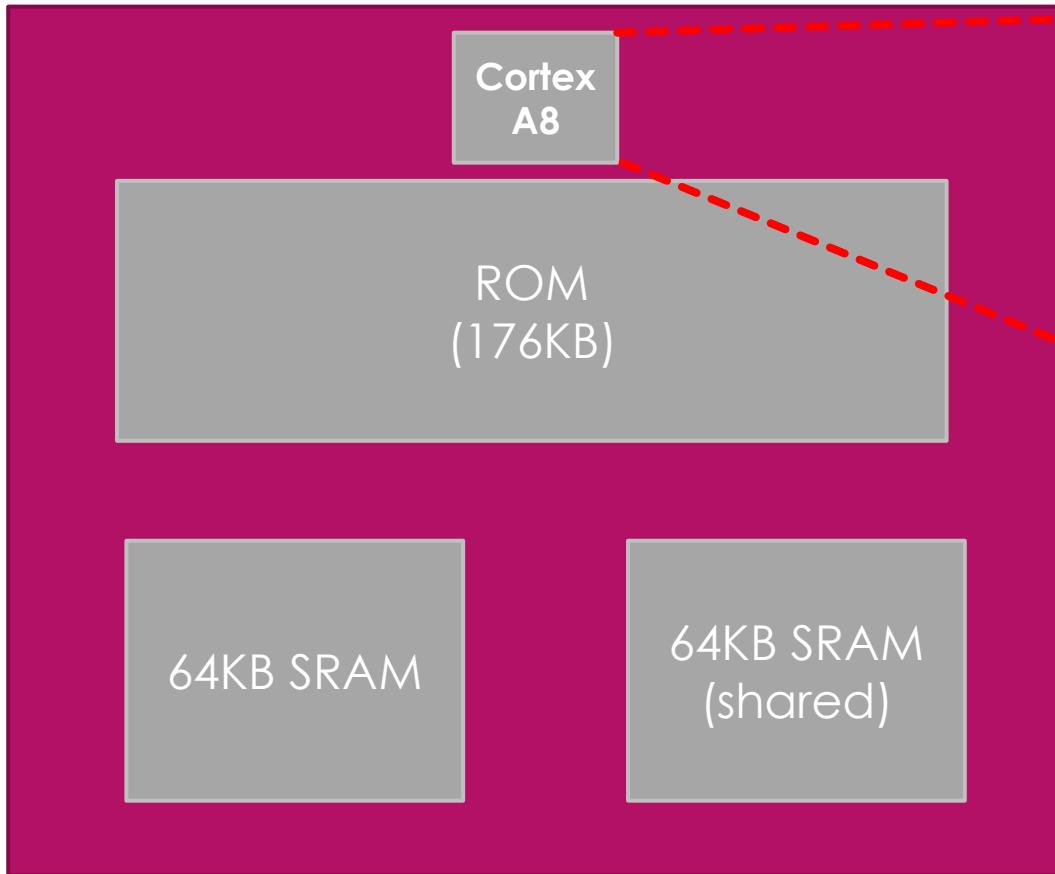


Functional Block Diagram of AM335x

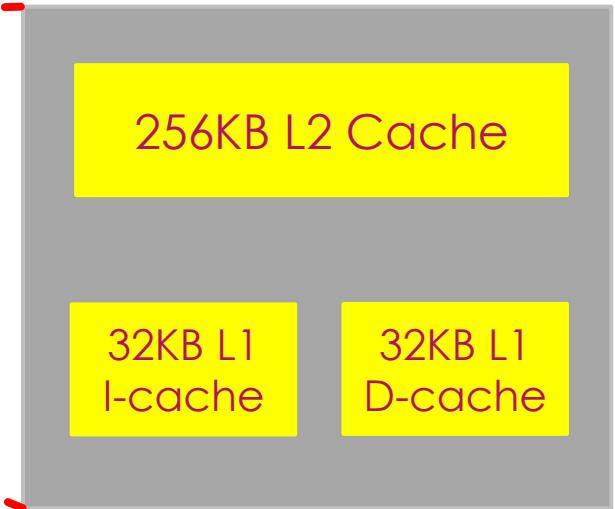


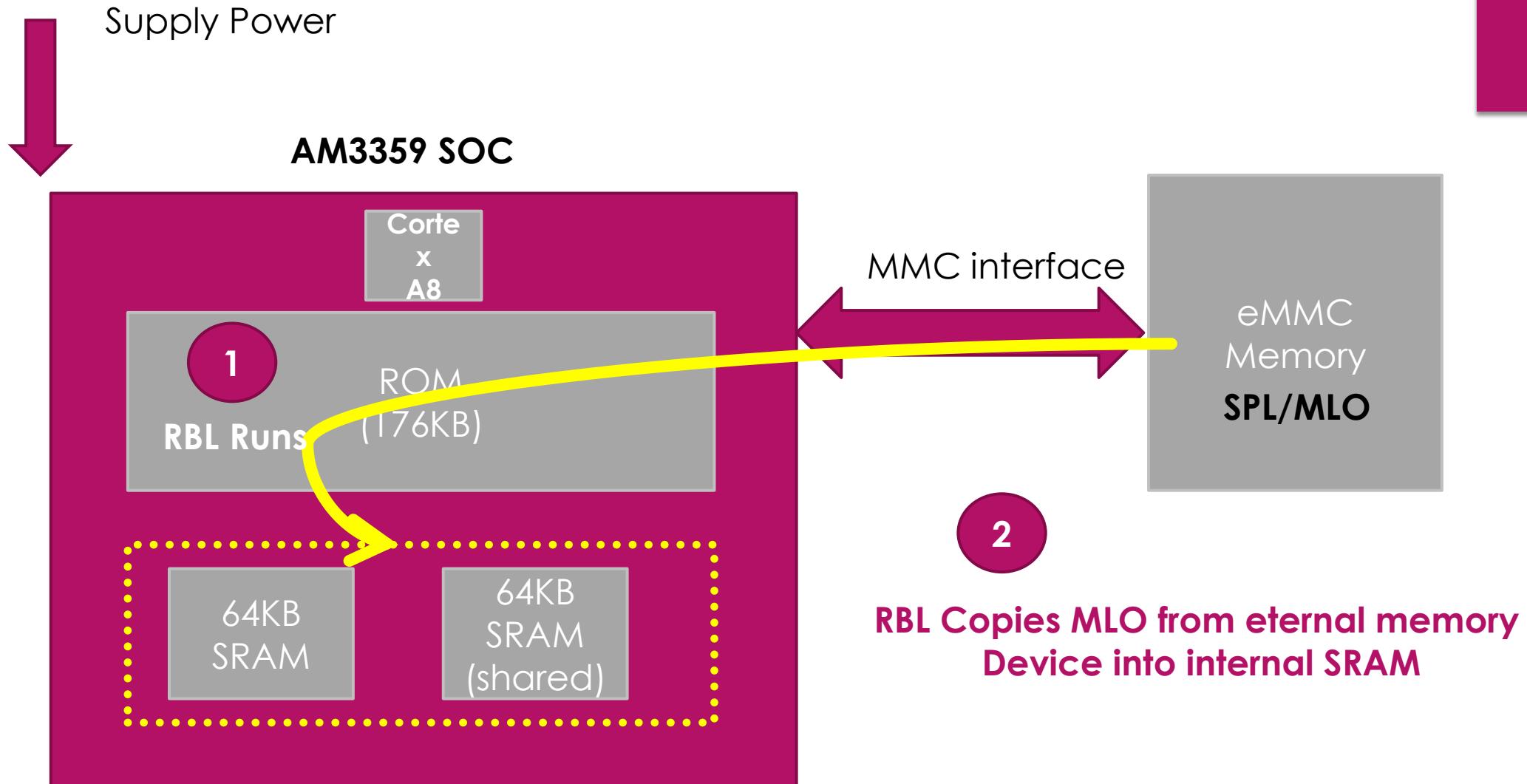
Copyright © 2016, Texas Instruments Incorporated

AM3359 SOC

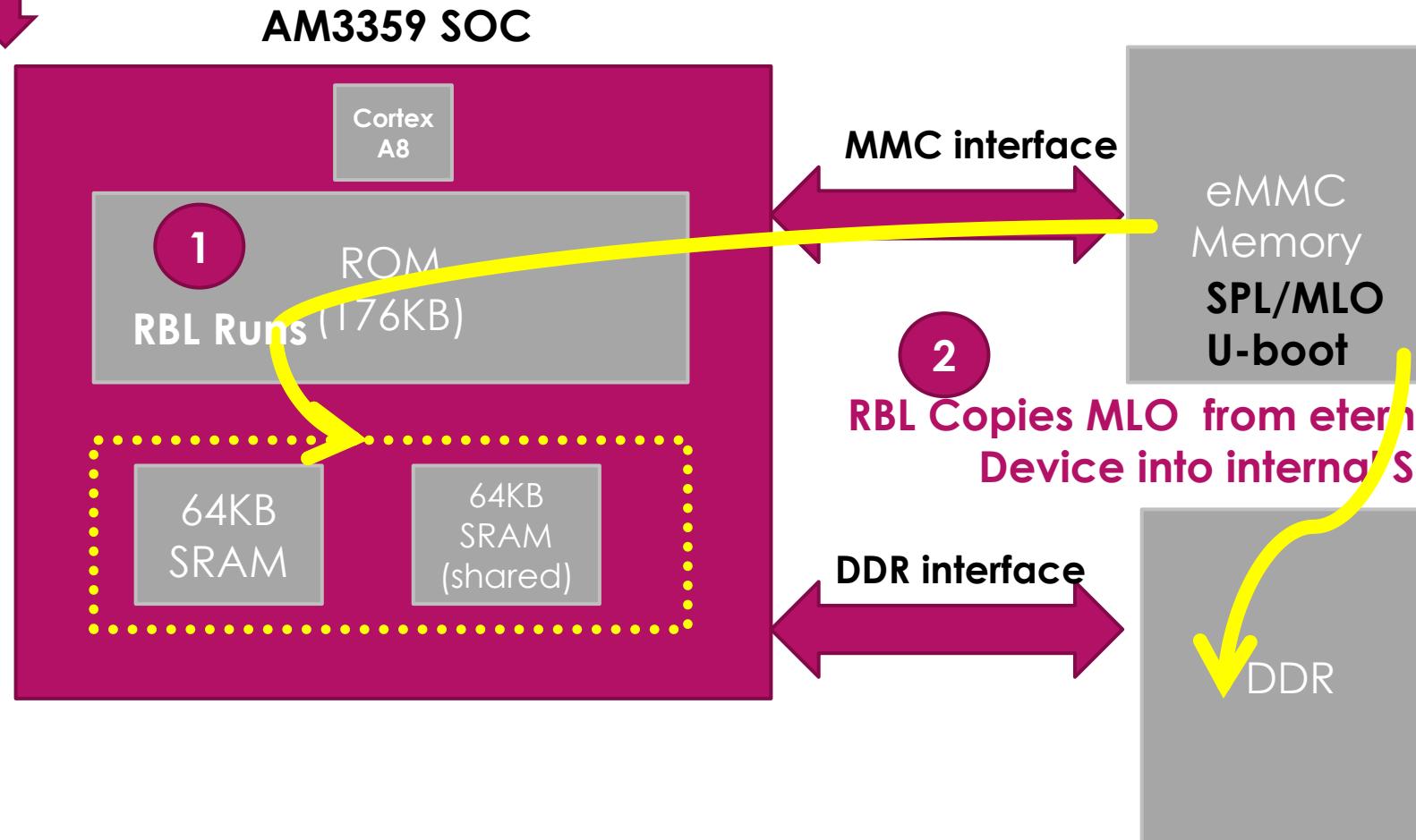


ARM Cortex-A8





Supply Power



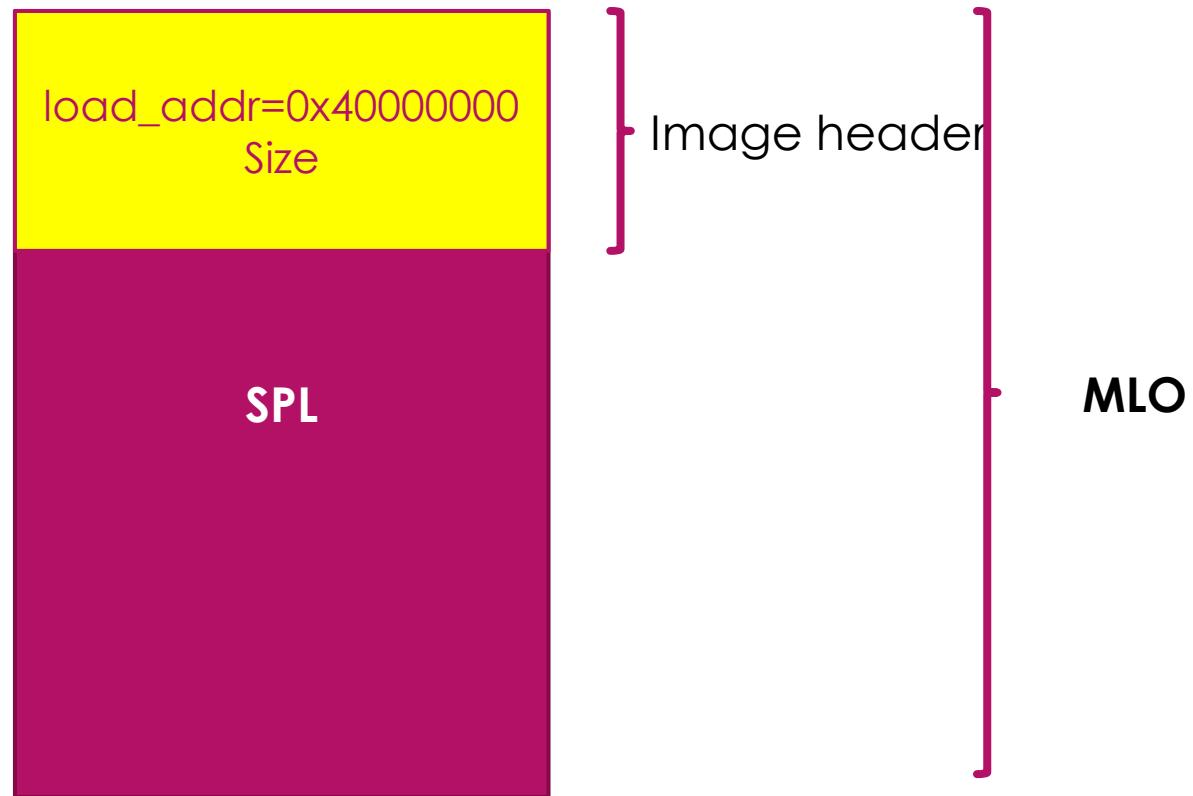
SPL



U-boot header info



MLO



SPL and MLO are almost same except the fact that, MLO has a header which contains some info like load address, size of the image to follow,etc

Summary : RBL

- ▶ **Stack Setup**
- ▶ **Watchdog timer 1 configuration (set to 3 minutes timeout),**
- ▶ **System clock configuration using PLL**
- ▶ **Search Memory devices or other bootable interfaces for MLO or SPL**
- ▶ **Copy “MLO” or “SPL” in to the internal SRAM of the chip**
- ▶ **Execute “MLO” or “SPL”**

RBL Code Initialization



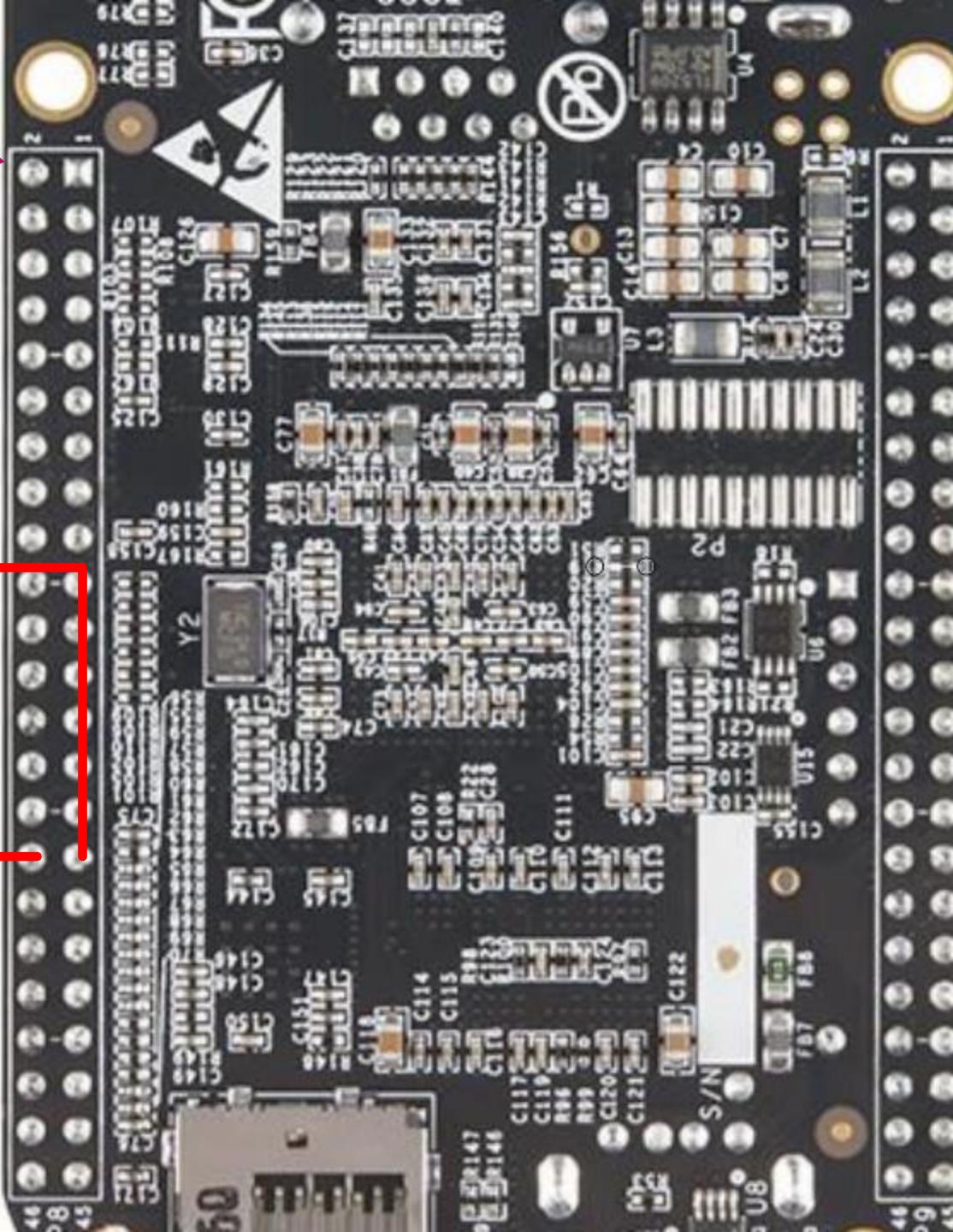
P8 expansion header

+VCC

SYS_BOOT14 (LCD_DATA14)

SYS_BOOT15 (LCD_DATA15)

GND



MLO/SPL job

**MLO or SPL initializes the SOC to a point that ,
U-boot can be loaded into external RAM(DDR
Memory)**

MLO/SPL Job

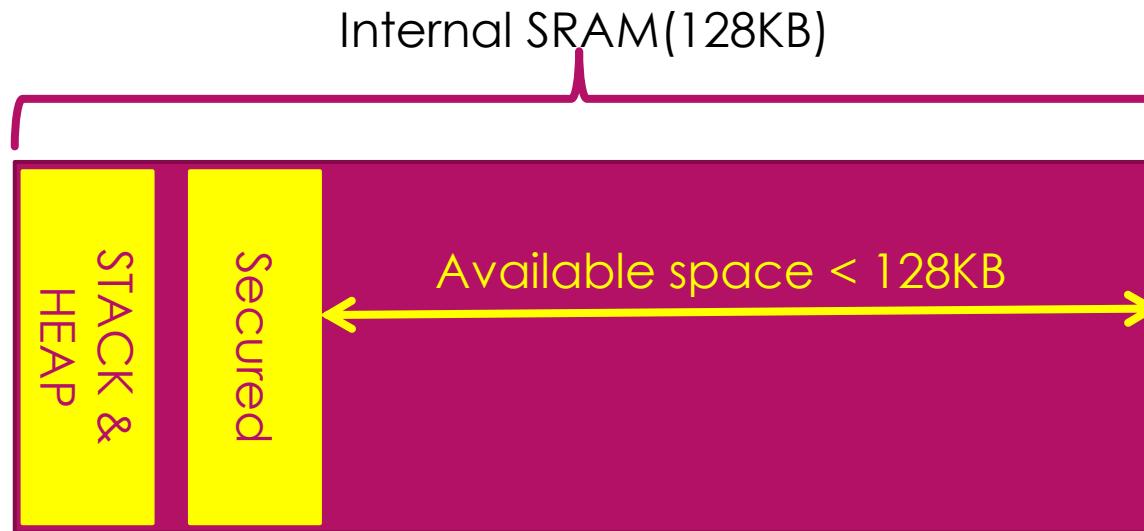
1. It does UART console initialization to print out the debug messages.
2. Reconfigures the PLL to desired value .
3. Initializes the DDR registers to use the DDR memory
4. Does muxing configurations of boot peripherals pin, because its next job is to load the u-boot from the boot peripherals
5. eg. If MLO is going to get the u-boot from the MMC0 or MMC1 interfaces, then it will do the mux configurations to bring out the MMC0 or MMC1 functionalities on the pins.
6. Copies the u-boot image into the DDR memory and passes control to it.

MLO/SPL Job

**Can we avoid using MLO or SPL by making
RBL directly loads the U-boot in to internal
SRAM ?**



Can we avoid using MLO or SPL by making RBL directly loads u-boot in to internal SRAM ?

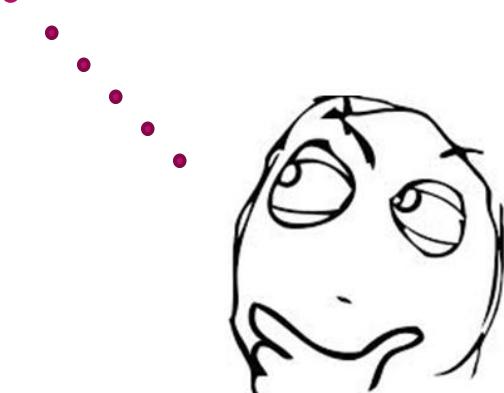


In order to load and run u-boot from the internal SRAM, you have to squeeze u-boot image to <128KB of size.

This is the reason why we use MLO or SPL

MLO/SPL Job

**Can we use RBL to load U-boot directly in
to DDR Memory of the Board and skip
using MLO or SPL ???**



**Can we use RBL to load U-boot directly in to DDR
Memory of the Board and skip using MLO or SPL ???**

Summary : MLO/SPL Job

1. **Reconfigures the PLL and also initializes the DDR**
2. **Configures PinMux for Boot Peripherals**
3. **Copies u-boot image to DDR at the load address by the u-boot image header.**
4. **Passes execution control to u-boot**

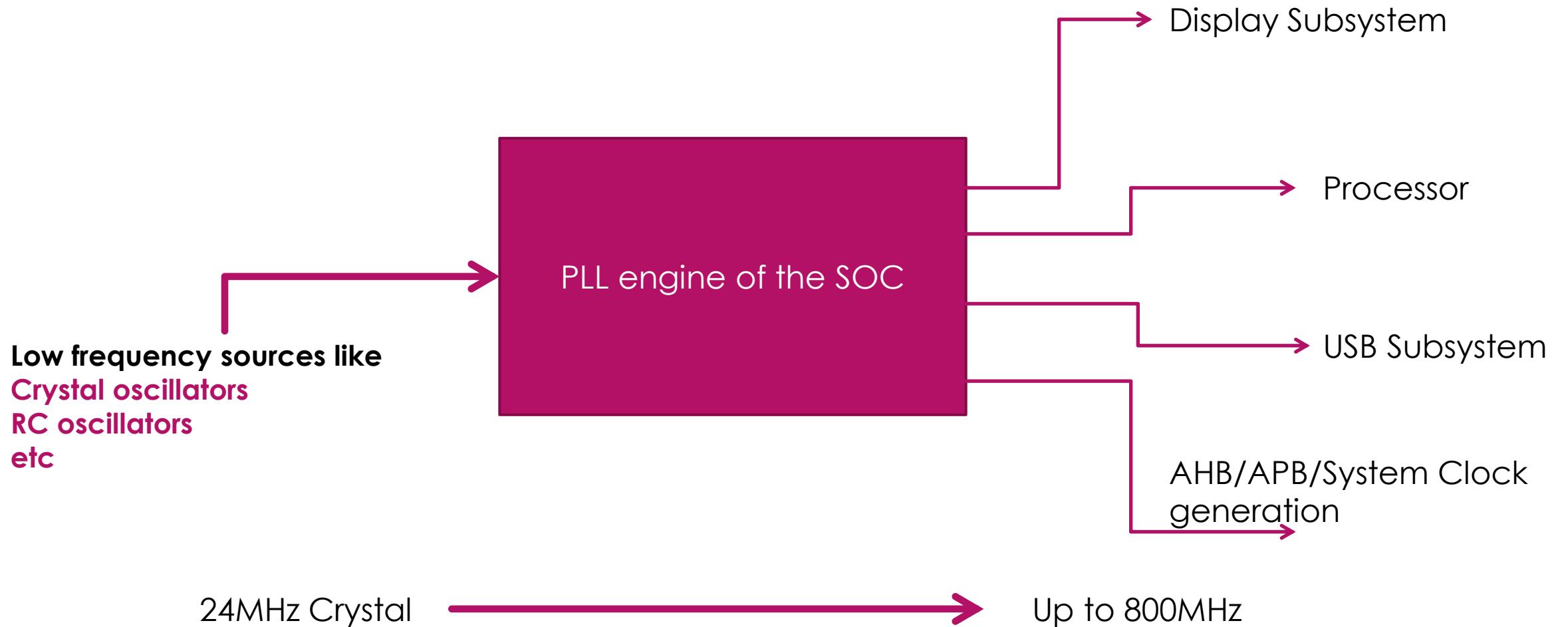


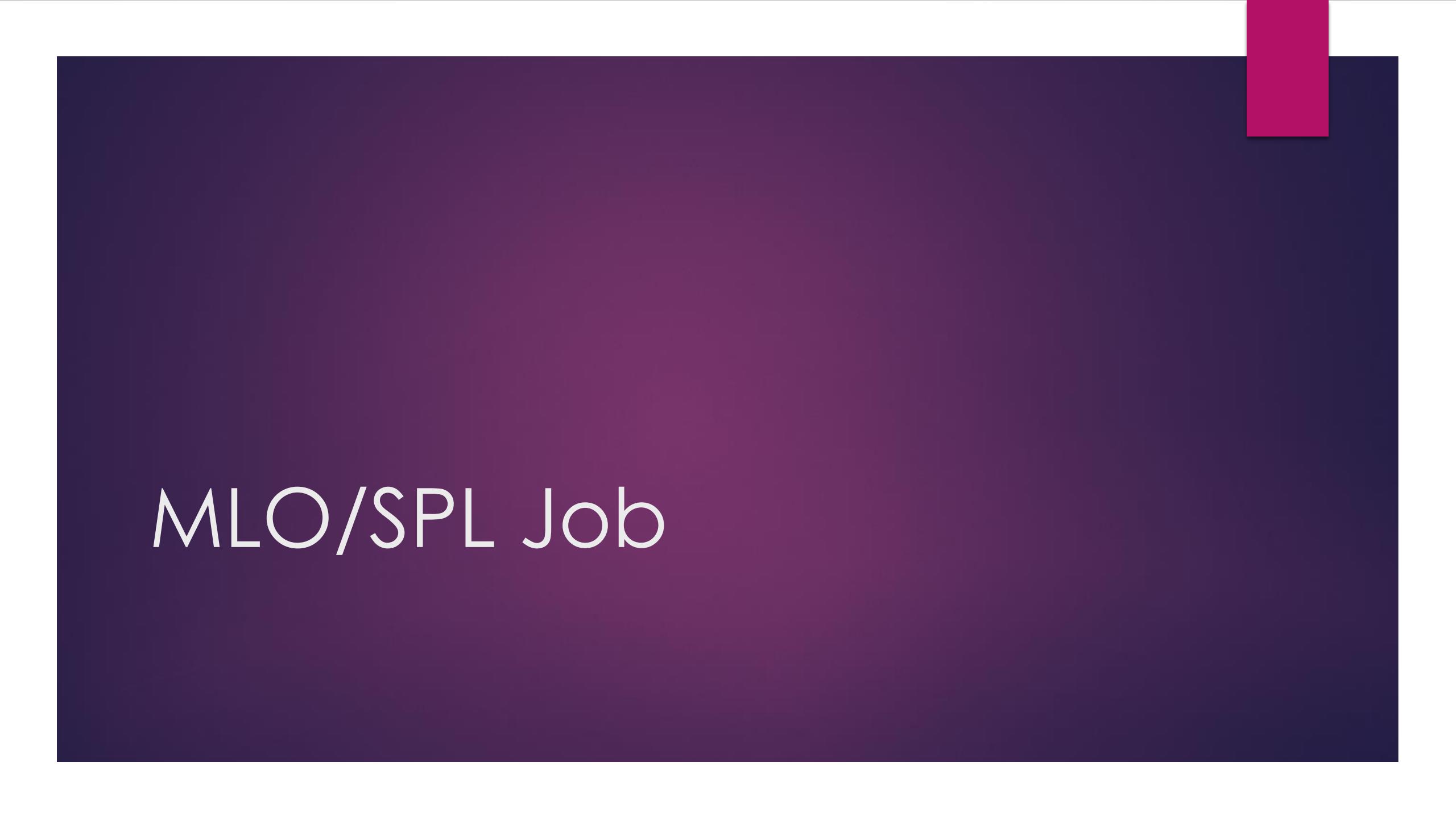
Table 26-5. Crystal Frequencies Supported

SYSBOOT[15:14]	Crystal Frequency
00b	19.2 MHz
01b	24 MHz
10b	25 MHz
11b	26 MHz

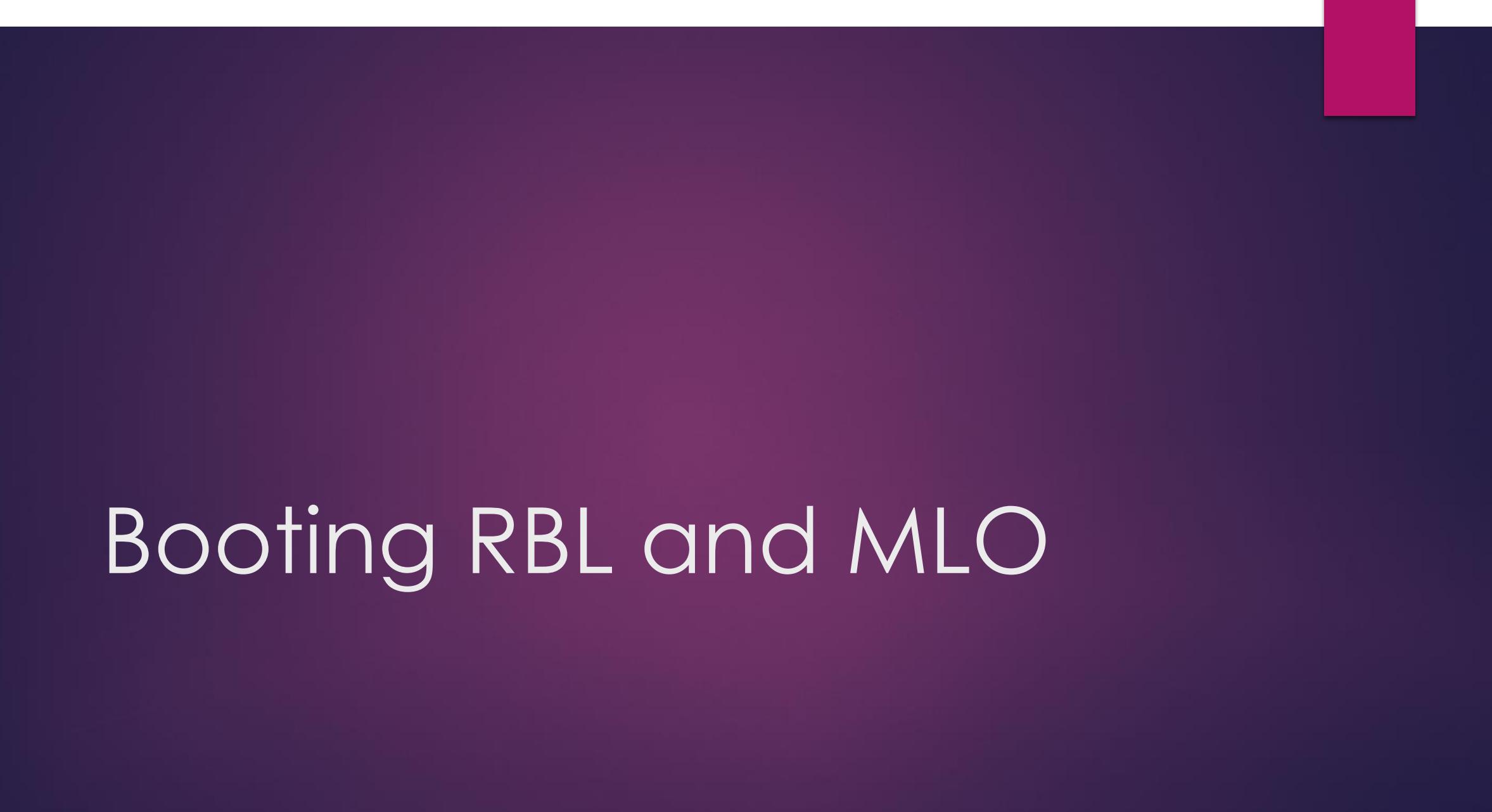


Next→

MLO/SPL Job



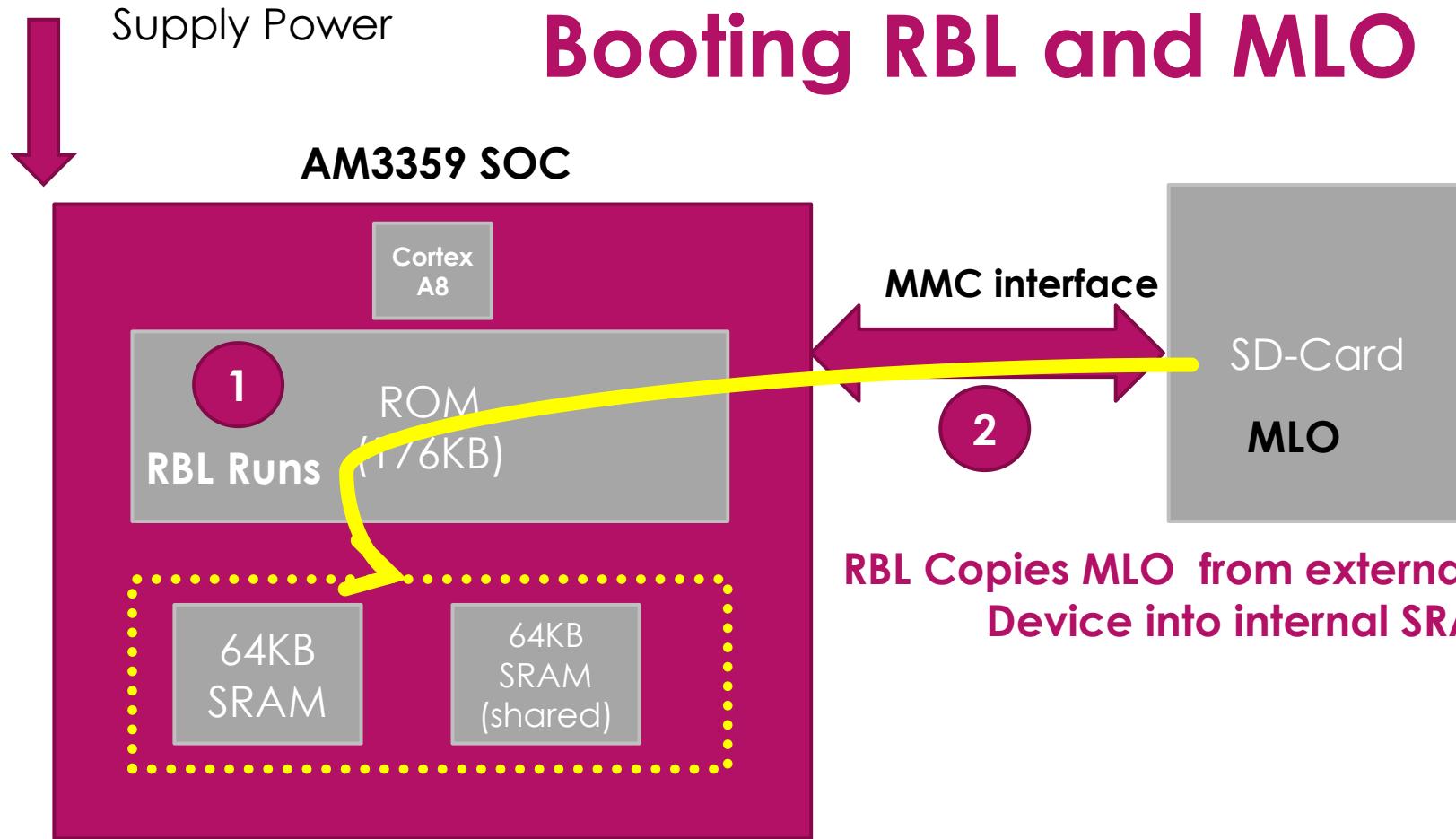
MLO/SPL Job



Booting RBL and MLO

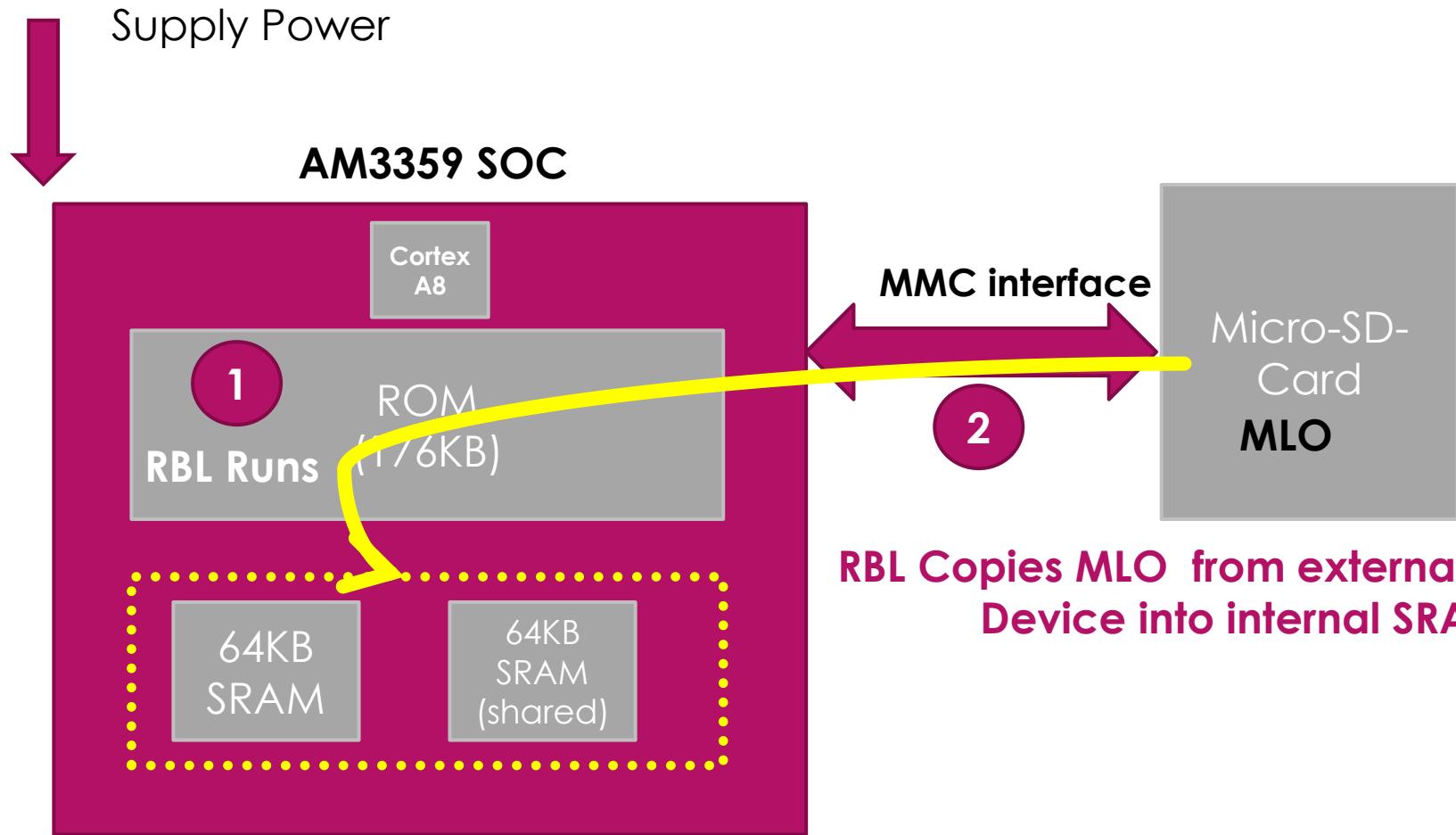
Booting ROM and MLO

Booting RBL and MLO



RBL Copies MLO from external memory Device into internal SRAM

Let's use the prebuilt images from the Angstrom Repository
Later in this course , we will learn how to generate images through compilation.



RBL Copies MLO from external memory Device into internal SRAM

Note:

- 1.Don't operate the Linux command line taking “Super User” control !!**

- 2.Take the help of “sudo” to get the required privileges whenever needed.**

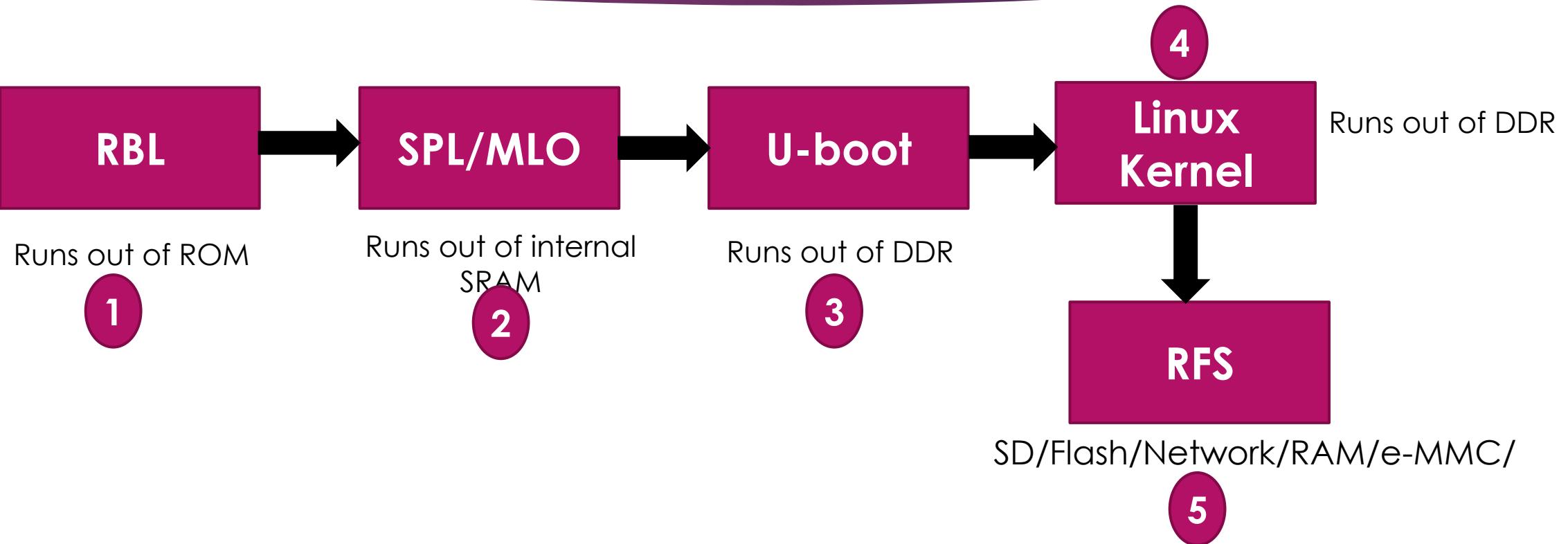
Very Important note :
Select the Correct device name which corresponds to your SD Card. Selecting Wrong device may destroy your Hard disk

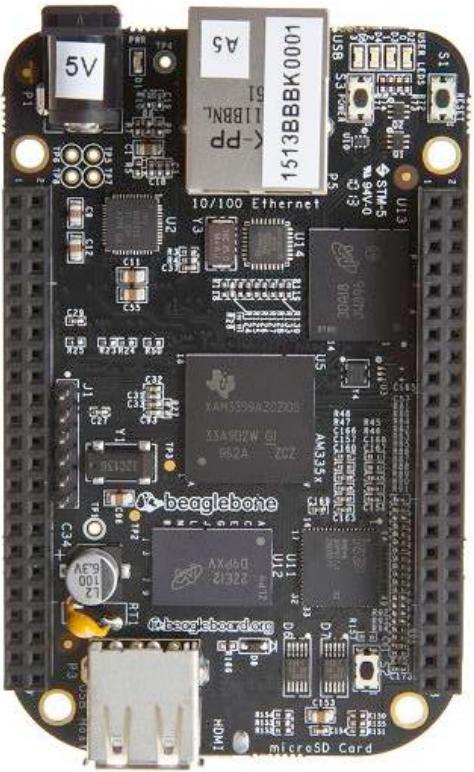
Linux Boot Requirements

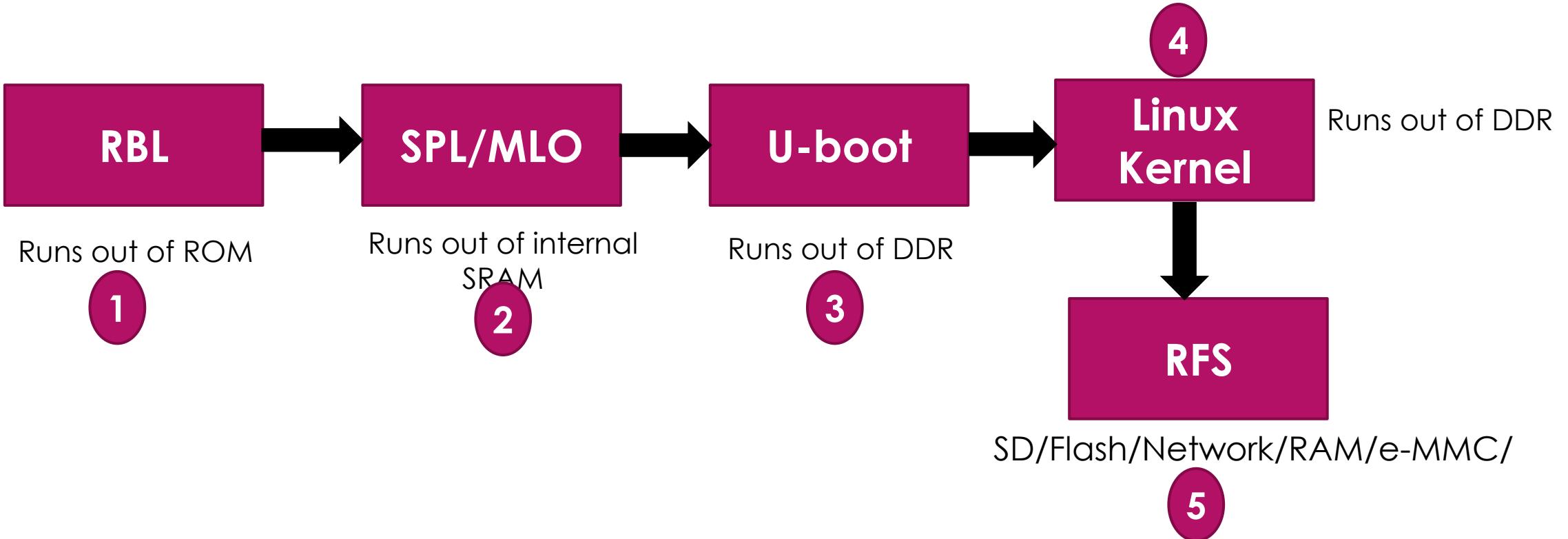
Linux Boot Requirements

**What do we need to successfully boot
Linux on the hardware such as
Beaglebone Black ??**

Linux Boot Requirements









BBB Linux boot sequence
discussion:
RBL and SPL

BBB Linux boot sequence discussion: Boot Strap Loader

Control Flow during boot

Boot Loader

U-boot

head.S

misc.c

Linux's Boot Strap Loader

head.S

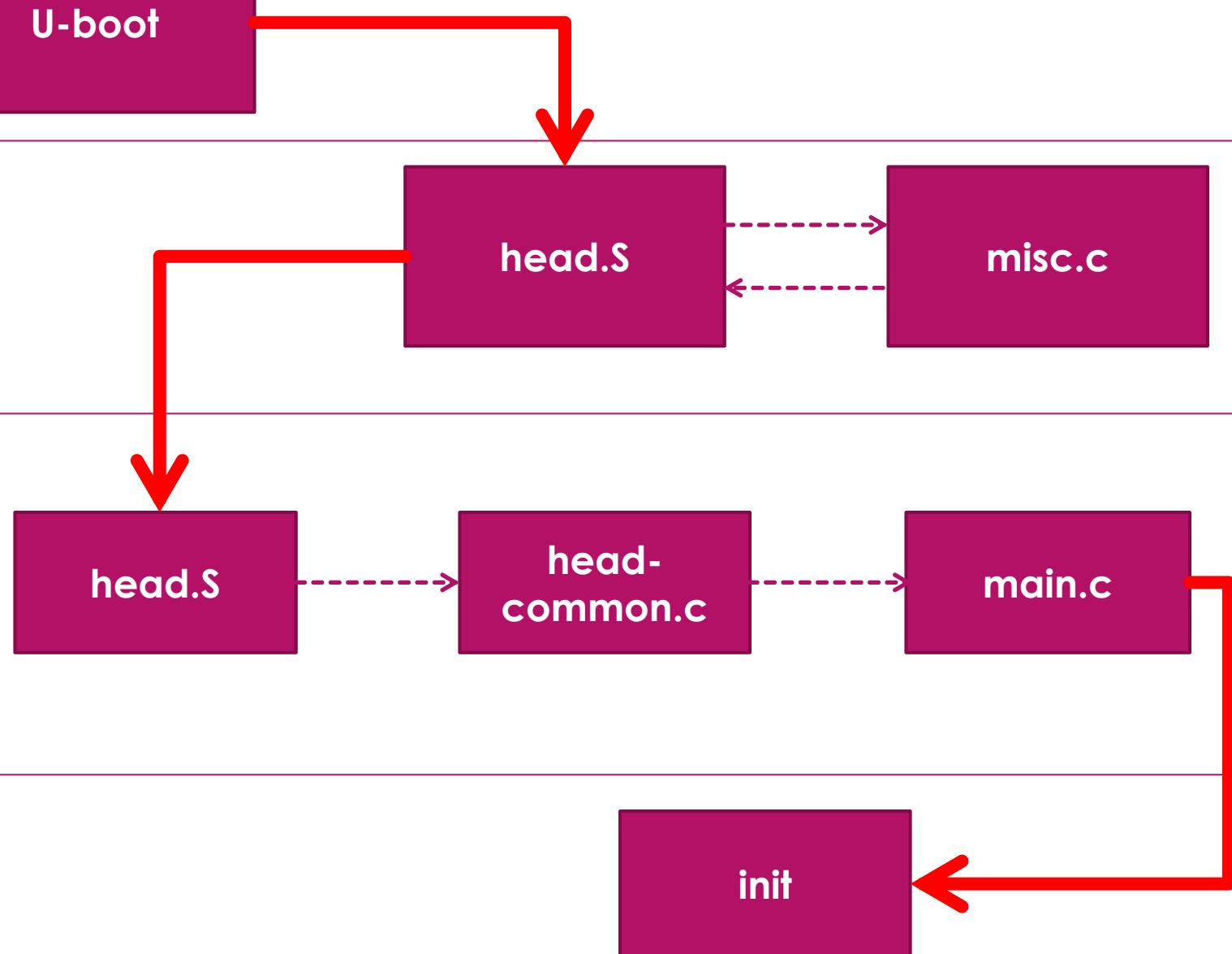
head-common.c

main.c

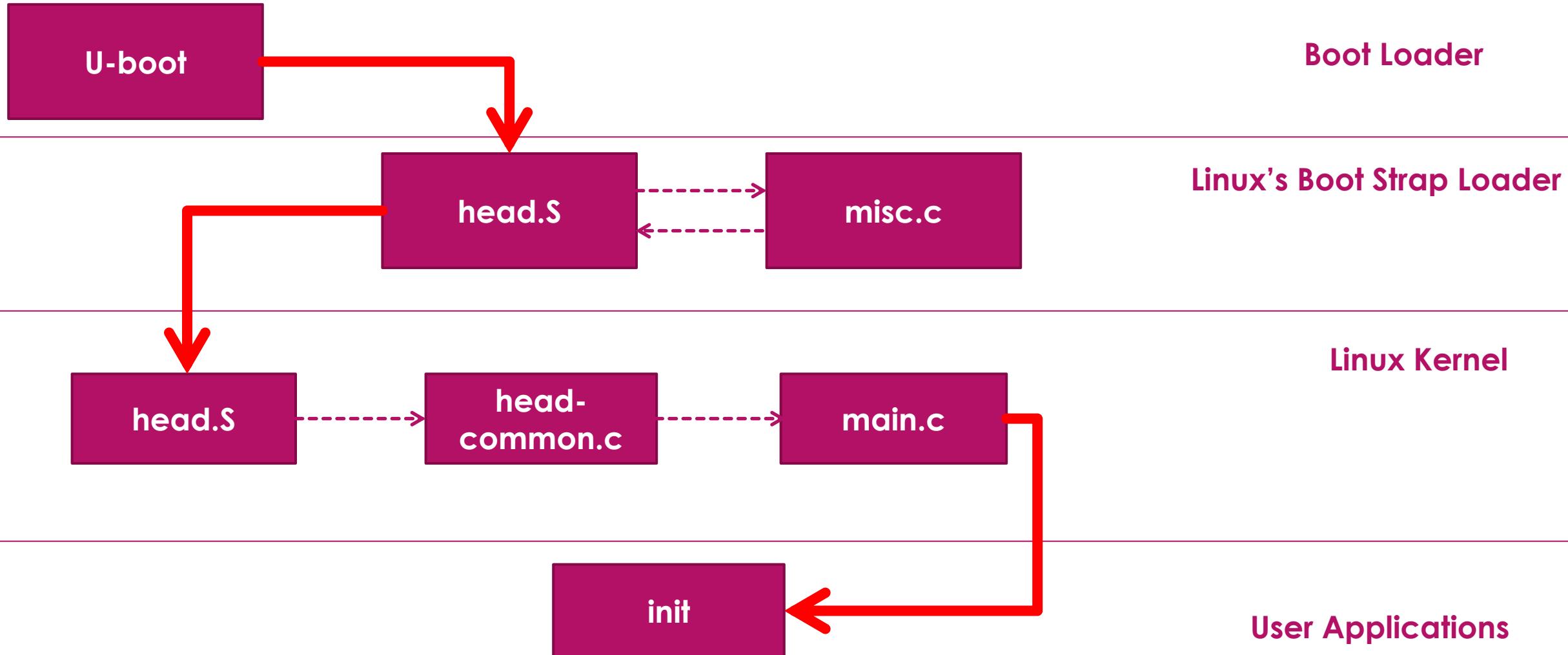
Linux Kernel

init

User Applications

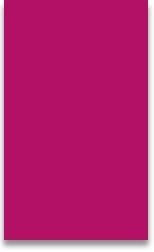


Control Flow during Linux boot



How U-boot hands off Control to the “Boot Strap Loader” of the Linux kernel ??

Let's explore from the source file “bootm.c” of the u-boot source code



U-boot's “bootm.c”

arch/arm/lib/bootm.c



Linux “Boot Strap Loader” location

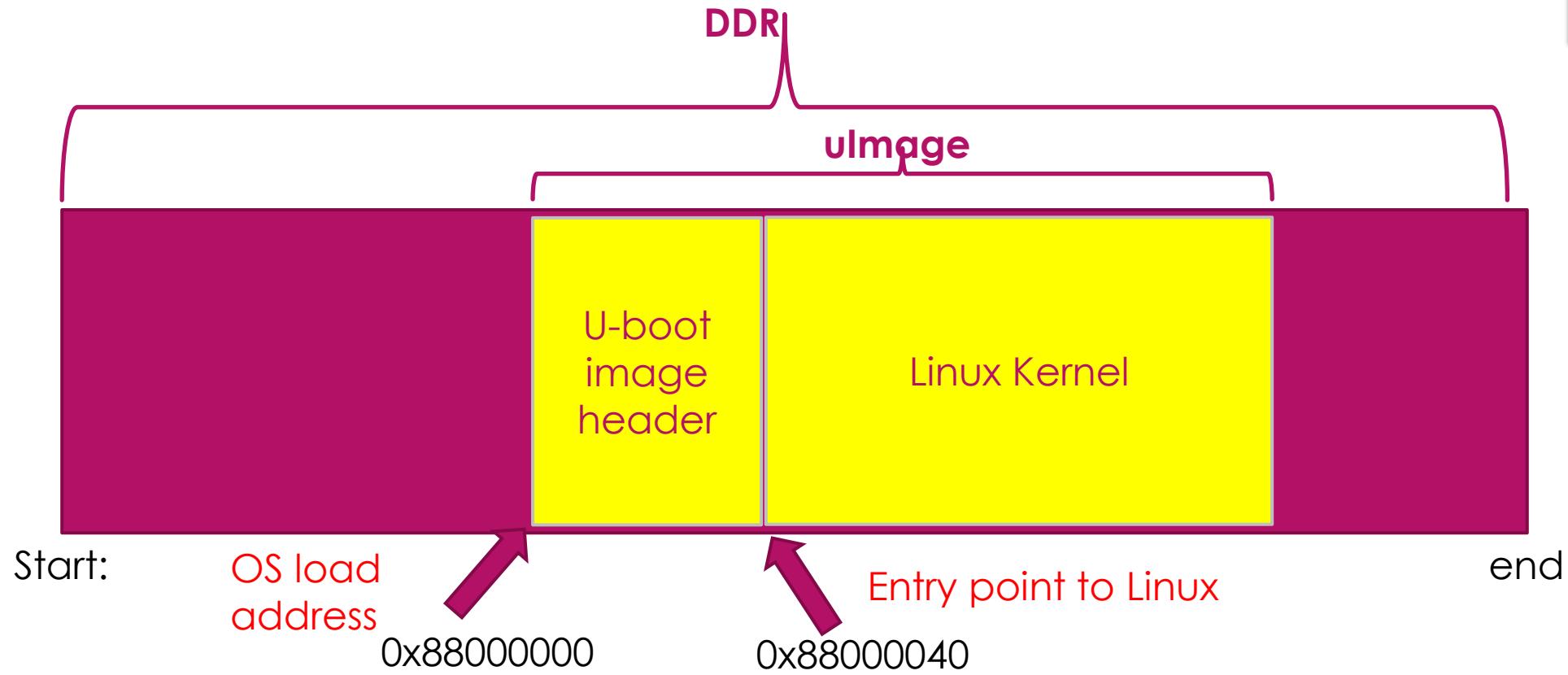
...arch/arm/boot/compressed/

Linux kernel “ARCH” Dependent files

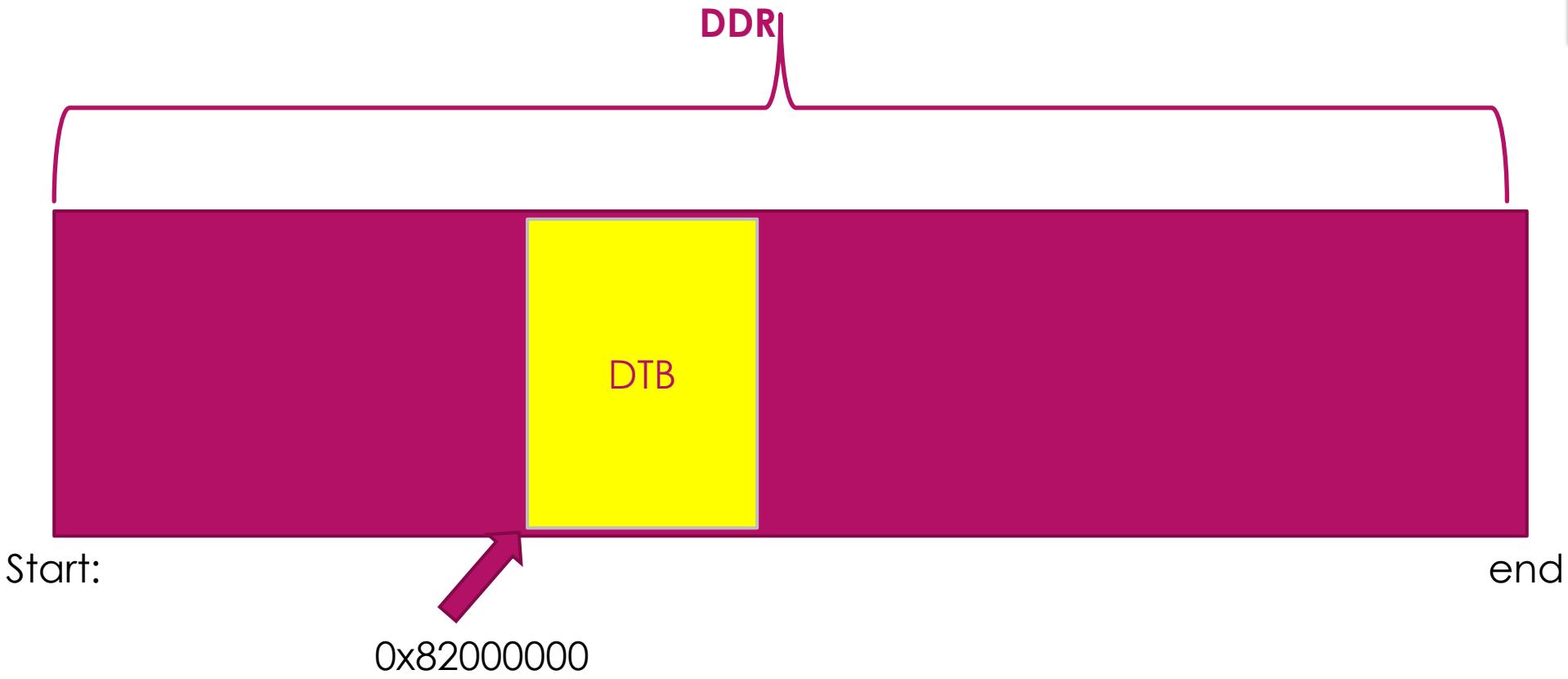
...arch/arm/kernel/

Linux Boot Requirements

**What do we need to successfully boot
Linux on the hardware such as
Beaglebone Black ??**



* Addresses shown are for example only, the actual addresses may vary



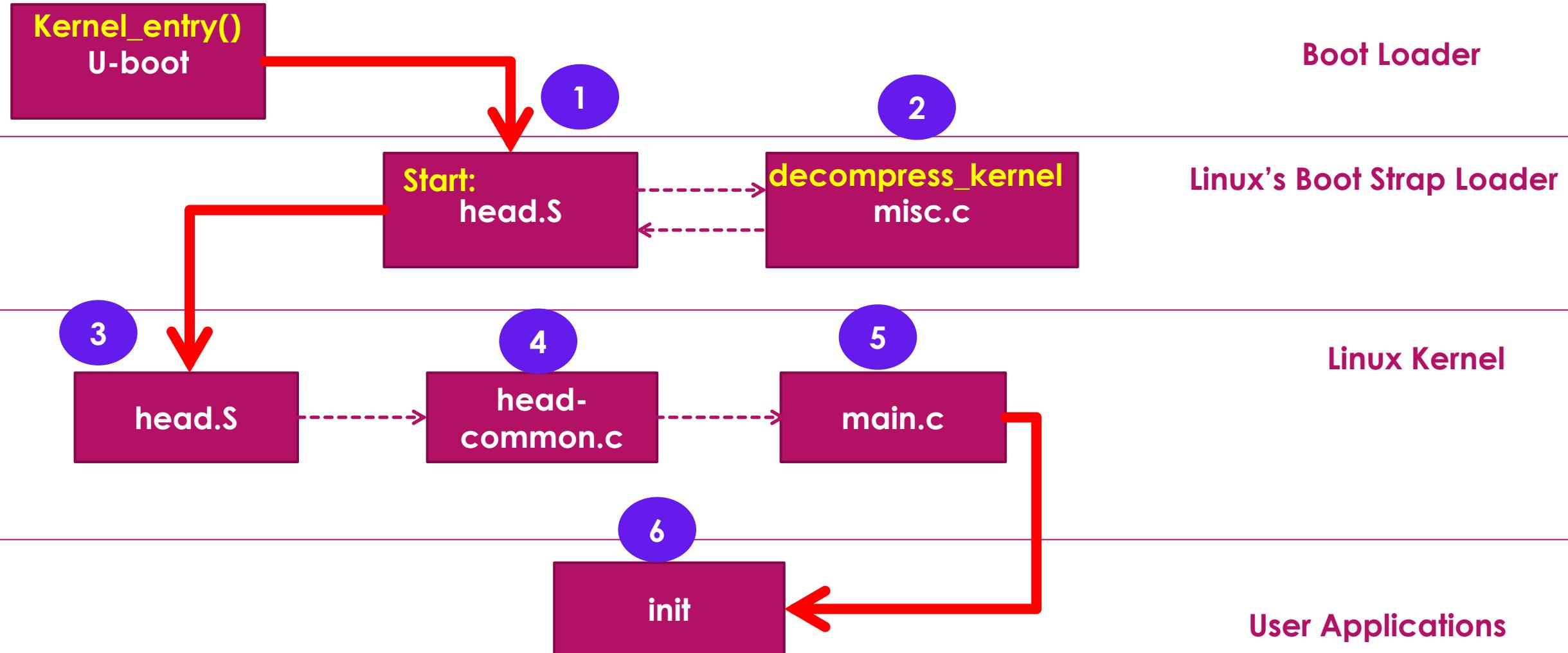
* Addresses shown are for example only, the actual addresses may vary

Machine id of the board which is
detected by the u-boot.
Passed to linux via Register “r1”

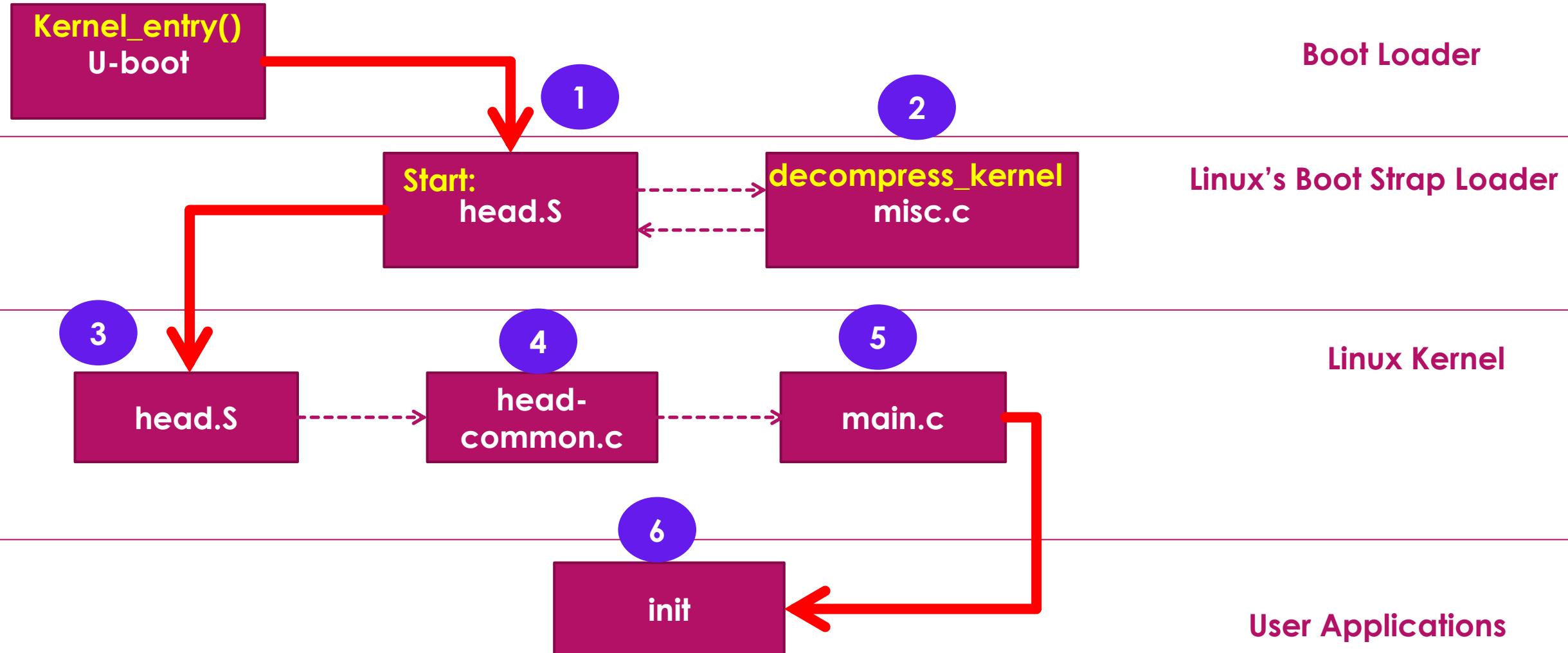
Kernel_entry(0, machid, r2);

Address of the “DTB” present in the
DDR RAM.
Passed to linux via Register “r2”

Control Flow during Linux boot



Control Flow during Linux boot



arch/arm/kernel/head.S and friends do:

1. CPU specific initializations
2. Checks for valid processor architecture.
3. Page table inits
4. Initialize and prepare MMU for the identified Processor Architecture
5. Enable MMU to support virtual memory
6. Calls “`start_kernel`” function of the main.c (“Arch” independent code)



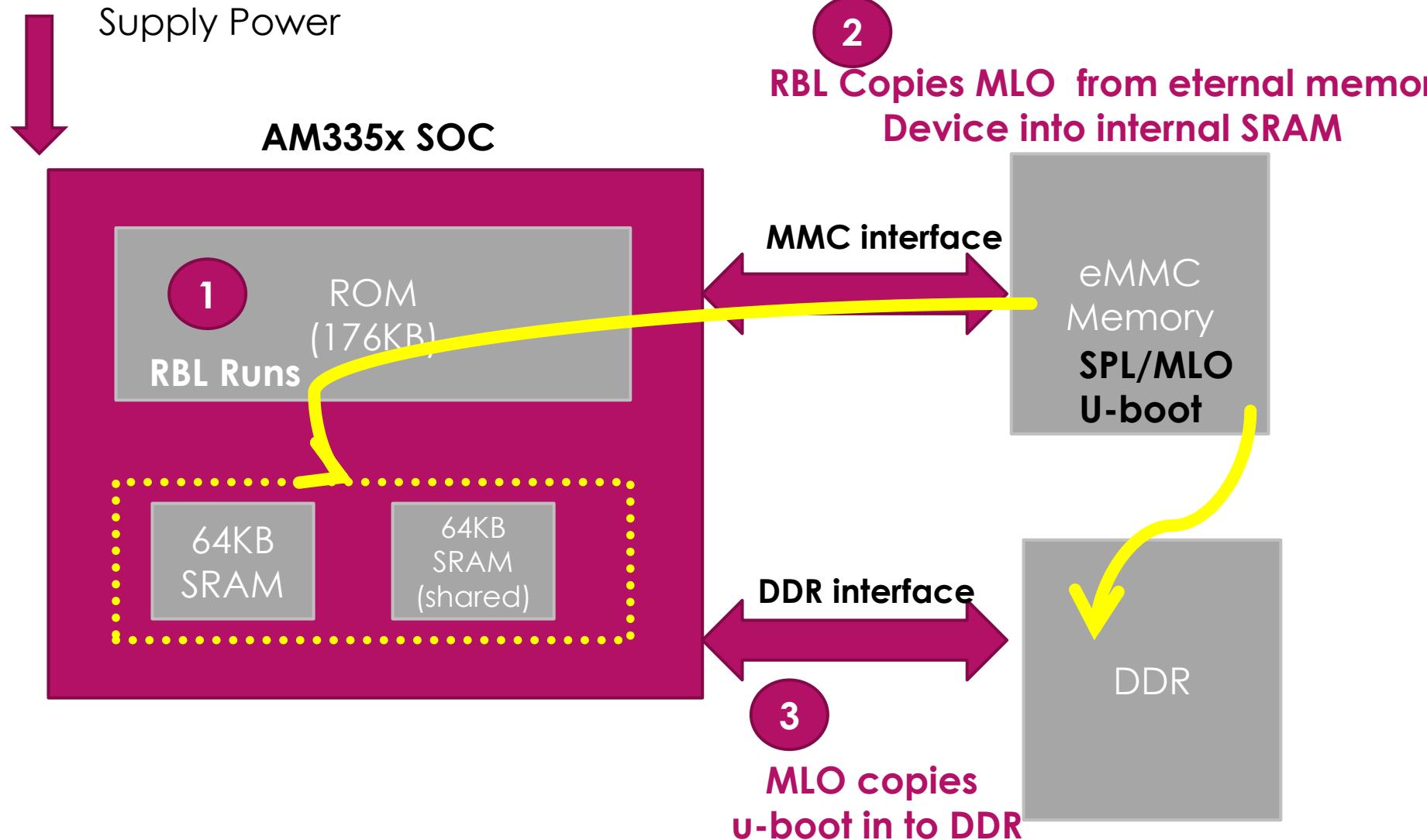
BBB Linux boot sequence
discussion:

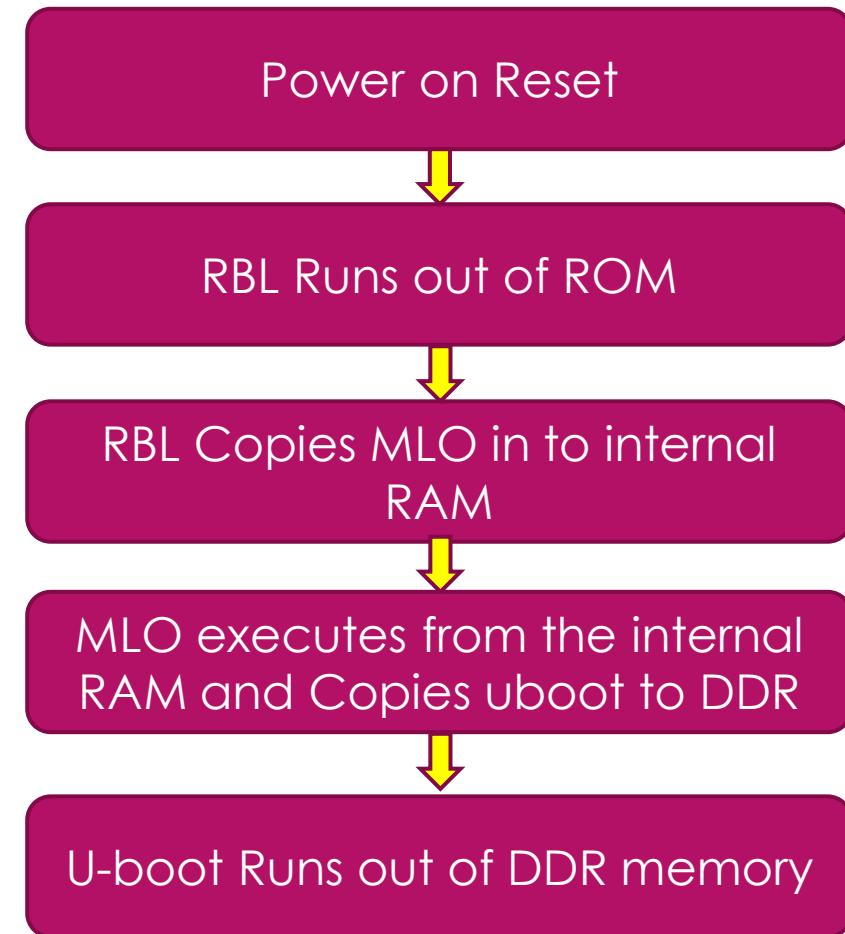
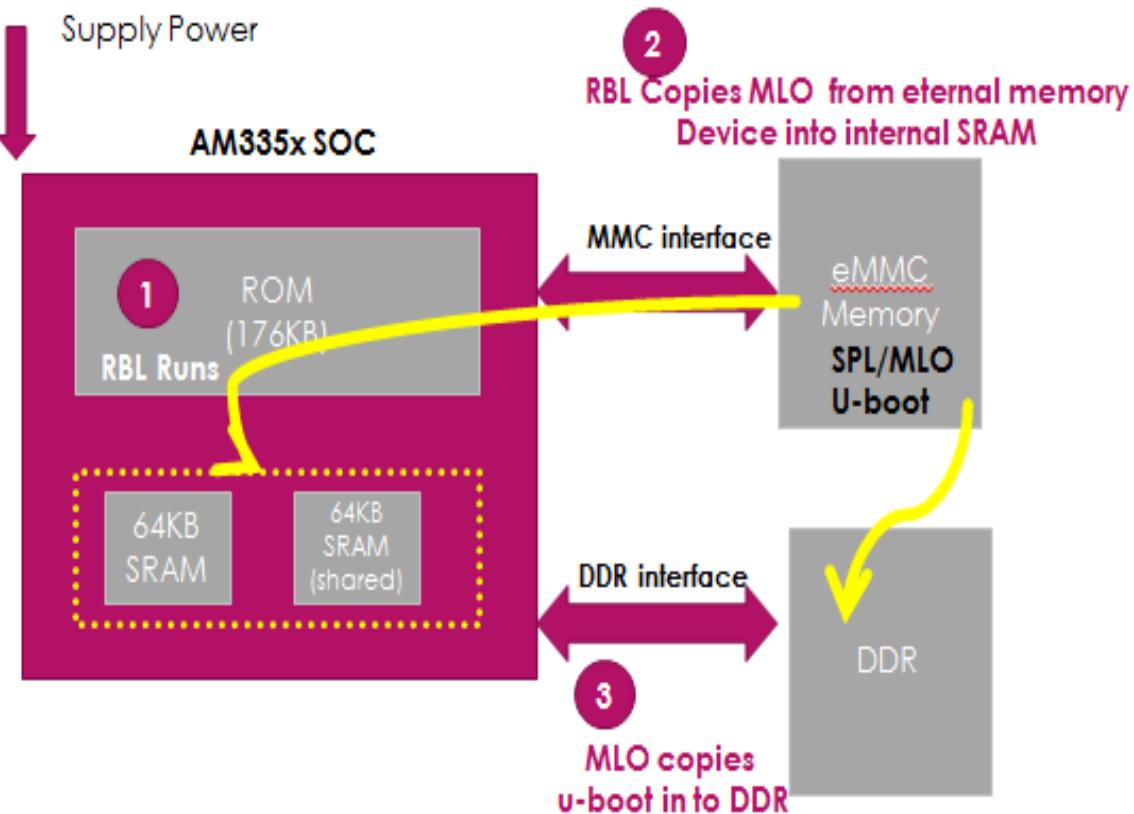
Linux Boot Strap Loader

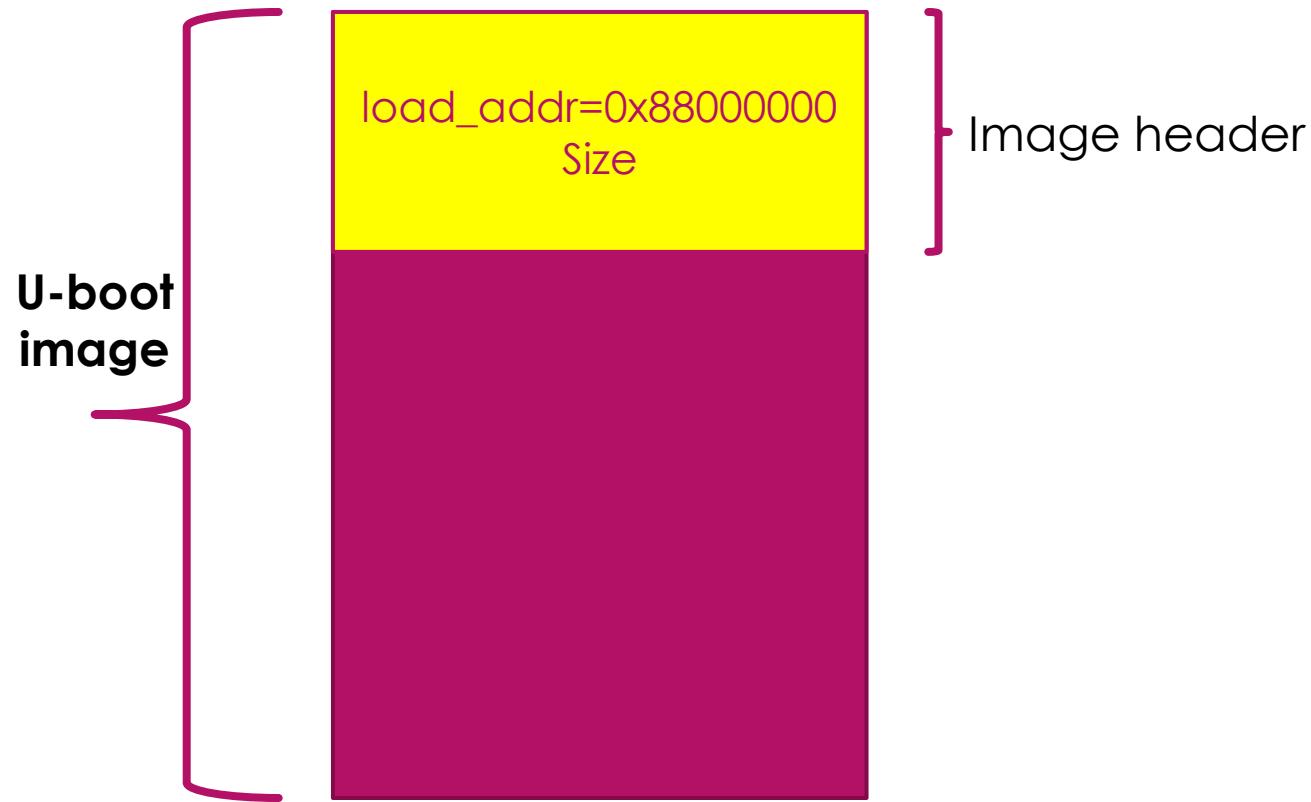
BBB Linux boot sequence discussion: ROM and SPL

Next →

Linux Boot Strap Loader







The job of “U-boot”

1. Initialize some of the peripherals like,I2C, NAND,FLASH, ETHERENT , UART,USB,MMC ,etc, because it supports loading kernel from all these peripherals
2. Load the Linux kernel image form various boot sources to the DDR memory of the board.
3. Boot sources : USB, eMMC, SD card, Ethernet , serial port, NAND flash,etc.
4. Passing of boot arguments to the kernel

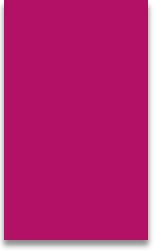
uEnv.txt

You can change the boot behavior of the u-boot by using a file called **uEnv.txt**.

Use this file to set the env variables which drives the uboot according to your need.

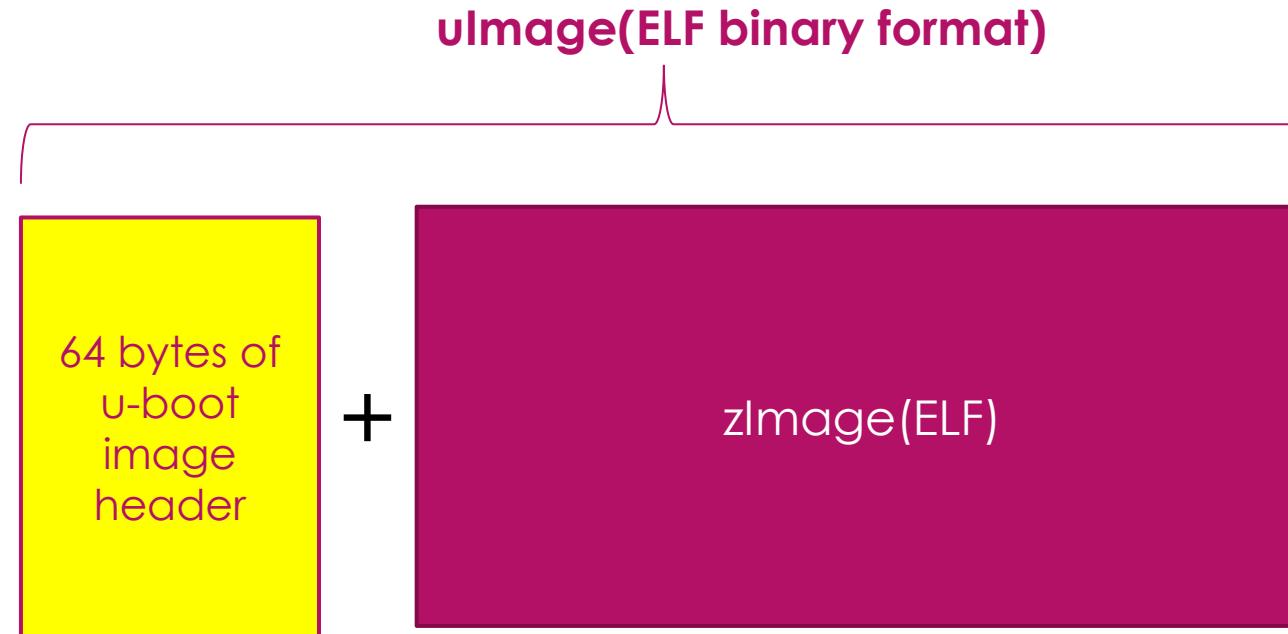
```
console=tty00,115200n8
ipaddr=192.168.7.2
serverip=192.168.7.1
absolutepath=/var/lib/tftp/
rootpath=/srv/nfs/bbb,nolock,wsize=1024,rsize=1024 rootwait rootdelay=5
loadtftp=echo Booting from network ...;tftpboot ${loadaddr} ${absolutepath}uImage; tftpboot
netargs=setenv bootargs console=${console} root=/dev/nfs rw nfsroot=${serverip}:${rootpa
uenvcmd=setenv autoload no; run loadtftp; run netargs; bootm ${loadaddr} - ${fdtaddr}
```

More on *uEnv.txt* later in this course



**Let's again visit the Ångström repository ,
download the prebuilt Linux kernel image
(ulimage) and will test on the hardware !**

Reading U-boot header information of the uImage manually by using U-boot commands

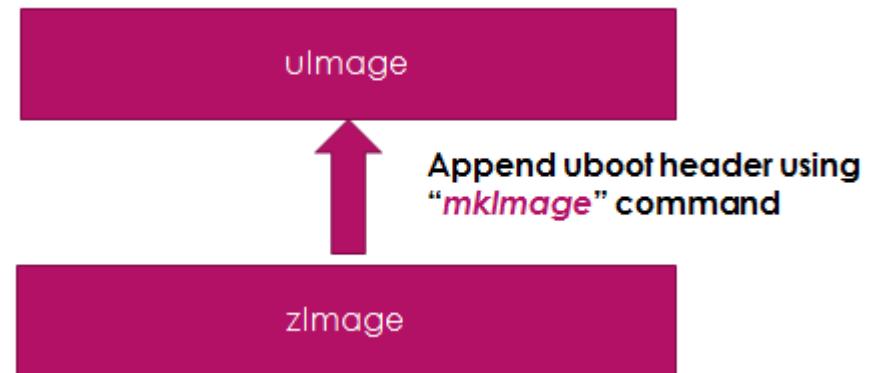
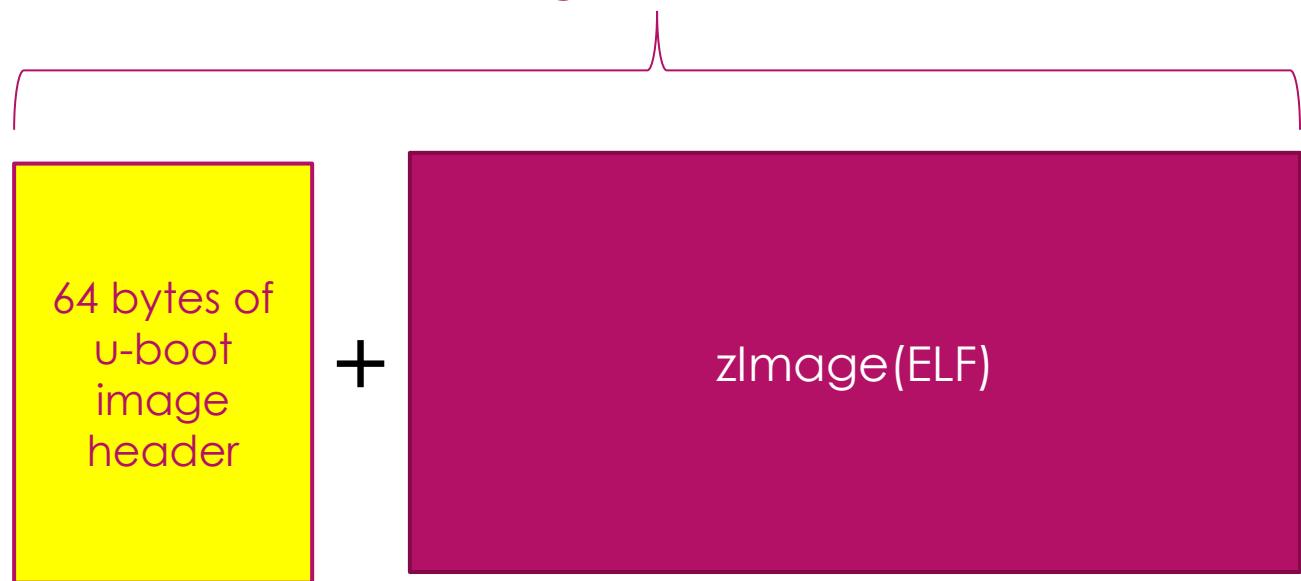


Reading U-boot header information of the ulmage manually

- 1. Load the ulmage from Memory device(SD card/eMMC) in to the DDR memory of the board**
- 2. Use the memory dump command of U-boot to dump header information .**



uImage(ELF binary format)



Uboot Linux Image header

```
/*
 * Legacy format image header,
 * all data in network byte order (aka natural aka bigendian).
 */

typedef struct image_header {
    uint32_t      ih_magic;          /* Image Header Magic Number */
    uint32_t      ih_hcrc;           /* Image Header CRC Checksum */
    uint32_t      ih_time;           /* Image Creation Timestamp */
    uint32_t      ih_size;           /* Image Data Size */
    uint32_t      ih_load;           /* Data Load Address */
    uint32_t      ih_ep;              /* Entry Point Address */
    uint32_t      ih_dcrc;           /* Image Data CRC Checksum */
    uint8_t       ih_os;              /* Operating System */
    uint8_t       ih_arch;            /* CPU architecture */
    uint8_t       ih_type;             /* Image Type */
    uint8_t       ih_comp;            /* Compression Type */
    uint8_t      ih_name[IH_NMLEN];   /* Image Name */
} image_header_t;
```

```
graph TD; zImage[zImage] -- "Append uboot header using  
\"mkImage\" command" --> uImage[uImage]
```

uImage

Append uboot header using
“***mkImage***” command

zImage

/

```
graph LR; / --- bin["bin/"]; / --- boot["boot/"]; / --- dev["dev/"]; / --- etc["etc/"]; / --- lib["lib/"]; / --- media["media/"]; / --- mnt["mnt/"]; / --- opt["opt/"]; / --- run["run/"]; / --- sbin["sbin/"]; / --- srv["srv/"]; / --- tmp["tmp/"]; / --- usr["usr/"]; / --- var["var/"]
```

Busybox: Introduction

Busybox: Compilation

Busybox: Kernel Modules installation



Testing boot images & busybox on BBB: Part-2



**First Let's build all the linux commands
source code as a static binary instead of
dynamic.**



**The default configuration generates around 89 commands in bin/ folder.
You can use “menuconfig” to increase or decrease these commands.**

The Swiss Army Knife of Embedded Linux

Busybox is nothing but a software tool, that enables you to create a customized root file system for your embedded linux products

Why Busybox ?

1. It enables you to create customized file system that meets your resource requirements.
2. If your product is resource limited in terms of memory, then you can customize the files system such a way that , it can fit , in to your product with limited memory space.
3. you can use this tool to remove all unwanted features which your product doesn't need , like you can remove unwanted Linux commands and features, directories, etc. using the customization tool.
4. Busybox has the potential to significantly reduce the memory consumed by various Linux commands by merging all the Linux commands in one single binary.

Why Busybox ?

Storage space consumed by Ubuntu PC

/bin/ directory = ~13MB

/sbin/ directory = ~14MB

/usr/bin + /usr/sbin = ~200MB

Busybox supports most of the commands that you can find in /bin and /sbin of your PC but with very minimal memory footprint.

***Busybox has only one single executable binary
Called “busybox”***

busybox

This single executable contains support for all the commands

ls
cp
cat
mkdir
echo
rm
mv
ifconfig
iptables
wc
find
grep
ssh
insmod

***Busybox has only one single executable binary
Called “busybox”***

busybox

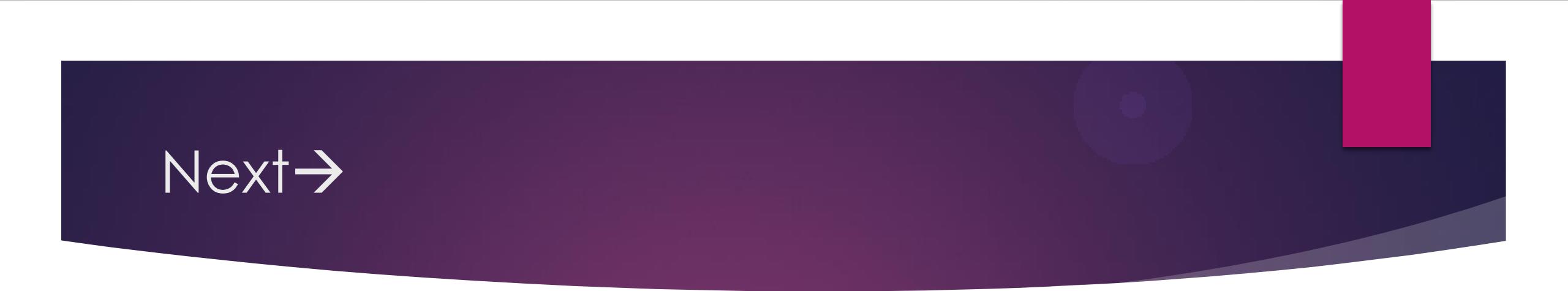
This single executable contains support for all the commands

```
switch ( command_name ) {  
  
    case "ls" :  
        execute_ls_functionality();  
    break;  
  
    case "cat" :  
        execute_cat_functionality();  
    break;  
  
    case "mv" :  
        execute_mv_functionality();  
    break;  
    -----  
    -----  
    -----  
}
```

INSTRUCTION

Summary

1. **Busybox is a tool to generate your own customized , memory friendly file system.**
2. **It does not generate individual Linux commands binaries, there is only one executable binary that's called "busybox" which implements all the required Linux commands which you can select using the configuration tool**



Next→

Busybox Compilation

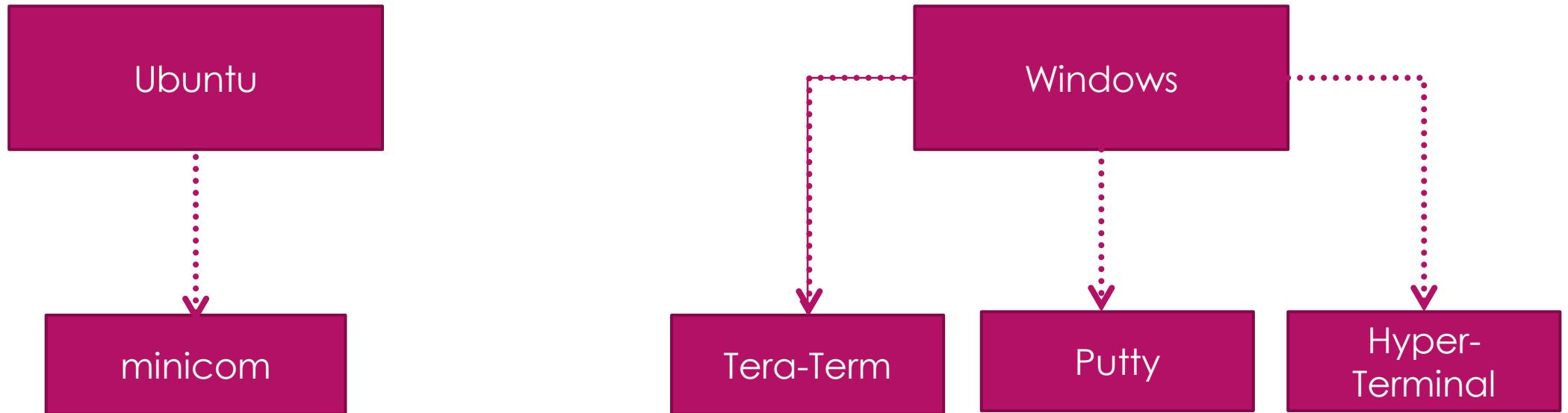
Decoding U-boot Header of ulmage Manually



**Skip this lecture if you already know how to
handle Tera term and putty**

Exploring Tera-Term and Putty

Serial Port Monitoring Software



- 1) Connect the Micro SD card to the BBB SD Card Connector
- 2) Power up the BBB (you can use mini USB(P4))
- 3) Press and hold S2 button then press and release S3 Button.



**Remember that board is just booting
debain linux from SD card. It is not going
to flash the eMMC until we enable the
eMMC flasher Script.**

Flashing of eMMC is over !!!!

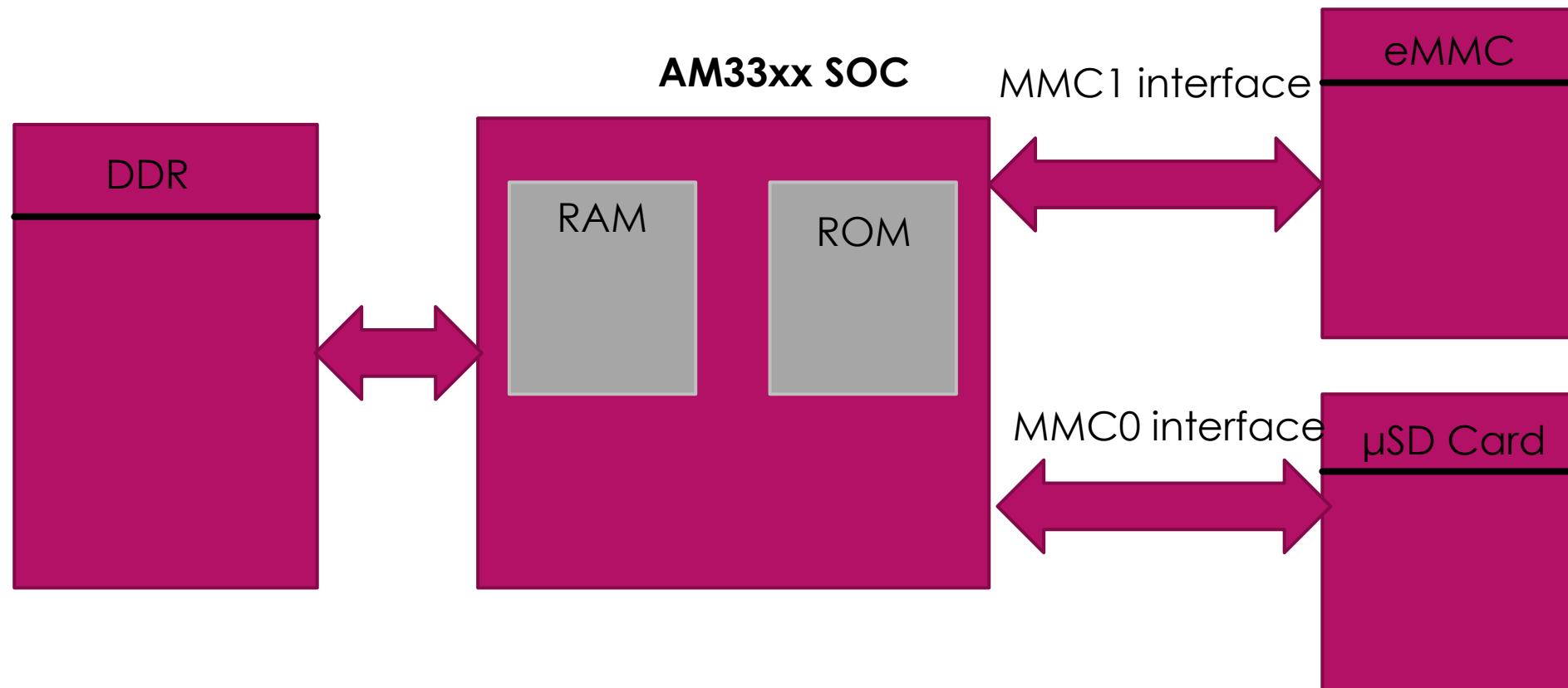
Remove power from the board

Remove SD card from the board

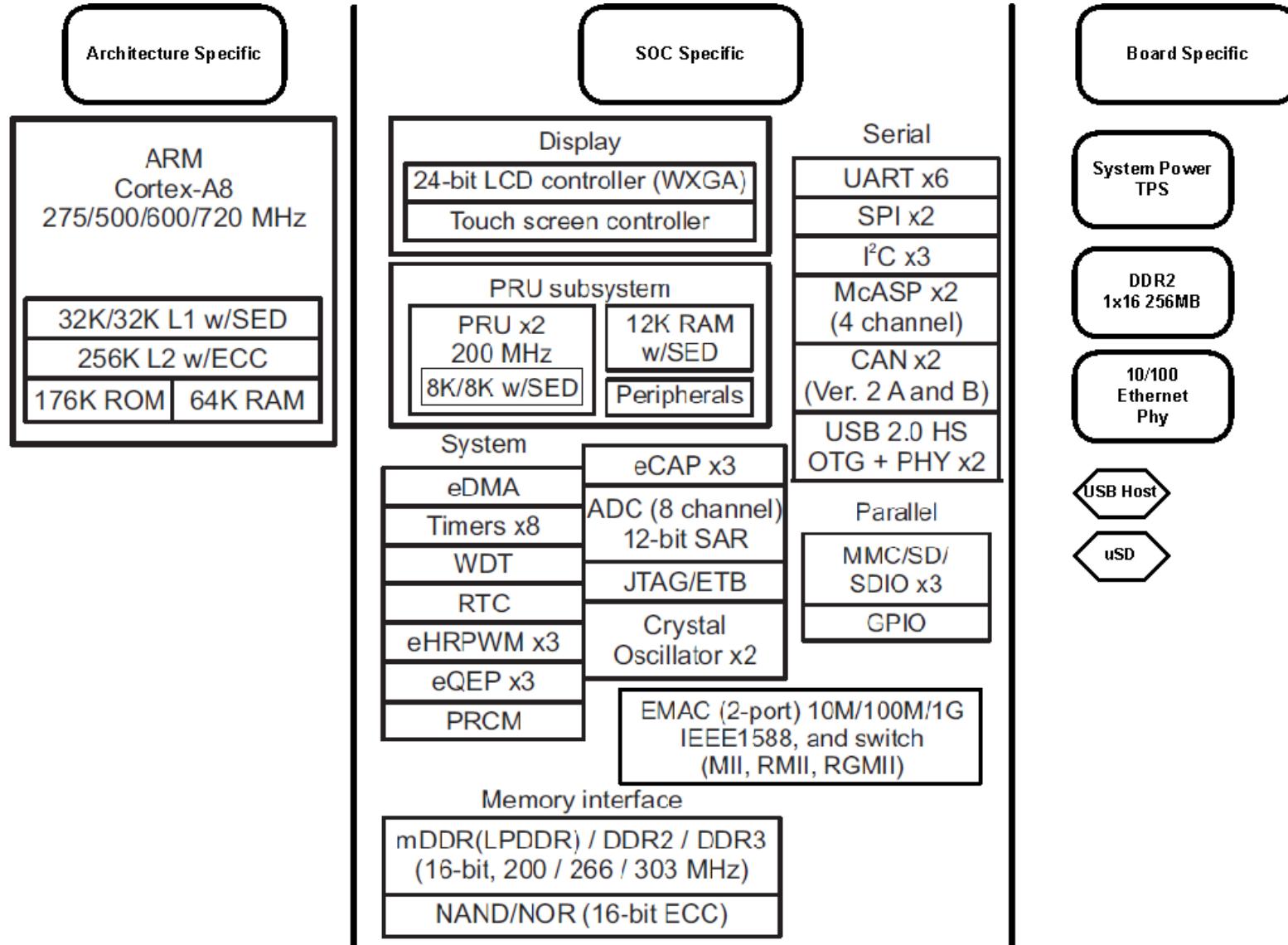
Re-Apply power to the board.

**Now the board should boot newly flashed
debian os from eMMC.**

eMMC Booting: Big Picture



Architecture vs. SOC vs. Board Porting



U-Boot Source Directory

- ▶ Using the existing am335x source directory
- ▶ The developer will be concentrating on one source directory and for the most part one include directory



board/ti/am335x

```
└── common_def.h  
└── evm.c  
└── Makefile  
└── mux.c  
└── pll.c  
└── pmic.h  
└── tps65217.h
```

evm.c –
- board_init
- ddr init
- clock init
- serial init
- tps65217

mux.c –
- pin mux config support functions
- initialized pin mux config structures

pll.c – support functions for multiple pll s

arch/arm/include/asm/arch-ti81xx

```
└── clock.h  
└── clocks_am335x.h  
└── clocks_ti814x.h  
└── clocks_ti816x.h  
└── cpu.h  
└── cdr_defs.h  
└── emac_defs.h  
└── hardware.h  
└── i2c.h  
└── rem.h  
└── mmc.h  
└── mmc_host_def.h  
└── rand.h  
└── cmap.h  
└── sys_proto.h
```

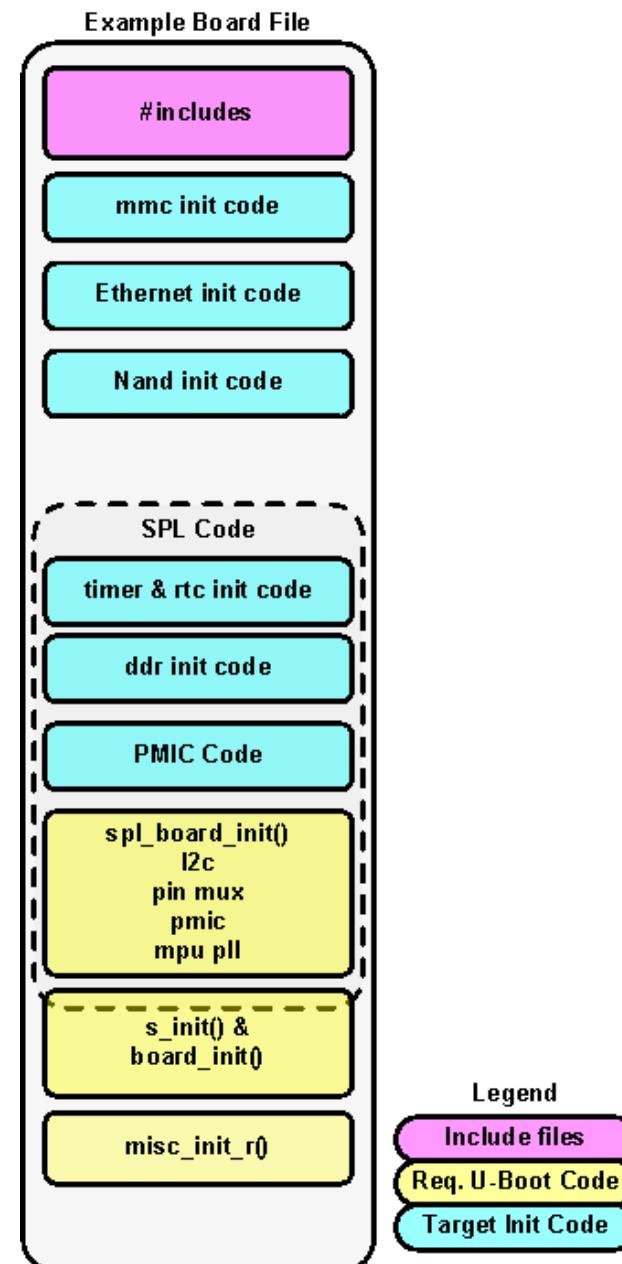
Architecture support include files

include/configs/am335x_evm.h

U-Boot configuration for the target processor

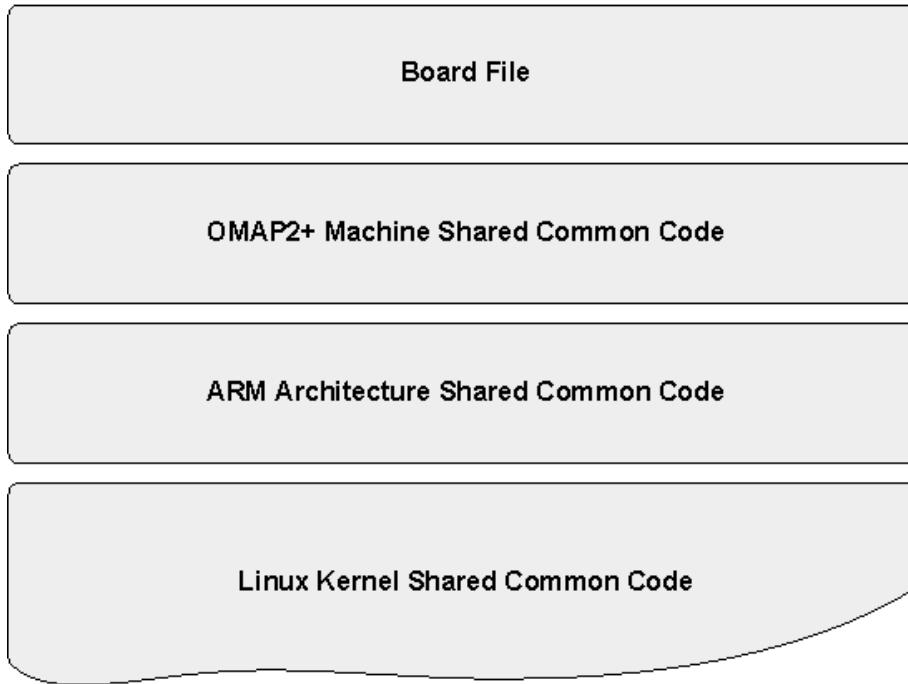
U-Boot Anatomy of a board File

- ▶ Defines Required interface functions for SPL and U-Boot
- ▶ One source file contains the code for both SPL and U-Boot and are separated by pre-processor flags
- ▶ SPL handles the initialization of clocks, DDR, Serial Port and PMIC
- ▶ Some functions are defined twice in both an SPL context and then again in a U-Boot context (`s_init` & `board_init`)
- ▶ The board file is where the developer will spend most of their effort for a port



How the Board File fits in the stack

- ▶ Board Developer will spend most of their time in the Board file.
- ▶ The Board file makes use of the machine shared common code
- ▶ The underlying port to the ARM Architecture Shared common code is already done and does not need to be looked at
- ▶ Finally everything rests on the Linux Kernel Shared Common Code.
 - The lower in the stack you go the less direct interaction the board developer will or need to have.



OMAP2+ Machine Shared Common Code

- There are several board files in the mach-omap2 directory. These board files typically use the support functions defined within this directory. Below is a sampling of some of the supporting common code, not all are mentioned here.

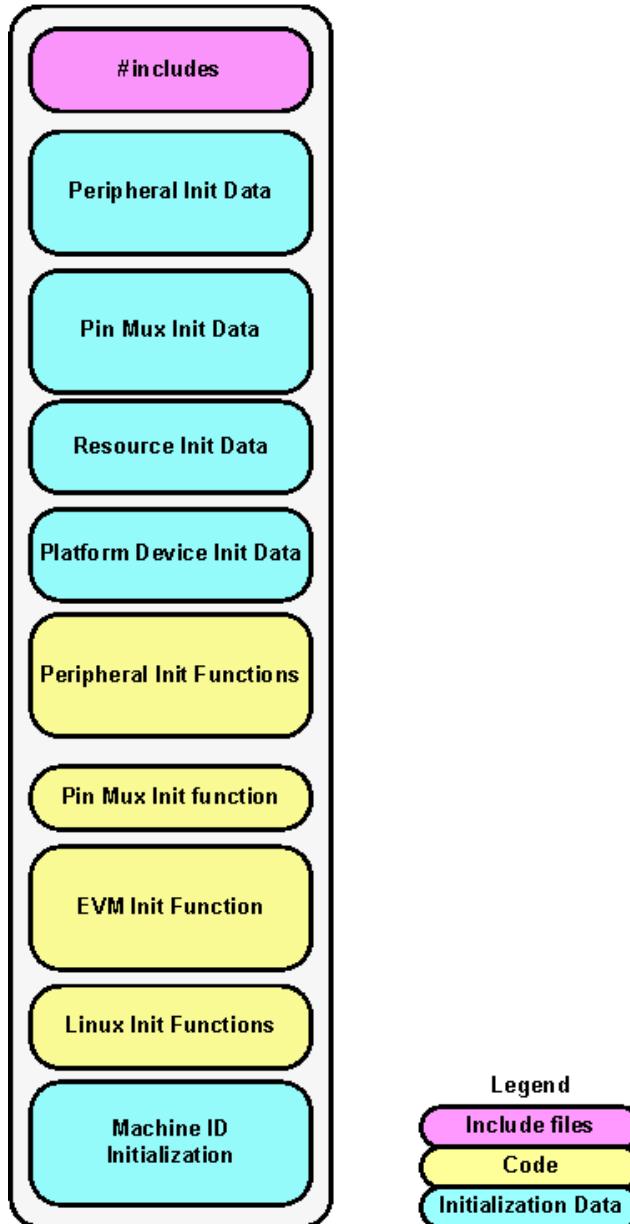
OMAP2 Machine Shared Common Code – arch/arm/mach-omap2

Not a complete listing of the interfaces, just a few are highlighted and to explain how they are used, review this directory to see additional interfaces

serial	Sets up UARTs including pin mux	gpio	Initialization function
devices	Init calls, platform registration for most peripherals	i2c	Reset and Mux functions
common	Init calls to define global address range for select interfaces	mux	Defines a Pin Mux abstraction with supporting functions
clocks	Define clock domain mgmt functions	hs mmc	Init functions, hw and platform data
control	OMAP2 control registers	sdrc	Init function for SDRC and SMS
display	Display init calls, handles the differences between OMAP2,3 and 4	voltage	Voltage domain support functions

Template Board File Anatomy

- ▶ Binds Linux to a particular target
- ▶ Interfaces with the OMAP2+ Machine Shared Common Code.
- ▶ Defines pin mux configuration
- ▶ The file contains device initialization functions and data.
- ▶ Defines the Machine ID and identifies to the Linux Kernel initialization functions



Initialization Function Call Sequence for MMC Enabling

- ▶ This sequence of code is adding in the MMC initialization code to the template file.

```
MACHINE_START(AM335XEV, "am335xevm")
/* Maintainer: Texas Instruments */
.atag_offset = 0x100,
.map_io     = am335x_evm_map_io,
.init_early  = am33xx_init_early,
.init_irq    = ti81xx_init_irq,
.handle_irq  = omap3_intc_handle_irq,
.timer       = &omap3_am33xx_timer,
.init_machine = am335x_evm_init,
MACHINE_END
```

```
static void __init am335x_evm_init(void)
{
    am33xx_cpuidle_init();
    am33xx_mux_init(NULL);
    omap_serial_init();
    am335x_rtc_init();
    clkout2_enable();
    omap_sdrc_init(NULL, NULL);

    /* Beagle Bone has Micro-SD slot which doesn't have Write Protect pin */
    am335x_mmc[0].gpio_wp = -EINVAL;
    mmc0_init();
}
```

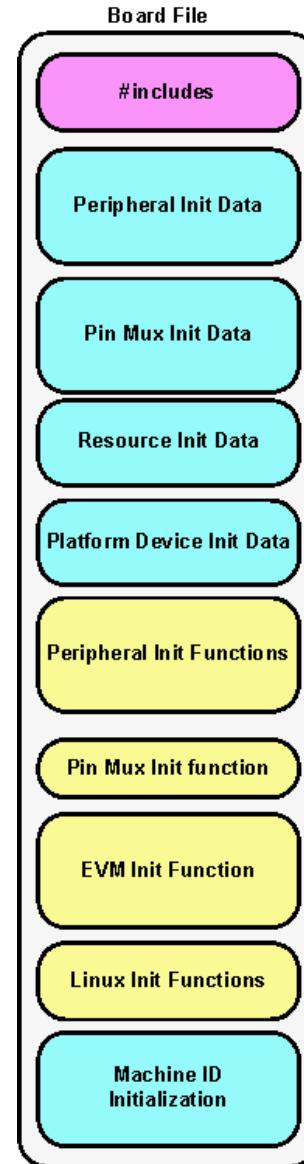
```
static void mmc0_init(void) {
    setup_pin_mux(mmc0_pin_mux);

    omap2_hsmmc_init(am335x_mmc);
    return;
}
```

Registers MMC init data with Linux

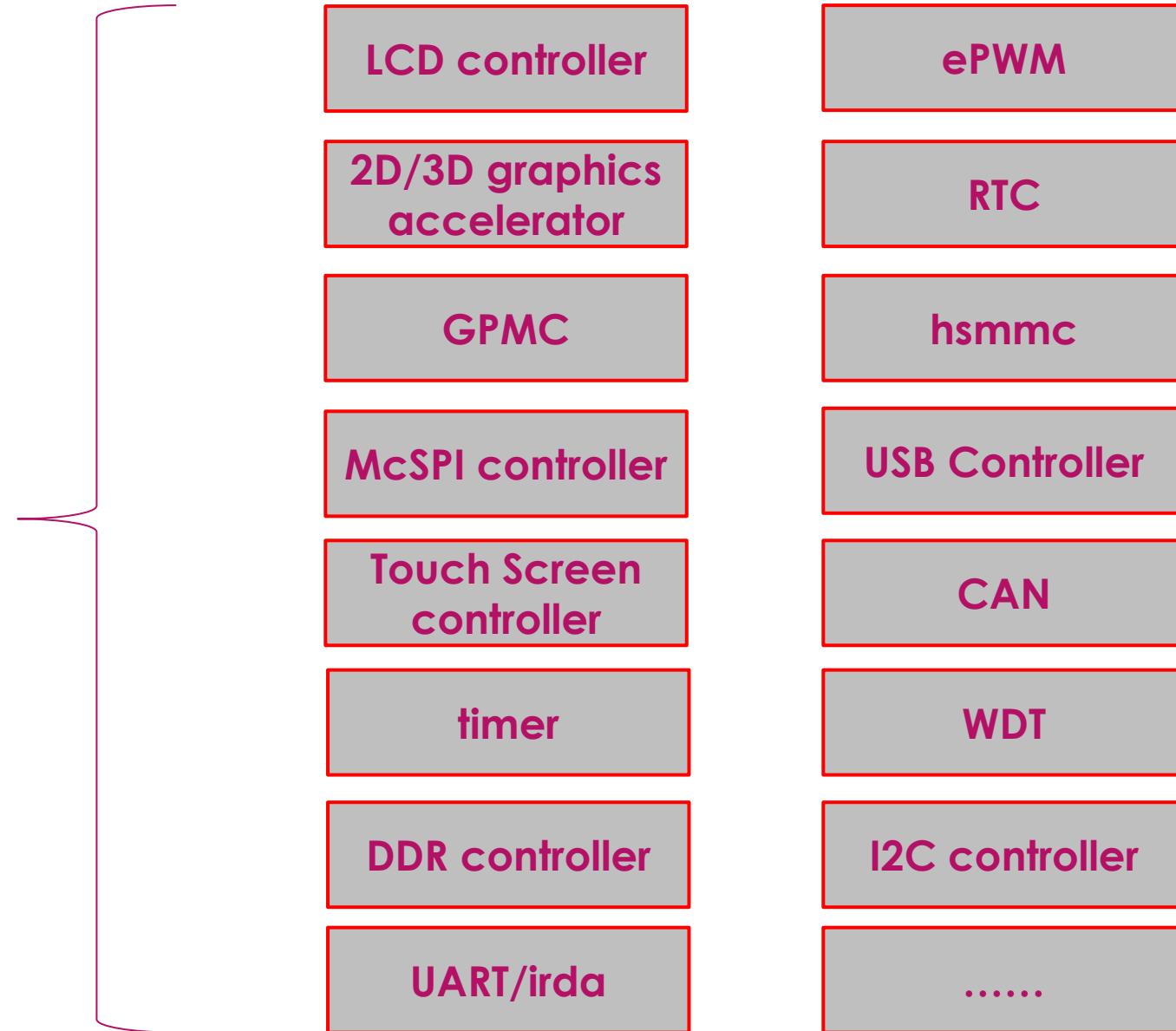
Steps to adding Ethernet to target board file

- ▶ Review system info to see how peripheral is attached
- ▶ Pin Mux
 - ▶ Use the Pin Mux Utility to configure Pin Init data
- ▶ Device/Platform Initialization data
 - ▶ None required for this integration
- Create Device Init function
 - ▶ Additional Init code required outside the board file
- Add Device Init Function to EVM Init Function





Locating Device Drivers
for your SOC Peripherals.



User Space

`drivers/video/fbdev/`

`drivers/input/touchscreen/ti_tsc.ko`

**Linux drivers
subsystem**

LCD controller

Touch screen
controller

SOC



Board

User Space

**Linux drivers
subsystem**

`drivers/mmc/host/omap_hsmmc.c`

3x Hsmmc

`drivers/spi/spi-omap2-
mcspi.c`

3x McSPI
controller

`drivers/i2c/busses/i2c-omap.c`

3x I2C controller

SOC

MMC/SD/SDIO
cards

sensors

Sensor/EEPROM/
etc

Board

User Space

`drivers/memory/omap-gpmc.c`

GPMC

NAND/NOR
flash

External SRAM

`drivers/usb/musb/musb_core.c`
`drivers/usb/musb/musb_gadget.c`
`drivers/usb/musb/musb_host.c`
`linux/drivers/usb/phy/phy-am335x.c`

**Linux drivers
subsystem**

USB controller

USB devices

SOC

Board

User Space

Linux drivers
subsystem

drivers/tty/serial/omap-serial.c

6 x UART

SOC

MODEMS/GPS/SENSORS

Board

User Space

`drivers/watchdog/ omap_wdt.c`

`drivers/rtc/rtc-omap.c`

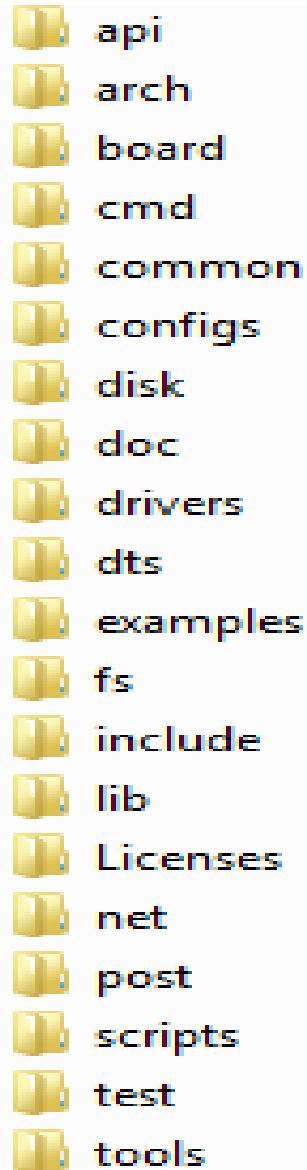
**Linux drivers
subsystem**

Watch dog
timer

RTC

SOC

Board



Board configuration defines

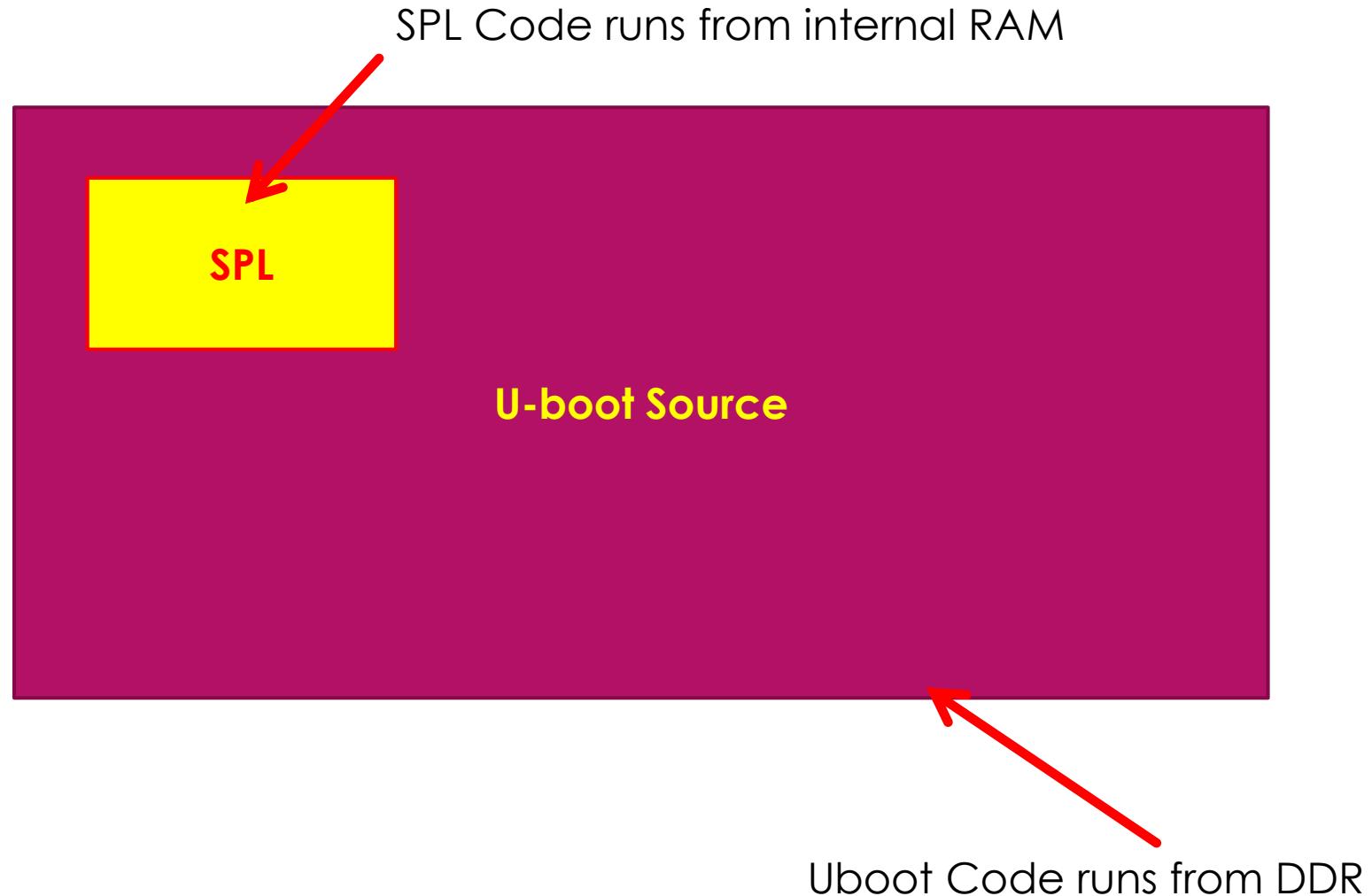
Board default configuration files

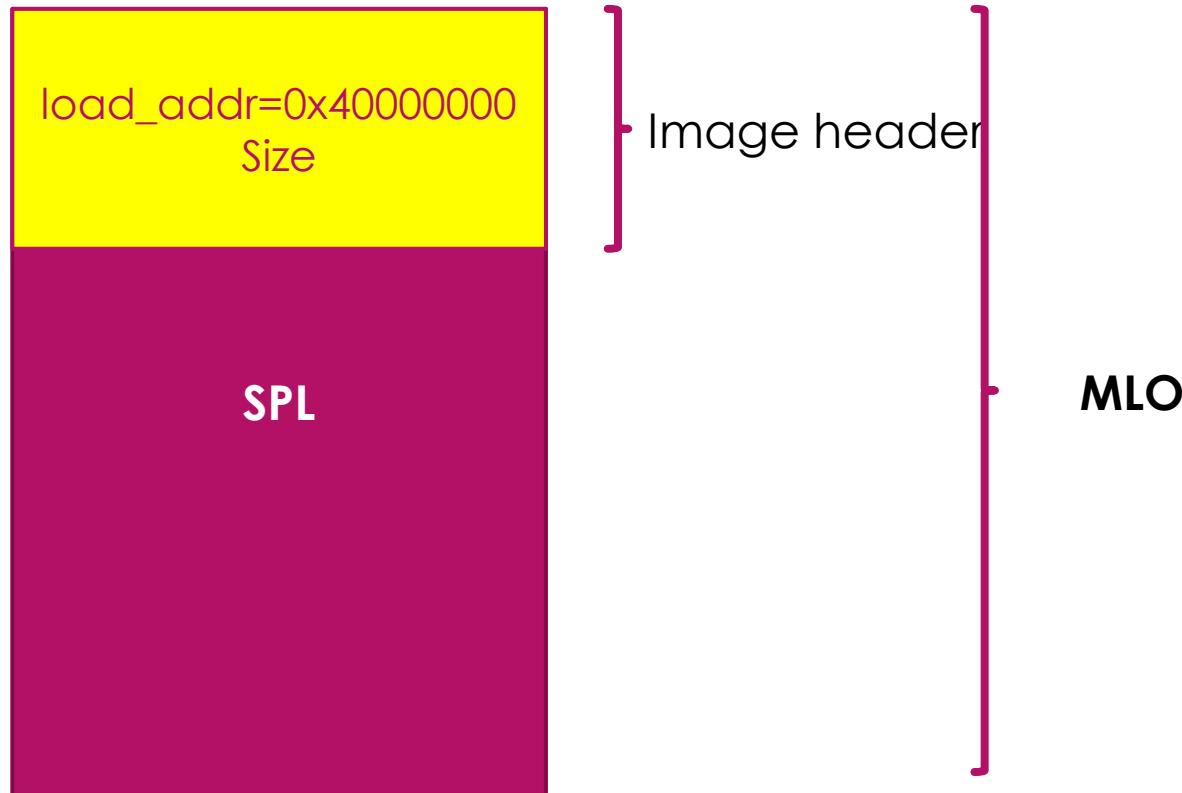
CPU Specific codes

Board Specific codes

Common code (Board independent)

Uboot vs SPL





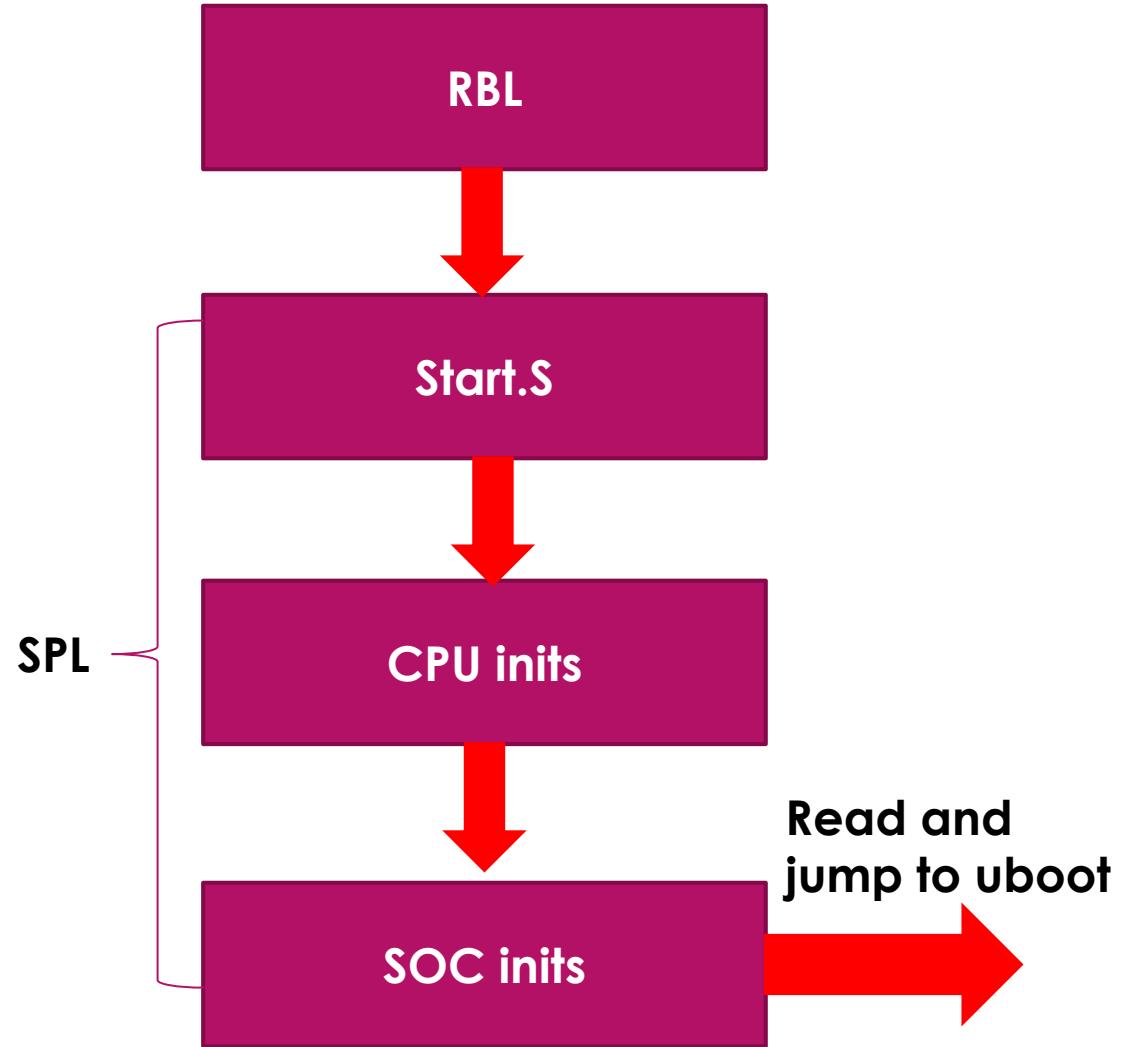
SPL and MLO are almost same except the fact that, MLO has a header which contains some info like load address, size of the image to follow,etc



MLO is produced by processing SPL using “mkImage” command which attaches the header info to “SPL” image.

Understanding Linux Source Tree.

ARM Board configuration files organization



Beagle_defconfig

U-boot



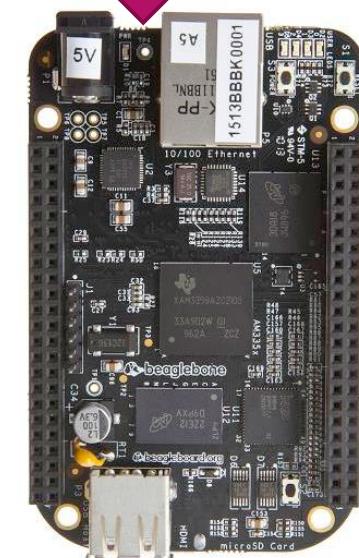
evm_defconfig

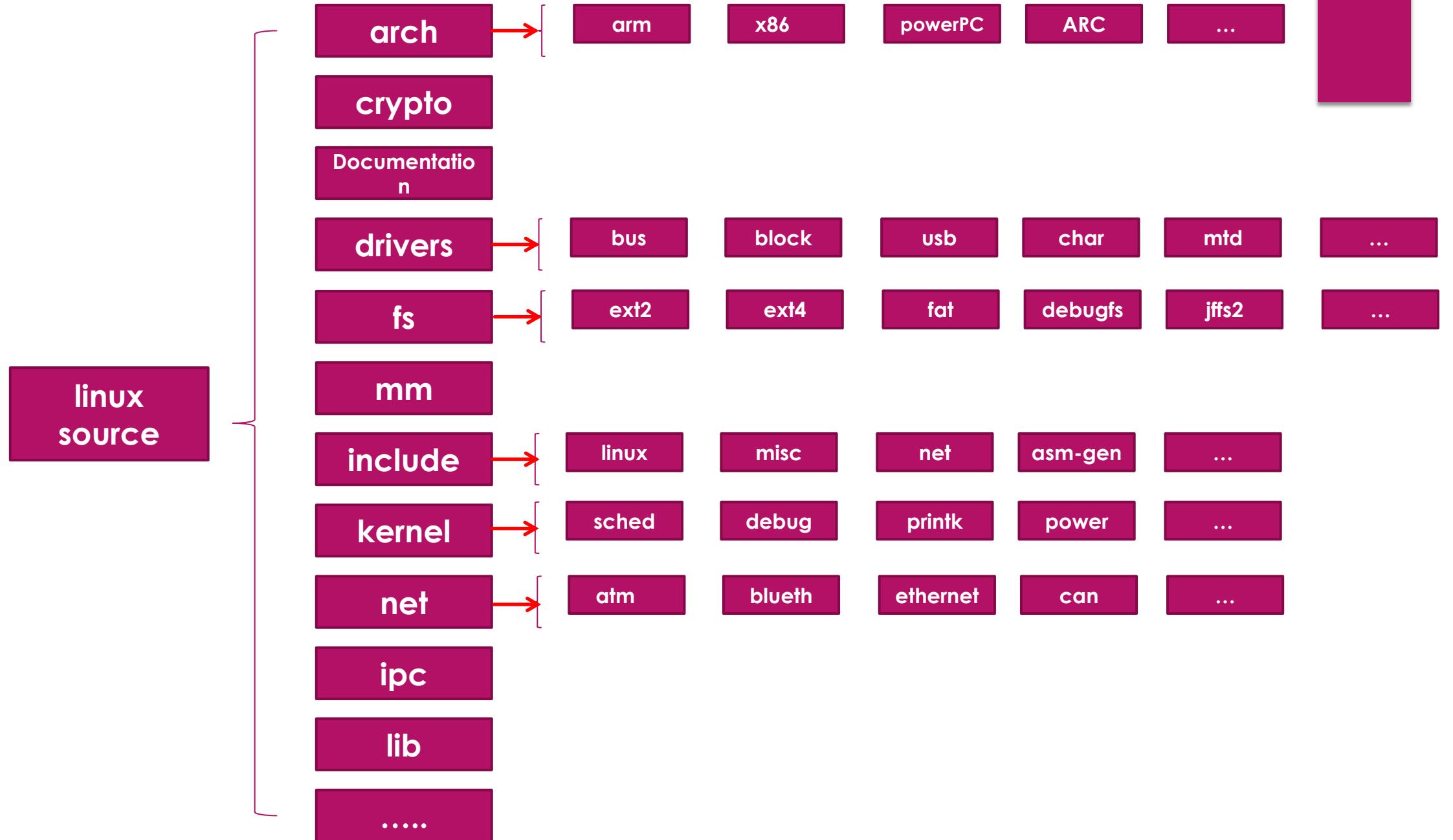
U-boot



bone_defconfig

U-boot





Linux different Processor Architecture Support

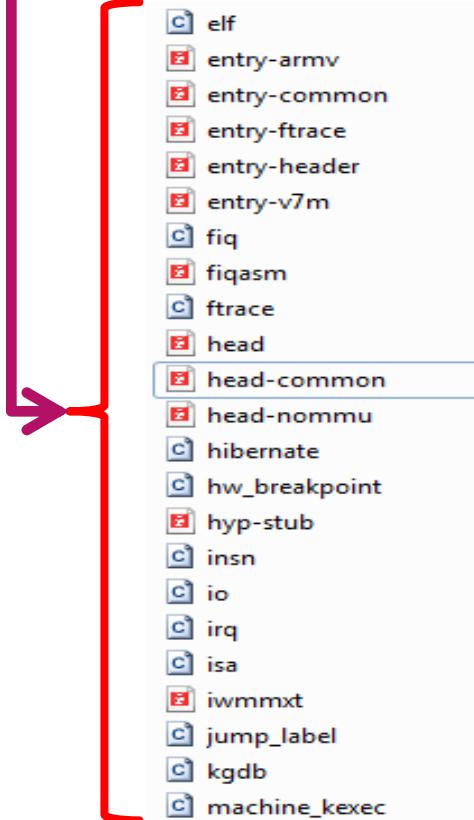
linux_src/arch/

-  alpha
-  arc
-  arm
-  arm64
-  avr32
-  blackfin
-  c6x
-  cris
-  frv
-  h8300
-  hexagon
-  ia64
-  m32r
-  m68k
-  metag
-  microblaze
-  mips
-  mn10300
-  nios2
-  openrisc
-  parisc
-  powerpc
-  s390

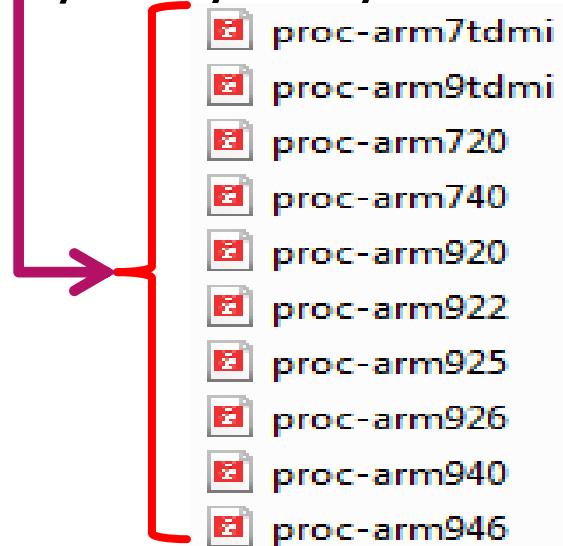
ARM code organization in Linux

Processor (CPU) specific codes go here

`arch/arm/kernel/`

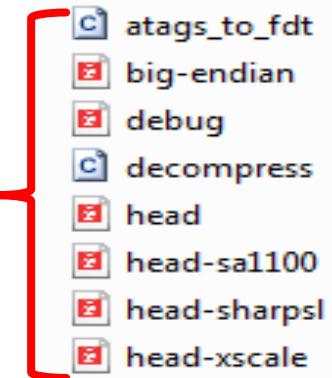


`arch/arm/mm/`



`arch/arm/boot/lib/`

`arch/arm/boot/compressed/`



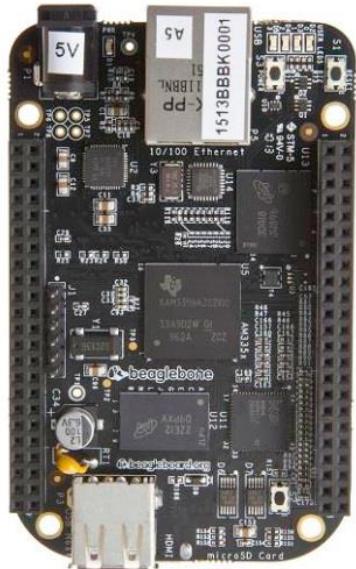
Processor

SOC (AM3358BZCZ100)

Board

ARM
Cortex-A8
Up to 1 GHz

32KB and 32KB L1 + SED
256KB L2 + ECC
176KB ROM | 64KB RAM



Graphics
PowerVR
SGX
3D GFX

Crypto
64KB
shared
RAM

Display

24-bit LCD controller
Touch screen controller

PRU-ICSS

EtherCAT, PROFINET,
EtherNet/IP,
and more

L3 and L4 interconnect

Serial

UART x6
SPI x2
I²C x3
McASP x2
(4 channel)
CAN x2
(Ver. 2 A and B)
USB 2.0 HS
OTG + PHY x2

System

eDMA
Timers x8
WDT
RTC
eHRPWM x3
eQEP x3
PRCM

Parallel

eCAP x3
ADC (8 channel)
12-bit SAR
JTAG
Crystal
Oscillator x2

Memory interface

mDDR(LPDDR), DDR2,
DDR3, DDR3L
(16-bit; 200, 266, 400, 400 MHz)
NAND and NOR (16-bit ECC)

512MB
DDR3L

USB HOST

2G eMMC

uSD

PHY

USB

HDMI

LED
4x

Processor

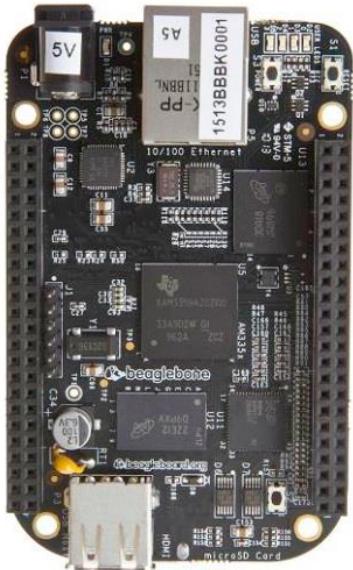
SOC
(AM3358BZCZ100)

Board

Processor (CPU) specific codes go here

ARM
Cortex-A8
Up to 1 GHz

32KB and 32KB L1 + SED
256KB L2 + ECC
176KB ROM 64KB RAM



arch/arm/kernel/

- elf
- entry-armv
- entry-common
- entry-ftrace
- entry-header
- entry-v7m
- fiq
- fiqasm
- ftrace
- head
- head-common
- head-nommu
- hibernate
- hw_breakpoint
- hyp-stub
- insn
- io
- irq
- isa
- iwmmxt
- jump_label
- kgdb
- machine_kexec

arch/arm/mm/

- proc-arm7tdmi
- proc-arm9tdmi
- proc-arm720
- proc-arm740
- proc-arm920
- proc-arm922
- proc-arm925
- proc-arm926
- proc-arm940
- proc-arm946

arch/arm/boot/compressed/

- atags_to_fdt
- big-endian
- debug
- decompress
- head
- head-sa1100
- head-sharpsl
- head-xscale

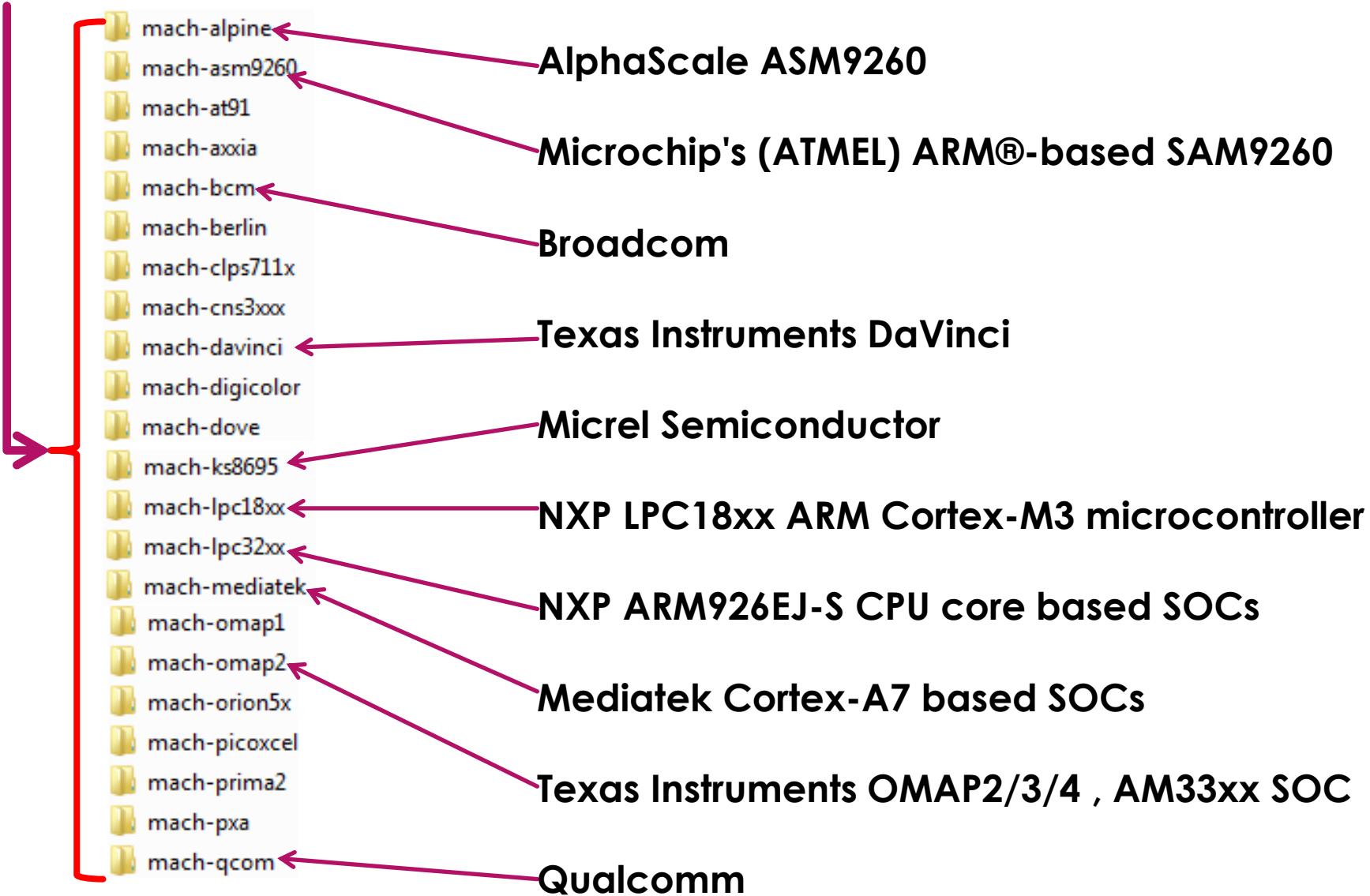
arch/arm/boot/lib/

mDDR(LPDDR), DDR2,
DDR3, DDR3L
(16-bit; 200, 266, 400, 400 MHz)
NAND and NOR (16-bit ECC)

LED
4x

SOC & board specific codes go here

arch/arm/



SOC & board specific codes

`arch/arm/mach- <*>`



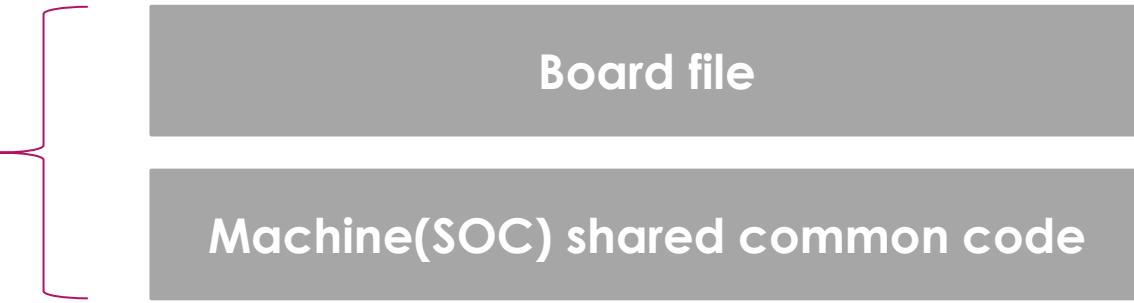
Board file

Machine(SOC) shared common code

The board file is a ‘C’ source file, which explains the various peripherals on the board (out side the SOC). This file usually contains the registration functions and data of various on board devices, e.g for Ethernet phy, eeprom, leds, lcds, etc. Basically the board vendor uses this board file to register various board peripherals with the Linux subsystem.

SOC & board specific codes

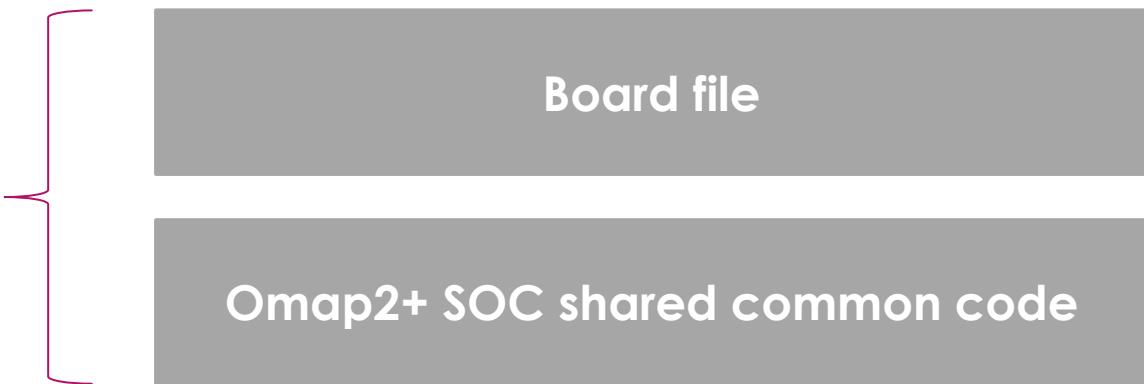
arch/arm/mach- <*>



machine shared common code, contains various ‘C’ source files which contain the helper functions to initialize and deal with the various on chip peripherals of the SOC and these codes are common among all the SOCs which share common IP for the peripherals.

SOC & board specific codes

arch/arm/mach-omap2



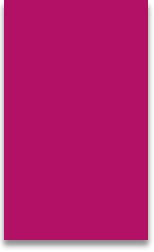
arch/arm/mach- <*>



Board file

Machine(SOC) shared common code

The board file is a 'C' source file, which explains the various peripherals on the board (out side the SOC). This file usually contains the registration functions and data of various on board devices, e.g for Ethernet phy, eeprom, leds, lcds, etc. Basically the board vendor uses this board file to register various board peripherals with the Linux subsystem.



machine shared common code, contains various ‘C’ source files which contain the helper functions to initialize and deal with the various on chip peripherals of the SOC and these codes are common among all the SOCs which share common IP for the peripherals.



Everything

Search



Products

Applications & designs

Tools & software

Support & training

Order Now

About TI

My History

Cart

Recommendations from TI.com:

Reference designs selected for you

Products of interest for you

Top technical documents for you



TI Home > Semiconductors > Processors > Sitara Processors > ARM Cortex-A8 > AM335x >

AM3358 (ACTIVE) Sitara Processor

[AM335x Sitara™ Processors \(Rev. J\)](#)

[AM335x Sitara Processors Silicon Errata \(Revs 2.1, 2.0, 1.0\) \(Rev. I\)](#)

Description & parametrics

Online datasheet

Technical documents

Tools & software

Order Now

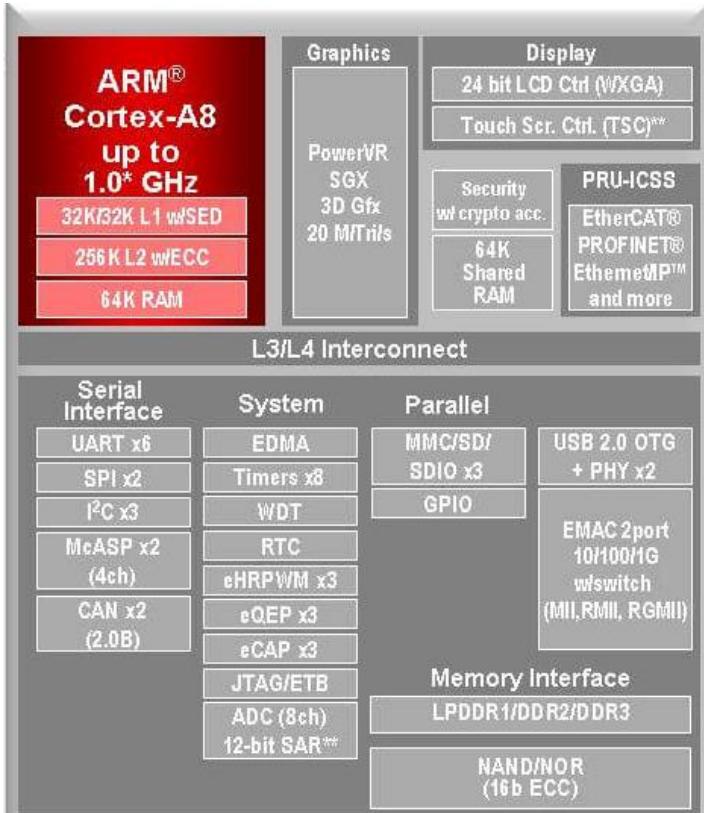
Compare

Quality & packaging

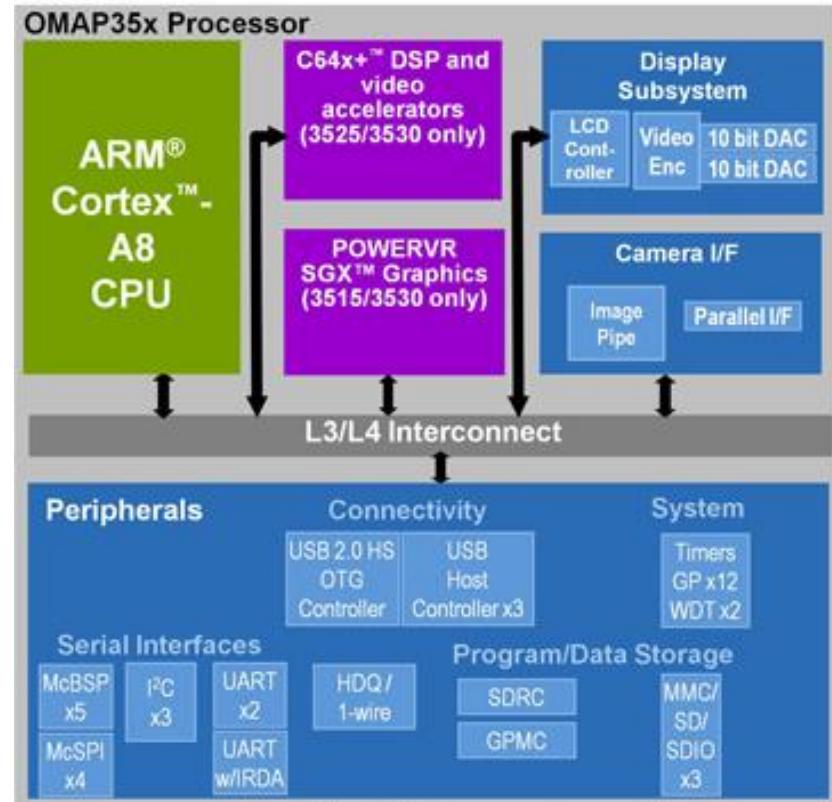
Supp

[Description](#) | [Features](#) | [Parametrics](#) | [Diagrams](#) | [Related end equipment](#) | [Complete your design](#)

AM335x



OMAP2/3/4/5



Same peripheral IPs



Note that OMAP and SITARA SOCs are almost same except the DSP engine, which is removed in SITARA AM335x

Serial	Sets up UARTs including pin mux	devices	Init calls, platform registration for most peripherals
gpio	Initialization functions	control	OMAP2 control registers
common	Init calls to define global address range for select interfaces	i2c	Reset and Mux functions
clocks	Define clock domain mgmt functions	mux	Defines a Pin mux abstracton with supporting functions
sdrc	Init functions for SDRC and SMS	voltage	Voltage domain support functions .
display	Display init calls, handles the difference between OMAP2,3,4	mmchs	Init functions, hw and platform data

OMAP2+ Machine Shared Common Code

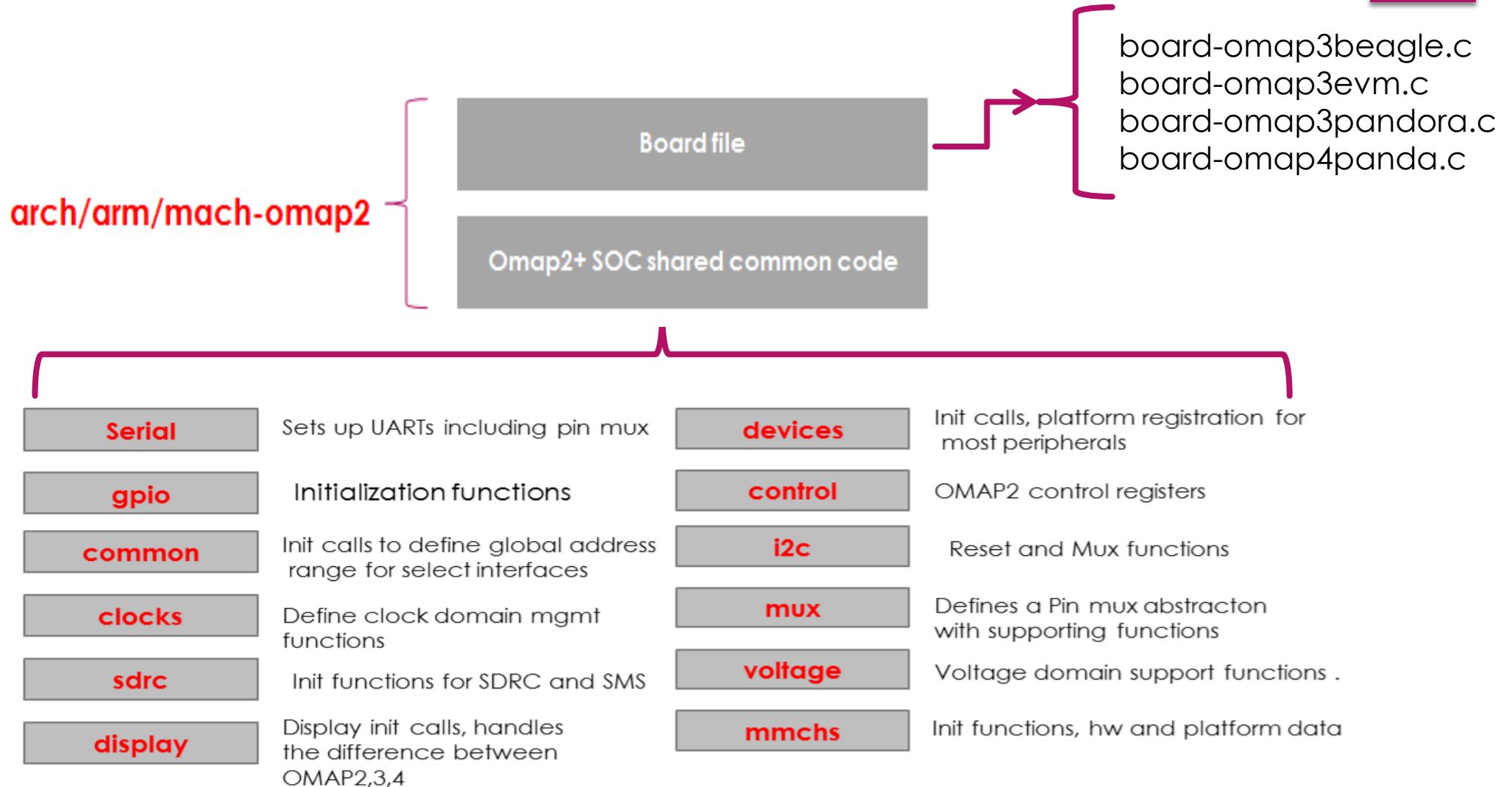
- ▶ There are several board files in the mach-omap2 directory. These board files typically use the support functions defined within this directory. Below is a sampling of some of the supporting common code, not all are mentioned here.

OMAP2 Machine Shared Common Code – arch/arm/mach-omap2

Not a complete listing of the interfaces, just a few are highlighted and to explain how they are used, review this directory to see additional interfaces

serial	Sets up UARTs including pin mux	gpio	Initialization function
devices	Init calls, platform registration for most peripherals	i2c	Reset and Mux functions
common	Init calls to define global address range for select interfaces	mux	Defines a Pin Mux abstraction with supporting functions
clocks	Define clock domain mgmt functions	hs mmc	Init functions, hw and platform data
control	OMAP2 control registers	sdrc	Init function for SDRC and SMS
display	Display init calls, handles the differences between OMAP2,3 and 4	voltage	Voltage domain support functions

SOC & board specific codes

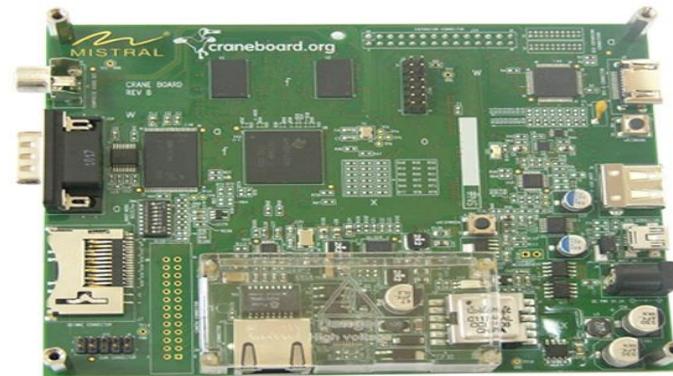


OMAP2430 sdp2430 board



board-2430sdp.c

AM3517 Craneboard



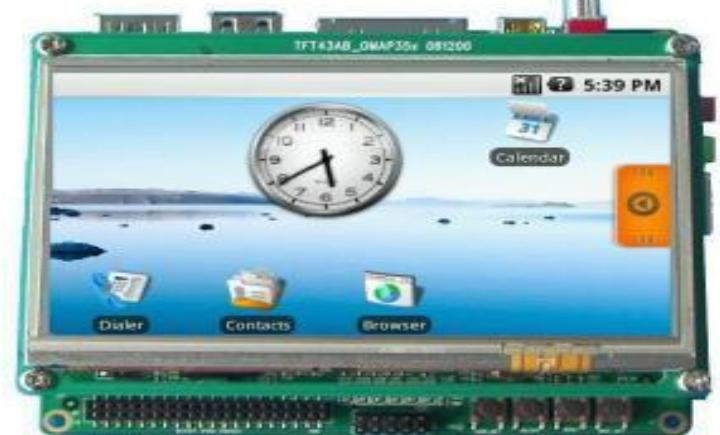
board-am3517crane.c

AM3517 EVM



board-am3517evm.c

OMAP3 Devkit8000



board-devkit8000.c

OMAP3 Beagle Board



board-omap3beagle.c

OMAP3 EVM



board-omap3evm.c



Manufacturer	OpenPandora GmbH
Type	Handheld game console / PDA hybrid
Release date	May 2010; 7 years ago
Operating system	Custom edition of Ångström Linux for gaming
CPU	OMAP3530 (600+ MHz Cortex-A8 (32-bit) and 430 MHz TMS320C64x+, NEON & TRADCE SIMD instructions set ^[1])
Memory	256 MB low power DDR-333
Storage	Dual SDHC slots, 512 MB internal NAND, USB external storage
Graphics	PowerVR SGX 530 at 110 MHz
Connectivity	Wi-Fi, USB 2.0, Bluetooth
Predecessor	GP2X (unofficial)
Successor	DragonBox Pyra ^[2]
Website	boards.openpandora.org



Always innovating Touchbook

board-omap3pandora.c

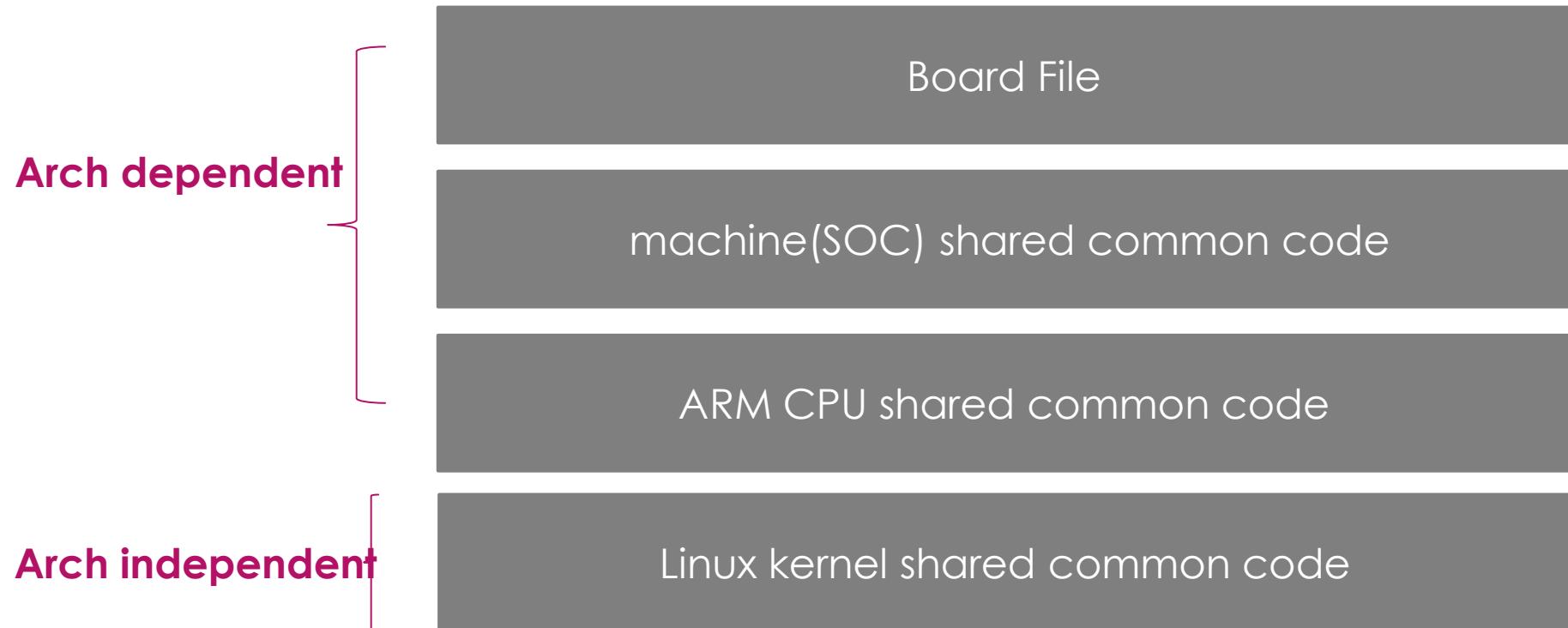
board-omap3touchbook.c

OMAP2 Board files

board-generic.c

- Generic OMAP2430
- Generic OMAP3
- Generic OMAP36xx
- Generic AM3517
- Generic ti814x
- Generic AM33XX
- Generic OMAP4
- Generic OMAP5
- Nokia RX-51 board
- etc

ARCH. Specific Code organization in Linux



Processor + SOC + Board

`arch/arm/machine-omap2/board-beagle.c`

`arch/arm/machine-omap2/board-pandora.c`

`arch/arm/machine-*/board-*.c`

Board File

Machine(SOC) shared common code

ARM CPU shared common code

Linux kernel shared common code

`arch/arm/machine-omap2/`
`arch/arm/machine-at91/`
`arch/arm/machine-*/`

`arch/arm/kernel/`
`arch/arm/mm/`
`arch/arm/boot/compressed/`
`arch/arm/boot/lib/`

Sched , Memory management, drivers, VFS , IPC, Security , Crypto ,

Processor + SOC + Board

`arch/arm/kernel/`

`arch/arm/mm/`

`arch/arm/boot/compressed/`

`arch/arm/boot/lib/`

`arch/arm/machine-omap2/board-beagle.c`

`arch/arm/machine-omap2/board-pandora.c`

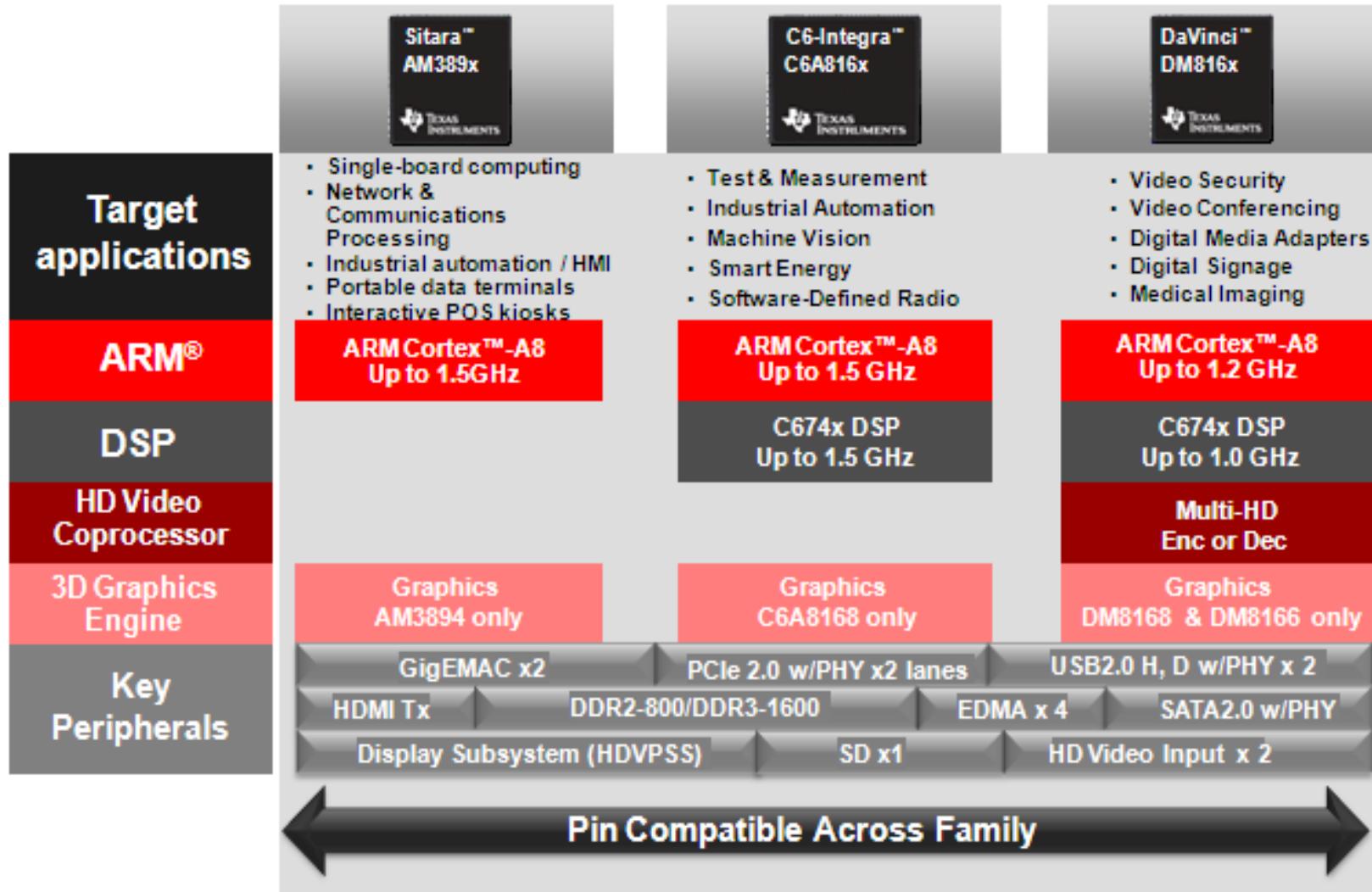
`arch/arm/machine-*/board-*.c`

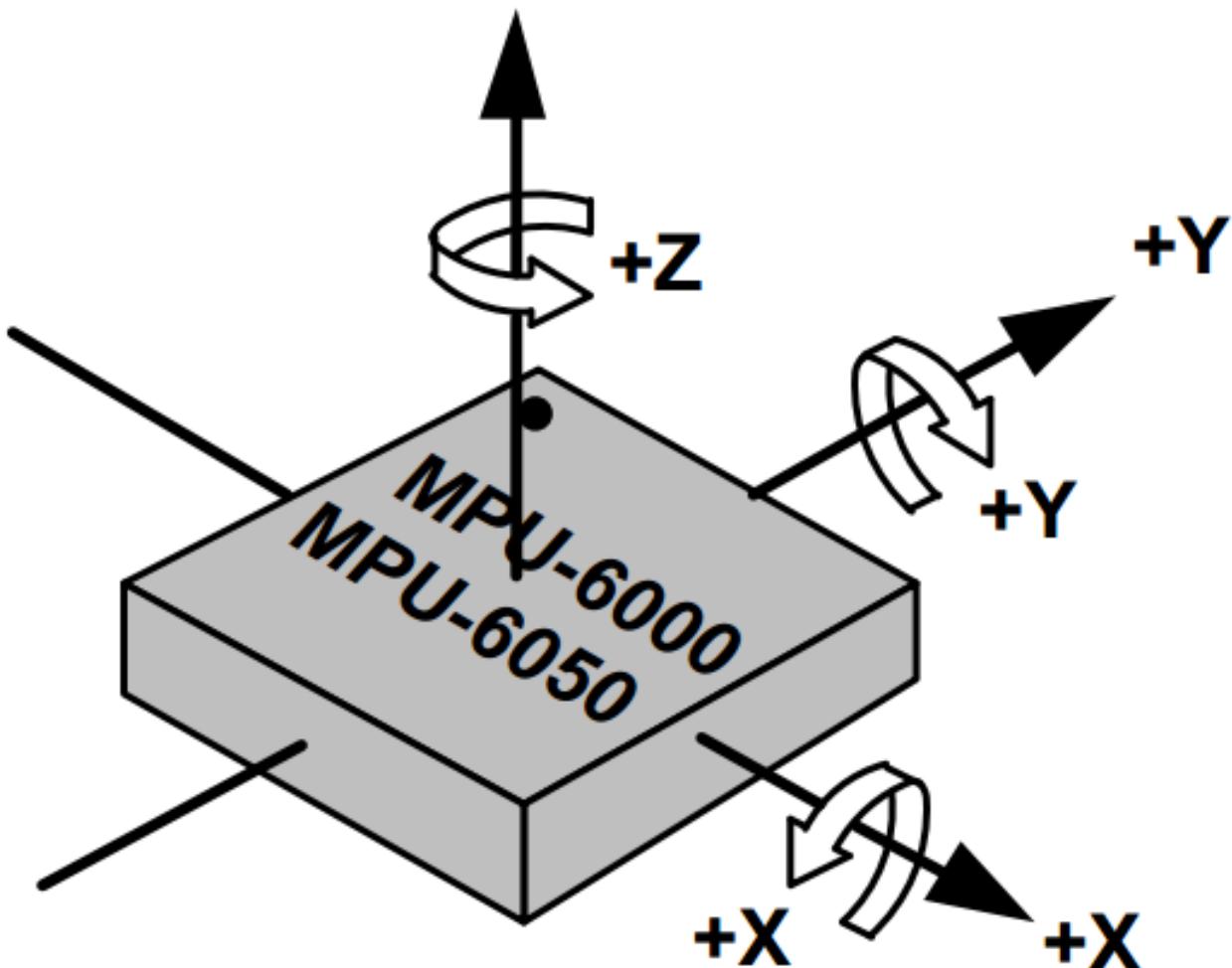
`arch/arm/machine-omap2/`

`arch/arm/machine-at91/`

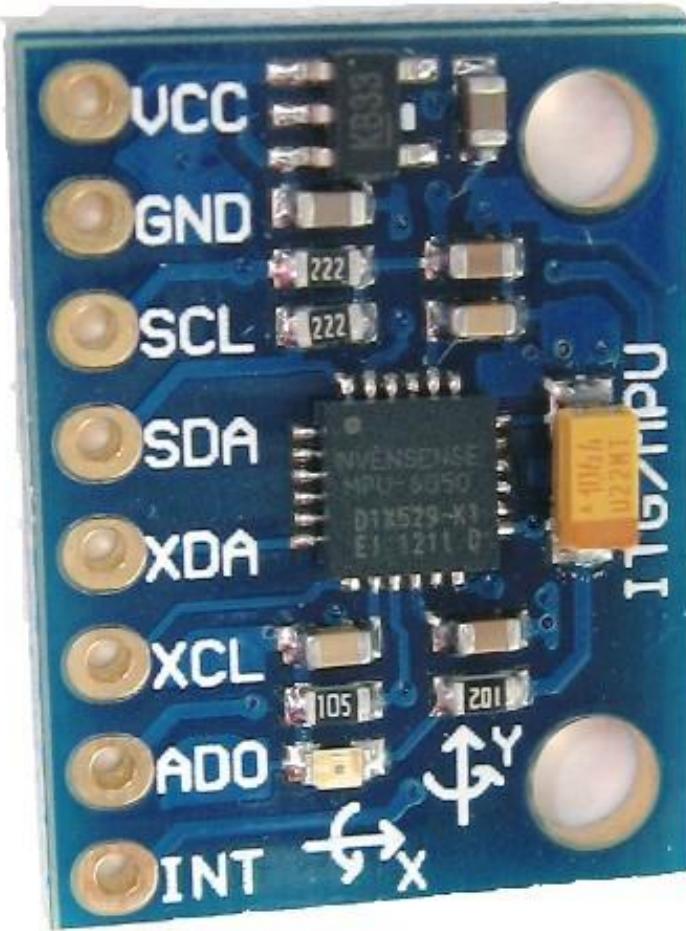
`arch/arm/machine-*/`

Sitara Vs DaVinci

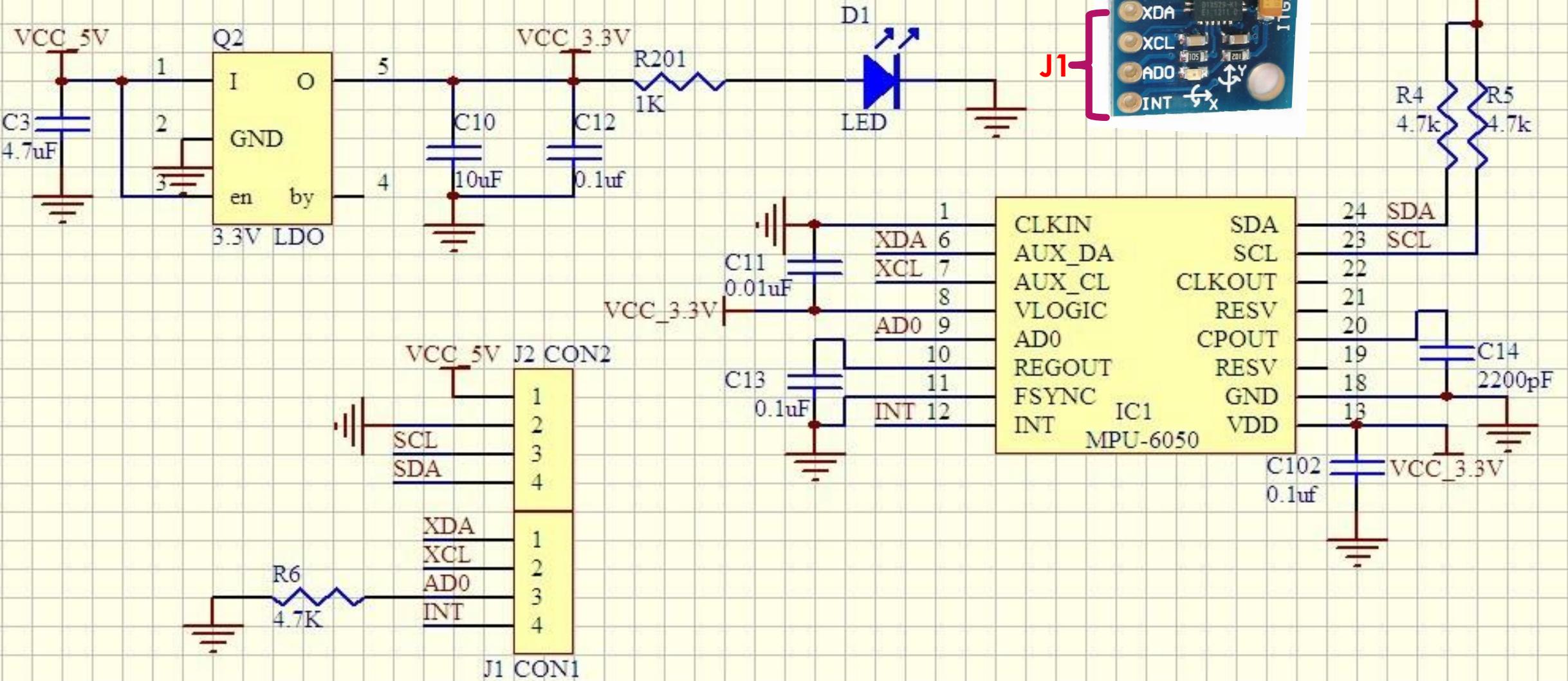




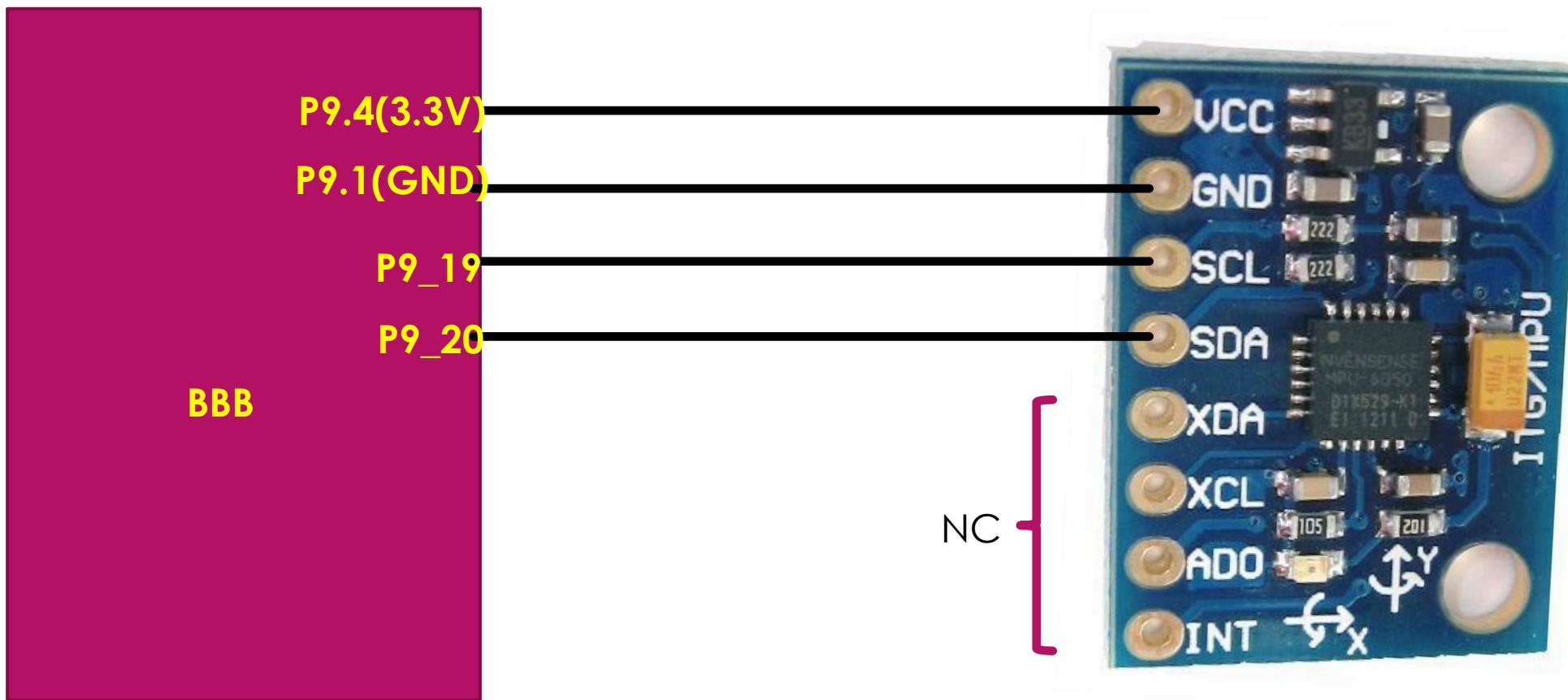
MPU6050 GY-251 Breakout board

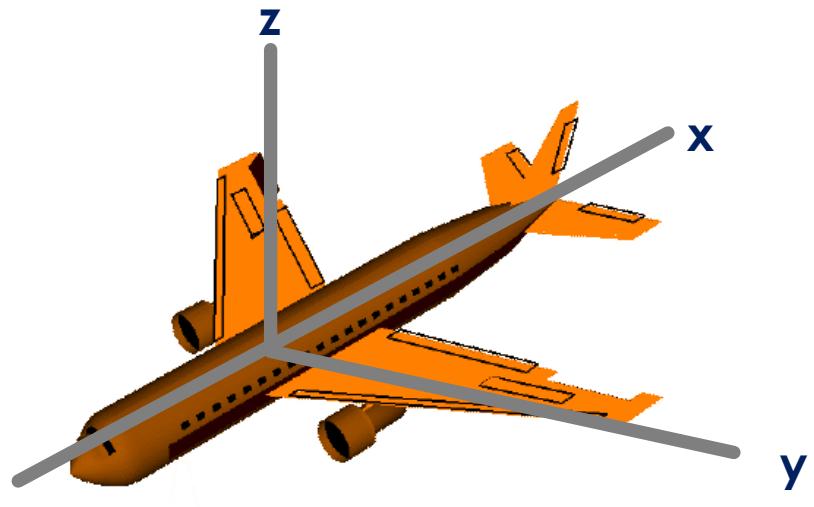


If $AD0 = LOW$, then address = $b1101000 = 0x68$
If $AD0 = HIGH$, then address = $b1101001 = 0x69$

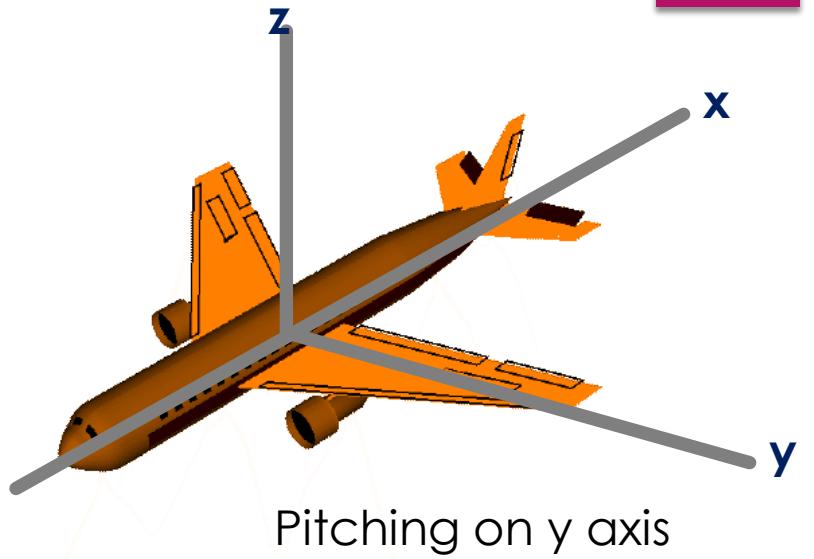


Connection between MPU6050 breakout board and BBB

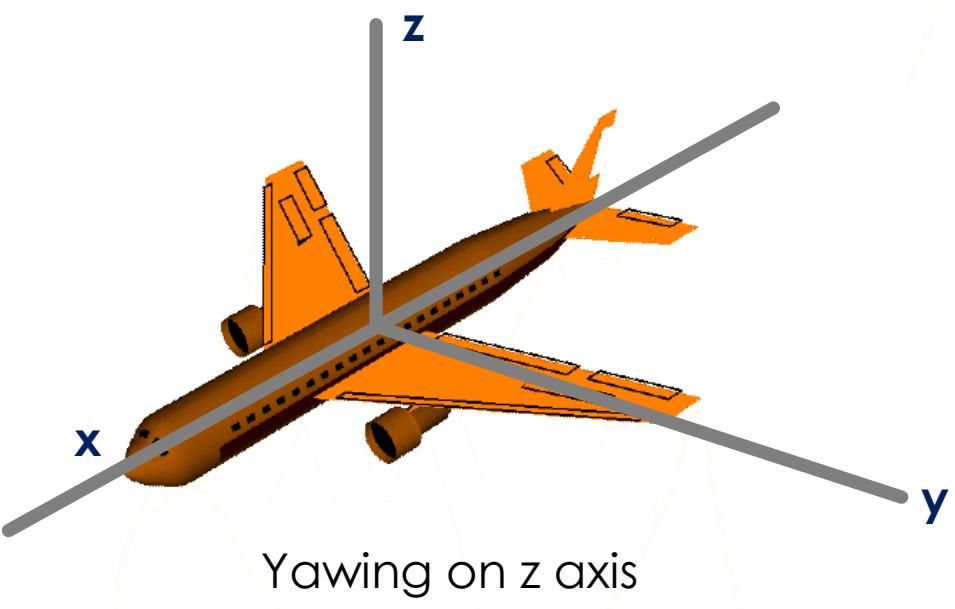




Rolling on X axis

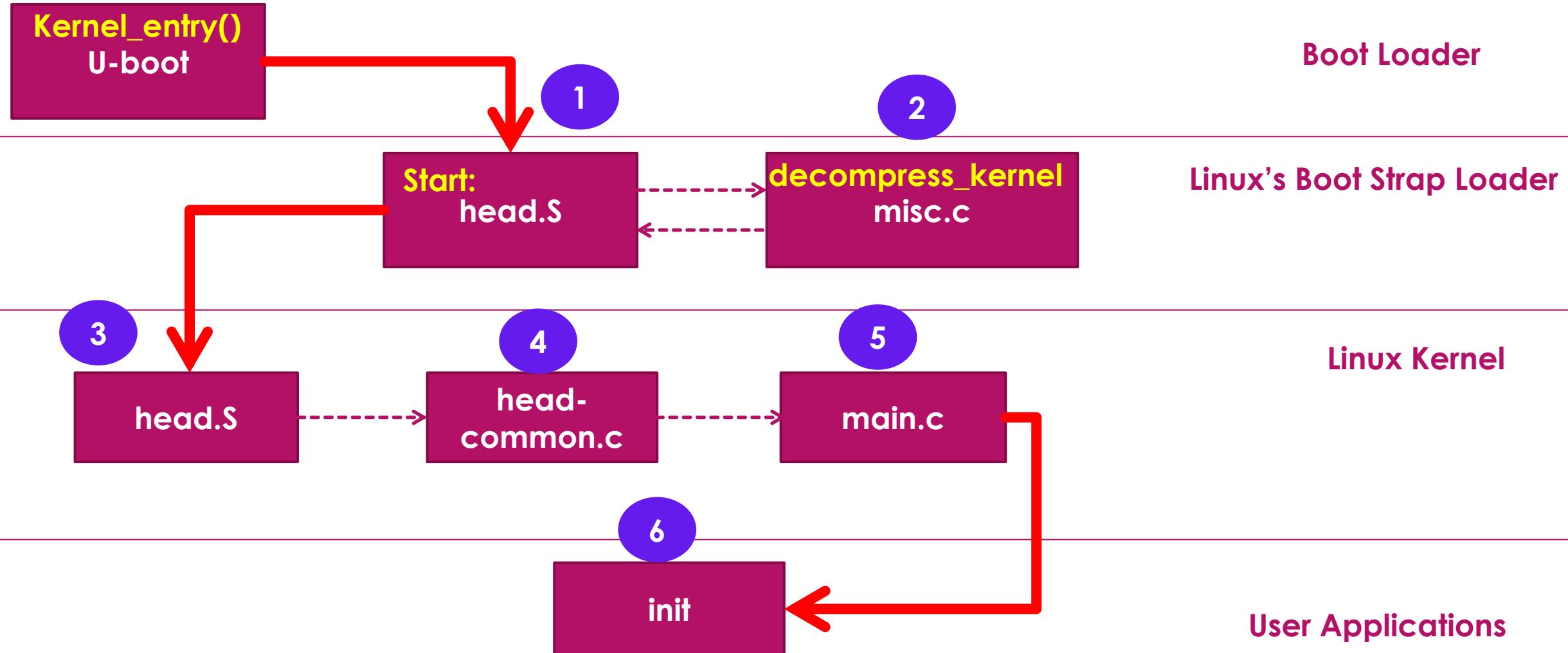


Pitching on y axis

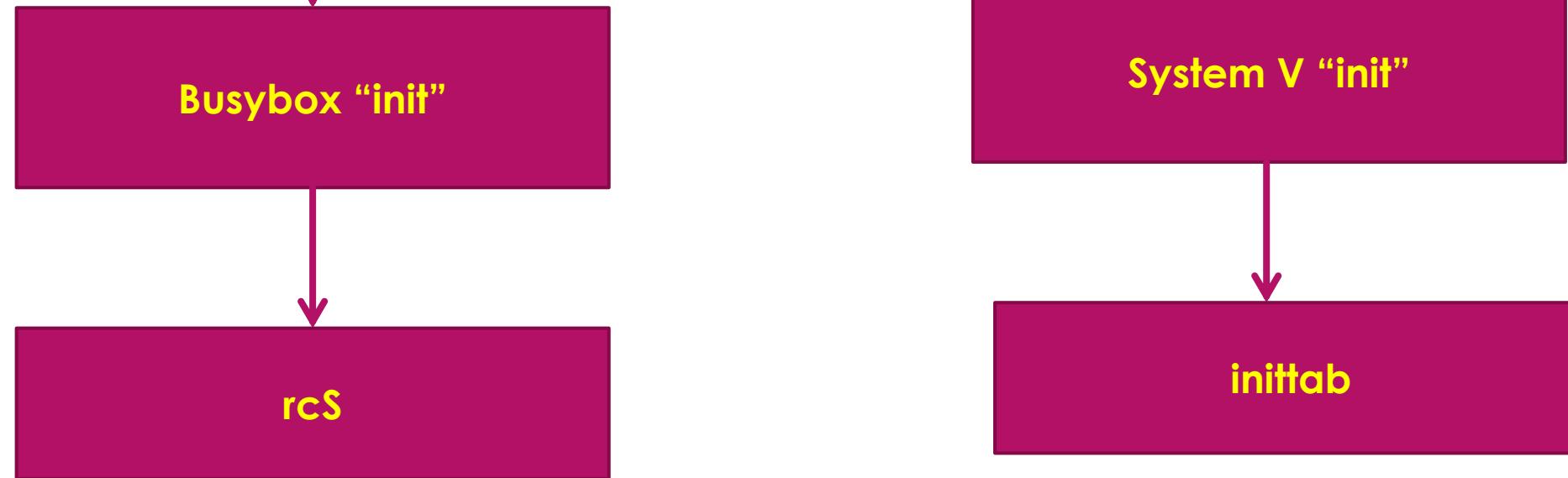


Yawing on z axis

Control Flow during Linux boot



Linux “init” Program



Busybox’s “init” looks for rcS script

System V “init” looks for inittab script

/etc/init.d/

rcS

Use rcS to launch services/daemons during startup

Start networking

Start sshd

Start NFS

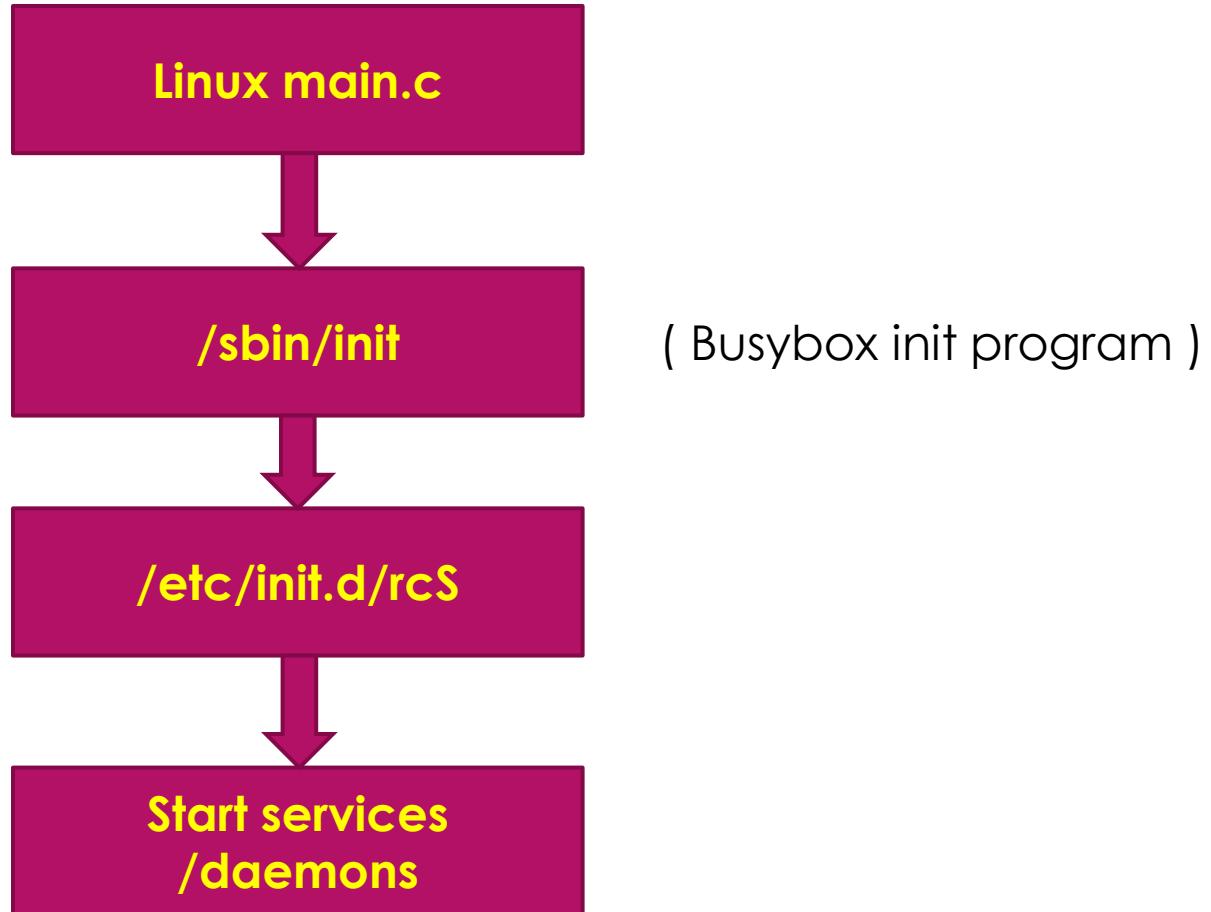
Load modules

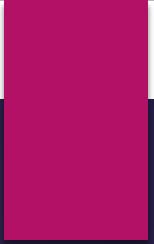
Start daemons

Start telnet

Mount FS.... etc

Conclusion





How to connect 7Segment Display to MCU ??

How to execute this application ?

./ BBB_led_control trigger heartbeat

./ BBB_led_control trigger oneshot

./ BBB_led_control trigger none

./ BBB_led_control brightness 1

./ BBB_led_control brightness 0



In this section you will learn,

- 1. AM335x GPIO subsystem and Expansion header details**
- 2. Sysfs entries to control the on board “LEDs”**
- 3. Interfacing and controlling external LED connected to BBB**
- 4. Writing ‘C’ application to control LEDs**

AM335x GPIOs

GPIO module
0

32 gpio pins

GPIO module
1

32 gpio pins

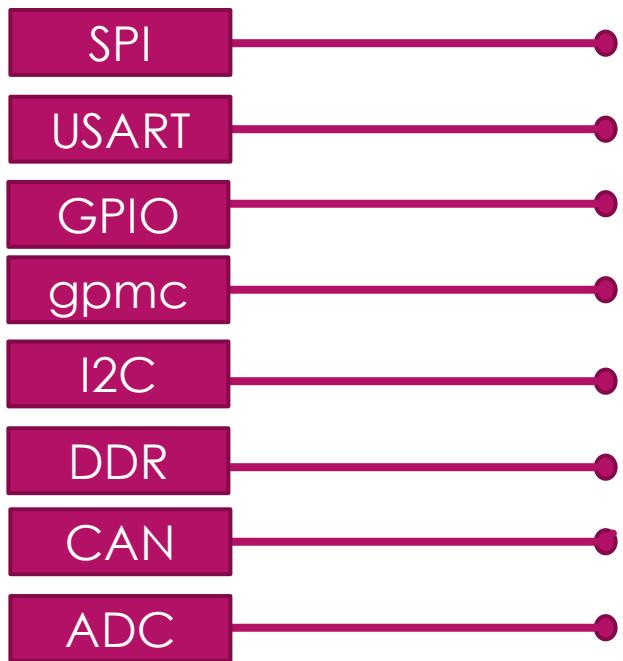
GPIO module
2

32 gpio pins

GPIO module
3

32 gpio pins

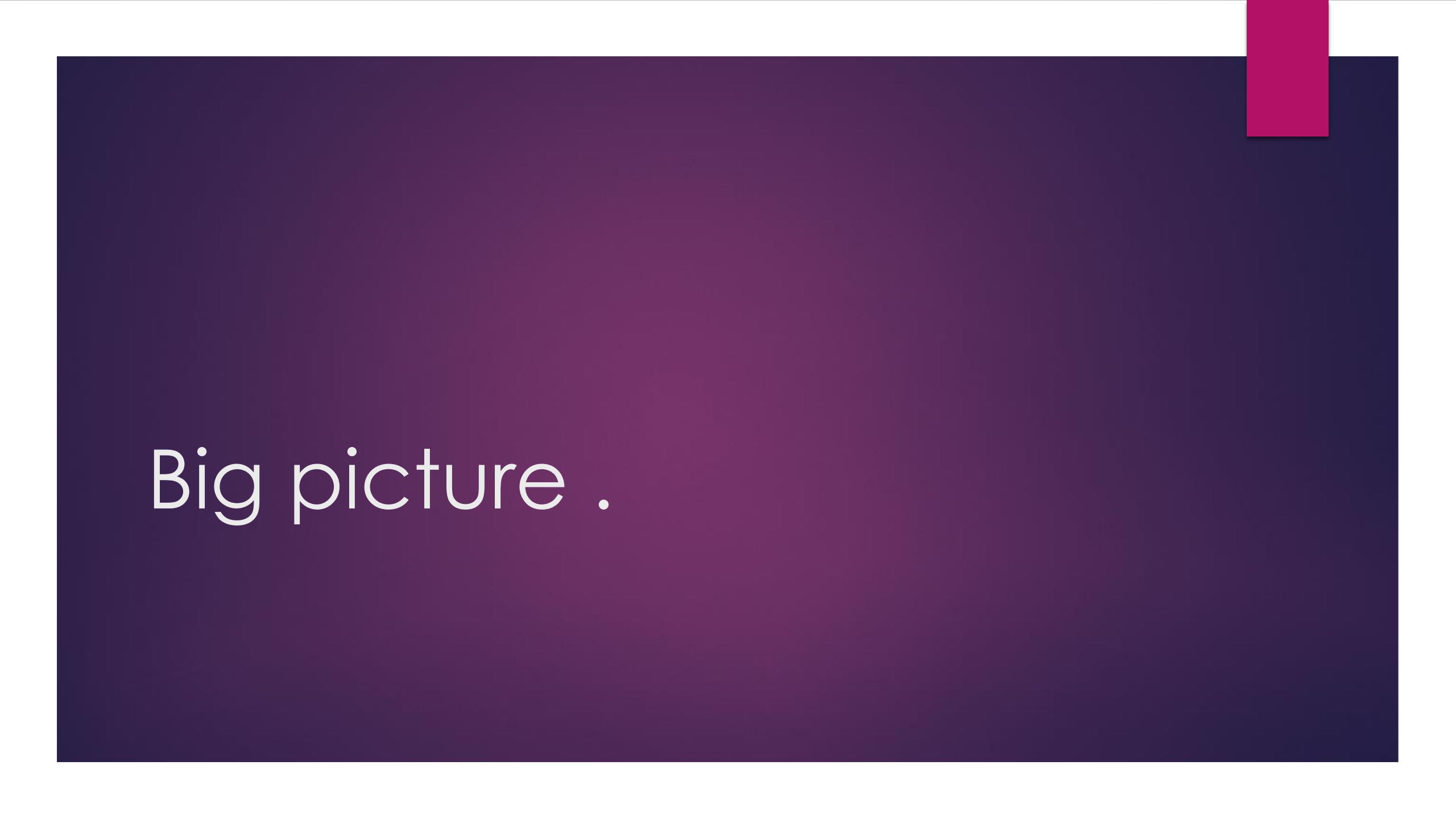
On chip Peripherals



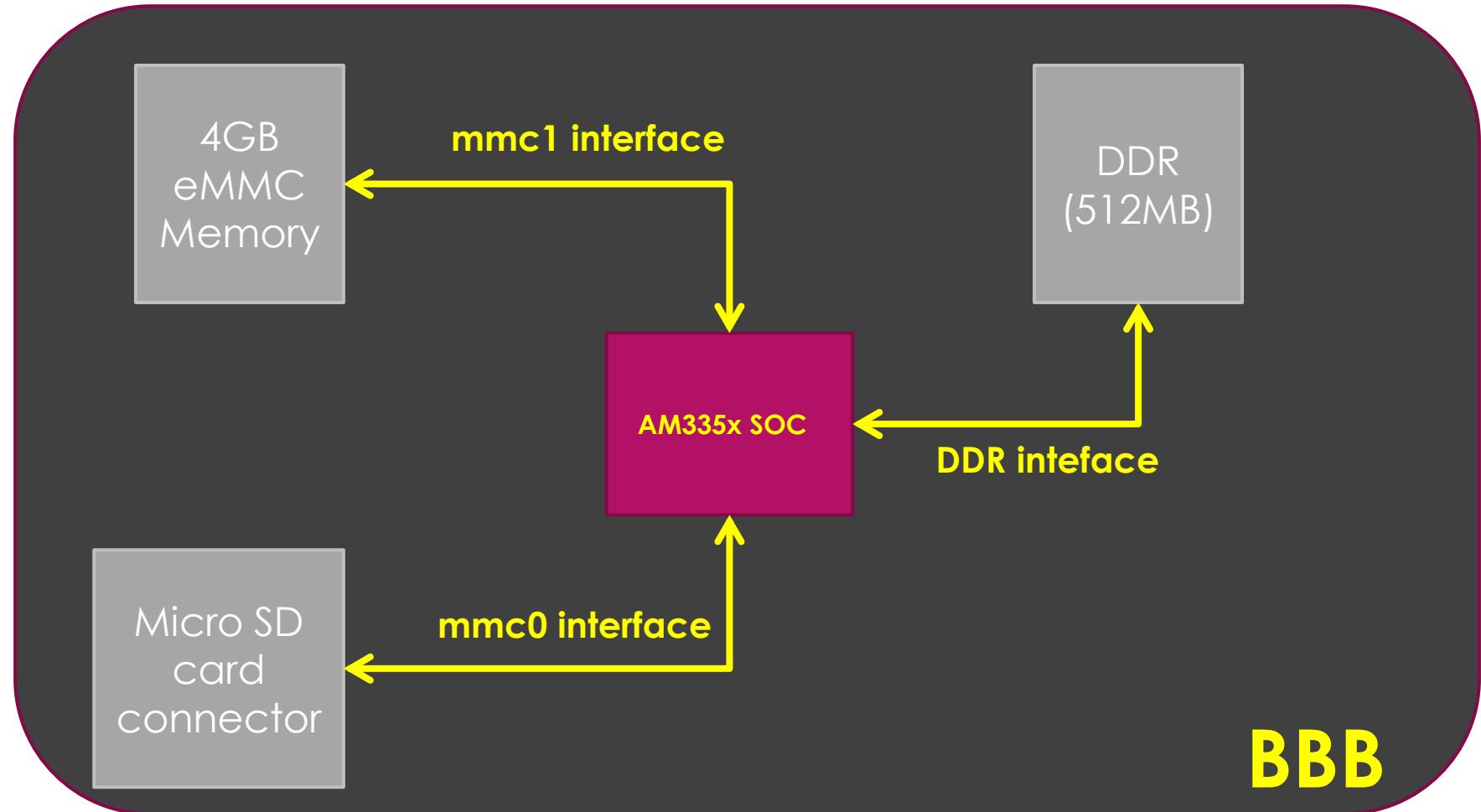
PIN MUXING
REGISTER

Physical pin of a MCU

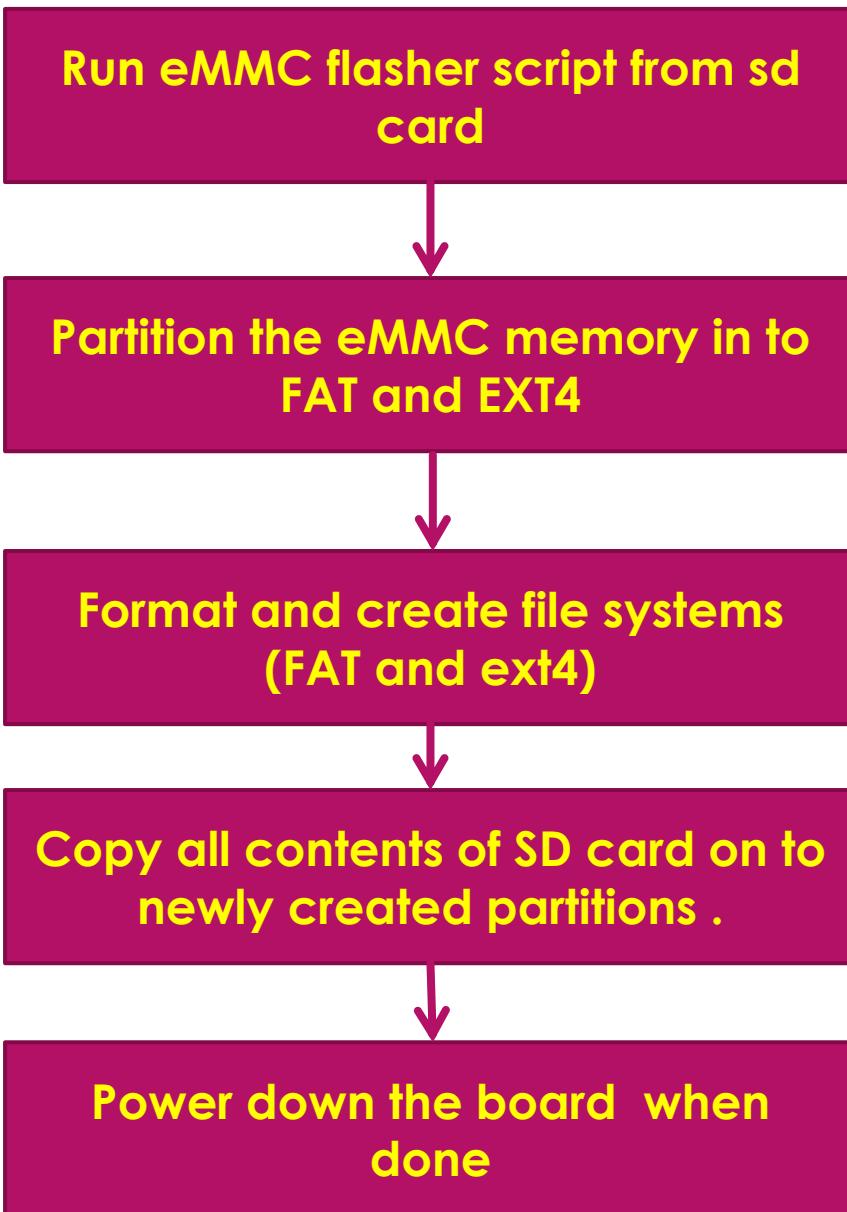
BBB eMMC update: Networking Configuration



Big picture .



- 1) Download the latest Debian OS**
- 2) Write the bootable image to the SD card first**
- 3) Boot the board from SD card**
- 4) Execute the eMMC flasher script**
- 5) eMMC flasher script will flash all the contents of the sd card on to the eMMC memory !**





In this section you will learn,

- ▶ Updating the eMMC memory with the latest debian os image both in windows and ubuntu host scenarios
- ▶ BBB Networking configurations in both windows and ubuntu host scenarios

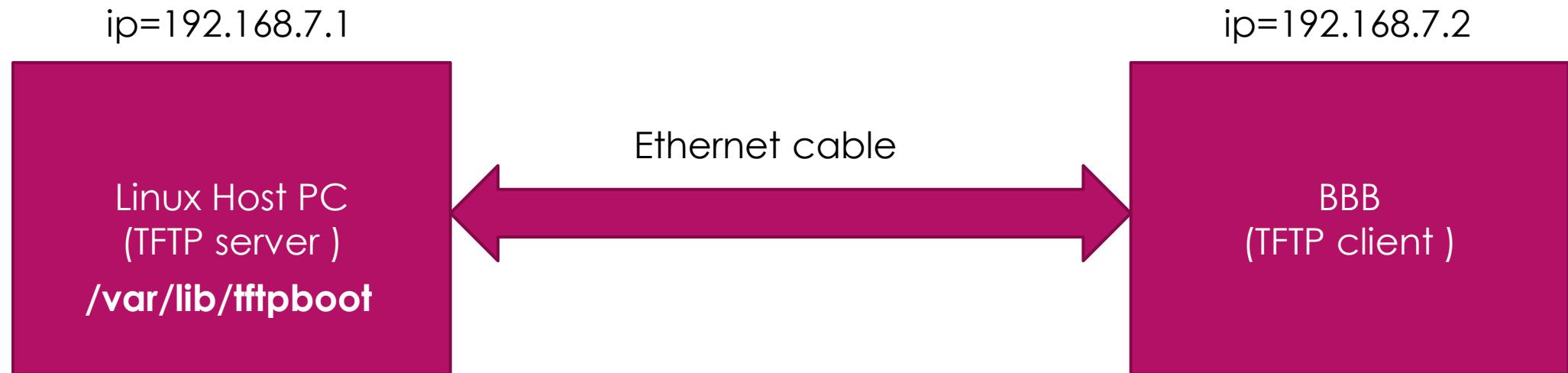


TFTP boot : Big picture



TFTP stands for *Trivial File Transfer Protocol* , which can be used to transfer files between a TFTP server and a TFTP client.

BBB booting using TFTP Protocol



Linux host PC
/var/lib/tftpboot

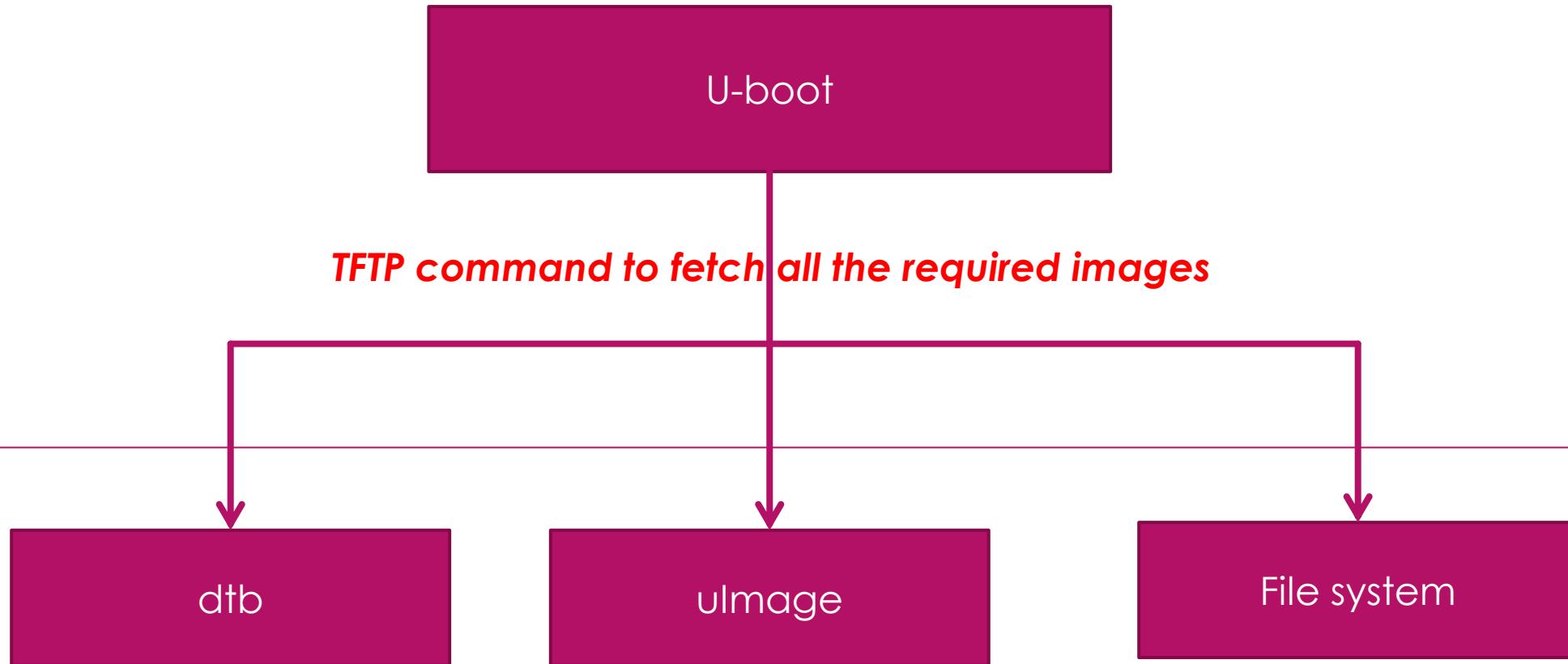
am335x-boneblack.dtb
ulimage
Initramfs

2) We transfer these images using TFTP from HOST to board over Ethernet

SD card

SPL
u-boot.img
uEnv.txt

1) First we boot the board via SD card with these images up to u-boot !!!



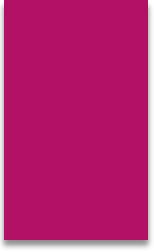
Images stored on the TFTP server(HOST PC)

Prepare uEnv.txt

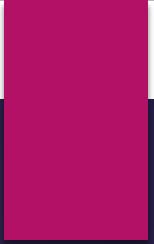
We should prepare a `uEnv.txt` file with all the required commands to automate the TFTP boot process !

Summary

- 1) we will keep the primary boot images like SPL, uboot and uEnv.txt on the SDcard .
- 2) we will keep other boot images like linux Kernel image, dtb and the initramfs on the linux host PC at the location **/var/lib/tftpboot**
- 3) After that we first boot the board via sd card up to uboot.
- 4) Then uboot reads the uEnv.txt file and executes the tftp commands to fetch and place the various boot images from tfpt server on to the DDR memory.
- 5) Then we will ask uboot to boot from the location where it placed the linux kernel image on the DDR memory



**Read the next article to prepare your HOST as a
TFTP server**



Serial boot : Big picture

Serial booting

Serial booting means transferring the boot images from **HOST** to the **board** via the serial port (UART) in order to boot the board .



What's the transfer protocol ?

Transfer file/information over IP network

TFTP
HTTP
FTP
SMTP
etc . . .

Transfer file/information over UART

Xmodem
Ymodem
Zmodem
Kermit
etc . . .

Serial booting

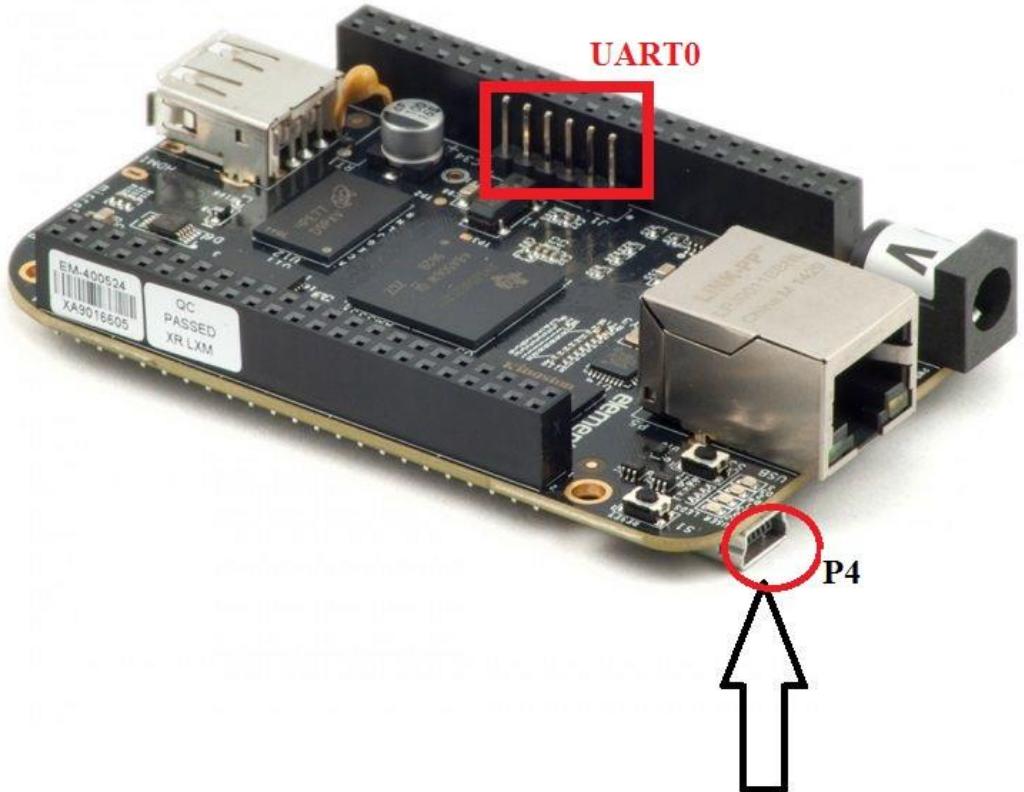
We will keep all the boot images like SPL, uboot, linux kernel image, DTD and the file system on the HOST PC.



HOW Serial booting work ??

First we have to make our board boots via UART Peripheral.

We have to boot exactly the same way how we used to boot via SD card, that is press and hold the S2 button, then press and release the s3 button, make sure that SD card is NOT inserted to the board.



HOW Serial booting work ??

When you keep the board in to UART boot mode, the ROM boot loader is waiting for the second stage boot loader that is SPL image over xmodem protocol only !

HOW Serial booting work ??

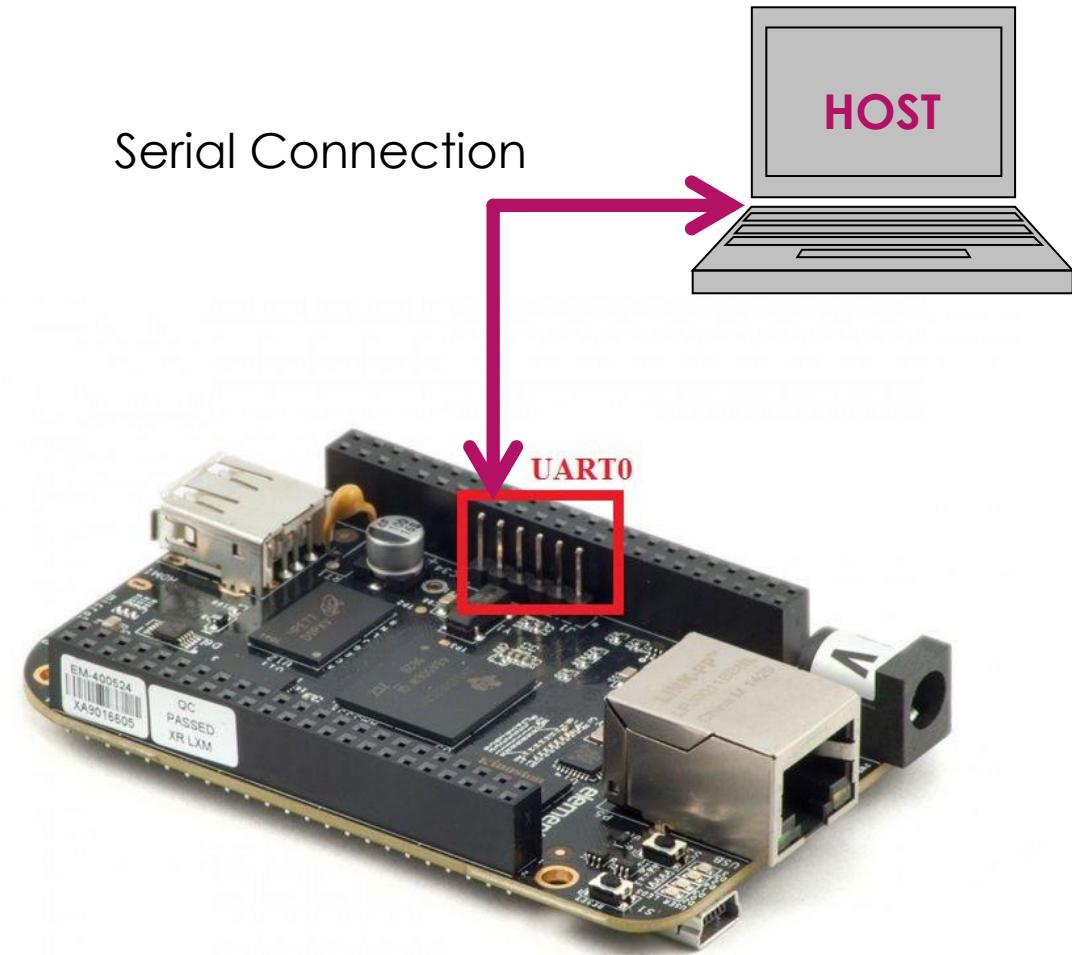
Once the SPL executes it also tries to get the third stage boot loader that is uboot image over xmodem protocol and you should send the uboot image over xmodem protocol from the host

HOW Serial booting work ??

When u-boot executes you can use u-boot commands such as xmodem or ymodem to load rest of the images like linux kernel image, DTB, initramfs in to the DDR memory of the board at recommend addresses

Summary

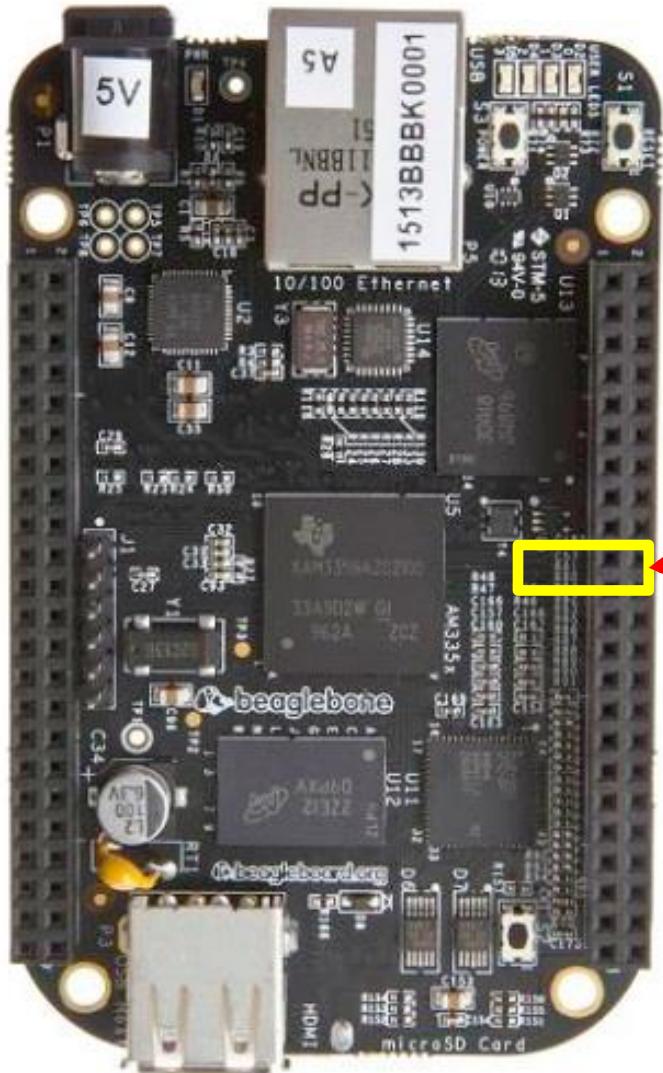
- ▶ First we should make our board slip in to the uart boot mode
- ▶ Once it goes to the uart boot mode, the ROM boot loader will be waiting to receive the SPL over UART via Xmodem protocol, so we will send SPL first from our host PC
- ▶ Then SPL executes on the board and it will be waiting to receive uboot image. So, we will send uboot also.
- ▶ When we get the Control of uboot on the board, we will use uboot's Xmodem and Ymodem protocol commands to download all the other boot images on to the DDR memory of the board and then we will boot the board.



Serial Connection

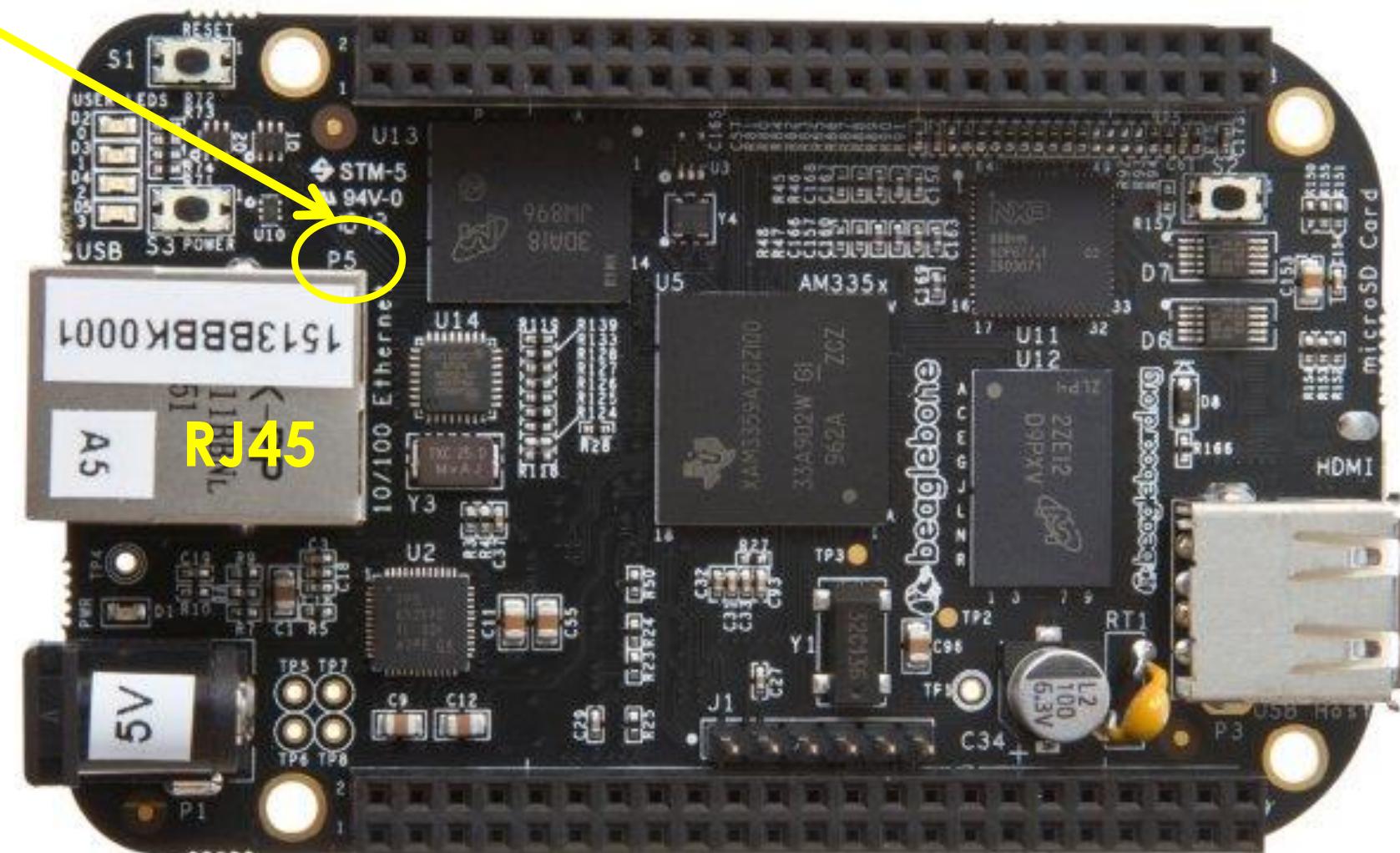
HOST

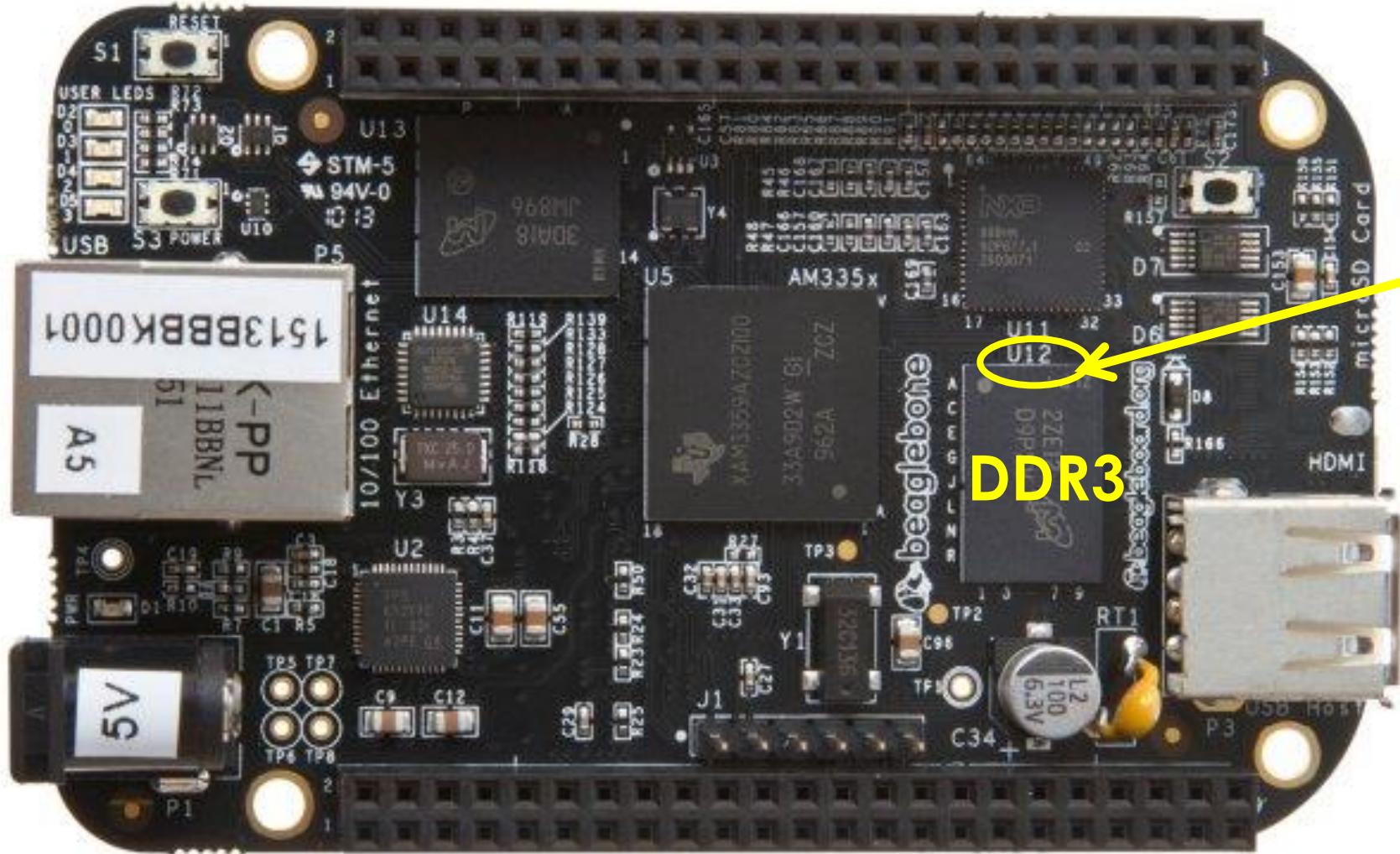
UART0



SOC
(AM3358BZCZ100)

Component number "P5"



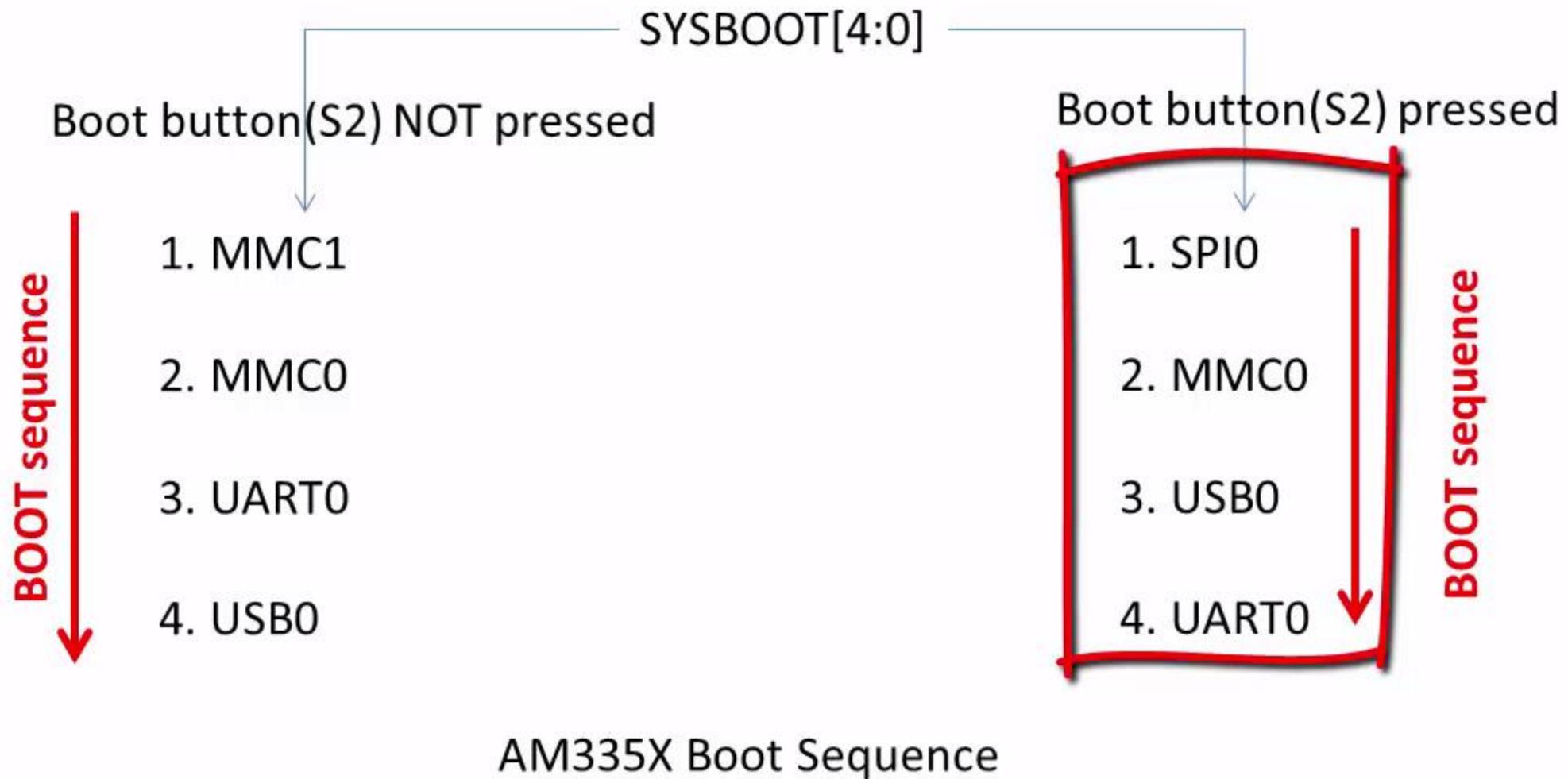


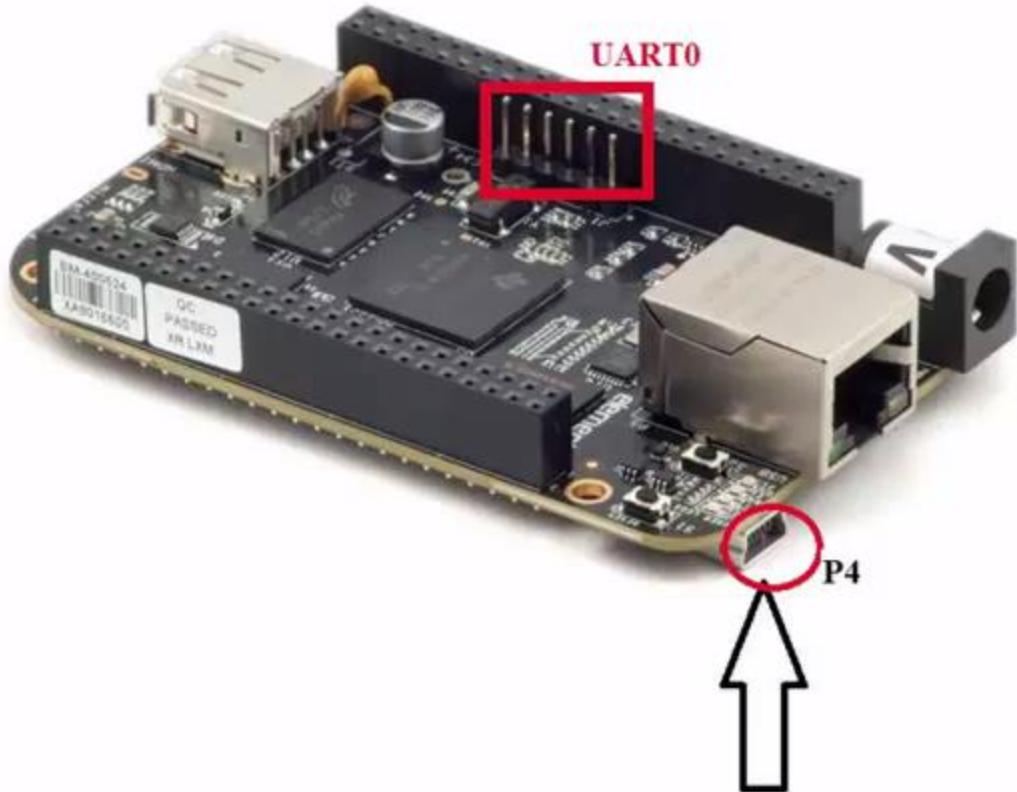
Component number "U12"

DDR3



Go through the links given in the Resource section to understand various peripherals driver guide !!!





SD card should NOT be connected

Mini USB Cable should NOT be connected at P4

Power up the board using power adapter

Single-Byte Write Sequence

Master	S	AD+W		RA		DATA		P
Slave			ACK		ACK		ACK	

Burst Write Sequence

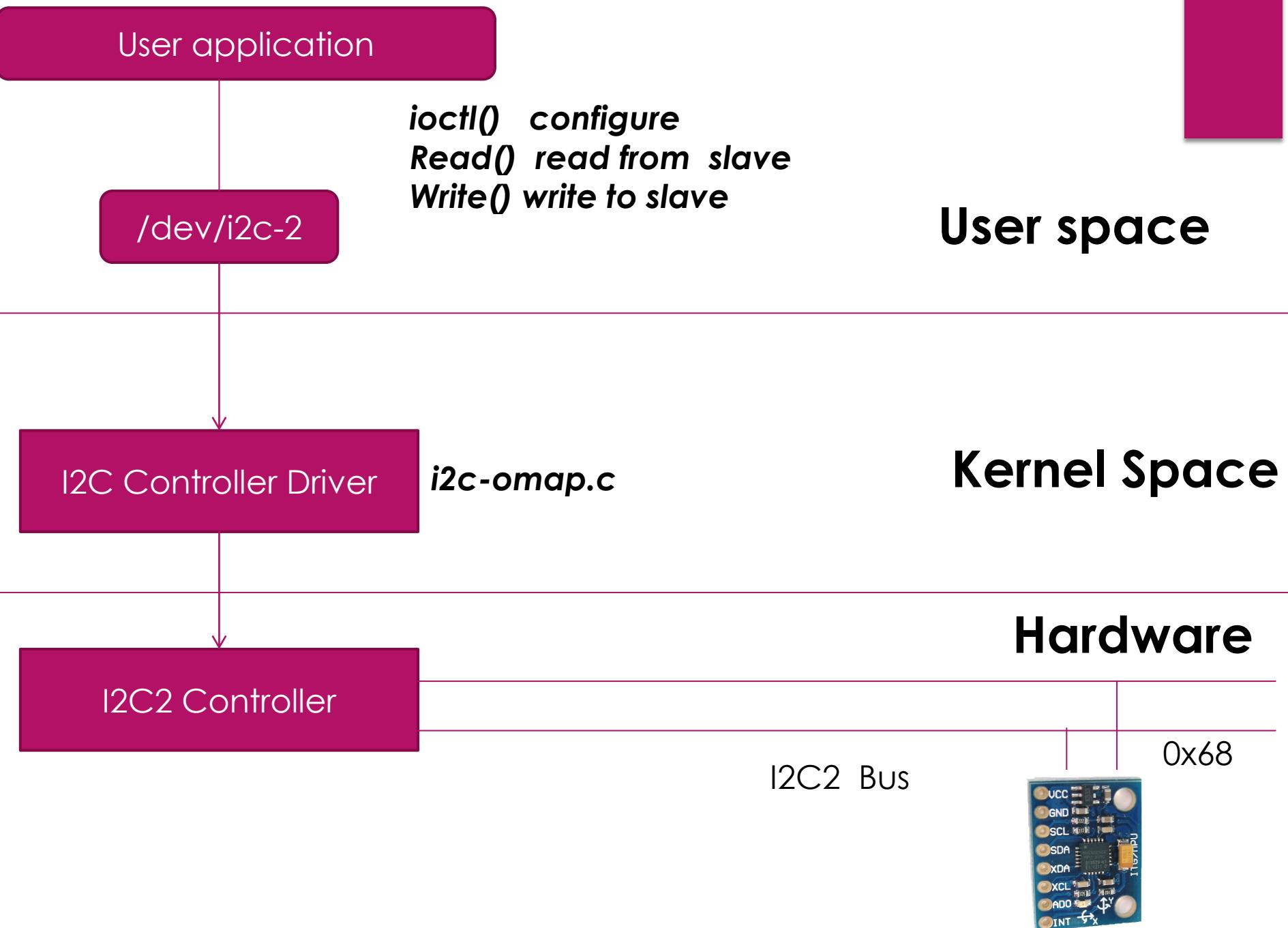
Master	S	AD+W		RA		DATA		DATA		P
Slave			ACK		ACK		ACK		ACK	

Single-Byte Read Sequence

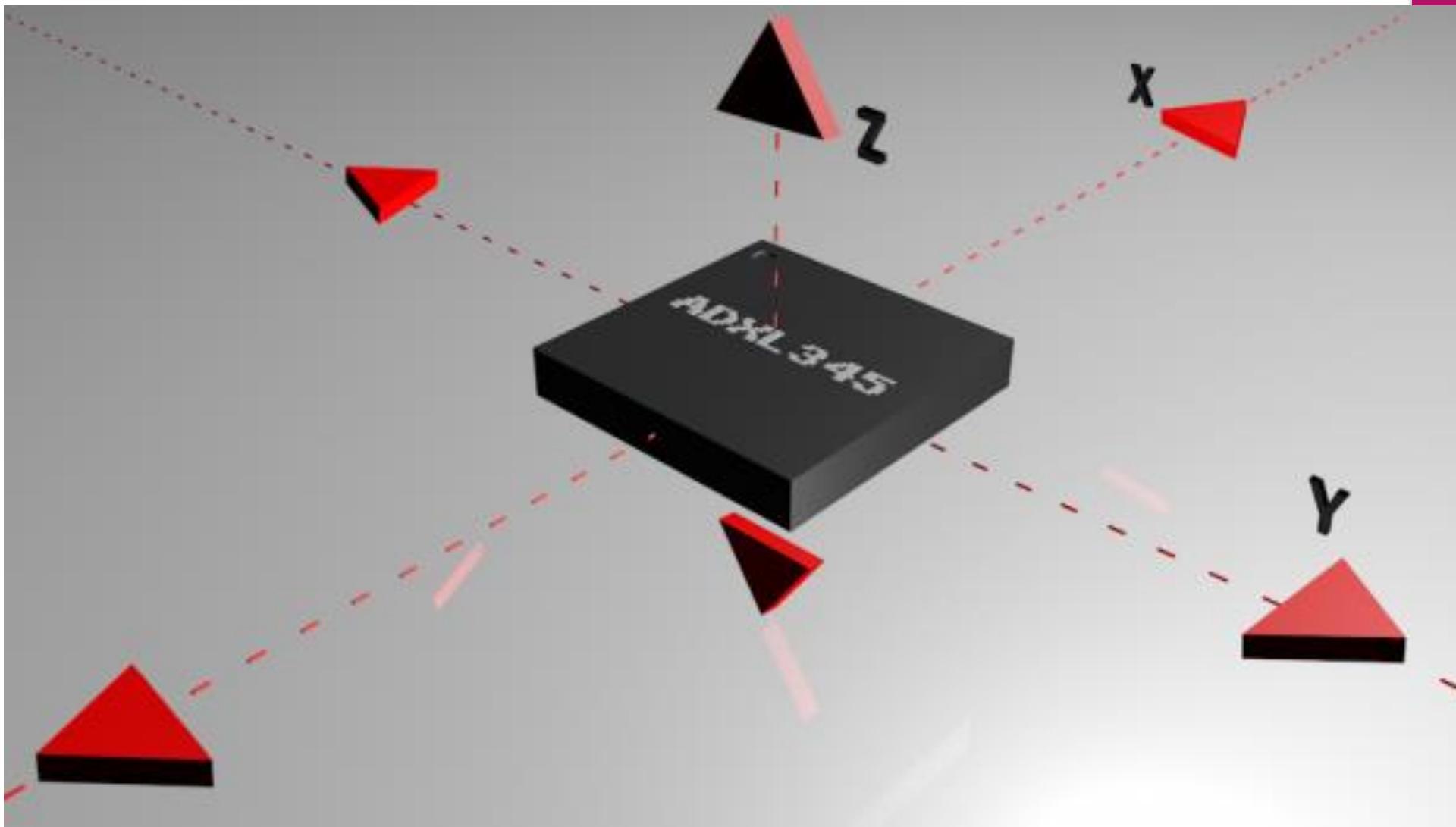
Master	S	AD+W		RA		S	AD+R			NACK	P
Slave			ACK		ACK			ACK	DATA		

Burst Read Sequence

Master	S	AD+W		RA		S	AD+R			ACK		NACK	P
Slave			ACK		ACK			ACK	DATA		DATA		



MPU6050 Basics



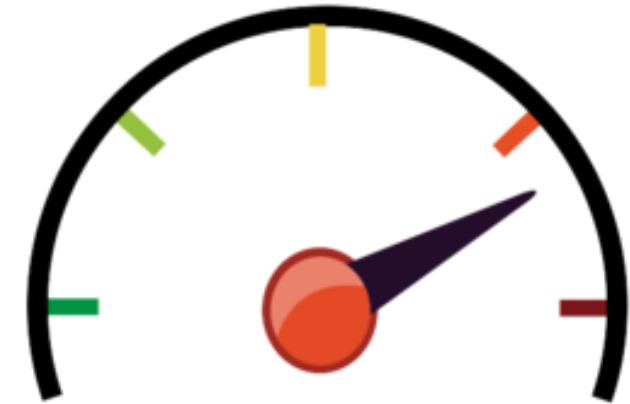
Accelerometer Full-Scale Range

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS	NOTES
ACCELEROMETER SENSITIVITY						
Full-Scale Range	AFS_SEL=0 AFS_SEL=1 AFS_SEL=2 AFS_SEL=3		± 2 ± 4 ± 8 ± 16		g g g g	
ADC Word Length	Output in two's complement format		16		bits	
Sensitivity Scale Factor	AFS_SEL=0 AFS_SEL=1 AFS_SEL=2 AFS_SEL=3		16,384 8,192 4,096 2,048		LSB/g LSB/g LSB/g LSB/g	

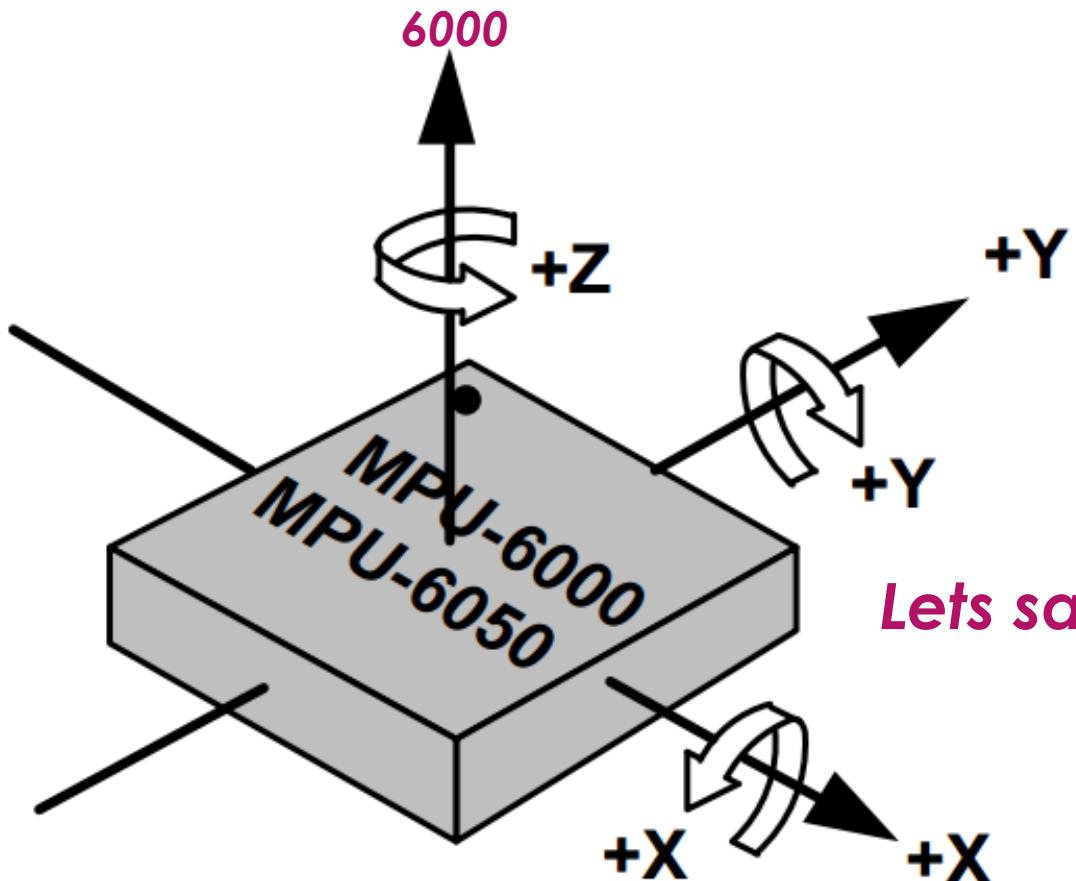


Gyroscope Full-Scale Range

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
GYROSCOPE SENSITIVITY					
Full-Scale Range	FS_SEL=0		±250		°/s
	FS_SEL=1		±500		°/s
	FS_SEL=2		±1000		°/s
	FS_SEL=3		±2000		°/s
Gyroscope ADC Word Length			16		bits
Sensitivity Scale Factor	FS_SEL=0		131		LSB/(°/s)
	FS_SEL=1		65.5		LSB/(°/s)
	FS_SEL=2		32.8		LSB/(°/s)
	FS_SEL=3		16.4		LSB/(°/s)

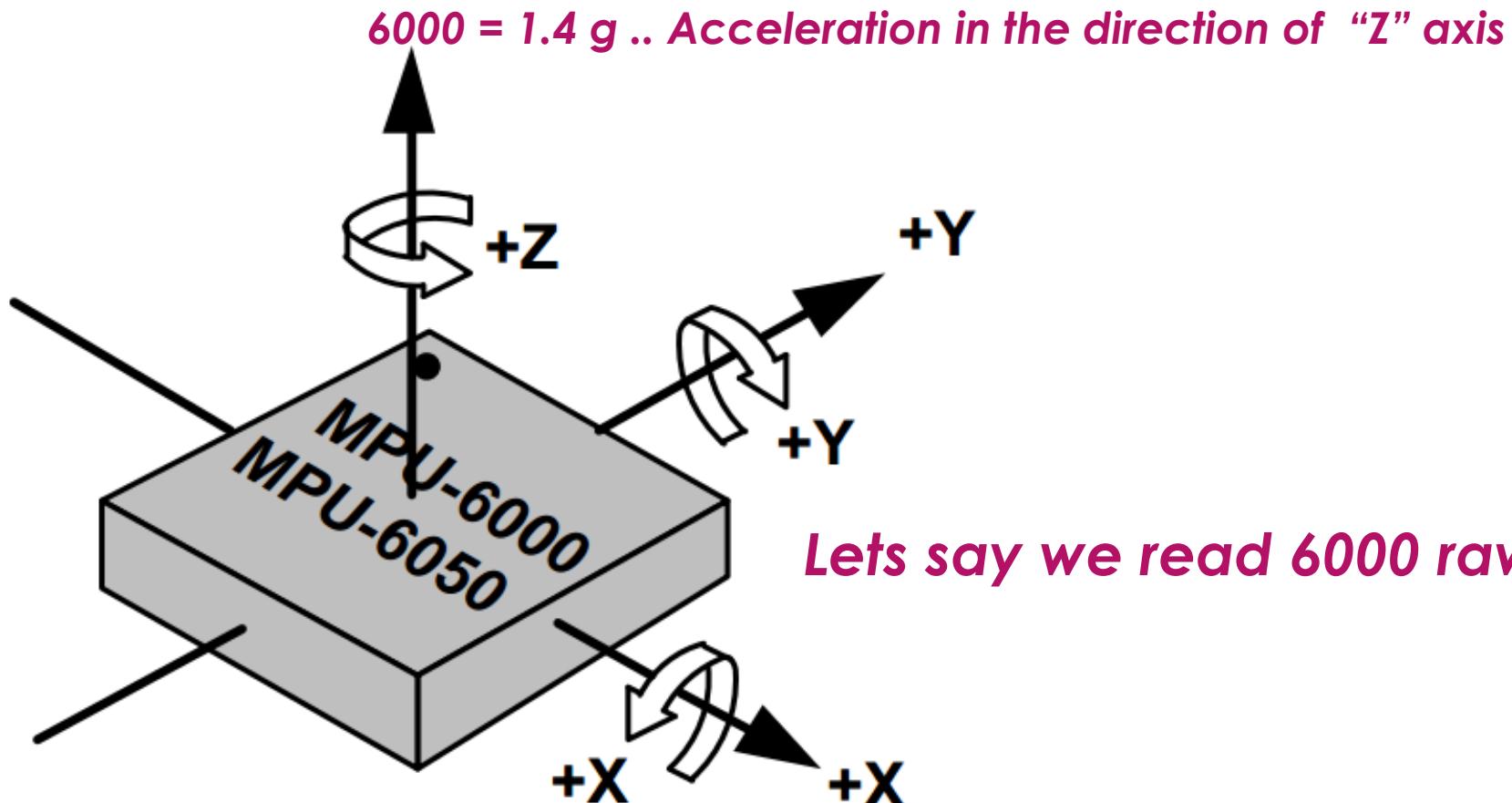


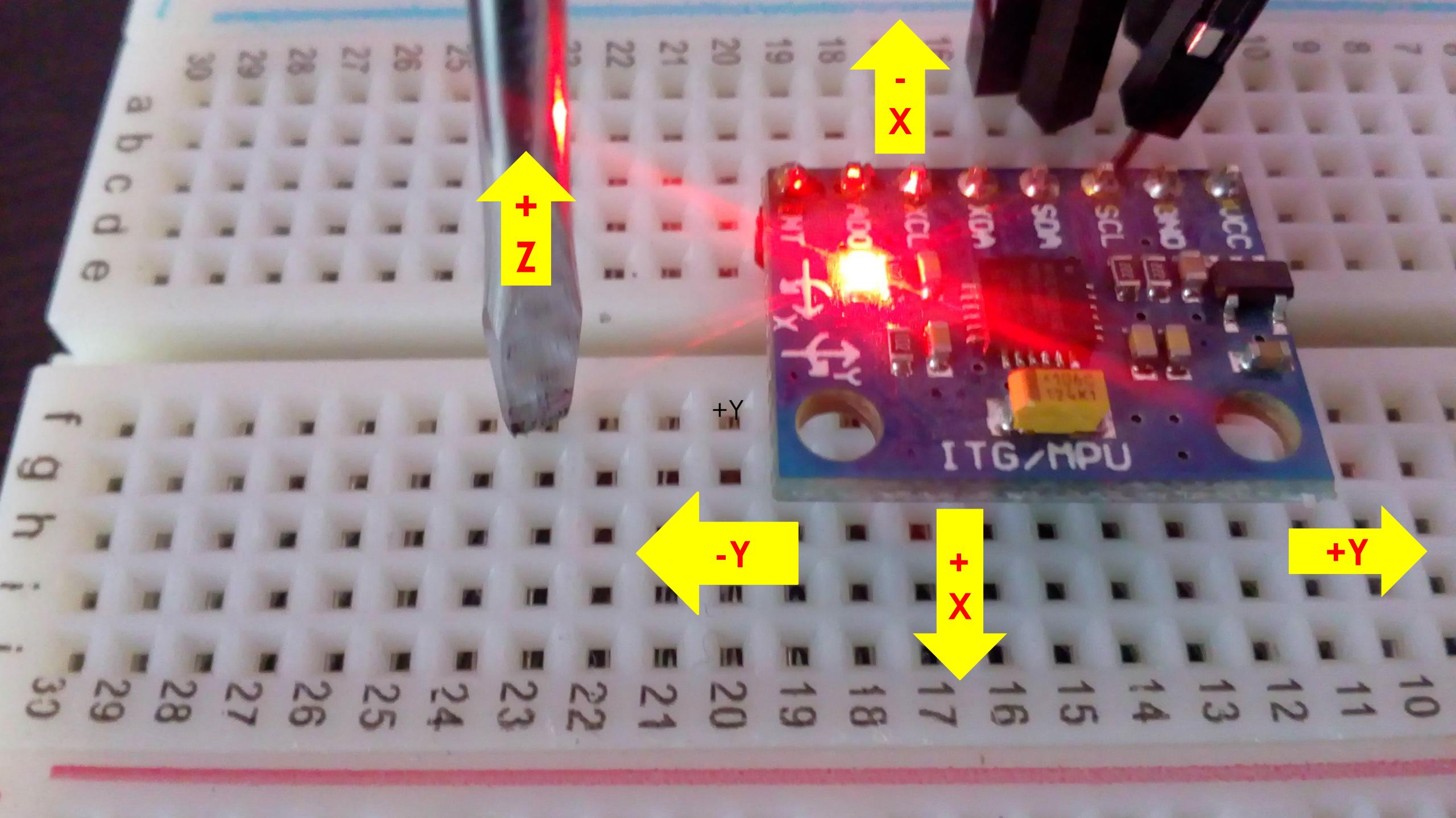
Converting “raw” value to “g” value



Lets say we read 6000 raw value on the Z AXIS

Converting “raw” value to “g” value





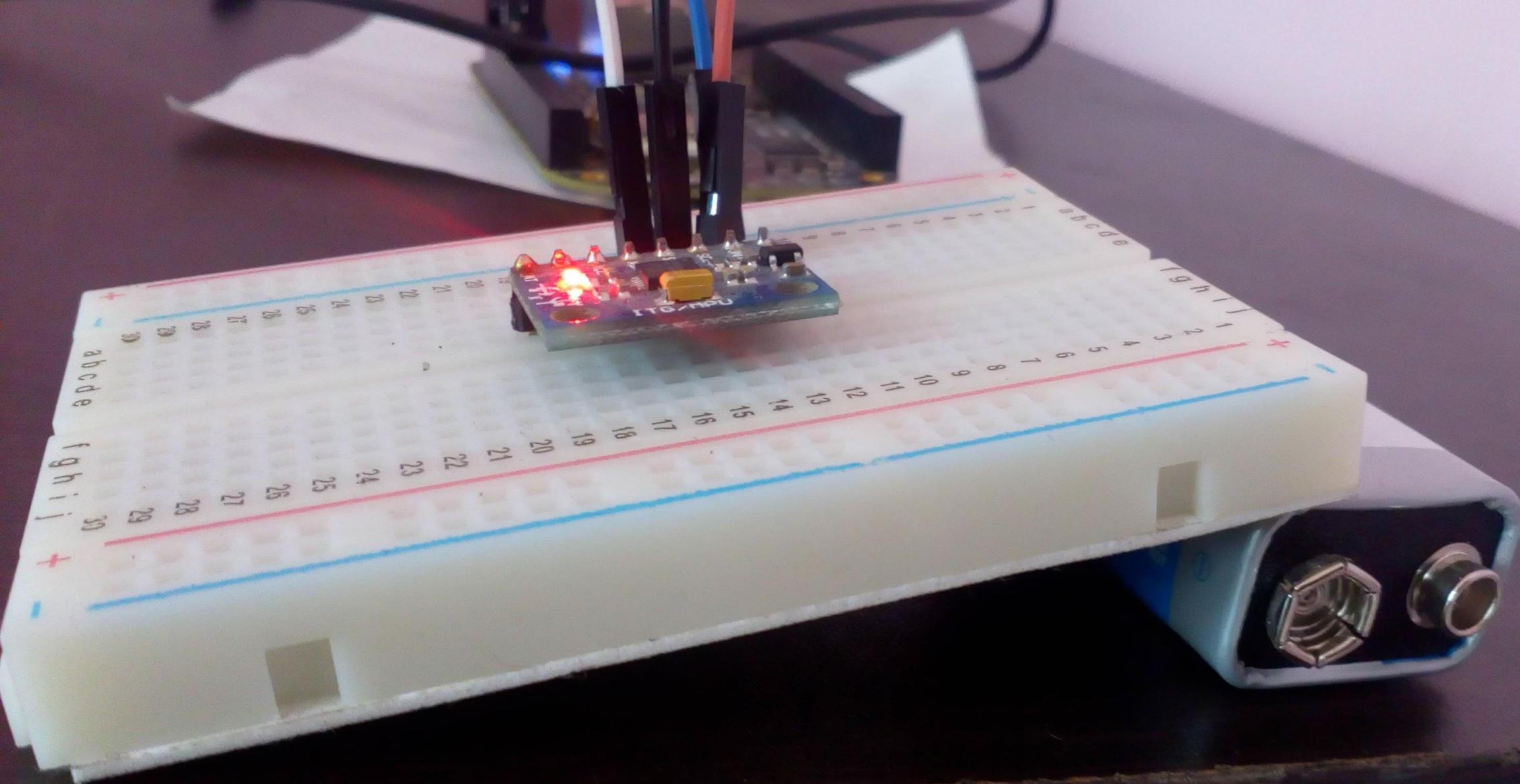
Tilt angle calculation on x and y axis

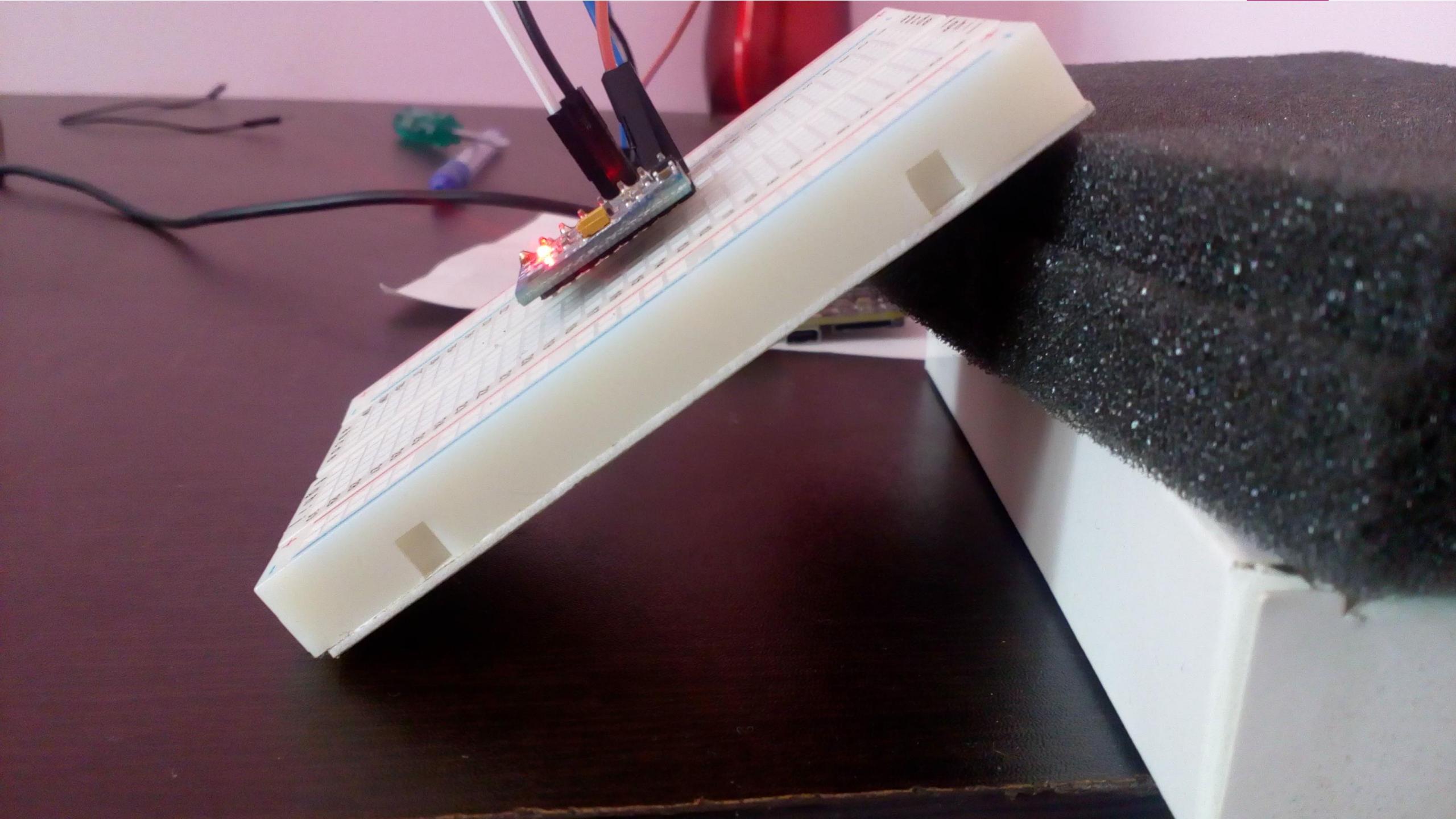
Tilt angle over “X” Axis :

$$\text{angle (degree)} = \sin^{-1} \frac{x}{\sqrt{x^2 + z^2}}$$

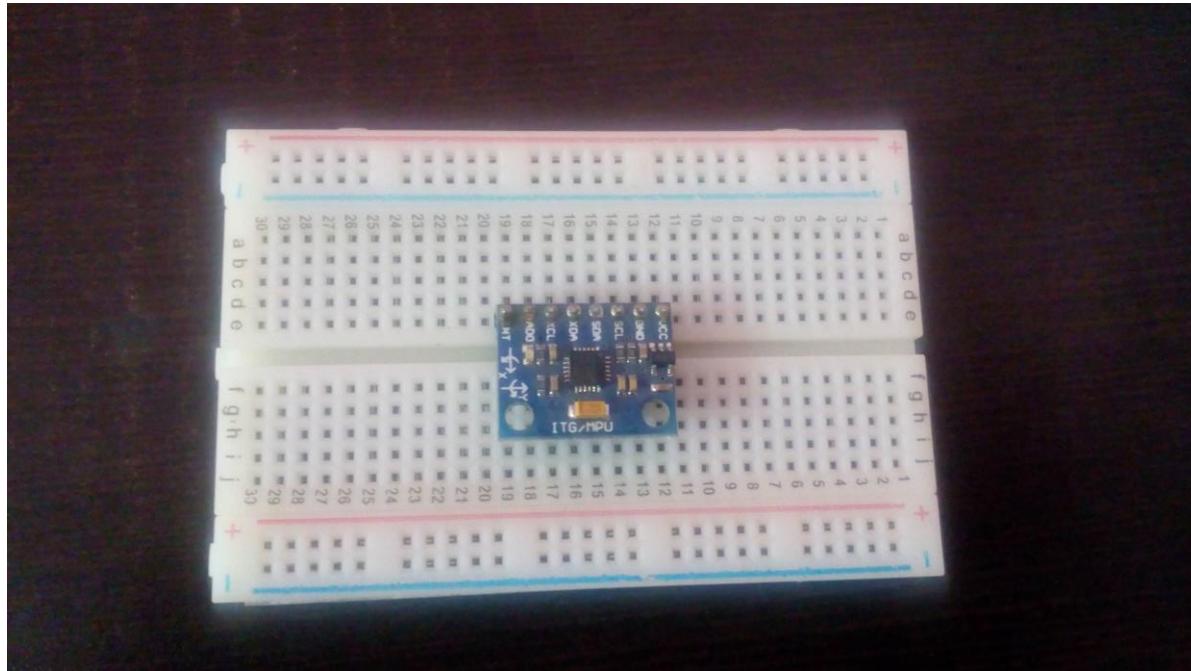
Tilt angle over “Y” Axis :

$$\text{angle (degree)} = \sin^{-1} \frac{y}{\sqrt{y^2 + z^2}}$$

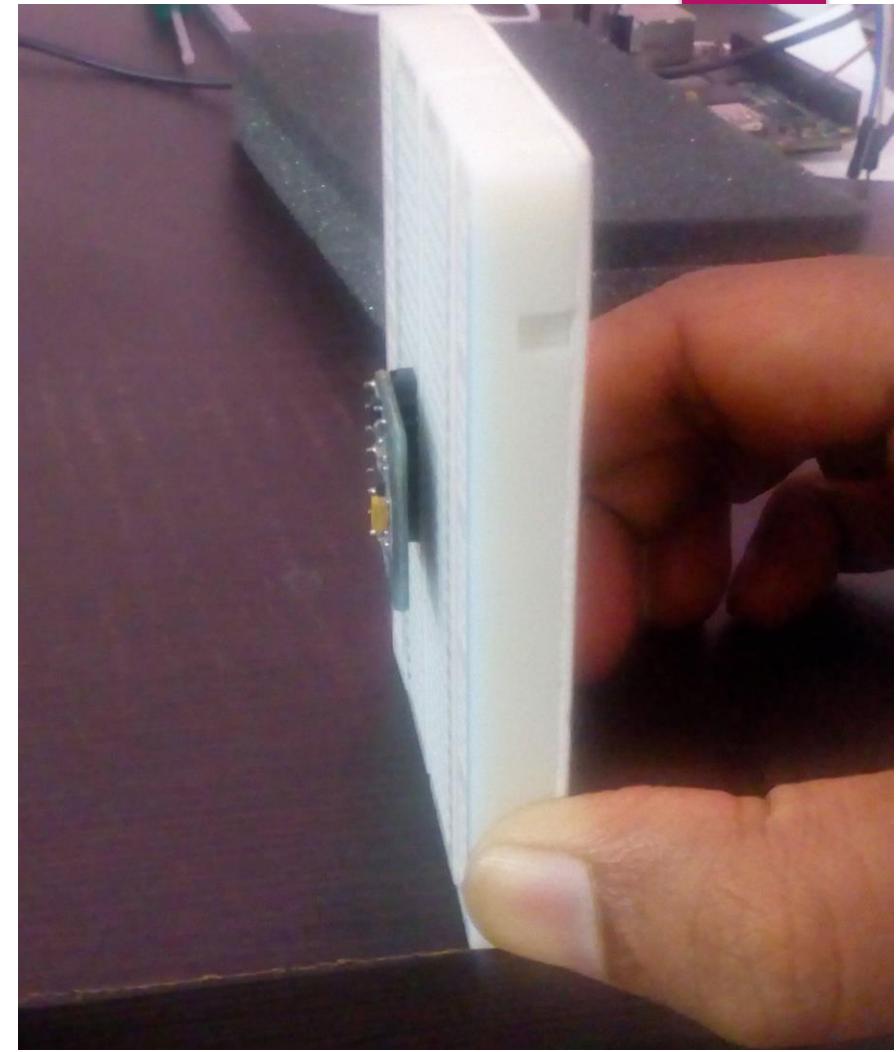




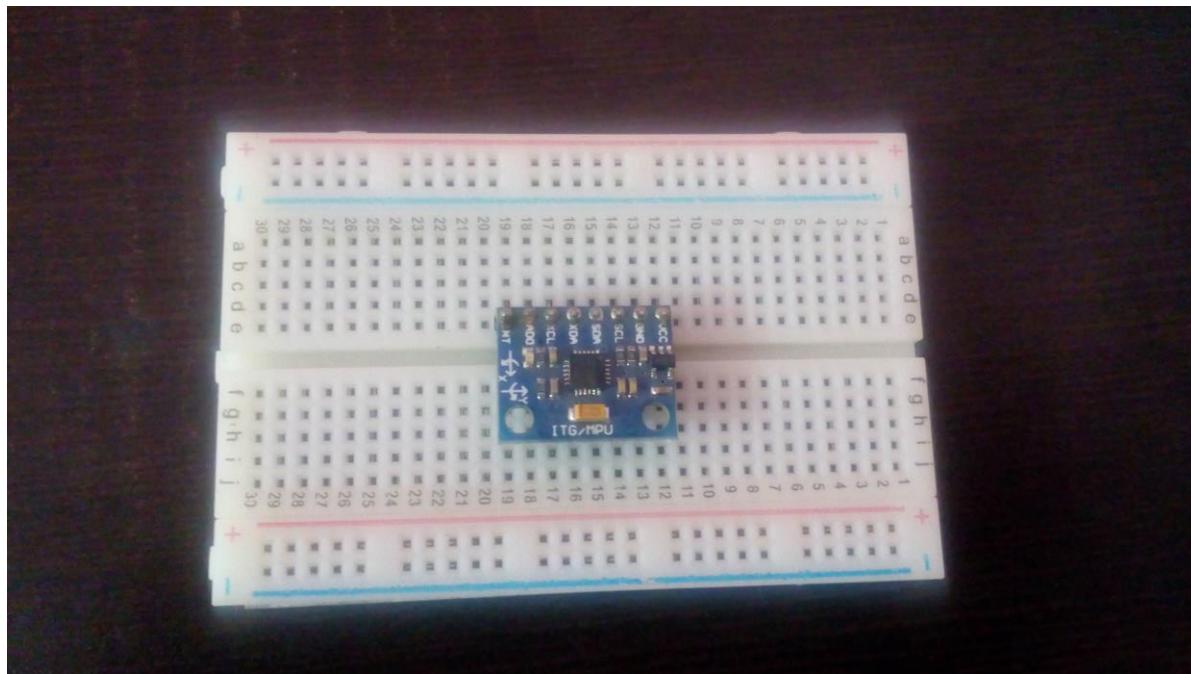




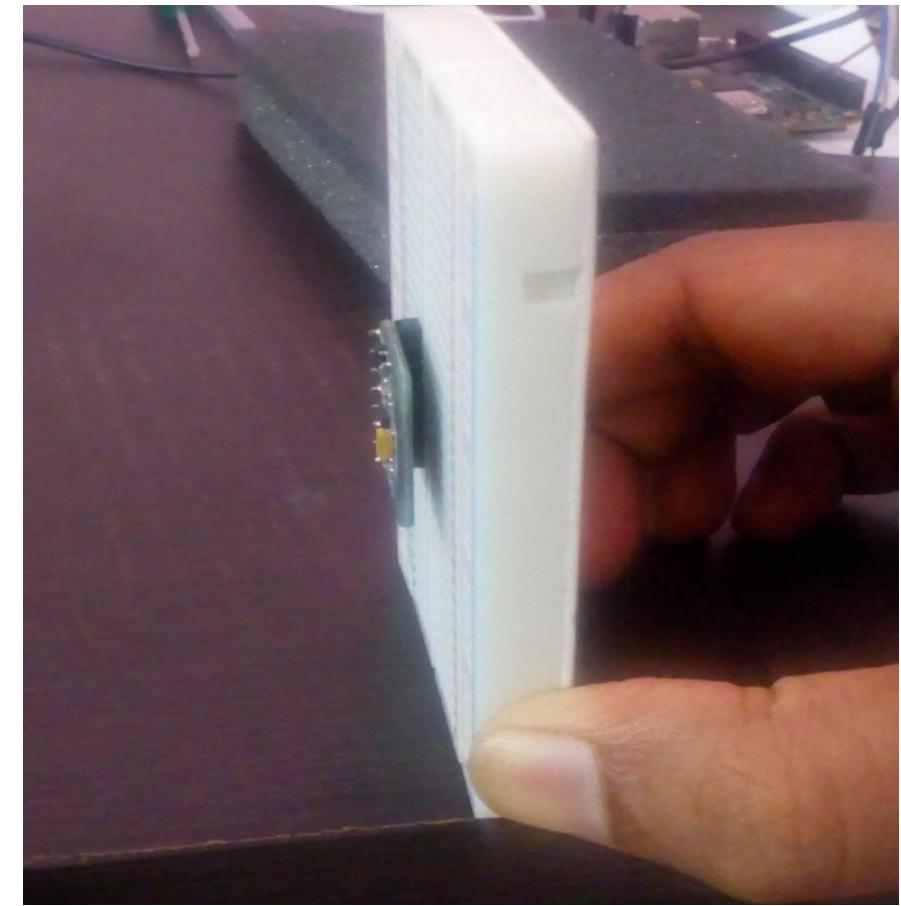
Before



After 90 degree tilt over "y"



Before



After 90 degree tilt over "y"



Why Accelerometers are used ?

What are accelerometers ?

“Accelerometer is a sensor which can measure the Acceleration forces which are acted on it ! “

Why Accelerometers are used ?

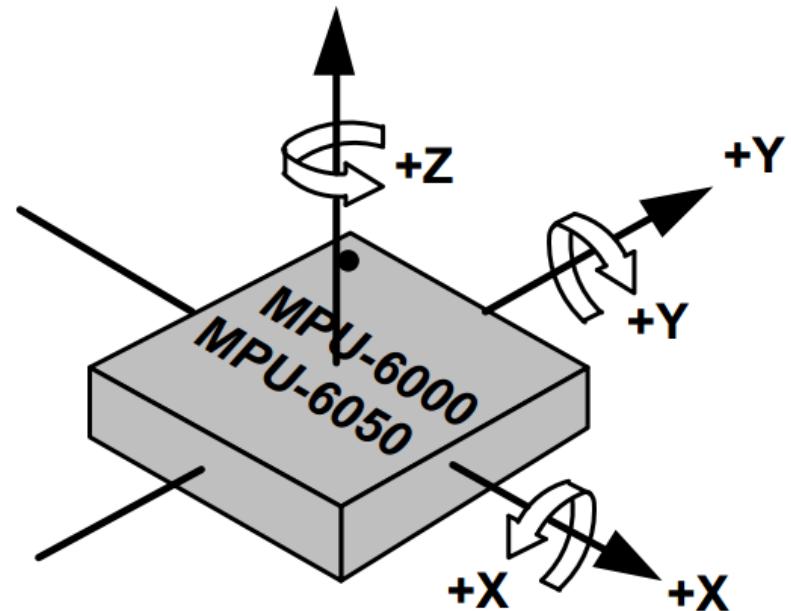
Force \propto acceleration

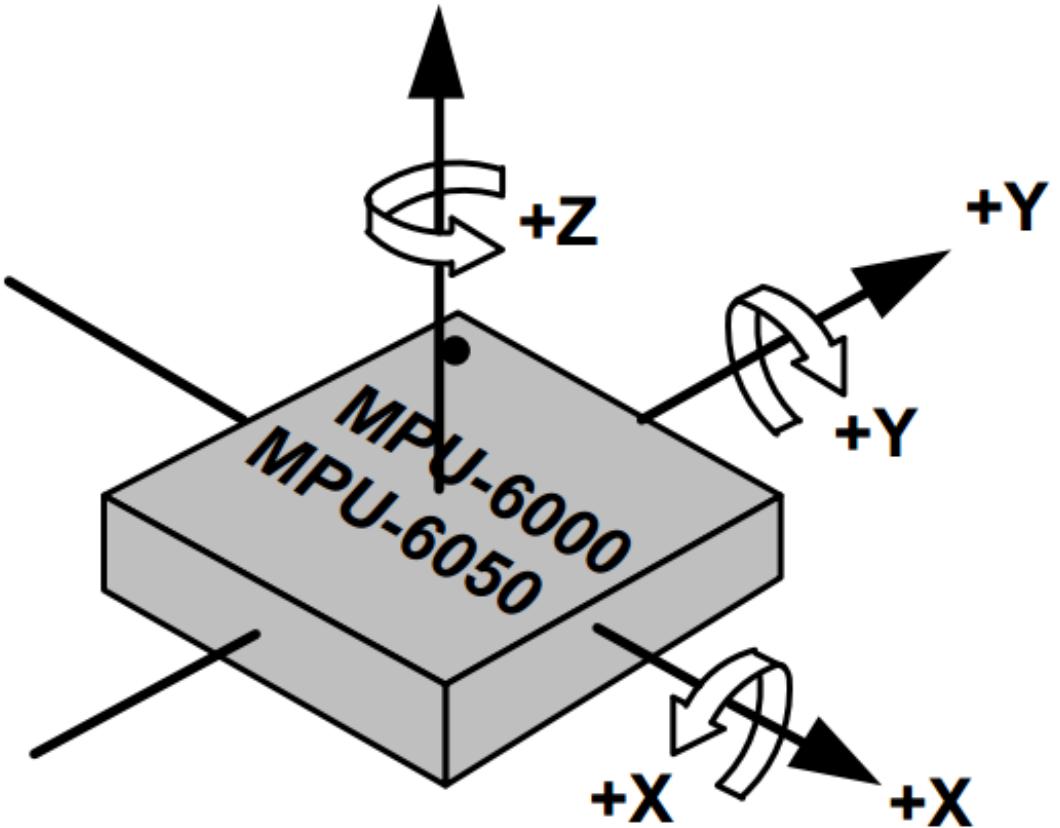
Why Accelerometers are used ?

Force \propto acceleration

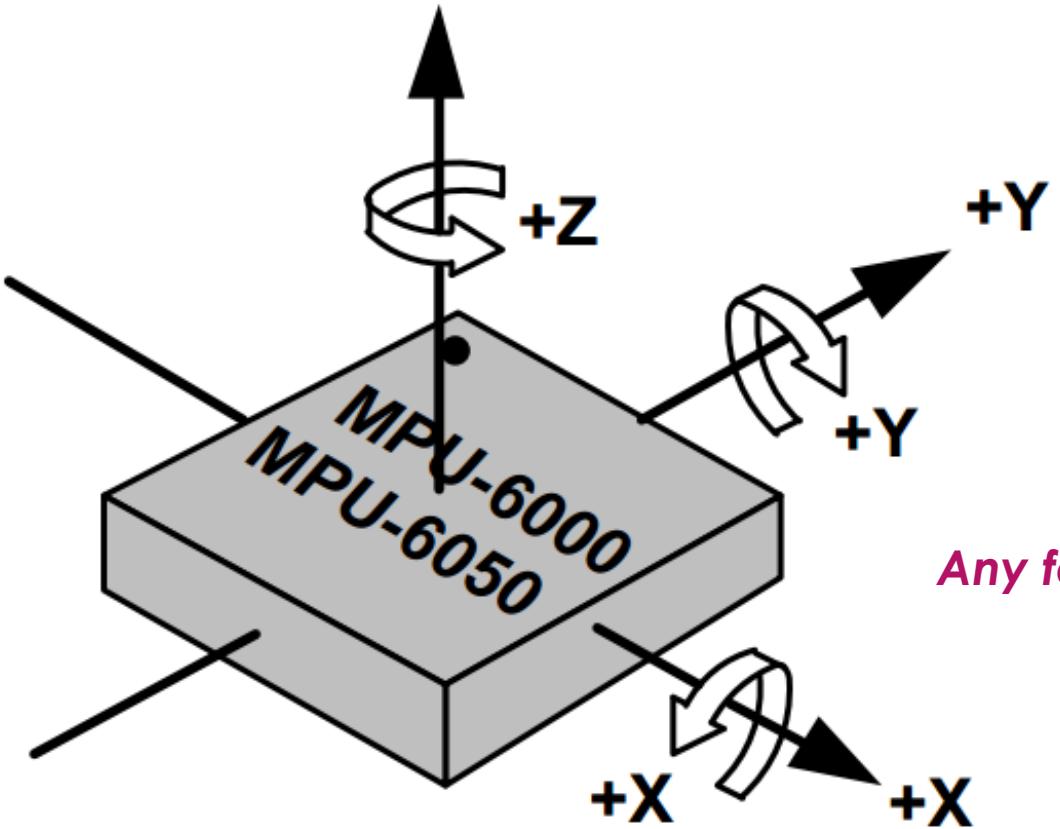
Why Accelerometers are used ?

Force ~~is~~ acceleration



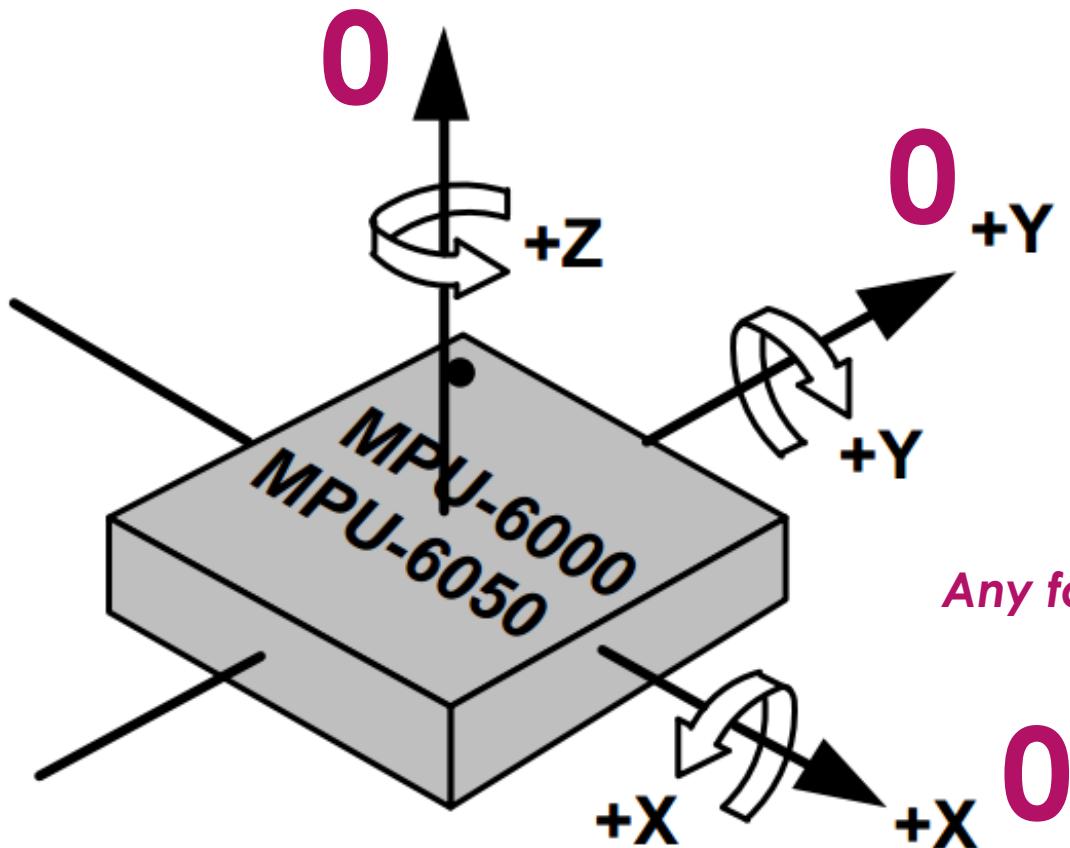


Assume the sensor is at rest on the table



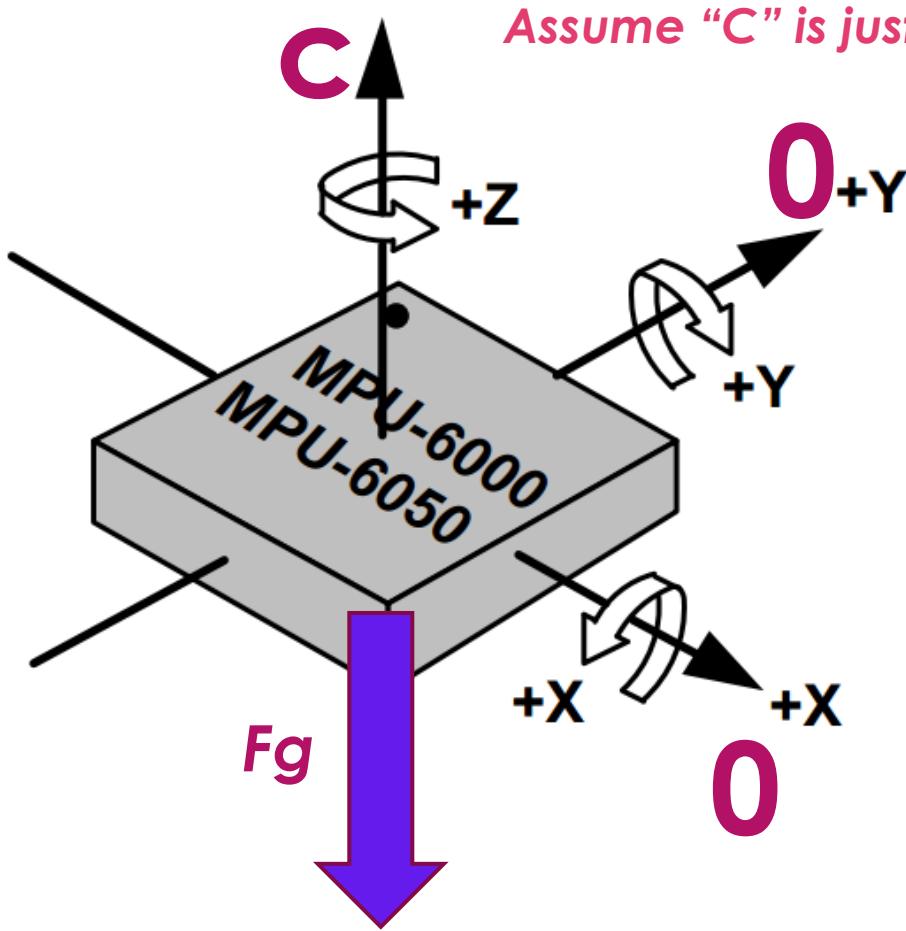
Any force is acting on this sensor

Assume the sensor is at rest on the table



Any force is acting on this sensor

*If no force is acting on this sensor then you should get all zeros across all the axes.
Remember acceleration and force are directly proportional and job of the accelerometer
Sensor is to produce readings for the amount of the force applied.*

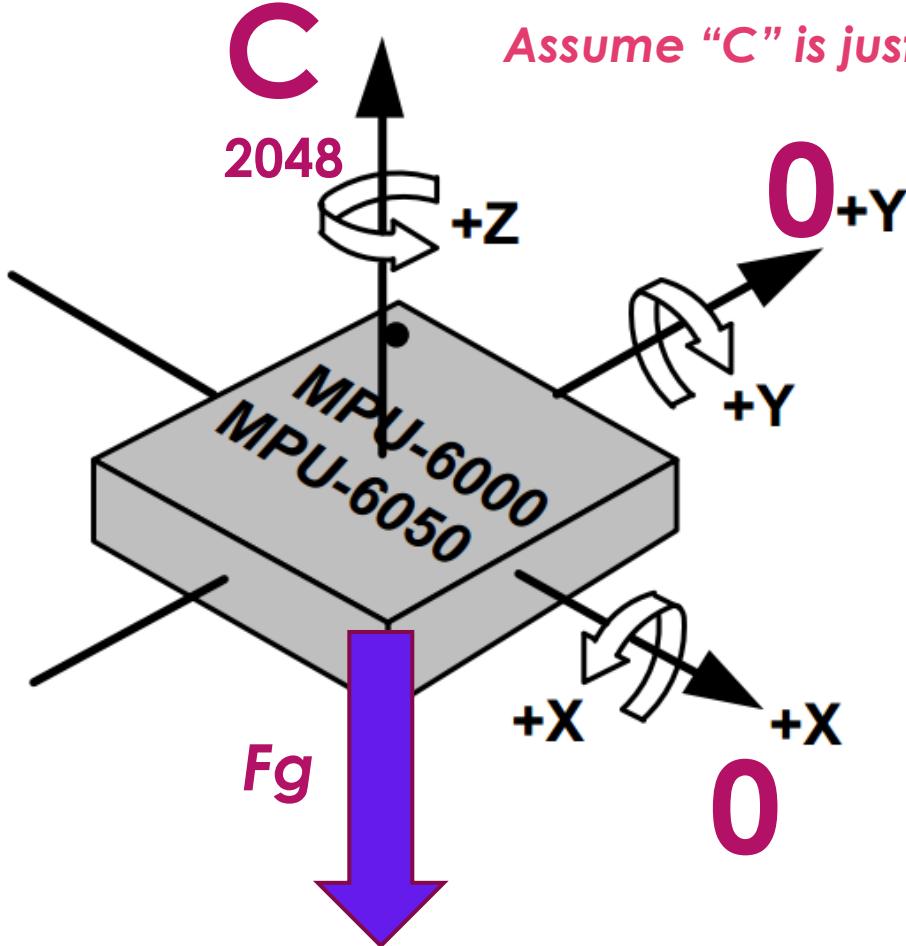


Even when this sensor is at rest it outputs some value which is equivalent to static force like earths gravity

Force due Gravity

Hence, acceleration = $1g$ (9.8 m/s^2)

So, we can say that "C" value is equivalent to $1g$ of acceleration caused by earths gravity



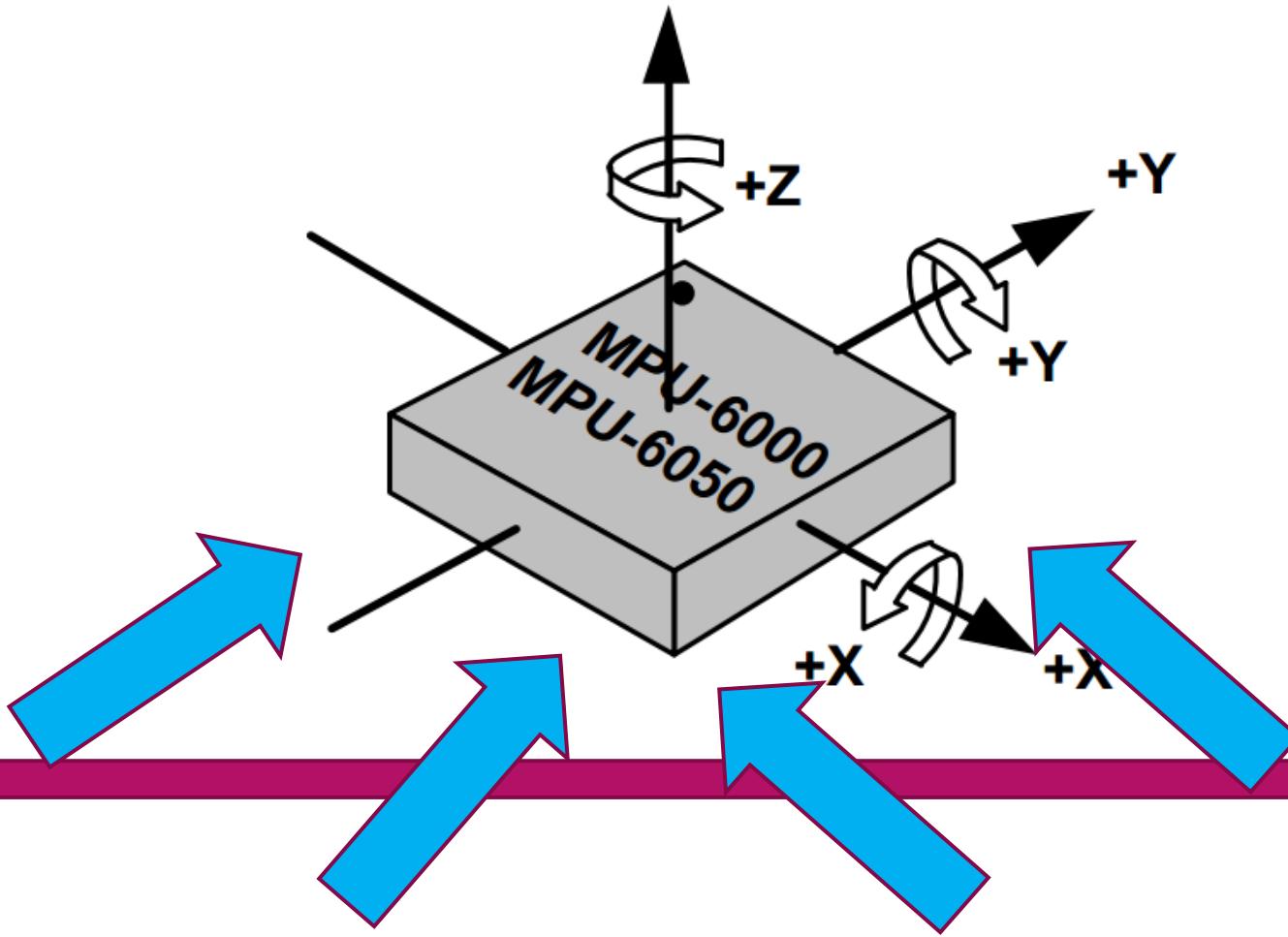
Assume "C" is just some number

Even when this sensor is at rest it outputs some value which is equivalent to static force like earths gravity

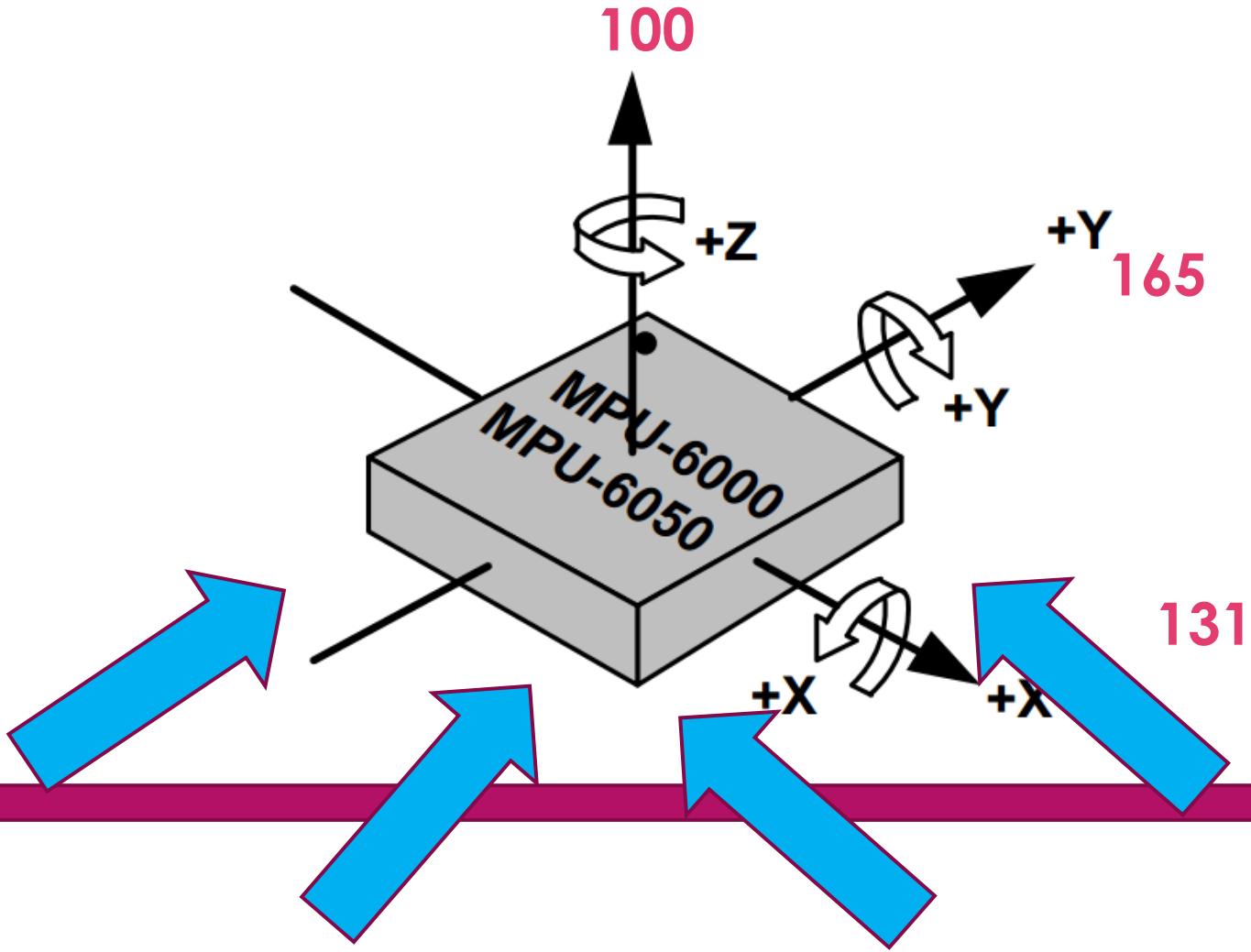
Force due Gravity

Hence, acceleration = 1g (9.8 m/s²)

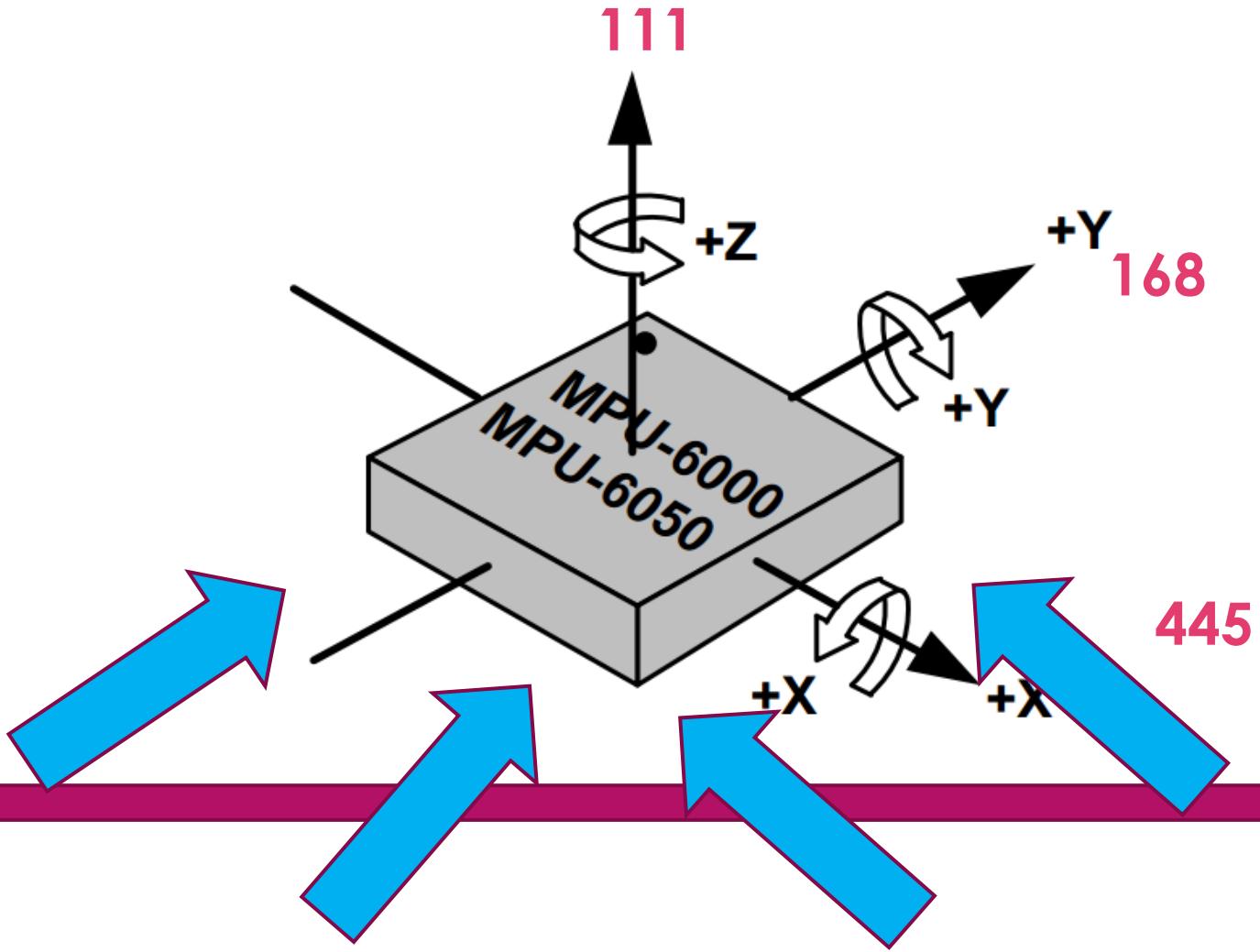
So, we can say that "C" value is equivalent to 1g of acceleration caused by earths gravity



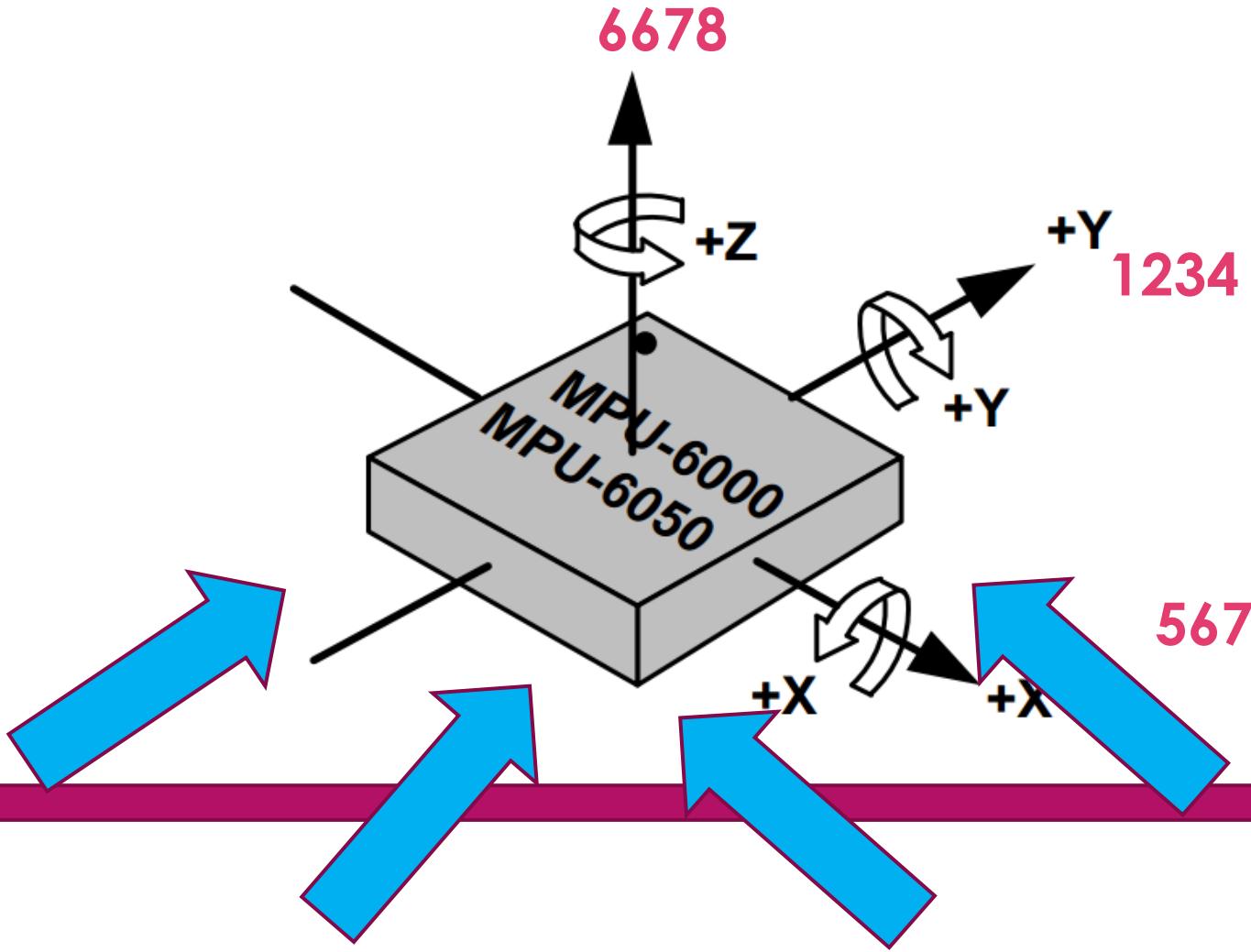
Forces induced on the sensor due to vibrations



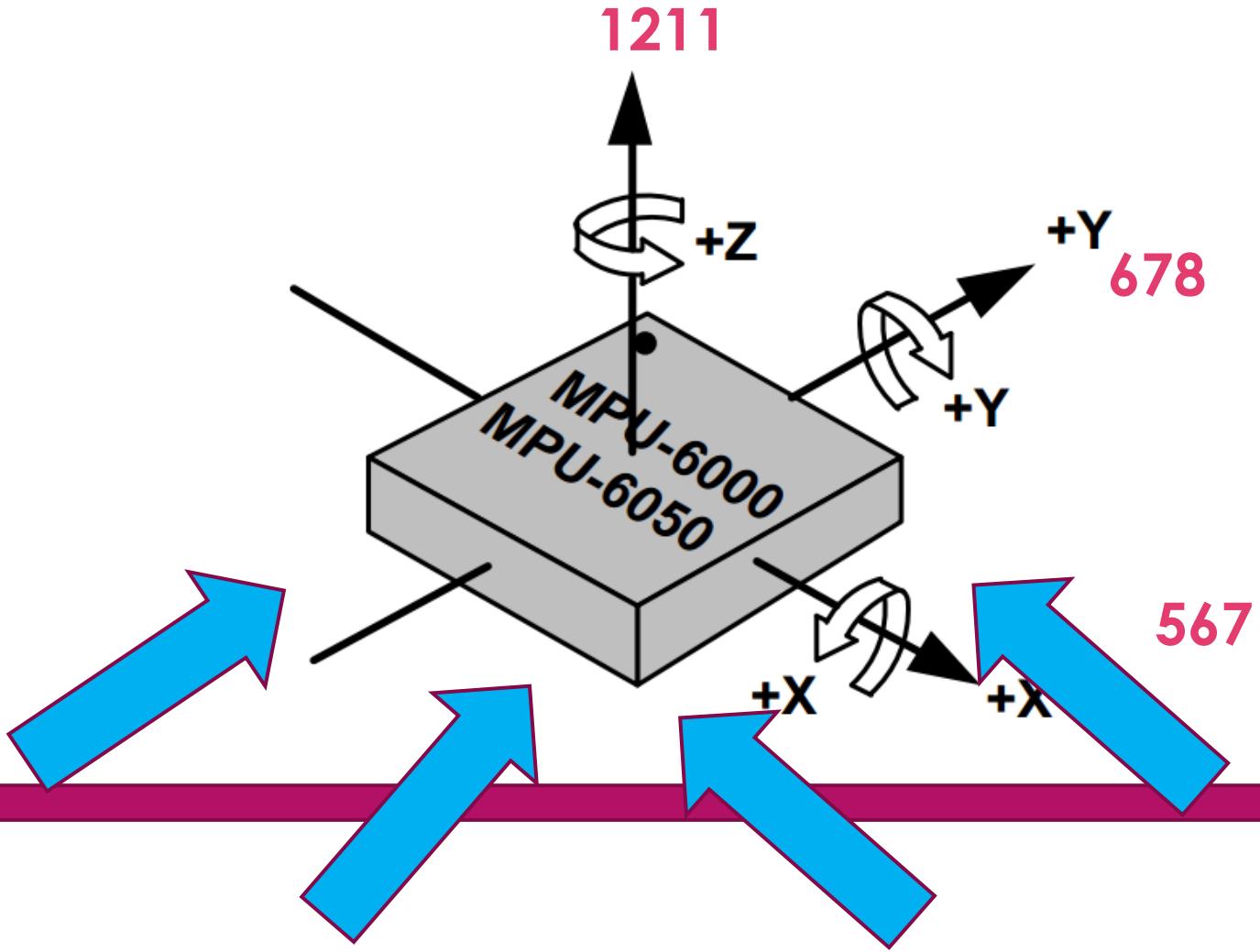
Forces induced on the sensor due to vibrations



Forces induced on the sensor due to vibrations



Forces induced on the sensor due to vibrations



Forces induced on the sensor due to vibrations

Why Accelerometers are used ?

Force  acceleration

So, the sensor produces the output which is directly proportional to the exerted forces on it from various axes !

Measuring Acceleration



X-direction



X-direction



1s



2s



3s

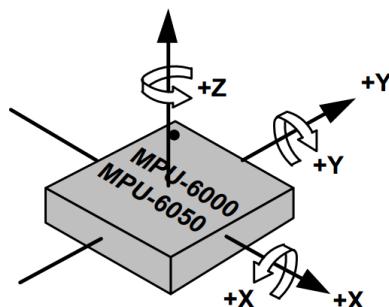


5s



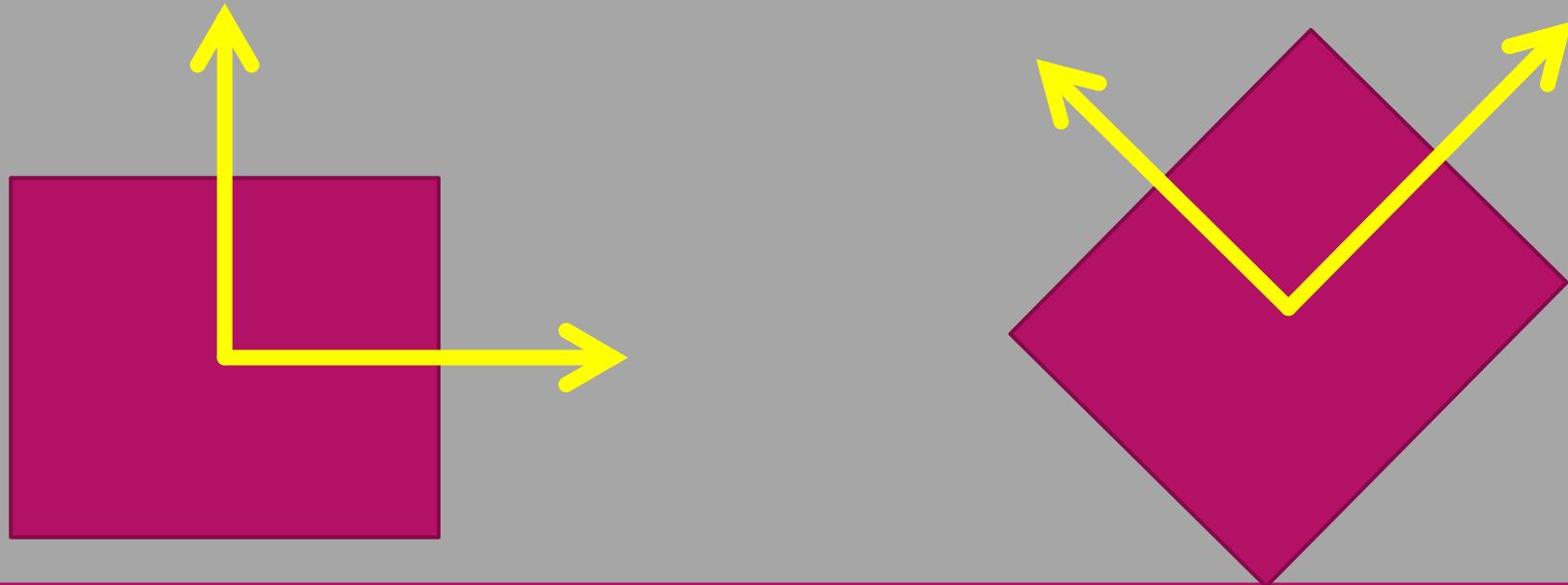
10s

X-direction



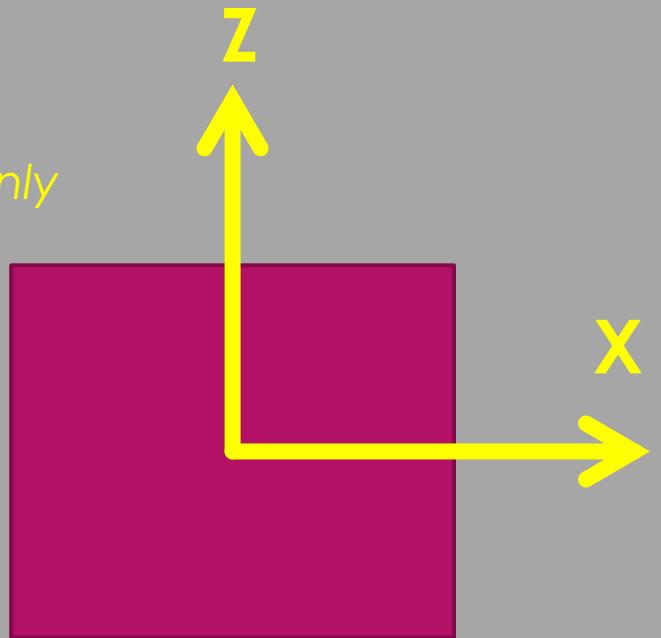
X: 4500
Y: 6789
Z: 1300

Convert to "g" values
1g means 9.8m/s² of acceleration

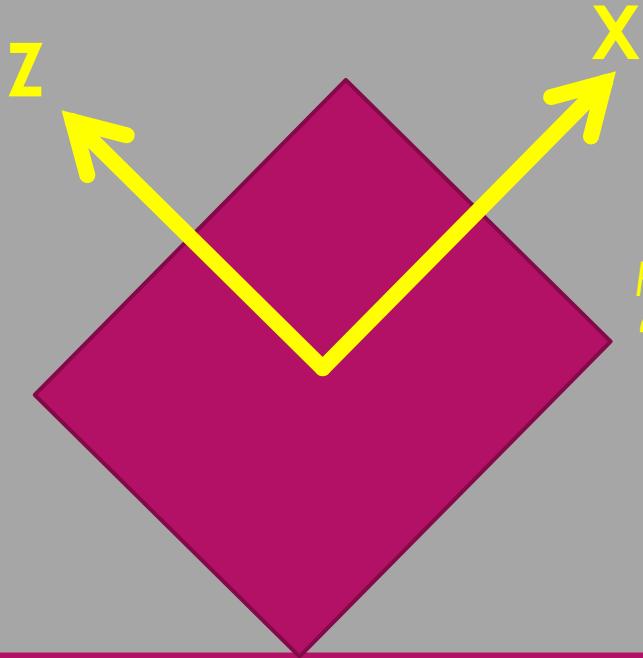


Accelerometers in “Tilt” sensing applications

F_g is acting only
on "Z" axis



F_g acting on both
"x" and "z" axis



X is parallel to the surface
Z is perpendicular to the surface
Y is also parallel (not shown)
So, readings will be
X: 0
Y: 0
Z: 2048

X is no longer parallel
Z is no longer perpendicular
Y remains parallel (not shown)
So, readings will be
X: 1024
Y: 0
Z: 1024

Summary

Accelerometers measure the vibrational forces which are acted upon them from various directions

The micro crystals inside the sensors will produce raw readings which are directly proportional to the forces exerted on the sensor.

The datasheet always tell us what is the reading at 1g of acceleration .

Based on this information we can convert any raw readings from the sensor to its equivalent “g” values.

Accelerometers are also used in tilt sensing application and to know the orientation of the body by taking earths gravity as a reference which you will see practically later in these lecture series.

Accelerometer vs Gyroscope

Accelerometer:

Accelerometer measures the Acceleration forces exerted on the axes x,y,z

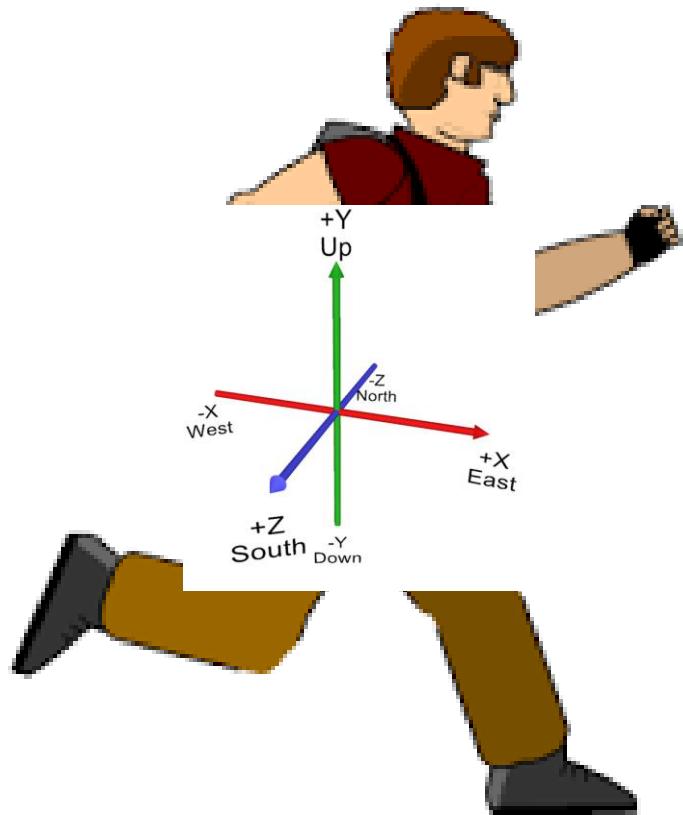
Gyroscope:

Gyro measures the rotational movement of an object over the axes x,y,z



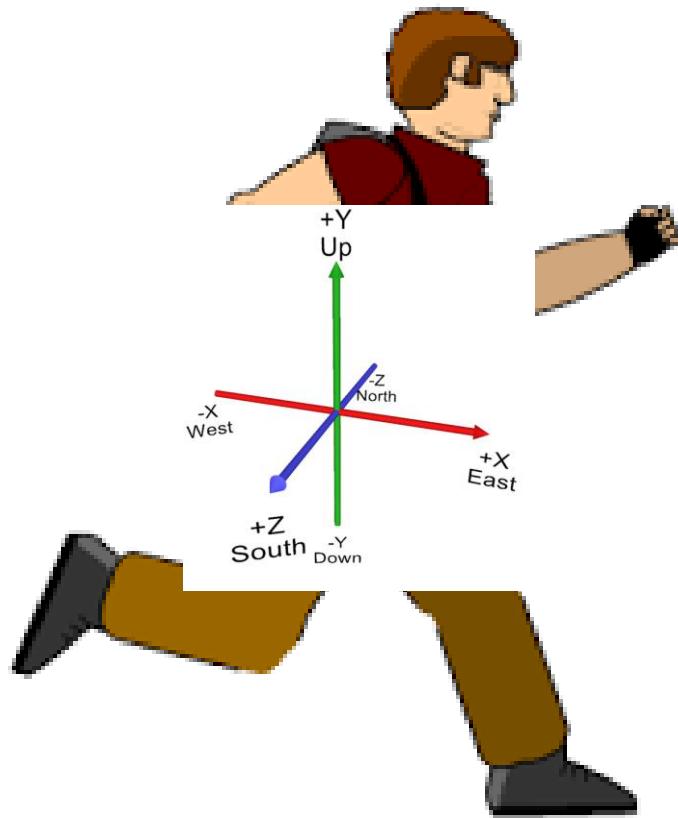
X-direction

In this case, If you are exerting force on the x axis , the “accelerometer” should show equivalent “g” values but “gyroscope” should be ideally unaffected.

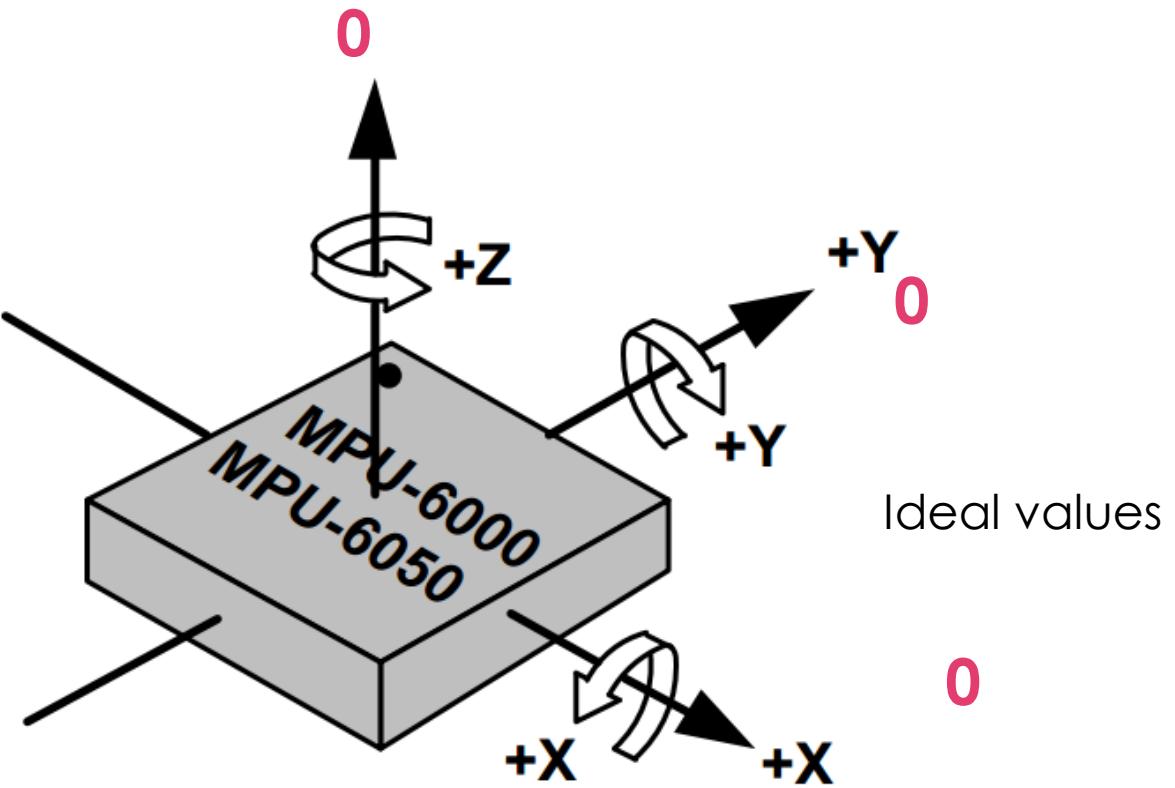


“gyro” measures the rotational movement on a given axis .. In this case rotation is over the “Z” axis.

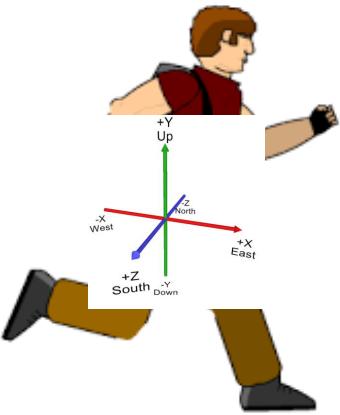
In this case “Accelerometer” also measures the “g” values variation on different axes.



In this case “gyro” will emit some raw value on the “Z” axis only . And you have to convert that “raw” value in to “deg/sec”

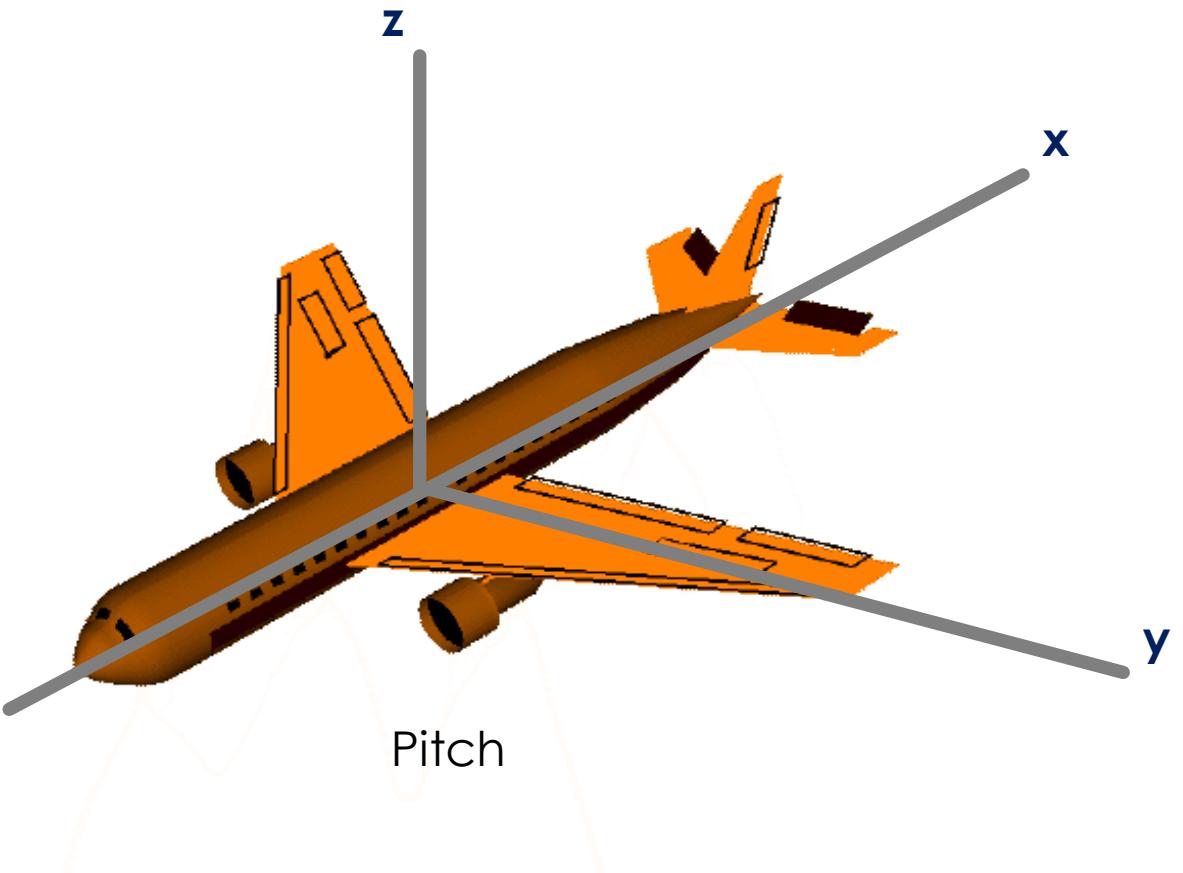


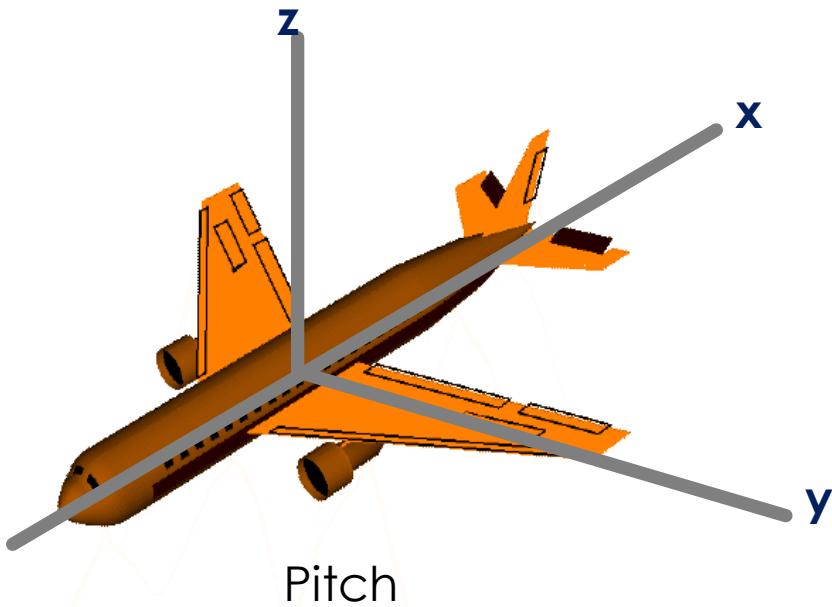
Gyro at rest



When to use “Gyro” & When to use “Accelerometer” ??

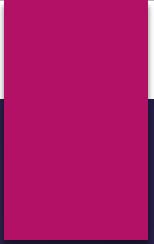






You have to use “gyro” here to decide by how much angle airplane should pitch or roll or yaw.

You can not use accelerometer here , because since it is a dynamic system with huge vibrations which make “accelerometer “ to produce wrong values as far as angle is concerned.



Why Gyroscopes are
used ?

Summary

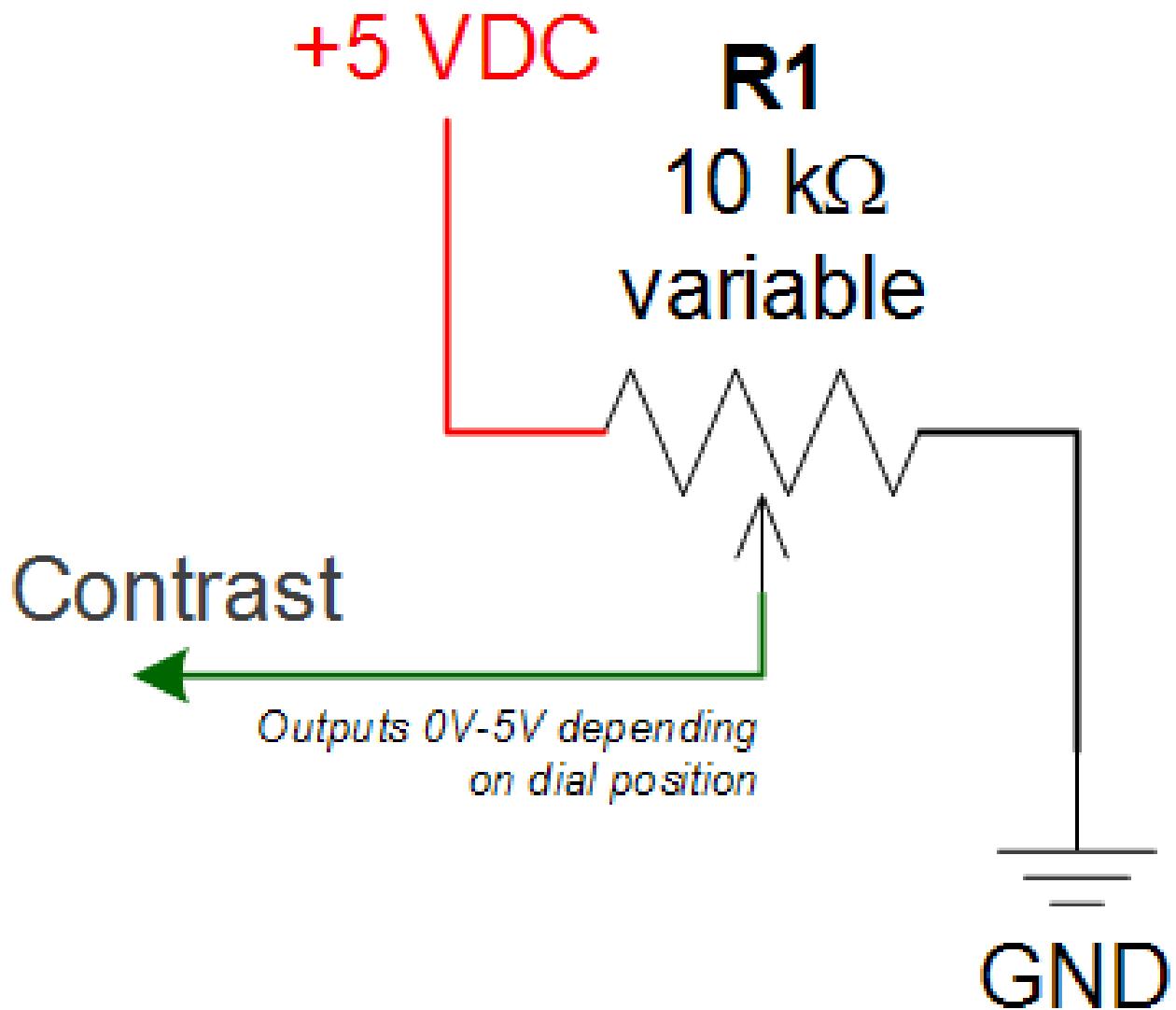
Use “Gyro” to calculate the “angle tilt” or to “measure the rotational speed” of an object

Angle tilt of an object can be measured using accelerometers also, but the object must be static . That means no external vibrational forces must be exerted on the sensor.

Example : to know by how much angle your quad-copter must pitch during flight you must use the “gyro” .

GND
Vcc
Contrast adjustment
Register Select (RS)
Read / Write (RW) Select
Enable (E)
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7

LED + (Pwr for backlight)
LED - (Pwr for backlight)





Display Data RAM

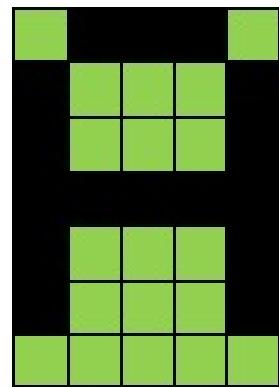
DDRAM
80 bytes

Character Generator RAM

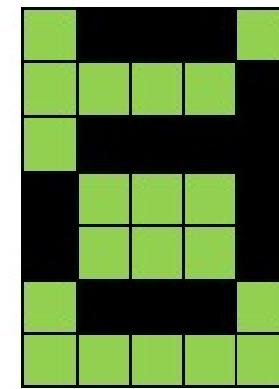
CGRAM
64 bytes

Character Generator ROM

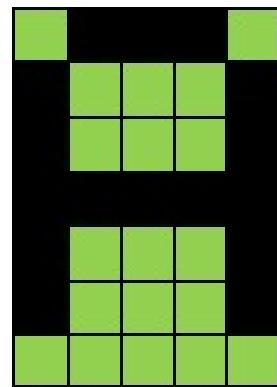
CGROM
9920 bits



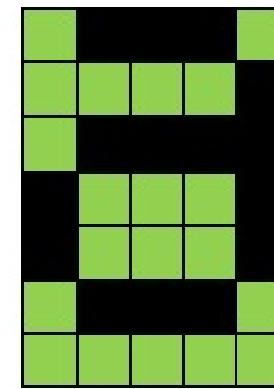
dot matrix character pattern for 'A'



dot matrix character pattern for 'a'



dot matrix character pattern for 'A'



dot matrix character pattern for 'a'

Initializing by Internal Reset Circuit

An internal reset circuit automatically initializes the HD44780U when the power is turned on. The following instructions are executed during the initialization. The busy flag (BF) is kept in the busy state until the initialization ends ($BF = 1$). The busy state lasts for 10 ms after V_{CC} rises to 4.5 V.

1. Display clear

2. Function set:

$DL = 1$; 8-bit interface data

$N = 0$; 1-line display

$F = 0$; 5×8 dot character font

3. Display on/off control:

$D = 0$; Display off

$C = 0$; Cursor off

$B = 0$; Blinking off

4. Entry mode set:

$I/D = 1$; Increment by 1

$S = 0$; No shift



Be sure the HD44780U is not in the busy state ($BF = 0$) before sending an instruction from the MPU to the HD44780U.

8x2



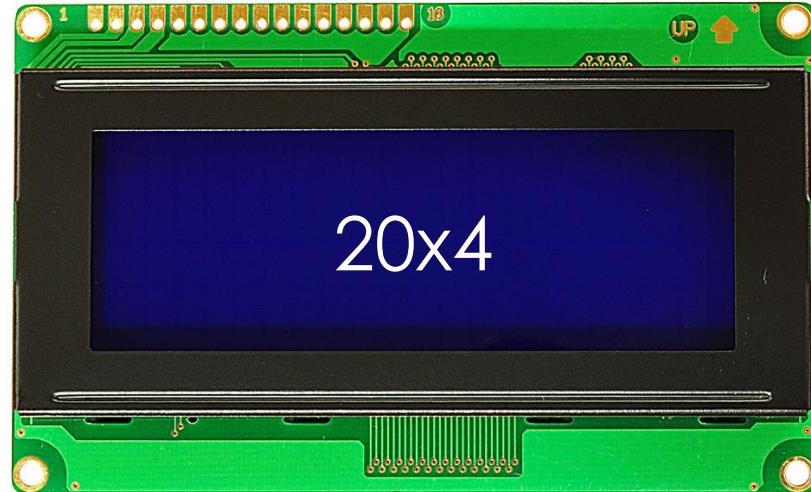
16x2



16x4

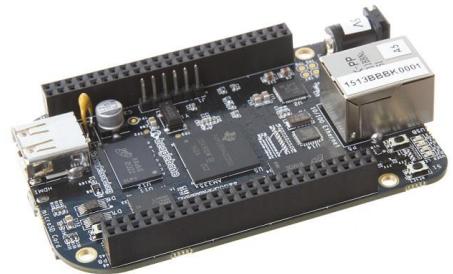


20x4



We will be using ,

1. Beaglebone hardware



2. 16x2 Lcd display

2. Potentiometer



2. Connecting wires, male to male



2. Breadboard

We will be using ,

1. Beaglebone hardware
2. 16x2 LCD display
3. Connecting wires, male to male
4. Breadboard
5. Potentiometer $100\text{K}\Omega$ or $10\text{K}\Omega$

In this section you will learn,

- 1.LCD pin descriptions
- 2.HD44780 LCD controller Details
- 3.Difference between DDARAM,CGRAM,CGROM
- 4.BBB and LCD connection details
- 5.LCD command sets and their usage
- 6.Code explanation : LCD driver and User application

We don't use any pre built libraries for LCD as we do in Arduino !