

BeagleBone Board Boot Options

Do you know that you can boot the AM335x SOC from the following boot sources???

- 1) NAND Flash
- 2) NOR Flash (eXecute In place, XIP)
- 3) USB
- 4) eMMC
- 5) SD card
- 6) Ethernet
- 6) UART
- 7) SPI

That means, you can keep the boot images in any of the above memory or peripheral and you can able to boot this SOC.

Please go to the page number 4106 of the TRM and you will find the below table.

Table 26-7. SYSBOOT Configuration Pins^[4]

| SYSBOOT[15:14] | SYSBOOT[13:12] | SYSBOOT[11:10] | SYSBOOT[9] | SYSBOOT[8] | SYSBOOT[7:6] | SYSBOOT[5] | SYSBOOT[4:0] | Boot Sequence | | | |
|--|---|--|---|---------------------------------------|-------------------------|---|---------------------|---------------|---|--------------------------------|---------|
| For all boot modes: Crystal Frequency | For all boot modes: Set to 00b for normal operation | For XIP boot: Muxed or non-muxed device For NAND boot: must be 00b | For NAND and NANDI2C boot: NAND ECC For Fast External Boot: must be 0b | For XIP boot: Bus width | For EMAC boot: PHY mode | For all boot modes: CLKOUT1 output enabled/disabled on XDMA_EVENT_INTR0 | | | | | |
| CONTROL_STATUS[23:22] | CONTROL_STATUS[21:20] | CONTROL_STATUS[19:18] | CONTROL_STATUS[17] | CONTROL_STATUS[16] | CONTROL_STATUS[7:6] | CONTROL_STATUS[5] | CONTROL_STATUS[4:0] | 1st | 2nd | 3rd | 4th |
| | | | | | | | 00000b | Reserved | | | |
| 00b = 19.2MHz 01b = 24MHz 10b = 25MHz 11b = 26MHz | 00b (all other values reserved) | For XIP boot: 00b = non-muxed device 10b = muxed device x1b = reserved | Don't care for ROM code ^[3] | 0 = 8-bit device 1 = 16-bit device | Don't care for ROM code | 0 = CLKOUT1 disabled 1 = CLKOUT1 enabled | 00001b | UART0 | XIP w/ WAIT ^[1] (MUX2) ^[2] 1 | MMC0 | SPI0 |
| 00b = 19.2MHz 01b = 24MHz 10b = 25MHz 11b = 26MHz | 00b (all other values reserved) | For NAND boot: must be 00b | 0 = ECC done by ROM 1 = ECC handled by NAND | Don't care for ROM code | Don't care for ROM code | 0 = CLKOUT1 disabled 1 = CLKOUT1 enabled | 00010b | UART0 | SPI0 | NAND | NANDI2C |
| 00b = 19.2MHz 01b = 24MHz 10b = 25MHz 11b = 26MHz | 00b (all other values reserved) | For XIP boot: 00b = non-muxed device 10b = muxed device x1b = reserved | Don't care for ROM code | 0 = 8-bit device 1 = 16-bit device | Don't care for ROM code | 0 = CLKOUT1 disabled 1 = CLKOUT1 enabled | 00011b | UART0 | SPI0 | XIP (MUX2) ^[2] 1 | MMC0 |
| 00b = 19.2MHz 01b = 24MHz 10b = 25MHz 11b = 26MHz | 00b (all other values reserved) | For XIP boot: 00b = non-muxed device 10b = muxed device x1b = reserved | 0 = ECC done by ROM 1 = ECC handled by | 0 = 8-bit device 1 = 16-bit device | Don't care for ROM code | 0 = CLKOUT1 disabled 1 = CLKOUT1 enabled | 00100b | UART0 | XIP w/ WAIT ^[1] (MUX1) | MMC0 | NAND |

Don't be scared by looking at the table, but just concentrate on **SYSBOOT[4:0]**

“SYSBOOT” is one of the register of this SOC and its first five bits decide the boot order .

Let's take an example,

When SYSBOOT [4:0] = 00000b (This is reserved, you cannot use this configuration)

When SYSBOOT [4:0] = 00001b

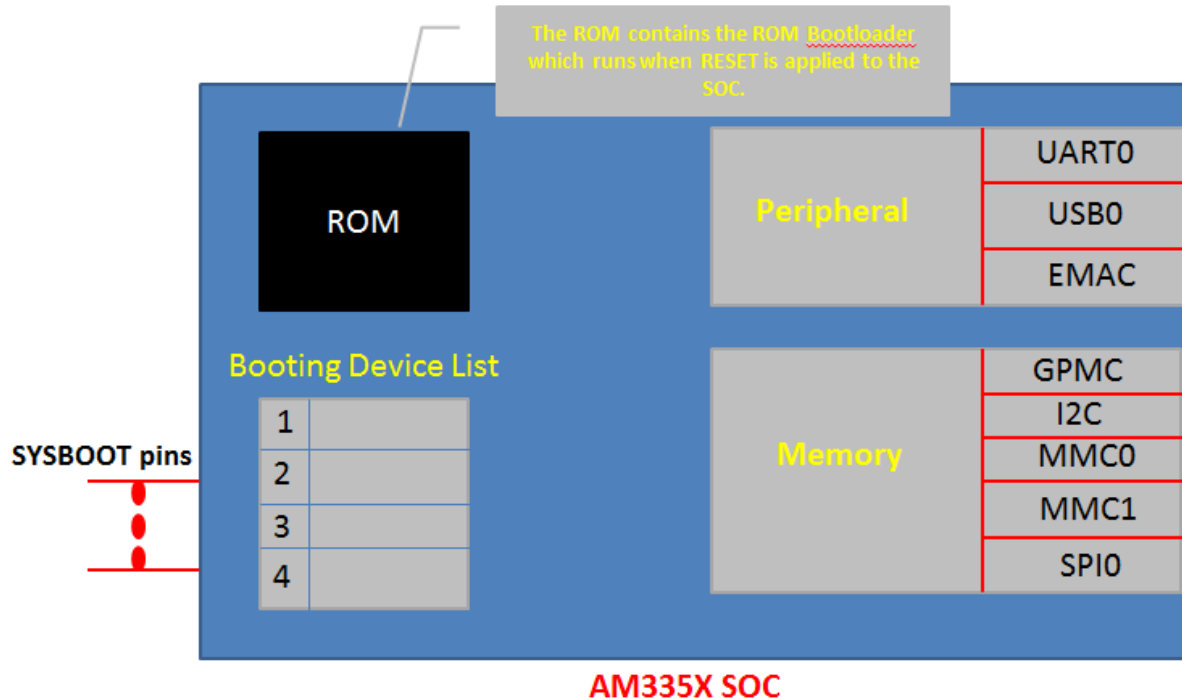
After the RESET if SYSBOOT [4:0] = 00001b , then SOC will try to boot from UART0 first , if fails, then it tries to boot from XIP(XIP stands for eXutable In Place memory like NOR Flash), if that also fails, then it will try to boot from MMC0, if no success, finally it tries to boot from SPI0, if that also fails, then SOC outputs the error message and stops.

Table 26-7. SYSBOOT Configuration Pins^[4]

| SYSBOOT[15:14] | SYSBOOT[13:12] | SYSBOOT[11:10] | SYSBOOT[9] | SYSBOOT[8] | SYSBOOT[7:6] | SYSBOOT[5] | SYSBOOT[4:0] | Boot Sequence | | | |
|--|---|--|---|---------------------------------------|-------------------------|---|---------------------|---------------|--|--------------------------------|---------|
| For all boot modes: Crystal Frequency | For all boot modes: Set to 00b for normal operation | For XIP boot: Muxed or non-muxed device For NAND boot: must be 00b | For NAND and NANDI2C boot: NAND ECC For Fast External Boot: must be 0b | For XIP boot: Bus width | For EMAC boot: PHY mode | For all boot modes: CLKOUT1 output enabled/disabled on XDMA_EVENT_INTR0 | | | | | |
| CONTROL_STATUS[23:22] | CONTROL_STATUS[21:20] | CONTROL_STATUS[19:18] | CONTROL_STATUS[17] | CONTROL_STATUS[16] | CONTROL_STATUS[7:6] | CONTROL_STATUS[5] | CONTROL_STATUS[4:0] | 1st | 2nd | 3rd | 4th |
| | | | | | | | 00000b | Reserved | | | |
| 00b = 19.2MHz 01b = 24MHz 10b = 25MHz 11b = 26MHz | 00b (all other values reserved) | For XIP boot: 00b = non-muxed device 10b = muxed device x1b = reserved | Don't care for ROM code ^[3] | 0 = 8-bit device 1 = 16-bit device | Don't care for ROM code | 0 = CLKOUT1 disabled 1 = CLKOUT1 enabled | 00001b | UART0 | XIP w/ WAIT ^[1] (MUX2) ^[2] 1 | MMC0 | SPI0 |
| 00b = 19.2MHz 01b = 24MHz 10b = 25MHz 11b = 26MHz | 00b (all other values reserved) | For NAND boot: must be 00b | 0 = ECC done by ROM 1 = ECC handled by NAND | Don't care for ROM code | Don't care for ROM code | 0 = CLKOUT1 disabled 1 = CLKOUT1 enabled | 00010b | UART0 | SPI0 | NAND | NANDI2C |
| 00b = 19.2MHz 01b = 24MHz 10b = 25MHz 11b = 26MHz | 00b (all other values reserved) | For XIP boot: 00b = non-muxed device 10b = muxed device x1b = reserved | Don't care for ROM code | 0 = 8-bit device 1 = 16-bit device | Don't care for ROM code | 0 = CLKOUT1 disabled 1 = CLKOUT1 enabled | 00011b | UART0 | SPI0 | XIP (MUX2) ^[2] 1 | MMC0 |
| 00b = 19.2MHz 01b = 24MHz 10b = 25MHz | 00b (all other values reserved) | For XIP boot: 00b = non-muxed device | 0 = ECC done by ROM 1 = ECC | 0 = 8-bit device 1 = 16-bit device | Don't care for ROM code | 0 = CLKOUT1 disabled 1 = CLKOUT1 enabled | 00100b | UART0 | XIP w/ WAIT ^[1] (MUX1) | MMC0 | NAND |

How SOC decides the boot order?

When you reset the SOC, the code stored in the ROM of the SOC runs first !



The code stored in the "ROM" is called **ROM boot loader**, this is programmed in to the ROM of the SOC during taping out of the chip, you cannot able to change it, why?? Because its ROM. Read only !!!!

The job of the ROM is to set up the SOC clock, watch dog timer, etc and also its major job is to load the **second stage boot loader**, what we call MLO or SPL, more on this later.

Now, from where it should load the second stage boot loader?

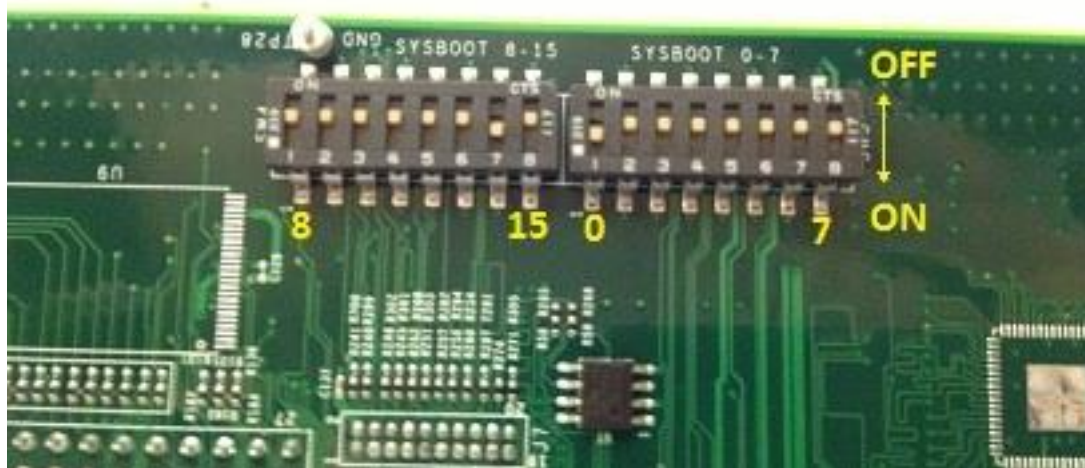
For that what ROM code does is, it reads the register **SYSBOOT [15:0]**, and based on the value of **SYSBOOT[4:0]** it prepares the list of booting devices.

The register *SYSBOOT [15:0]* value is decided by the voltage level on the *SYSBOOT pins*.

That is, let's say, if **SYSBOOT[4:0] = 00011b**, then boot order will be ,
UART0,SPI0.XIP,MMC0 (look at the table please !!!)

So, we can say that, The **SYSBOOT pins** configure the boot device order (set by SYSBOOT[4:0]).

Some board, will give you the control to change the SYSBOOT[15:0] value by using dip switches like below.



Here, by configuring the dip switches you can decide the value of SYSBOOT pins , thus enforcing the boot order which you like.

But in BBB, there are no such dip switches to configure the SYSBOOT pins. BBB has some other circuitry to decide the SYSBOOT pins voltage level, read on!!

BBB Boot order configuration circuit

In BBB you will find this circuitry, (you will find in SRM , not in TRM)

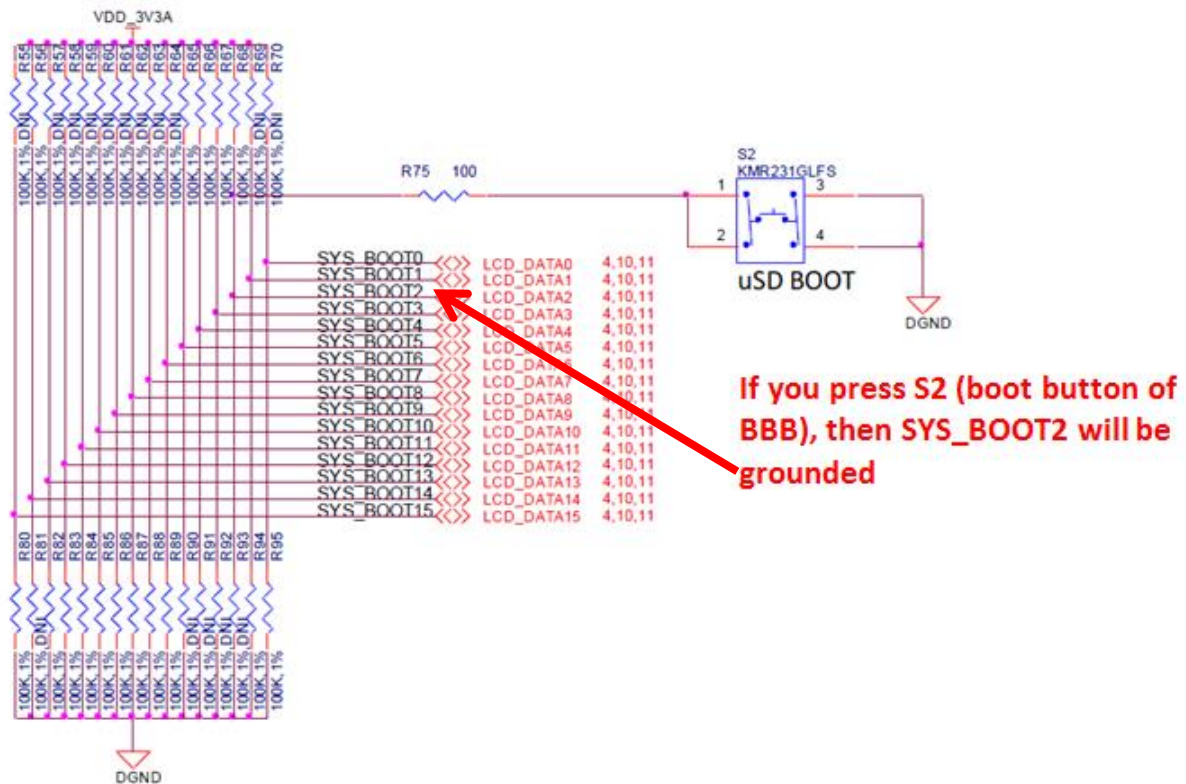


Figure 35. Processor Boot Configuration Design

Here observe that **SYS_BOOT2** is connected to a button S2 of the BBB (S2 is the boot button)

When you simply give power to the board, You will find the voltage level as below.

SYS_BOOT0 = 0V

SYS_BOOT1 = 0V

SYS_BOOT2 = 1V

SYS_BOOT3 = 1V

SYS_BOOT4 = 1V

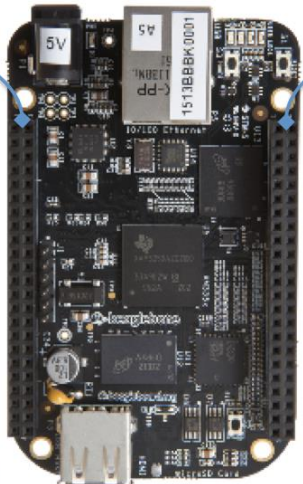
You can confirm this by measuring the voltage level using Mutlimeter,

If you have Multimeter, measure the voltage of 45,44,43,41,40 pins of the expansion header P8 of the board you will find SYSBOOT[4:0] = 11100

Expansion Headers

P9

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



P8

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

LEGEND

| |
|------------------------|
| Power/Ground/Reset |
| AVAILABLE DIGITAL |
| AVAILABLE PWM |
| SHARED I2C BUS |
| RECONFIGURABLE DIGITAL |
| ANALOG INPUTS (1.8V) |

Measure the voltage of these pins

and when you press the button S2, **SYS_BOOT2** will be grounded , so **SYSBOOT[4:0]= 11000**

Great! Now based on S2 (BBB boot button) we got 2 boot configurations

1) S2 Released (SYSBOOT[4:0] = 11100)

The boot order will be

1. MMC1 (eMMC)
2. MMC0 (SD card)
3. UART0
4. USB0

2) S2 pressed (SYSBOOT[4:0] = 11000) , The boot order will be

1. SPI0
2. MMC0 (SD card)
3. USB0
4. UART0

So, to conclude, there are 5 boot sources supported for this board including SPI.

1) eMMC Boot(MMC1) :

eMMC is connected over MMC1 interface, This is the fastest boot mode possible, eMMC is right there on your board, so need not to purchase any external components or memory chip. This is the default boot mode. As soon as you reset the board, the board start booting from loading the images stored in the eMMC.

If no proper boot image is found in the eMMC, then Processor will automatically try to boot from the next device on the list.

2) SD Boot :

If the default (that is booting from eMMC) boot mode fails, then it will try to boot from the SD card you connected to the sd card connector at MMC0 interface.

If you press S2 and then apply the power, then the board will try to boot from the SPI first, and if nothing is connected to SPI, it will try to boot from the MMC0 where our SD card is found

Also remember that we can use SD card boot to flash boot images on the eMMC. So if you want to write new images on the eMMC then you can boot through sd card, then write new images to eMMC, then reset the board, so that your board can boot using new images stored in the eMMC.

We will do these experiments later in this course. Don't worry!

3) Serial boot :

In this mode, the ROM code of the SOC will try to download the boot images from the serial port.

We have separate experiment on this boot mode and its very interesting.

4) USB BOOT :

You may be familiar with this boot mode, that is booting through usb stick!

You would have booted your PC through the usb stick. What you do is, you restart the PC, then press bios button to put the PC in to bios mode, there you select boot form usb, right?

It is very similar, when you reset the board, you can make your board to boot from the USB stick.

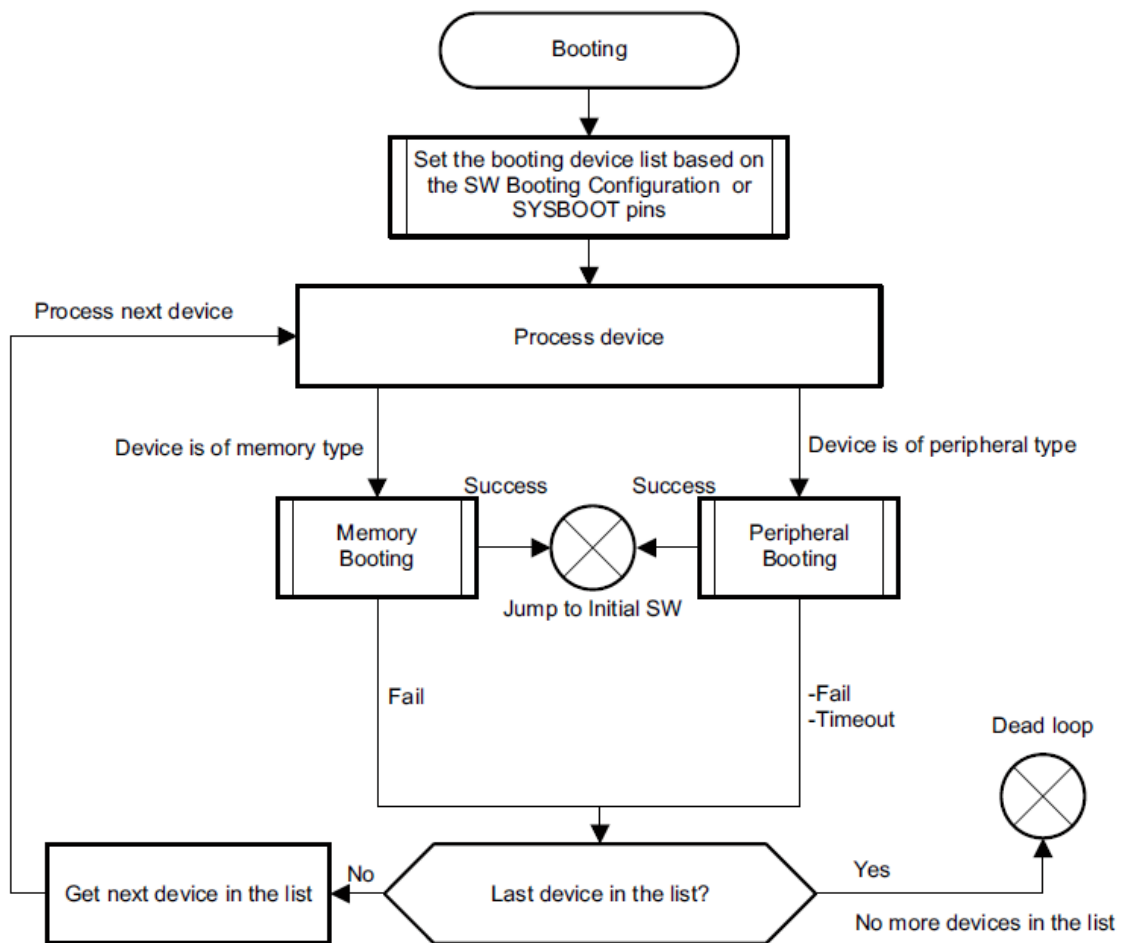
More details about ROM Code Booting procedure!

These are the 2 flow charts, you can find in the TRM of the am335x SOC

Please visit the page 4103 of the TRM

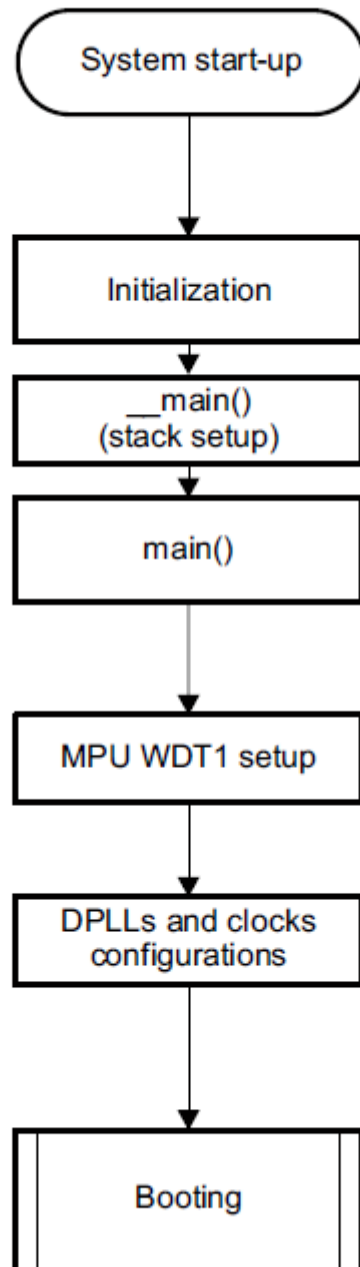
You will find the below image

Figure 26-6. ROM Code Booting Procedure



Here, you can see that the ROM code goes through its boot device list to load the second stage boot loader. ***It will prepare the boot devices list based on the value of SYSBOOT pins.***

Figure 26-5. ROM Code Startup Sequence



Great, so, that was the intro to the BBB boot configurations and boot modes. We will explore more about this practically as we make a progress in the course. If you have any questions feel free to post it in the course discussion board.

