AM335x board bringup tips

From Texas Instruments Wiki
Jump to: navigation, search

This page provides tips on debugging initialization and bring-up issues with the AM335x devices.

Contents

Power sequencing

- Ensure that correct power sequencing is occurring on your board. Power sequencing specifications can be found in the data manual. In general there are no timing related specs, only that each voltage rail in the sequence must ramp individually and reach 90% of its voltage before the following rail can start ramping up. For TPS65910 devices, this is done automatically as long as the boot signals of the TPS65910 are set correctly: BOOT0=0 and BOOT1=1. For TPS65217, this is also done automatically. You may need to verify correct connections between the PMIC and AM335x. Application notes are available for each PMIC device on the appropriate product page: TPS65910 TPS65217
- Ensure correct voltage levels on each voltage rail from the PMIC. Check for stable voltage on all rails. Unstable voltage on MPU rail may be a consequence of not connecting VDDS_MPU_MON correctly. Check our reference EVMs or other boards to ensure this is properly connected. Double check that VDDS_SRAM_CORE_BG and VDDS_SRAM_MPU_BB are powered by a 1.8V source.
- You can also check certain output voltages from internal LDOs. Note that these are **outputs** from internal LDOs, they are not to be powered from an external supply, or to be used to power other devices on the board. These outputs are typically signals called CAP_xxx. If these outputs are not at the following voltages, it may indicate that some of the input power rails are not correct:

    - CAP_VDD_SRAM_CORE = 1.2V
    - CAP_VDD_SRAM_MPU = 1.2V
    - CAP_VBB_MPU = 1.1V
    - CAP_VDD_RTC = 1.1V (Note: this signal is an LDO output when RTC_KALDO_ENn is low. When RTC_KALDO_ENn=1, it is an input power supply, which needs to be powered by a 1.1V supply. See AM335x Schematic Checklist - RTC Section for more info)
- Check the system clock (ie, the high frequency clock 19.2, 24, 25, 26 MHz). Ensure this is outputting the expected frequency and the signal is swinging rail to rail (should be a 1.8V signal). If you have an external oscillator, it should be outputting a square wave at the desired frequency. If you are using a crystal (ie, using the internal oscillator of the AM335x), you should have a sine wave at the desired frequency at XTALIN. XTALOUT should be similar (may be distorted a little). If using a crystal connected directly to AM335x (XTALIN and XTALOUT), ensure capacitor values are correct.
- Check the power on reset signal PORZ. This signal should stay low throughout the power sequencing and go high when the power AND high frequency clock are stable.

Is the device alive

If you have gone through the power sequencing check list above, there are a couple of things to try:

- If you connect a UART cable to your debug port (which should be connected to UART0), you should see a steady stream of 'C' characters output on a terminal window (such as Teraterm). This means that the ROM is executing and is attempting to boot. If you see this you can move on to the next section. Note that you will only see this output if UART0 in in your boot sequence. If you don't have the ability to change the boot sequence, then try the JTAG method below.
- If you don't see the 'C' output with UART0 in the boot sequence, you may have something wrong with your device. However, you may have an issue with the UART0 data path, or you may have a bad cable. Probe the UART0_TXD signal and see if you observe any toggling on the signal. If you do, there is most likely a problem with your UART transceiver or cable.
- Another way to check is to connect a JTAG cable and see if you can connect using Code Composer Studio or other JTAG debugger. If you can successfully connect through JTAG, you can move on to the next section
- Setting SYSBOOT[5] = 1 will output the system clock on CLKOUT1. You should see a square wave clock of the same frequency as the system input clock. This will give you a good indication that the clock is propagating correctly through the device.

Boot sequence

Read through this section if your device does not produce any console output (especially when trying to boot linux), or is otherwise not booting as expected. Boot issues can be quite involved depending on what source you are booting from.

- First check your SYS_BOOT signals to ensure that you have the correct boot sequence. Boot sequence tables can be found in the initialization chapter of the TRM. These define which sources the device will potentially boot from, and in what order. If you have access to memory (via JTAG, for example), you can see what the processor read from the SYS_BOOT signals by reading the following registers (check the TRM for more details):
    - CONTROL_STATUS register (0x44E10040)
        - Bits 7:0 will reflect what the processor read on SYSBOOT[7:0]
        - Bit 16:23 will reflect what the processor read on SYSBOOT[15:8]
- The ROM will attempt to boot from each boot source in the defined boot sequence. If the first boot source fails to boot, the ROM will move on to the next one in the sequence. Keep in mind that some boot sources take some time to timeout if that boot source isn't avaiable (for example, USB or Ethernet). Thus, your bootup make take several seconds longer if your typical boot source is further down in the boot sequence. This is particularly evident on the AM335x EVM, where you will see that MMC boot takes a few seconds because MMC0 is fourth in the boot list.
- ensure that the SYSBOOT[15:14] indicates the correct input frequency that you are using.
- If you can connect through JTAG, bring up a disassembly window to see where the program counter (PC) is. It should be somewhere in 0x20000-0x2C000 (ie, somewhere in ROM code). If you are in the interrupt table vectors (0x20080-0x200BC), the code is stuck in a dead loop. Refer to the Initialization chapter of the TRM to determine which dead loop is it. This will give you a clue on what went wrong. Typically it will be a data abort, which means the processor attempted to access a bad address.

If the above is confirmed, move on to the next section for tips on determining why your boot source is not working properly

Peripheral/Memory Booting Issues

If you are trying to boot Linux, and you get no console output (other than a series of 'C' characters), then the boot ROM could not successfully boot from any of the boot sources dictated by the SYS_BOOT sequence. Check the hardware signaling between the AM335x and the device you are booting from. For example, if you are booting from NAND, simple checks that the chip enable (CE) signal or read (RD) signal are toggling during boot will help determine if you have an interface issue. If booting from MMC/SD card, you can check to see if you get a MMC_CLK during boot. You can check the signals of other interfaces to determine if you at least have some activity.

If you see proper activity and signal levels on the interfaces you are trying to boot from, here are some further tips to identify boot problems:

- Signal Conflicts - The ROM will attempt to boot from the interfaces defined in the boot sequence. When the first boot source fails, the ROM will move on to the next boot source, however, the signals it configured for the first boot source will remain in that configuration. Subsequently, the ROM configures interfaces for the attempted boot source, but does not reconfigure the interfaces back to default states when it moves on to the next source in the sequence. If your design relies on the default IO configurations of certain signals, the ROM may reconfigure these to undesirable states. Ensure signals to external components are in the proper state during the desired boot modes.
- NAND - you may have software related issues. Ensure that you are using the correct ECC scheme when programming the NAND. Also ensure that the initial boot header is correct (check the Initialization chapter of the TRM for specific header information). The AM335x CCS Flashing Tools Guide wiki has information on how to use a CCS tool to program the NAND on the EVM.
- MMC/SD card - you may have software related issues. Ensure that your card is formatted correctly and that a file names MLO is present on the FAT file partition of the card. More info can be found here: http://processors.wiki.ti.com/index.php/AM335x_U-Boot_User%27s_Guide. Also be aware of some of the restrictions on certain MMC ports with respect to booting that are outlined in the TRM in section 26.1.7.5:
    - MMC0 supports booting from the MMC/SD card cage and also supports booting from eMMC/eSD/managed NAND memory devices with less than 4GB capacity.
    - MMC1 supports booting from eMMC/eSD/managed NAND memory device with 4GB capacity or greater.

- UART - ensure the UART signals are toggling and the transceiver is connected correctly. Ensure proper voltage is powering the transceiver. Also ensure that you are using UART0
- USB - refer to the AM335x errata for silicon issues concerning USB boot. Ensure proper signaling (USB0_DP/DM) and USB0_ID is set up for peripheral mode (USB0_ID should be unconnected). You may need to connect a USB protocol analyzer to ensure proper USB transactions are being performed
- EMAC - refer to the AM335x errata for silicon issues concerning Ethernet boot. Ensure that the Ethernet PHY is properly powered up before the ROM attempts to access it. The Ethernet PHY may require some time for power sequencing and reset signaling before it can be accesses. Depending on the boot sequence chosen, the ROM may access the Ethernet PHY before it is ready.
- SPI - Make sure the MISO/MOSI signals are connected according to the details outlined in the Initialization chapter of the TRM (see section 26.1.7.6 for details on SPI boot)

You may also be able to use the Tracing Vectors (see section below) to determine which boot sources the ROM attempted.

If you see some console messages, then the ROM successfully found a boot source with a valid image. If the boot still fails in uboot or kernel boot, see the next section on DDR configuration. You most likely have configuration issues with your DDR memory.

DDR configuration

If you are trying to boot linux and are getting stuck in the uboot or the kernel boot, double check that your DDR configuration is correct. This wiki has more details on configuring the DDR controller and DDR PHY registers:
http://processors.wiki.ti.com/index.php/AM335x_EMIF_Configuration_tips

For every custom board design, you must configure the DDR AC timings specific to the DDR that you are using. Check the AC timings spreadsheet on the above wiki to help determine these register values. Furthermore, you must perform the DDR PHY leveling procedure to tune the DDR PHY timings for your layout. More details for this can be found in the DDR PHY Registers section in the above mentioned wiki.

If you believe that you have the correct configuration, but are still getting stuck in u-boot, you can double check your configuration by connecting with JTAG. After a boot failure, connect your JTAG pod, launch CCS, and connect the target (make sure a loaded GEL file is not doing any initialization upon connect). Open a memory window at 0x80000000 (this is the beginning of DDR), and see if you can successfully peek and poke values. If you see unstable results in the window (eg, other values changing after writing a value), then your DDR configuration is still not correct. Double check the above procedure and the above mentioned wiki.

If you are getting stuck in the kernel boot (a lot of times heaving DDR loading can occur during the initial kernel decompression), you can still try to break in with JTAG as described above, but you will need to disable MMU in CCS to be able to access the DDR at 0x80000000 (When connected to the target, go to Tools->ARM Advanced Features, and uncheck MMU Enabled).

You may also want to check any hardware issues. If you are seeing single bits that are incorrect, sometimes this can occur with an improper VREF. VREF should be half of the VDDS_DDR voltage and needs to be precise. This is the reason that 1% resistors are required for the voltage divider (or your design may have VREF being generated from the VTT regulator). Ensure this voltage is at the correct level and stable throughout the boot up sequence.

Analyzing Boot Issues with CCS and JTAG

Using a DSS script for CCS you can easily parse some of the important register details into easily digestible info.

1. Download am335x-boot.dss.
2. Launch CCS.
3. Create an appropriate target configuration file for connecting to your board.
    - File -> New -> Target Configuration File
    - Supply a name/location for the file.
    - View -> Target Configurations to see the available target configurations (yours should now be among them!).
    - Double-click your file in the Target Configurations panel to open it for editing.
    - Select your emulator and processor. Save.
4. Launch the debugger, but **do not connect to any CPUs**.
    - In the Target Configurations window, right-click on your ccxml file and select "Launch Selected Configuration"
5. Launch the scripting console by going to View -> Scripting Console.
6. Load am335x-boot.dss in the scripting console by executing "loadJSFile <path-to-dss-file>/am335x-boot.dss".
7. It will generate a am335x-boot-analysis_yyyy-mm-dd_hhmmss.txt file on your desktop that can be used for quick analysis.

Still having booting issues...

If you have gotten to this point and still cannot boot correctly, you most likely have software issues which goes beyond the intent of this wiki. Here are some resources which can provide further help:

- Texas Instruments AM335x E2E forum
- Sitara Linux Software Developer's Guide

Further Information

Tracing vectors

- Tracing vectors give you some indication what the ROM code executed in its attempt to boot. This can help you determine how far the ROM got before failing to boot, and can give you some idea as to why it failed. Information on the tracing vector can be found in the Initialization chapter of the TRM. Here are some tips:

- These are the pertinent registes for tracing data. The first 3 (0x40, 0x44, and 0x48) contain the tracing data from the last boot sequence. The tracing vector is generally around 96 bits of information, spread across 3 32-bit words.

    -

    - 0x4030CE40 Current tracing vector, word 1
    - 0x4030CE44 Current tracing vector, word 2
    - 0x4030CE48 Current tracing vector, word 3

- This register contains a copy of the reset status register (PRM_RSTST), which will give you an indication of why the processor got reset (for example, External warm reset, MPU watchdog reset, Global software reset, Global cold reset)
    - 0x4030CE4C Current copy of the PRM_RSTST register (reset reasons)

Here are some examples on how to interpret the tracing vectors:

Tracing Example 1

SYSBOOT = 0x4021 (boot from UART0,XIP w/ WAIT, MMC0, SPI0) After failed boot, cold reset tracing vector reads:

- 0x4030CE40 = 0x0000907F
- 0x4030CE44 = 0x00001000
- 0x4030CE48 = 0x00011004

Comparing this to the tracing vector table in the Initialization Chapter of the TRM, we see that the following bits are set:

Bit	**Meaning**	**Comment**
Trace Vector 1		
0	Passed the public reset vector	Reset occurred
1	Entered main function	ROM code hit the main() function in the C code
2	Running after cold reset	
3	Main booting routine entered	
4	Memory booting started	Indicates that the ROM attempted to boot from a memory source
5	Peripheral booting started	Indicates that the ROM attempted to boot from a peripheral source
6	Booting loop reached last device	Indicates that the ROM tried all boot sources in the booting list
12	Device Initialized	Indicates ROM tried initialized a peripheral boot source
15	Peripheral booting failed	Indicates that booting from the peripheral device failed. (in this cas
Trace Vector 2		
12	Memory booting trial 0	Indicates the ROM started memory booting

Trace Vector 3		
2	Memory booting device XIPWAIT	Indicates boot was attempted from XIPWAIT (in this case, that is s list)
16	Peripheral booting device UART0	Indicates UART booting was attempted

Tracing Example 2

SYSBOOT = 0x4021 (boot from UART0,XIP w/ WAIT, MMC0, SPI0) After failed boot, cold reset tracing vector reads:

- 0x4030CE40 = 0x001090BE
- 0x4030CE44 = 0x00011000
- 0x4030CE48 = 0x00011004

Comparing this to the tracing vector table in the Initialization Chapter of the TRM, we see that the following bits are set:

Bit	Meaning	Comment
Trace Vector 1		
0	Passed the public reset vector	Reset occurred
1	Entered main function	ROM code hit the main() function in the C code
2	Running after cold reset	
3	Main booting routine entered	
4	Memory booting started	Indicates that the ROM attempted to boot from a memory source
5	Peripheral booting started	Indicates that the ROM attempted to boot from a peripheral source
6	Booting loop reached last device	Indicates that the ROM tried all boot sources in the booting list
12	Device Initialized	Indicates ROM tried initialized a peripheral boot source
15	Peripheral booting failed	Indicates that booting from the peripheral device failed. (in this cas
Trace Vector 2		
12	Memory booting trial 0	Indicates the ROM started memory booting
Trace Vector 3		
2	Memory booting device XIPWAIT	Indicates boot was attempted from XIPWAIT (in this case, that is s list)
16	Peripheral booting device UART0	Indicates UART booting was attempted

Category:

- AM335x

Navigation menu

Personal tools

- Log in
- Request account

Namespaces

- Page
- Discussion

Variants

Views

- Read
- View source
- View history

More

Search

Search Go

Navigation

- Main Page
- All pages
- All categories
- Recent changes
- Random page
- Help

Toolbox

- What links here
- Related changes
- Special pages
- Printable version
- Permanent link
- Page information

- This page was last edited on 31 January 2017, at 20:31.
- Content is available under Creative Commons Attribution-ShareAlike unless otherwise noted.

- Privacy policy
- About Texas Instruments Wiki
- Disclaimers
- Terms of Use

- 

-