Ronetix Application Note AN011: DaVinci DM365 Linux Kernel Debugging and NAND Flash Programming

1. Overview

This article shows you how to debug Linux kernel and completely to program the DM365 EVM board using PEEDI. This method can be used for restoring the factory images to the board as well as end product manufacture programming. First it is described how to do it manually and after that how to configure PEEDI to do it by a single push of a button.

2. Prerequisites

- a PEEDI
- a DM365 board
- UBL, U-BOOT, KERNEL and RAMDISK images
- A host machine running an FTP server

3. Setting PEEDI

First you need to prepare your CFG file. Since the CFG file itself is not that simple, you can download it from https://download.ronetix.at/peedi/cfg_examples/arm9/tms320dm365-DM365EVM.cfg. The following covers only the FLASH settings. As you need to program four different images (UBL, U-BOOT, Linux kernel and the RAMDISK image), you need to define four FLASH profiles in the CFG. You can set the CORE0_PATH parameter to point to where the files are located. This way in the FLASH profiles or with the commands you can use only the file name, not the whole path. Later, if you decide to use the PEEDI in stand-alone mode you can copy the files onto an MMC/SD card and you will need to change only the CORE0_PATH. For now I set this parameter to point my FTP server root directory.

```
[PLATFORM_ARM]
…
CORE0_FLASH0 = NAND_UBL
CORE0_FLASH1 = NAND_UBOOT
CORE0_FLASH2 = NAND_KERNEL
CORE0_FLASH3 = NAND_RAMDISK
…
CORE0_PATH = "ftp://user:[email protected]/"

[NAND_UBL]
CPU = TMS320DM365
CHIP = NAND_FLASH
DATA_BASE = 0x02000000 ; data
CMD_BASE = 0x02000010 ; commands (CLE)
ADDR_BASE = 0x0200000A ; addresses (ALE)
FILE = ubl.bin, BIN, 0x40000
OOB_INFO = DM365_BOOT
BURST_MODE = YES
DAVINCI_UBL_DESC_TYPE = 0
DAVINCI_UBL_DESCRIPTOR_MAGIC = 0xA1ACED00
DAVINCI_UBL_DESCRIPTOR_ENTRY_POINT = 0x100
```

```
DAVINCI_UBL_DESCRIPTOR_LOAD_ADDR = 0
DAVINCI_UBL_MAX_IMAGE_SIZE = 30*1024

[NAND_UBOOT]
CPU = TMS320DM365
CHIP = NAND_FLASH
DATA_BASE = 0x02000000 ; data
CMD_BASE = 0x02000010 ; commands (CLE)
ADDR_BASE = 0x0200000A ; addresses (ALE)
FILE = u-boot.bin, BIN, 0x320000
OOB_INFO = DM365_BOOT
BURST_MODE = YES
DAVINCI_UBL_DESC_TYPE = 0
DAVINCI_UBL_DESCRIPTOR_MAGIC = 0xA1ACED66
DAVINCI_UBL_DESCRIPTOR_ENTRY_POINT = 0x81080000
DAVINCI_UBL_DESCRIPTOR_LOAD_ADDR = 0x81080000

[NAND_KERNEL]
CPU = TMS320DM365
CHIP = NAND_FLASH
DATA_BASE = 0x02000000 ; data
CMD_BASE = 0x02000010 ; commands (CLE)
ADDR_BASE = 0x0200000A ; addresses (ALE)
FILE = uImage, BIN, 0x00600000
OOB_INFO = DM365_LINUX
BURST_MODE = YES

[NAND_RAMDISK]
CPU = TMS320DM365
CHIP = NAND_FLASH
DATA_BASE = 0x02000000 ; data
CMD_BASE = 0x02000010 ; commands (CLE)
ADDR_BASE = 0x0200000A ; addresses (ALE)
FILE = ramdisk.bin, BIN, 0x00900000
OOB_INFO = DM365_LINUX
BURST_MODE = YES
```

As you can see from the first two profiles – PEEDI can auto program the image descriptors using info given by the user.

The “CPU = TMS320DM365” parameter tells PEEDI to check for Bad Block tables (main and mirror) and if not found, to create them on the last two good blocks. Now we are ready to program into the NAND chip UBL, U-BOOT, KERNEL and RAMDISK.

4. Programming the board

Using **flash set** command you can select the FLASH profile you want to use and then **flash program** to program the NAND:

```
peedi> flash set 0
peedi> flash erase
peedi> flash program
peedi> flash set 1
peedi> flash program
peedi> flash set 2
peedi> flash program
```

```
peedi> flash set 3
peedi> flash program
```

Disconnect the board and verify it boots normally. You might need to set the U-BOOT environment variables to load the kernel and pass proper command line. You could dump the U-BOOT environment and later just program it, so you don't need to set it manually every time you erase it. Now you can put all the commands into a script for an easy automated execution:

```
[prog_all]
flash set 0
flash erase
flash program
flash set 1
flash program
flash set 2
flash program
flash set 3
flash program
```

And put the script in the [ACTIONS] list so it can be started using the PEEDI buttons:

```
[ACTIONS] ; user defined scripts
;AUTORUN = 1 ; executed on every target connect
1 = prog_all
```

The script can be started in three ways:

- by using the PEEDI **run $prog_all** command
- by selecting and starting script 1 using the front PEEDI buttons
- if the AUTORUN = 1 line is uncommented the script will be started every time a board is connected to PEEDI.

The way 1 is useful for a board restore. Ways 2 and 3 are useful for production programming since non-qualified personnel can easily be instructed on how to work with PEEDI.

5. Linux kernel debugging

For Linux kernel debugging you need an ELF of the kernel compiled with debug info and optimization switched off (-g -O0). The board should be set to load and start the kernel either from the NAND or any other source. For debugging, best is loading it via the Ethernet, this way you don't need to program the NAND from build to build. Once you build the kernel, locate the vmlinux file (it should be in the root directory of the linux kernel project) and use the nm tool to find the address of the start_kernel function like this:

```
nm vmlinux | grep start_kernel
```

Now use CFG without INIT section so after PEEDI connects to the CPU it does not make any initialization and make sure breakpoint mode is set to SOFT (we are going to debug in RAM). After PEEDI is connected use "break add hard ADDRESS" to add a hardware breakpoint at the start_kernel address. Now "go" to start the target and wait until it breaks at the breakpoint. Next you can start insight in the kernel root directory pointing the kernel ELF image:

```
arm-elf-insight vmlinux
```

Now connect to PEEDI in the insight console:

target remote YOUR_PEEDI_IP:2000Next use the “si” command just to refresh the view. From now on you can add breakpoints, step, go, stop and so on the target. To start execution use the C button or the (c)ontinue command in the insight console or the resume button in eclipse. Once you do the previous, you can automate your work by putting this in the INIT section:

```
break add hard ADDRESS ; add hardware break at start_kernel
go ; start the CPU
wait 30000 stop ; wait for the target to break with timeout of 30 sec, it might need more time,
check the board boot up time
break del all ; remove the break
beep 500 500 ; beep to signal ready for debug
```

And you can also put the target remote and si command in a .gdbinit file so these are auto executed on gdb/insight start. Of course you can use eclipse too for debugging.

6. Conclusion

At first glance things might seem too complicated, but it gets easier after the first iteration. On the other hand all the complicated setting is made once and after that everything is easy as one push of a button.

Share on LinkedinShare on FacebookShare on Reddit on TwitterShare Via EmailShare on Whatsapp Share