SMP863x tutorial

How to build and use the components of the standalone SMP863x development kit

Revision history

when	who	what
04/12/07	Cat	Update for GCC4.0.3, uClibC 0.9.23.3, binutils 2.17
02/17/07	Cat	Update for GCC4 compile
2006-10-19	Cat	Update CPUKey and expand explanation for kernel 2.6.15 compile
	IE Team	Helped debug typos
2006-05-17	Bertrand	Corrected factual mistake on slide 16
2006-04-21	Bertrand	Added 2.6 kernel specific instructions
2006-03-10	Bertrand	Added “Board Recovery Guide” tutorial

Contents

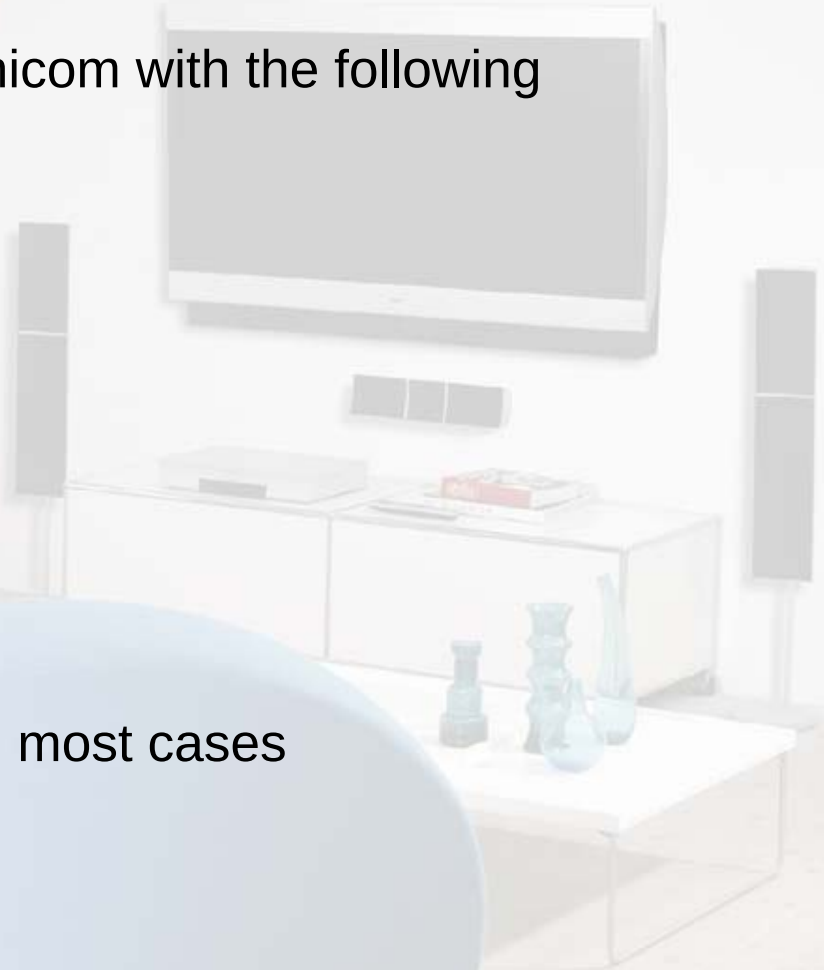
Getting started

- What you need:
  - Odyssey SMP863x board with back plane or a standalone SMP863x board (Envision or Vantage). This tutorial assumes the board already comes with a boot loader.
  - Serial port adapter board and a serial (null modem) cable.
  - PC running Linux (2.4 or 2.6 kernel) with:
    - NFS server
    - minicom, uuencode (from the 'sharutils' package)
    - genromfs, flex, bison, gettext

Sigma's Software

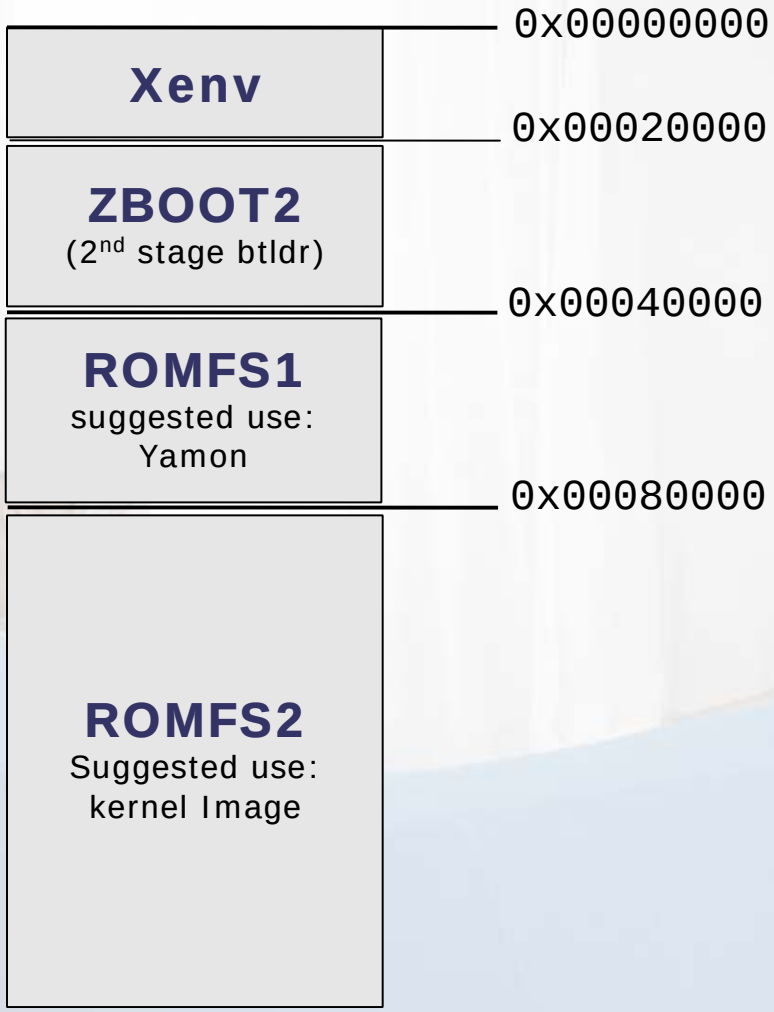
- Sigma Designs provides:
  - smp86xx_toolchain_2.x.yyy.z package
  - smp86xx_boot_loader_2.x.yyy.z package
  - smp86xx_kernel_source_2.x.yyy.z package
  - mrua_SMP8634_2.x.yyy.z_dev.mips package
  - CPU_KEYS_SMP8634_xosMcX-Y

Set the host up



1. On the host, you need to setup minicom with the following parameters
    1. Baud rate = 115200 bps
    2. Data = 8 bits
    3. Parity = none
    4. Stop bit = 1
    5. Flow control = none
    6. Port = /dev/ttyS#
    7. These are the default values in most cases
1. Configure the network on the host.
2. Plug-in everything and let's start!

Parallel Flash Organization



Flash organization – offsets are default values

- **The parallel flash is organized in four zones:**
  - **the Xenv, holding setup data**
  - **the reset vector, zboot2**
  - **two ROMFS zones**
- **XOS will check the Xenv, then run zboot2 on the CPU in DRAM.**
- **In turn, zboot2 will scan the root folder of either ROMFS zone, and process files with xload format.**
- **zboot2 looks for the ROMFS zones at a location specified in the Xenv**

What we need to produce

- We need to build the following elements to get a running system:
  - a MIPS toolchain for the host, to build the other elements
  - a valid Xenv in tune with the hardware
  - a zboot2 binary, signed and optionally encrypted
  - a Yamon binary, with a ZBF header, signed and optionally encrypted, in a ROMFS
  - a Linux binary, with a ZBF header, signed and optionally encrypted, in a ROMFS
  - the MRUA drivers and sample applications

The toolchain

- Use the smp86xx_toolchain_2.x.yyy.z.tar.bz2 package
- On the host:
  - `host:$ tar xjf smp86xx_toolchain_2.x.yyy.z.tar.bz2`
  - `host:$ cd smp86xx_toolchain_2.x.yyy.z`
  - `host:$ make menuconfig`
    - under 'Build options', choose the 'Toolchain and header file location' and 'Download site‘

    Default toolset are: GCC 3.4.2, binutils 2.15.91.0.2, uClibc 0.9.27.

    If you want to use newer GCC 4.0.4, binutils 2.17, uClibc 0.9.28.3:

    - Under ‘toolchain options’
    - Select newer uClibc 0.9.28.3, uClibc 4.0.4, and binutils 2.17

    You can also optionally select:

    * Kernel 2.6.11 headers and gdb 6.5

The toolchain

- Exit menuconfig & save
- Make sure that internet connection is available
  - `host:$ make`
- After 30 to 45 minutes, your toolchain is ready. The build process creates a (bash) environment file 'toolchain-path.env'
  - `host:$ source toolchain-path.env`

Signature and encryption keys

- Development keys for signature and encryption of zboot2, YAMON and the Linux kernel are provided in the `CPU_KEYS_SMP8634_xosMc0-1.tar.gz` package.
- Simply untar the package:
  - `host:$ tar xzf CPU_KEYS_SMP8634_xosMc0-1.tar.gz`
- Source the env file:
  - `host:$ source CPU_KEYS.env`

The boot loaders (1)

- Use the smp86xx_boot_loader_2.x.yyy.z.tar.bz2 package
- On the host:
  - `host:$ tar xjf smp86xx_boot_loader_2.x.yyy.z.tar.bz2`
  - `host:$ cd smp86xx_boot_loader_2.x.yyy.z`
- Run 'make' with no argument to be reminded what targets are available.
- Update mktemp (mktemp-1.5-7.i386.rpm) and OpenSSL 0.98a or later if your Linux installation provides older versions.

The boot loaders (2): signature block

- There are several generic Xenv configurations designed for Sigma Designs development platforms, available in the 'tools/genxenv/configs' directory.
- However, there is no default configuration.
- In order to use a given configuration, you need to place it in a file named `xenv.config` in the main package directory.
  - hint: to use a Sigma Designs generic configuration, simply create a soft link to it, named `xenv.config`, in the main package directory

The boot loaders (3): zboot

ln –s tools/genxenv/configs/852-E2.config xenv.config

(852-E2.config is Envision/Vantage standalone and 849-E2.config is Odyssey PCI)

- Simply run '`make zboot`' in the main package directory.
- You get the the zboot2 binary with the embedded Xenv block in the '`bin`' directory as '`zboot2.bin`'
- Note: this image comprises the Xenv block as well as the zboot2 image. It currently assumes that Xenv defines the zboot2 location at 0x20000 from parallel flash.

The boot loaders (4): YAMON

- Simply run '`make yamon`' in the main package directory
- The build process takes care of
  - producing the YAMON binary,
  - adding the ZBF header suitable for processing by zboot
  - signing it
  - putting the resulting binary in a ROMFS
- You get the ROMFS binary in the 'bin' directory as 'zbimage-yamon'

The boot loaders (5):
Committing to the parallel flash

- Remember:
  - You need a board with a working version of YAMON to proceed (to be able to flash)
  - If upgrading from an old board (where the reset vector was that of the YAMON package), you need to write zboot and YAMON at the same time (!)
  - You will copy some data to DRAM first. Any address between 0xb0100000 and 0xb1000000 is OK
- Follow these simple steps:
  - Initialize the parallel flash subsystem
    - `YAMON> pflash probe`

The boot loaders (6): Commiting to the parallel flash

- Download the zboot image to the DRAM
  - `YAMON> load uu -z 0xb0100000`
  - `host:$ gzip -c bin/zboot2.bin | uuencode x > /dev/ttyS0`
  - When downloading is done, the final size of the image (after unzipping) is given by YAMON
- Example:

```
Received 170297 (0x29939) bytes at address 0xb11400
Checksum = 0x1a89751f
Unzipping into address 0xb0100000
Output length: 0x00029b64(170852)
```

The boot loaders (7): Commiting to the parallel flash

- Flash the downloaded image to offset 0x0
  - `YAMON> pflash write 0x0 0xb0100000 <output length>`
  - `Example:pflash write 0x0 0xb0100000 0x29b64`
- If the command above fails, and if you really know what you are doing, you can use the '-f' option for pflash so that image will be updated blindly.
  - `YAMON> pflash write -f 0x0 0xb0100000 <size>`

The boot loaders (8): Commiting to the parallel flash

- Repeat the steps with the YAMON ROMFS container:
  - Download the image to the DRAM
    - `YAMON> load uu -z 0xb0100000`
    - `host:$ gzip -c bin/zbimage-yamon | uuencode x > /dev/ttyS0`
  - Flash the downloaded image to offset 0x40000
    - `YAMON> pflash write 0x40000 0xb0100000 <size>`
- If you are also upgrading your XOS, now is the time to do so. In specific cases, we will have provided you with documentation about that upgrade process (e.g. the zboot->zboot2 upgrade).
- Otherwise, you can reset the board. You should get back to the YAMON prompt.
- If you don't see the YAMON prompt, you will have to use either JTAG or programmer to reprogram the flash.

The Linux kernel: overview

- There are two components to a running Linux system: the kernel and the root file system
- In the Sigma Designs release, the two components are integrated and interacting:
  - the kernel source can install its modules into the root file system image
  - the kernel source can pull the root file system image to use it as an initial embedded RAMdisk.
- Both components thus need to be built together.
- Detailed explanations are available in the README files in the kernel source and rootfs packages.

The root file system (1): get the source

- Use the smp86xx_rootfs_2.x.yyy.z.tar.bz2 package
- On the host:
  - `host:$ tar xjf smp86xx_rootfs_2.x.yyy.z.tar.bz2`
  - `host:$ cd smp86xx_rootfs_2.x.yyy.z`
- **mktemp needs to be (1.5-7 or later)**
- Run 'make' to confirm the configuration (same as ‘make menuconfig’)
  - Choose ‘Package selection for the target’ menu.
  - Make sure ‘customized’ is chosen
  - You can optionally choose busybox version 1.2.2.1 here
  - Exit & save

The root file system (1): get the source

- To prepare for usage with DCC-HD:
  - cd target/generic/target_skeleton/
  - mkdir tango cdrom
- Run ‘make’ again
- Run 'source rootfs-path.env' for integration with the kernel source package.

The root file system (2): build it

- The root file system also needs to pull the runtime shared libraries from the toolchain.
  - In order for this to happen, you must define the 'SMP86XX_TOOLCHAIN_PATH' environment variable to point to the main directory of your toolchain.
  - If you have built the toolchain, sourcing 'toolchain-path.env' defines 'SMP86XX_TOOLCHAIN_PATH'.
  - Else, you can simply define it by hand.
- Run 'make' in the rootfs package main directory
- You now have a preliminary root file system for you to work with the kernel source package in the next step.

The kernel source (1): get the source

- Use the smp86xx_kernel_source_2.x.yyy.z.tar.bz2 package
- On the host:
  - `host:$ tar xjf smp86xx_kernel_source_2.x.yyy.z.tar.bz2`
  - `host:$ cd smp86xx_kernel_source_2.x.yyy.z`
- Run '`make`' with no argument to be reminded what targets are available.
- Run '`make kernel-source-<version>`' (where version is 2.4.30 or 2.6.15) to prepare the kernel source.
- Your pre-configured kernel source is ready under 'linux-2.4.30' or 'linux-2.6.15'.

The kernel source (2): build and install the modules

- You now have a regular Linux kernel source tree. Follow the usual steps to build the kernel:
  - `host:$ cd linux-<version>`
  - `host:$ make menuconfig //to configure the kernel & save`
  - `host:$ make dep  // only for kernel 2.4.x`
  - `host:$ make vmlinux // [*] modprobe modules purpose`
  - `host:$ make modules`
  - `host:$ make modules_install //put them in rootfs`
- After these steps, you have built the kernel modules and they are installed in the root file system package's customization directory.
- [*] This step, though it might not make much sense, is necessary in order for the kernel system map to be available at module installation time. This is a consequence of using an embedded system with a read-only file system.

The root file system (3): build it again

- The root file system now needs to get all kernel modules built from the last step in the kernel source package
- Run 'make' in the rootfs package main directory
- You now have a completed root file system with integrated kernel modules.

Kernel source (3): build the kernel image

- You now build the final kernel image with the correct ZBF header, after pulling the completed root file system from the rootfs package directory. The build system also produces a ROM file system image suitable for flashing, named “zbimage-linux-xrpc.”
- Go to the kernel source directory, smp86xx_kernel_source_2.7.xx.0/linux-2.X.YY/
- For kernel 2.4.30:
  - run 'make'
  - Kernel image ‘zbimage-linux-xrpc’ is in same directory
- For kernel 2.6.15:
  - run 'make zbimage-linux-xrpc’
  - Kernel image ‘zbimage-linux-xrpc’ is in ./arch/mips/boot/ directory

kernel source (4): commit kernel image to flash

- You can flash the ROMFS image to the parallel flash:
  - YAMON> load uu 0xb0100000
  - host:$ uuencode zbimage-linux-xrpc x > /dev/ttyS0
  - YAMON> pflash write 0x80000 0xb0100000 <received size>

Verify the kernel image(1)

- At the YAMON prompt, load the kernel image from its location on the flash to RAM. The xload and ZBF header take care of the gory details:
  YAMON> pflash probe
  YAMON> xrpc 0xac080090//after that will get
  message ‘xrpc succeeded’
  YAMON> load zbf 0xb3000000
- Pass any kernel boot parameter as the value to the a.linux_cmd Xenv key (see 'Documentation/kernel-parameters.txt' in the kernel source tree):
- Start the kernel with the YAMON go command.

Verify the kernel image(2)

- After the kernel boots up, enter 'root' at the login prompt and press return
- You are now logged in on the Linux system running on your standalone SMP8634!

The MRUA drivers

- **Untar the package and build the driver and sample applications:**
  - `host:$ tar xzf mrua_smp8634_2.x.yyy.z_dev.mips.tgz`
  - `cd mrua_smp8634_2.x.yyy.z_dev.mips`
  - `make // create MRUA.env`
  - `source MRUA.env`
  - `export UCLINUX_KERNEL=<path_to_your_mips_kernel>`
  - `make // see what's available`
  - `make kernel`
  - `make apps`
- **The kernel modules are now available under modules/mips-2.4.30 or modules/2.6.15**
- **The sample applications are now available under 'bin'**

Play HD video (1): system setup

- On your target system, network configuration:

  - ifconfig eth0 172.30.3.49 netmask 255.255.192.0
    - route add default gw 172.30.0.3
    - eth0 -Realtek
    - eth1 –internal
- For kernel 2.6 after build 142. Enable NFS because it was compiled as a module.
  - modprobe nfs
- Mount the remote NFS directory from your host:
  - mount -o nolock 172.30.2.48:/export/shared /mnt
  - export LD_LIBRARY_PATH=/mnt/mrua_SMP8634_2.7.xx.0_dev.mips/lib
  - cd /mnt/mrua_SMP8634_2.x.yyy.z_dev.mips

Play HD video (2): play!

- **For kernel 2.6.15 only**

  Make node:

  – **`mknod /dev/mum0 c 126 0`**

  – **`mknod /dev/em8xxx0 c 127 0`**

  Insert the drivers:

  – **`insmod modules/2.6.15/llad.ko`**

  – **`insmod modules/2.6.15/em8xxx.ko`**

- **For kernel 2.4.30 only**

  Insert the drivers:

  – **`insmod modules/mips-2.4.30/llad.o`**

  – **`insmod modules/mips-2.4.30/em8xxx.o`**

Play HD video (3): play!

Do the color bars test:

- `./bin/colorbars`

- **Play a video file:**
  - `./bin/play_video -pv 2hd /mnt/movie/moviefile.m2v`

MPEG-2 HD video is displayed on your TV!!!

Default output is to composite & S-Video

Annex A: tips and tricks

- In this section, we will see how to handle issues that one can run into, including:
  - how to find out xos version
  - how to upgrade xos
  - how to get a withhost board back to a bootable state
  - how to get a standalone board back to a bootable state

How to manipulate xos

- The key is the 'xrpc' or ‘ruaxrpc’ tool
- There are two locations within the MRUA package:
  - in a standalone package, ‘ruaxrpc' is under 'targettools' and runs on the target standalone board (Envision/Vantage SMP8634)
  - in a withhost package (Odyssey), ‘ruaxrpc' is under 'hosttools' and runs on the host system.
- ‘xrpc’ is on Yamon

How to obtain the xos version

- Running '`./ruaxrpc -v`' in SMP8634 environment or '`xrpc –v`' in Yamon gives the SHA-1 sum of the running xos image.
- The last three hex digits of the SHA-1 sum tells the xos version.
- For example, on an SMP8634:

```
$ ./ruaxrpc -v
[Output] xos SHA-1 = 366...d8500ca
```

The last three digits are '`0ca`' indicating version xosMca.

How to upgrade xos

- In the same directory as the 'xrpc' tool, there are two xos upgrade image:
  - `xosu-xosMba-8634_ES4_dev_0007.xload`
  - `xrpc_xload_xosu-xosMba-8634_ES4_dev.bin`
- Simply run

  ```
  $ xrpc -xload xosu-xosMba-8634_ES4_dev_0007.xload
  ```

  to upgrade your xos image. Alternatively, you can use the old format and run

  ```
  $ xrpc xrpc_xload_xosu-xosMba-8634_ES4_dev.bin
  ```
- Your standalone board will be reboot after xos is updated.

More xrpc fun

- The 'xrpc' tool has many more capabilities!
- See the README.xrpc file or the tool help (`xrpc -h`) for more information.

How to recover Yamon/Zboot

- It may happen that a wrong manipulation, or a hardware failure render a board not-bootable.
- If YAMON cannot boot, then the board must be handled through external means: JTAG. Please see JTAG section.

Preparing a ZB image

- A ZB image will be available on the developer web site, but you can also produce your own.
- For that, prepare the zboot, YAMON and kernel image as explained in this tutorial.
- You then want to concatenate these images, along with padding (use /dev/urandom or /dev/zero), so that the zboot image lies at the beginning of the ZB image, the YAMON image lies at offset given by the Xenv key x.boot0 the kernel image lies at offset given by the Xenv key x.boot1.
- To operate the concatenation, standard Unix tools such as 'dd' and '`cat`' are your friends.

Using the JTAG interface

- The only external mean to access a standalone board is through the JTAG interface.
- All of the SMP863x tools support the JTAG interface, so this method also applies to the OdysseySMP863x boards.
- The JTAG support files are distributed with the standalone MRUA SMP863x packages, in the 'targettools/jtag' directory.
- Please refer to the documentation in that directory, and to additional documentation on the developer web site about how to use the JTAG interface.

Annex B: board recovery guide

- This is a step-by-step guide to recover a board that does not boot to YAMON
- It requires a JTAG interface, along with the smp86xx_boot_loader package
- Board recovery starts at step <0>

Step <0> does XOS boot?

- Do not connect the JTAG interface yet.
- Does the UART show 'xos version' at power on? [hint: the SMP8634 always uses 115200 8N1, no flow control, no hw control]
  - yes: go to step <1>
  - no: this is a power issue or a UART connection issue, or a bad chip, or a chip that was not initialized (no XOS on serial flash).

Step <1> does monice run?

- Connect the JTAG interface
- Does monice run? It does if the following command displays the cache sizes:

```
$ monice -v4Kec -d <probe network name>:e -l
```

  - yes: go to step <2>
  - no:
    - check the JTAG presence and status, check the JTAG connector orientation, reset the probe
    - disable boot flash, change boot flash, short addr0 and addr1 pins of boot flash and reboot

Step <2> build recovery YAMON

- Prepare the boot loader build environment (see slide 14) and run

  make yamon-patched

  to prepare the YAMON source
- Edit 'build/yamon/bin/Makefile' and change

  USE_XENV to n

  USE_SIGBLOCK to n

  WITH_RESET to y
- Run make yamon to build the recovery image and save

  build/yamon/bin/reset-02.06-SIGMADESIGNS-01-<version>.elf

  build/yamon/bin/EL/yamon-02.06-SIGMADESIGNS-01-<version>_el.elf

  to a safe location.

Step <2> reset the board

- Force-reconfigure DRAM controller 0 to a close-to-working configuration:

  $ monice ...

  MON> ew 0xa003fffc =3

  MON> ew 0xa003fffc =2

- For boards with 128MB on DRAM0

  MON> ew 0xa0030000 =0xf34111ba

- For boards with 32MB or 64MB on DRAM0

  MON> ew 0xa0030000 =0xe34111ba

- For both types, continue with:

  MON> ew 0xa0030004 =0xa4444

  MON> ew 0xa003fffc =0

Step <2> upload recovery YAMON

- Still within monice, run:

  MON> ci

  MON> ew 0xa006f000 =0x60000

  MON> l yamon-02.06-SIGMADESIGNS-01-<version>_el.elf

  MON> l reset-02.06-SIGMADESIGNS-01-<version>.elf

  MON> g 0xbfc00000

- You should obtain the YAMON prompt.
  - yes: go to step <3>
  - no:
    - the DRAM chip is not soldered properly
    - you have uncommon config parameters
    - check PLL3 config and mux settings

Step <3> flash a valid boot image

- Create a valid zboot/xenv and a valid YAMON image as explained at the beginning of this tutorial.
- Copy both images to the directory you are running monice from, and checksum both images:

  $ cksum zboot2.bin

  $ cksum zbimage-yamon
- The cksum command reports the size and check sum of each image.

Step <3> flash a valid boot image

- Interrupt the 'g' command from the end of step <2>, then write both images to the board DRAM:

  MON> fr m zboot2.bin 0xb0100000

  MON> fr m zbimage-yamon 0xb0200000

  MON> g

- Switch back to the YAMON prompt and verify the cksum for each image:

  YAMON> cksum 0xb0100000 <zboot2.bin-size>

  YAMON> cksum 0xb0200000 <zbimage-yamon-size>

Step <3> flash a valid boot image

- You can then write both images at the correct flash offset through the usual YAMON commands (see slide 15 through 18):

  YAMON> pflash probe

  YAMON> pflash write 0 0xb0100000 <zboot2.bin-size>

  YAMON> pflash write 0x40000 0xb0200000 <zbimage-yamon-size>

- Finally, verify the checksum of the written images:

  YAMON> cksum 0x8c000000 <zboot2-image-size>

  YAMON> cksum 0x8c400000 <zbimage-yamon-size>

- Rebooting one last time should now correctly boot you to the YAMON prompt.

Thank you!
-and-
Happy Coding!