Getting ION to run on the AT91SAM9G20

Scott Burleigh and Paul Muri

22 December 2012

Introduction

This is a sort of cookbook for getting ION to run on the AT91SAMG20 system board, based on the ARM926EJ-S processor. It has been shown to work twice, but we offer ABSOLUTELY NO GUARANTEE that it will work for you; the supporting software changes and no two development environments are exactly alike. We do hope that it will at least provide some guidance as to the kinds of things you will need to do, but over time the details will undoubtedly change and this document probably will not keep up with them. Good luck!

Get the hardware

The ***target*** platform for this exercise is the AT91SAM9G20-EK evaluation kit from ATMEL. Download http://www.atmel.com/Images/doc6413.pdf to view the User's Guide for this board.

The ***host*** platform should be a Linux machine with a serial (RS-232) connector: for flashing new software onto the target machine there will need to be both USB and serial connections, with the serial connection being used to monitor target activity in a minicom window on the host. Fedora Core 14 (64 bits) is assumed here, as it is known to have been tested successfully with SAM-BA version 2.11 for Linux.

Get the software

On the host machine, create a top-level directory named (for example) "AT91SAM9G20" into which all software is downloaded.

SAM-BA

The target comes with a disk, but the software on the disk is all for a Windows development host. For the Linux host, it's necessary to download and install SAM-BA 2.11 for Linux from ATMEL; see link at the bottom of http://www.atmel.com/tools/SAM9G20-EK.aspx.

To install SAM-BA:

- Make sure Tcl, Tk, and Tclx are installed.
- Unzip the downloaded file.
- USB CDC Serial driver mount procedure:
    - Login with administrator rights

- Unload usbserial module if it is already running:

  ```
  # rmmod usbserial
  ```

- Load usbserial kernel module

  ```
  # modprobe usbserial vendor=0x03eb product=0x6124
  ```

- Verify that the USB connection is established

  ```
  # lsusb -d 03eb:6124
  Bus 004 Device 006: ID 03eb:6124 Atmel Corp. at91sam SAMBA boot loader
  ```

- Know which USB connection is established

  ```
  # dmesg
  ...
  kernel: usb 4-2: new full speed USB device using uhci_hcd and address 5
  kernel: usb 4-2: configuration #1 chosen from 1 choice
  kernel: usbserial_generic 4-2:1.0: generic converter detected
  kernel: usbserial_generic: probe of 4-2:1.0 failed with error -5
  kernel: usbserial_generic 4-2:1.1: generic converter detected
  kernel: usb 4-2: generic converter now attached to ttyUSBx
  ```

  - This signifies that you will have to use /dev/ttyUSBx to connect to your board.

- Assuming the machine is 64-bit Fedora, must install the relevant 32-bit libraries (because the SAM-BA executable is pre-built for 32-bit architectures):
  - # yum install glibc libstdc++ libX11

    This upates the 64-bit libraries in order to have the same version as the 32-bit libraries. If you don't do this, Fedora may complain that you have multilib versions.

  - # yum install glibc.i686 libstdc++.i686 libX11.i686
  - # yum install libXScrnSaver.686
  - Must also install 32-bit ("686") versions of libXft and zlib. This will require some hunting through the Internet. rpm.pbone.net may have libXft. For zlib, search on "zlib-1.2.5-2.fc14.i686.rpm".
- Give sam-ba execute permission if needed:
  - chmod +x sam-ba
- Add yourself into dialout group: edit the /etc/group file and add your username at the end of the line starting with 'dialout'.

  ```
  dialout:x:18:myusername
  ```

- Also add **$(HOME)/AT91SAM9G20/sam-ba_cdc_cdc_linux** to your $PATH.
- Logout and login.

- Connect the AT91SAM9G20 board with jumpers J33 and J34 open so that the operating system can detect the board and update /dev.
- Create a symlink on /dev/ttyACM0:
  - # ln -s /dev/ttyACM0 /dev/ttyUSB0

buildroot

The development toolchain is based on the "buildroot" tool and the uClibc library. Get the latest version of buildroot from http://buildroot.uclibc.org/download.html and install it as follows:

Prerequisites

Host build system should be a Linux system with necessary software installed. List of required software on the host machine:

GNU C/C++
GNU make
wget
sed
flex
bison
m4
patch
gettext
libtool
texinfo
autoconf
automake
ncurses library (development install)
zlib library (development install)
libacl library (development install)
lzo2 library (development install)
glib library (development install) - if some packages selected

"development install" means you need the "install" variant of the package as well:

ncurses ncurses-devel
zlib zlib-devel
libacl libacl-devel
lzo lzo-devel

Using yum install or apt-get install may require you have root privilege.

For example, on Fedora you can check 1 through 12 by xxx --version, and 13 through 16 by rpm -q xxx:

```
$ rpm -q lzo
```

```
lzo-2.02-2.fc6
```

To install lzo packages under Fedora, you can use:

```
# yum install lzo lzo-devel
```

Some old distributions can only install lzo-1.xx by means of yum, so you need to download an lzo2 package by yourself from http://www.oberhumer.com/opensource/lzo/download/ and install it by hand. For lzo2, there should be a lib file liblzo2.so in /usr/lib/, and a folder lzo/ in /usr/include/.

Build the toolchain

(No need to be root for this.)

```
$ make menuconfig                    [Exit and Save]

$ make 2>&1 | tee at91sam9.log
```

This is the same procedure that will later be used to create new target system builds, but on the first time through it will also create the toolchain.

Notes on menuconfig selections:

- Target architecture "arm (little endian)".
- Choose Target architecture variant "arm926t".
- Target ABI : "EABI".
- Under "Build options", all the defaults should be kept except for the simultaneous job number; set this to 1.
- "Toolchain": select gdb for the target and "Enable large file support." Additionally, choose the kernel headers to match the kernel that will be installed later: select "Kernel Headers (Linux 2.6 manually specified version)", and under the "linux version" field, specify 2.6.30.2. Choose uClibc version 0.9.33.2 and gcc version 4.7.0.
- Default "System Configuration" should be fine.
- Some useful packages: "i2c-tools" and "mtd/jffs2 utilities" under "Hardware Handling". "openssl" under "Libraries/Crypto". "dropbear", "pppd", and "rsync" under "Networking Applications".
- Select a "JFFS2 Root File System" under "Filesystem Images", and don't worry about any options under "Bootloaders" or "Kernel" for initial development. The kernel is really only necessary if you need to modify any BSP or kernel module options.

If the build halts because of missing host system packages that are needed by buildroot, install the missing package (should be output by buildroot) and you can continue the build process.

The toolchain has now been built. To confirm, check for the rootfs.jffs2 file in the following directory:

$(HOME)/AT91SAM9G20/buildroot-xxxx.yy/output/images

Add the following directory to your $PATH:

$(HOME)/AT91SAM9G20/buildroot-xxxx.yy/output/staging/usr/bin

(Re-source your .bash_profile, or whatever, to pick up this change.)

Tcl script

The easiest way to flash new builds onto the AT91SAM9G20 board is to use a Tcl script, and the easiest way to develop that script is to tweak an existing script. For this purpose:

- Download and unzip the Demo Archive at ftp://www.at91.com/pub/demo/linux4sam_1.7/linux4sam-buildroot-at91sam9g20ek.zip. This will be the directory from which new images will be flashed.
    - cd linux4sam-buildroot-at91sam9g20ek
    - cp ~/AT91SAM9G20/buildroot-2010.02/output/images/rootfs.arm.jffs2 .
- Modify the script named **at91sam9g20ek_2mmc_demo_linux_nandflash.tcl**:
    - Change the "set kernelfile" command to reference "uImage".
    - Change the "set rootfsFile" command to reference "rootfs.arm.jffs2".

Linux kernel

The kernel file referenced in the above revisions to the Tcl script is a kernel that is built by downloading http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.30.2.tar.bz2 into buildroot-2010.02/dl, unpacking it with "tar -xf linux-2.6.30.2.tar.bz2", moving the unpacked directory linux-2.6.30.2 into buildroot-2010.02/output/build, cd'ing into that directory, and then proceeding as follows:

(AT91 kernel patch)

- wget http://maxim.org.za/AT91RM9200/2.6/2.6.30-at91.patch.gz
- zcat 2.6.30-at91.patch.gz | patch -p1
- wget ftp://www.at91.com/pub/linux/2.6.30-at91/2.6.30-at91-exp.4.tar.gz

(Experimental patches)

- wget ftp://www.at91.com/pub/linux/2.6.30-at91/2.6.30-at91-exp.4.tar.gz
- tar -xf 2.6.30-at91-exp.4.tar.gz
- for p in 2.6.30-at91-exp.4/*; do patch -p1 < $p ; done

(Kernel Config)

- wget ftp://www.at91.com/pub/linux/2.6.30-at91/at91sam9g20ek_2mmc_defconfig
- cp at91sam9g20ek_2mmc_defconfig .config
- make ARCH=arm menuconfig
    - Just Exit menuconfig immediately and Save, to create .config file for the build.
- make ARCH=arm CROSS_COMPILE=/buildroot-2013.02/output/host/usr/bin/arm-linux-

Then copy the new kernel to the flashing directory. Convert the kernel format from zImage to uImage:

- cd arch/arm/boot
- cp zImage ~/AT91SAM9G20/linux4sam-buildroot-at91sam9g20ek
- sudo yum install uboot-tools
- mkimage -A arm -O linux -C none -T kernel -a 20008000 -e 20008000 -n linux-2.6.30.2 -d zImage uImage

Build ION for AT91SAM9G20

This is the easy part.

1. Download the ion-3.2.0.tar.gz file from SourceForge into buildroot-xxxx.yy/dl, unpack it with "tar xzf ion-3.2.0.tar.gz", move the unpacked source directory to buildroot-xxxx.yy/output/build, and cd to that directory.
2. To prevent the file from being re-downloaded and re-extracted, the following files must exist in the directory (they are shipped with all recent versions of ION):
    a. .stamp_downloaded
    b. .stamp_extracted
    c. .stamp_patched
    d. .stamp_configured
    e. .stamp_staging_installed
3. Create a new "ion" subdirectory under buildroot-xxxx.yy/package and copy the arch-uClibc files Config.in and ion.mk into that subdirectory.
4. Add the following line to the "Networking applications" menu of buildroot-xxxx.yy/package/Config.in, in alphabetical sequence:

    source "package/ion/Config.in"

5. In buildroot-xxxx.yy, "make menuconfig" and select the new "ion" package in the Networking menu, then Exit and Save the configuration, then "make 2>&1 | tee at91sam9.log". This produces a new rootfs.arm.jffs2 file.

Install the build

Configure minicom for operating on the AT91SAMG20-EK board: `minicom -s`

The minicom serial communication parameters are 115200 8-N-1 :

Baud rate	115200
Data	8 bits
Parity	None
Stop	1 bit
Hardware Flow control	None

Connect a USB cable from the target board to the host. Make sure the serial cable is connected from the DGBU connector on the target to the RS-232 connector on the host.

Start minicom on the host.

Open the J33 (serial data flash) and J34 (NaND flash) jumpers. Also make sure the J7 jumper is open.

Power up the target board. "RomBOOT >" should appear in the minicom display.

Start SAM-BA and connect it to the AT91SAM9G20 via ttyACM0.

Close the J34 and J33 jumpers.

Place the bootstrapFile, ubootFile, kernelFile, and rootfsFile in the sam-ba_cdc_cdc_linux directory

```
set bootstrapFile    "nandflash_at91sam9g20ek.bin"
set ubootFile        "u-boot-1.3.4-exp.3-at91sam9g20ek_2mmc-nandflash.bin"
set kernelFile       "uImage"
set rootfsFile       "rootfs.arm.jffs2"
```

Using the "Script File" pull-down, choose "Execute Script File" and navigate to where you can select the **at91sam9g20ek_2mmc_demo_linux_nandflash.tcl** script for execution. Click Open and wait for the script to finish running.

Exit from SAM-BA.

Power down the board and remove the USB cable.

To force a boot from the nandflash, open jumper J33 (leaving jumper J34 closed).

Power up the board to boot the new build.

Log in as root and verify that ionadmin is in the /opt/bin/ directory.

Setup your PATH on the target board:

- `PATH=$PATH:/opt/bin`
- `export PATH`
- `LD_LIBRARY_PATH=/opt/lin`
- `export LD_LIBRARY_PATH`

Annex – "ion" package directory

Config.in file:

```
config BR2_PACKAGE_ION
       bool "ION implementation of Delay-Tolerant Networking"
       help
```

```
ION is an implementation of DTN that is designed to run
efficiently in embedded systems, including robotic spacecraft.
It includes full implementations of Contact Graph Routing,
Bundle Streaming Service, and the Licklider Transmission
Protocol.  It also includes implementations of the CCSDS
standard protocols for file transfer (CFDP) and messaging
middleware (AMS).

https://ion.ocp.ohiou.edu/
```

ion.mk file:

```
#############################################################
#
# ion
#
#############################################################

ION_VERSION_MAJOR=3
ION_VERSION_MINOR=0.1
ION_VERSION:=$(ION_VERSION_MAJOR).$(ION_VERSION_MINOR)
ION_SOURCE:=ion-$(ION_VERSION).tar.gz
#ION_SITE:=http://sourceforge.net/projects/ion-dtn/files

define ION_BUILD_CMDS
 mkdir -p $(TARGET_DIR)/opt
 mkdir -p $(TARGET_DIR)/opt/bin
 mkdir -p $(TARGET_DIR)/opt/lib
 mkdir -p $(TARGET_DIR)/opt/include
 mkdir -p $(TARGET_DIR)/opt/man
 mkdir -p $(TARGET_DIR)/opt/man/man1
 mkdir -p $(TARGET_DIR)/opt/man/man3
 mkdir -p $(TARGET_DIR)/opt/man/man5

 $(MAKE) TCC="$(TARGET_CC)" TLD="$(TARGET_LD)" ROOT="$(TARGET_DIR)/opt" -C
$(@D)/ici/arm-uClibc all
 $(MAKE) TCC="$(TARGET_CC)" TLD="$(TARGET_LD)" ROOT="$(TARGET_DIR)/opt" -C
$(@D)/ici/arm-uClibc install
 $(MAKE) TCC="$(TARGET_CC)" TLD="$(TARGET_LD)" ROOT="$(TARGET_DIR)/opt" -C
$(@D)/dgr/arm-uClibc all
 $(MAKE) TCC="$(TARGET_CC)" TLD="$(TARGET_LD)" ROOT="$(TARGET_DIR)/opt" -C
$(@D)/dgr/arm-uClibc install
 $(MAKE) TCC="$(TARGET_CC)" TLD="$(TARGET_LD)" ROOT="$(TARGET_DIR)/opt" -C
$(@D)/ltp/arm-uClibc all
 $(MAKE) TCC="$(TARGET_CC)" TLD="$(TARGET_LD)" ROOT="$(TARGET_DIR)/opt" -C
$(@D)/ltp/arm-uClibc install
 $(MAKE) TCC="$(TARGET_CC)" TLD="$(TARGET_LD)" ROOT="$(TARGET_DIR)/opt" -C
$(@D)/bp/arm-uClibc all
 $(MAKE) TCC="$(TARGET_CC)" TLD="$(TARGET_LD)" ROOT="$(TARGET_DIR)/opt" -C
$(@D)/bp/arm-uClibc install
 $(MAKE) TCC="$(TARGET_CC)" TLD="$(TARGET_LD)" ROOT="$(TARGET_DIR)/opt" -C
$(@D)/cfdp/arm-uClibc all
 $(MAKE) TCC="$(TARGET_CC)" TLD="$(TARGET_LD)" ROOT="$(TARGET_DIR)/opt" -C
$(@D)/cfdp/arm-uClibc install
```

```
 $(MAKE) TCC="$(TARGET_CC)" TLD="$(TARGET_LD)" ROOT="$(TARGET_DIR)/opt" -C
$(@D)/ams/arm-uClibc all
 $(MAKE) TCC="$(TARGET_CC)" TLD="$(TARGET_LD)" ROOT="$(TARGET_DIR)/opt" -C
$(@D)/ams/arm-uClibc install
 $(MAKE) TCC="$(TARGET_CC)" TLD="$(TARGET_LD)" ROOT="$(TARGET_DIR)/opt" -C
$(@D)/bss/arm-uClibc all
 $(MAKE) TCC="$(TARGET_CC)" TLD="$(TARGET_LD)" ROOT="$(TARGET_DIR)/opt" -C
$(@D)/bss/arm-uClibc install
endef

define ION_INSTALL_TARGET_CMDS
 $(MAKE) TCC="$(TARGET_CC)" TLD="$(TARGET_LD)" ROOT="$(TARGET_DIR)/opt" -C
$(@D)/ici/arm-uClibc install
 $(MAKE) TCC="$(TARGET_CC)" TLD="$(TARGET_LD)" ROOT="$(TARGET_DIR)/opt" -C
$(@D)/dgr/arm-uClibc install
 $(MAKE) TCC="$(TARGET_CC)" TLD="$(TARGET_LD)" ROOT="$(TARGET_DIR)/opt" -C
$(@D)/ltp/arm-uClibc install
 $(MAKE) TCC="$(TARGET_CC)" TLD="$(TARGET_LD)" ROOT="$(TARGET_DIR)/opt" -C
$(@D)/bp/arm-uClibc install
 $(MAKE) TCC="$(TARGET_CC)" TLD="$(TARGET_LD)" ROOT="$(TARGET_DIR)/opt" -C
$(@D)/cfdp/arm-uClibc install
 $(MAKE) TCC="$(TARGET_CC)" TLD="$(TARGET_LD)" ROOT="$(TARGET_DIR)/opt" -C
$(@D)/ams/arm-uClibc install
 $(MAKE) TCC="$(TARGET_CC)" TLD="$(TARGET_LD)" ROOT="$(TARGET_DIR)/opt" -C
$(@D)/bss/arm-uClibc install
endef

define ION_CLEAN_CMDS
 $(MAKE) TCC="$(TARGET_CC)" TLD="$(TARGET_LD)" ROOT="$(TARGET_DIR)/opt" -C
$(@D)/ici/arm-uClibc clean
 $(MAKE) TCC="$(TARGET_CC)" TLD="$(TARGET_LD)" ROOT="$(TARGET_DIR)/opt" -C
$(@D)/dgr/arm-uClibc clean
 $(MAKE) TCC="$(TARGET_CC)" TLD="$(TARGET_LD)" ROOT="$(TARGET_DIR)/opt" -C
$(@D)/ltp/arm-uClibc clean
 $(MAKE) TCC="$(TARGET_CC)" TLD="$(TARGET_LD)" ROOT="$(TARGET_DIR)/opt" -C
$(@D)/bp/arm-uClibc clean
 $(MAKE) TCC="$(TARGET_CC)" TLD="$(TARGET_LD)" ROOT="$(TARGET_DIR)/opt" -C
$(@D)/cfdp/arm-uClibc clean
 $(MAKE) TCC="$(TARGET_CC)" TLD="$(TARGET_LD)" ROOT="$(TARGET_DIR)/opt" -C
$(@D)/ams/arm-uClibc clean
 $(MAKE) TCC="$(TARGET_CC)" TLD="$(TARGET_LD)" ROOT="$(TARGET_DIR)/opt" -C
$(@D)/bss/arm-uClibc clean
endef

$(eval $(call GENTARGETS,package,ion))
```