Prerequisites#

- QEMU with the Linaro patches (https://launchpad.net/qemu-linaro): The Beagle board emulation has not yet been merged into the official QEMU distribution. Note that the Ubuntu 11.04 QEMU package is already patched and includes support for the Beagle board.
- Momentics for QNX 6.5.0
- The QNX Beagle BSP for QNX 6.5.0 (http://community.qnx.com/sf/wiki/do/viewPage/projects.bsp/wiki/Bspdown_ti_omap_3530_evm)
- X-loader and U-boot for the Beagle board (QEMUAsAnOMAP3530BeagleboardTarget/beagle-nand.tar.gz)

IFS#

To build the IFS, simply type `make` at the top level directory of the unzipped BSP archive.

PMIC Problem#

There is currently a problem with the IFS's PMIC driver on the emulated Beagle board. To disable the PMIC driver, comment the relevant lines in `$BSP_DIR/src/hardware/startup/boards/omap3530/beagle.build`:

```
#display_msg Configure power management chip
#pmic_tw4030_cfg
...
#display_msg Starting USB EHCI Host and OTG host driver...
#io-usb -dehci-omap3 ioport=0x48064800,irq=77  -domap3530-mg ioport=0x480ab000,irq=92
#waitfor /dev/io-usb/io-usb 4
```

NAND Drive#

The board will be booting and running off an emulated 256 MB NAND drive. This drive is composed of 2K pages, each followed by a 64 byte checksum value. It is thus insufficient to create a zeroed 256 MB file to use as an emulated file. The attached archive contains a script, `mknand.sh`, that creates the drive file and loads the different components (X-loader, U-boot, IFS) onto it. Whenever the drive file is modified, a checksum calculator should be executed to fix the checksum values. The calculator's source code is included in the archive as `nand_ecc.c`.

When `mknand.sh` is first executed, it will create the NAND drive file and load X-loader and U-boot into their appropriate locations. The script should also be given the path to the IFS image, which, if loaded from the BSP, is `$BSP_DIR/images/ifs-omap350-beagle.bin`. Subsequently, any change to the IFS requires `mknand.sh` to be invoked again in order to update the drive, followed by `ecc_nand` to compute the new checksum.

NOTE: the `mknand.sh` script in 'beagle-nand.tar.gz' will silently truncate your IFS if it exceeds the 4M allocated for it. And if you use the build script from the BSP, it will. The separately attached `mknand.sh` script has been updated to warn you if your image is truncated.

The `mknand.sh` script is invoked with the following command line:

```
$ ./mknand.sh DRIVE_FILE IFS_PATH
```

Where `DRIVE_FILE` is a name for the NAND image and `IFS_PATH` is the full path to the QNX Beagle IFS file. The checksum calculator should be executed as follows:

```
$ ./nand_ecc DRIVE_FILE 0x0 0xe80000
```

Where `DRIVE_FILE` is the file created by `mknand.sh`.

Makefile#

A Makefile included in the archive automates the steps above. Simply set the top-level directories as required, and type "make" to create the drive.

Running QEMU#

QEMU should be executed as follows:

```
$ qemu-system-arm -M beagle -m 256 -mtdblock DRIVE_FILE -nographic
```

Where `DRIVE_FILE` is the file created by `mknand.sh`. To emulate a serial connection to the board, append -serial pty to the command line (other serial emulators are available, such as a TCP socket).

Booting QNX#

By default, the image displays the U-boot menu. You need to manually stop the boot process (press any key), and use the following command to boot QNX:

```
$ nand read 0x80100000 0x280000 0x400000; go 0x80100000
```

This boot command can be written into the NAND drive and executed automatically on subsequent boots:

```
$ setenv bootcmd 'nand read 0x80100000 0x280000 0x400000; go 0x80100000'
$ saveenv
```

Kernel Debugger#

Building#

To use the kernel debugger, apply the following changes to the `beagle.build` file:

- Change the `.0.` part of the startup command to `.9600.` (decent baud rate for the debugger):

```
startup-omap3530 -L 0x87E00000,0x200000 -v -D8250.0x49020000^2.9600.48000000.16 beagle
```

- Add the debugger to the bootstrap section, following the startup command (the `-K` for an initial breakpoint is optional):

```
gdb_kdebug -D0 -v -K
```

- Add a `keeplinked` directive to the `procnto` line:

```
[+keeplinked uid=0 gid=0 perms=0700] PATH=:/proc/boot:/sbin:/bin:/usr/sbin:/usr/bin LD_LIBRARY_PATH=:/proc/boot:/li
```

- Rebuild the image

Debugging#

Run QEMU with a serial emulator (such as PTY). If you are using a serial terminal, such as minicom, it **must** be disconnected before the debugger is attached. Run GDB with the following command-line:

```
$ ntoarm-gdb SYM_FILE
```

Where `SYM_FILE` is the file created by the `keeplinked` directive, in the image directory (`$BFS_DIR/images`). Then issue the GDB commands:

```
(gdb) set remotebaud 9600
(gdb) target remote DEVICE
```

Where `DEVICE` is the serial device connected to with QEMU's `-serial` option. If using a PTY, QEMU will report the device path (e.g., `/dev/pts/5`) when it starts.