Table of Contents

CMEM

Introduction

The Linux C library user mode `malloc()` function allocates memory scattered in pages (4096 bytes each) across the physical memory. As the Joule core on Davinci does not have an MMU to compensate for this scattering, the DSP needs to work with contiguous buffers. This means `malloc()` cannot be used for allocating buffers to be used by the DSP for processing.

The CMEM module is an ARM side module which allocates contiguous memory and maps it to user space. It has a kernel mode as well as a user mode component. It currently only tested on Davinci, but have no direct dependencies on the chip. It requires Linux kernel 2.6.x.

Downloading and compiling CMEM for Davinci

Make sure you have the Montavista Pro 4.0 toolchain in your path (`arm_v5t_le-gcc` etc.), **as well** as the tool chain you used to build your kernel (if different than the Montavista Linux 4.0 LSP, which shouldn't be the case once we have a real Montavista drop). This because the CMEM kernel module is built using the kernel's own build system which will invoke this compiler.

Edit the file `Rules.make` in the CMEM top level directory to reflect your configuration. The options in this file are well documented, and the file should be self explanatory.

When `Rules.make` has been edited execute `make` to build both the kernel and user space components, followed by a `make install` to put the resulting binaries on your target filesystem.

If you want to generate the API documentation type `make docs` and the documentation will be generated in HTML under the `docs` directory.

Using CMEM on a target

You first need to insert the cmemk kernel module. After starting your target, make sure you are in the directory into where you installed the `cmemk.ko` kernel module, and execute (for example):

```
/sbin/insmod cmemk.ko phys_start=0x87800000 phys_end=0x88000000
pools=4x5000,3x10000
```

This will create 2 pools. One will have 4 buffers of size 5000 bytes, and one will have 3 buffers of 10000 bytes each. Note that these are the requested sizes of buffers which might differ from the actual page aligned allocated size (which will always be >= the requested size). The exact size allocated and other information about the pools can be inspected from the file `/proc/cmem`.

The memory pools will be located between the physical addresses `phys_start` and `phys_end` (an 8MB chunk in this example). Note that you need enough memory here to fit all your pools, and that your *mem=xM* kernel boot parameter must reflect this memory configuration.

If you set the configuration parameter to `USE_UDEV` you should be good to go. If you did not (perhaps you don't have udev support in your kernel), you need to create `/dev/cmem` manually using the `mknod` command (see `man mknod`). You need to know the "major number" assigned to cmem when it was inserted as this is done dynamically. Either read it from `/proc/devices`, or you can type `dmesg` to get the kernel output, in which the debug version of the kernel module outputs which major number was allocated. Suppose the allocated major number is 254, then you would execute `mknod /dev/cmem c 254 0` to create the device file.

Now you can either write your own program or execute the `apitest` application (the `apitestd` application is the same application but with debugging information enabled). The `apitest` application allocates two buffers of the same size (size given in bytes as a parameter on the command line) and fills them with the hexadecimal patterns `0xbeef` and `0xfeeb`. If you want to verify the application from the DSP just open up CCS on the DSP and look at the physical addresses for the buffers given by the application and you should see these patterns (the DSP has to be unlocked for CCS to be able to connect).