Noob Guide

BBB - Bare Metal Core

First, download a demo or mother zipfile and look at Makefile and memmap.lds. You will need to run this Makefile with "make".

I code on a Windows :-(laptop, so I use the GNUwin32 :-) "make" utility from a DOS prompt. Cygwin sucks and Linaro is worse. Also, Eclipse is a waste of time. I use Notepad++ for editing and build with "make". The load script memmap.lds tells Makefile how to build the image. Do not touch them until you know what you are doing. "Embedded Programming with the GNU Toolchain" by Vijay Kumar at bravegnu.org will teach you the fundamentals of image building. But skip all this for now.

The Makefile compiler/assembler/linker uses the arm-none-eabi toolchain. So you must install that from launchpad.net/gcc-arm-embedded/+download

Once you build the image successfully (no mean feat), you will need to copy it to an MMCard or use X-modem for UART booting. Very nebulous procedures for that are on the TI website and beagleboard forums, so follow Appendix 1 below. DieHards can use a jtag cable (XDS100V2 $70) to load/boot from the command prompt with loadti. I tested and documented all this and I am still boggled. Do not blame me for inconsistencies. Free does not imply easy. The zipfile contains a premade bootable image for you doubting Thomases.

You will need a terminal emulator to talk to your fledgling bare-metal core. Terminal emulators minicom (for Linux) or ExtraPuTTY (for Windows). If the Windows USB-Serial driver fails to connect, you must roll back the driver to PL2303_Prolific_Vista_332102.

Get the AM335x tech ref manual called spruh73x.pdf and ARM C/Assembly language references etc... from ARM and TI and GNU. "The ARM System Developer's Guide" by Sloss et al will teach you basic ARM architecture.

You must get a grip on all this. It is hard at first, but it gets easy because it all makes sense and builds upon a structure. I provide the all the build files and documented driver code with tested demos. That should save you about a year.

If you are a complete noob to ARM Assembly, get a Raspberry Pi :-(*) and do the youtube tutorial series on Assembly programming by "Computer Science Videos". It will give you a taste of the raw power of bare metal. The Pi is not truly open-source and their docs suck.

If you do not like Assembly, have no fear. You can skip all this and use the drivers from within a C main program, as in the demos.

happy hacking.....dd

Appendix 1
Preparing a Bootable MMC

Follow procedure in pdf: AM335X StarterWare Booting And Flashing

Windows Instructions for BOOTABLE MMC formatting:
MUST format with: HP USB Disk Storage Format Tool v2.0.6
Other formatters (from SDcard Corp) do not work. Possibly because
their default sector size is not 4096. This is an uncertain area.

Linux Instructions for BOOTABLE MMC formatting:
NB: for bbb boot-ability: heads=255, sects=63 & FAT32 bootable partition 1
parted /dev/sda1
print, ... , help
align-check 1
set 1 boot on
unit chs print (cyls, heads, sects)
quit
mkfs.vfat -n <vol_name> -F 32 /dev/sda1
then remove media & reinsert. it should auto mount.
==========
After compiling/linking your application don't forget to run:

$(CROSSCOMPILE)objcopy rts.elf app -O binary

This reformats the image for UART or MMC booting. It is in Makefile.
==========
For MMC booting you must prepend the image with a small header
to indicate load address and size with the ti image program:
tiimage 0x80000000 NONE app my_app

Rename "my_app" to "app"
Then copy the files named "app" and "MLO" to the MMCard root directory.

The MLO bootloader must be the version for MMC booting, not the UART
version. Both versions are provided pre-built. Do not monkey with them.
==========
If you mod/rebuild the MLO, the bootloader must be prepended with a header also:
tiimage 0x402F0400 MMCSD boot.bin MLO
==========
Blurb from AM335X StarterWare Booting And Flashing p8:
Stage 1: On reset, the ROM code loads the MLO bootloader by reading the
file "MLO" from the MMCard and copying it to the address embedded
on its header, 0x402F0400, then execution jumps to that address.
Stage 2: The MLO bootloader loads the application by reading the file
named "app" from the MMCard and copying it to the address embedded on
its header, 0x80000000. Then execution jumps to that address.
Stage 3: The application executes.