1 Introduction

Be it for performance, complexity or ability to audit, at times embedded developers will find themselves in a situation where running a bare metal application directly on OSD335x without Linux is needed. One common scenario for this is in time sensitive or real-time applications when better control is needed over every aspect of program execution. Another scenario would be if an application developer felt the need to eliminate the OS on the processor and of any overhead associated with it. However, Linux does provide a number of benefits that can aid in program development, so careful evaluation should be completed when determining whether to create a bare metal application. Regardless, the OSD335x Family of devices can run bare metal applications easily. This application note will show you how to load an application from an external SD Card (or eMMC or TFTP) to the DDR3 of the OSD335x and run it.

For this application note, you will need a Host Computer running Ubuntu and an OSD335x Family based development or prototype board, such as the Beagleboard.org® BeagleBone® Black Wireless, PocketBeagle® or OSD3358-SM-RED board. This development board should be running a Debian BeagleBone® image (http://beagleboard.org/latest-images) in order to leverage the U-Boot bootloader to load your bare metal application (the Linux portion of the image will not be used and may be removed).

Table of Contents

2 Revision History

Revision Number	Revision Date	Changes	Author
1	3/26/2018	Initial Release	Eshtaartha Basu

3 Installing StarterWare for AM335x

First, we will need to install the development environment for creating and building a bare metal application on the AM335x, StarterWare.

NOTE: All instructions given in this article were tested using a host machine running Ubuntu 16.04 and the OSD3358-SM-RED Reference, Evaluation, Development Board for the OSD335x-SM SiP Products running BeagleBone Debian 9.3 (U-Boot version: U-Boot 2017.01-00006-gb2bbabfe41)

You can download StarterWare installer for AM335x (v2.00.xx.xx) and BeagleBone Black patch (*Starterware_BBB_support*) from http://www.ti.com/tool/starterware-sitara. You will be asked to create a TI account and enter contact details to download all the files.

1. On the host Ubuntu Machine, open a Terminal (*Ctrl + Alt + T* on the keyboard)
2. Create a directory named bareMetal to hold the StarterWare files:

```
$ mkdir ~/bareMetal
```

3. Move the downloaded StarterWare installer file to the bareMetal directory:

```
$ mv ~/Downloads/AM335x_Starterware_02_00_01_01_Setup.bin ~/bareMetal
```

4. Make the StarterWare installer executable:

```
$ cd ~/bareMetal
$ chmod +x AM335X_StarterWare_02_00_01_01_Setup.bin
```

5. Install the dependencies to install StarterWare:

```
$ sudo apt-get update
$ sudo apt-get install zlib1g-dev libncurses5-dev
```

6. Run the installer:

```
$ ./AM335X_StarterWare_02_00_01_01_Setup.bin
```

7. During the installation, change the install path to *~/bareMetal/AM335X_StarterWare_02_00_01_01* and Installation Type to *Typical*.
8. Given we are running our bare metal application on a variant of the BeagleBone Black, we need to install the BeagleBone Black patch. To do this simply move the downloaded *StarterWare_BBB_support.tar.gz* to *AM335X_StarterWare_02_00_01_01* folder and extract it:

```
$ mv ~/Downloads/StarterWare_BBB_support.tar.gz
~/bareMetal/AM335X_StarterWare_02_00_01_01
$ cd ~/bareMetal/AM335X_StarterWare_02_00_01_01
$ tar xvfz StarterWare_BBB_support.tar.gz
```

At this point, StarterWare should be successfully installed on your host Ubuntu system.

4 Installing Linaro GCC Compiler

Next, we need to install a cross compiler that will allow us to build AM335x applications on our host Ubuntu system.

Download the Linux Installation Tarball (version: 4.7-2012q4) from https://launchpad.net/gcc-arm-embedded/4.7/4.7-2012-q4-major There are always improvements being made to GCC, so while a specific version is reference and tested in this app note, this procedure should be similar for future versions of the compiler.

1. Make a folder for compiler and move the downloaded file into it:

```
$ mkdir ~/bareMetal/linaro-gcc
$ mv ~/Downloads/gcc-arm-none-eabi-4_7-2012q4-20121208-linux.tar.bz2
~/bareMetal/linaro-gcc
```

2. Extract the compiler:

```
$ cd ~/bareMetal/linaro-gcc
$ tar xvfj gcc-arm-none-eabi-4_7-2012q4-20121208-linux.tar.bz2
```

3. Add the compiler location to StarterWare Makefile using your favorite editor (we used gedit):

```
$ gedit ~/bareMetal/AM335X_StarterWare_02_00_01_01/build/armv7a/gcc/makedefs
```

4. Modify the “PREFIX” variable with the following:

```
ifndef PREFIX
PREFIX=${LIB_PATH}/bin/arm-none-eabi-
endif
```

5. Setup path to toolchain:

```
$ export LIB_PATH=${HOME}/bareMetal/linaro-gcc/gcc-arm-none-eabi-4_7-2012q4
```

We now have StarterWare installed and a cross compiler that will allow us to build the StarterWare code for the AM335x.

5 Building StarterWare Example

Now, we will use the tools from Sections 3 and 4 to build a StarterWare application.

For the StarterWare examples, we will disable the MMU, instruction and data caches, and some exception handling in the initialization assembly file (*init.S* for gcc) to avoid any unexpected behavior. Depending on your application, you may need to enable some or all of these features. If you do, then please update the initialization file appropriately.

1. Open *init.S* using your favorite editor (again we use gedit):

```
$ gedit ~/bareMetal/AM335X_StarterWare_02_00_01_01/system_config/armv7a/gcc/init.S
```

2. Add the following after "Entry:" at line ~88 ("Entry" is the Reset handler in StarterWare. It is used to set the stack pointers and initialize hardware registers before "main()" function is called and is the best place to make this modification):

```
SUB r0, r0, r0
MCR   p15, 0, r0, c1, c0, 0
```

To learn more about the above two lines of assembly instructions, refer:
http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0344h/BCGFFBFD.html
http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0344f/Bgbciiaf.html

3. Save the file and close it.

4. Build StarterWare demo examples:

```
$ cd
~/bareMetal/AM335X_StarterWare_02_00_01_01/build/armv7a/gcc/am335x/beaglebone
$ make
```

5. Once the build process is complete, check if the build was successful by navigating to:

```
$ ls -lA
~/bareMetal/AM335X_StarterWare_02_00_01_01/binary/armv7a/gcc/am335x/beaglebone/
gpio/Release
```

You should be able to see *gpioLedBlink.bin* file.

At this point, you now have a binary application file that can be loaded into the DDR memory of the OSD335x / OSD335x-SM device and run. For your bare metal application, you should copy and extend the structure and build process defined in StarterWare. The build process outlined above should be similar for your application.

6 Running Demo Application on OSD335x using SD Card or eMMC

Next, we need to run the binary application that was built in Section 5 to the OSD335x / OSD335x-SM device. One method to do this is to use non-volatile storage such as an SD Card or eMMC attached to the OSD335x / OSD335x-SM to store the binary application so that it can be loaded from U-Boot.

NOTE: For simplicities sake, we are leveraging the fact that the SD card / eMMC has a valid Linux image with a file system that we can use to read and write our binary image. In a production scenario where you want to leverage the size savings of using bare metal, you will need the MLO and U-Boot portions of the image and then will need a file system to store the binary application.

First, we need to transfer the image to the OSD335x / OSD335x-SM development board.

1. Make sure that the development board has fully booted Linux; is connected to the host computer via a USB cable and login to the board from the host computer through SSH:

```
$ sudo ssh 192.168.7.2 –l debian
```

The default password for all BeagleBoard images is *temppwd*. Enter it when prompted.

2. Copy *gpioLEDBlink.bin* file from Host Ubuntu Computer to OSD335x board using SCP:

```
$ cd
~/bareMetal/AM335X_StarterWare_02_00_01_01/binary/armv7a/gcc/am335x/beaglebone/
gpio/Release

$ sudo scp gpioLedBlink.bin debian@192.168.7.2:/home/debian
```

Next, we need to access the U-Boot console to load and boot the application. In BeagleBone images, the console defaults to the UART0 peripheral. It is straight forward to access UART0 in most development boards (see individual board documentation for the port / pins to access UART0).

NOTE: For instructional purposes, we are demonstrating the interactive way to load and boot the bare metal application. In a production scenario, you would need to modify the “bootcmd” variable within the U-Boot environment to automatically load and boot the bare metal application.

1. Connect a *UART to USB cable* between UART0 pins of OSD335x and Host Computer. Use a free Serial Client like *Putty* to open a serial terminal and observe boot messages on boot up.

2. Set the terminal to *115200 Baud Rate (8N1).*

3. When the development board boots, U-Boot messages are shown on the serial terminal. To load and run a bare metal application from either an SD Card or eMMC, we must prevent U-Boot from

automatically booting Linux. To do so, hit the *Enter* on the keyboard *within first 2 seconds of OSD335x boot*.

Once Linux Boot is interrupted, U-Boot command line prompt can be seen as shown in Figure 1

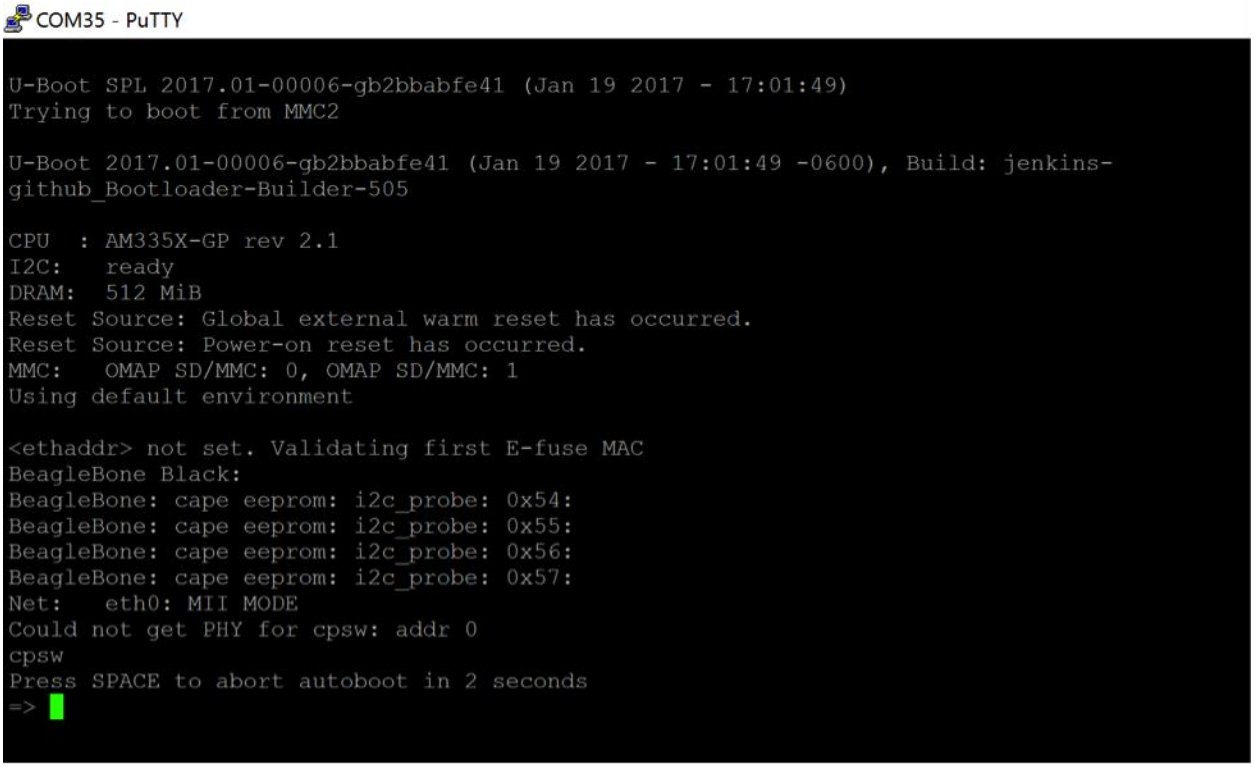

Figure 1 U-Boot Command Line

To load and run the bare metal application from either the SD Card or eMMC, we need to understand the correct device and partition where the bare metal application is stored. In the OSD3358-SM-RED development board, the SD Card is on MMC0 and the eMMC is on MMC1 and in the default BeagleBone image, the Linux filesystem is on partition 1.

4. To load and run the application from an SD Card:

```
load mmc 0:1 0x80000000 <path to your .bin file; e.g. /home/Debian/gpioLedBlink.bin>
go 0x80000000
```

Similarly, to load and run the application from eMMC:

```
load mmc 1:1 0x80000000 <path to your .bin file; e.g. /home/Debian/gpioLedBlink.bin>
go 0x80000000
```

7 Running Demo Application on OSD335x using TFTP

Another way, we can load and run the binary application that was built in Section 5 on the OSD335x / OSD335x-SM device is to use TFTP to transfer the image directly to U-Boot so that it can load and run the application.

Before we can use TFTP with U-Boot, you'll have to configure the host Ubuntu machine with a TFTP server.

1. Install TFTP server:

```
$ sudo apt-get install tftpd-hpa
```

2. Install TFTP client:

```
$ sudo apt-get install tftp
```

3. Configure TFTP parameters using your favorite editor:

```
$ sudo gedit /etc/default/tftpd-hpa
```

4. Modify the contents of the file:

```
TFTP_USERNAME="tftp"
TFTP_ADDRESS="0.0.0.0:69"
TFTP_OPTIONS="--create --listen --verbose /home/userName/cmpt433/public"
RUN_DAEMON="yes"
```

In the above block of code, make sure to replace *userName* with the username of your system.

5. Verify if TFTP is running:

```
$ netstat -a | grep tftp
```

You should see something like this: *udp 0 0 *:tftp*

6. Restart the server

```
$ sudo service tftpd-hpa restart
```

Now, we need to test if TFTP is working. To do this, we can just use the Ubuntu host machine to perform a loopback test.

1. Create a test directory and test file to transfer:

```
$ mkdir ~/bareMetal/TFTP
$ echo 'Testing TFTP successful' > ~/bareMetal/TFTP/testTFTP.txt
```

2. Verify TFTP file transfer by transferring a file from Ubuntu host to itself:

```
$ tftp
tftp> connect 127.0.0.1
tftp> get home/ubuntu/bareMetal/TFTP/testTFTP.txt
Received 17 bytes in 0.0 seconds
tftp> quit
```

Note: In the above command, replace *ubuntu* with the username of your Host Machine.

3. Verify the contents of the transferred file:

```
$ cat test_tftp.txt
Testing TFTP successful
```

At this point we are ready to use TFTP within U-Boot to get the bare metal application so that it can be loaded and run on the development board.

NOTE: Again, the steps shown below are how to interactively load and run the bare metal application. In a production scenario, the U-Boot environment will need to be modified to automatically load and run the application.

1. Both host Ubuntu machine and OSD335x / OSD335x-SM development board should be connected to the same local area network. This can be done over Ethernet or over USB using the RNDIS gadget framework that allows the development board to do IP transfers over USB.

2. Use Steps 1, 2 and 3 of Section 6 to get to a U-Boot command prompt on the development board.

3. Using a single command on the U-Boot command line, load and run the app using TFTP:

```
=> setenv ipaddr <board ip>;setenv loadaddr 0x80000000; setenv serverip <server
ip>;setenv tftproot /home/userNameOfHostMachine/bareMetal/TFTP;setenv bootfile
${tftproot}/gpioLedBlink.bin;tftp ${loadaddr} ${bootfile};echo -----Booting and loading
BareMetal App through TFTP-----;go ${loadaddr};
```

8 Booting Bare Metal Applications

At this point, you are now able to create, compile, load and run bare metal applications on the OSD335x Family of devices using either non-volatile storage such as an SD Card or eMMC, or over the network using TFPT. If you get any errors, please check that all the paths are configured correctly. Also, you can post questions or comments on our forum (https://octavosystems.com/forums/)

9 References

For more information, please refer to the following links:

- Bare Metal on the BeagleBone, Simon Fraser University, 2017
  https://www.cs.sfu.ca/CourseCentral/433/bfraser/other/BareMetalGuide.pdf
- Driver Creation Guide for BeagleBone, Simon Fraser University, 2017
  https://www.cs.sfu.ca/CourseCentral/433/bfraser/other/DriverCreationGuide.pdf