Articles » Internet of Things » Boards / Embedded devices » General

Getting Mono Running on a Beagleboard C4 Using Yocto

Richard Dunkley, 9 Nov 2014 CPOL

★★★★★ 5.00 (1 vote)

Demonstrates how to use the Yocto Project to get a custom Linux operating system with Mono up and running on the Beagleboard C4 development board.

Introduction

This article is a tutorial that describes how to get a custom Linux baseline and Mono (a cross platform .NET runtime environment) up and running on a Beagleboard (revision C4) development board. It is part of a larger article that explains how to use the Yocto Project to establish a custom Linux baseline across multiple hardware configurations. The outcome of this tutorial is that our Beagleboard development board will be setup to run C# applications and libraries. These applications and libraries can then be developed using a high level Integrated Development Environment (IDE) such as Visual Studio, SharpDevelop, or Xamarin Studio. This allows software developers with limited experience with embedded environments to write software for an embedded Internet of Things (IoT) device.

Setup

The basic setup of the Beagleboard (power connections, jumper settings, serial output, etc.) is not covered in this article since it is assumed that an owner of a board wishing to use this tutorial would have a basic understanding of these hookups from the supplied quick start guides or reference manuals provided with the board. The focus of this article is using Yocto to build the desired environment targeting the development board.

The following software and hardware components are used in this tutorial:

- Linux development system of your choosing (tutorial uses Mint Linux 17 64bit)
- Poky Build System v1.6.2 (Daisy)
- meta-ti metadata layer (Daisy branch)
- meta-mono metadata layer (Daisy branch)
- Beagleboard development board (revision C4)

The main Yocto Project tool that we will be using is the Poky build system. In order to use this tool we need a Linux development environment. Most mainstream Linux distributions are supported. I used Mint 17

64bit for mine. The Yocto Project website also has a "Build Appliance" which contains a VMWare Virtual Image of a Linux system all setup to play with. If using Poky on your own Linux environment be sure to install the required dependencies (see the Quick Start or the Reference Manual on the Yocto Project website for more information www.yoctoproject.org). The packages I had to install were the following: g++, texinfo, libsdl1.2-dev, texi2html, cvs, subversion, bison, flex, and mono-complete (to build mono).

I also recommend you read some of the Quick Start documentation on the Yocto Project website. I will outline the steps to create boot images and files for the Beagleboard in this article, but a more in depth understanding of the capabilities and features of Yocto can be obtained from Yocto's online documentation.

Using the code

Commands that should be run at the terminal on the Linux development system side are shown with the $ prompt:

```
$
```

Commands that should be run on the development board are shown with the # prompt:

```
#
```

File modifications will use "..." to represent the text in the file before and after the section referred to. This will allow us to show the sections in the files that need to be changed without displaying the entire file contents.

```
...
Text in the file
...
```

Tutorial

Create a folder for the Beagleboard poky build environment:

```
$ mkdir ~/beagleboard
```

Clone the git Yocto project core into the poky working directory:

```
$ git clone -b daisy git://git.yoctoproject.org/poky.git ~/beagleboard
```

We now need to pull in the board support package associated with our processor or board. The meta-ti board support package will work. This metadata layer is located on the Yocto project website. If you cannot locate a BSP for your target board using the Yocto website, try googling "yocto <board or processor> bsp" and you may find a community maintained one.

Clone the meta-ti board support package into the poky working directory. Note: when pulling in a board support package, be sure to read the README file associated with the package or the repo "About" section online. Sometimes they have additional steps that must be taken in order to use the BSP.

```
$ git clone -b daisy git://git.yoctoproject.org/meta-ti ~/beagleboard/meta-ti
```

Clone the meta-mono metadata layer into the poky working directory:

```
$ git clone -b daisy git://git.yoctoproject.org/meta-mono ~/beagleboard/meta-mono
```

Source the build environment script to create the configuration files:

```
$ cd ~/beagleboard
$ source oe-init-build-env
```

Add the meta-ti board support package layer and mono metadata layer to the build environment. Do this by opening up the ~/beagleboard/build/conf/bblayers.conf file and adding the layers to the BBLAYERS variable after meta-yocto-bsp. The file should look something like the following when finished:

```
...
  /home/<username>/beagleboard/meta-yocto-bsp \
  /home/<username>/beagleboard/meta-ti \
  /home/<username>/beagleboard/meta-mono \
  "
...
```

Now change the MACHINE variable to beagleboard. This tells the build environment which board to target when building the Linux environment. To do this edit the ~/beagleboard/build/conf/local.conf file. Scan down the file until you reach the following part:

```
...
#MACHINE ?= "edgerouter"
#
# This sets the default machine to be qemux86 if no other machine is selected:
MACHINE ??= "qemux86"
...
```

Add the beagleboard reference as the MACHINE above the default so the file reads:

```
...
#MACHINE ?= "edgerouter"
MACHINE ?= "beagleboard"
#
# This sets the default machine to be qemux86 if no other machine is selected:
MACHINE ??= "qemux86"
...
```

This sets the build environment to target the Beagleboard. If you don't know what the machine name is you can look under the BSP folder (meta-ti) subfolder: conf/machine (~/beagleboard/meta-ti/conf /machine). The names of the .conf files in this folder are all the names of the machines that can be targeted by the board support package. Also note that the default machine is qemux86. QEMU is an open source machine emulator. You can use this emulator to try out the custom built Linux distribution prior to targeting a specific board.

You may want to build a minimal image at this point to determine if your build environment is setup correctly. You can do this by running "bitbake core-image-minimal". We are going to skip this step to move onto building the minimal image with mono included.

There are lots of ways to mix and match packages and recipes. The mono metadata layer already provides us with some recipes to add mono to any image we want to create; however, here we are going to create our own image recipe that includes the minimal Linux image with the mono runtime included. This means the GUI elements of mono (classes that rely on libgdiplus) are not going to work because the basic minimal image does not include GDI. Since a lot of IoT applications may not contain a graphical interface we decided this was the best approach to show the minimal requirements to run C# on a board.

The first thing we need to do is create an include file that tells the build environment what we need to include. You can do this by running the following:

```
$ (cat << EOF
> IMAGE_INSTALL += "mono mono-helloworld"
> EOF
> ) > ~/beagleboard/meta-mono/recipes-mono/images/core-image-minimal-mono.inc
```

This command generates the file in the meta-mono layer. You can also create your own layer for this. If you compare this include file with the ~/beagleboard/meta-mono/recipes-mono/images/core-image-mono.inc file you'll notice that the only difference is the libgdiplus dependency is left out.

Now let's create the actual bitbake recipe file. You can do this by running the following:

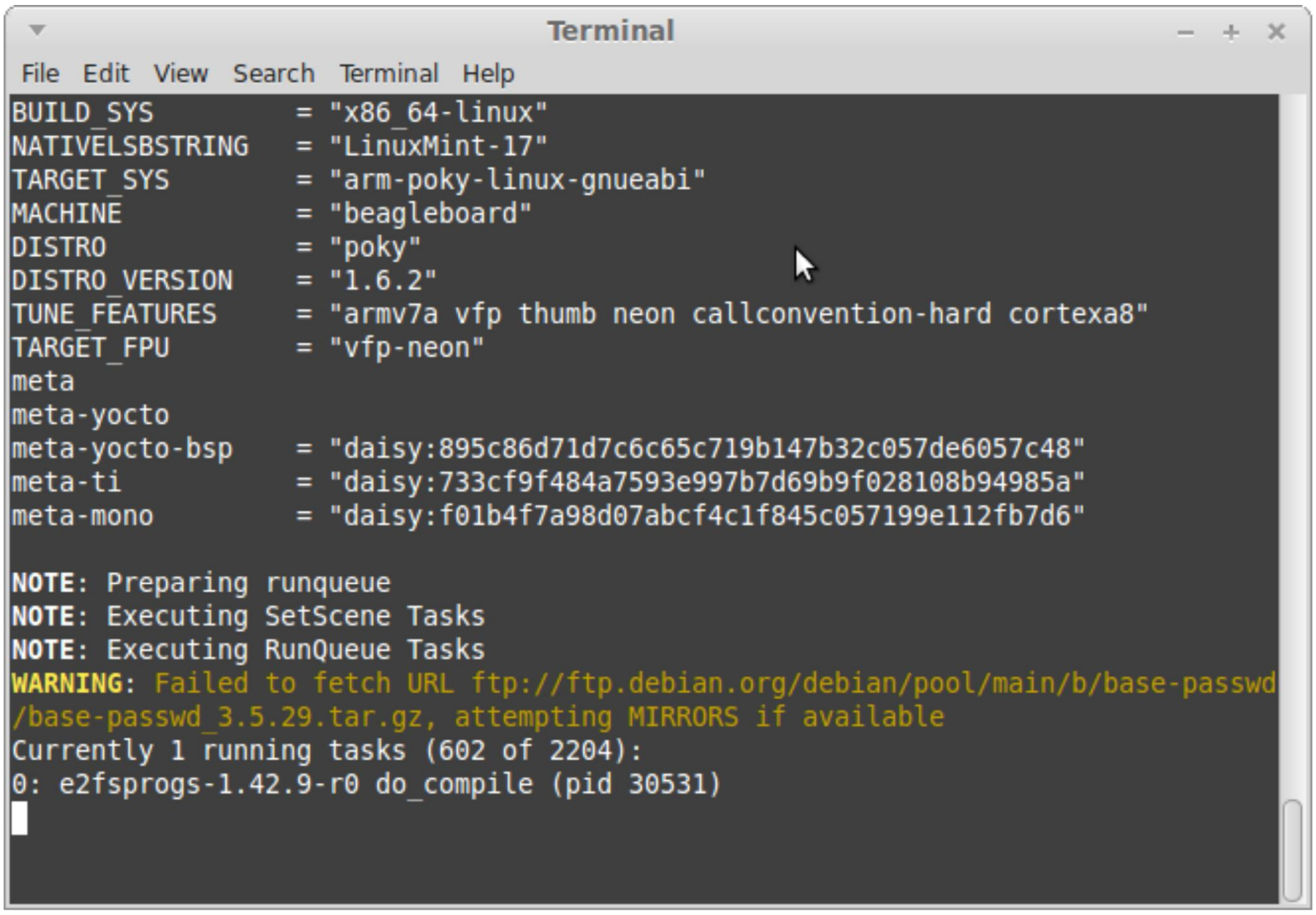
```
$ (cat << EOF
> require recipes-core/images/core-image-minimal.bb
> require core-image-minimal-mono.inc
> EOF
> ) > ~/beagleboard/meta-mono/recipes-mono/images/core-image-minimal-mono.bb
```

This command generates the bitbake recipe file. This recipe is fairly simple. The first line tells the build environment to include the recipe to make the core-image-minimal image. The second line tells the environment to add the mono files to the image.

We now have our image recipe and are ready to bake. Type the following to build the image (make sure you are in the build directory):

```
$ bitbake core-image-minimal-mono
```

This process may take a very long time depending on your environment and internet connection. It will first parse all the recipes and determine what source code needs to be downloaded from the various repositories. Note, even though we only included two recipes in our recipe file these two recipes represent a hierarchy of a large amount of dependent recipes. The parsing traverses the dependencies and determines all the files that need to be downloaded. It downloads the source code and caches it locally. The advantage is that subsequent builds that may require the source do not need to go out to the internet to obtain them. The disadvantage of this is depending on the recipe you are trying to build you may have a large amount of files that get downloaded and stored. After this build process our ~/beagleboard folder grew to 22GB. This is representative of the amount of storage required for even a very minimal build.



```
BUILD_SYS         = "x86_64-linux"
NATIVELSBSTRING   = "LinuxMint-17"
TARGET_SYS        = "arm-poky-linux-gnueabi"
MACHINE           = "beagleboard"
DISTRO            = "poky"
DISTRO_VERSION    = "1.6.2"
TUNE_FEATURES     = "armv7a vfp thumb neon callconvention-hard cortexa8"
TARGET_FPU        = "vfp-neon"
meta
meta-yocto
meta-yocto-bsp    = "daisy:895c86d71d7c6c65c719b147b32c057de6057c48"
meta-ti           = "daisy:733cf9f484a7593e997b7d69b9f028108b94985a"
meta-mono         = "daisy:f01b4f7a98d07abcf4c1f845c057199e112fb7d6"

NOTE: Preparing runqueue
NOTE: Executing SetScene Tasks
NOTE: Executing RunQueue Tasks
WARNING: Failed to fetch URL ftp://ftp.debian.org/debian/pool/main/b/base-passwd
/base-passwd_3.5.29.tar.gz, attempting MIRRORS if available
Currently 1 running tasks (602 of 2204):
0: e2fsprogs-1.42.9-r0 do_compile (pid 30531)
```

The image above gives an example of what should be displayed on the terminal while the custom Linux image is being created.

When the process completes successfully you can find the required boot files in the ~/beagleboard/build /tmp/deploy/images/beagleboard directory. There are a lot more files in this folder than are needed to boot the board. We will need the following files:

MLO
u-boot.img
uImage
core-image-minimal-mono-beagleboard.tar.gz
modules-beagleboard.tgz

The MLO file can be considered the first stage boot loader. It loads u-boot in to the system and runs it. u-boot is a boot loader that can then read the Linux kernel image (uImage) and boot the Linux environment with specific parameters. The image file (tar.gz) and modules file contain the Linux base file system and additional modules that need to be installed into it.

Next we need to generate the file that contains the specific parameters for u-boot to setup the Linux environment with. The file can be created using the following commands:

```
$ (cat << 'EOF'
> autoload=no
> hostname=BBC4
> console=ttyO2,115200n8
> mpurate=auto
> buddy=none
> vram=12M
> dvimode=hd720 omapfb.vram=0:8M,1:4M,2:4M
> defaultdisplay=dvi
> loadaddr=0x80200000
> mmcroot=/dev/mmcblk0p2 rw
> mmcrootfstype=ext3 rootwait
> mmcargs=setenv bootargs console=${console} ${optargs} mpurate=${mpurate} buddy=${buddy}
camera=${camera} vram=${vram} omapfb.mode=dvi:${dvimode} omapdss.def_disp=${defaultdisplay}
root=${mmcroot} rootfstype=${mmcrootfstype}
> mmcboot=echo Booting from mmc (uEnv.txt configuration) ...; run mmcargs; bootm ${loadaddr}
> loaduimagefat=fatload mmc ${mmcdev} ${loadaddr} uImage
> uenvcmd=run loaduimagefat; run mmcboot
> EOF
> ) > ~/beagleboard/build/tmp/deploy/images/beagleboard/uEnv.txt
```

If you are familiar with u-boot some of these parameters probably look familiar. There are a lot of different kernel parameters that can be included in this file. The one above is a subset of one posted by Oscar Gomez Fuente.

We now have all the files we need to boot the system. Let's create the bootable SD card to place in the system. The Beagleboard uses a two partition SD card to boot the system. The first partition contains a FAT file system containing the kernel and some boot files. The second partition is an ext3 partition containing the file system. I am not going to address partitioning an SD card since there are multiple tutorials on the web for this and it typically depends on the size and parameters of your card.

If you have a partitioned card already I recommend reformatting the two partitions using the following commands in order to start with a clean file system (needed for the FAT portion). Switch the 'mmcblk0p1' and 'mmcblk0p2' partitions for the ones on your SD card (may be sda1 and sda2, etc.). Be careful to get the right partitions (you don't want to format the wrong drive). Note, these instructions are based on the Beagleboard instructions in the Dora Yocto BSP project site.

```
$ umount /dev/mmcblk0p1
$ umount /dev/mmcblk0p2
$ sudo mkfs.vfat -F 16 -n "boot" /dev/mmcblk0p1
$ sudo mke2fs -j -L "root" /dev/mmcblk0p2
```

At this point on my Mint system the card comes up with the boot and root partitions mounted in /media /mint/boot and /media/mint/root. (mint is my username.) You may have to manually mount the partitions depending on your Linux environment. The following commands will use my mounted folders. Be sure to adjust the commands to adhere to your system's mount folders.

Now let's copy over the required files to the SD card. The order the files are copied over is important because the MLO file must be in the first sector in the first partition's FAT file system. This is so the chip knows where to find the first stage boot loader.

```
$ cp ~/beagleboard/build/tmp/deploy/images/beagleboard/MLO /media/mint/boot/MLO
$ cp ~/beagleboard/build/tmp/deploy/images/beagleboard/u-boot.img /media/mint/boot/u-boot.img
$ cp ~/beagleboard/build/tmp/deploy/images/beagleboard/uImage /media/mint/boot/uImage
$ cp ~/beagleboard/build/tmp/deploy/images/beagleboard/uEnv.txt /media/mint/boot/uEnv.txt
```

That covers the boot partition, now lets copy over our custom Linux file system to the second partition.

```
$ sudo tar xf ~/beagleboard/build/tmp/deploy/images/beagleboard/core-image-minimal-
mono-beagleboard.tar.gz -C /media/mint/root
$ sudo tar xf ~/beagleboard/build/tmp/deploy/images/beagleboard/modules-beagleboard.tgz -C /media
/mint/root
```

We are now setup to boot from the SD card. Eject the card and plug it into the Beagleboard. If an image is already stored in the NAND you will need to clear it using the following commands. Hold down the 'USER' button when hitting reset and you will get the u-boot prompt. Type the following commands and then reset without holding the 'USER' button:

```
# nand unlock
# nand erase
# nand erase.chip
```

If the SD card boots up to a login that looks similar to the following:

```
Poky (Yocto Project Reference Distro) 1.6.2 beagleboard /dev/ttyO2

beagleboard login:
```

Use "root" as the username to login and you should get access to the shell. Now type the following to determine if mono was setup properly:

```
# mono --version
```

It should show something similar to below:

```
Mono JIT compiler version 3.4.0 (tarball Sat Nov  8 10:52:14 MST 2014)
Copyright (C) 2002-2014 Novell, Inc, Xamarin Inc and Contributors. www.mono-project.com
        TLS:           __thread
        SIGSEGV:       normal
        Notifications: epoll
        Architecture:  armel,vfp+hard
        Disabled:      none
        Misc:          softdebug
        LLVM:          supported, not enabled.
        GC:            sgen
```

Since we included the mono-helloworld recipe in our image you should be able to run the following command:

```
# mono /usr/lib/helloworld/helloworld.exe
```

If "HelloWorld" printed out on the console then you just successfully ran a .NET application on your development board.

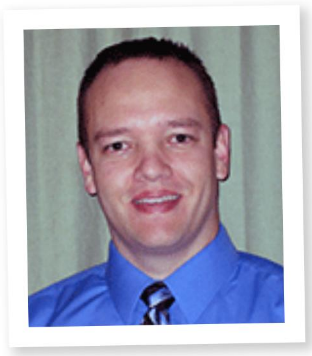
History

November 9th, 2014 - Initial Submission

License

Share

About the Author



Richard Dunkley
Engineer
United States

I earned my Bachelor of Science in Computer Engineering from Utah State University in 2004 with a minor in Mathematics and Computer Science. I also obtained a Masters of Science in Electrical Engineering from Utah State University with an emphasis in Embedded Systems. I am proficient in C#, C, assembly, and VHDL and have written a variety of applications targeting embedded processors. My experience includes robotics, LIDAR imaging, FPGAs, and various communication protocols. I currently work for the Air Force as the Technical Lead over a team of engineers at Hill Air Force Base, Utah.

Comments and Discussions

1 message has been posted for this article Visit **http://www.codeproject.com/Articles/840470/Getting-Mono-Running-on-a-Beagleboard-C-Using-Yoc** to post and view comments on this article, or click **here** to get a print view with messages.