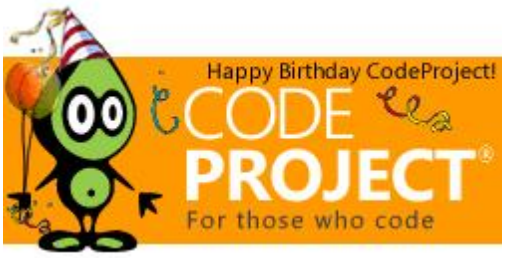

Writing Software for a Variety of IoT Devices Using the Yocto Project

Richard Dunkley, 9 Nov 2014 CPOL

0.00 (No votes)

Demonstrates how to use the Yocto Project to get a custom Linux baseline operating system containing Mono for use on a variety of different hardware configurations.

Introduction

The internet-of-things is here, but it is a very different paradigm than what most companies are used to. For decades, software development companies have written software that for the most part was capable of running on the majority of computing devices used by their customers. The problem with the Internet of Things (IoT) paradigm is how do you leverage your existing code to target a variety of devices with very different hardware requirements. For example, a home automation company may want to have a variety of sensors and IoT devices that feed into the overall control of the system. The hardware requirements for a device to turn on and off a light are very different than a device which performs facial recognition. Typically the company would need to hire embedded software developers to write custom software for each device. This results in higher engineering costs and extended time to market for products.

Additionally many developers have struggled in balancing the various dependency requirements for a software framework. For example, you may download an image for a Raspberry Pi device that allows you to run Linux, but later on you find out the kernel headers aren't included in the image, or the version of the kernel or libc library included is not the version your software requires. It can be difficult to line up all the dependent versions for your system when you are dependent

on a 3rd party for your base Linux distribution images.

There have been a lot of technologies that have come out recently that have mitigated these risks by giving developers the ability to maintain a consistent operating system across a variety of very different hardware configurations. One of these technologies that we will address in this article is the Yocto Project.

"*The Yocto project is an open source collaboration project that provides templates, tools and methods to help you create custom Linux-based systems for embedded products regardless of the hardware architecture*" - www.yoctoproject.org

This means we can build our own custom Linux distribution that contains all the libraries and dependencies we require for our software, but is consistent across very different hardware devices. Initially this may sound a lot more complicated, we now have a consistent operating system baseline, but we had to build it ourselves. The Yocto Project uses Poky as a build system. It leverages off of OpenEmbedded (OE) and BitBake to divide the operating system up into recipes. These recipes are then combined into layers. A board support package (BSP) can then be used to provide Poky with the required hardware information that the other layers build upon. You can switch out the BSP to target very different devices while maintaining the other layers. You can then define a recipe that includes all the layers and recipes you need for your custom Linux system. This enables you to have the same kernel version, library versions, and dependencies across very different hardware platforms.

Don't worry; it is much easier than it sounds.

Background

The purpose of this article is to demonstrate how to use the Yocto Project's tools to create a custom Linux operating system that is consistent across multiple hardware platforms. This will give us a baseline framework for developing an example software application targeting very different IoT devices. The development boards chosen are all made by different manufacturers, have different purposes, and different processors. Because of their differences, I thought they would provide a good example on how easily the Yocto Project can be used to develop the same Linux operating system for very different devices. The following development boards were chosen (I will be hopefully adding to this list later as well):

- Raspberry Pi - This is one of the most popular development boards out there. I decided to use it because... hey... we all have one lying around. The SoC on the board is a Broadcom BCM2835 containing an

ARM1176JZFS core. It also has a built in Videocore 4 GPU capable of Blu-ray quality playback (using H.264) at 40MBits/s. The GPU can also be used to write custom OpenGL ES2.0 or OpenVG applications. This device would represent a more capable IoT device that is used to drive a display or stream audio/video.

- Beagleboard Rev. C4 - This board uses a Texas Instruments OMAP3530 SoC. The SoC contains an ARM Cortex-A8 core, TMS320C64x+ DSP core, and a PowerVR SGX GPU. It can be used in video processing applications. It is an older development board and newer versions of this platform have been developed (BeagleBone-xM, BeagleBone), but it was chosen because it has a different boot process than the other examples, is known to be finicky in its boot loader, and has been the source of a lot of frustration for me in the past in getting all the software dependencies to line up on custom-built or provided images. It is a good example of how much easier the Yocto Project tools are to use than past solutions.
- MicroZed - This board uses a Xilinx Zynq SoC containing Dual ARM Cortex-A9 processors. It does not have any built in graphic cores, but does have built in FPGA logic. This allows very custom hardware designs to be developed and added to the chip. For example, if your IoT device needed to perform some custom DSP algorithms for filtering of a sensor or even performing high end DSP algorithms such as facial recognition or object detection, that logic can be synthesized into the logic in the FPGA portion. I thought this would be a good example of a more basic ARM system (no video cores) and also one capable of very custom hardware interaction.

The purpose of this article is to show the ease in generating custom Linux development environments for a variety of hardware platforms. As such I won't spend a lot of time on the development board setups. It is assumed that if a reader has one of these boards and is attempting to run the custom built image on them that they would be familiar with the necessary power hookups, board jumper configurations, etc. to do so. Also note that these development boards all have readily available Board Support Packages (BSPs) for Yocto's Poky build environment. Because the Beagleboard is older than the other two, a BSP targeting the current Poky version is not available; however, I did find that the current Texas Instrument's BSP layer will still work for the older board. If there is not an available BSP for your board or processor provided on Yocto's Project Website (very few can be browsed to) you can try googling "Yocto <board or processor name> BSP" to see if a community or 3rd party one has been developed. If still no luck you can try customizing an available BSP that has the same or similar SoC or ARM core. The

creation of a new BSP is not part of this project.

Application Development

Using the Yocto Project we will be able to create a custom Linux environment, but I also wanted to show how easy it is to get a runtime environment setup for developing, debugging, and running applications that have been developed with a high level programming language. The ability to re-use code that has been developed using C#, Ruby, Perl, Java, etc. would be valuable to a developer that may have libraries already written in that language. It also absolves the developer from having to be familiar with lower level C++ and C coding on an embedded system. If you know nothing about how an ARM processor, cross-compilers, or embedded Linux works, you can still develop software for your IoT device.

As part of this tutorial, I will demonstrate how to include Mono (a cross platform .NET runtime environment) in the Linux environment so that our operating system is ready to run our C# source code and libraries on each of the development boards. In later articles I will demonstrate how to develop applications to target the IoT devices using high level development environments such as Visual Studio and Xamarin Studio (MonoDevelop) running on an external PC. For this tutorial; however, we will focus on getting the operating system and runtime environment up and running on the boards. We will also do a quick validation of the system to ensure the Mono runtime is functioning properly.

Development Board Tutorials

I decided to break the board tutorials out into separate articles. This article was getting lengthy and I thought each individual tutorial might be a good reference for developers seeking to establish a Mono runtime on their board without having to wade through additional board tutorials. I also wanted to add a few tutorials for boards I have laying around. It would be easier to add a link later than attempt to add in an entire section for an additional tutorial.

Much of the tutorials are the same for the different boards and can be catered for boards or processors not covered here. The Yocto tools also allow you to target QEMU which is a hardware emulation environment. Even if you don't have a development board you should be able to run through these tutorials (with some slight tweaking) using QEMU. This is also a nice advantage of using Yocto. Not only do you get a consistent custom Linux baseline across different hardware configurations, you also get a standardized way of creating and building software for those various embedded systems.

You can also begin with QEMU developing software prior to actually having the hardware in place.

Here are links to the individual board tutorial articles:

- Beagleboard C4
- Raspberry Pi
- MicroZed

So now we have a variety of hardware devices that have relatively the same Operating System and software framework environment. What's next? We now have the ability to begin to develop communication software that will allow the very different boards to communicate with each other or a Web service. After all, isn't that the point of IoT devices is to be able to send out the drones to capture information and process that information in the cloud. In subsequent tutorials, I will demonstrate how easily this can be done.

Yocto Advantages and Disadvantages

One concern you may have after going through the tutorials is it almost seems too good to be true. Why isn't everyone using this already? Fortunately, the development boards chosen above have BSPs readily available. If a BSP was not available for a specific processor or board, it would take some additional expertise for a developer to modify one or develop one from scratch. This risk is a valid concern; however, the Yocto project has a lot of big name participants and it continues to grow its list every day. For example, Intel, AMD, Texas Instruments, Dell, Broadcom, and LG are all companies that are actively involved as a participant in the project. Those are just a few of the official participants. The Mono and Raspberry Pi metadata layers used in the tutorials are community maintained. As Yocto becomes more widely used, more board support packages and layers will be supported. In the end, even if the processor you are targeting does not have any BSPs that are similar, the Yocto project has a large amount of documentation to get you started with your own recipes and BSPs.

We now have a custom Linux Baseline with the same library versions and Linux kernel across very different development boards. Notice that although the build process was relatively the same for the different boards, the output files that are used to boot the boards are very different. The Raspberry Pi generated an SD card image that we could write directly to an SD card. The Beagleboard required the creation of additional files. We also had to partition the SD card and copy the files over in a specific order. The bad news is that using Yocto does

not always mean that we won't have to perform additional customization steps for our different pieces of hardware. (That is why the README files on the BSP websites are so important.) The good news is that it has proved dynamic enough to accommodate very different boot methods from different chip vendors.

Conclusion

In the end, the Yocto Project has been an excellent tool for providing us the capabilities to produce a custom Linux distribution across various embedded hardware configuration. It is also an excellent tool for developing IoT device applications. Here are some of the advantages of its use:

1. Companies and developers can now develop a variety of IoT devices with very different hardware requirements, but leverage off the same software and libraries.
2. Shorter time to market for new IoT devices by re-using source code and software developed for other devices.
3. High level source code such as C# can be re-used on the Web server, Desktop Application, and IoT device.
4. If BSP packages are available for the target processor, then a developer with no prior experience in embedded applications can begin developing software for that device.
5. Consistent build process for tracking and building images for a variety of IoT devices.

History

November 9th, 2014 - Initial Submission.

November 11th, 2014 - Added the MicroZed development board and fixed a link.

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

0 messages have been posted for this article Visit **http://www.codeproject.com/Articles/840492/Writing-Software-for-a-Variety-of-IoT-Devices-Usin** to post and view comments on this article, or click **here** to get a print view with messages.