Yocto or Debian for Embedded Systems

White Paper

Document name: Yocto/Debian Comparison White Paper				
Document owner: Mads Doré Hansen				
Filename: Yocto_Debian_Whitepaper			Modified: 2017-08-09	
Classification: Public	Client: Prevas	Project/Process: White Paper	Version: R1	Page: 1(8)

Contents

1 Introduction

This white paper presents a comparison of the Yocto Project and Debian for development and maintenance of an embedded Linux platform.

Strictly speaking, the term Linux refers to an operating system kernel, and not all the individual parts that make up a complete system. A given configuration of Linux kernel, bootloader and root-filesystem is referred to as a Linux Distribution.

In the context of this paper, Yocto means a Linux system based on the Yocto Project:

https://www.yoctoproject.org

Yocto is a tool to configuration full Linux Distribution and is not a Linux distribution in itself.

And Debian means a Linux system based on:

https://www.debian.org

Debian is a Linux distribution (or a number of Linux basic distributions) and is not a tool for making distributions in itself.

Hence different nature of Yocto and Debian a comparison of the two as foundation for development and maintenance of embedded Linux systems will mostly be a comparison between using a tool to make a customized Linux distribution or starting with a specific Linux distribution and then change it into a partly customized Linux distribution.

1.1 Executive Summary

At present, using a Debian distribution as the foundation for an embedded Linux system will be more proprietary, less portable to other hardware types, harder to maintain and with lower traceability than using Yocto to build a similar embedded Linux system. The Debian based system will require more resources (RAM / Storage) than a tightly adapted and optimized Yocto based Linux system.

The discussion amongst Linux developers regarding Debian vs. Yocto is seldom based on facts it is merely based on feeling and habits. As presented in this white paper Debian and Yocto comparison is mostly like comparing apples and pears, they were not made to solve the same problems. Shown in the fact based comparison of chapter 3 the two each have their strengths and weaknesses.

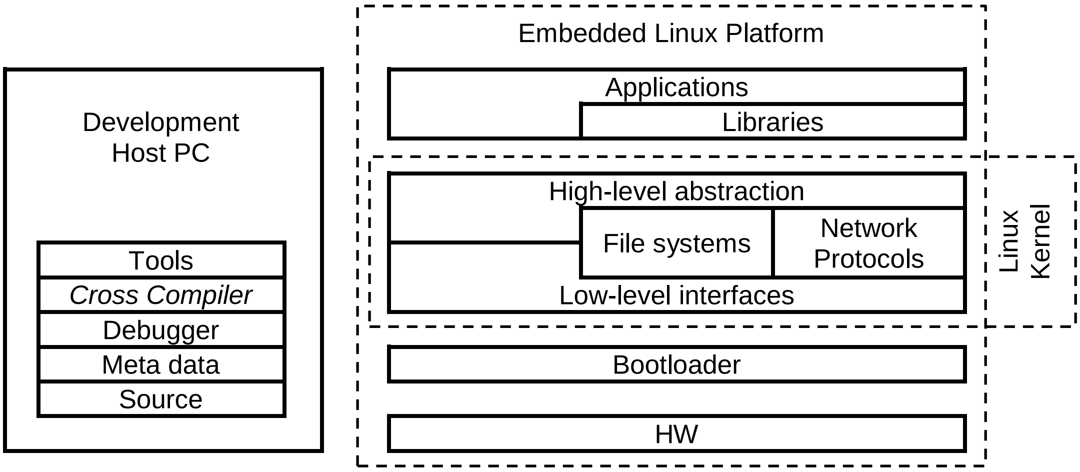
- Debian is good for fast trials, proof of concept and desktop like environments on hardware already supported by Debian with large memories and limited requirements for maintenance, traceability and reuse across different hardware targets.

- Yocto is good for customized embedded environments with various hardware support and small to large memories and requirements for maintenance, traceability, longevity and reuse across difference hardware targets.

The two difference sets of strengths of Debian and Yocto sadly often results in Debian being used for early prototyping (which makes sense) and afterwards being used for the final customized Linux system, with the argument: “We have Debian already, must be cheaper to continue than switch to Yocto”. The strength and weaknesses, as presented in chapter 3, shows that it would in almost any longevity industrial Linux system by more rational to used Debian (or other prebuilds) for early prototyping and switch to Yocto as soon as detailed customization is started.

2 Linux System Parts and Development Short Overview

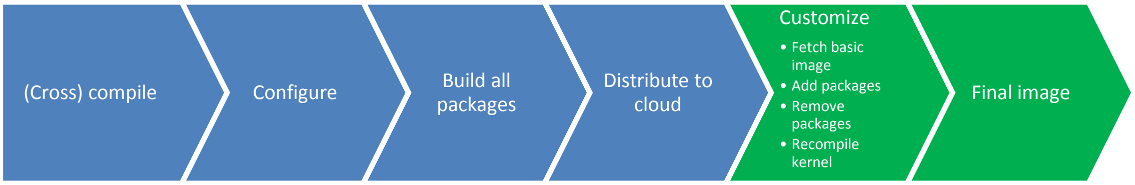
Regardless if a Linux distribution is based on Debian or Yocto the development environment and final Embedded Linux Platform consists of the parts shown in the illustration below.



Cross compiling is almost only used for Yocto based systems, Debian mostly uses on target compilation (therefore Cross Compiler is marked in *Italic*).

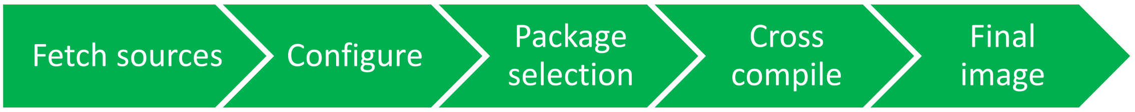
Steps involved in make a Linux distribution with the two difference approaches are shown roughly below. Blue marking steps that the developer is without influence on and green marking the steps the developer has influence on.

Debian steps



Normally done manually or in proprietary scripts of different kind.

Yocto steps



Normally done in the recipe structure and syntax of Yocto.

3 Yocto vs. Debian

Even though the source code, kernel and bootloader is the same (or at lease often originate from the same FOSS projects) the conceptual differences are huge in how everything is developed. The main differences are shown in the following overview.

Issue	Yocto	Debian
Compilation	Yocto fully relies on cross compiling. Some open source packages was originally not designed for cross compiling, putting stress on use of these packages in a Yocto based system. But throughout the last 5-10 years most common Linux packages/features have be transferred to allow cross compiling. So the discussion on using/not using cross compilation is today often more a matter of habit than technical difficulties.	Dependent the HW platform in question the Development Host PC will have or not have a Cross Compiler enabling e.g. compiling of kernel for a Debian based Linux system off-target. Most often Debian based systems rely on on-target compiling. **Debian is thereby stressed on smaller targets. E.g. on-target compiling of a Linux kernel on a 400MHz system is quite slow.**
Package/Feature addition	Added through recipes into a full firmware build (or single package build) with cross compilation on the Development Host PC. Either add only through full firmware updates or a package manager by choice (could be the Debian package manager). Using a package manager would require the developer to build that package in Yocto in the right context.	Added through package installation on-target via the Debian Package manager.
Configuration and customization of single features	Each software package/feature has its own recipe and/or meta-layer,that provides configuration of how that package/feature is build. Allowing full configuration and customization of every detail within the structured scope of Yocto. All configuration can in Yocto be tracked and controlled between two updates of the system.	Most software package/feature comes (partly) pre-build and configured, with is one of the great advantages of getting a running platform fast in Debian. This often results in fewer possible configuration and customization possibilities, as the developer then relies on the previous choices of the Debian package provider. Full configuration and customization of single features in Debian thereby often leads to manual, non-standadized, compilations with harder or no reproducibility. It is hard to know if a given configuration is/will be the same between two updates of the system.

Issue	**Yocto**	**Debian**
Configuration and customization of the full platform	The full platform is also configured and customized in recipes and/or meta-layers, enabling the possibility for maximization of source and meta-data reuse across differently configured/customized HW platforms. E.g. if the same configuration is need both on an x86 and an ARM based HW the bootloader configuration and kernel configuration is changed in Yocto and the full system is recompiled. Allowing full reuse all other software parts on both HW types.	As Debian is based mostly on prebuild packages a differently configured/customized platform most often means a complete new separate Debian Linux in both development and maintenance. E.g. if the same configuration is need both on an x86 and an ARM based HW the full manual work is needed to make two Linux systems, one per HW to support. Often it is seen that this leads to great variation on the features of the Linux, as not all features exists for both ARM and x86 in same versions and configuration in the pre-build Debian packages. For small custom systems a Debian tool called debootstrap exists, but has limitation as e.g. ARM systems cannot be configured on an x86 host. Prevas have not seen this tool in use at any of our customers yet, the usual setup is proprietary and home made.
Learning curve	Yocto has a steeper learning curve than Debian, mostly due to: - Cross compiling. - Meta-layer concept. - Recipe build-up. - “Non-forgiveness” of missing dependencies. **The high learning curve is often the most elaborated reason the developers to avoid Yocto.**	Debian has a lower learning curve than Yocto, mostly due to: - Package installation looks and feels like a Desktop Linux. - First running system is up fast.

Issue	Yocto	Debian
Reproducibility	Is very high as everything (if not made “messy” on purpose) is recipe based and cross compiled, often in a Docker environment, with possible source mirroring. Enabling full automated builds in a build server environment from scratch without human interference.	Is very low as the Package installation nature of Debian relies on cloud based pre-compiled packages, ending up in manually build gold-samples of the final Linux system. This is often counteracted by developers by build proprietary script based tools around Debian working a given host machine, adding customized knowledge and making a steeper learning curve for newcomers to a given project. **Missing or extremely expensive reproducibility is the most common reason companies discontinues Debian based Linux’s and switches to Yocto.**
Patching	The level of patching used in Yocto often limited, relying on patches for each software package already available upstream and the patches need to provide a build for the given target platforms.	The patch level of Debian is often considered high as Debian is more generic and the project has some rules about how Debian shall look. Heavy patching results in more complex systems with respect to debug and maintenance, as patches on patches make is a lot harder to determine what code is actually running on the system. Likewise impact from updating the lowest layer in that patch stack is often impossible to forsee.
Host tool dependencies	Yocto is dependent on a number of host tools to work. The list is constantly getting shorter, but still is exists and will never go away completely. It has become good practice for Yocto developers to use Docker images as an efficient tool the overcome the final host tool dependencies, reducing them the host being able to run a Linux Docker image.	If on-target building is used and manual packages installation Debian does not have any critical host tool dependencies. **Often a reason for a quick selection of Debian, as it is very easy to get started on a generally good supported HW platform.**
Scaling and Automation	With request to scaling and automation of Linux development onto multiple platforms and developers the recipe and meta-layer approach of Yocto provides significant advantages. Mostly as the work can be split efficiently between developers and build/test can be automated without human interference.	Debian “golden copy” approach is not very feasible in respect to scaling to multiple HW types and more developers. Likewise automated build/test requires proprietary tools and on-target SW build is not very feasible for efficient automation and scaling.
FOSS license overview	An overview of included licenses in a distribution is provided as output from a Yocto build.	Must be manually extracted for each software feature included in the basic system and installed with the package manager.

Issue	Yocto	Debian
Testing	When the Linux system has been made the same testing possibilities exists. The testing possibility differences only/mostly adheres to automation of system build and cross-compiling SDK tests, address tests of error occurring between SDK/cross-compiler for application and the features available on the target.	
Debugging	The same tools are available for both Yocto and Debian.	
		A Debian there are bindings between the Debian version running on the host machine and on the target. Without a match the debug tools might not work.
Boot time optimization	The nature of developing a customized distribution bottom-up usually provides “automatic” optimization of boot time, with respect to a more general distribution.	It is most often seen that Debian based distribution is starting a number of un-required services (as it is that nature of general systems) and thereby adding the boot-time. Currently a project is running in Prevas where similar distributions is made for the same project in both Yocto and Debian. Current the Yocto based distribution boots in about 10 seconds and the Debian based is booting in about 60 seconds.
RootFS size	As Yocto is used as a configuration tool the root file system is initially reduced and the configuration can be controlled to minimise the footprint of the distribution to “only what is actually needed”. Current a project is running a Prevas making a 30MB unzipped root filesystem for an ARM board in Yocto.	A commonly seen lightweight Debian system is e.g. armbian (Debian for ARM). This has a 250MB gzipped root file system.
	On the “standard” distributions with Graphics and without much optimization the root file system footprints (unzipped) for the embedded distributions are normally seen around: Debian ~ 1 GB. Yocto ~ 250 MB.	
Kernel sizes	Are the same for both types of systems.	