

Arrow's Embedded Linux Workshop for Intel SoC FPGAs

**Building A Custom Linux Board Support Package
(BSP) for SoC FPGA Systems
Using the Yocto Project**

Workshop Goal

BUILD A CUSTOM BSP USING THE YOCTO PROJECT

Create a Bootable Linux image, for a SOC FPGA based product, by creating a custom Yocto Project layer

Workshop Prerequisite

DEVELOPMENT KIT

- Use any of the development kits listed below
 - DE10-nano
 - purchase at
 - DataStorm DAQ
 - purchase at
 - SoCKit
 - request a loaner from Arrow



Workshop Prerequisite

REQUIRED SETUP



- Follow the instructions on the [Prerequisite page](#)
- Includes installation of the host Virtual Machine

Virtual Machine

A Virtual Machine (VM) running a Linux OS, is required when building the Workshops. The VM needs to run inside of a player like *VMware®*.

Micro SD Card

A Micro SD Card with at least 8GB of capacity is required.

USB to Micro SD Card Adapter

A USB to Micro SD card adapter is required. [One like this](#) can be used.

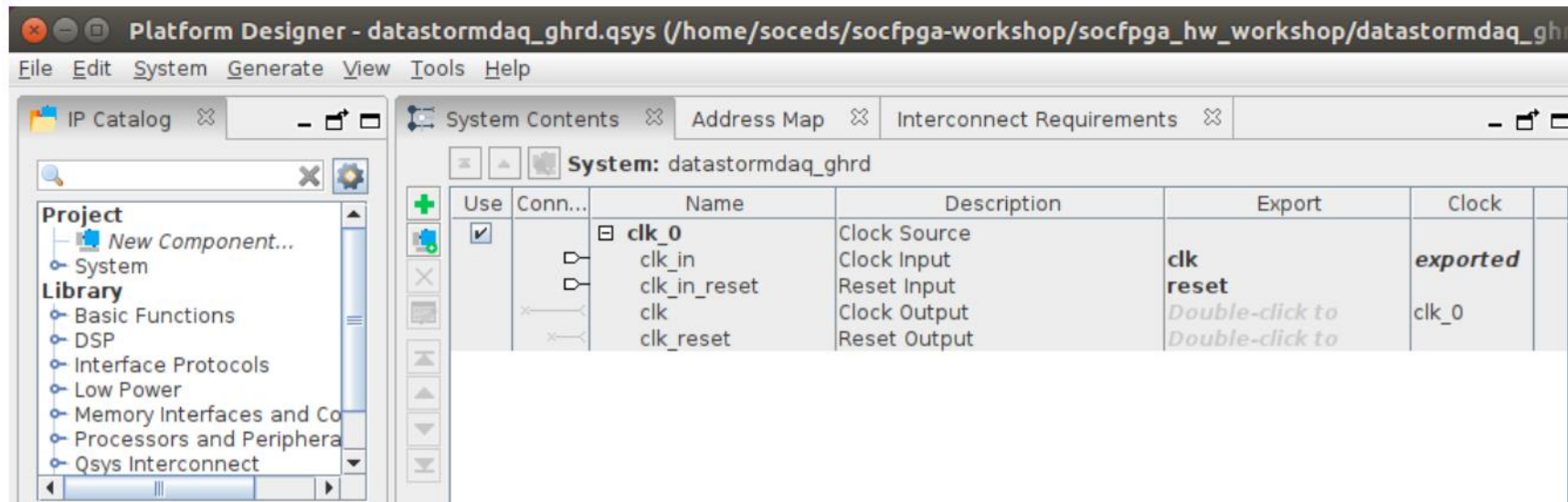
Install the Arrow USB Programmer

The Arrow USB Programmer provides a JTAG path to the Cyclone V SoC device. It supports programming and debug capability for developers. In order to use it with the Intel FPGA tools in the VM, the Windows drivers must be installed.

Workshop Prerequisite

COMPLETE THE HW WORKSHOP

- Attend the hardware workshop
- Built the GHRD for one of the three available development kits



Agenda

- Linux Overview
- Linux Board Support Package (BSP)
- Introduction to Yocto
- Build a custom Linux BSP using Yocto
- Linux OS for Intel SoC FPGAs

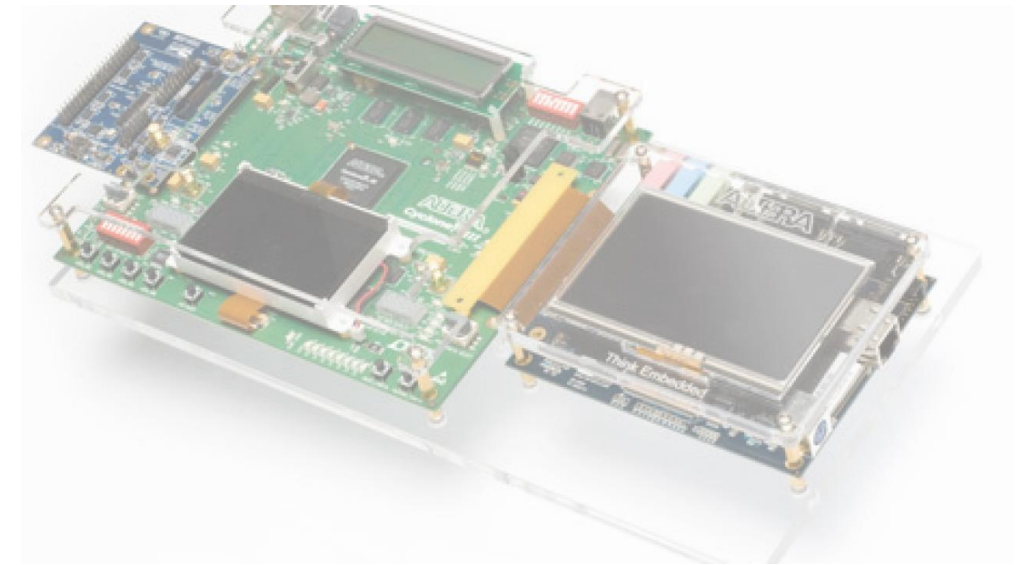


A dark-colored sports car, possibly a Porsche Carrera GT, is driving on a winding asphalt road that curves through a mountainous landscape. The road is flanked by steep, rocky slopes. In the background, there are snow-capped mountain peaks under a bright blue sky with scattered white clouds. The car has several sponsor logos, including 'NOW' on the side and 'GRAN TURISMO' on the windshield.

Linux Overview

What is Embedded Linux?

- Embedded refers to a computer system that has a dedicated function within a larger mechanical or electrical system
- Linux is an operating system
 - One of the most popular platforms on the planet
 - Android is powered by the Linux operating system.
 - Open source, community supported, Linus Torvalds 1991
- An operating system is software that manages all the hardware resources associated with a product
- A Linux system is comprised of Bootloader, Device tree, Kernel and root file system. These terms will be discussed in detail in coming slides



Who uses Linux

IT'S EVERYWHERE

- Super Computers
- NASA
- Space Robots
- Games Consoles
- The Large Hadron Collider
- U.S. Dept of Defense
- Smart TVs
- Smartwatches
- The Amazon Kindle
- Instagram (And Basically The Entire Internet)
- In-Car Entertainment
- Self Driving Cars
- Smart Refrigerators
- Advanced Air Traffic Control
- Chromebooks
- Japanese High-Speed Train
- The New York Stock Exchange

Why use Linux in an embedded system

EMBEDDED REFERS TO A COMPUTER SYSTEM THAT HAS A DEDICATED FUNCTION

- Open Source
 - No royalties or licensing fees
 - Freedom to read, modify, and redistribute the source code as the user sees fit
- A stable kernel, in use by large numbers of programmers
- Allows developers to program hardware “close to the metal.”
- Multiple suppliers for software, development and support
- Networking ability
- High security
- Runs on any hardware
- Strong community support

A dark-colored sports car, possibly a Lotus Evija, is driving on a winding asphalt road that curves through a mountainous landscape. The road is flanked by steep, rocky slopes with patches of snow. In the background, a vast valley with a lake and distant mountains is visible under a bright blue sky with scattered white clouds. The car has several sponsor logos, including 'NOVON' and 'FRAM', and a 'GRAND TOURING' decal on the windshield.

The Linux Board Support Package (BSP)

The Linux Board Support Package (BSP)

HOW TO SUPPORT A CUSTOM BOARD

- From the Yocto Project

A Board Support Package (BSP) is a collection of information that defines how to support a particular hardware device, set of devices, or hardware platform. The BSP includes information about the hardware features present on the device and kernel configuration information along with any additional hardware drivers required.

- A Linux BSP is comprised of

- Bootloader
- FPGA image (for an SOC FPGA system)
- Device tree
- Kernel
- Root File System (rootfs)



Linux OS

- A custom board will require a unique BSP

The Bootloader

WHAT IS A BOOTLOADER

- Bootloader function is to load the Linux OS kernel and the device tree into memory, then point the CPU to it to begin executing the Linux OS
- Das U-Boot, bootloader of choice. **Open-source**, primary **boot loader** used in embedded devices to package the instructions to boot the device's operating system kernel
 - Maintained by Denx Software Engineering, <http://www.denx.de>
 - Used/contributed to by many companies: Intel, NXP, Qualcomm, etc.
 - The code for both the first-stage and second-stage bootloaders for the SoC devices is based on U-Boot
 - U-Boot is GNU General Public License (GPL) licensed (copyleft license)
- SoC FPGA U-Boot source code hosted on [altera-opensource](#) GitHub repository

The Bootloader

USING THE BOOTLOADER

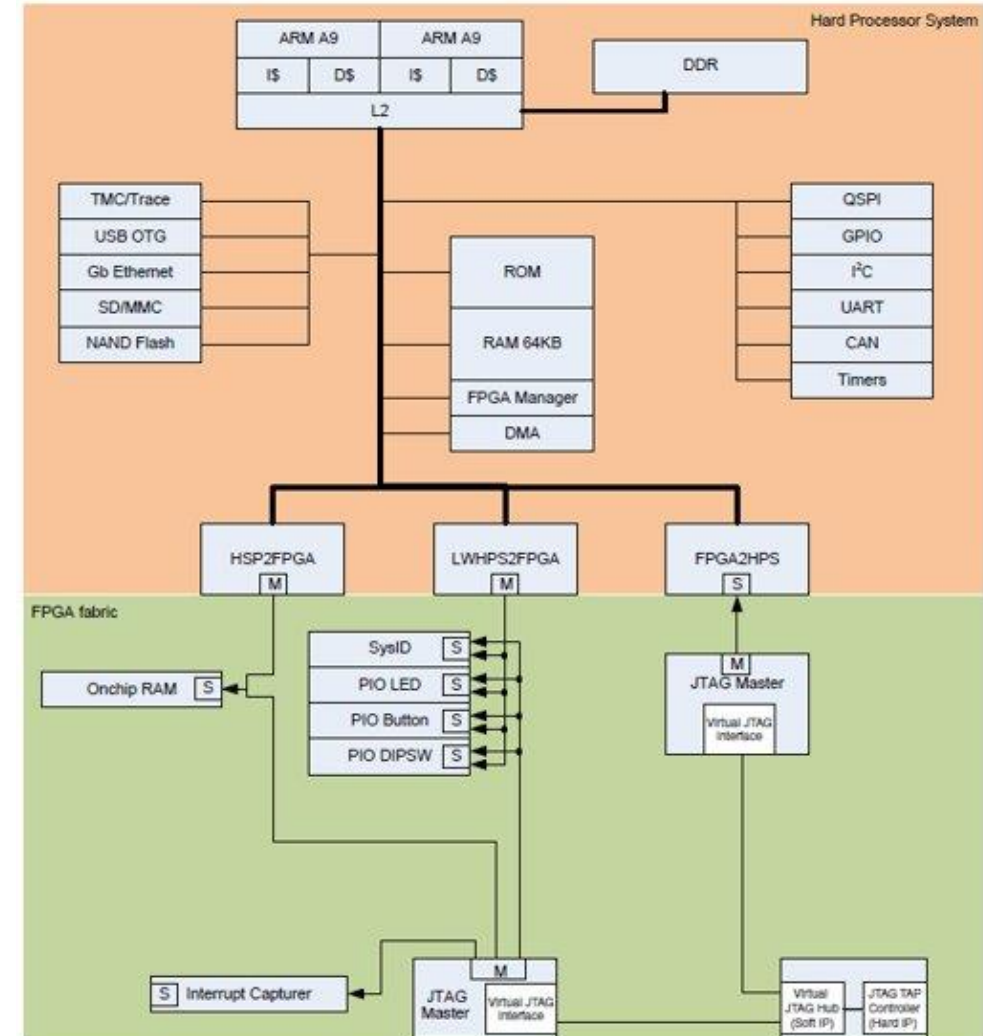
- You can stop the boot process during the U-Boot stage by pressing a key during the boot process
 - Default configuration gives you 5 seconds
- By default, runs commands defined by the environment variable **bootcmd**
- To see the defined environment variables, use the command **printenv**
- To change an environment variable use the command **setenv**
- U-Boot offers many other commands if needed

```
Hit any key to stop autoboot: 0
SOCFPGA_CYCLONE5 # printenv bootcmd
bootcmd=run callscript; run mmcload; run mmcboot
SOCFPGA_CYCLONE5 # printenv fdtimage
fdtimage=socfpga.dtb
SOCFPGA_CYCLONE5 # setenv fdtimage rmem.dtb
SOCFPGA_CYCLONE5 # printenv fdtimage
fdtimage=rmem.dtb
SOCFPGA_CYCLONE5 #
```


The FPGA Image

CONFIGURE THE FPGA

- Extend HPS peripherals with FPGA logic
- FPGA not configured at Power On
- FPGA configured by ARM CortexTM-A9
 - by U-boot before Linux
 - or after Linux boot complete
 - or not at all



The Device tree

WHAT DOES A DEVICE TREE DO?

- Tells the kernel what hardware is present in the system
- Passed in by U-Boot
- Used by the drivers and other functions within the kernel (like memory management) to know how to configure everything
 - Device tree settings (called “bindings”) are documented in the kernel source tree in the Documentation/ directory
 - Documentation/devicetree/bindings/

```
#include "socfpga_cyclone5.dtsi"

/ {
    model = "Altera SOCFPGA Cyclone V SoC Development Kit";
    compatible = "altr,socfpga-cyclone5-socdk", "altr,socfpga-cyclone5", "altr,socfpga";

    chosen {
        bootargs = "earlyprintk";
        stdout-path = "serial0:115200n8";
    };

    memory@0 {
        name = "memory";
        device_type = "memory";
        reg = <0x0 0x40000000>; /* 1GB */
    };

    aliases {
        /* this allow the ethaddr uboot environment variable contents
        * to be added to the gmac1 device tree blob.
        */
        ethernet0 = &gmac1;
    };

    leds {
        compatible = "gpio-leds";
        hps0 {
            label = "hps_led0";
            gpios = <&portb 15 1>;
        };
        hps1 {
            label = "hps_led1";
            gpios = <&portb 14 1>;
        };
        hps2 {
            label = "hps_led2";
            gpios = <&portb 13 1>;
        };
    };
};
```

The Device tree

RECOMMENDED ADDITIONAL READING

● [Rocketboards - How to Create a Device Tree](#)

● [Device Trees for Intel SoC FPGAs](#)

HOWTO Create a Device Tree

Advice on how to create a device tree for SoC FPGA linux environments.

📅 04 Nov 2020 - 05:00 | 🗨 Version 16 | 👤 NagarjunM | 📁 Altera SoC Workshop Series
overlay, [sopc-create-header-files](#), training, tutorial

Required environment.

Overview.

Initial CV SoC DK board setup, tools setup, and other setup.

CV SoC device tree orientation.

Build and test the default CV SoC DK example device tree.

What should the device tree enable in the HPS?

What should the device tree enable in the FPGA?

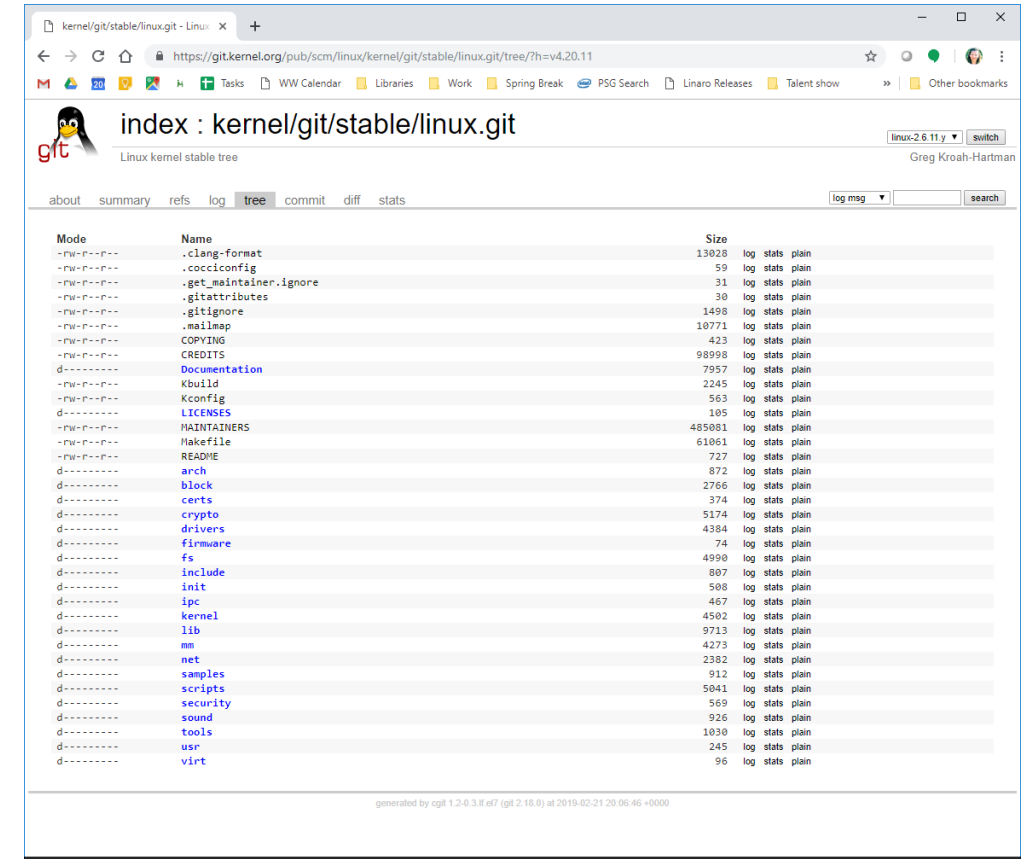
Add the FPGA peripherals to the CV SoC device tree.



The Kernel

WHAT IS THE KERNEL?

- What is often referred to as “Linux” when you grab sources
- Open-source at the Intel FPGA Github or master sources at kernel.org
- Includes memory and process management, virtual filesystem mapping, device drivers, and knowledge of the specific device architecture



The screenshot shows the 'index : kernel/git/stable/linux.git' page on the kernel.org website. The page displays a table of files and directories in the kernel source tree, categorized by mode (e.g., -rw-r--r-- for files, d----- for directories). The table includes columns for Name, Size, log, stats, and plain. The files listed include .clang-format, .cocciconfig, .get_maintainer.ignore, .gitattributes, .gitignore, .mailmap, COPYING, CREDITS, Documentation, Kbuild, Kconfig, LICENSES, MAINTAINERS, Makefile, README, arch, block, certs, crypto, drivers, firmware, fs, include, init, ipc, kernel, lib, mm, net, samples, scripts, security, sound, tools, usr, and virt. The page also features a search bar and a 'log msg' button.

Mode	Name	Size	log	stats	plain
-rw-r--r--	.clang-format	13028	log	stats	plain
-rw-r--r--	.cocciconfig	59	log	stats	plain
-rw-r--r--	.get_maintainer.ignore	31	log	stats	plain
-rw-r--r--	.gitattributes	30	log	stats	plain
-rw-r--r--	.gitignore	1498	log	stats	plain
-rw-r--r--	.mailmap	10771	log	stats	plain
-rw-r--r--	COPYING	423	log	stats	plain
-rw-r--r--	CREDITS	98998	log	stats	plain
d-----	Documentation	7957	log	stats	plain
-rw-r--r--	Kbuild	2245	log	stats	plain
-rw-r--r--	Kconfig	563	log	stats	plain
d-----	LICENSES	105	log	stats	plain
-rw-r--r--	MAINTAINERS	485081	log	stats	plain
-rw-r--r--	Makefile	61061	log	stats	plain
-rw-r--r--	README	727	log	stats	plain
d-----	arch	872	log	stats	plain
d-----	block	2766	log	stats	plain
d-----	certs	374	log	stats	plain
d-----	crypto	5174	log	stats	plain
d-----	drivers	4384	log	stats	plain
d-----	firmware	74	log	stats	plain
d-----	fs	4990	log	stats	plain
d-----	include	807	log	stats	plain
d-----	init	508	log	stats	plain
d-----	ipc	467	log	stats	plain
d-----	kernel	4502	log	stats	plain
d-----	lib	9713	log	stats	plain
d-----	mm	4273	log	stats	plain
d-----	net	2382	log	stats	plain
d-----	samples	912	log	stats	plain
d-----	scripts	5041	log	stats	plain
d-----	security	569	log	stats	plain
d-----	sound	926	log	stats	plain
d-----	tools	1030	log	stats	plain
d-----	usr	245	log	stats	plain
d-----	virt	96	log	stats	plain

The Root Filesystem

WHAT IS THE ROOTFS?

- Linux must always have a root filesystem, which is an initial implementation of the entire file structure
 - In Linux “everything is a file,” so this is essential to the operating system
- Two different ways of implementing a root filesystem
 - On media (SD card, QSPI, etc)
 - U-Boot passes in the location of the file system to Linux
 - As a “ramfs”
 - A **very small** filesystem that can fit into the RAM
 - So you don’t have to have any media device, or so the boot & execution can be very fast
 - Compiled INTO the kernel

A dark blue sports car, possibly a Lotus Evija, is driving on a winding asphalt road that curves through a mountainous landscape. The road is flanked by steep, rocky slopes with patches of snow. In the background, a vast valley with a lake and distant mountains is visible under a bright blue sky with scattered white clouds. The car has 'LOTUS' and 'EVIA' branding on its side and front. The title 'Introduction to Yocto' is overlaid in white text across the middle of the image.

Introduction to Yocto

The Yocto Project

A LINUX BUILD SYSTEM

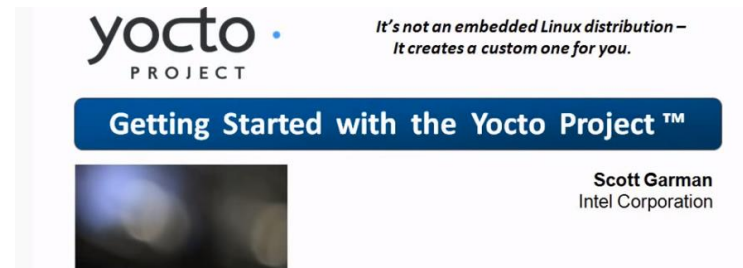
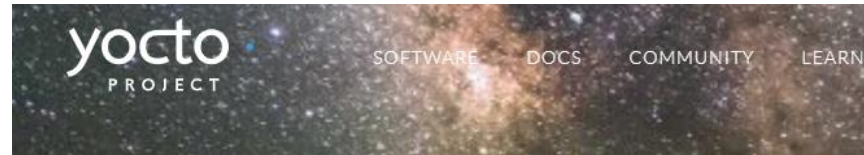
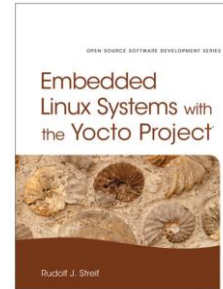
- Enables the rapid, repeatable development of Linux-based embedded systems in which every aspect of the development process can be customized.
- A Linux Foundation collaborative open-source project whose goal is to produce tools and processes that enable the creation of Linux distributions for embedded and IoT software that are independent of the underlying architecture of the embedded hardware.
- Provides
 - Interoperable tools
 - Metadata
 - Processes



Yocto Resources

REFERENCE FOR LEARNING

- Very deep subject. Lots of good reference resources available
- [Embedded Linux Systems with Yocto](#)
- [Yocto Project website](#)
- [Youtube videos](#)



[Getting Started with the Yocto Project](#)

Practical Introduction to Yocto

BUILDING YOCTO

- Introduction by way of an initial build
- Specify a Linux host build system
- Setup the build host
- Install Poky
- Configure the build environment
- Launch the build
- Run the completed Linux image using an emulator

Yocto Build Prerequisites

TYPICAL HOST MACHINE SOFTWARE AND HARDWARE REQUIREMENTS

- PC hardware
 - X86 architecture (32 or 64bit)
 - 4GB RAM (16+ GB better)
 - 50 GB disk space (100 GB better)
 - Good internet connection
- Software (Linux host)
 - CentOS
 - Fedora
 - openSUSE
 - Ubuntu
- A good internet connection (100 Mbit+) is preferred.

Yocto Build Setup

CONFIGURE THE BUILD ENVIRONMENT

- Setup build host
 - Install host dependencies
- Install Poky from Yocto Project website
 - `git clone -b warrior git://git.yoctoproject.org/poky.git`
- Configure the build environment
 - `source oe-init-build-env` (provide snapshot)
 - selection of default build targets available within poky

Yocto Build Launch

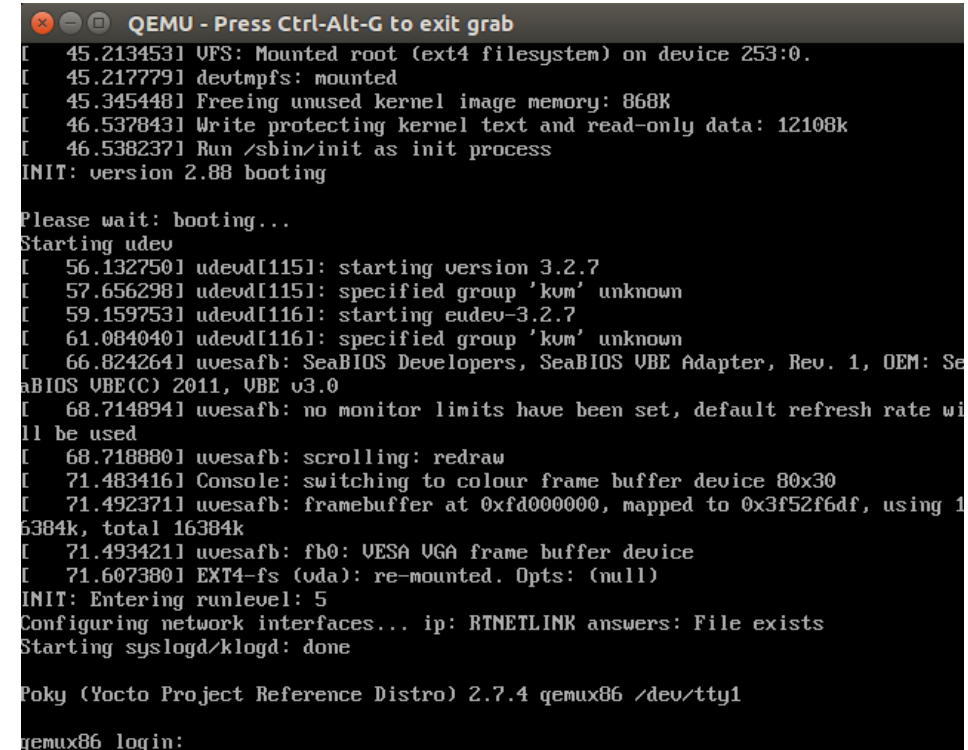
CONFIGURE THE BUILD SETTINGS

- Build configuration (local.conf) variable settings
 - BB_NUMBER_THREADS
 - MACHINE
- Launch the build
 - bitbake <build-target> e.g. bitbake core-image-minimal

Yocto Build Verify

QEMU X86 EMULATOR

- Target machine is an emulated system
- Launch the Linux emulation using QEMU
 - `$ runqemu qemux86`
- Fully interactive Linux system
 - Network available
 - Develop and debug user space applications



```
QEMU - Press Ctrl-Alt-G to exit grab
[ 45.213453] UFS: Mounted root (ext4 filesystem) on device 253:0.
[ 45.217779] devtmpfs: mounted
[ 45.345448] Freeing unused kernel image memory: 868K
[ 46.537843] Write protecting kernel text and read-only data: 12108k
[ 46.538237] Run /sbin/init as init process
INIT: version 2.88 booting

Please wait: booting...
Starting udev
[ 56.132750] udevd[1151]: starting version 3.2.7
[ 57.656298] udevd[1151]: specified group 'kvm' unknown
[ 59.159753] udevd[1161]: starting eudev-3.2.7
[ 61.084040] udevd[1161]: specified group 'kvm' unknown
[ 66.824264] uvesafb: SeaBIOS Developers, SeaBIOS VBE Adapter, Rev. 1, OEM: SeaBIOS VBE(C) 2011, VBE v3.0
[ 68.714894] uvesafb: no monitor limits have been set, default refresh rate will be used
[ 68.718880] uvesafb: scrolling: redraw
[ 71.483416] Console: switching to colour frame buffer device 80x30
[ 71.492371] uvesafb: framebuffer at 0xfd000000, mapped to 0x3f52f6df, using 16384k, total 16384k
[ 71.493421] uvesafb: fb0: VESA UGA frame buffer device
[ 71.607380] EXT4-fs (vda): re-mounted. Opts: (null)
INIT: Entering runlevel: 5
Configuring network interfaces... ip: RTNETLINK answers: File exists
Starting syslogd/klogd: done

Poky (Yocto Project Reference Distro) 2.7.4 qemux86 /dev/tty1
qemux86 login:
```

Workshop

SECTION ONE

- Build a Linux image and execute it with the QEMU emulator
- Complete sections 1 and 2
 1. Introduction to Yocto
 - i. Workshop goals
 - ii. Recommended References and Reading
 2. Getting started
 - i. Clone Poky
 - ii. Initialize the Build Environment
 - iii. Build the Image
 - iv. Simulate Your Image Using QEMU



Yocto Project

Yocto Project

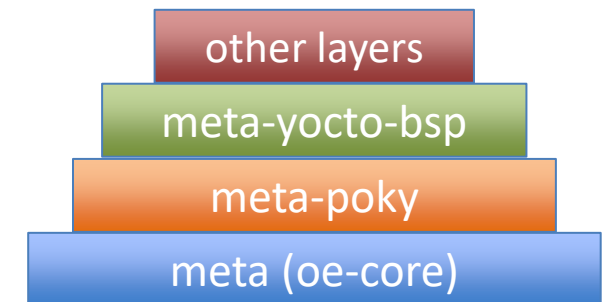
BUILDING YOCTO

- The Yocto Project Overview
- OpenEmbedded
- BitBake
- Poky Linux
- Yocto Linux
- Layers

Yocto Project Overview

BUILDING YOCTO

- Collection of tools and methods enabling
 - Rapid evaluation of embedded Linux on many popular off-the-shelf boards
 - Easy customization of distribution characteristics
- Supports x86, ARM, MIPS, Power
- Based on technology from the [OpenEmbedded Project](#)
- Layer architecture allows for easy re-use of code



Yocto Project Overview

BUILDING YOCTO

- Yocto Project builds packages - then uses these packages to build bootable images
- Supports use of popular package formats including
 - Rpm, deb, ipk
- Releases on a 6 months cadence
- Latest (stable) kernel, toolchain and packages, documentation
- Application Development Tools including Eclipse plugin, SDK, toaster

Yocto Project Overview

YOCTO IS BASED ON THE OPENEMBEDDED CORE

The screenshot shows the OpenEmbedded Core repository tree view on the cggit website. The browser address bar shows the URL `cggit.openembedded.org/cgit.cgi/openembedded-core/tree/meta?h=master`. The page title is "index : openembedded-core". The OpenEmbedded logo is visible on the left. The page shows the "tree" view of the repository, with the path "root/meta" selected. The table lists the contents of the "meta" directory, including files like COPYING.GPLv2, COPYING.MIT, classes, conf, files, lib, and various recipe directories (recipes-bsp, recipes-connectivity, recipes-core, recipes-devtools, recipes-extended, recipes-gnome, recipes-graphics, recipes-kernel, recipes-lsb4, recipes-multimedia, recipes-qt, recipes-rt, recipes-sato, recipes-support, recipes.txt) and a site directory. The table also shows the size of each file/directory and the log file path.

Mode	Name	Size
-rw-r--r--	COPYING.GPLv2	17987 log plain
-rw-r--r--	COPYING.MIT	1035 log plain
d-----	classes	7261 log plain
d-----	conf	794 log plain
d-----	files	236 log plain
d-----	lib	60 log plain
d-----	recipes-bsp	728 log plain
d-----	recipes-connectivity	850 log plain
d-----	recipes-core	1307 log plain
d-----	recipes-devtools	3174 log plain
d-----	recipes-extended	2809 log plain
d-----	recipes-gnome	477 log plain
d-----	recipes-graphics	1694 log plain
d-----	recipes-kernel	675 log plain
d-----	recipes-lsb4	64 log plain
d-----	recipes-multimedia	745 log plain
d-----	recipes-qt	248 log plain
d-----	recipes-rt	102 log plain
d-----	recipes-sato	948 log plain
d-----	recipes-support	2161 log plain
-rw-r--r--	recipes.txt	1407 log plain
d-----	site	1506 log plain


generated by cggit v0.9.2-21-gd82e at 2014-08-19 14:31:48 (GMT)

Metadata describing approximately 1000 "core" recipes used for building boot images. Includes support for graphics, Qt, networking, kernel recipes, tools, much more.

OpenEmbedded Build System

INTRODUCTION

- The OpenEmbedded Project co-maintains OE-core build system
 - bitbake build tool and scripts
 - Metadata and configuration
- Acts as a central point for adding new metadata (OE layer index)

OpenEmbedded Layer Index			
Branch: master ▾			
Layers Recipes Machines Classes Distros			
Search layers 			
Filter layers ▾			
Layer name	Description	Type	Repository
openembedded-core	Core metadata	Base	git://git.openembedded.org/openembedded-core
meta-oe	Additional shared OE metadata	Base	git://git.openembedded.org/meta-openembedded
de-ensc-bpi-router	router + telephony bsp	Machine (BSP)	https://gitlab.com/ensc-groups/bpi-router/de.ensc.bpi-router
e100-bsp	Ettus E1XX series BSP	Machine (BSP)	git://github.com/EttusResearch/meta-ettus.git
e300-bsp	Ettus E3XX Series BSP	Machine (BSP)	https://github.com/EttusResearch/meta-ettus.git

OpenEmbedded Build System

WHAT IS BITBAKE?

• Bitbake

- Powerful and flexible build engine based on Python
- Reads metadata
- Determines dependencies
- Schedules tasks

• Metadata

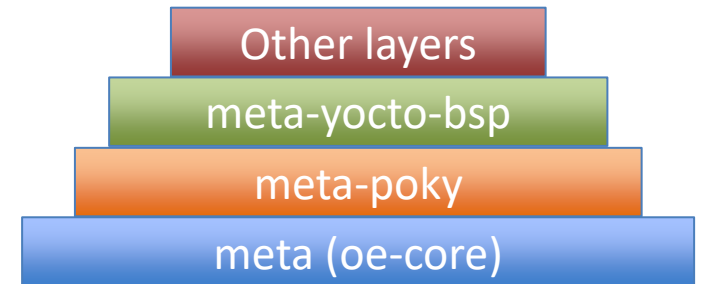
- Structured collection of recipes which tell Bitbake what to build
- Organized in layers



Poky

WHAT IS POKY?

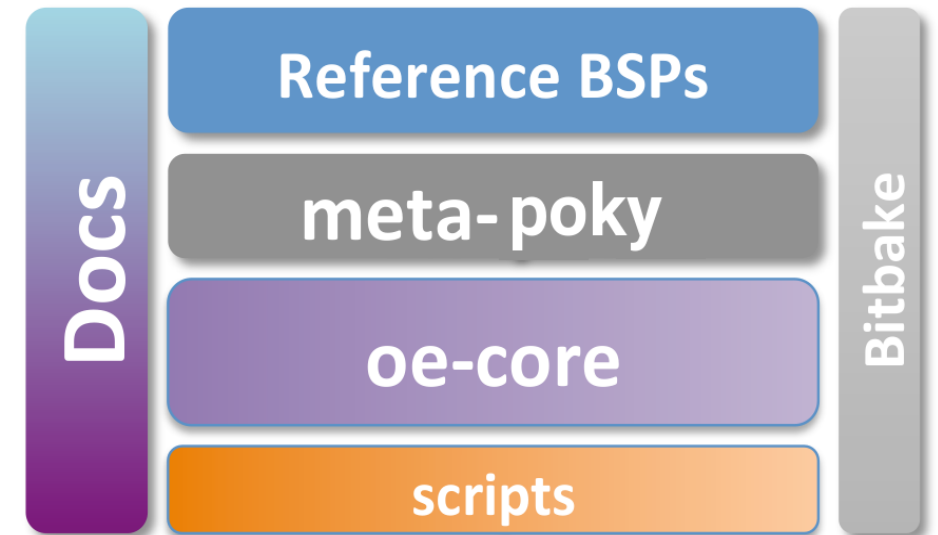
- Poky is a reference distribution and has its own git repo
 - `git clone git://git.yoctoproject.org/poky`
- Primary Poky layers include
 - oe-core (meta)
 - meta-poky
 - meta-yocto-bsp
- Poky is where we begin building with the Yocto Project



Poky

DETAIL

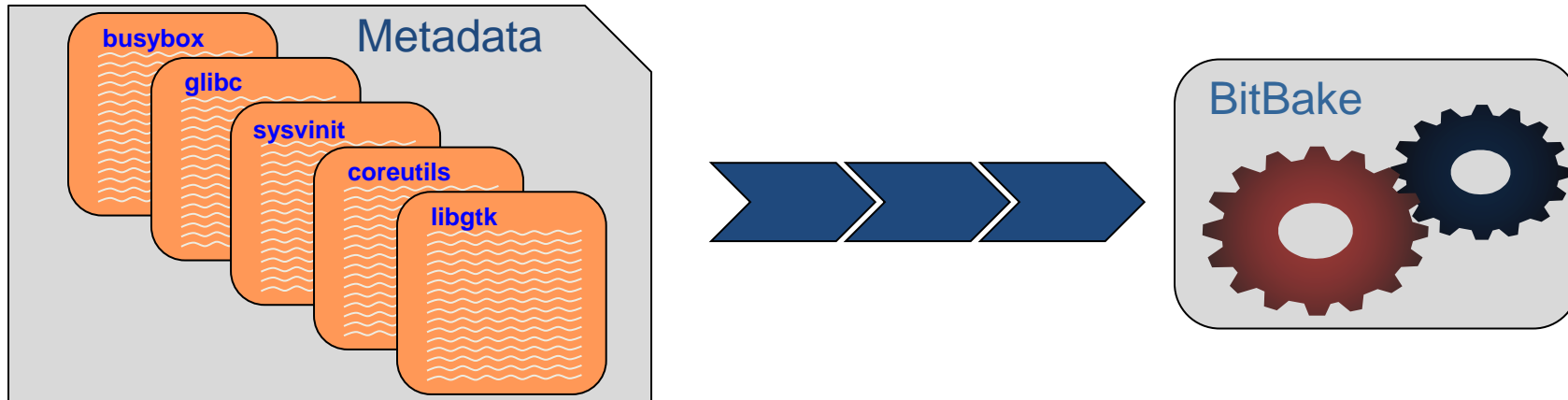
- Includes the following core components
 - Bitbake tool: Python based build engine
 - Build scripts
 - Foundation package recipes (oe-core)
 - meta-poky (includes distribution policy)
 - Some reference BSPs
 - Yocto Project documentation



Metadata and Bitbake

BUILDING WITH RECIPES

- The most common metadata are recipes
- Recipes provide a “list of ingredients” and “cooking instructions”
- Defines settings and a set of tasks used by Bitbake to build binary packages



Metadata

WHAT IS IT?

- There are four main categories of metadata
 - Recipes (*.bb). Describe build instructions for a single package
 - PackageGroups (special *.bb). Used to group packages together for a File System Image
 - Classes (*.bbclass). Inheritance mechanism for common functionality
 - Configuration (*.conf). Drives the overall behavior of the build process
- Additional type of metadata
 - Append files (*.bbappend). Defines additional metadata for a similarly named .bb file. Can add to or override previously set values
 - Include files (*.inc)

Bitbake

INTRODUCTION TO BITBAKE

- Bitbake is a task executor and scheduler
- By default the build task for the specified recipe is executed
 - `bitbake myrecipe`
- You can indicate which task you want run
 - `bitbake -c clean myrecipe`
 - `bitbake -c cleanall myrecipe`
- You can get a list of tasks with
 - `bitbake -c listtasks myrecipe`

Bitbake

BUILDING RECIPES

- By default the highest version of a recipe is built
 - `bitbake myrecipe`
- You can specify the version of the package you want built
 - `bitbake myrecipe-1.0`
- You can also build a particular revision of the package metadata
 - `bitbake myrecipe-1.0-r0`
- Or you can provide a recipe file to build
 - `bitbake -b mydir/myrecip.bb`

Bitbake

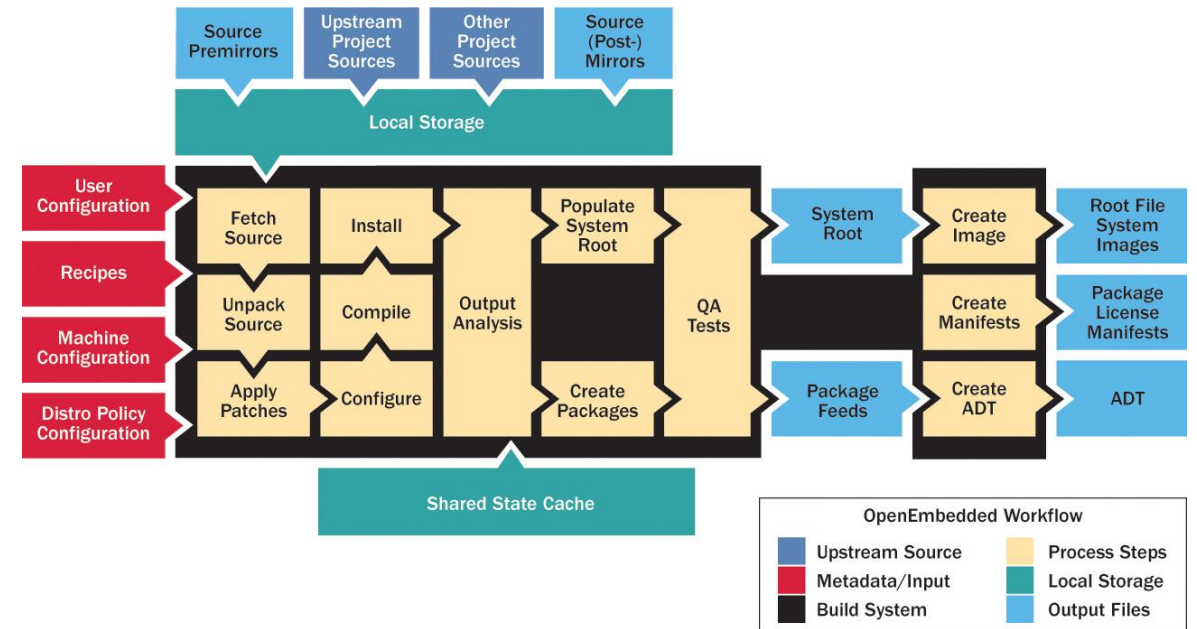
RUNNING BITBAKE FOR THE FIRST TIME

- When you do a really big build, running with `--continue (-k)` means bitbake will proceed as far as possible after finding an error
 - `bitbake -k core-image-minimal`
 - When running a long build (e.g. overnight) you want as much of the build done as possible before debugging issues
- Running bitbake normally will stop on the first error found
 - `bitbake core-image-minimal`

Building Open Source Software Packages

BITBAKE BUILD FLOW

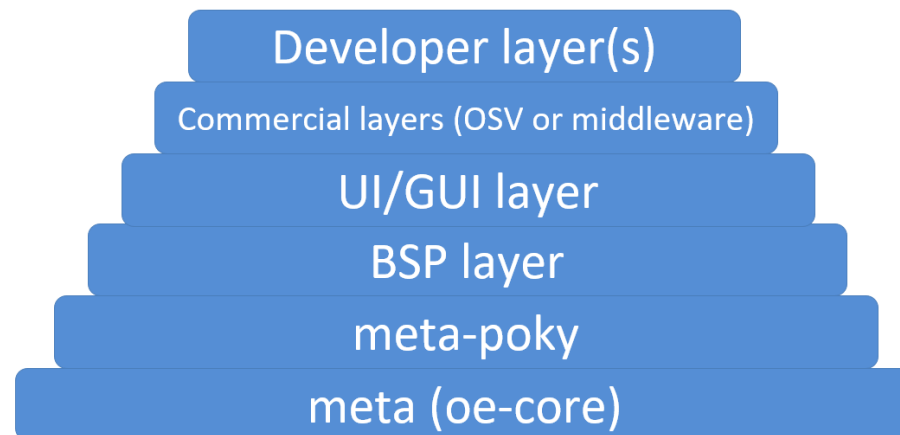
- **1. Fetch:** Obtain the source code.
- **2. Extract:** Unpack the source code.
- **3. Patch:** Apply patches for bug fixes and added functionality.
- **4. Configure:** Prepare the build process according to the environment.
- **5. Build:** Compile and link.
- **6. Install:** Copy binaries and auxiliary files to their target directories.
- **7. Package:** Bundle binaries and auxiliary files for installation on other systems.



Layers

INFLUENCING BUILD ARCHITECTURES

- Metadata is provided in a series of layers which allow the developer to override any value without editing the originally provided files.
- A layer is a logical collection of metadata in the form of recipes
- A layer can represent the oe-core, a Board support package, an application stack or user developed code
- All layers have a priority and can override policy, metadata and config settings of layers of lesser priority



Recommended Reference

- The material in this section on Yocto has been taken from the presentation linked below. It provides a lot more detail and is recommended reading.

Yocto Project Developer Day
Intro to Yocto Project



Creating a Custom Embedded Linux
Distribution for Any Embedded Device
Using the Yocto Project



Jan-Simon Möller
Behan Webster
The Linux Foundation
Oct 26, 2017
(CC BY-SA 4.0)

<https://www.yoctoproject.org/learn-items/devday-prague-2017/>

Workshop

SECTION TWO

- Build a Linux image using the meta-altera layer and execute it on
 - DE10-nano development kit or
 - SoCKit development kit
- Complete [section 3](#)
 3. Customizing your Build for Specific Hardware
 - i. Clone the Layer
 - ii. Change the Configuration to Build for a Specific Machine
 - iii. Change the Configuration to Build for a Specific Linux Kernel
 - iv. Change the Configuration to set the number of Processor Threads
 - v. Add Your Layer to the Layer Configuration File

A dark blue sports car, possibly a Lotus Evija, is driving on a winding asphalt road that curves through a mountainous landscape. The road is flanked by steep, rocky slopes with patches of snow. In the background, a vast valley with a lake and distant mountains is visible under a bright blue sky with scattered white clouds. The car has several sponsor logos, including 'NOVUS' and 'FRANK' on the front, and 'GRAND TOURING' on the windshield.

Build a custom Linux BSP with Yocto

Build a Custom Linux BSP with Yocto

BUILDING YOCTO

- Yocto BSP
- System architecture considerations
- Create a custom layer

Yocto BSP

WHAT IS IT?

“Essentially, a Yocto Project BSP is nothing but a specialized metadata layer that includes additions and modifications to the core layers for the support of the target hardware.”

Embedded Linux Systems with the Yocto Project, Streif, Rudolf J., Prentice Hall

System Architecture Considerations

BOOT CUSTOMIZATION OPTIONS








- FPGA configuration options
 - Configure before Linux loads?
 - Configure after Linux loads?
 - Support devicetree overlays?
- Bootloader options
 - Boot sequence
 - U-boot runtime options
- Linux drivers?
- Root File System packages?
- Linux initialization?

Create a Custom layer

WHAT DOES THE LAYER NEED TO INCLUDE?

- Layer and MACHINE configuration information
- Hardware specific to the custom board
 - Custom FPGA image
 - Custom Bootloader configuration
 - Custom Linux configuration
 - Custom Bitbake build images
 - Custom SD card configuration
- License file
- README file



meta-my-custom-soc-board	
Name	
	conf
	recipes-bsp
	recipes-images
	recipes-kernel
	wic
	COPYING.MIT
	README.md

Layer Configuration file

CONF/LAYER.CONF

- Make the current layer searchable by Bitbake
 - `BBPATH .= ":{LAYERDIR}"`
 - `BBFILES += "${LAYERDIR}/recipes*/*/*.bb \`
`${LAYERDIR}/recipes*/*/*.bbappend"`
- Define the name of the layer and set its priority level.
 - `BBFILE_COLLECTIONS += "meta-my-custom-soc-board"`
 - `BBFILE_PATTERN_meta-my-custom-soc-board := "^${LAYERDIR}/"`
 - `BBFILE_PRIORITY_meta-my-custom-soc-board = "7"`
 - `LAYERSERIES_COMPAT_meta-my-custom-soc-board = "warrior"`

MACHINE Configuration file

CONF/MACHINE/MY-CUSTOM-SOC-BOARD.CONF

- Distinguishes a BSP layer from a regular layer
- Creates a new MACHINE: my-custom-soc-board
- Uses include file hierarchy to define socfpga specific build and run-time parameters
 - Selects socfpga version of Linux kernel (linux-altera)
 - Selects socfpga version of U-boot (u-boot-socfpga)
 - Defines the U-boot load address and runtime entry point in memory
 - Targets a specific cpu architecture (cortexa9)
- Specifies the U-boot build configuration (defconfig) via board selection.
 - Defined by U-boot recipe include file extension

MACHINE Configuration file

CONF/MACHINE/MY-CUSTOM-SOC-BOARD.CONF

- Defines Linux boot arguments
 - SERIAL_CONSOLES ?= "115200;ttyS0"
 - UBOOT_EXTLINUX_KERNEL_ARGS_default ?= "rootwait rw earlycon"
 - UBOOT_EXTLINUX_ROOT_default ?= "root=/dev/mmcblk0p3"
- Creates the extlinux.conf bootloader configuration file. U-boot specific boot parameters to boot Linux
 - Specifies the location of the kernel
 - Defines which Linux devicetree to load.
 - Includes the Linux boot arguments.

MACHINE Configuration file

CONF/MACHINE/MY-CUSTOM-SOC-BOARD.CONF

- Specifies SD card image boot files to include on SD card
 - `IMAGE_BOOT_FILES ?= " \`
`${FPGA_CONFIG} \`
`${KERNEL_DEVICETREE} \`
`${KERNEL_IMAGETYPE} \`
`extlinux.conf;extlinux/extlinux.conf \`
- Specifies the SD card image partitions
 - Uses the WIC methodology for creating images
 - Partitions specified in `sdimage-cyclone5-arria5.wks` file (discussed later)

FPGA Configuration Recipe

RECIPES-BSP/RBF

- FPGA must be configured with an RBF file at or after system Power On
 - RBF file was created in the Hardware workshop
 - RBF file is placed by the developer in the recipes-bsp/rbf/files subdirectory
- FPGA RBF recipe
 - Creates an SD card “images” \$WORKDIR directory
 - Installs the RBF file in the “images” \$WORKDIR directory
 - Later copies the RBF file to the “images” \$DEPLOY_DIR directory
- Placed by exlinux.conf in the FAT partition of the SD card.
- U-boot at run time loads the RBF file into the FPGA

Bootloader Configuration Recipe

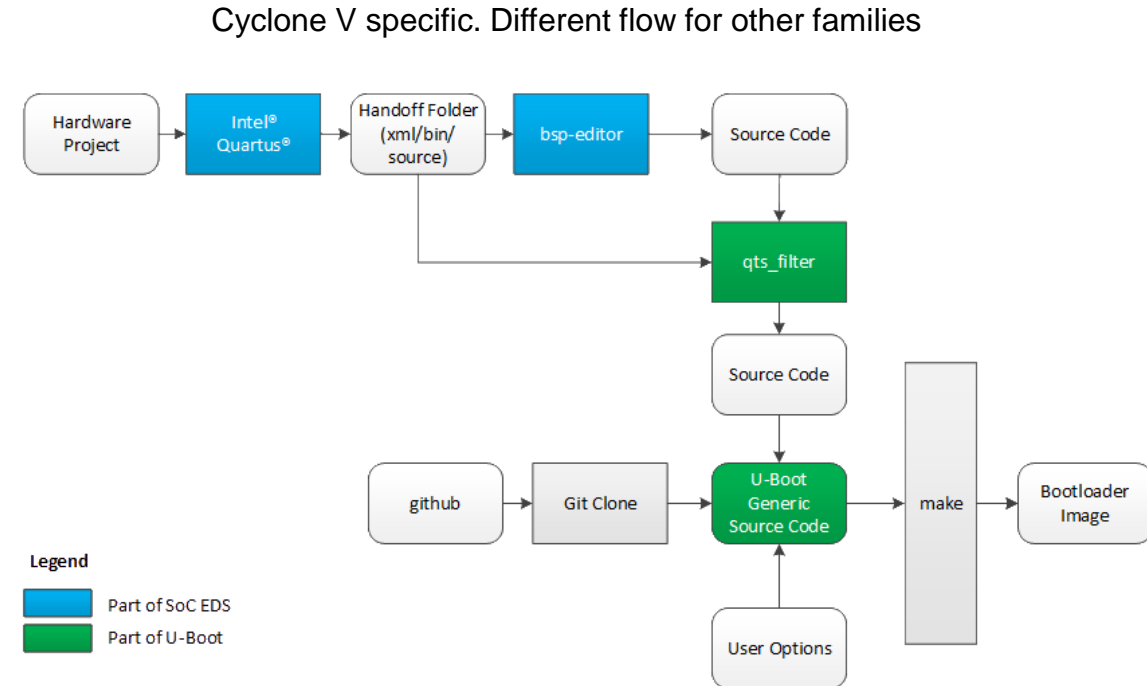
REVIEW SOC FPGA BOOT STAGES

Stages	Where is it stored?	Where does it run from?	What is its purpose?
Reset	N/A	N/A	Putting device in a known state, starts running from reset vector
Boot Preparation	Intel® Stratix® 10 and Agilex™ SoCs: controlled by SDM, others: Boot ROM	Intel® Stratix® 10 and Agilex™ SoCs: SDM subsystem, others: Boot ROM	Places first-stage bootloader in on-chip RAM
1 st Stage Bootloader (FSBL)*	Intel® Stratix® 10 and Agilex™ SoCs: QSPI attached to SDM, others: configurable, SD is default	On-Chip RAM within the HPS	Setting up the SDRAM and the interface where the SSBL is stored
2 nd Stage Bootloader (SSBL)	Configurable, SD Card is default	SDRAM	Initializes a robust set of peripherals to boot Linux* OS from, and provides early user functions
Linux* OS	Configurable, SD Card is default	SDRAM	Embedded OS
Application	Configurable, SD Card is default	SDRAM	Whatever a user can code up!

Bootloader Configuration Recipe

REVIEW THE U-BOOT BUILD FLOW FOR SOC FPGAS

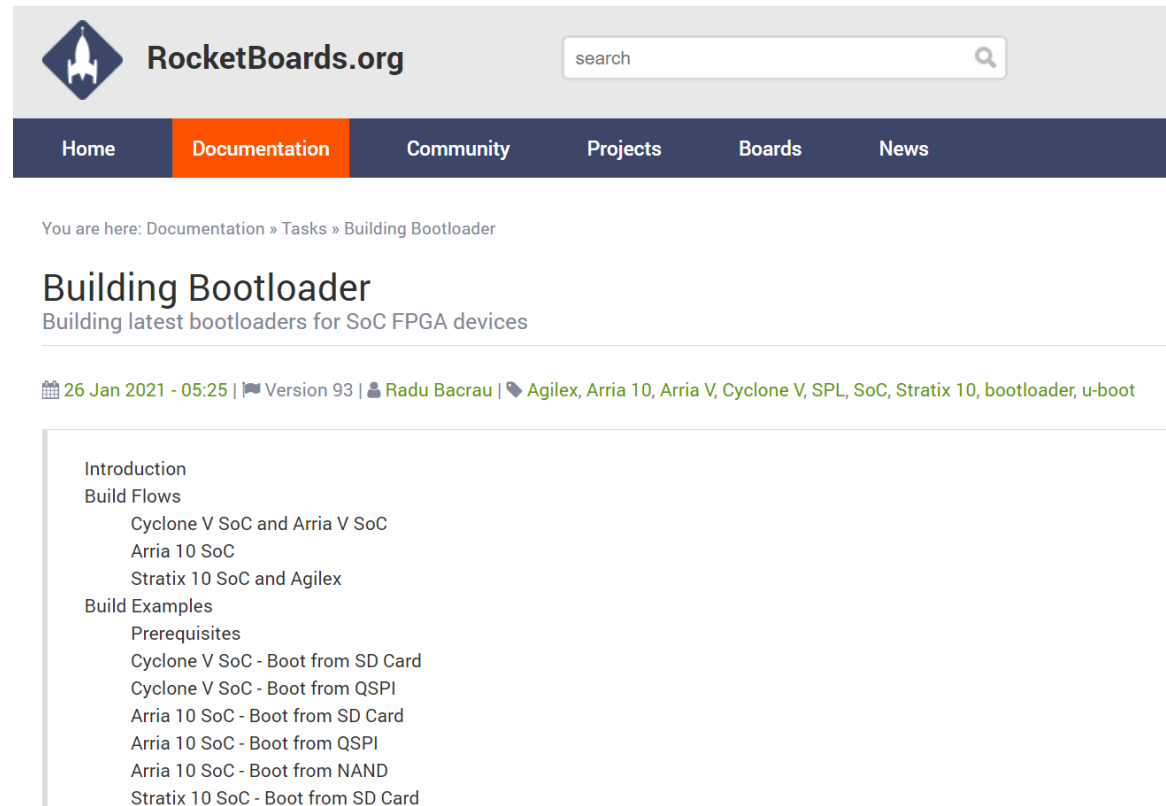
- Quartus creates a Handoff Folder based on Platform Designer, HPS customization
 - DDR , IO & Peripheral selection
- bsp-editor converts it to custom source code
- qts_filter packages custom source code into U-Boot friendly format
- Generic U-Boot source code cloned from GitHub.
- Generic and custom source code combine to create Bootloader image.



Bootloader Configuration Recipe

RECOMMENDED READING

● [Rocketboards – Building Bootloader](#)



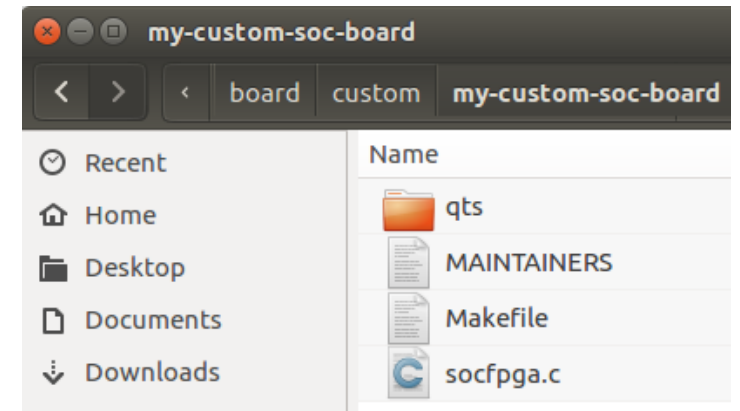
The screenshot shows the RocketBoards.org website. The header includes the RocketBoards.org logo, a search bar, and a navigation menu with links to Home, Documentation (highlighted), Community, Projects, Boards, and News. Below the navigation menu, a breadcrumb trail reads: "You are here: Documentation » Tasks » Building Bootloader". The main heading is "Building Bootloader" with the subtitle "Building latest bootloaders for SoC FPGA devices". Below this, a metadata line shows the date "26 Jan 2021 - 05:25", version "Version 93", author "Radu Bacrau", and tags "Agilex, Arria 10, Arria V, Cyclone V, SPL, SoC, Stratix 10, bootloader, u-boot". The content area lists the following topics:

- Introduction
- Build Flows
 - Cyclone V SoC and Arria V SoC
 - Arria 10 SoC
 - Stratix 10 SoC and Agilex
- Build Examples
 - Prerequisites
 - Cyclone V SoC - Boot from SD Card
 - Cyclone V SoC - Boot from QSPI
 - Arria 10 SoC - Boot from SD Card
 - Arria 10 SoC - Boot from QSPI
 - Arria 10 SoC - Boot from NAND
 - Stratix 10 SoC - Boot from SD Card

Bootloader Configuration Recipe

U-BOOT SUPPORT FOR CUSTOM BOARDS

- Source code structure supports addition of custom boards
- Structure requirements are specific to SoC FPGA families (Stratix 10, Arria 10, Cyclone V)
- Support for custom Cyclone V SOC FPGA board requires the following changes or additions to the source code
- Add a new board directory
 - board/custom/my-custom-soc-board
 - move the qts_filter generated files into a qts sub-folder



Bootloader Configuration Recipe

U-BOOT SUPPORT FOR CUSTOM BOARDS

• Add a custom U-boot defconfig file

- configs/socfpga_my_custom_soc_board_defconfig
- referenced in MACHINE configuration file
- defines custom board specific U-boot build flow
- defines custom board TARGET
- specifies U-boot and Linux device tree filenames
- specifies U-boot runtime boot command
 - copy FPGA RBF from SD card into DRAM
 - configure the FPGA and enable the HPS bridges
 - complete loading the Linux device tree and kernel and boot

```
11 CONFIG_USE_BOOTCOMMAND=y
12 CONFIG_BOOTCOMMAND="fpgadata=0x20000000; fatload mmc 0:1
```

Bootloader Configuration Recipe

U-BOOT SUPPORT FOR CUSTOM BOARDS

- **Modify the U-boot device tree Makefile**
 - arch/arm/dts/Makefile
 - add socfpga_cyclone5_my_custom_soc_board.dtb as a make target
- **Add U-boot device tree**
 - arch/arm/dts/socfpga_cyclone5_my_custom_soc_board.dts
 - device tree used by U-boot for its own boot process

Bootloader Configuration Recipe

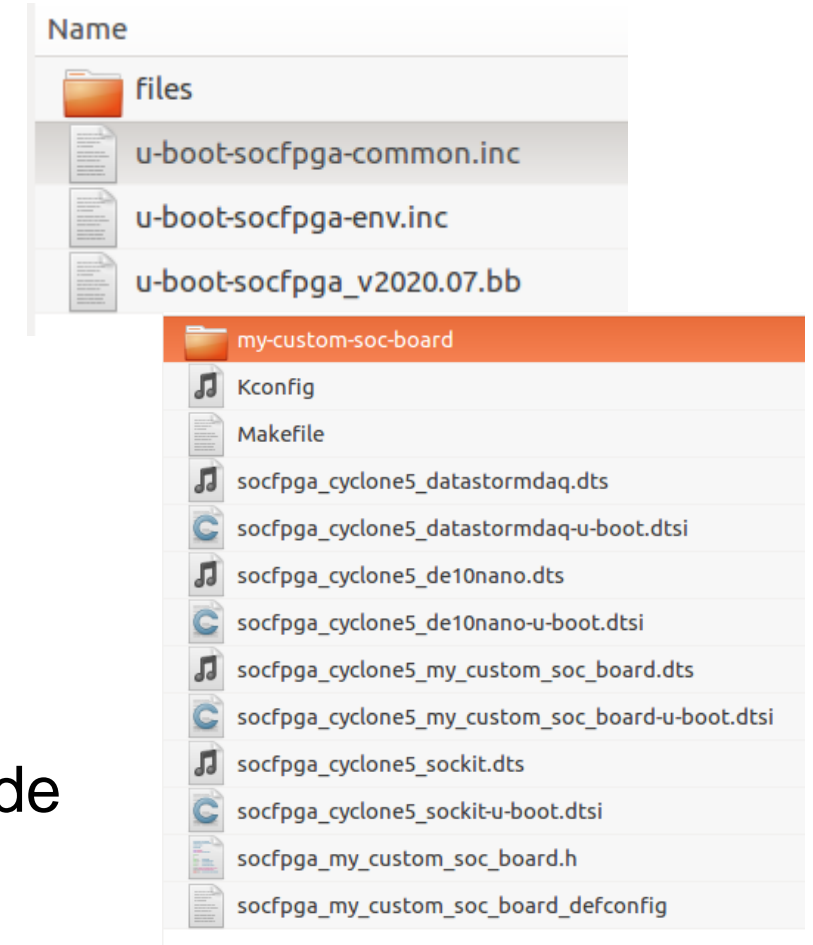
U-BOOT SUPPORT FOR CUSTOM BOARDS

- Add U-boot board specific configuration header file
 - include/configs/socfpga_cyclone_my_custom_soc_board.h
 - define SDRAM physical size
 - define Linux LOADADDR
- Modify SOCFPGA Machine Kconfig build configuration
 - arch/arm/mach-socfpga
 - adds custom board TARGET
 - adds PATH to qts folder for custom board

Bootloader Configuration Recipe

RECIPES-BSP/U-BOOT

- custom board files in files sub-directory
- u-boot-socfpga_v2020.07.bb
 - custom board u-boot recipe
 - includes u-boot-socfpga-common.inc which defines
 - u-boot source code location
 - custom u-boot board targets
 - U-boot source code revision
 - creates a new board sub-folder in U-boot source code structure
 - defines rules required to move custom boards files into appropriate U-boot source code sub directories



Linux Configuration Recipe

KERNEL CONFIGURATION

- Kernel is modular
 - Modules can be loaded at runtime or
 - Modules can be compiled into the kernel
 - Required for hardware that must be available at boot time
 - Choice affects boot time and footprint size

Linux Configuration Recipe

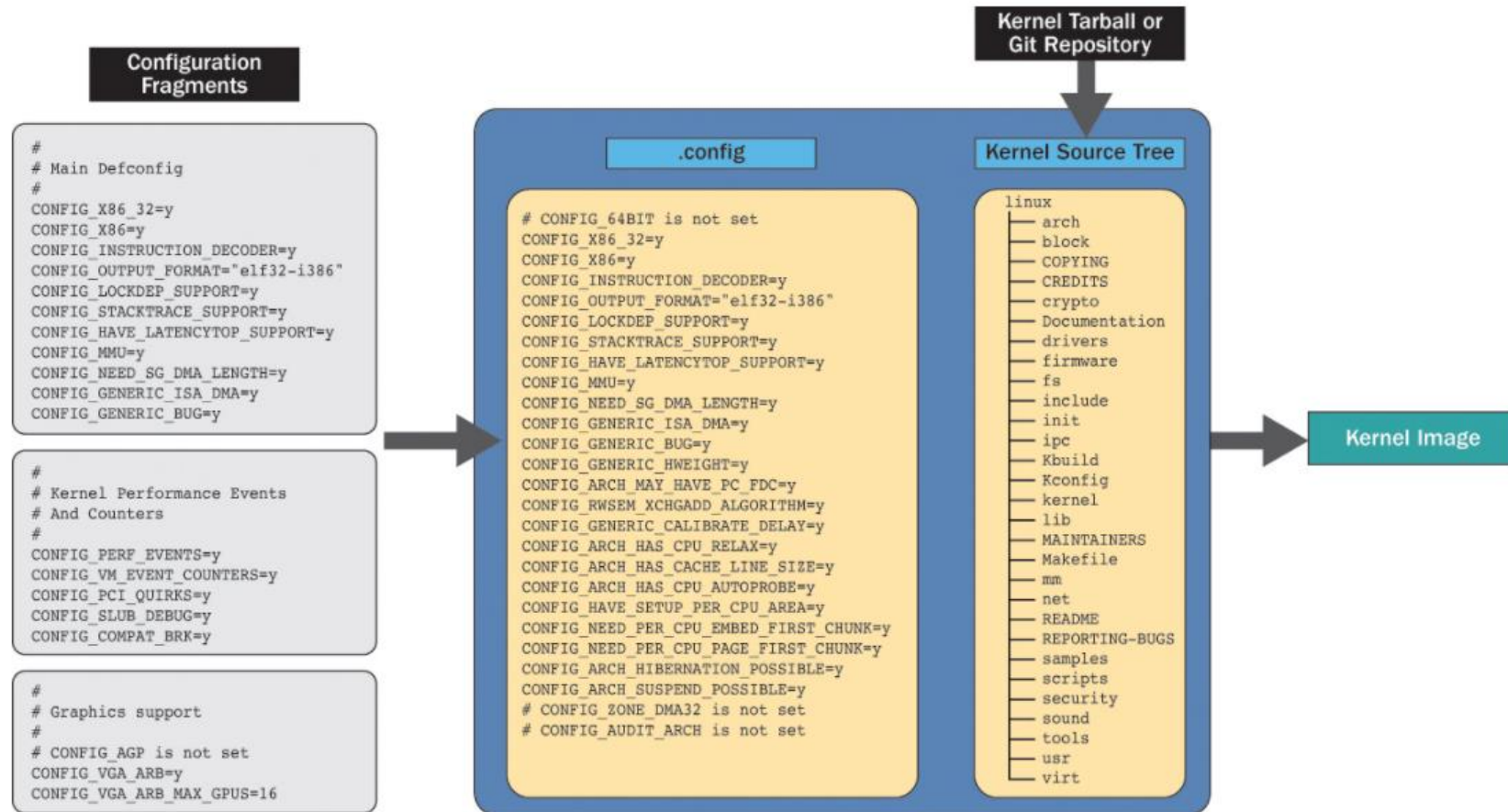
KERNEL CONFIGURATION SYSTEM

- Kernel has its own configuration system – kconfig
- kconfig is a configuration database
 - Organized in a tree structure
 - All configuration options stored in a single .config file
 - Settings in .config are distributed by kernel build system to all kernel source files
 - Kernel settings can be modified using
 - Menuconfig graphical utility
 - Using a board / architecture specific defconfig file (arch/arm/configs)
 - socfpga_defconfig for all Intel FPGA SoC devices

Linux Configuration Recipe

DEFCONFIGS AND CONFIGURATION FRAGMENTS

- .config can be modified by merging with configuration fragment files



Linux Configuration Recipe

DEVICE TREE

- Data structure describing the custom hardware platform
 - Memory address space
 - I/O addresses
 - Interrupt values
 - Runtime configuration options
 - 1080p vs 720p for video driver
- Source file is human readable (dts)
- Device Tree Compiler (DTC) converts it to a binary format (dtb)
- Examined by the kernel at boot time

Linux Configuration Recipe

RECIPES-KERNEL/LINUX

- Hierarchy of recipes and include files
- linux-altera-configs.inc
 - specifies the build system defconfig (socfpga_defconfig)
- linux-altera.inc
 - specifies the kernel source details including repository location
- linux-altera-ltsi_4.14.130.bb
 - Specifies Linux kernel version and suffix (4.14.130-ltsi)
- linux-altera-ltsi_4.14%.bbappend

Linux Configuration Recipe

RECIPES-KERNEL/LINUX

- linux-altera-ltsi_4.14%.bbappend
 - specifies the location of the custom board device tree source files
 - specifies the location of the kernel configuration fragment files
 - moves the device tree source files to the required build directory
 - arch/arm/boot/dts

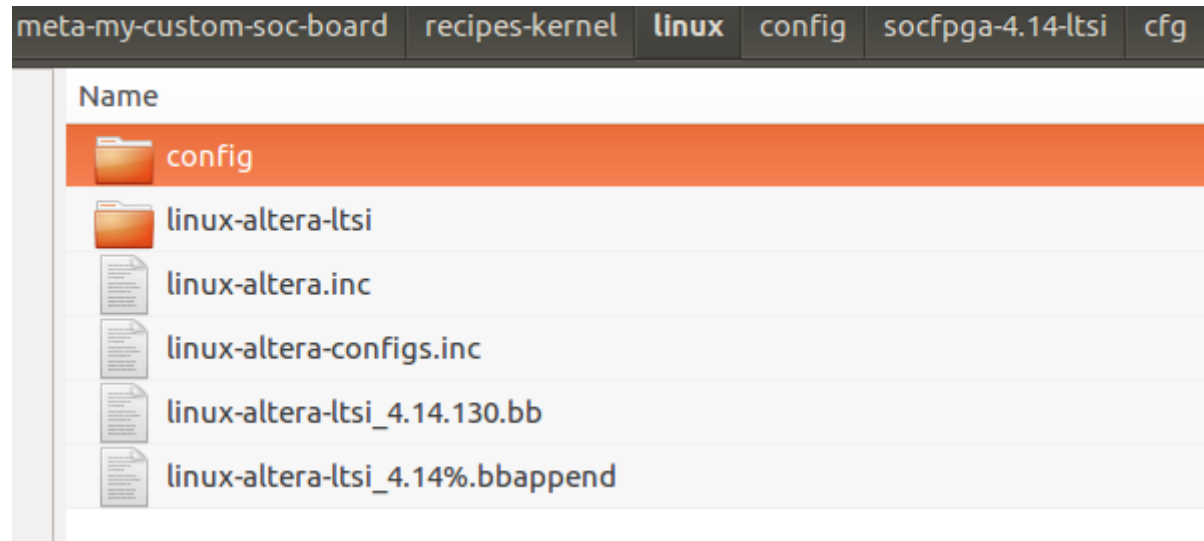


Image Recipe

YOCTO DISTRIBUTION IMAGES

- Yocto and OpenEmbedded include a number of example images
- Offer root filesystem configurations for typical Linux OS stacks
- Range from simple that boot to a command line to complex that include X11 server and graphical user interface.
- Referred to as *core-images*
- Core images are a good starting point for creating custom images

Image Recipe

CUSTOM IMAGE RECIPE

- Packages or package groups can be added to a core image using IMAGE_INSTALL
- New image recipes created for the custom board are placed in
 - recipes-images/yocto sub-directory
- New images referenced in bitbake build
 - bitbake console-image

```
1 inherit core-image
2
3 IMAGE_INSTALL += " \
4     u-boot-socfpga \
5     my-custom-soc-board-rbf \
6     bash \
7     ethtool \
8     gcc \
9     gdb \
10    gdbserver \
11    i2c-tools \
12    iw \
13    kernel-dev \
14    kernel-image \
15    kernel-modules \
16    net-tools \
17    nfs-utils-client \
18    openssh \
19    python \
20    tar \
21    usbutils \
22    vim \
23    vim-vimrc \
24 "
25
26 export IMAGE_BASENAME = "console-image"
```


Image Recipe

REUSE EXISTING RECIPES/PACKAGES

- Existing packages available at [OpenEmbedded](#)
- Select corresponding branch (Warrior)
- Add layer to /conf/bblayers.conf

OpenEmbedded Layer Index

Branch: warrior ▾

Layers

Recipes

Machines

Classes

Distros

snmp

search

Recipe name	Version	Description	Layer
gsnmp	0.3.0	SNMP library written on top of glib and gnet.	meta-montavista-cgl
luci-app-snmpd	1.0	Net-SNMP LuCI interface	meta-tanowrt

SD Card Configuration

CREATE A PARTITIONED IMAGE

- Bitbake creates a partitioned bootable image using the wic plugin
- Based on the OpenEmbedded Image Creator (oeic, renamed to wic)
- Layer is prepared to use the wic plugin by
 - Defining wic as an IMAGE_FSTYPE
 - Using a wic kickstart file (wks) to define the partitions and their content
 - Workshop custom layer includes the following line items in /conf/machine/my-custom-soc-board.conf

```
WKS_FILE ?= "sdimage-cyclone5-arria5.wks"  
IMAGE_FSTYPES += "wic"
```

SD Card Configuration

WKS FILE FOR CYCLONE 5

- Layer has a wic sub-folder which includes the wks definition files

```
part --source bootimg-partition --ondisk mmcblk --fstype=vfat --label boot --active --align 4 --size 16
part --source rawcopy --sourceparams="file=u-boot-with-spl.sfp" --ondisk mmcblk --system-id=a2 --align 4
part / --source rootfs --ondisk mmcblk --fstype=ext4 --label root --align 4|
```

- Three partitions are defined

- 1: vfat partition
 - uses bootimg-partition plugin
 - Creates a boot partition populating it with files defined by the IMAGE_BOOT_FILES variable
 - IMAGE_BOOT_FILES defined in my-custom-soc-board.conf

```
IMAGE_BOOT_FILES| ?= " \
    ${FPGA_CONFIG} \
    ${KERNEL_DEVICETREE} \
    ${KERNEL_IMAGETYPE} \
    extlinux.conf;extlinux/extlinux.conf \
"
```

SD Card Configuration

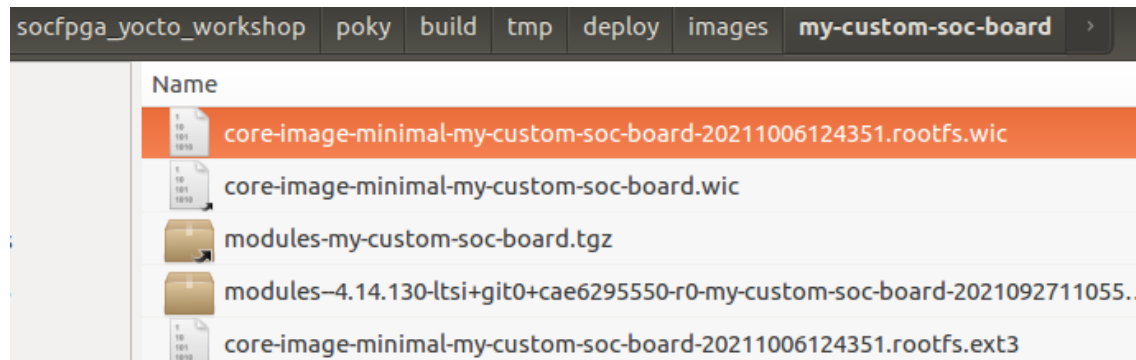
WKS FILE FOR CYCLONE 5

- Three partitions are defined
 - 2: binary partition
 - uses rawcopy plugin
 - Creates a binary partition populating it with the second stage bootloader file (u-boot-with-spl.sfp)
 - 3: rootfs partition
 - uses rootfs plugin
 - creates an ext4 journaling file system
 - populates it with the rootfs defined by the IMAGE_ROOTFS bitbake variable
 - creates a partition as big as needed by the rootfs. Use --size option to specify exact size

SD Card Configuration

DEPLOY DIRECTORY CONTENT

- All content created by bitbake for SD card image creation are place in a DEPLOY directory
- This includes the wic image file



- Use dd in the Linux host to copy the image to a SD card
 - `sudo dd if=tmp/deploy/images/my-custom-soc-board/console-image-minimal-my-custom-soc-board.wic of=/dev/sdX bs=1M seek=0`

Workshop

SECTION THREE

- Build a Linux image using a custom layer and execute it on
 - DE10-nano development kit or
 - DataStormDAQ or
 - SoCKit development kit
- Complete [sections 4 through 6](#)
 - 4. Create a Custom Board Layer
 - i. Requirements for Booting a custom SoC FPGA board
 - ii. Examine the custom layer directory structure
 - iii. Review custom MACHINE configuration
 - iv. Review FPGA configuration recipe
 - v. Create u-boot for the custom board
 - a. Typical HPS Boot Flow
 - b. HPS customization and u-boot

A dark-colored sports car, possibly a Porsche Carrera GT, is driving on a winding asphalt road that curves through a mountainous landscape. The road is flanked by steep, rocky slopes with patches of snow. In the background, a vast valley with a lake and distant mountains is visible under a blue sky with scattered white clouds. The car has several sponsor logos, including 'NOVON' and 'PRAHA', and a 'GRAND TOUR' banner on the windshield.

Linux OS for Intel SoC FPGAs

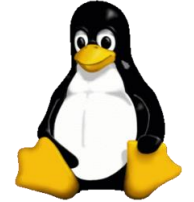
Intel® FPGA Linux* OS Portal: RocketBoards.org

- **THE place to start**
- <https://rocketboards.org/>
- For everything on SoC and Nios® II Processor Linux* OS
 - Support
 - Latest news
 - Latest Linux OS code
 - Reference designs
- **Contribute to Forum and Projects**



Linux OS Strategy

INTEL SUPPORT STRATEGY?



High quality Linux* OS support	Same kernel source tree for all SoC's and Nios® II processor
Modern release strategy	Support for the latest stable kernel, Long Term Support Initiative (LTSI) kernel, and LTSI with the PREEMPT_RT (real-time) patch
Community enablement	Upstream and maintain to kernel.org

Industry Leading Linux OS Support

INTEL SUPPORT STRATEGY?

- Intel® FPGA keeps up with the Linux* OS community
 - Kernel is upgraded every 3 months against every new kernel.org release
 - Intel FPGA is the maintainer for the SoCFPGA architecture folder (mach-socfpga)
- Allows developers to program hardware “close to the metal.”
- Multiple suppliers for software, development and support
- Networking ability
- High security
- Runs on any hardware
- Strong community support

Intel® FPGA Maintained Kernel Branches

Latest stable kernel	<ul style="list-style-type: none">• Updated periodically against kernel.org releases	<p>All branches are updated with:</p> <ul style="list-style-type: none">• Bug fixes• New features
LTSI	<ul style="list-style-type: none">• More stable code base for 24 months• Fixes, improvements and new features back-ported• No changes of API	
Real Time LTSI	<ul style="list-style-type: none">• LTSI kernel with the PREEMPT_RT patch installed	

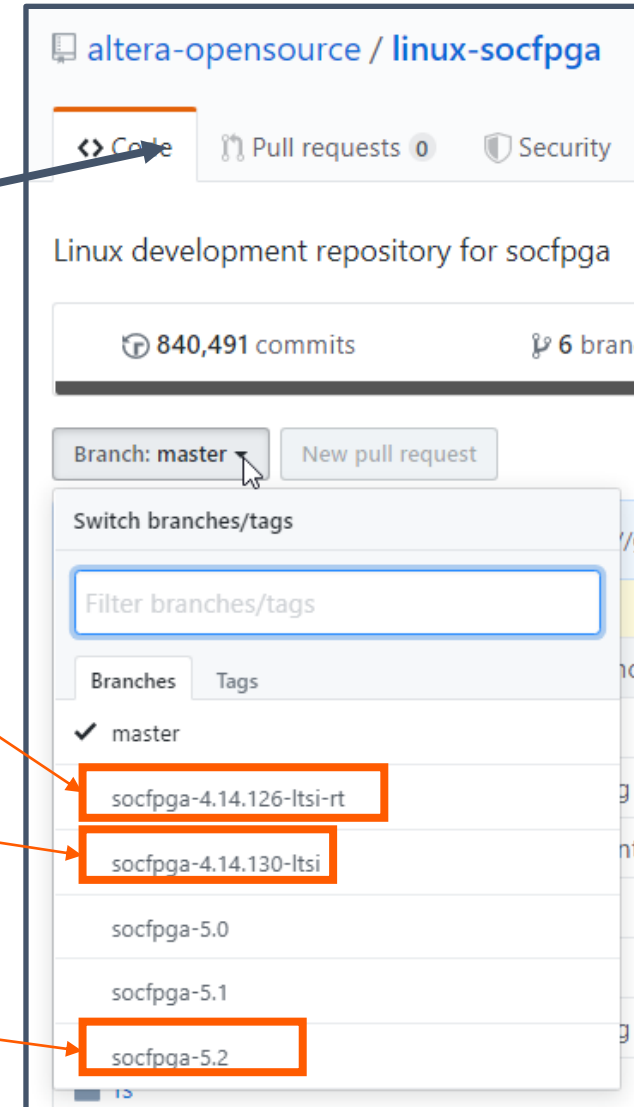
How to Find the Latest Kernels

github.com/altera-opensource/linux-socfpga

The LTSI kernel with the PREEMPT_RT patch will have -ltsi-rt at the end of the name

The LTSI kernel will have -ltsi at the end of the name

The latest stable kernel will be the highest number branch



LTSI Project

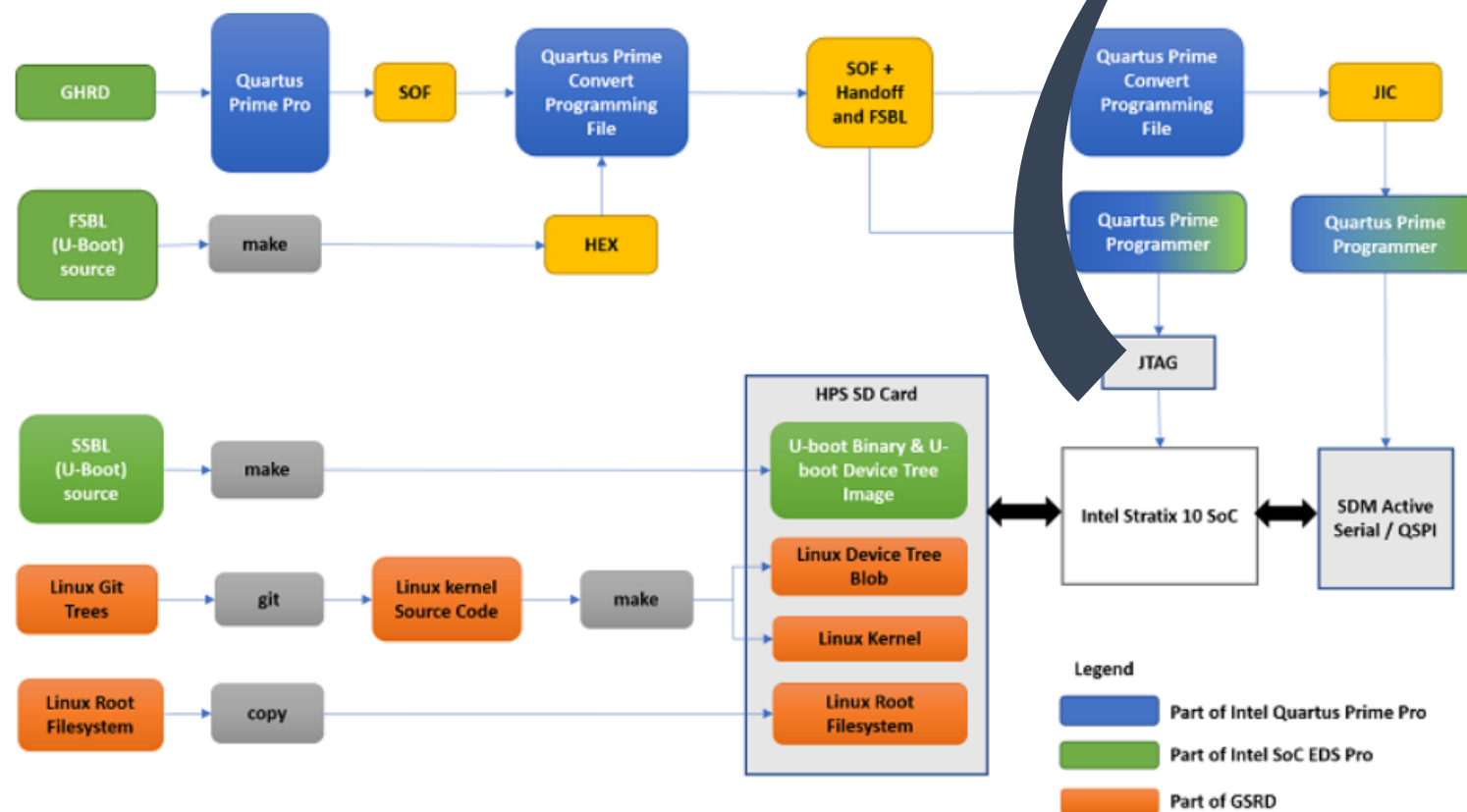


- Maintain very stable Linux* OS kernel version
 - Security and bug fixes based on the LTS kernel tree
 - Also, priority features are back ported: emerging device support, performance improvement, better power management
- Minimize fragmentation of kernel version
 - To accelerate further innovation of Linux OS by incorporating fixes and innovations from embedded system engineers
- <https://ltsi.linuxfoundation.org/>

Preempt Real Time Patch

- Linux* OS PREEMPT_RT is a 95% Real Time System
 - System compensates 5% of the time
- RT Preempt Changes ...
 - Threaded Interrupts
 - Pre-emptible mutual exclusion (“Sleeping” Spinlocks)
 - Priority Inheritance
 - High Resolution Timer
 - Real time scheduling policies – SCHED_RR and SCHED_FIFO
- “Real Time” applications are expected to make good choices in the application design
 - Smart processor and memory management
 - Smart priority assignment and management
- Simply using the RT Preempt patch does not solve all problems. Users must do some work too.
- User must be careful with affinities and priorities
- https://rt.wiki.kernel.org/index.php/Main_Page

Golden System Reference Design (GSRD)



name

- ds5
- hps_isw_handoff
- ip
- output_files
- soc_system
- software
- tgz
- construct_subsys_pcie.tcl
- create_ghrd_qsys.tcl
- create_ghrd_quartus.tcl
- create_ghrd_top.tcl
- cti_tapping.stp
- design_config.tcl
- fpga_pcie.sdc
- ghrd_reset.tcl
- ghrd_sc_script.tcl
- ghrd_top.v
- hps_common_board_info.xml
- Makefile
- sd_fat.tar.gz
- soc_system.dtb

Complete set of files resulting from the hardware and software build flows for the SoC FPGA development kits for each family

RocketBoards.org: GSRD User Manual

- Comprehensive information about the Golden System Reference Design for each device family
- THE best place to start for your own design
- Get started by simply burning the software image to an SD card and loading the FPGA design to the board.

Simply search for “GSRD User Manual” at RocketBoards.org

