Configuring and Building a Heterogenous System Using the Yocto Project

Mark Hatle (AECG)

Summary

Modern embedded systems are becoming more and more complex. This complexity is driving designs that require heterogenous systems. The talk will discuss why using a system device-tree may be a good approach to defining such as system, how a Yocto Project build project is configured for these types of systems, and an example of automating the configuration using a system device-tree. Once the system is configured, it can be used to construct and package the components for the defined heterogenous system, including Linux, bare metal applications, and firmware. Implementation details will be covered, as well as strategies to deal with binary only components. Mark will also discuss an example of how he has used these items to designed a heterogenous implementation with the Yocto Project for an FPGA based system that includes (aarch64) cortex-A, cortex-r5, and Microblaze architectures.

Introduction

I have been using and developing for Linux since 1993, and have been focusing on embedded Linux since 2000. I am an active contributor to both OpenEmbedded and the Yocto Project, and was involved in the creation of the Yocto Project. I've also been a maintainer of multiple projects, including various Yocto Project layers, and the cross-prelinker. At AMD AECG (formerly Xilinx), I have been working on Yocto Project integration, as well as bare metal toolchains and application support.

Topics

- Description of a heterogenous system
- System Device-Trees
- Yocto Project and multiconfig overview
- Going from System Device-Tree to YP Config to Built Image

Heterogeneous Systems

What is a Heterogenous System?

- A system of multiple CPUs that share resources.
- The components in the system may work independently, but the items are used together for full functionality.

- Examples:
  - VME, CompactPCI, and newer backplane based
  - Multiple compute units in a single chassis
  - Shared PCI bus
  - Each board (usually) has a specific purpose, and may use one or more architectures

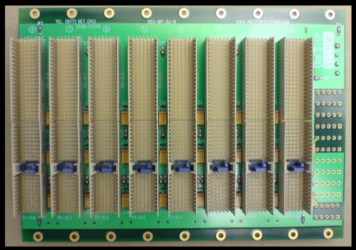
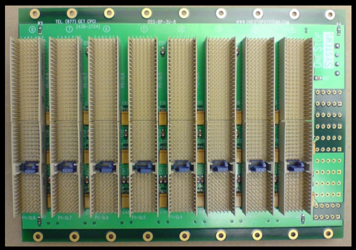


Image: https://en.wikipedia.org/wiki/CompactPCI#/media/File:CompactPCI-3U.jpg

What is a Heterogenous System?

- A system of multiple CPUs that share resources.
- The components in the system may work independently, but the items are used together for full functionality.

- Examples:
  - VME, CompactPCI, and newer backplane based
  - Multiple compute units in a single chassis
  - Shared PCI bus
  - Each board (usually) has a specific purpose, and may use one or more architectures



  - Virtual Machines
  - Multiple operating system instances on the same hardware, but shared hardware resources

  - Containers
  - Different dedicated applications in custom operating system containers

Image: https://en.wikipedia.org/wiki/CompactPCI#/media/File:CompactPCI-3U.jpg

[AMD Official Use Only]

What is a Heterogenous System?

- A system of multiple CPUs that share resources.
- The components in the system may work independently, but the items are used together for full functionality.

- CPU & DSP:
  - TI OMAP
  - Contains a general purpose ARM CPU and a DSP

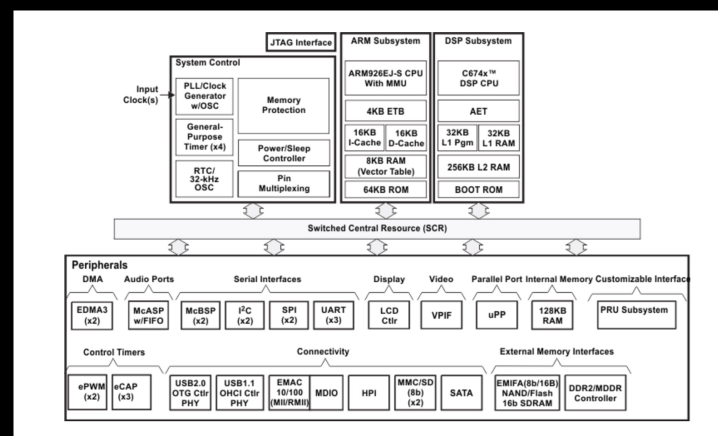


Image: https://www.ti.com/data-sheets/diagram.tsp?genericPartNumber=OMAP-L138&diagramId=SPRS586J

AMD

What is a Heterogenous System?

- A system of multiple CPUs that share resources.
- The components in the system may work independently, but the items are used together for full functionality.

- System On Chip:
  - AMD (Xilinx) Zynq UltraScale+ FPGA
  - General Purpose ARM Cortex-A53
  - ARM Cortex-R5
  - Microblaze Platform Management Unit
  - FPGA software defined CPUs

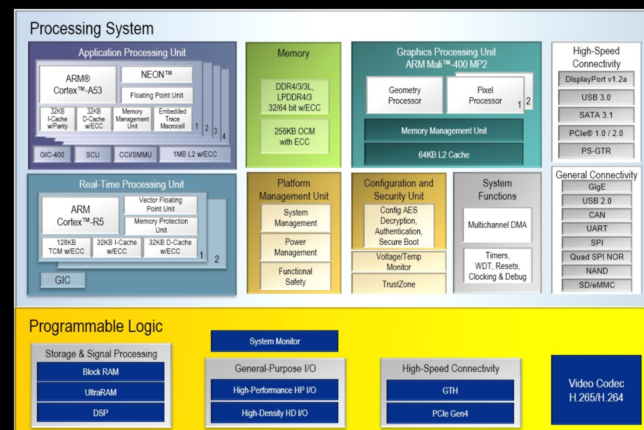


Image: https://www.xilinx.com/content/dam/xilinx/imgs/products/zynq/zynq-ev-block.PNG

Complications in building software for heterogeneous systems

- For a complex system that includes multiple operating systems (including baremetal), different CPUs and applications often a complex mixture of different build systems are required.
  - No single build system is capable of building everything
- Describing the system can be difficult (in software terms), as each component may require hardware to be defined in different ways.
- Custom integration is required to combine the various parts and pieces into an overall system "image".
  - Note: an "image" may not be a single disk or partition, but a set of them. An "image" comprises of a single build/version of the system software

System Device Tree

Device Tree

- What is a device tree?
  - (Wikipedia) a **devicetree** (also written **device tree**) is a data structure describing the hardware components of a particular computer so that the operating system's kernel can use and manage those components, including the CPU or CPUs, the memory, the buses and the integrated peripherals.
- Device trees express hardware information relevant to *specific operating environment*, such as u-boot, Linux kernel, and Zephyr.
- Because a device tree is often *operating environment* specific, HW nodes, topologies and memory addresses are described as seen from *one specific address space*.

Device Tree

- What is a device tree?
  - (Wikipedia) a **devicetree** (also written **device tree**) is a data structure describing the hardware components of a particular computer so that the operating system's kernel can use and manage those components, including the CPU or CPUs, the memory, the buses and the integrated peripherals.
- Device trees express hardware information relevant to *specific operating environment*, such as u-boot, Linux kernel, and Zephyr.
- Because a device tree is often *operating environment* specific, HW nodes, topologies and memory addresses are described as seen from *one specific address space*.
- Often heterogenous systems are described with multiple device trees for each operating environment.
  - Added complexity when the system design changes, i.e. how do you keep the trees in sync?

System Device Tree

- What is a system device tree?
  - See: https://static.linaro.org/connect/san19/presentations/san19-115.pdf
  - Describes the overall system hardware, no specific operating environment
  - Defines domains and operating environment information
  - A system device tree can be processed/transformed to generate a traditional OS/address space specific device tree
    - Processing done through pruning, and transformation based on domain and operating system specific processing

System Device Tree Transformations

```
[…]
cpus_a72 {
   compatible = "cpus,cluster";
   bus-master-id = <&lpd_xppu 0x0> , <&pmc_xppu 0x0> , <&lpd_xppu 0x1>, <&pmc_xppu 0x1>;
   #ranges-address-cells = <0x2>;
   #ranges-size-cells = <0x2>;
   address-map = <0x0 0xf0000000 &amba 0x0 0xf0000000 0x0 0x10000000>;
   […]
};
cpus_r5 {
   compatible = "cpus,cluster";
   bus-master-id = <&lpd_xppu 0x0> , <&pmc_xppu 0x0> , <&lpd_xppu 0x1>, <&pmc_xppu 0x1>;
   #ranges-address-cells = <0x1>;
   #ranges-size-cells = <0x1>;
   address-map = <0xf0000000 &amba 0xf0000000 0x10000000>;
   […]
};
cpus_microblaze_1: cpus_microblaze1 {
   compatible = "cpus,cluster";
   bus-master-id = <&lpd_xppu 0x247> , <&pmc_xppu 0x247>;
   #ranges-address-cells = <0x1>;
   #ranges-size-cells = <0x1>;
   address-map = <0xf0000000 &amba 0xf0000000 0x10000000>,
   […]
};
[…]
```

lop-domain-linux-a53

lop-domain-linux-a53-prune

lop-r5-imux

baremetal-microblaze

lopper

```
[…]
cpus {
   #address-cells = <0x01>;
   #size-cells = <0x00>;
   cpu@0 {
      compatible = "arm,cortex-a72";
      device_type = "cpu";
      reg = <0x00>;
};
[…]
```

```
[…]
cpus {
   #address-cells = <0x01>;
   #size-cells = <0x00>;
   cpu@0 {
      compatible = "arm,cortex-r5";
      device_type = "cpu";
      reg = <0x00>;
};
[…]
```

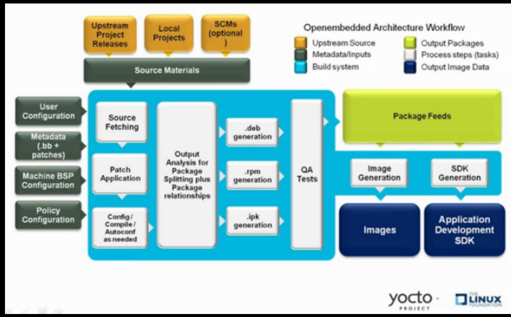
```
[…]
cpus {
   #address-cells = <0x01>;
   #size-cells = <0x00>;
   cpu@0 {
      compatible = "xlnx,microblaze";
      device_type = "cpu";
      reg = <0x00>;
};
[…]
```

Yocto Project

Yocto Project

The Yocto Project is not a Embedded Linux distribution, it helps you create a customized OS.

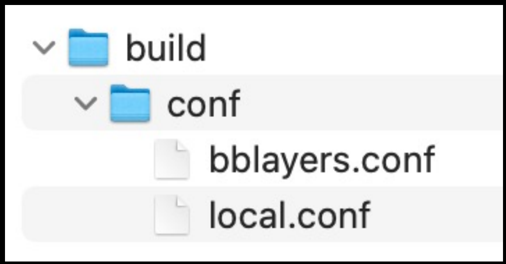
The Yocto Project is not a Single Open Source Project, but an ecosystem of projects.



- It can be configured to build Linux operating systems (default), FreeRTOS, Zephyr, bare metal, etc. This is captured by the ***distro*** configuration.
- A ***machine*** defines the hardware configuration for the build to target, uses ***DEFAULTTUNE*** variable to define CPU instruction set.
- ***Layers*** collect metadata and instructions for how to configure and build various software. This metadata is made up of .conf (configuration files), *recipes* and related data.
- ***Recipes*** define the metadata and tasks to be executed for a build, *tasks* define the steps to execute to accomplish work.
- Everything is controlled by *bitbake*, which like make, is used to run a specific target, i.e:
  - `bitbake core-image-minimal`

Yocto Project Configuration

- The build configuration is defined in bblayers.conf, and local.conf.
- bblayers.conf – defines which layers to use in your project, in other words what metadata and recipes are available to the build system
- local.conf – How your local project build is to configured. At a minimum this must define a ***machine*** to build, and a ***distro*** (OS) to target.
- Together these two files and the metadata in the layers either directly or indirectly configure the build system in specific ways, to construct a customized OS specific to the target machine.



Yocto Project Using Multiconfig

- Configurations are all variations of the default (local.conf) configuration

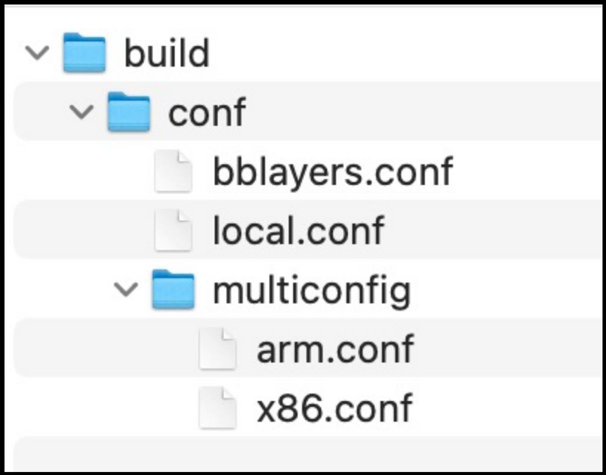
- Different ways of using multiconfigs can be done. For example you can:
  - Define machine and distro for each multiconfig
  - Define machine, but use the same distro as the default for each multiconfig
  - Use the same machine, but a different distro for each multiconfig
  - Any combination of the above!

- For a heterogenous system design, the only real variation is each multiconfig is no longer independent, but some sort of cross dependencies are desired

- Often the default (local.conf) will be the overall system configuration and depend on output from various multiconfigs (or external binaries from other build systems)
  - i.e. use firmware, kernels, bootable images, etc.

Yocto Project Multiconfig

- *local.conf* sets system wide defaults
- *local.conf* define which multiconfigs are available using:
  `BBMULTICONFIG = “x86 arm”`
- The configurations in the multiconfig directory specify specifics for each multiconfig (differences from defaults)
  - At a minimum requires a unique TMPDIR
  - Usually defines MACHINE and DISTRO as well



- Building:
  - (Default) `bitbake core-image-minimal`
  - (Multiconfig) `bitbake mc:arm:core-image-minimal mc:x86:bash`
- Recipe Dependencies:
  - Same config: *task[depends] = “recipe:task”*
  - Multiconfig: *task[mcdepends] = “mc:from_multiconfig:to_multiconfig:recipe:task”*
  - *Note: from_multiconfig* may be blank and will affect all configurations

See: https://docs.yoctoproject.org/dev-manual/common-tasks.html#building-images-for-multiple-targets-using-multiple-configurations

AMD

Going from System Device-Tree to YP Config to Built Image

Zynq UltraScale+ Example

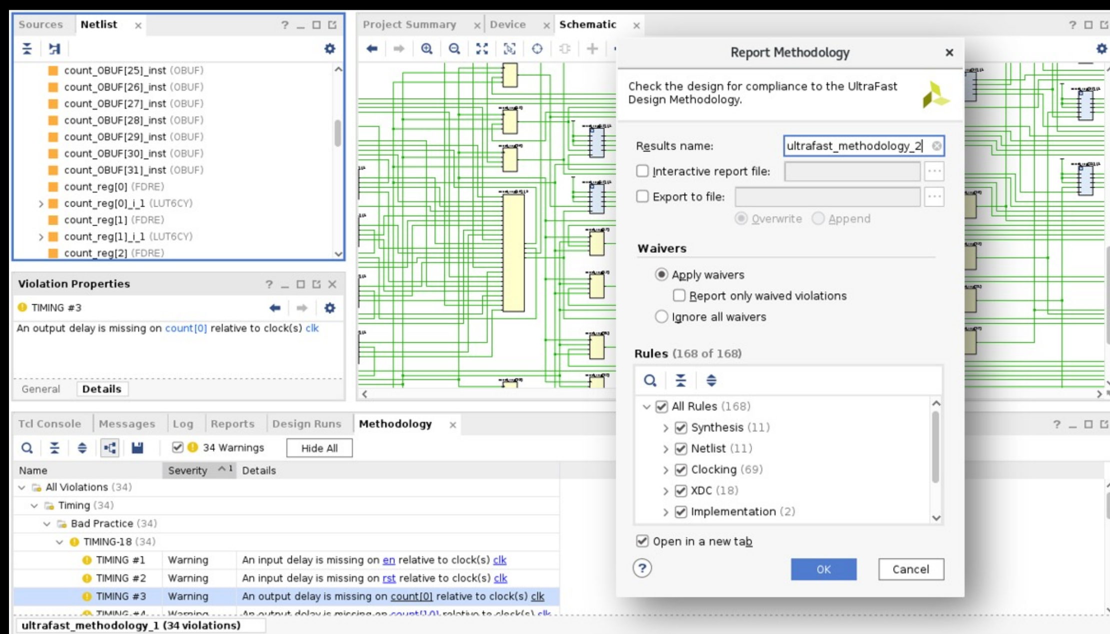
Zynq UltraScale+ Tools

A few tools that I want to define before we begin:

- Vivado: Software for synthesis and analysis of HDL designs. Output is a hardware description or XSA file.
- DTG++: Device Tree Generator, converts the data from the XSA file into a system device tree.
- Embeddedsw or ESW: Xilinx Embedded Software (baremetal) firmware components.
- Libxil: Baremetal hardware description library used by newlib (libc)
- FSBL: First Stage Bootloader based on ESW and libxil
- PMUFW: Platform Management Unit Firmware, based on ESW and libxil
- Bootbin/BIF: Tool that sets up the boot components, based on a BIF configuration

Hardware Flow

- The hardware developer creates their system design in Vivado, and outputs an XSA file.



Hardware / System Software

- The Hardware and System Software users then use DTG++, and output a System Device Tree (and associate files):
- system-top.dts is the primary interface, it includes the other files as appropriate

```
/dts-v1/;
#include "zynqmp.dtsi"
#include "zynqmp-clk-ccf.dtsi"
#include "zcu102-rev1.0.dtsi"
#include "pcw.dtsi"
/ {
        psu_ocm_ram_0_memory: memory@fffc0000 {
                compatible = "xlnx,psu-ocm-ram-0-1.0" , "mmio-sram";
                device_type = "memory";
                reg = <0x0 0xFFFC0000 0x0 0x40000>;
        };
…
};
```

```
.
├── include
│   └── dt-bindings
│       ├── clock
│       │   └── xlnx-zynqmp-clk.h
│       ├── dma
│       │   └── xlnx-zynqmp-dpdma.h
│       ├── gpio
│       │   └── gpio.h
│       ├── input
│       │   └── input.h
│       ├── interrupt-controller
│       │   └── irq.h
│       ├── phy
│       │   └── phy.h
│       ├── pinctrl
│       │   └── pinctrl-zynqmp.h
│       ├── power
│       │   └── xlnx-zynqmp-power.h
│       └── reset
│           └── xlnx-zynqmp-resets.h
├── pcw.dtsi
├── pl.dtsi
├── psu_init.c
├── psu_init.h
├── system-top.dts
├── zcu102-rev1.0.dtsi
├── zynqmp-clk-ccf.dtsi
└── zynqmp.dtsi
```


AMD

System Software Configuration

- Using the Yocto Project (and meta-xilinx/meta-xilinx-standalone-experimental layer) we build the recipe:
  - meta-xilinx-setup
  - This is a combination of the required components necessary to run the configuration tool, including python3 modules, lopper and the configuration tool itself.

- The configuration tool, *dt-processor.sh*, uses lopper to transform the system device tree into individual configuration device trees, as well as helps configure the Yocto Project itself.

```
./dt-processor.sh
   -c <config_dir>              Location of the build conf directory
   -s <system_dtb>              Full path to system DTB
   -d <domain_file>             Full path to domain file (.yml/.dts)
   [-o <overlay_dtb>]           Generate overlay dts
   [-e <external_fpga>]         Apply a partial overlay
   [-m <machine>]               zynqmp or versal
   [-p <psu_init_path>]         Path to psu_init files, defaults to system_dtb path
   [-i <pdu_path>]              Path to the pdi file
   [-l <config_file>]           write local.conf changes to this file
   [-P <petalinux_schema>] Path to petalinux schema file
```

dt-processor.sh

- dt-processor.sh -c conf -s /project/zu_sdt/system-top.dts -l conf/local.conf

local.conf:

```
# Adjust BASE_TMPDIR if you want to move the tmpdirs elsewhere
BASE_TMPDIR = "${TOPDIR}"
require conf/cortexa53-zynqmp-linux.conf
SYSTEM_DTFILE = "/project/zu_sdt/system-top.dts"

BBMULTICONFIG += " cortexa53-zynqmp-fsbl-baremetal
cortexa53-zynqmp-baremetal cortexa53-zynqmp-freertos
cortexr5-zynqmp-fsbl-baremetal cortexr5-zynqmp-baremetal
cortexr5-zynqmp-freertos microblaze-pmu"

FSBL_DEPENDS = ""
FSBL_MCDEPENDS = "mc::cortexa53-zynqmp-fsbl-baremetal:fsbl-firmware:do_deploy"
FSBL_DEPLOY_DIR = "${BASE_TMPDIR}/tmp-cortexa53-zynqmp-fsbl-baremetal/deploy/images/${MACHINE}"
R5FSBL_DEPENDS = ""
R5FSBL_MCDEPENDS = "mc::cortexr5-zynqmp-fsbl-baremetal:fsbl-firmware:do_deploy"
R5FSBL_DEPLOY_DIR = "${BASE_TMPDIR}/tmp-cortexr5-zynqmp-fsbl-baremetal/deploy/images/${MACHINE}"
PMU_DEPENDS = ""
PMU_MCDEPENDS = "mc::microblaze-pmu:pmu-firmware:do_deploy"
PMU_FIRMWARE_DEPLOY_DIR = "${BASE_TMPDIR}/tmp-microblaze-pmu/deploy/images/${MACHINE}"
```

```
├── bblayers.conf
├── cortexa53-zynqmp-linux.conf
├── dtb
│   ├── cortexa53-zynqmp-baremetal.dtb
│   ├── cortexa53-zynqmp-freertos.dtb
│   ├── cortexa53-zynqmp-fsbl-baremetal.dtb
│   ├── cortexa53-zynqmp-linux.dtb
│   ├── cortexa53-zynqmp-linux.dts
│   ├── cortexr5-zynqmp-baremetal.dtb
│   ├── cortexr5-zynqmp-freertos.dtb
│   ├── cortexr5-zynqmp-fsbl-baremetal.dtb
│   ├── lop-domain-linux-a53-prune.dts.dtb
│   └── microblaze-pmu.dtb
├── local.conf
├── microblaze.conf
├── multiconfig
│   ├── cortexa53-zynqmp-baremetal.conf
│   ├── cortexa53-zynqmp-freertos.conf
│   ├── cortexa53-zynqmp-fsbl-baremetal.conf
│   ├── cortexr5-zynqmp-baremetal.conf
│   ├── cortexr5-zynqmp-freertos.conf
│   ├── cortexr5-zynqmp-fsbl-baremetal.conf
│   ├── includes
│   │   ├── cortexa53-zynqmp-distro.conf
│   │   ├── cortexa53-zynqmp-fsbl-distro.conf
│   │   ├── cortexa53-zynqmp-fsbl-libxil.conf
│   │   ├── cortexa53-zynqmp-libxil.conf
│   │   ├── cortexr5-zynqmp-distro.conf
│   │   ├── cortexr5-zynqmp-fsbl-distro.conf
│   │   ├── cortexr5-zynqmp-fsbl-libxil.conf
│   │   ├── cortexr5-zynqmp-libxil.conf
│   │   ├── microblaze-pmu-distro.conf
│   │   └── microblaze-pmu-libxil.conf
│   └── microblaze-pmu.conf
└── templateconf.cfg
```

dt-processor.sh

- dt-processor.sh -c conf -s /project/zu_sdt/system-top.dts -l conf/local.conf

local.conf:

```
# Adjust BASE_TMPDIR if you want to move the tmpdirs elsewhere
BASE_TMPDIR = "${TOPDIR}"
require conf/cortexa53-zynqmp-linux.conf
SYSTEM_DTFILE = "/project/zu_sdt/system-top.dts"

BBMULTICONFIG += " cortexa53-zynqmp-fsbl-baremetal
cortexa53-zynqmp-baremetal cortexa53-zynqmp-freertos
cortexr5-zynqmp-fsbl-baremetal cortexr5-zynqmp-baremetal
cortexr5-zynqmp-freertos microblaze-pmu"

FSBL_DEPENDS = ""
FSBL_MCDEPENDS = "mc::cortexa53-zynqmp-fsbl-baremetal:fsbl-firmware:do_deploy"
FSBL_DEPLOY_DIR = "${BASE_TMPDIR}/tmp-cortexa53-zynqmp-fsbl-baremetal/deploy/images/${MACHINE}"
R5FSBL_DEPENDS = ""
R5FSBL_MCDEPENDS = "mc::cortexr5-zynqmp-fsbl-baremetal:fsbl-firmware:do_deploy"
R5FSBL_DEPLOY_DIR = "${BASE_TMPDIR}/tmp-cortexr5-zynqmp-fsbl-baremetal/deploy/images/${MACHINE}"
PMU_DEPENDS = ""
PMU_MCDEPENDS = "mc::microblaze-pmu:pmu-firmware:do_deploy"
PMU_FIRMWARE_DEPLOY_DIR = "${BASE_TMPDIR}/tmp-microblaze-pmu/deploy/images/${MACHINE}"
```

```
├── bblayers.conf
├── cortexa53-zynqmp-linux.conf
├── dtb
│   ├── cortexa53-zynqmp-baremetal.dtb
│   ├── cortexa53-zynqmp-freertos.dtb
│   ├── cortexa53-zynqmp-fsbl-baremetal.dtb
│   ├── cortexa53-zynqmp-linux.dtb
│   ├── cortexa53-zynqmp-linux.dts
│   ├── cortexr5-zynqmp-baremetal.dtb
│   ├── cortexr5-zynqmp-freertos.dtb
│   ├── cortexr5-zynqmp-fsbl-baremetal.dtb
│   ├── lop-domain-linux-a53-prune.dts.dtb
│   └── microblaze-pmu.dtb
├── local.conf
├── microblaze.conf
├── multiconfig
│   ├── cortexa53-zynqmp-baremetal.conf
│   ├── cortexa53-zynqmp-freertos.conf
│   ├── cortexa53-zynqmp-fsbl-baremetal.conf
│   ├── cortexr5-zynqmp-baremetal.conf
│   ├── cortexr5-zynqmp-freertos.conf
│   ├── cortexr5-zynqmp-fsbl-baremetal.conf
│   ├── includes
│   │   ├── cortexa53-zynqmp-distro.conf
│   │   ├── cortexa53-zynqmp-fsbl-distro.conf
│   │   ├── cortexa53-zynqmp-fsbl-libxil.conf
│   │   ├── cortexa53-zynqmp-libxil.conf
│   │   ├── cortexr5-zynqmp-distro.conf
│   │   ├── cortexr5-zynqmp-fsbl-distro.conf
│   │   ├── cortexr5-zynqmp-fsbl-libxil.conf
│   │   ├── cortexr5-zynqmp-libxil.conf
│   │   ├── microblaze-pmu-distro.conf
│   │   └── microblaze-pmu-libxil.conf
│   └── microblaze-pmu.conf
└── templateconf.cfg
```

dt-processor.sh (cont)

- Local.conf includes cortexa53-zynqmp-linux.conf – file defines part of the default configuration:

```
CONFIG_DTFILE = "${TOPDIR}/conf/dtb/cortexa53-zynqmp-linux.dtb"
MACHINE = "zynqmp-generic"
# We don't want the kernel to build us a device-tree
KERNEL_DEVICETREE:zynqmp-generic = ""
# We need u-boot to use the one we passed in
DEVICE_TREE_NAME:pn-u-boot-zynq-scr = "${@os.path.basename(d.getVar('CONFIG_DTFILE'))}"
# Update bootbin to use proper device tree
BIF_PARTITION_IMAGE[device-tree] = "${RECIPE_SYSROOT}/boot/devicetree/${@os.path.basename(d.getVar('CONFIG_DTFILE'))}"
# Remap boot files to ensure the right device tree is listed first
IMAGE_BOOT_FILES = "devicetree/${@os.path.basename(d.getVar('CONFIG_DTFILE'))} ${@get_default_image_boot_files(d)}"
```

- Default Configuration includes the following:
  - `MACHINE = “zynqmp-generic”`
  - `SYSTEM_DTFILE = “/project/zu_sdt/system-top.dts"`
  - `CONFIG_DTFILE = “${TOPDIR}/conf/dtb/cortexa53-zynqmp-linux.dtb"`
- Other lines are Linux specific or workaround for implementation details

dt-processor.sh (Linux config generation)

- “${TOPDIR}/conf/dtb/cortexa53-zynqmp-linux.dtb” generated using:

```
LOPPER_DTC_FLAGS="-b 0 -@" lopper -f --enhanced -i lops/lop-a53-imux.dts \
        -i lops/lop-domain-linux-a53.dts \
        -i lops/lop-domain-linux-a53-prune.dts \
        /project/zu_sdt/system-top.dts conf/dtb/cortexa53-zynqmp-linux.dtb
```

Uses three lopper transforms, lop-a53-imux, lop-domain-linux-a53, lop-domain-linux-a53-prune to construct the device tree.

AMD

dt-processor.sh (Microblaze config generation)

- Microblaze processors have many variants, capabilities of the CPU are defined in the device tree entries for the CPU.
- Lopper processes these and constructs a TUNE configuration.

```
lopper -f --enhanced -i lops/lop-microblaze-yocto.dts \
       /project/zu_sdt/system-top.dts > microblaze.conf
```

Output (microblaze.conf):

```
AVAILTUNES += "microblaze-cpu0"
TUNE_FEATURES:tune-microblaze-cpu0 = " microblaze v9.2 barrel-shift \
  pattern-compare reorder fpu-soft"
PACKAGE_EXTRA_ARCHS:tune-microblaze-cpu0 = "${TUNE_PKGARCH}"
TUNE_FEATURES:tune-pmu-microblaze = "${TUNE_FEATURES:tune-microblaze-cpu0}"
```


AMD

dt-processor.sh (Baremetal config generation)

- "${TOPDIR}/conf/dtb/cortexa53-zynqmp-baremetal.dtb" generated using:

```
LOPPER_DTC_FLAGS="-b 0 -@" lopper -f --enhanced -i lops/lop-a53-imux.dts \
       /project/zu_sdt/system-top.dts conf/dtb/cortexa53-zynqmp-baremetal.dtb
```

- Generates Yocto Project baremetal distribution configuration using lopper:

```
lopper -f /project/zu_sdt/system-top.dts -- baremetaldrvlist_xlnx \
       cortexa53-zynqmp lops/share/embeddedsw
```

…-distro.conf:

```
DISTRO_FEATURES = "avbuf axipmon canps clockps common coresightps-dcc csudma ddrcpsu dppsu emacps
gpiops iicps ipipsu pciepsu qspipsu resetps rtcpsu scugic sdps sysmonpsu ttcps uartps usbpsu video-
common wdtps zdma"
```

…-libxil.conf:

```
PACKAGECONFIG[avbuf] = "${RECIPE_SYSROOT}/usr/lib/libavbuf.a,,avbuf,,"
…
```

Microblaze generated multiconfig file

```
CONFIG_DTFILE = "${TOPDIR}/conf/dtb/microblaze-pmu.dtb"
ESW_MACHINE = "microblaze-pmu"

require conf/microblaze.conf
DEFAULTTUNE = "microblaze"
TUNE_FEATURES:tune-microblaze:forcevariable = "${TUNE_FEATURES:tune-pmu-microblaze}"

TARGET_CFLAGS += "-DPSU_PMU=1U"

TMPDIR = "${BASE_TMPDIR}/tmp-microblaze-pmu"

DISTRO = "xilinx-standalone"

LIBXIL_CONFIG = "conf/multiconfig/includes/microblaze-pmu-libxil.conf"
require conf/multiconfig/includes/microblaze-pmu-distro.conf
```

- Instead of overriding *MACHINE*, uses existing machine and simply changes the *DEFAULTTUNE*
- Same approach is used to cortex-r5 to switch the MACHINE to that CPU type

- DISTRO is overridden to xilinx-standalone, our special bare metal configuration

AMD

dt-processor.sh

- dt-processor.sh -c conf -s /project/zu_sdt/system-top.dts -l conf/local.conf

local.conf:

```
# Adjust BASE_TMPDIR if you want to move the tmpdirs elsewhere
BASE_TMPDIR = "${TOPDIR}"
require conf/cortexa53-zynqmp-linux.conf
SYSTEM_DTFILE = "/project/zu_sdt/system-top.dts"

BBMULTICONFIG += " cortexa53-zynqmp-fsbl-baremetal
cortexa53-zynqmp-baremetal cortexa53-zynqmp-freertos
cortexr5-zynqmp-fsbl-baremetal cortexr5-zynqmp-baremetal
cortexr5-zynqmp-freertos microblaze-pmu"

FSBL_DEPENDS = ""
FSBL_MCDEPENDS = "mc::cortexa53-zynqmp-fsbl-baremetal:fsbl-firmware:do_deploy"
FSBL_DEPLOY_DIR = "${BASE_TMPDIR}/tmp-cortexa53-zynqmp-fsbl-baremetal/deploy/images/${MACHINE}"
R5FSBL_DEPENDS = ""
R5FSBL_MCDEPENDS = "mc::cortexr5-zynqmp-fsbl-baremetal:fsbl-firmware:do_deploy"
R5FSBL_DEPLOY_DIR = "${BASE_TMPDIR}/tmp-cortexr5-zynqmp-fsbl-baremetal/deploy/images/${MACHINE}"
PMU_DEPENDS = ""
PMU_MCDEPENDS = "mc::microblaze-pmu:pmu-firmware:do_deploy"
PMU_FIRMWARE_DEPLOY_DIR = "${BASE_TMPDIR}/tmp-microblaze-pmu/deploy/images/${MACHINE}"
```

```
├── bblayers.conf
├── cortexa53-zynqmp-linux.conf
├── dtb
│   ├── cortexa53-zynqmp-baremetal.dtb
│   ├── cortexa53-zynqmp-freertos.dtb
│   ├── cortexa53-zynqmp-fsbl-baremetal.dtb
│   ├── cortexa53-zynqmp-linux.dtb
│   ├── cortexa53-zynqmp-linux.dts
│   ├── cortexr5-zynqmp-baremetal.dtb
│   ├── cortexr5-zynqmp-freertos.dtb
│   ├── cortexr5-zynqmp-fsbl-baremetal.dtb
│   ├── lop-domain-linux-a53-prune.dts.dtb
│   └── microblaze-pmu.dtb
├── local.conf
├── microblaze.conf
├── multiconfig
│   ├── cortexa53-zynqmp-baremetal.conf
│   ├── cortexa53-zynqmp-freertos.conf
│   ├── cortexa53-zynqmp-fsbl-baremetal.conf
│   ├── cortexr5-zynqmp-baremetal.conf
│   ├── cortexr5-zynqmp-freertos.conf
│   ├── cortexr5-zynqmp-fsbl-baremetal.conf
│   ├── includes
│   │   ├── cortexa53-zynqmp-distro.conf
│   │   ├── cortexa53-zynqmp-fsbl-distro.conf
│   │   ├── cortexa53-zynqmp-fsbl-libxil.conf
│   │   ├── cortexa53-zynqmp-libxil.conf
│   │   ├── cortexr5-zynqmp-distro.conf
│   │   ├── cortexr5-zynqmp-fsbl-distro.conf
│   │   ├── cortexr5-zynqmp-fsbl-libxil.conf
│   │   ├── cortexr5-zynqmp-libxil.conf
│   │   ├── microblaze-pmu-distro.conf
│   │   └── microblaze-pmu-libxil.conf
│   └── microblaze-pmu.conf
└── templateconf.cfg
```

Cross multiconfig dependencies

```
# Adjust BASE_TMPDIR if you want to move the tmpdirs elsewhere
BASE_TMPDIR = "${TOPDIR}"
require conf/cortexa53-zynqmp-linux.conf
SYSTEM_DTFILE = "/project/zu_sdt/system-top.dts"

BBMULTICONFIG += " cortexa53-zynqmp-fsbl-baremetal
cortexa53-zynqmp-baremetal cortexa53-zynqmp-freertos
cortexr5-zynqmp-fsbl-baremetal cortexr5-zynqmp-baremetal
cortexr5-zynqmp-freertos microblaze-pmu"

FSBL_DEPENDS = ""
FSBL_MCDEPENDS = "mc::cortexa53-zynqmp-fsbl-baremetal:fsbl-firmware:do_deploy"
FSBL_DEPLOY_DIR = "${BASE_TMPDIR}/tmp-cortexa53-zynqmp-fsbl-baremetal/deploy/images/${MACHINE}"
R5FSBL_DEPENDS = ""
R5FSBL_MCDEPENDS = "mc::cortexr5-zynqmp-fsbl-baremetal:fsbl-firmware:do_deploy"
R5FSBL_DEPLOY_DIR = "${BASE_TMPDIR}/tmp-cortexr5-zynqmp-fsbl-baremetal/deploy/images/${MACHINE}"
PMU_DEPENDS = ""
PMU_MCDEPENDS = "mc::microblaze-pmu:pmu-firmware:do_deploy"
PMU_FIRMWARE_DEPLOY_DIR = "${BASE_TMPDIR}/tmp-microblaze-pmu/deploy/images/${MACHINE}"
```

- Global definitions to define where to build firmware binaries, for both the builder and also packager

default:fsbl → mc:cortexa53-zynqmp-fsbl-baremetal:fsb-firmware

default:pmufw → mc:microblaze-pmu:pmu-firmware

AMD

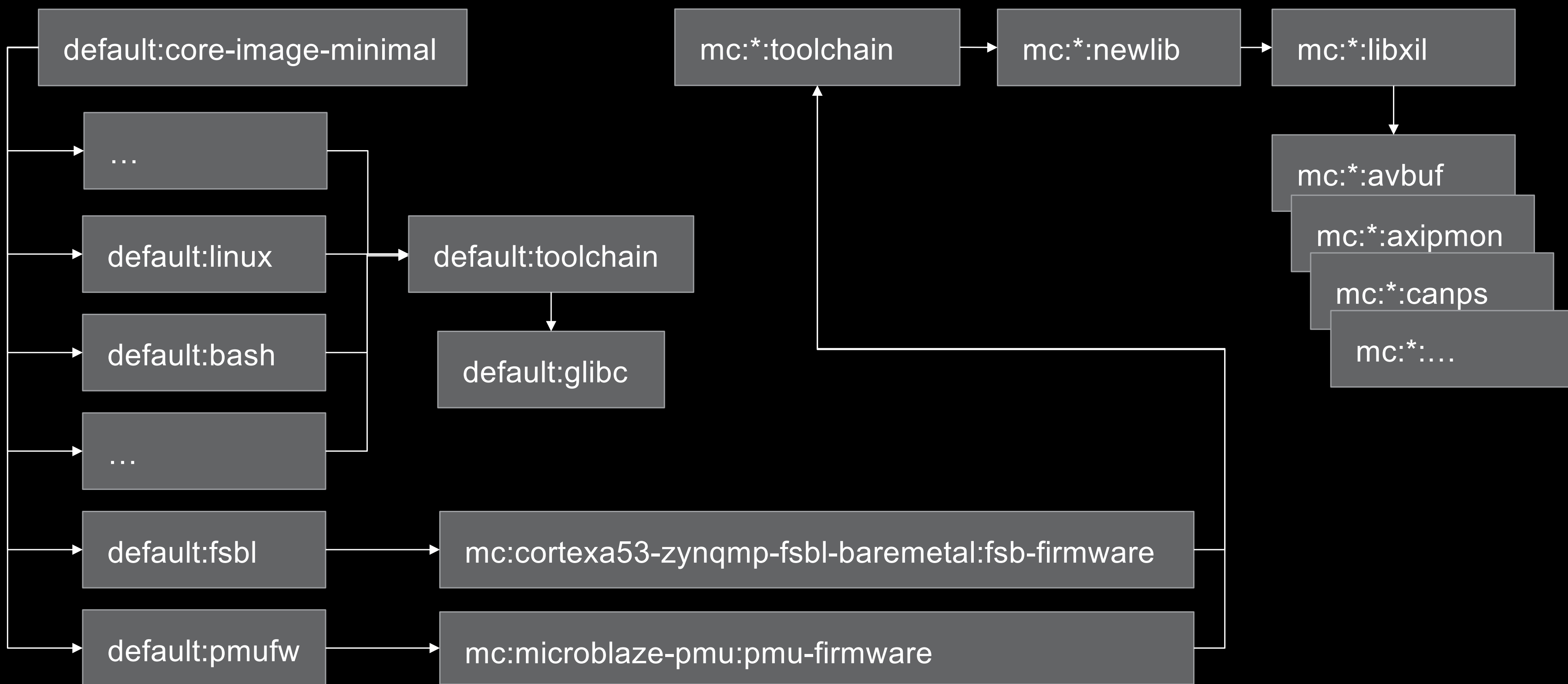
Recipe Implementation (Consumer)

```
VAR_DEPENDS ??= “”
VAR_MCDEPENDS ??= “”
VAR_DEPLOY_DIR ??= “${DEPLOY_DIR_IMAGE}”
VAR_DEPLOY_DIR[vardepsexclude] += “TOPDIR”
VAR_IMAGE_NAME ??= “provider-${MACHINE}”

do_fetch[depends] += “${VAR_DEPENDS}”
do_fetch[mcdepends] += “${VAR_MCDEPENDS}”

do_install() {
        install -Dm 0644 ${VAR_DEPLOY_DIR}/${VAR_IMAGE_NAME} ${D}/dest_dir/dest_filename
}
SHOULD_DEPLOY = "${@'false' if (d.getVar(VAR_DEPLOY_DIR')).startswith(d.getVar('DEPLOY_DIR_IMAGE')) else 'true'}"
do_deploy() {
  # If the item is already in OUR deploy_image_dir, nothing to deploy!
  if ${SHOULD_DEPLOY}; then
    install -Dm 0644 ${VAR_DEPLOY_DIR}/${VAR_IMAGE_NAME} ${DEPLOYDIR}/dest_filename
  fi
}
```

Recipe Implementation (Provider)

```
VAR_IMAGE_NAME ??= “provider-${MACHINE}”

do_configure() {
        …
}
do_compile() {
        …
}
# no do_install, as the output is not “packaged”, only deployed
do_deploy() {
        install -Dm 0644 … ${DEPLOYDIR}/${VAR_IMAGE_NAME}
}
```

System Software Build Map



Lessons Learned/Next Steps?

Lessons Learned:

- Building a multiconfig system is very reasonable, but requires additional machine resources
  - Had cases were developers machines ran out of memory due to multiple toolchain builds at the same time
- Current output is a compiled DT-B in most cases, have had requests to store DT-S instead so users could modify source, if needed
- Using binaries from other build systems is mandatory to build into the system
  - If someone just gives you a prebuilt firmware, may be necessary to just use it instead of always build it
- Always generating files configuration for each users build can lead to problems

Next Steps:

- Instead of configuring in the project, generate a machine.conf, multiconfigs and device trees so can be placed in a layer and shared more easily (avoids user needing to generate them)
  - User's local.conf only need `MACHINE = …`, and `BBMULTICONFIG = …`
- Generate additional configuration (QEMUBOOT args, etc) in new machine.conf
- Lots of workflow cleanup to make it easier, more intuitive to enable using external binaries