THE LINUX FOUNDATION OPEN SOURCE SUMMIT EUROPE | Embedded Linux Conference Europe

XILINX

One Build To Rule Them All: Building FreeRTOS & Linux Using The Yocto Project

M.S. Alejandro Enedino Hernandez Samaniego
Xilinx Inc.

Github @ aehs29

THE LINUX FOUNDATION

Outline

Introduction

- Heterogeneous Systems
- RTOS + Linux
- Workflows

Outline

OpenEmbedded Development

- FreeRTOS on Yocto
- Newlib + Libgloss
  - tclibc-newlib
- meta-freertos
  - class, recipes
  - BSP vs App
  - QEMU
  - Automated Tests

Outline

One Build To Rule Them All

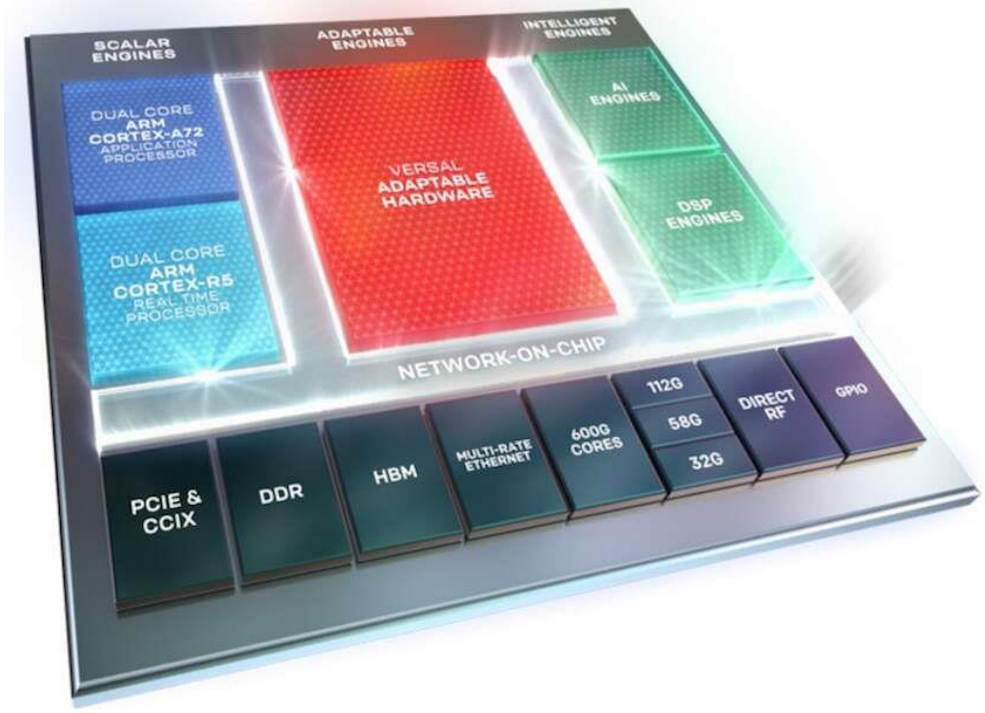
- Multiconfig Builds
- Multiconfig Dependencies
- One Build To Rule Them All

Heterogeneous Devices

Xilinx Versal Architecture

Processing System:

- Application Processing Unit
- Real-Time Processing Unit



It Depends

Embedded Applications:

- Linux + RTOS
- RTOS
- Linux + Baremetal
- Baremetal

They all use different workflows!

FreeRTOS + The Yocto Project

FreeRTOS + The Yocto Project

ARM on QEMU already on OE-Core BSP versatile92s (qemuarmv5)

ARM Embedded Toolchain[1]:

- GNU C Compiler
- Binutils
- GDB
- Newlib
  - Newlib is a C library intended for use on embedded systems. It is a conglomeration of several library parts, all under free software licenses that make them easily usable on embedded products. [2]

[1] https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm

[2] https://sourceware.org/newlib/

FreeRTOS + The Yocto Project

Previous work:

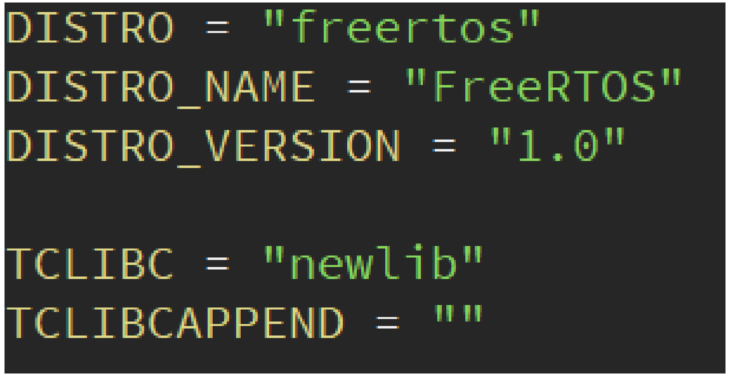
- Newlib + Libgloss Recipes on OE-Core
- tclibc-newlib (TCLIBC)

Create Layer:

- Meta-FreeRTOS
  - Create DISTRO
  - Application is the OS
  - BSP vs Application
  - Use a class to simply workflows and create abstractions
  - Run automated tests on RTOS applications using the OpenEmbedded testing infrastructure.

FreeRTOS + The Yocto Project

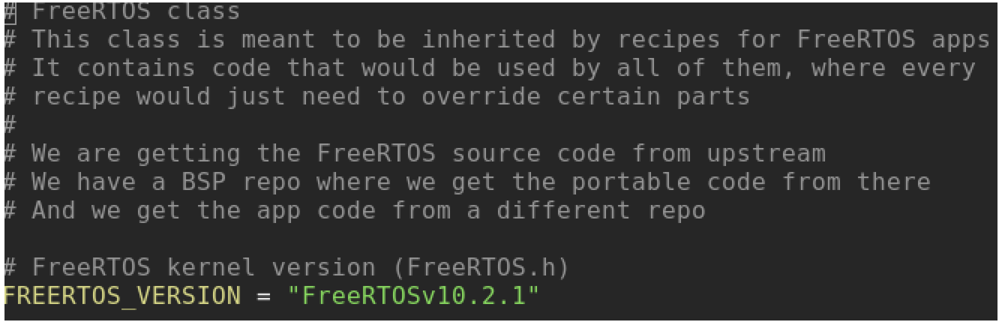
DISTRO



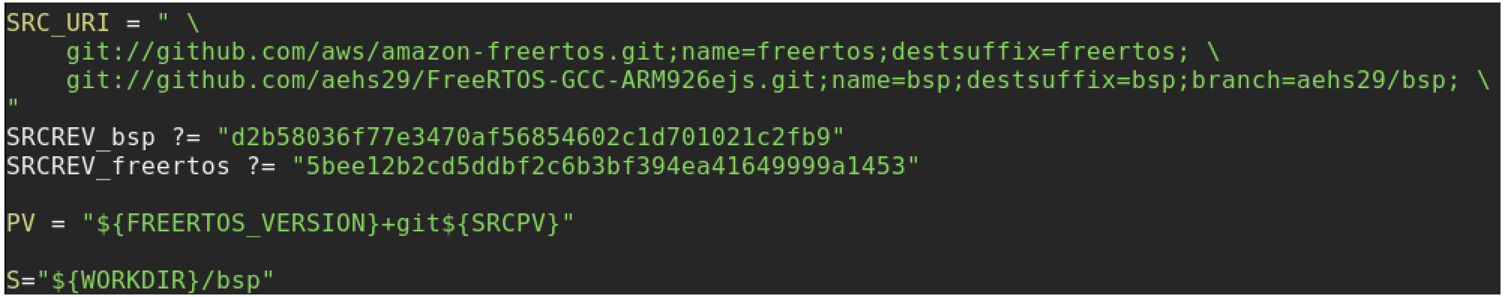
```
DISTRO = "freertos"
DISTRO_NAME = "FreeRTOS"
DISTRO_VERSION = "1.0"

TCLIBC = "newlib"
TCLIBCAPPEND = ""
```

FreeRTOS + The Yocto Project

CLASS: freertos-app

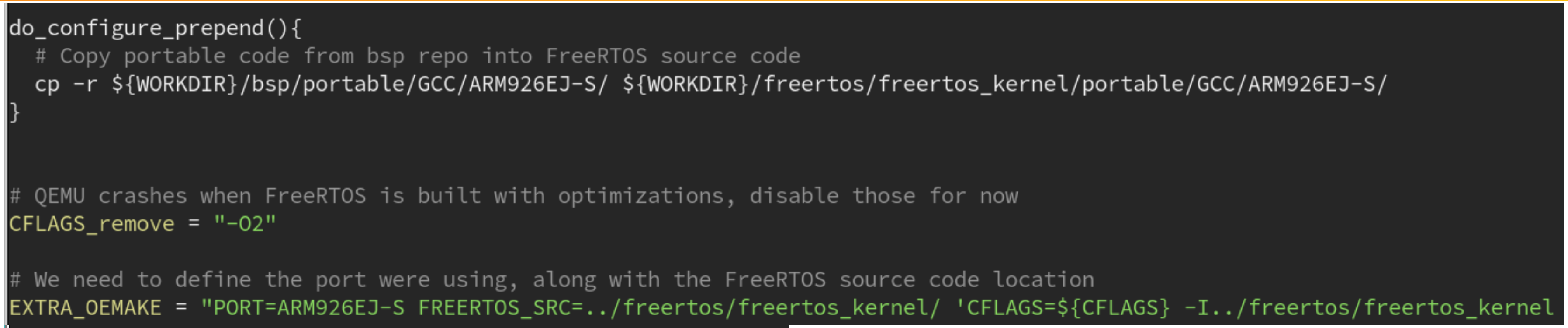


```
# FreeRTOS class
# This class is meant to be inherited by recipes for FreeRTOS apps
# It contains code that would be used by all of them, where every
# recipe would just need to override certain parts
#
# We are getting the FreeRTOS source code from upstream
# We have a BSP repo where we get the portable code from there
# And we get the app code from a different repo

# FreeRTOS kernel version (FreeRTOS.h)
FREERTOS_VERSION = "FreeRTOSv10.2.1"
```

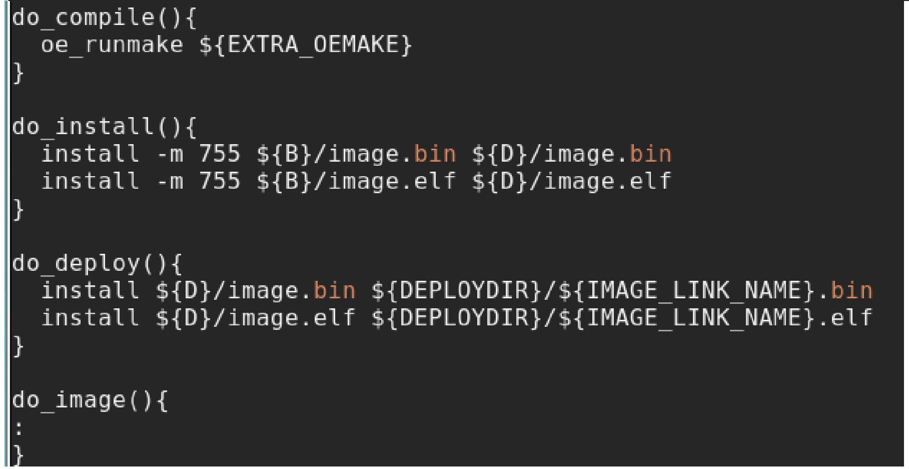


```
SRC_URI = " \
    git://github.com/aws/amazon-freertos.git;name=freertos;destsuffix=freertos; \
    git://github.com/aehs29/FreeRTOS-GCC-ARM926ejs.git;name=bsp;destsuffix=bsp;branch=aehs29/bsp; \
"
SRCREV_bsp ?= "d2b58036f77e3470af56854602c1d701021c2fb9"
SRCREV_freertos ?= "5bee12b2cd5ddbf2c6b3bf394ea41649999a1453"

PV = "${FREERTOS_VERSION}+git${SRCPV}"

S="${WORKDIR}/bsp"
```

FreeRTOS + The Yocto Project



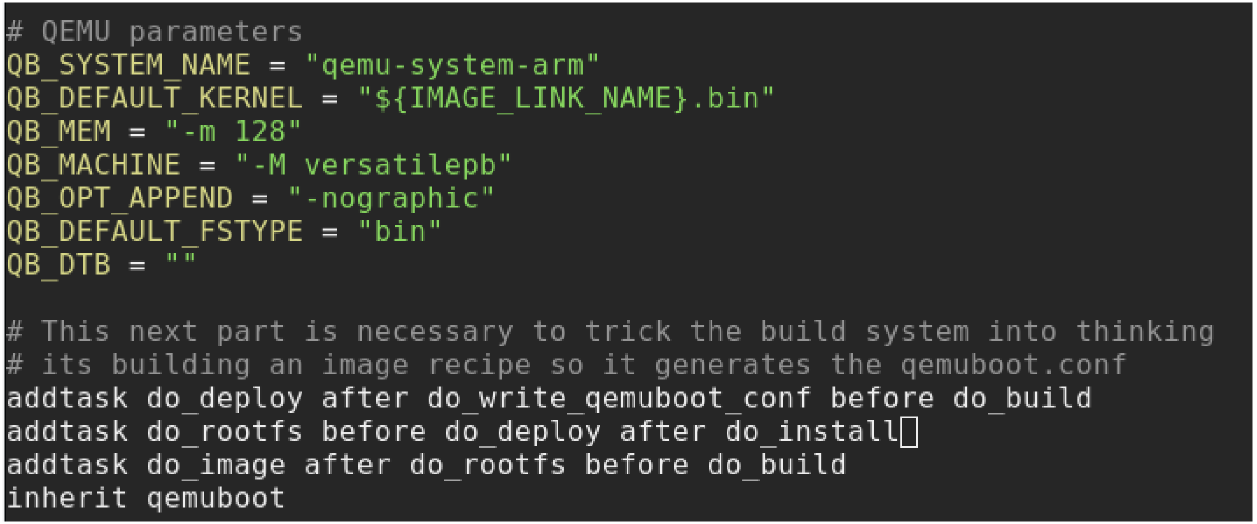
```
do_configure_prepend(){
  # Copy portable code from bsp repo into FreeRTOS source code
  cp -r ${WORKDIR}/bsp/portable/GCC/ARM926EJ-S/ ${WORKDIR}/freertos/freertos_kernel/portable/GCC/ARM926EJ-S/
}


# QEMU crashes when FreeRTOS is built with optimizations, disable those for now
CFLAGS_remove = "-O2"

# We need to define the port were using, along with the FreeRTOS source code location
EXTRA_OEMAKE = "PORT=ARM926EJ-S FREERTOS_SRC=../freertos/freertos_kernel/ 'CFLAGS=${CFLAGS} -I../freertos/freertos_kernel
```



```
do_compile(){
  oe_runmake ${EXTRA_OEMAKE}
}

do_install(){
  install -m 755 ${B}/image.bin ${D}/image.bin
  install -m 755 ${B}/image.elf ${D}/image.elf
}

do_deploy(){
  install ${D}/image.bin ${DEPLOYDIR}/${IMAGE_LINK_NAME}.bin
  install ${D}/image.elf ${DEPLOYDIR}/${IMAGE_LINK_NAME}.elf
}

do_image(){
:
}
```

FreeRTOS + The Yocto Project

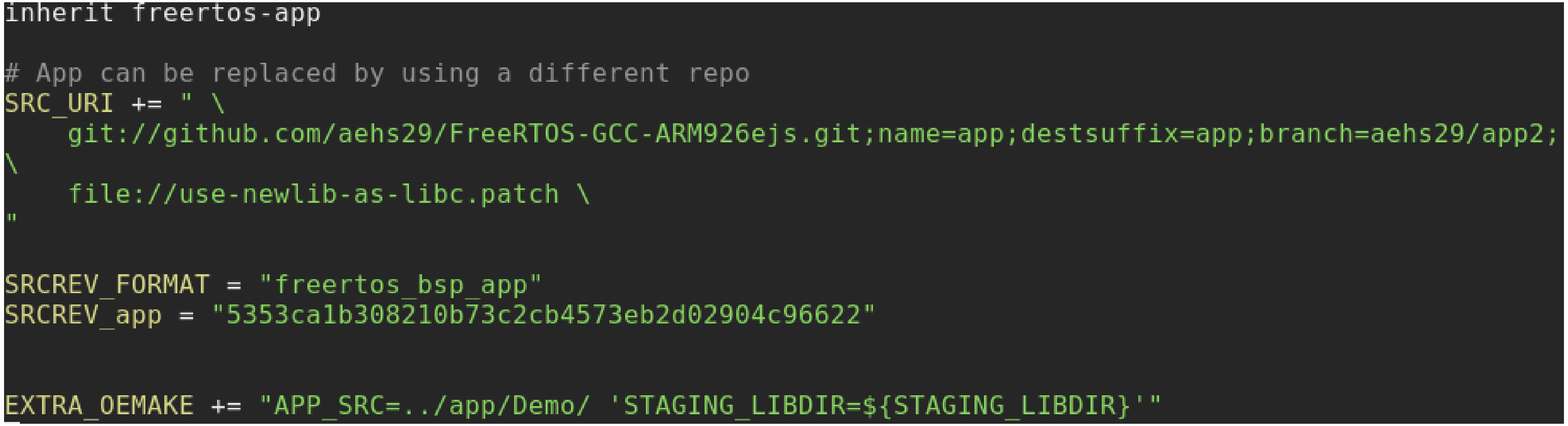
CLASS



```
# QEMU parameters
QB_SYSTEM_NAME = "qemu-system-arm"
QB_DEFAULT_KERNEL = "${IMAGE_LINK_NAME}.bin"
QB_MEM = "-m 128"
QB_MACHINE = "-M versatilepb"
QB_OPT_APPEND = "-nographic"
QB_DEFAULT_FSTYPE = "bin"
QB_DTB = ""

# This next part is necessary to trick the build system into thinking
# its building an image recipe so it generates the qemuboot.conf
addtask do_deploy after do_write_qemuboot_conf before do_build
addtask do_rootfs before do_deploy after do_install
addtask do_image after do_rootfs before do_build
inherit qemuboot
```

FreeRTOS + The Yocto Project

Demo Recipe: freertos-demo



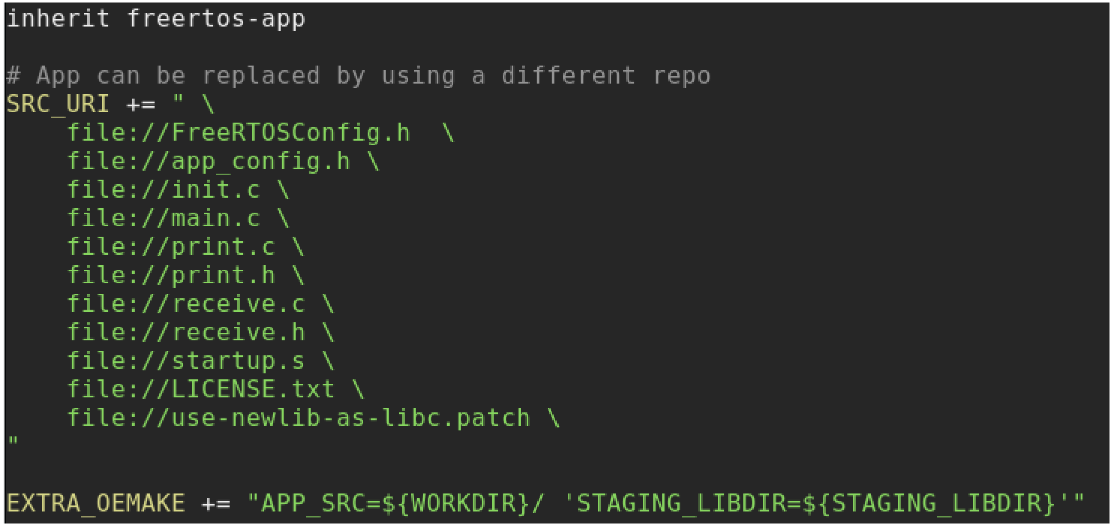
```
inherit freertos-app

# App can be replaced by using a different repo
SRC_URI += " \
    git://github.com/aehs29/FreeRTOS-GCC-ARM926ejs.git;name=app;destsuffix=app;branch=aehs29/app2;
\
    file://use-newlib-as-libc.patch \
"

SRCREV_FORMAT = "freertos_bsp_app"
SRCREV_app = "5353ca1b308210b73c2cb4573eb2d02904c96622"


EXTRA_OEMAKE += "APP_SRC=../app/Demo/ 'STAGING_LIBDIR=${STAGING_LIBDIR}'"
```

ARM926 Port: [1] https://github.com/jkovacic/FreeRTOS-GCC-ARM926ejs

Extra functionality: Task can be woken up by notification using xTaskNotify API

FreeRTOS + The Yocto Project

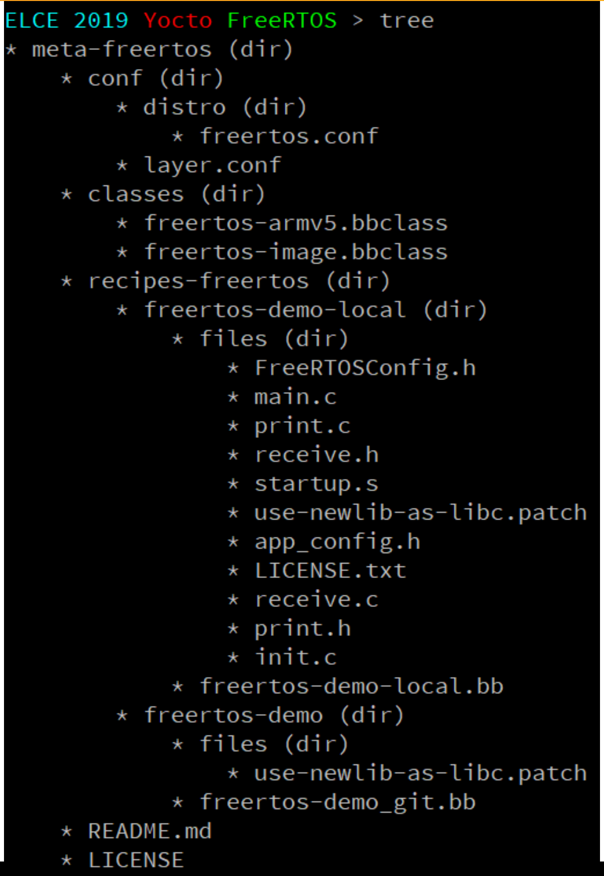
Demo Recipe: freertos-demo-local



```
inherit freertos-app

# App can be replaced by using a different repo
SRC_URI += " \
    file://FreeRTOSConfig.h  \
    file://app_config.h \
    file://init.c \
    file://main.c \
    file://print.c \
    file://print.h \
    file://receive.c \
    file://receive.h \
    file://startup.s \
    file://LICENSE.txt \
    file://use-newlib-as-libc.patch \
"

EXTRA_OEMAKE += "APP_SRC=${WORKDIR}/ 'STAGING_LIBDIR=${STAGING_LIBDIR}'"
```

FreeRTOS + The Yocto Project

Meta layer file hierarchy:



```
ELCE 2019 Yocto FreeRTOS > tree
* meta-freertos (dir)
    * conf (dir)
        * distro (dir)
            * freertos.conf
        * layer.conf
    * classes (dir)
        * freertos-armv5.bbclass
        * freertos-image.bbclass
    * recipes-freertos (dir)
        * freertos-demo-local (dir)
            * files (dir)
                * FreeRTOSConfig.h
                * main.c
                * print.c
                * receive.h
                * startup.s
                * use-newlib-as-libc.patch
                * app_config.h
                * LICENSE.txt
                * receive.c
                * print.h
                * init.c
            * freertos-demo-local.bb
        * freertos-demo (dir)
            * files (dir)
                * use-newlib-as-libc.patch
            * freertos-demo_git.bb
    * README.md
    * LICENSE
```

FreeRTOS + The Yocto Project

Demo & Test

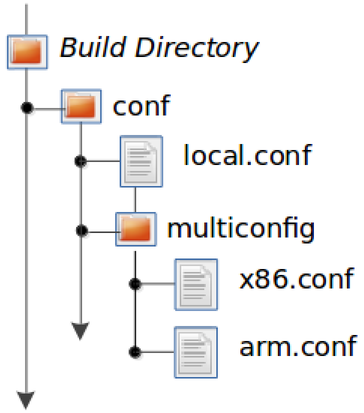
To reproduce the video demo, perform the instructions from the README on the layer repository:

https://github.com/aehs29/meta-freertos

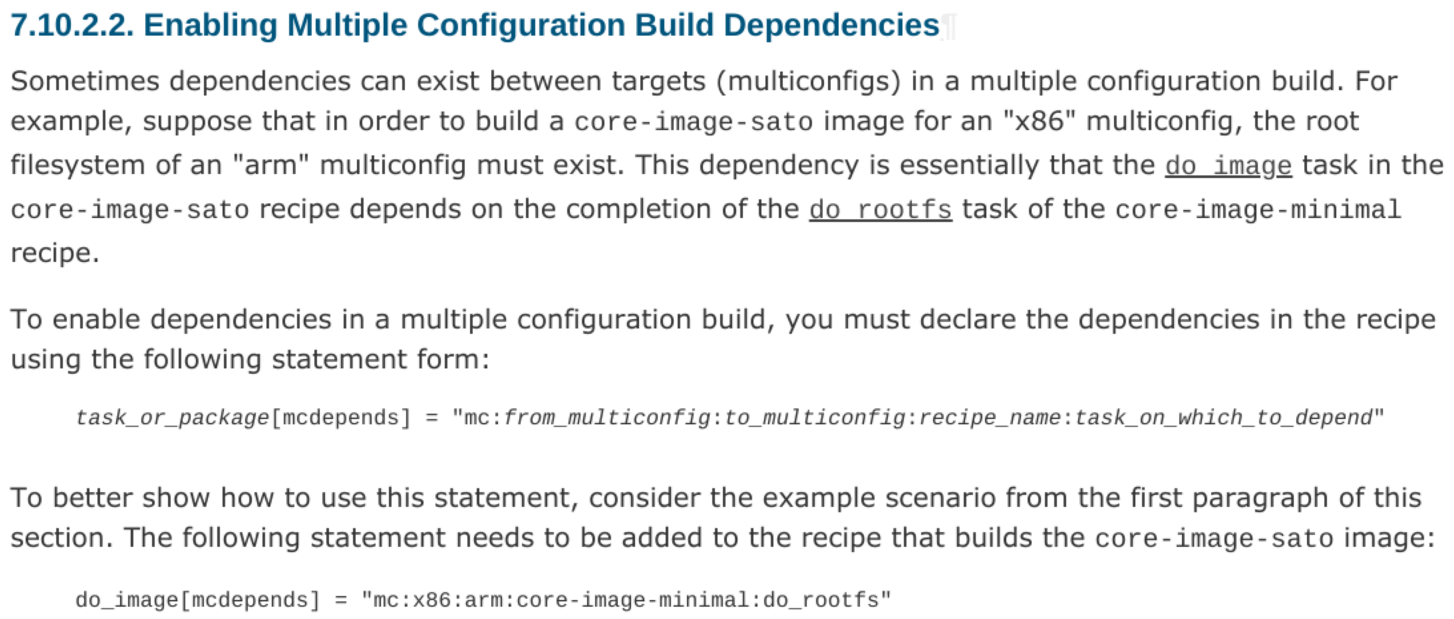
One Build To Rule Them All

Multiconfig Builds

You can use a single bitbake command to build multiple images or packages for different targets where each image or package requires a different configuration (multiple configuration builds). [1] https://www.yoctoproject.org/docs/latest/mega-manual/mega-manual.html



Multiconfig Depedencies



7.10.2.2. Enabling Multiple Configuration Build Dependencies¶

Sometimes dependencies can exist between targets (multiconfigs) in a multiple configuration build. For example, suppose that in order to build a `core-image-sato` image for an "x86" multiconfig, the root filesystem of an "arm" multiconfig must exist. This dependency is essentially that the `do_image` task in the `core-image-sato` recipe depends on the completion of the `do_rootfs` task of the `core-image-minimal` recipe.

To enable dependencies in a multiple configuration build, you must declare the dependencies in the recipe using the following statement form:

```
task_or_package[mcdepends] = "mc:from_multiconfig:to_multiconfig:recipe_name:task_on_which_to_depend"
```

To better show how to use this statement, consider the example scenario from the first paragraph of this section. The following statement needs to be added to the recipe that builds the `core-image-sato` image:

```
do_image[mcdepends] = "mc:x86:arm:core-image-minimal:do_rootfs"
```

Future Work

- Multiconfig Optimizations
  - Shared State Cache *
  - Parsing
- Support more Architectures/Ports on Meta-FreeRTOS
- Fine Tuning Meta-FreeRTOS
- Upstream tests and CI

Takeaways

- The Yocto Project / Bitbake provides scalability
- FreeRTOS was just used as a test case
- Support more OSs/Applications on Yocto?
- Unify workflows across teams
- Control over toolchain and get reproducibility

Thanks!
Q & A

Embedded Linux Conference Europe

Alejandro Hernandez
alejandr@xilinx.com
https://layers.openembedded.org/layerindex/branch/master/layer/meta-freertos/

XILINX