Introduction to Yocto

Advanced Embedded Linux Development

with **Dan Walkes**

University of Colorado **Boulder**

Learning objectives:

Introduce the BitBake tool

Introduce the Poky distribution

Yocto layer model

Yocto terminology

What is Yocto?

- Way to roll your own Linux **Distribution** for embedded devices
  - Not just root filesystem
  - Includes redistributable packages for each piece of software
- Launched in 2011
- Includes embedded build tools and an Embedded Distribution Poky (sounds like hockey)

What is Yocto?

- Build system is licensed MIT
- Uses text file configuration and “bitbake” tool based on Python/Bash
- Builds most everything from source, including all build tools
  - Used to support binary reproducibility

https://www.yoctoproject.org/
https://en.wikipedia.org/wiki/Yocto_Project
https://bootlin.com/pub/conferences/2016/elc/belloni-petazzoni-buildroot-oe/belloni-petazzoni-buildroot-oe.pdf
https://wiki.yoctoproject.org/wiki/Reproducible_Builds

Setting up the Yocto Environment

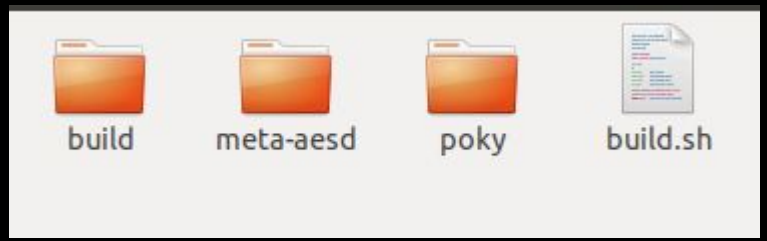
- Basic steps at https://docs.yoctoproject.org/brief-yoctoprojectqs/index.html
  - Clone the “poky” project a submodule in your repository
    - Use branch specified in assignment
    - git add/git commit with the submodule at the correct branch before running build.sh
  - **source poky/oe-init-build-env**
  - bitbake core-image-minimal
  - Work on something else for a few hours!

Setting up the Yocto Environment

- source poky/oe-init-build-env
  - Uses the source command (synonym for .)
- source <filename>
  - Any variables created or modified by the script <filename> will remain available after the script exits
  - Runs in the same subshell (whereas ./filename runs in a new shell)
- What does this mean?
  - poky/oe-init-build-env sets up environment variables for our current terminal session

Poky Build Directory

- A “build” directory at the root folder is created for you with the first “source poky/oe-init-build-env” step
- This is where all build output and log files will be generated



Yocto Terminology

- Builds are performed by the bitbake utility
- How do you bake bits?
  - Recipes are used to bake bits!
- Recipes are a set of instructions processed by the build engine.
- Recipes are defined in .bb and .inc text files

Yocto Terminology

- Recipes are defined in .bb and .inc text files
- .bb typically contains source and version information
- .inc contains build and deploy instructions, may contain python or bash fragments

Yocto Terminology

- Packages contain binary artifacts from the build
- Images are build outputs
  - Binary root filesystem image
  - Linux kernel image
  - u-boot or grub bootloader image

Yocto Layer Model

- Designed to support both collaboration and customization.
- Collection of directories on the filesystem virtually “layered” to make an equivalent build hierarchy

meta-aesd/recipes-aesd-assignments/images/core-image-aesd.bb
meta-aesd/recipes-aesd-assignments/aesd-assignments/aesd-assignments_git.bb
meta-aesd/recipes-core/busybox/busybox.bbappend

meta-oe/recipes-core/busybox/busybox.inc
meta-oe/recipes-core/images/core-image-base.bb
...

Filesystem equivalent

recipes-aesd-assignments/images/core-image-aesd.bb
recipes-aesd-assignments/aesd-assignments/aesd-assignments_get.bb
recipes-core/busybox/busybox.inc
recipes-core/busybox/busybox.bbappend
recipes-core/images/core-image-base.bb
...

https://www.yoctoproject.org/software-overview
https://www.slideshare.net/alexgonzalezgarcia/introduction-to-yocto
https://www.openembedded.org/Layers_FAQ
https://docs.yoctoproject.org/dev-manual/common-tasks.html?highlight=understanding+creating+layers#understanding-and-creating-layers

Yocto Layers

- Organized in a way to override or add to recipes
  - Might make the build work a slightly different way
    - Apply a patch for a specific version of hardware
    - Configure for a GUI vs terminal only
- Might add support for new recipes
  - In our case aesd-assignments
- Might add support for new images
  - In our case core-image-aesd

Adding a Yocto Layer

- source poky/oe-init-build-env
- bitbake-layers create-layer ../../meta-aesd
- bitbake-layers add-layer ../../meta-aesd
  - Already done for your starter repo
  - See meta-aesd/conf/layer.conf and ./build.sh

```
bitbake-layers show-layers | grep "meta-aesd" > /dev/null
layer_info=$?

if [ $layer_info -ne 0 ];then
        echo "Adding meta-aesd layer"
        bitbake-layers add-layer ../meta-aesd
else
        echo "meta-aesd layer already exists"
fi
```

Adding a Custom Image

- Add core-image-aesd.bb with single line “inherit core-image”
- Pulls a core set of packages (like busybox) into our build.

https://docs.yoctoproject.org/dev-manual/common-tasks.html?#customizing-images-using-custom-image-features-and-extra-image-features
https://wiki.yoctoproject.org/wiki/Building_your_own_recipes_from_first_principles#The_bbexample_recipe
https://github.com/cu-ecen-aeld/yocto-assignments-base/blob/master/meta-aesd/recipes-aesd-assignments/images/core-image-aesd.bb

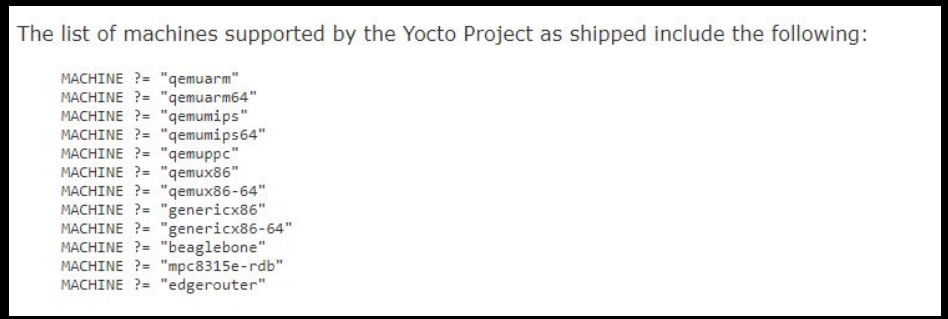
Yocto MACHINE

- How did we specify the target architecture in buildroot?
  - Specified in our defconfig file

```
DEFAULT_DEFCONFIG=configs/qemu_aarch64_virt_defconfig
```

```
echo "Using default defconfig ${DEFAULT_DEFCONFIG}"
make -C ${BUILDROOT} defconfig BR2_DEFCONFIG=${DEFAULT_DEFCONFIG} BR2_EXTERNAL=${EXTERNAL_REL_BUILDROOT}
```

- Yocto uses a MACHINE variable in build/conf/local.conf



The list of machines supported by the Yocto Project as shipped include the following:

```
MACHINE ?= "qemuarm"
MACHINE ?= "qemuarm64"
MACHINE ?= "qemumips"
MACHINE ?= "qemumips64"
MACHINE ?= "qemuppc"
MACHINE ?= "qemux86"
MACHINE ?= "qemux86-64"
MACHINE ?= "genericx86"
MACHINE ?= "genericx86-64"
MACHINE ?= "beaglebone"
MACHINE ?= "mpc8315e-rdb"
MACHINE ?= "edgerouter"
```

https://github.com/cu-ecen-5013/buildroot-assignments-base/blob/master/build.sh#L7
https://www.yoctoproject.org/docs/latest/ref-manual/ref-manual.html#var-MACHINE