XILINX

Building Bare Metal Toolchains

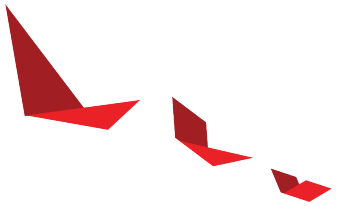
Crosstool-ng and Yocto Project

Mark Hatle
Senior Software Engineer
June 30th, 2020

Presentation Summary

Recently I was tasked to create a bare metal toolchain to create software for a variety of embedded processor architectures and configurations. Crosstool-ng is often used to create these toolchains, but Yocto Project SDK builder is capable of doing this as well. This presentation will compare booth crosstool-ng and the Yocto Project for this task, include my experience working with both tools, include Yocto Project configuration information and give the audience an understanding when they may want to use one tool vs the other.

Agenda



- Crosstool-ng
- Yocto Project SDK Builder
- Experience
- Yocto Project Configuration
- Recommendations
- Questions

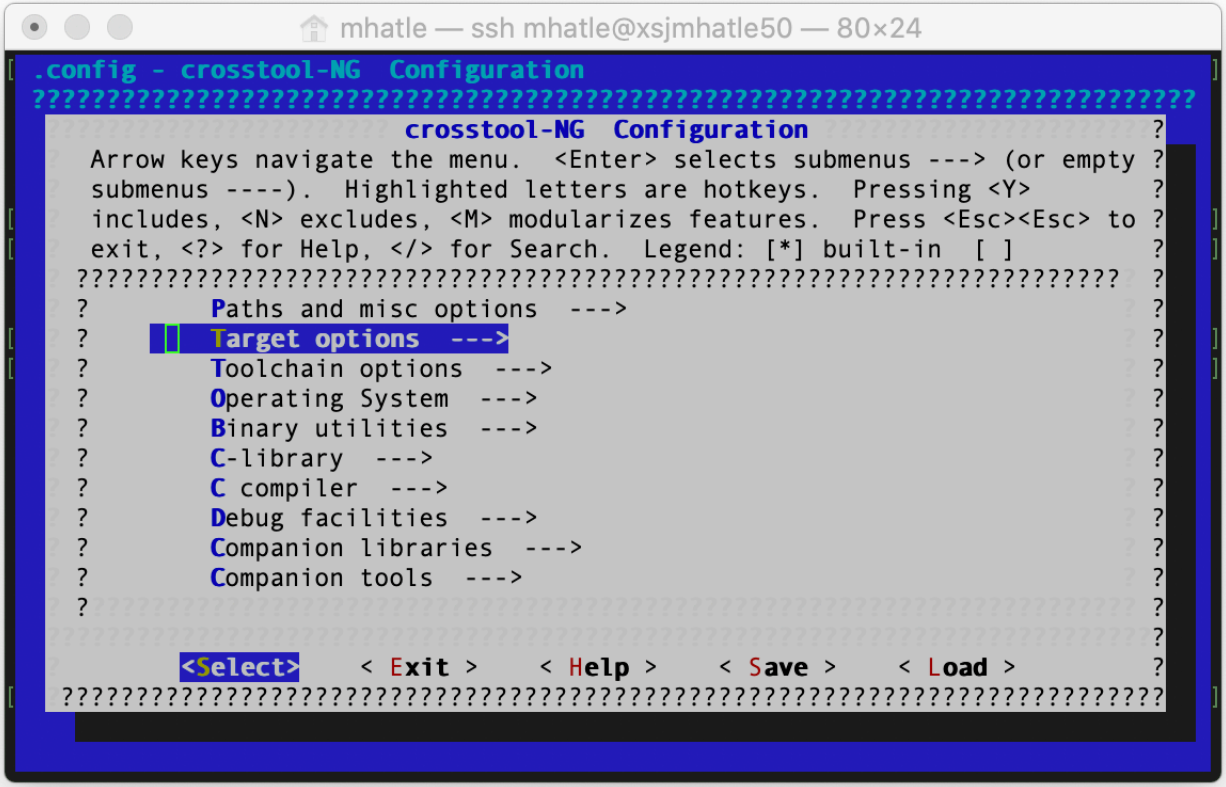
Crosstool-ng (http://crosstool-ng.github.io)

- “Crosstool-NG is a versatile (cross) toolchain generator. It supports many architectures and components and has a simple yet powerful menuconfig-style interface.”
- Latest release 1.24.0
- ct-ng menuconfig
  - Most items have help entry
- Good way to construct a toolchain, especially for beginners
  - ct-ng list-samples -- Many sample configurations
- Easily reproducible from source builds

Crosstool-ng (http://crosstool-ng.github.io)

- Runtime relocatable
  - Can be installed in one location and mounted/run from many
- Binaries are specific to the host environment they were built for
  - I.e. Binaries built on latest Ubuntu likely would not work on RHEL 7
  - Building for a different platform, i.e. Cygwin requires a separate Cygwin cross compiler to be available

Crosstool-ng



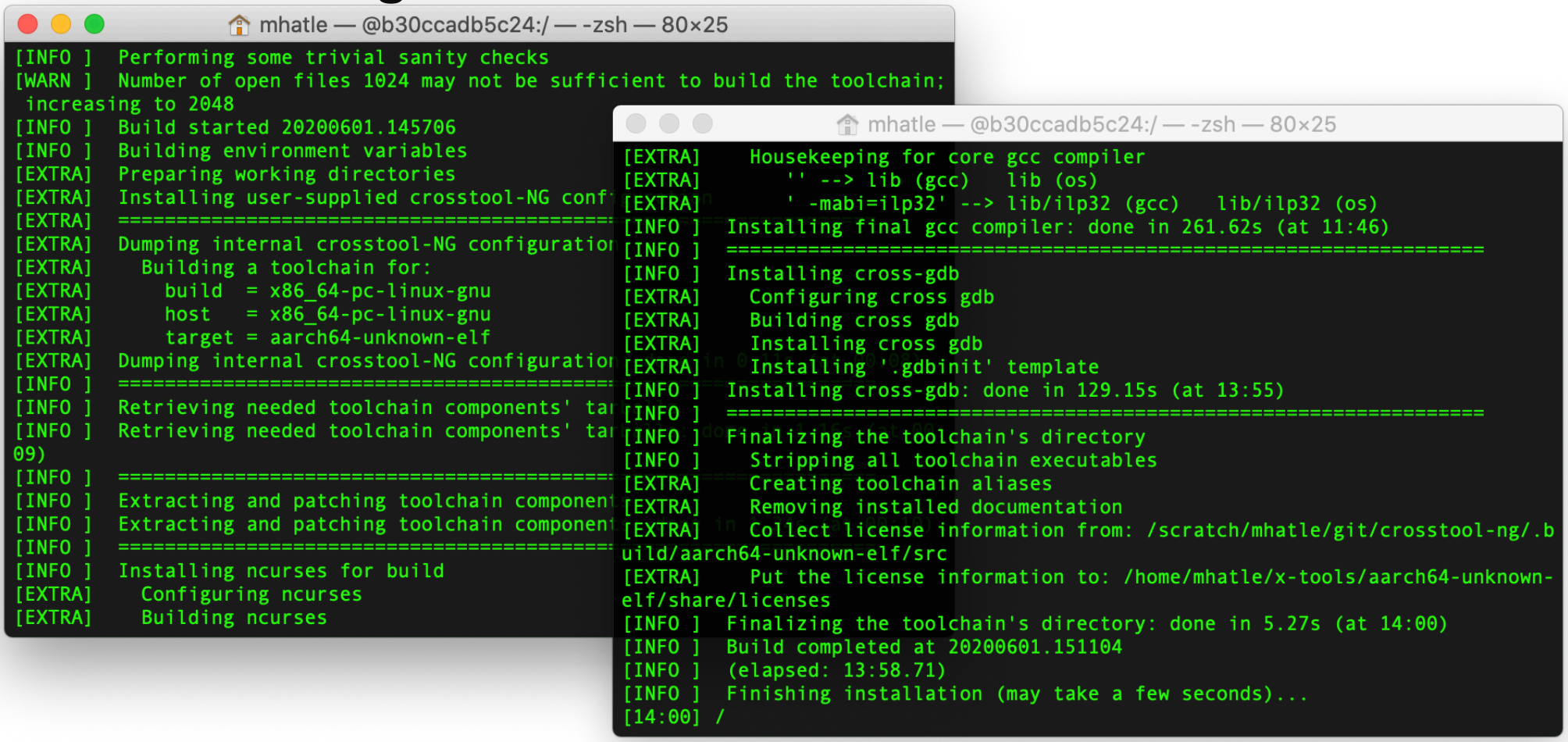
XILINX.

Crosstool-ng

```
…
CT_USE_PIPES=y
CT_EXTRA_CFLAGS_FOR_BUILD=""
CT_EXTRA_LDFLAGS_FOR_BUILD=""
CT_EXTRA_CFLAGS_FOR_HOST=""
CT_EXTRA_LDFLAGS_FOR_HOST=""
…
CT_ARCH="microblaze”
CT_ARCH_SUPPORTS_BOTH_MMU=y
CT_ARCH_SUPPORTS_BOTH_ENDIAN=y
CT_ARCH_DEFAULT_HAS_MMU=y
…
CT_ARCH_DEFAULT_BE=y
CT_ARCH_BE=y
CT_TARGET_CFLAGS=""
CT_TARGET_LDFLAGS=""
CT_ARCH_microblaze=y
```

```
CT_MULTILIB=y
CT_ARCH_USE_MMU=y
CT_ARCH_ENDIAN="big”
CT_ARCH_FLOAT=""
CT_USE_SYSROOT=y
CT_SYSROOT_NAME=""
CT_SYSROOT_DIR_PREFIX=””
…
CT_CC_GCC_HAS_LTO=y
CT_CC_GCC_USE_LTO=y
CT_CC_GCC_HAS_PKGVERSION_BUGURL=y
CT_CC_GCC_HAS_BUILD_ID=y
CT_CC_GCC_HAS_LNK_HASH_STYLE=y
…
```

Crosstool-ng



XILINX.

Yocto Project

- The Yocto Project is a full distribution build environment. Each distribution configuration is based on a local project configuration, distribution configurations, and machine (target) configuration.
- The Yocto Project, while Linux distributions is its historic target, can build different operating systems and even bare metal.
  - FreeRTOS
  - OpenAMP
  - Bare Metal
- Outputs include
  - Run-time images
  - SDK / eSDK

Yocto Project SDK

- The Yocto Project SDK purpose is to provide an application build environment.
- Targeted SDK
  - An SDK that matches an operating system runtime environment
- Defined SDK
  - An SDK where each component is defined to be included in the SDK
- SDKs can be multilib enabled
  - Multilibs are built independently of each other
  - Slower, but safer approach for complex configurations

Yocto Project SDK

- Self-extracting installation file
- Built to isolate the SDK environment from the host system
  - SDK includes its own glibc, and some runtime components
  - SDK can build its own cross-compilers for Cygwin and other environments, as needed
- Installation and runtime locations must be the same
  - Automatic runtime relocation is not supported

Yocto Project SDK

mhatle — @b30ccadb5c24:/ — ssh -o ServerAliveInterval=50 mhatle@xsjmhat...

```
[mhatle@xsjmhatle50$ . ./oe-init-build-env build-baremetal-tc

### Shell environment set up for builds. ###

You can now run 'bitbake <target>'

Common targets are:
    core-image-minimal
    core-image-sato
    meta-toolchain
    meta-ide-support

You can also run generated qemu images with a command like 'runqemu qemux86'

Other commonly useful commands are:
 - 'devtool' and 'recipetool' handle common recipe tasks
 - 'bitbake-layers' handles common layer tasks
 - 'oe-pkgdata-util' handles common target package tasks
[mhatle@xsjmhatle50$ bitbake-layers add-layer ../../meta-xilinx/meta-xilinx-bsp
[../../meta-xilinx/meta-xilinx-standalone
NOTE: Starting bitbake server...
mhatle@xsjmhatle50$ DISTRO=xilinx-standalone MACHINE=microblaze-tc bitbake meta
[-toolchain
Parsing recipes:   3% |#                                    | ETA:  0:15:01
```

mhatle — @b30ccadb5c24:/ — ssh -o ServerAliveInterval=50 mhatle@xsjmhat...

```
Currently 32 running tasks (1023 of 6592)  15% |####                          |
0: libmblem64bspfpd-gcc-cross-microblazeel-9.2.0-r0 do_compile - 6m10s (pid 1753
3)
1: libmblem64bspm-gcc-cross-microblazeel-9.2.0-r0 do_compile - 6m10s (pid 17792)
2: libmblem64bspmfpd-gcc-cross-microblazeel-9.2.0-r0 do_compile - 6m10s (pid 174
24)
3: libmblem64bsfpd-gcc-cross-microblazeel-9.2.0-r0 do_compile - 6m9s (pid 23631)
4: libmblem64bsmfpd-gcc-cross-microblazeel-9.2.0-r0 do_compile - 6m9s (pid 19095
)
5: libmblem64bsm-gcc-cross-microblazeel-9.2.0-r0 do_compile - 6m9s (pid 24643)
6: libmblem64bsp-gcc-cross-microblazeel-9.2.0-r0 do_compile - 6m8s (pid 27456)
7: libmblem64pfpd-gcc-cross-microblazeel-9.2.0-r0 do_compile - 6m6s (pid 2442)
8: libmblem64pmfpd-gcc-cross-microblazeel-9.2.0-r0 do_compile - 6m6s (pid 32735)
9: libmblem64pm-gcc-cross-microblazeel-9.2.0-r0 do_compile - 6m5s (pid 8218)
10: libmblem64mfpd-gcc-cross-microblazeel-9.2.0-r0 do_compile - 6m4s (pid 27654)
11: libmblem64fpd-gcc-cross-microblazeel-9.2.0-r0 do_compile - 5m59s (pid 14200)
12: libmblem64m-gcc-cross-microblazeel-9.2.0-r0 do_compile - 5m58s (pid 19583)
13: libmblem64p-gcc-cross-microblazeel-9.2.0-r0 do_compile - 5m58s (pid 20208)
14: libmblem64bs-gcc-cross-microblazeel-9.2.0-r0 do_compile - 5m57s (pid 28685)
15: libmblebspfpd-gcc-cross-microblazeel-9.2.0-r0 do_compile - 5m57s (pid 32185)
16: libmblebsfpd-gcc-cross-microblazeel-9.2.0-r0 do_compile - 5m57s (pid 6568)
17: libmblebsm-gcc-cross-microblazeel-9.2.0-r0 do_compile - 5m56s (pid 10370)
18: libmblefpd-gcc-cross-microblazeel-9.2.0-r0 do_compile - 5m56s (pid 17320)
19: libmblebspm-gcc-cross-microblazeel-9.2.0-r0 do_compile - 5m56s (pid 2421)
20: libmblebsp-gcc-cross-microblazeel-9.2.0-r0 do_compile - 5m56s (pid 13401)
21: libmblem-gcc-cross-microblazeel-9.2.0-r0 do_compile - 5m55s (pid 17597)
22: libmblep-gcc-cross-microblazeel-9.2.0-r0 do_compile - 5m55s (pid 17929)
23: libmblebsmfpd-gcc-cross-microblazeel-9.2.0-r0 do_compile - 5m55s (pid 5761)
24: libmblepfpd-gcc-cross-microblazeel-9.2.0-r0 do_compile - 5m55s (pid 15979)
25: libmblepm-gcc-cross-microblazeel-9.2.0-r0 do_compile - 5m55s (pid 16301)
26: libmblebspmfpd-gcc-cross-microblazeel-9.2.0-r0 do_compile - 5m55s (pid 30473
)
27: libmblemfpd-gcc-cross-microblazeel-9.2.0-r0 do_compile - 5m54s (pid 16513)
28: libmblebs-gcc-cross-microblazeel-9.2.0-r0 do_compile - 5m53s (pid 22933)
29: libmblepmfpd-gcc-cross-microblazeel-9.2.0-r0 do_compile - 5m52s (pid 14720)
30: python3-native-3.7.6-r0 do_populate_sysroot - 2m33s (pid 11507)
31: gcc-crosssdk-x86_64-oesdk-linux-9.2.0-r0 do_compile - 11s (pid 4779)
```


XILINX.

Yocto Project SDK Configuration xilinx-standalone.conf

```
DISTRO_NAME = "Xilinx Standalone Distro”
DISTRO_VERSION = "1.0”
TARGET_VENDOR = "-xilinx”
TCLIBC = "newlib”
TCLIBCAPPEND =”"
SDK_VERSION = "xilinx-standalone”

# Hold this until it gets merged in core, we need libc.a and libgloss.a for cross-Canadian
LIBC_DEPENDENCIES_append = " newlib-staticdev libgloss-staticdev”

# Clear defaults
DISTRO_FEATURES_BACKFILL_xilinx-standalone = ""
VIRTUAL-RUNTIME_init_manager_xilinx-standalone = ""
PREFERRED_PROVIDER_virtual/kernel = "linux-dummy”

# No cached configsite files
TOOLCHAIN_NEED_CONFIGSITE_CACHE = ""

# Workaround for pulling in nativesdk-mingw-w64-winpthreads
TOOLCHAIN_HOST_TASK_append_sdkmingw32 = " nativesdk-mingw-w64-winpthreads"
```

Yocto Project SDK Configuration microblaze-tc.conf

```
require conf/multilib.conf
require conf/machine/include/microblaze/arch-microblaze.inc
require conf/machine/include/baremetal-tc.conf

# Define all of the multilibs supported by this configuration
MULTILIB_GLOBAL_VARIANTS =
"${@extend_variants(d,'MULTILIBS','multilib')}"
MULTILIBS += "multilib:libmble"
MULTILIBS += "multilib:libmbbs"
MULTILIBS += "multilib:libmbp"
...
MULTILIBS += "multilib:libmblem64bspmfpd"

# Base configuration
# CFLAGS:
DEFAULTTUNE = "microblaze"
AVAILTUNES += "microblaze"
BASE_LIB_tune-microblaze = "lib"
TUNE_FEATURES_tune-microblaze = "microblaze bigendian"
PACKAGE_EXTRA_ARCHS_tune-microblaze =
"${TUNE_PKGARCH}"

# le
# CFLAGS: -mlittle-endian
DEFAULTTUNE_virtclass-multilib-libmble = "microblazele"

AVAILTUNES += "microblazele"
BASE_LIB_tune-microblazele = "lib/le"
TUNE_FEATURES_tune-microblazele = "microblaze"
PACKAGE_EXTRA_ARCHS_tune-microblazele =
"${TUNE_PKGARCH}"

# bs
# CFLAGS: -mxl-barrel-shift
DEFAULTTUNE_virtclass-multilib-libmbbs = "microblazebs"

AVAILTUNES += "microblazebs"
BASE_LIB_tune-microblazebs = "lib/bs"
TUNE_FEATURES_tune-microblazebs = "microblaze bigendian
barrel-shift"
PACKAGE_EXTRA_ARCHS_tune-microblazebs =
"${TUNE_PKGARCH}"

…
```

Yocto Project SDK Changes

- Binutils
  - Set different defaults to match prior toolchains
- GCC
  - Set different default to match prior toolchains
  - Restore some previously disabled newlib options (i.e. sysroot settings)
  - Only build SDK GCC *once*, create symlinks for other multilibs (gnu-toolchain-canadian.bb)
- Newlib/Libgloss
  - Adjust defaults
  - Workaround an issue where multilibs conflicted
  - Workaround an issue where the libgloss/newlib dependency wasn't multilib aware

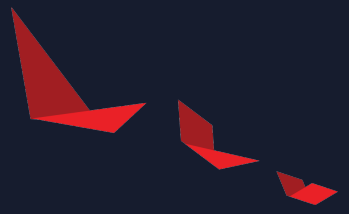
Crosstool-ng vs Yocto Project SDK

Crosstool-ng

- Easy to use sample configurations
- Functionality limited to toolchains
- Host operating system library dependencies
- Runtime relocatable
- Mingw, with external cross compiler

Yocto Project SDK

- Linux sample configurations, but not bare metal
- Full features, toolchains, libc and application libraries
- Host operating system separation
- Not runtime relocatable
- Supports mingw output



My experiences crosstool-ng to Yocto Project

XILINX

Xilinx Bare Metal Toolchains

- Transitioned from Crosstool-ng to Yocto Project SDK to unify toolchain source and testing
- Transition was not painless from a development perspective
  - Differences in the way Crosstool-ng and Yocto Project configured toolchain components
- Lots of questions about multilib configurations
  - Things had always been done a certain way, and people who made those decisions were either external or no longer with the company
- Initial goal was compatibility with former crosstool-ng and custom ARM toolchain
  - This includes pseudo runtime relocation capable

Xilinx Bare Metal Multilibs

- See: https://github.com/Xilinx/meta-xilinx/blob/rel-v2020.1/meta-xilinx-bsp/conf/machine/aarch32-tc.conf
- Aarch32
  - Standard ARM (A profile) 32-bit instruction set
  - Multilib config based on GNU/ARM defaults
  - 17 multilibs defined
    - aarch32, armv5tesoftfp, armv5tehard, armnofp, armv7nofp, armv7fpsoftfp, armv7fphard, armv7anofp, armv7afpsoftfp, armv7afpthf, armv7asimdsoftfp, armv7asimdhard, armv7vesimdsoftfp, armvtvesimdhf, armv8anofp, armv8asimdsoftfp, armv8asimdhard
    - Custom defined to match GNU/ARM settings

Xilinx Bare Metal Multilibs

- See: https://github.com/Xilinx/meta-xilinx/blob/rel-v2020.1/meta-xilinx-bsp/conf/machine/arm-rm-tc.conf
- ARM R/M
  - Real-time Profile and M Microcontroller Profile
  - Multilib config based on GNU/ARM defaults
  - 22 multilibs defined
    - armrm, armv5tesoftfp, armv5tehard, armnofp, armv7nofp, armv7fpsoftfp, armv7fphard, armv6mnofp, armv7mnofp, armv7emnofp, armv7emfpsoftfp, armv7emfphard, armv7emdpsoftfp, armv7emdphard, armv8mbasenofp, armv8mmainnofp, armv8mmainfpsoftfp, armv8mmainfphard, armv8mmaindpsoftfp, armv8mmaindphard
    - Note these are custom defined, see include/tune-cortexrm.inc for ‘armrm’ definition

Xilinx Bare Metal Multilibs

- See: https://github.com/Xilinx/meta-xilinx/blob/rel-v2020.1/meta-xilinx-bsp/conf/machine/aarch64-tc.conf
- Aarch64
  - Standard ARM (A profile) 64-bit instruction set
  - Multilib config based on GNU/ARM defaults
  - 2 multilibs defined
    - cortexa72-cortexa53, cortexa72-cortexa53-ilp32
  - This is a generic 64-bit arm configuration, but the requirement was that it run with acceptable performance on both a72 and a53.

Xilinx Bare Metal Multilibs

- See: https://github.com/Xilinx/meta-xilinx/blob/rel-v2020.1/meta-xilinx-bsp/conf/machine/microblaze-tc.conf
- Microblaze (microblaze-tc.conf)
  - Common microblaze instruction set permutations
  - Multilib config based on prior Xilinx defaults
  - 48 multilibs defined
    - microblaze, microblazele, microblazebs, microblazep, microblazem, microblazefpd, microblazemfpd, microblazepm, microblazepfpd, microblazepmfpd, microblazebsp, microblazebsm, microblazebsfpd, microblazebsmfpd, microblazebspm, microblazebspfpd, microblazebspmfpd, microblazele64, microblazelebs, microblazelep, microblazelem, microblazelefpd, microblazelemfpd, microblazelepm, microblazelepfpd, microblazelepmfpd, microblazelebsp, microblazelebsm, microblazelebsfpd, microblazelebsmfpd, microblazelebspm, microblazelebspfpd, microblazelebspmfpd, microblazele64bs, microblazele64p, microblazele64m, microblazele64fpd, microblazele64mfpd, microblazele64pm, microblazele64pfpd, microblazele64pmfpd, microblazele64bsp, microblazele64bsm, microblazele64bsfpd, microblazele64bsmfpd, microblazele64bspm, microblazele64bspfpd, microblazele64bspmfpd
  - Microblaze is a configurable FPGA processor. It has numerous configurable traits and would exceed even these 48 to do all permutations!

Xilinx Yocto Project Bare Metal Toolchain Configuration

- See: https://github.com/Xilinx/meta-xilinx/tree/rel-v2020.1/meta-xilinx-standalone/recipes-devtools
- Binutils:
  - Disable GOLD as LD, disable gprof, disable shared, enable-lto, enable-static, enable-multilib
  - ARM: enable-interwork
  - Microblaze: disable-initfini-array

Xilinx Yocto Project Bare Metal Toolchain Configuration

- See: https://github.com/Xilinx/meta-xilinx/tree/rel-v2020.1/meta-xilinx-standalone/recipes-devtools
- GCC:
  - Disable-libstdcxx-pch, with-newlib, disable-threads, enable-plugins, with-gnu-as, disable-libitm
  - Aarch64: disable-multiarch, with-arch=armv8-a
  - Arm: with-multilib-list={aprofile or rmprofile}
  - Arm R/M: disable-tls, disable-decimal-float
  - Microblaze: enable-target-optspace, without-long-double-128, disable-initfini-array, disable-__cxa_atexit
  - To emulate multilibs, need to symlink each individual multilib into a common area, etc…
  - Hack to only build GCC *once* and then symlink multilib versons to the main

Xilinx Yocto Project Bare Metal Toolchain Configuration

- See: https://github.com/Xilinx/meta-xilinx/tree/rel-v2020.1/meta-xilinx-standalone/recipes-core
- Newlib/Libgloss:
  - Xilinx baremetal implements device specific items using libxil, but we use libgloss to provide the framework
  - enable-newlib-io-c99-formats, enable-newlib-io-long-long, enable-newlib-io-float, enable-newlib-io-long-double, ***disable-newlib-supplied-syscalls***
  - Libgloss didn't understand multilib depends automatically for some reason
    - DEPENDS_append = " ${MLPREFIX}newlib"

Lessons Learned

- The more multilibs, the longer the initial project parse time.
  - Microblaze parse time is nearly an hour (48 multilibs)
    - Development workaround, build for only one multilib!
  - Mingw32 builds are sequential with the Linux version, using a shared sstate-cache
    - These take roughly 15 minutes on the same machine for the mingw parts

Time	Parse	mlibs	Build Step	Ct-ng time	Ct-ng mlibs
84m	8m	48	Microblaze-tc for Linux	32m	18
46m	1m28s	17	Aarch32-tc for Linux	13m	3
26m	8s	2	Aarch64-tc for Linux	14m	2
51m	8s	22	Arm-rm-tc for Linux	10m	1

Intel Xeon Gold 6130 (32 thread) @ 2.10 GHz w/ 128 GB ram running Ubuntu 18.04

Recommendations

- For a quick toolchain, firmware users, etc. Crosstools-ng is far easier.
- You can still use crosstool-ng with a common source base with the Yocto Project, but configuration switches are different by default.
- Yocto Project makes the most sense if you need Cygwin, or toolchains that run standalone with their own environments.
- Yocto Project MAY take more time, but will provide an easy way for common source and configuration switches with Linux builds. May simplify defect handling and propagation of fixes/features between Yocto Project systems and baremetal.


XILINX
Thank You
© Copyright 2020 Xilinx