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
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
...

© Copyright 2020 Xilinx
Crosstool-ng

[INFO] Performing some trivial sanity checks
[WARN] Number of open files 1024 may not be sufficient to build the toolchain; increasing to 2048
[INFO] Build started 20200601.145706
[INFO] Building environment variables
[EXTRA] Preparing working directories
[EXTRA] Installing user-supplied crosstool-NG config
[EXTRA] Dumping internal crosstool-NG configuration
[EXTRA] Building a toolchain for:
  build = x86_64-pc-linux-gnu
  host = x86_64-pc-linux-gnu
  target = aarch64-unknown-elf
[EXTRA] Dumping internal crosstool-NG configuration
[INFO] Retrieving needed toolchain components' tags
[INFO] Retrieving needed toolchain components' tags
[INFO] Extracting and patching toolchain component
[INFO] Extracting and patching toolchain component
[INFO] Installing ncurses for build
[EXTRA] Configuring ncurses
[EXTRA] Building ncurses

[EXTRA] Housekeeping for core gcc compiler
[EXTRA] ' --libgcc --libos' --> lib/ilp32 (gcc) lib/ilp32 (os)
[INFO] Installing final gcc compiler: done in 261.62s (at 11:46)
[INFO] Installing cross-gdb
[EXTRA] Configuring cross gdb
[EXTRA] Building cross gdb
[EXTRA] Installing cross gdb
[EXTRA] Installing '.gdbinit' template
[INFO] Installing cross-gdb: done in 129.15s (at 13:55)
[INFO] Finalizing the toolchain's directory
[INFO] Stripping all toolchain executables
[EXTRA] Creating toolchain aliases
[EXTRA] Removing installed documentation
[EXTRA] Collect license information from: /scratch/mhatle/git/crosstool-ng/build/aarch64-unknown-elf/src
[EXTRA] Put the license information to: /home/mhatle/x-tools/aarch64-unknown-elf/share/licenses
[INFO] Finalizing the toolchain's directory: done in 5.27s (at 14:00)
[INFO] Build completed at 20200601.151104
[INFO] (elapsed: 13:58.71)
[INFO] Finishing installation (may take a few seconds)...
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

You can now run 'bitbake <target>'

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

You can also run generated qemu images with a command like 'runqemu qemu-x86'

Other commonly useful commands are:
- 'devtool' and 'recipe-tool' handle common recipe tasks
- 'bitbake-layers' handles common layer tasks
- 'oe-pkgdata-util' handles common target package tasks

NOTE: Starting bitbake server...

Parsing recipes: 3% |# |
| ETA: 0:15:01 |

© Copyright 2020 Xilinx
Yocto Project SDK Configuration
xilinx-standalone.conf

DISTRO_NAME = "Xilinx Standalone Distro"
DISTRO_VERSION = "1.0"
TARGET_VENDOR = "-xilinx"
TCLIBC = "newlib"
TCLIBCAPPPEND ="
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:libblem64bspmfpe"

# 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 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


- **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, armv7vesimdsofff, armv7vesimdhf, armv8anofp, armv8asimdsoftfp, armv8asimdhard
    - Custom defined to match GNU/ARM settings
Xilinx Bare Metal Multilibs


- 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, armv8mmaindpphard
    - Note these are custom defined, see include/tune-cortexrm.inc for ‘armrm’ definition
Xilinx Bare Metal Multilibs


- 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


- Microblaze (microblaze-tc.conf)
  - Common microblaze instruction set permutations
  - Multilib config based on prior Xilinx defaults
  - 48 multilibs defined
    - microblaze, microblazele, microblazesbs, microblazep, microblazem, microblazefpd, microblazemfpd, microblazepm, microblazepfpd, microblazepmfpd, microblazebsp, microblazebsm, microblazebsfpd, microblazebsmfpd, microblazebspm, microblazebsfpm, microblazebspmfpd, microblazele64, microblazelebs, microblazelepm, microblazelems, microblazelefpd, microblazelempf, microblazelepmfpd, microblazelebsmp, microblazelebsfpm, microblazelebspmfpd, microblazelebspmfpd, microblazele64m, microblazele64fpd, microblazele64mfpd, microblazele64pm, microblazele64pfpd, microblazele64pmfpd, microblazele64bsp, microblazele64bsm, microblazele64bsfpd, microblazele64bsmfpd, microblazele64bspm, microblazele64bspmpd, 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


- 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


- 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 versions to the main
Xilinx Yocto Project Bare Metal Toolchain Configuration


- Newlib/Libgloss:
  - Xilinx baremetal implements device specific items using libxil, but we use libgloss to provide the framework
  - 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 multlib!
  - 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

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

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.