BUILDING EMBEDDED LINUX SYSTEMS WITH CLANG

KHEM RAJ

EMBEDDED LINUX CONFERENCE 2016

SAN DIEGO, CA

AGENDA

- Introduction to clang
- Overview of features and goals
- Clang for Embedded Linux application
- Using Clang as system compiler
- Building Distributions
- Using Clang SDKs
- Additional Clang Tools
- Using Clang runtime

INTRODUCTION TO CLANG

- Native compiler FrontEnd to LLVM Infrastructure
  - Supports C/C++ and Objective-C
  - The LLVM Project is a collection of modular and reusable compiler and toolchain technologies. - llvm.org
- First release in 2003
- Latest Release 3.8.0 (March 2016)
- Pronounced as /klaNG/

THIS IS NOT ABOUT CLASH OF TITANS !!

MISSING IN CLANG, AVAILABLE IN GCC

- You surely need GCC ....
  - GCC supports more language front-ends
    - ADA, Java, Fortran, Golang
- Supports more targets
  - SH, AVR32, ARC, old ARM <ARMv5...
  - https://gcc.gnu.org/backends.html

CLANG – FEATURES AND GOALS

- Newer codebase designed with C++ to support API based architecture
- Focuses on making it light and fast
- User friendly diagnostics
- offers fix-it hints, highlights

```
kraj@haswell ~ % aarch64-poky-linux-musl-clang --sysroot=/opt/poky/2.0+snapshot/sysroots/aarch64-poky-linux-musl -Ofast
test.c -c
test.c:9:21: warning: implicit declaration of function 'canonicalize_file_name' is invalid in C99 [-Wimplicit-function-
declaration]
    resolved_path = canonicalize_file_name(path);
                    ^
test.c:9:19: warning: incompatible integer to pointer conversion assigning to 'char *' from 'int' [-Wint-conversion]
    resolved_path = canonicalize_file_name(path);
                  ^ ~~~~~~~~~~~~~~~~~~~~~~~~~~~
2 warnings generated.
```

CLANG – FEATURES AND GOALS

- GCC compatibility
  - All extensions are recognized and marked as extension diagnostics
- IDE integration
- Uses LLVM BSD license
- Language conformance, ISO C, C++

CLANG – FEATURES AND GOALS (DIAGNOSTICS)

- Detected errors

```
/home/ubuntu/work/oe/openembedded-core/build/tmp-glibc/work/aarch64-oe-linux/xf86-video-omap/2_0.4.3-r0/xf86-video-omap-0.4.3/src/
drmmode_display.c:780:30: error: use of logical '&&' with constant operand
      [-Werror,-Wconstant-logical-operand]
              if (props && (props->flags && DRM_MODE_PROP_ENUM)) {
                                         ^  ~~~~~~~~~~~~~~~~~~
```

- Resulted in https://github.com/freedreno/xf86-video-freedreno/commit/8008da3ba97bf35a1ddd617401dcea48f4b1834f

```
/mnt/oe/build/tmp-glibc/work/raspberrypi2-oe-linux-gnueabi/userland/git-r5/git/interface/mmal/util/mmal_param_convert.c:47:7: error:
variable 'i' is incremented both in the loop header and in the loop body [-Werror,-Wfor-loop-analysis]
      i++;
      ^
```

- Fixed in https://github.com/raspberrypi/userland/pull/292/commits/a4a9286da4e864743e393d1fd2cee7ac963f3c6b

WHO IS PLAYING WITH CLANG

- Debian experimental
  - Optional compiler ~90% packages can compile
- LLVMLinux
  - Compile Linux Kernel with Clang
- The ELLCC Embedded Compiler Collection
- FreeBSD
- OpenMandriva
- OpenEmbedded/Yocto Project
- ....

USING CLANG FOR EMBEDDED LINUX

- Embedded Linux systems are cross-compiled (mostly)
  - Requires not only cross-compiler but cross toolchains
- Clang itself is cross compiler
- Universal Driver
  - Defines a single binary to invoke
  - set of cmdline options instruct driver to invoke the right tools pipeline

USING CLANG FOR EMBEDDED LINUX APPLICATIONS – PREBUILT TOOLCHAINS

- Cross compiling applications
  - install clang on your host distribution (Debian, Arch ..)
  - Download prebuilt toolchain from Yocto Project
    - http://autobuilder.yoctoproject.org/pub/nightly/CURRENT/toolchain/x86_64/
  - Linaro toolchain releases for arm
    - https://releases.linaro.org/components/toolchain/binaries/latest-5/arm-linux-gnueabihf/
  - Install and add the cross toolchain to PATH

```
% /usr/bin/clang --target=aarch64 -ccc-gcc-name aarch64-poky-linux-gcc hello.cpp --sysroot=/opt/poky/2.0+snapshot/
sysroots/aarch64-poky-linux
```

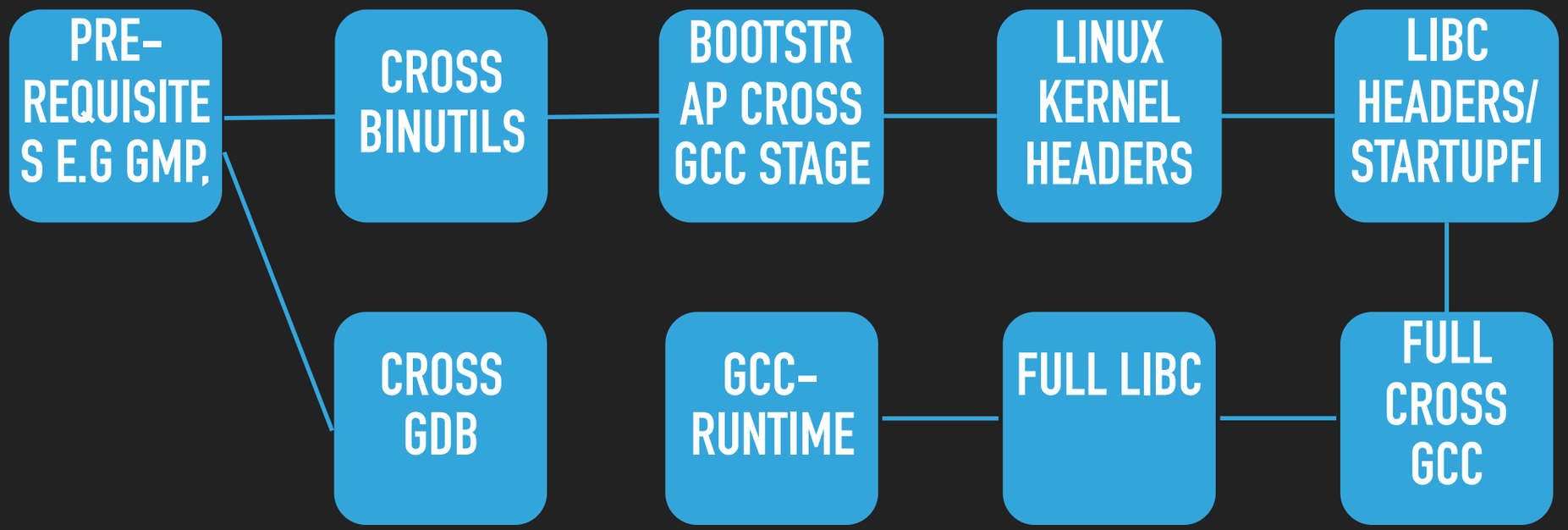
USING CLANG FOR EMBEDDED LINUX APPLICATIONS

- This would _only_ compile the given application with clang
  - Rest of system is still precompiled
    - GNU binutils will be used for linking and assembling
- Same setup can be leveraged for building Linux kernel
  - Export the CROSS_COMPILE and CC variables and its ilk correctly.

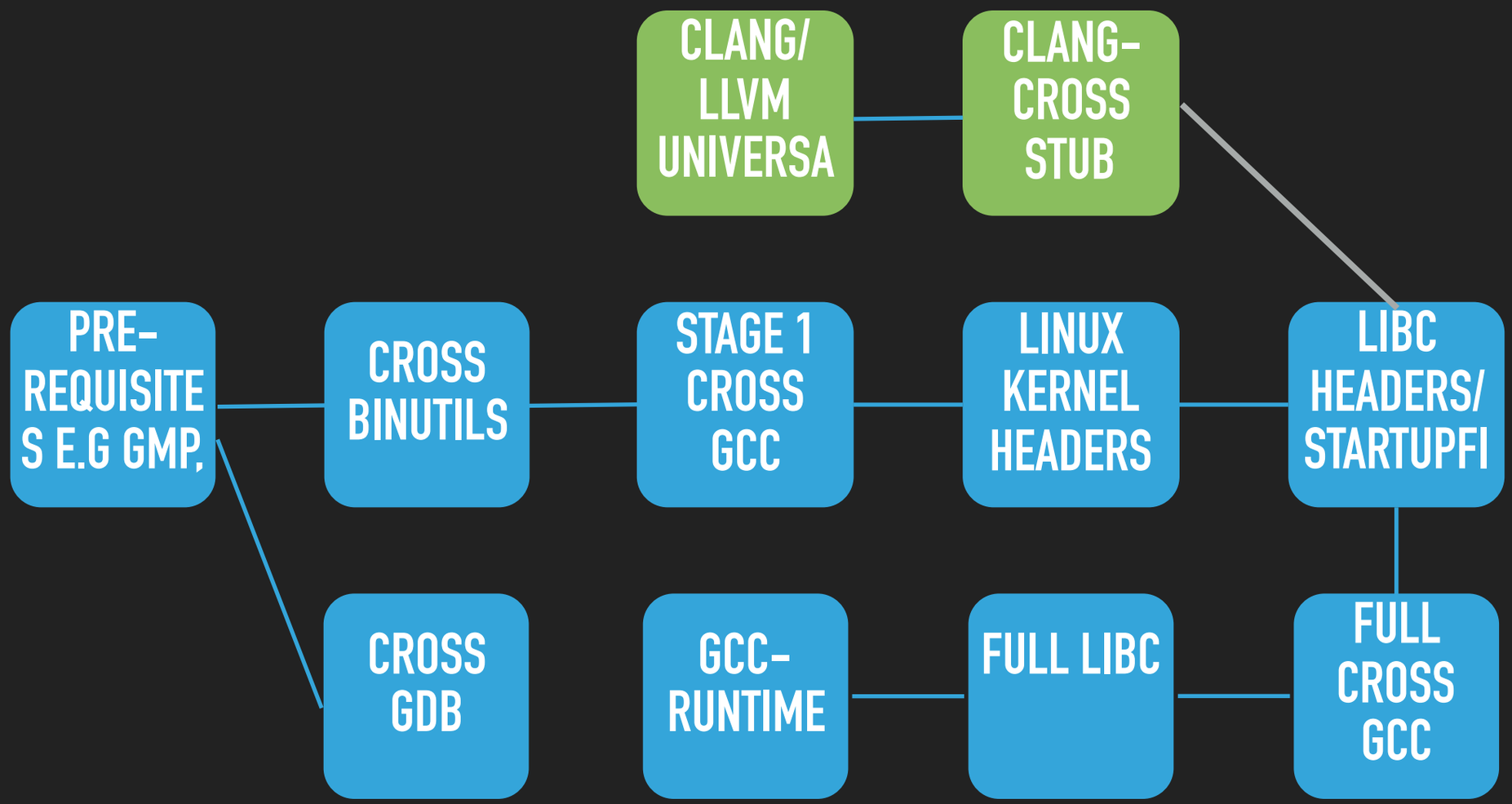
USING CLANG FOR EMBEDDED LINUX – PLATFORM

- Clang can not _yet_ build every bit of Embedded Linux Platform
- Linux Kernel effort
  - http://llvm.linuxfoundation.org/index.php/Main_Page
- System C library e.g. glibc does not compile with clang
  - https://sourceware.org/glibc/wiki/GlibcMeetsClang

TOOLCHAIN BUILD SEQUENCE



TOOLCHAIN BUILD SEQUENCE – CLANG



USING CLANG FOR EMBEDDED LINUX – PLATFORM

- Hybrid approach is needed (both GCC and Clang)
- Chromium OS
  - has overlays for clang
- OpenEmbedded
  - provides a layer meta-clang

USING CLANG FOR EMBEDDED LINUX – PLATFORM

- OpenEmbedded approach
  - meta-clang - when added switches default system compiler to clang
- Defines TOOLCHAIN variable (one of gcc, clang)
  - gcc - Enable gcc as default compiler for the package
  - clang - Enable clang as default compiler for package

USING CLANG FOR EMBEDDED LINUX – OPENEMBEDDED

- OE Clang Layer
  - https://github.com/kraj/meta-clang
- Certain packages known to not compile with clang are excluded
  - https://github.com/kraj/meta-clang/tree/master/recipes-excluded/nonclangable
    - Sets TOOLCHAIN = "gcc"

USING CLANG FOR EMBEDDED LINUX – OPENEMBEDDED

- Non-clangable recipes
  - Use specific gcc extensions not implemented in clang
    - Nested functions
    - VLAs in structs
- Has been fixed but patches not accepted
- Has been fixed but not updated in OE yet
- Has valid diagnostics
- Laziness..

TEXT

USING CLANG FOR EMBEDDED LINUX – OPENEMBEDDED

```
$ git clone git://git.yoctoproject.org/poky
```

```
$ cd poky
```

```
$ git clone git://github.com/kraj/meta-clang
```

```
$ . ./oe-init-build-env
```

```
$ bitbake-layers add-layer ../meta-clang
```

USING CLANG FOR EMBEDDED LINUX – OPENEMBEDDED

- Can build images
  - core-image-sato - X based Graphical image
  - core-image-minimal - Small console image
- Generates SDK for application development
  - bitbake -cpopulate_sdk core-image-minimal
    - Self installing SDK is
    - tmp/deploy/sdk/poky-glibc-x86_64-core-image-minimal-aarch64-toolchain-2.0+snapshot.sh
- Installing SDK
  - ./tmp/deploy/sdk/poky-glibc-x86_64-core-image-minimal-aarch64-toolchain-2.0+snapshot.sh

USING SDKS

- Using SDK
  - Setup Environment

```
. /opt/poky/2.0+snapshot/environment-setup-aarch64-poky-linux
```

- SDK contains both clang and gcc cross compilers
  - CC,CXX,CPP variables for gcc based cross compilers
  - CLANGCC, CLANGCXX,CLANGCPP for clang based c/c++ compiler

USING SDK – AUTOTOOLS BASED APPLICATIONS

- Building GNU hello world
  - wget http://ftp.gnu.org/gnu/hello/hello-2.10.tar.gz
  - tar xf hello-2.10.tar.gz
  - cd hello-2.10
  - . /opt/poky/2.0+snapshot/environment-setup-aarch64-poky-linux-musl
  - CC=${CLANGCC} ./configure –host=aarch64-poky-linux
  - make V=1
  - make install DESTDIR=/tmp/hello
  - scp /tmp/hello/usr/local/bin/hello <target>

USING SDK – KERNEL

- Building llvmlinux kernel

```
$ git clone git://git.linuxfoundation.org/llvmlinux/kernel.git llvmlinux

$ cd llvmlinux

$ make ARCH=arm64 CC=${CLANGCC} LDFLAGS="" defconfig

$ make ARCH=arm64 CC=${CLANGCC} LDFLAGS="" -j vmlinux
```

- It ends in compiler errors :(
  - What have you been waiting for - Fix it!!

USING CLANG FOR EMBEDDED LINUX – SYSTEM

- Thus far
  - Clang/Clang++ for Compiler
- Everything else remains same
  - System C library
  - Compiler C/C++ runtime

CLANG – ADDITIONAL TOOLS

- Clang Static Analyzer http://clang-analyzer.llvm.org/
  - Static analysis of musl (C library)
    - Configure

/a/builder/home/kraj/work/oe/musl/configure --enable-debug --target=arm CC=/a/build/tmp/sysroots/x86_64-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-clang CFLAGS="--sysroot=/a/build/tmp/sysroots/raspberrypi2" LDFLAGS="-lgcc_s"

- Compile

scan-build --use-analyzer /a/build/tmp/sysroots/x86_64-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-clang --use-cc /a/build/tmp/sysroots/x86_64-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-clang make -j

- Results e.g. https://busybox.net/~kraj/scan-build-2016-03-02-225259-30448-1/

CLANG – ADDITIONAL TOOLS

- musl scan-build runs found some issues which resulted in improvements
  - http://www.openwall.com/lists/musl/2015/09/23/4
  - http://www.openwall.com/lists/musl/2015/09/23/5

CLANG – ADDITIONAL TOOLS

- Clang-check - A syntax checker
  - Selective runs with diagnostics for subset of files
  - Helps integrate with IDEs
  - Use it in fix-it mode

CLANG – ADDITIONAL TOOLS

- clang-format
  - Reformat C++ source files
  - Useful for IDE integration
  - Commit policy
- clang-tidy
  - Lint tool

CLANG COMPILER RUNTIME – USING LIBC++

- libc++ is C++ runtime implementation
  - STL - libc++
  - ABI - libc++abi
    - EH support
      - libunwind
      - llvm-libunwind
- Control with -stdlib option

```
kraj@haswell ~ % clang++ -std=c++11 -stdlib=libc++ -lc++abi ~/hello.cpp
kraj@haswell ~ % ./a.out
1: Hello dude!
2: Hello dude!
3: Hello dude!
kraj@haswell ~ % readelf -d ./a.out

Dynamic section at offset 0x1c18 contains 28 entries:
  Tag        Type                         Name/Value
 0x0000000000000001 (NEEDED)             Shared library: [libc++abi.so.1]
 0x0000000000000001 (NEEDED)             Shared library: [libc++.so.1]
 0x0000000000000001 (NEEDED)             Shared library: [libm.so.6]
 0x0000000000000001 (NEEDED)             Shared library: [libgcc_s.so.1]
 0x0000000000000001 (NEEDED)             Shared library: [libc.so.6]
```

CLANG COMPILER RUNTIME (COMPILER-RT)

- Compiler-RT provides
  - compiler built-ins
  - Full support for libgcc interfaces
- Sanitizer runtimes
  - Support libraries sanitizer instrumented code
- Profile
  - Coverage collection

CLANG COMPILER RUNTIME (SANITIZERS)

- AddressSanitizer -fsanitize=address
  - memory error detection e.g. out of bound accesses
    - Compiler instrumentation and runtime code
- ThreadSanitizer (64bit arches only) -fsanitize=thread
  - Detect Data Races
- MemorySanitizer -fsanitize=memory
  - Detects uninitialized reads
- LeakSanitizer -fsanitize=address (only x86_64)
  - Run-time memory leak detector (WIP x86_64)
- DataFlowSanitizer - Provides Data flow analysis

CLANG COMPILER RUNTIME (LIBUNWIND)

- implements system unwinder
  - High level APIs
    - implement _Unwind_* functions needed by libcxxabi
  - low level APIs
    - unw_* functions
      - HP libunwind compatible APIs

CLANG COMPILER RUNTIME

- Use libunwind & libc++ runtimes
- before

```
kraj01@eos ~ % aarch64-poky-linux-clang++ --sysroot=/opt/poky/2.0+snapshot/sysroots/aarch64-poky-linux hello.cpp
kraj01@eos ~ % aarch64-poky-linux-readelf -d ./a.out

Dynamic section at offset 0xdd8 contains 27 entries:
  Tag        Type                         Name/Value
 0x0000000000000001 (NEEDED)             Shared library: [libstdc++.so.6]
 0x0000000000000001 (NEEDED)             Shared library: [libm.so.6]
 0x0000000000000001 (NEEDED)             Shared library: [libgcc_s.so.1]
 0x0000000000000001 (NEEDED)             Shared library: [libc.so.6]
```

- After

```
kraj01@eos ~ % aarch64-poky-linux-clang++ --sysroot=/opt/poky/2.0+snapshot/sysroots/aarch64-poky-linux -stdlib=libc++ -
nodefaultlibs -lc++ -lc++abi -lc -lunwind  hello.cpp
kraj01@eos ~ % aarch64-poky-linux-readelf -d ./a.out

Dynamic section at offset 0x1dd8 contains 27 entries:
  Tag        Type                         Name/Value
 0x0000000000000001 (NEEDED)             Shared library: [libc++.so.1]
 0x0000000000000001 (NEEDED)             Shared library: [libc++abi.so.1]
 0x0000000000000001 (NEEDED)             Shared library: [libc.so.6]
 0x0000000000000001 (NEEDED)             Shared library: [libunwind.so.1]
```

CHALLENGES

- Fix application packages not _yet_ compilable with clang
  - e.g. https://github.com/kraj/meta-clang/tree/master/recipes-excluded/nonclangable
- Integrate cross SDKs into IDEs e.g. eclipse, develop etc.
- Upstream kernel doesn’t yet compile

THANK YOU