THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
NORTH AMERICA

Embedded Linux Conference

Do You Need GCC to Build Embedded Linux ?

Khem Raj

19-May-2026

Agenda

- Toolchains for Linux systems
- LLVM/Clang
- Extra tools
- Kernel and LLVM
- Userspace
- Building distributions using Clang

Introduction

- Linux Distributions use GCC based toolchains as system compiler
- Provides cross toolchain packages for other architectures
- Some distributions are trying new stuff
  - Mageia
- BSD distributions use LLVM as system compiler
- Embedded System builders
  - Yocto provides an option

LLVM/Clang toolchain

- LLVM project - The LLVM Compiler Infrastructure Project
  - Open-source compiler infrastructure project (started 2000, Chris Lattner at UIUC)
  - Modular, reusable compiler and toolchain technologies
  - Not an acronym — originally "Low Level Virtual Machine," name retained after scope expanded
  - Supports targets: x86, ARM, AArch64, RISC-V, WebAssembly, and more
- Clang
  - LLVM's C/C++/Objective-C frontend (replaces GCC's cc1)
  - Faster compilation, lower memory usage than GCC
  - Expressive, precise diagnostics with fix-it hints
- Key Components
  - clang / clang++ — C/C++ compiler driver
  - lld — fast, drop-in linker
  - lldb — debugger
  - clang-tidy — static analysis and linting
  - clang-format — automated code formatting
  - AddressSanitizer, ThreadSanitizer, UBSan — runtime sanitizers

LLVM/Clang toolchain

- Why Use It?
- Permissive Apache 2.0 license (vs. GPL for GCC)
- Better cross-compilation support out of the box
- Rich API for building custom tools (refactoring, analysis, IDEs)
- Industry adoption: Apple, Google, Microsoft, Android NDK

LLVM/Clang and U-boot

- Building U-Boot with Clang

- Configure for target board

```
make CROSS_COMPILE=aarch64-linux-gnu- ARCH=arm64 qemu_arm64_defconfig
```

Build with Clang

```
make CROSS_COMPILE=aarch64-linux-gnu- ARCH=arm64 \
CC=clang \
LD=ld.lld \
OBJCOPY=llvm-objcopy \
OBJDUMP=llvm-objdump \
-j$(nproc)
```

- Or use LLVM=1 shorthand (supported in recent U-Boot versions, mirrors kernel convention)

```
make LLVM=1 ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- qemu_arm64_defconfig
make LLVM=1 ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu-
```

LLVM/Clang for Kernel

- Linux has supported LLVM/Clang since kernel 4.4 (clangbuiltlinux project)
- Enables LTO and CFI
- LLD linker is quick with incremental builds, better cross-compiler ergonomics
- Minimal build commands

```
make LLVM=1
```

- Android Kernel mandates clang since 5.10

LLVM/Clang Kernel Cross-compilation

- Cross-Compilation
  - ARM64 target, fully LLVM (no GCC cross-tools needed)
  - make LLVM=1 ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu-
- Use LLVM integrated assembler — skip binutils entirely
  - make LLVM=1 LLVM_IAS=1 ARCH=arm64
  - LLVM_IAS=1 uses Clang's built-in assembler instead of GNU as

LLVM/Clang Kernel Advanced options

Sanitizers for Kernel Debugging

```
CONFIG_KASAN=y        # Kernel Address Sanitizer
CONFIG_UBSAN=y        # Undefined Behavior Sanitizer
CONFIG_KCSAN=y        # Kernel Concurrency Sanitizer (data races)
```

- KASAN + Clang supports hardware tag-based mode on ARMv8.5-A (MTE)

Version Requirements

- Clang ≥ 11 for basic x86/arm64 kernel build
- Clang ≥ 13 for CFI support
- Clang ≥ 14 for LLVM_IAS=1 on all architectures
- Check: scripts/min-tool-version.sh clang

LLVM/Clang Explicit use for Userspace

- System / Core Libraries
  - Glibc - Experimental Clang support, not yet default, active effort
  - Musl – Clang supported and regularly tested
  - Compiler-rt – Clang provided compiler builtins, sanitizers runtime
  - Libc++/libc++abi – LLVM provided C++ stdlib
- Compilers/language runtimes
  - Mesa – Clang used
  - LuaJIT – LLVM IR
  - V8 – Needs clang
  - SpiderMonkey – Uses Msan/Asan
  - WASI – Compiles to webassembly (needs clang)

LLVM/Clang Explicit use for Userspace

- Observability and Trace
  - BPF compiler collection (bcc) uses libclang for compiling BPF C programs
  - Bpftrace – Uses libclang + llvm to JIT tracing scripts
- BPF programs require Clang — GCC cannot target BPF backend

  clang -O2 -target bpf -c prog.bpf.c -o prog.bpf.o

Notable distros using Clang/LLVM

- Android – AOSP: Complete userspace
- Fuchsia: Complete userspace
- ChromeOS: Complete userspace
- OpenMandriva: Uses clang as default compiler
- Chimera Linux: musl-based, clang-only (no GCC)
- Yocto Project: Supports Clang as alternative compiler
    - Some distros e.g. YoeDistro build systems with clang

Using Clang to build Embedded Linux with Yocto

- Usage Modes

Full Distro — Clang as Default Compiler

conf/local.conf or distro conf

```
TOOLCHAIN = "clang"
```

- Replaces GCC for all recipes in the build
- meta-clang patches recipes that fail with Clang (ongoing effort)
- Some recipes still force GCC via TOOLCHAIN:pn-<recipe> = "gcc"

Using Clang to build Embedded Linux with Yocto

Per-Recipe Clang Override

Force a specific recipe to use Clang
TOOLCHAIN:pn-myapp = "clang"

Force a specific recipe to stay on GCC
TOOLCHAIN:pn-somelib = "gcc"

Using Clang to build Embedded Linux with Yocto

RUNTIME = "llvm" — Full LLVM Runtime Stack

In conf/local.conf
TOOLCHAIN = "clang"
RUNTIME = "llvm"

- Replaces libgcc → compiler-rt
- Replaces libstdc++ → libc++
- Replaces libunwind (GCC) → llvm-libunwind

Using Clang to build Embedded Linux with Yocto

Known Limitations

- ~5–10% of OE recipes still fail with Clang — mostly due to GNU-specific inline asm or GCC extensions
- FORTRAN support via Clang (flang) is not enabled yet
- Some kernel module recipes (*.ko) revert to GCC even when TOOLCHAIN = "clang"
- lld as linker can fail on linker scripts written with GNU ld-only syntax
- Build times longer on first run — LLVM itself is a large build (~20–40 min)

Using Clang to build Embedded Linux with Yocto

Practical Recommendation for Embedded Projects

- Use hybrid setup
  TOOLCHAIN = "clang"
  RUNTIME = "llvm"

- Known-problematic recipes pinned to GCC
  TOOLCHAIN:pn-gcc-runtime = "gcc"
  TOOLCHAIN:pn-glibc = "gcc"
  TOOLCHAIN:pn-linux-yocto = "gcc“ unless you want LLVM kernel build

Summary

- Clang/llvm C/C++ compiler can build bootloaders and kernel
- Clang can build most of userspace except some packages e.g. glibc
- Certain packages require clang to be used explicitly e.g. chromium
- LLVM runtime (compiler-rt, libcxx, crt objects, llvm-runtime, libopmp) provides working alternatives
- Sometimes packages require fixes to work with libc++
- Upstream package CI should use clang as well

Thank you

THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
NORTH AMERICA

Embedded Linux Conference