Google

Modern Embedded Development with LLVM

LLVM Dev Mtg '24

Petr Hosek (phosek@google.com)

October 2024

Last year we presented early results...

[LLVM Toolchain for Embedded Systems] Prabhu Karthikeyan Rajasekaran

...and we made a lot of progress since then

Made by Google

What has been our experience?

Most baremetal projects use toolchains provided by their silicon vendors which are **frequently outdated** and **seldomly updated**.

Silicon vendors use their toolchains for their platform libraries and can (inadvertently) force their toolchains onto their customers through API and ABI design.

Every toolchain comes with its own C and C++ library and aligning API and ABI between different C and C++ libraries can be challenging, and in some cases impossible when custom proprietary extensions are involved.

Arm, RISC-V, etc.

toolchain per target × toolchain per host

Linux, macOS, Windows, etc.

Google

Update toolchain every ? years

vs

Live at HEAD

Google

We “live at HEAD”

We aim to provide an **open-source cross-compiling Clang/LLVM toolchain** for baremetal without any legacy components and roll it on a continuous basis.

- Clang, LLD and LLVM tools as an alternative to GCC and GNU binutils.
- LLVM libc, libc++ and compiler-rt instead of newlib/picolibc, libstdc++ and libgcc.

We aim to provide the same experience on every supported target and host platform.

“Live at HEAD”

relies on automation & testing

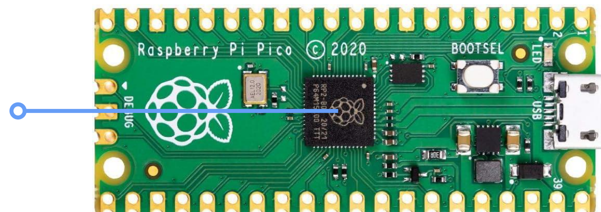
Google

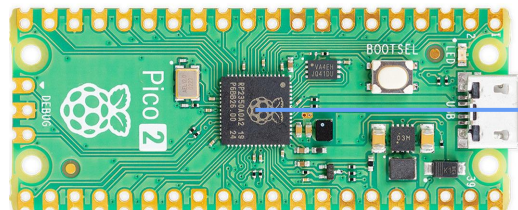
We need a reference platform to run tests on

RP2040

Arm Cortex-M0+

264kB SRAM, 2MB flash





RP2350

Arm Cortex-M33 & RISC-V RV32

520kB SRAM, 4MB flash

We worked closely with the Raspberry Pi Foundation and the Pigweed team, making Pico SDK the first baremetal project that can be built using the open-source Clang/LLVM toolchain with compiler-rt, LLVM libc and libc++.

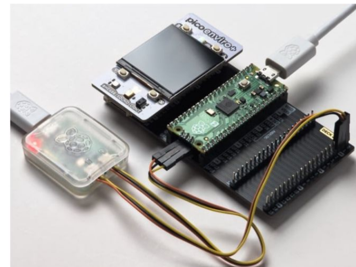
Raspberry Pi

For home For industry Hardware Software Documentation News Forums Foundation Contact

Google Pigweed comes to our new RP2350

8th Aug 2024 Chris Boross

We love Google Pigweed! Pigweed is an open source project launched by Google in 2020. We love it because it helps programmers and teams of developers build great software for embedded devices that use microcontrollers like our new RP2350 and its predecessor, RP2040. We are also partial to funny product names around here; we are thrilled that our Pico W has gone down in the community's vernacular as "pie cow".



Google's demo built on the new Pigweed SDK uses Pimoroni's Enviro+ Pack add-on to help showcase all the neat stuff Pigweed does for developers

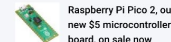
We've been working with the Pigweed team for almost a year now. Last month they upstreamed Bazel support into our Pico SDK, and will continue to maintain it going forward.

Bazel is an important part of the Pigweed project, and the team believes it's going to be the future of embedded software development, making it easier for large, professional embedded development teams to build prototypes and products on top of RP2350. Head over to Bazel's launch blog post to learn more about the benefits of Bazel for embedded.

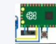
The Pigweed team has built a great demo for you to try on your Raspberry Pi Pico 1 or Pico 2. This demo shows off lots of complex stuff handled and enabled by Pigweed, including:

- Hermetic building, flashing, and testing through Bazel
- Fully open-source Clang/LLVM toolchain for embedded that includes a compiler, linker, and C/C++ libraries with modern performance, features, and standards compliance
- Structuring your codebase around sensible, hardware-agnostic C++ through Pigweed's extensive collection of libraries
- Communicating with your Pico over RPC
- Viewing Pico logs and sending commands to the Pico over an interactive and customizable REPL
- Authoring in Visual Studio Code with C++, Starlark code intelligence, and Bazel command integration
- Cross-platform builds and toolchains, development on macOS or Linux (Windows support is on its way)
- Device simulation on your host computer
- Continuous building and testing with GitHub Actions

RELATED POSTS



Raspberry Pi Pico 2, our new $5 microcontroller board, on sale now

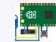


Real-time ML audio noise suppression on Raspberry Pi Pico 2

NEXT POST

Raspberry Pi Pico 2, our new $5 microcontroller board, on sale now

PREVIOUS POST



Real-time ML audio noise suppression on Raspberry Pi Pico 2

Share this post

```
$ git clone https://github.com/raspberrypi/pico-sdk
$ cmake -S pico-sdk -B build -G Ninja \
  -D PICO_COMPILER=pico_arm_cortex_m33_clang \
  -D PICO_PLATFORM=rp2350-arm-s \
  -D PICO_TOOLCHAIN_PATH=?
$ ninja -C build
```


Google

```cmake
# Pico.cmake
set(CMAKE_BUILD_TYPE Release CACHE STRING "")

set(LLVM_TARGETS_TO_BUILD ARM;RISCV CACHE STRING "")
set(LLVM_ENABLE_PROJECTS clang;lld;llvm CACHE STRING "")
set(LLVM_ENABLE_RUNTIMES compiler-rt;libcxx;libc CACHE STRING "")

set(CLANG_DEFAULT_CXX_STDLIB libc++ CACHE STRING "")
set(CLANG_DEFAULT_LINKER lld CACHE STRING "")
set(CLANG_DEFAULT_RTLIB compiler-rt CACHE STRING "")

set(LLVM_INSTALL_TOOLCHAIN_ONLY ON CACHE BOOL "")
set(LLVM_TOOLCHAIN_TOOLS
llvm-ar;llvm-cov;llvm-objcopy;llvm-objdump;llvm-profdata;llvm-ranlib;llvm-readelf;llvm-readob
j;llvm-size;llvm-strings;llvm-strip;llvm-symbolizer CACHE STRING "")
set(LLVM_DISTRIBUTION_COMPONENTS
builtins;clang;clang-resource-headers;lld;runtimes;${LLVM_TOOLCHAIN_TOOLS} CACHE STRING "")
```


Google

```cmake
# Pico.cmake
set(LLVM_BUILTIN_TARGETS armv8m.main-none-eabi;riscv32-unknown-elf CACHE STRING "")
foreach(target ${LLVM_BUILTIN_TARGETS})
  set(BUILTINS_${target}_CMAKE_SYSTEM_NAME Generic CACHE STRING "")
  set(BUILTINS_${target}_CMAKE_BUILD_TYPE MinSizeRel CACHE STRING "")
  set(BUILTINS_${target}_COMPILER_RT_BAREMETAL_BUILD ON CACHE BOOL "")
endforeach()
set(BUILTINS_armv8m.main-none-eabi_CMAKE_SYSTEM_PROCESSOR arm CACHE STRING "")
set(BUILTINS_riscv32-unknown-elf_CMAKE_SYSTEM_PROCESSOR RISCV CACHE STRING "")
foreach(lang C;CXX;ASM)
  set(BUILTINS_armv8m.main-none-eabi_CMAKE_${lang}_FLAGS "-march=armv8m.main+fp+dsp
-mcpu=cortex-m33 -mfloat-abi=softfp" CACHE STRING "")
  set(BUILTINS_riscv32-unknown-elf_CMAKE_${lang}_FLAGS
"-march=rv32imac_zicsr_zifencei_zba_zbb_zbs_zbkb -mabi=ilp32" CACHE STRING "")
endforeach()
```


Google

```cmake
# Pico.cmake
set(LLVM_RUNTIME_TARGETS armv8m.main-none-eabi;riscv32-unknown-elf CACHE STRING "")
foreach(target ${LLVM_RUNTIME_TARGETS})
  set(RUNTIMES_${target}_CMAKE_SYSTEM_NAME Generic CACHE STRING "")
  set(RUNTIMES_${target}_CMAKE_BUILD_TYPE MinSizeRel CACHE STRING "")
  set(RUNTIMES_${target}_CMAKE_TRY_COMPILE_TARGET_TYPE STATIC_LIBRARY CACHE STRING "")
  set(RUNTIMES_${target}_LLVM_ENABLE_RUNTIMES libc;libcxx CACHE STRING "")
  set(RUNTIMES_${target}_LLVM_ENABLE_ASSERTIONS OFF CACHE BOOL "")
endforeach()
set(RUNTIMES_armv8m.main-none-eabi_CMAKE_SYSTEM_PROCESSOR arm CACHE STRING "")
set(RUNTIMES_riscv32-unknown-elf_CMAKE_SYSTEM_PROCESSOR RISCV CACHE STRING "")
foreach(lang C;CXX;ASM)
  set(RUNTIMES_armv8m.main-none-eabi_CMAKE_${lang}_FLAGS "-march=armv8m.main+fp+dsp
-mcpu=cortex-m33 -mfloat-abi=softfp" CACHE STRING "")
  set(RUNTIMES_riscv32-unknown-elf_CMAKE_${lang}_FLAGS
"-march=rv32imac_zicsr_zifencei_zba_zbb_zbs_zbkb -mabi=ilp32" CACHE STRING "")
endforeach()
```


Google

```
$ git clone https://github.com/llvm/llvm-project
$ cmake -S llvm-project/llvm -B build -G Ninja -C Pico.cmake
$ ninja -C build distribution
$ ninja -C build install-distribution-stripped
```

[CMake cache file for building Pico SDK toolchain](#) #113267

Google

compiler-rt

compiler-rt builtins is mostly a replacement for libgcc, and we haven't encountered any issues.

LLVM isn't aware of the compiler runtime is being used and cannot take advantage of routines that are compiler-rt specific.

LLVM libc

libc is becoming ready for broader adoption, and offers permissively licensed C library implementation that can be scaled down to baremetal projects.

We had to implement a number of missing functionality including:

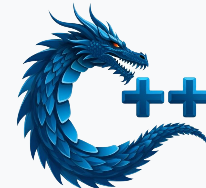
- high order math functions;
- malloc implementation suitable for embedded;
- basic I/O support.

Some parts, most notably ***startup***, ***semihosting I/O*** *and* ***threading***, are still missing.

```
struct __llvm_libc_stdio_cookie;

extern "C" struct __llvm_libc_stdio_cookie __llvm_libc_stdin_cookie;
extern "C" struct __llvm_libc_stdio_cookie __llvm_libc_stdout_cookie;
extern "C" struct __llvm_libc_stdio_cookie __llvm_libc_stderr_cookie;

extern "C" ssize_t __llvm_libc_stdio_read(void *cookie, char *buf, size_t size);
extern "C" ssize_t __llvm_libc_stdio_write(void *cookie, const char *buf,
                                           size_t size);

extern "C" [[noreturn]] void __llvm_libc_exit(int status);

extern "C" int *__llvm_libc_errno();
```


Google

LLVM libc++



libc++ has a number of options to disable unnecessary functionality, but we need more configuration points to better support embedded environment.

There are still places where libc++ relies on functionality that is undesirable in baremetal projects: **dynamic memory allocation**, **TLS** or **float support**.

We had to introduce support for building libc++ against libc.

Support use of libc++ with LLVM libc in embedded development #84879

```cmake
# Pico.cmake
foreach(target ${LLVM_RUNTIME_TARGETS})
  …
  set(RUNTIMES_${target}_LLVM_LIBC_FULL_BUILD ON CACHE BOOL "")
  set(RUNTIMES_${target}_LIBC_ENABLE_USE_BY_CLANG ON CACHE BOOL "")
  set(RUNTIMES_${target}_LIBCXX_CXX_ABI none CACHE STRING "")
  set(RUNTIMES_${target}_LIBCXX_LIBC llvm-libc CACHE STRING "")
  set(RUNTIMES_${target}_LIBCXX_ENABLE_SHARED OFF CACHE BOOL "")
  set(RUNTIMES_${target}_LIBCXX_ENABLE_FILESYSTEM OFF CACHE BOOL "")
  set(RUNTIMES_${target}_LIBCXX_ENABLE_RANDOM_DEVICE OFF CACHE BOOL "")
  set(RUNTIMES_${target}_LIBCXX_ENABLE_LOCALIZATION OFF CACHE BOOL "")
  set(RUNTIMES_${target}_LIBCXX_ENABLE_UNICODE OFF CACHE BOOL "")
  set(RUNTIMES_${target}_LIBCXX_ENABLE_WIDE_CHARACTERS OFF CACHE BOOL "")
  set(RUNTIMES_${target}_LIBCXX_ENABLE_EXCEPTIONS OFF CACHE BOOL "")
  set(RUNTIMES_${target}_LIBCXX_ENABLE_RTTI OFF CACHE BOOL "")
  set(RUNTIMES_${target}_LIBCXX_ENABLE_THREADS OFF CACHE BOOL "")
  set(RUNTIMES_${target}_LIBCXX_ENABLE_MONOTONIC_CLOCK OFF CACHE BOOL "")
  set(RUNTIMES_${target}_LIBCXX_USE_COMPILER_RT ON CACHE BOOL "")
endforeach()
```


Google

Building a toolchain is becoming easier

It used to be harder to build a toolchain (and impossible from within llvm-project.git), but there is still a lot of room for improvement.

The runtimes build has a basic multilib support, but every variant requires a separate CMake subbuild and there's no support for the new `multilib.yaml` configuration.

Ideally we wouldn't be distributing binary libraries and **build runtimes from source on demand**. We have a minimal prototype in ***Pigweed***, but it's build system specific.

Performance still has gaps

Most baremetal projects are sensitive to binary size and memory usage, making everything fit can require a lot of cargo culting.

GCC -Os better balances size and performance, Clang -Oz produces smaller binaries but disables useful optimizations which can pessimize performance.

Stack allocation in LLVM has a number of issues which can lead to inefficient stack usage that is frequently causing issues on baremetal.

Trivial memset optimization not applied to loops under -Oz #50308
Improve stack slot reuse #109204

Google

```
CFLAGS = -Oz -finline-max-stacksize=256 -fomit-frame-pointer -fshort-enums
-ffunction-sections -fdata-sections -fseparate-named-sections -ffixed-point
# Enable ML inliner
CFLAGS += -mllvm -enable-ml-inliner=release
# Enable GV sink/hoist
CFLAGS += -mllvm -enable-gvn-sink=1 -mllvm -enable-gvn-hoist=1

CXXFLAGS = -fno-c++-static-destructors -fno-exceptions -fno-rtti
# Remove the unused RTTI component from vtables
CXXFLAGS += -Xclang -fexperimental-omit-vtable-rtti

LDFLAGS = -Wl,--gc-sections -Wl,--icf=all -Wl,-O2
```


Google

Debugging can be challenging

LLVM and LLDB don't support all DWARF CFI constructs which are often needed to support unwinding through hand-written Assembly such as interrupt handlers.

LLDB support for baremetal has many gaps, for example the 32-bit RISC-V support is largely missing.

Google

There's a lot of value in LLVM embedded developers could benefit from, but embedded systems have constraints

Google

Support for heterogeneous memory

Embedded systems use attributes and linker scripts to place symbols in specific memory regions, we need to figure out how to make these compatible with LTO/PGO.

- LTO can reduce binary size by >20%, but doesn't support linker scripts.
- PGO can improve performance by >20%, but doesn't support memory placement.

We would also like to explore post-link optimization for baremetal.

Bringing link-time optimization to the embedded world: (Thin)LTO with Linker Scripts Tobias Edler von Koch
Link-Time Attributes for LTO: Incorporating Linker Knowledge Into the LTO Recompile Todd Snider
Higher-Level Linker Scripts for Embedded Systems Daniel Thornburgh

Sanitizers on memory constrained devices

Sanitizers are great tools for finding runtime bugs, but using them in embedded systems is challenging.

- UBSan has several runtime implementations, none of them is a great fit.
- ASan instrumentation overhead is too high, we need ways to reduce it.
- Other sanitizers may be infeasible due to high overhead.

Sanitizer runtimes were developed for POSIX (and later Windows), making them difficult to (re)use for baremetal platforms.

Address Sanitizer on Baremetal Targets Oliver Stannard

Google

Source-based code coverage for baremetal

Code coverage is an important testing metric and LLVM has great tooling for coverage, but the profile runtime has structural issues that complicate porting.

There are opportunities for reducing the instrumentation overhead:

- Single-byte counters
- Conditional counter updates
- Saturating 32-bit counters

Google

LLVM is a great fit for embedded

Cross compilation	• LLVM is a cross-compiler so a single toolchain can support a variety of targets • We are making it easier to build a complete toolchain in LLVM build
C/C++ library	• LLVM offers modular permissively licensed C and C++ standard library • We are making LLVM libc and libc++ usable on baremetal
Continuous testing	• There are baremetal projects using Clang/LLVM toolchain that “Live at HEAD” • Pico SDK now supports open-source Clang/LLVM toolchain
Tools & features	• We are trying to enable the use of profiling and LTO on baremetal • There is a lot of interest in sanitizers and coverage for embedded

How to get involved?

There's an active and growing embedded community within LLVM.

- **LLVM Embedded Toolchains Working Group** Thursday 9am PST every 4 weeks
- **LLVM Embedded Toolchains Workshop** as part of LLVM Developers' Meeting

embedded label in GitHub as a way to find issues related to baremetal uses.