How to Cross Compile Compiler-rt Builtins For Arm

Introduction

This document contains information about building and testing the builtins part of compiler-rt for an Arm target, from an x86_64 Linux machine.

While this document concentrates on Arm and Linux the general principles should apply to other targets supported by compiler-rt. Further contributions for other targets are welcome.

The instructions in this document depend on libraries and programs external to LLVM, there are many ways to install and configure these dependencies so you may need to adapt the instructions here to fit your own local situation.

Prerequisites

In this use case we'll be using CMake on a Debian-based Linux system, cross-compiling from an x86_64 host to a hard-float Armv7-A target. We'll be using as many of the LLVM tools as we can, but it is possible to use GNU equivalents.

- A build of LLVM/clang for the llvm-tools and llvm-config
- The qemu-arm user mode emulator
- An arm-linux-gnueabihf sysroot

See https://compiler-rt.llvm.org/ for more information about the dependencies on clang and LLVM.

qemu-arm should be available as a package for your Linux distribution.

The most complicated of the prequisites to satisfy is the arm-linux-gnueabihf sysroot. The *How To Cross-Compile Clang/LLVM using Clang/LLVM* has information about how to use the Linux distributions multiarch support to fulfill the dependencies for building LLVM. Alternatively, as building and testing just the compiler-rt builtins requires fewer dependencies than LLVM, it is possible to use the Linaro arm-linux-gnueabihf gcc installation as our sysroot.

Building compiler-rt builtins for Arm

We will be doing a standalone build of compiler-rt using the following cmake options.

- path/to/llvm/projects/compiler-rt
- -DCOMPILER_RT_BUILD_BUILTINS=ON
- -DCOMPILER_RT_BUILD_SANITIZERS=OFF
- -DCOMPILER_RT_BUILD_XRAY=OFF
- -DCOMPILER_RT_BUILD_LIBFUZZER=OFF
- -DCOMPILER_RT_BUILD_PROFILE=OFF
- -DCMAKE_C_COMPILER=/path/to/clang
- -DCMAKE_AR=/path/to/llvm-ar
- -DCMAKE_NM=/path/to/llvm-nm
- -DCMAKE_RANLIB=/path/to/llvm-ranlib
- -DCMAKE_EXE_LINKER_FLAGS="-fuse-ld=lld"
- -DCMAKE_C_COMPILER_TARGET="arm-linux-gnueabihf"
- -DCOMPILER_RT_DEFAULT_TARGET_ONLY=ON
- -DLLVM_CONFIG_PATH=/path/to/llvm-config
- -DCMAKE_C_FLAGS="build-c-flags"

The build-c-flags need to be sufficient to pass the C-make compiler check and to compile compiler-rt. When using a GCC 7 Linaro arm-linux-gnueabihf installation the following flags are needed:

- --target=arm-linux-gnueabihf
- --march=armv7a
- --gcc-toolchain=/path/to/dir/toolchain
- --sysroot=/path/to/toolchain/arm-linux-gnueabihf/libc

Depending on how your sysroot is laid out, you may not need --gcc-toolchain. For example if you have added armhf as an architecture using your Linux distributions multiarch support then you should be able to use --sysroot=/.

Once cmake has completed the builtins can be built with ninja builtins

Testing compiler-rt builtins using qemu-arm

To test the builtins library we need to add a few more cmake flags to enable testing and set up the compiler and flags for test case. We must also tell cmake that we wish to run the tests on qemu-arm.

- -DCOMPILER_RT_EMULATOR="qemu-arm -L /path/to/armhf/sysroot
- -DCOMPILER_RT_INCLUDE_TESTS=ON
- -DCOMPILER_RT_TEST_COMPILER="/path/to/clang"
- -DCOMPILER_RT_TEST_COMPILER_CFLAGS="test-c-flags"

The /path/to/armhf/sysroot should be the same as the one passed to --sysroot in the “build-c-flags”.

The “test-c-flags” can be the same as the “build-c-flags”, with the addition of "-fuse-ld=lld if you wish to use lld to link the tests.

Once cmake has completed the tests can be built and run using ninja check-builtins

Modifications for other Targets

Arm Soft-Float Target

The instructions for the Arm hard-float target can be used for the soft-float target by substituting soft-float equivalents for the sysroot and target. The target to use is:

- -DCMAKE_C_COMPILER_TARGET=arm-linux-gnueabi

Depending on whether you want to use floating point instructions or not you may need extra c-flags such as -mfloat-abi=softfp for use of floating-point instructions, and -mfloat-abi=soft -mfpu=none for software floating-point emulation.

AArch64 Target

The instructions for Arm can be used for AArch64 by substituting AArch64 equivalents for the sysroot, emulator and target.

- -DCMAKE_C_COMPILER_TARGET=aarch64-linux-gnu
- -DCOMPILER_RT_EMULATOR="qemu-aarch64 -L /path/to/aarch64/sysroot

The CMAKE_C_FLAGS and COMPILER_RT_TEST_COMPILER_CFLAGS may also need: "--sysroot=/path/to/aarch64/sysroot --gcc-toolchain=/path/to/gcc-toolchain"

Armv6-m, Armv7-m and Armv7E-M targets

If you wish to build, but not test compiler-rt for Armv6-M, Armv7-M or Armv7E-M then the easiest way is to use the BaremetalARM.cmake recipe in clang/cmake/caches.

You will need a bare metal sysroot such as that provided by the GNU ARM Embedded toolchain.

The libraries can be built with the cmake options:

- -DBAREMETAL_ARMV6M_SYSROOT=/path/to/bare/metal/sysroot
- -DBAREMETAL_ARMV7M_SYSROOT=/path/to/bare/metal/sysroot
- -DBAREMETAL_ARMV7EM_SYSROOT=/path/to/bare/metal/sysroot
- -C /path/to/llvm/source/tools/clang/cmake/caches/BaremetalARM.cmake

Note that for the recipe to work the compiler-rt source must be checked out into the directory llvm/runtimes and not llvm/projects.

To build and test the libraries using a similar method to Armv7-A is possible but more difficult. The main problems are:

- There isn't a qemu-arm user-mode emulator for bare-metal systems. The qemu-system-arm can be used but this is significantly more difficult to setup.
- The target to compile compiler-rt have the suffix -none-eabi. This uses the BareMetal driver in clang and by default won't find the libraries needed to pass the cmake compiler check.

As the Armv6-M, Armv7-M and Armv7E-M builds of compiler-rt only use instructions that are supported on Armv7-A we can still get most of the value of running the tests using the same qemu-arm that we used for Armv7-A by building and running the test cases for Armv7-A but using the builtins compiled for Armv6-M, Armv7-M or Armv7E-M. This will not catch instructions that are supported on Armv7-A but not Armv6-M, Armv7-M and Armv7E-M.

To get the cmake compile test to pass the libraries needed to successfully link the test application will need to be manually added to CMAKE_CFLAGS. Alternatively if you are using version 3.6 or above of cmake you can use CMAKE_TRY_COMPILE_TARGET=STATIC_LIBRARY to skip the link step.

- -DCMAKE_TRY_COMPILE_TARGET_TYPE=STATIC_LIBRARY
- -DCOMPILER_RT_OS_DIR="baremetal"
- -DCOMPILER_RT_BUILD_BUILTINS=ON
- -DCOMPILER_RT_BUILD_SANITIZERS=OFF
- -DCOMPILER_RT_BUILD_XRAY=OFF
- -DCOMPILER_RT_BUILD_LIBFUZZER=OFF

- -DCOMPILER_RT_BUILD_PROFILE=OFF
- -DCMAKE_C_COMPILER=${host_install_dir}/bin/clang
- -DCMAKE_C_COMPILER_TARGET="your *-none-eabi target"
- -DCMAKE_AR=/path/to/llvm-ar
- -DCMAKE_NM=/path/to/llvm-nm
- -DCMAKE_RANLIB=/path/to/llvm-ranlib
- -DCOMPILER_RT_BAREMETAL_BUILD=ON
- -DCOMPILER_RT_DEFAULT_TARGET_ONLY=ON
- -DLLVM_CONFIG_PATH=/path/to/llvm-config
- -DCMAKE_C_FLAGS="build-c-flags"
- -DCMAKE_ASM_FLAGS="${arm_cflags}"
- -DCOMPILER_RT_EMULATOR="qemu-arm -L /path/to/armv7-A/sysroot"
- -DCOMPILER_RT_INCLUDE_TESTS=ON
- -DCOMPILER_RT_TEST_COMPILER="/path/to/clang"
- -DCOMPILER_RT_TEST_COMPILER_CFLAGS="test-c-flags"

The Armv6-M builtins will use the soft-float ABI. When compiling the tests for Armv7-A we must include "-mthumb -mfloat-abi=soft -mfpu=none" in the test-c-flags. We must use an Armv7-A soft-float abi sysroot for qemu-arm.

Unfortunately at time of writing the Armv7-M and Armv7E-M builds of compiler-rt will always include assembler files including floating point instructions. This means that building for a cpu without a floating point unit requires something like removing the arm_Thumb1_VFPv2_SOURCES from the arm_Thumb1_SOURCES in builtins/CMakeLists.txt. The float-abi of the compiler-rt library must be matched by the float abi of the Armv7-A sysroot used by qemu-arm.

Depending on the linker used for the test cases you may encounter BuildAttribute mismatches between the M-profile objects from compiler-rt and the A-profile objects from the test. The lld linker does not check the BuildAttributes so it can be used to link the tests by adding -fuse-ld=lld to the COMPILER_RT_TEST_COMPILER_CFLAGS.