



LLVM for Embedded

LLVM Dev Meeting 2023
Prabhu Karthikeyan Rajasekaran
Google LLC

Introduction

- Part of an LLVM toolchain team focused on low level operating systems.
- We follow “upstream first” principle for LLVM development.
- We build & ship our LLVM toolchains continuously from upstream.

Building an Operating System from Scratch with LLVM
[\[video\]](#)[\[slides\]](#)

– Petr Hosek, Keynote LLVM Dev meeting 2021.



Motivation

Motivation

- For embedded projects,
 - Binary size
 - One-off toolchains are harder to improve 
 - Take advantage of the rich tooling available
 - Unlock optimization opportunities (e.g. LTO, ML Inliner)
- For the LLVM community,
 - Number of projects in embedded domain is continuing to grow
 - Interesting challenges and problems to be worked on which can have far reaching impact beyond the embedded context



Pigweed (pigweed.dev)

- Open source collection of embedded-targeted libraries
- One-stop-shop to get started on embedded projects
- Pigweed ships our LLVM toolchain out of the box
- Vibrant open source community
 - [Discord](#)
- Several projects across Google use Pigweed



Pixel Buds Pro

- Complex Architecture
 - Multiple cores – ARM Cortex M0, M4
 - Charging Case, Bluetooth, Sensors and Bootloaders Applications
 - Different RTOSes – FreeRTOS, EmbOS, ThreadX
- Binary size constraints: Approximately 550 KiB
- Hundreds of developers and millions of users
- **The recent feature drop for Pixel Buds Pro was built with our toolchain!**



Part 1 – Porting

Approach

- Incremental
 - One target at a time
 - One toolchain component at a time
- Maintain backward compatibility with the GNU toolchain.

Compiling Android userspace and Linux Kernel with LLVM

[\[video\]](#)[\[slides\]](#)

– Nick Desaulniers, Greg Hackmann, and Stephen Hines, LLVM Dev meeting 2017.

Other past efforts: google3, Fuchsia

Compiling and Linking

Compiler: GCC to Clang

- Better diagnostics

```
pw::sync::Mutex& decoder_mutex_;  
pw::hdlc::Decoder decoder_  
    PW_GUARDED_BY(decoer_mutex_);  
                ^^ TYPO - missing 'd'
```

- Clang warnings
 - Unused variables/members, Constexpr violations, Integer type mismatches, Shadowing variables, Potential unaligned access and more
 - Addressing these warnings generally resulted in improved code quality

Linker: BFD LD to LLD

- Linker scripts offer control over memory layout
- No automatic way to pack objects across disjoint memory regions in LLD
- Symbol resolution logic varies between linkers
- Linker script syntax accepted has minor differences
- Linkmap file formats differ

```
MEMORY {
    REGION_1 (rx): ORIGIN = REGION_1_START, LENGTH =
        (REGION_2_START - REGION_1_START)

    REGION_2 (rx): ORIGIN = REGION_2_START, LENGTH =
        (REGION_2_END - REGION_2_START)
}

SECTIONS {
    REGION_1_AREA : {
        *obj1*.o:* (.text*);
        *obj2*.o:* (.text*);
        *obj3*.o:* (.text*);
    } > REGION_1
    ...
    REGION_2_AREA : {
        *(.text.BootEntry);
        *(.text);
        *(.text.*);
    } > REGION_2
}
```

Runtime Crashes

Different defaults

- “-fshort-enum”
 - ARM GCC sets “-fshort-enum” by default
 - Layout mismatch between GCC and Clang built modules

Latent bugs

- Underspecified Inline Assembly

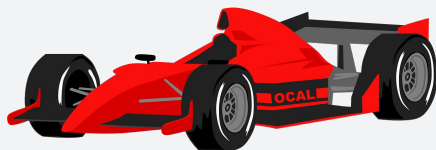
```
int main(void) {
    uint64_t in = 1;
    uint64_t out;
    __asm__ volatile (
        "movs %[in], %[out]" /* out = in */
        "adds %[out], %[out], #0x1" /* out = out + 1 */
        "movs %[in], %[out]" /* out = in */
        "adds %[out], %[out], #0x1" /* out = out + 1 */
        : [out] "=r" (out)
        : [in] "r" (in)
        :
    );
    assert(out == 2);
}
```

Missing earlylobber constraint "&"

```
: [out] "=&r" (out)
```

Latent bugs

- Race condition
 - Application code was accessing a ring buffer data structure from across threads **without locking**





Differences in stack sizes

- No comprehensive stack usage analysis available for developers
- Inlining decisions affect stack size
- Debugging:
 - Clang/LLVM, however, has excellent support for understanding stack frame layouts
 - “-Rpass-analysis=stack-frame-layout”

```
remark:  
Function: _ZNZax322SpatialAudioManager11HandleEventEN2w12event_router9ComponentEjPv  
Offset: [SP-4], Type: Spill, Align: 4, Size:  
Offset: SP-81, Type: Spill, Align: 4, Size:  
Offset: [SP-12], Type: Spill, Align: 4, Size:  
Offset: [SP-16], Type: Spill, Align: 4, Size:  
Offset: [SP-201], Type: Spill, Align: 4, Size:  
Offset: [SP-24], Type: Spill, Align: 4, Size: 4  
Offset: SP-552], Type: Variable, Align: 8, Size: 528  
    large_inlined_variable1 @spatial_audio_manager.cc:21  
Offset: [SP-1080], Type: Variable, Align: 8, Size: 528  
    large_inlined_variable2 @spatial_audio_manager.cc:22 [-Rpass-analysis=stack-frame-layout]  
    53 bool SpatialAudioManager::HandleEventcomponent component,  
    ^
```


All Targets

Component	Goal	Status
Compiler	GCC to Clang	DONE 
Linker	BFD to LLD	DONE 
C library	Newlib-nano to LLVM Libc	Experimental

C++ library	GNU stdlib to LLVM libc++	TBD
Runtimes	GNU libgcc to compiler-rt builtins	TBD

Part 2 – Future Directions

Link Time Optimization

- Linker scripts have assumptions that doesn't account for LTO compilation
- Initial experiments suggest LTO can provide >10% size savings possible

LLD – Wish list

- Missing features
 - Automatic object packing support across disjoint memory regions
 - “enable-non-contiguous-memory-regions”
 - Overlays in LLD missing “NOCROSSREFS” support
- Improvements
 - Support JSON format output for linkmaps
 - “-print-gc-sections” – Print group signatures
- Wish list – <https://bugs.fuchsia.dev/u/phosek@google.com/hotlists/LLVM-Embedded-Features>

More involved open-ended problems

- libc++ support for embedded
 - ABI stability, bloat, configurability etc.
- Precise stack size analysis to identify how much stack to allocate
- Generic approaches to support runtimes in size constrained targets
 - How can we achieve profiling on target devices?
 - How can we adapt sanitizers to be employed on target devices?

Summary

- We've shipped a real-world embedded project built with our Clang/LLVM toolchain
- LLVM based toolchains have the potential to become the "de facto" toolchains for embedded projects
- Offering first class support to embedded scenarios can have far reaching positive impact across domains
- Community engagement
 - LLVM Embedded Toolchains Working Group. Meets [monthly](#)

Q&A

Backup slides

Binary Size Optimizations

- Clang
 - `-fomit-frame-pointer`
 - `-fshort-enums`
 - `-Oz`
 - GVN sink/hoist
 - `"-mllvm", "-enable-gvn-sink=1",`
 - `"-mllvm", "-enable-gvn-hoist=1",`
- LLD
 - `icf=all`
 - `-O2`
- ML Inliner from the MLGO project
 - `"-mllvm", "-enable-ml-inliner=release"`