ez-clang

https://echtzeit.dev/ez-clang

C++ REPL for bare metal embedded devices

Stefan Gränitz · May 11, 2022
European LLVM Developers' Meeting
Code runs on connected development board

Cling-based REPL prompt for C++ and meta commands

Docker with QEMU and Arduino Due support v0.0.5: https://hub.docker.com/r/echtzeit/ez-clang

Firmware reference implementations:
https://github.com/echtzeit-dev/ez-clang-arduino
https://github.com/echtzeit-dev/ez-clang-qemu

Current development state of mind: go fast and break things
Agenda

1. Terminology
2. Hardware Dimensions
3. REPL Pipeline
4. Transform: Return Value Extraction
5. Firmware Documentation
6. Outlook
7. Questions
Terminology

- Command Prompt
- Clang Frontend
- JIT Backend
- Prebuilt Libraries
- Tools
- RPC

- e.g. ez/stdio/printf.a
- Bundled firmwares, scripts, etc.
- ORCv2 + JITLink

- Host
- Device
- Serial connection
- ez-clang firmware

- multi-threaded
- single-threaded
Hardware Dimensions

Raspberry Pi 4 vs. Bare Metal Microcontrollers

- **Processor**: 4x 1.5GHz
- **RAM**: 4GB
- **Disk**: Typical MicroSD: 32GB

*Note: Arduino Due and Raspberry Pi 4 are compared graphically.*
**Hardware Dimensions**

**Bare Metal Microcontrollers**

<table>
<thead>
<tr>
<th>Processor</th>
<th>RAM</th>
<th>ROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>50 MHz</td>
<td>8 KB</td>
<td>64 KB</td>
</tr>
<tr>
<td>70 MHz</td>
<td>16 KB</td>
<td>256 KB</td>
</tr>
<tr>
<td>84 MHz</td>
<td>96 KB</td>
<td>512 KB</td>
</tr>
</tbody>
</table>

- ez-clang
- Stellaris lm3s811evb
- MicroPython (min. requirements*)
- Arduino Due

* v1.18 ESP8266 guide: “The minimum requirement for flash size is 1Mbyte. There is also a special build for boards with 512KB, but it is highly limited comparing to the normal build”
Hardware Dimensions
Bare Metal Microcontrollers

Processor
- 50 MHz
- 70 MHz
- 84 MHz

RAM
- 8 KB
- 16 KB
- 96 KB

ROM
- 64 KB
- 256 KB
- 512 KB

- ez-clang
- Stellaris lm3s811evb
- MicroPython (min. requirements)
- Arduino Due
## Competitors?

<table>
<thead>
<tr>
<th></th>
<th>MicroPython</th>
<th>ez-clang</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Language</strong></td>
<td>(Reduced) Python Dialect</td>
<td>Standard C++</td>
</tr>
<tr>
<td><strong>Standard Libraries</strong></td>
<td>Subset of Python Stdlib</td>
<td>Newlib instead of glibc</td>
</tr>
<tr>
<td></td>
<td>Feature-set depends on device capacity</td>
<td>STL adaptations like ETL</td>
</tr>
<tr>
<td><strong>Execution Model</strong></td>
<td>Interpreted, Interpreter on device</td>
<td>Compiled, Toolchain on host, Minimal stub on device</td>
</tr>
</tbody>
</table>
REPL Pipeline

<table>
<thead>
<tr>
<th>Input</th>
<th>Bitcode</th>
<th>IRUnit</th>
<th>LinkGraph</th>
<th>Binary</th>
<th>Output</th>
</tr>
</thead>
</table>

Read → Evaluate → Print

Loop
**REPL Pipeline**

<table>
<thead>
<tr>
<th>Input</th>
<th>Bitcode</th>
<th>IRUnit</th>
<th>LinkGraph</th>
<th>Binary</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Compile</td>
<td>Commit</td>
<td>Lookup</td>
<td>Finalize</td>
<td>Print</td>
</tr>
<tr>
<td>Clang for arm-none-eabi</td>
<td>Transform</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Promote declarations, wrap explicit initialisers, insert NULL checks, attach value reporter</td>
<td></td>
<td>e.g. ez::printf()</td>
<td></td>
<td>Symbols Addresses</td>
<td>Device Firmware</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Transfer</td>
<td>Execute</td>
</tr>
</tbody>
</table>

- **Compile**
  - Read
  - Clang for arm-none-eabi
  - Promote declarations, wrap explicit initialisers, insert NULL checks, attach value reporter

- **Transform**
  - Transform
  - e.g. ez::printf()

- **Commit**
  - Commit

- **Lookup**
  - Lookup
  - e.g. vsnprintf()
Transform: Return value formatter
Cling in-process vs. ez-clang out-of-process

qemu> int a = 1 + 2
(int) 3
Transform: Return value formatter

Cling in-process

- Synthesizes runtime call that invokes `printValue()` in static code
- Pass context through JITed code as `void*`
Transform: Return value formatter

ez-clang out-of-process

```c
void __ez_clang_report_value(uint32_t SeqNo, const char *Blob, size_t Size) {
    // The REPL uses this function to print expression values.
    // It knows the type of the data in this blob.
    sendMessage(ReportValue, SeqNo, Blob, Size);
}
```

- Built-in firmware function: release/0.0.5/docs/runtime.md
- Sends asynchronous ReportValue message to host
- Return value formatter:
  - Synthesizes runtime call that sends back expression result memory
  - Registers response handler that stores type info and dumps the result
Transform: Return value formatter
ez-clang out-of-process

due> int a = 1 + 2
FunctionDecl ID0001 <input_line_1> __ez_clang_expr0 'void ()'
 `-CompoundStmt
  `-DeclStmt
   `-VarDecl ID0002 a 'int' cinit
      `-BinaryOperator 'int' '+'
          | -IntegerLiteral 'int' 1
          `-IntegerLiteral 'int' 2

- Step 1: Wrap in function and compile
Transform: Return value formatter

ez-clang out-of-process

due> int a = 1 + 2
FunctionDecl ID0001 <input_line_1> __ez_clang_expr0 'void ()'
 `−CompoundStmt
   `−DeclStmt
   `−VarDecl ID0002 a 'int' cinit
     `−BinaryOperator 'int' '+'
      |−IntegerLiteral 'int' 1
      `−IntegerLiteral 'int' 2

DeclExtractor:
FunctionDecl ID0001 <input_line_1> __ez_clang_expr0 'void ()'
 `−CompoundStmt
   `−DeclRefExpr ID0003 'int' lvalue Var ID0002 'a' 'int'

- Step 2: Promote declarations and initialisers to global scope
Transform: Return value formatter

ez-clang out-of-process

due> int a = 1 + 2

ValueExtractionSynthesizer:
FunctionDecl ID0001 <input_line_1> __ez_clang_expr0 'void ()'
  `-CompoundStmt
    `-CallExpr 'void'
      | `-ImplicitCastExpr 'void (*)(unsigned int, const char *, unsigned int)'`-DeclRefExpr lvalue Function  '__ez_clang_report_value'
      | `-DeclRefExpr ID0003 'int' lvalue Var ID0002 'a' 'int'
      | `-CStyleCastExpr 'const char *' <BitCast>
      | `-UnaryOperator 'int *' prefix '&' cannot overflow
        | `-DeclRefExpr ID0003 'int' lvalue Var ID0002 'a' 'int'
        | `-UnaryExprOrTypeTraitExpr 'unsigned int' sizeof
          | `-DeclRefExpr ID0003 'int' lvalue Var ID0002 'a' 'int'
          (int) 3

> Step 3: Pass expression result to __ez_clang_report_value()
Device firmware
Interface documentation

Lookup Request

Resolve device addresses for a number of symbols. Takes an array of symbol names. Returns same-sized array of addresses. For symbols that are not found, the respective index holds a NULL value.

\[ \text{__ez_clang_rpc_lookup} \text{<array<string>>} \rightarrow \text{expected array<addr>} \]

### Input

<table>
<thead>
<tr>
<th>Field</th>
<th>Bytes</th>
<th>Example</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>B</td>
<td>02 00 00 00 00 00 00 00</td>
<td>Request for two symbols</td>
</tr>
<tr>
<td>Name 1</td>
<td>B + N</td>
<td>16 00 00 00 00 00 00 00 5f 5f 65 7a 5f 63 6c 61 6e 67 5f 72 70 63 6f 64 65 63 6f 6e 64 65 63 6c 61 6e 67 5f 72 70 63 6f 64 65 63 6f 6e 64 65 63 6c</td>
<td>First symbol: __ez_clang_rpc_execute</td>
</tr>
<tr>
<td>Name 2</td>
<td>B + N</td>
<td>17 00 00 00 00 00 00 00 5f 5f 65 7a 5f 63 6c 61 6e 67 5f 72 70 63 6f 64 65 63 6f 6e 64 65 63 6c 61 6e 67 5f 72 70 63 6f 64 65 63 6f 6e 64 65 63 6c</td>
<td>Second symbol: __ez_clang_report_value</td>
</tr>
</tbody>
</table>

### Output

<table>
<thead>
<tr>
<th>Field</th>
<th>Bytes</th>
<th>Example</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success Code</td>
<td>1</td>
<td>00</td>
<td>No errors during lookup</td>
</tr>
<tr>
<td>Count</td>
<td>B</td>
<td>02 00 00 00 00 00 00 00</td>
<td>Result with two addresses</td>
</tr>
<tr>
<td>Address 1</td>
<td>B</td>
<td>01 1a 00 00 00 00 00 00</td>
<td>First symbol @ 0x0001A01</td>
</tr>
<tr>
<td>Address 2</td>
<td>B</td>
<td>01 1b 00 00 00 00 00 00</td>
<td>Second symbol @ 0x0001B01</td>
</tr>
</tbody>
</table>

Documentation of the RPC endpoint __ez_clang_rpc_lookup

Interfaces are subject to change and documented by version:

- RPC interface:
  - [release/0.0.5/docs/rpc.md](https://github.com/echtzeit-dev/ez-clang-arduino)
  - [release/0.0.5/docs/runtime.md](https://github.com/echtzeit-dev/ez-clang-qemu)

- Reference implementations:
  - [https://github.com/echtzeit-dev/ez-clang-arduino](https://github.com/echtzeit-dev/ez-clang-arduino)
  - [https://github.com/echtzeit-dev/ez-clang-qemu](https://github.com/echtzeit-dev/ez-clang-qemu)
Outlook

Next few weeks

1. Publish device configuration API
2. Add support for ARMv6 CPUs (Cortex® M0 and M0+)
3. Load standard libraries at runtime
4. Bugfixing and stability
5. Prototype APIs for external Command Line and Compiler
6. Target AVR
Can’t wait to hear your questions!

QEMU:

→ docker run --rm -it echtzeit/ez-clang:0.0.5

Device at <port>:

→ docker run --device=<port>:/dev/ttyACM0 \
  --rm -it echtzeit/ez-clang:0.0.5

Subscribe to monthly updates: https://echtzeit.dev/ez-clang