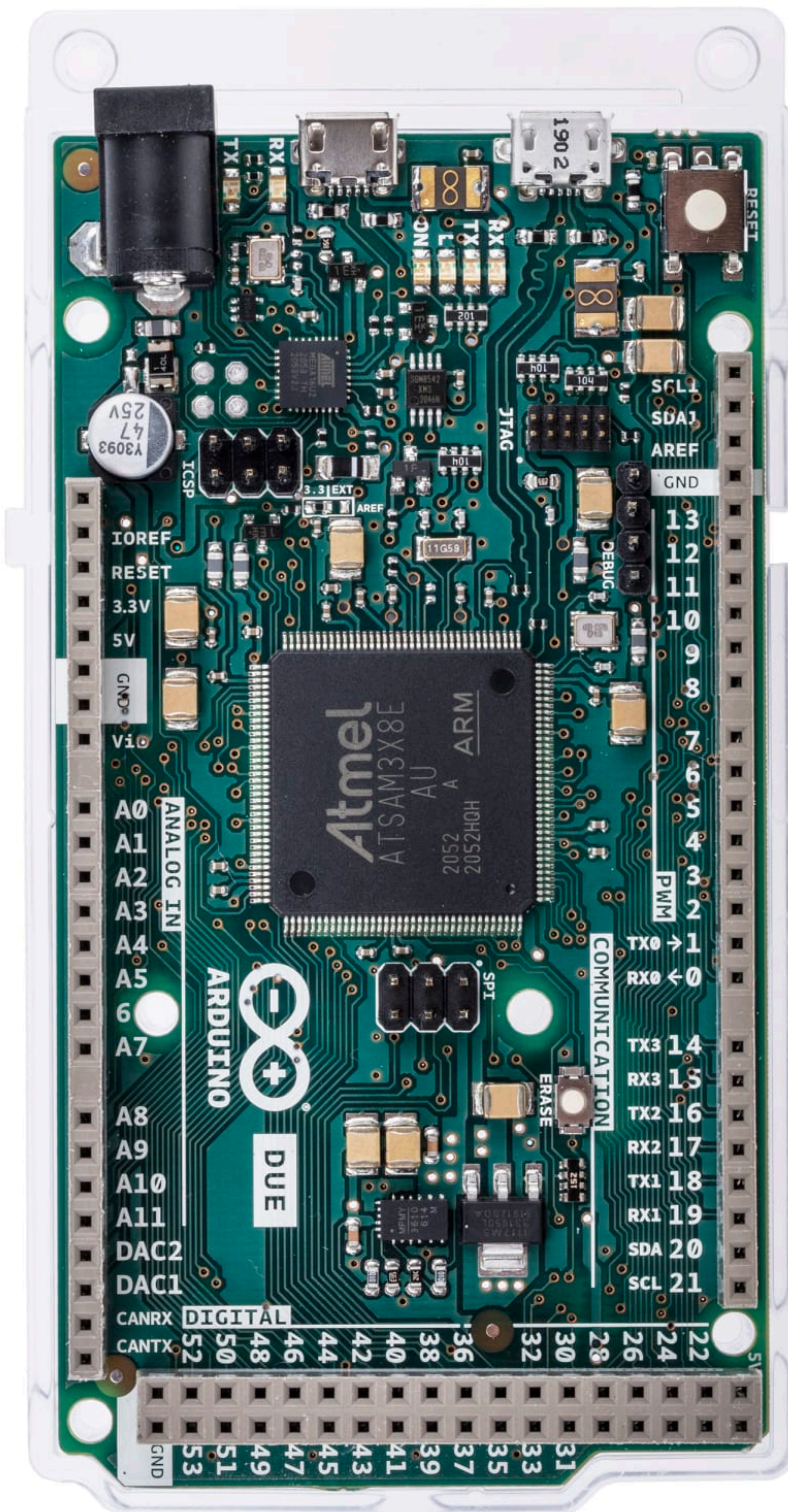


ez-clang

https://echtzeit.dev/ez-clang

C++ REPL for bare metal embedded devices

Stefan Gränitz · May 11, 2022

European LLVM Developers' Meeting

ez-clang

- 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

Host

Serial connection

Device

Command Prompt

Clang Frontend

JIT Backend

ORCv2 + JITLink

Prebuilt Libraries

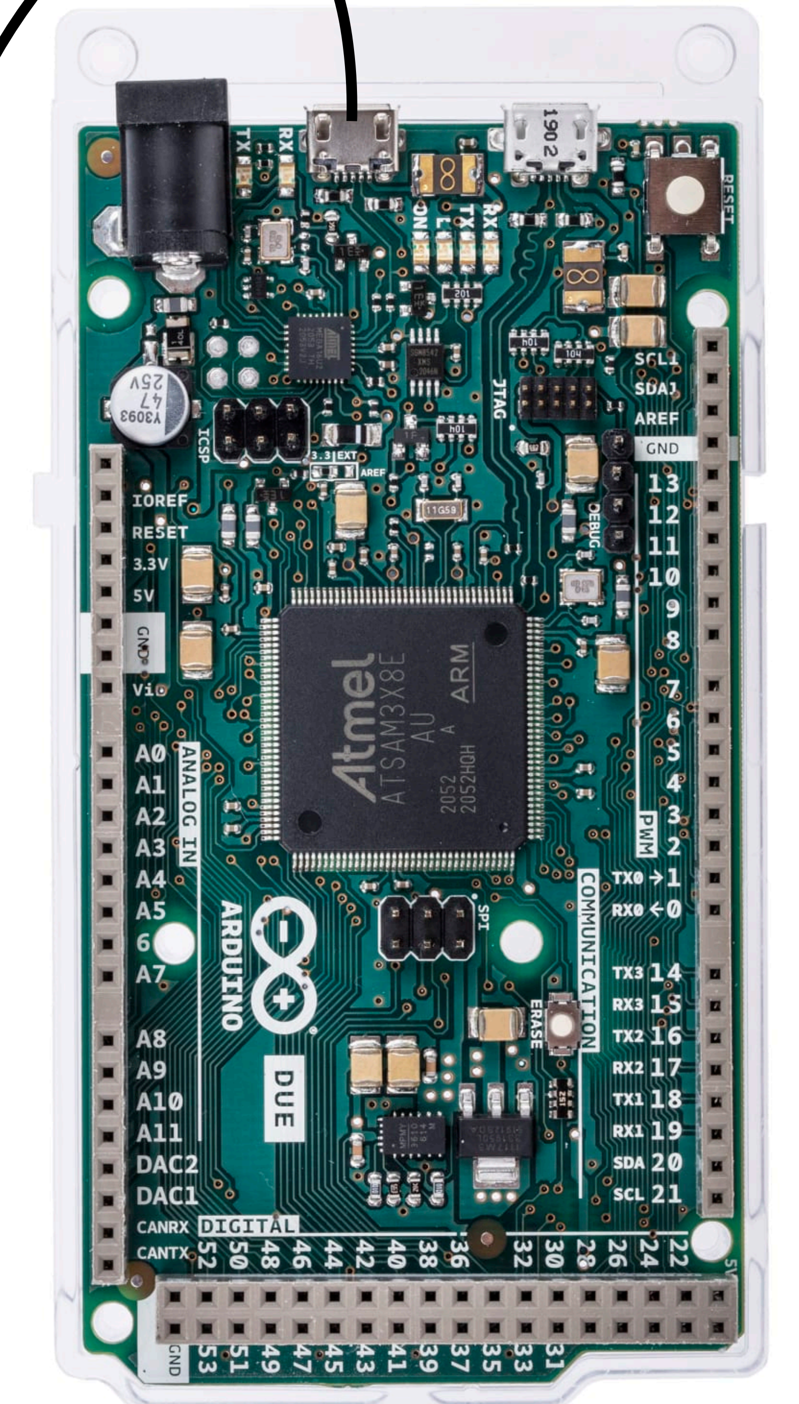
e.g. ez/stdio/printf.a

Tools

Bundled firmwares, scripts, etc.

RPC

multi-threaded

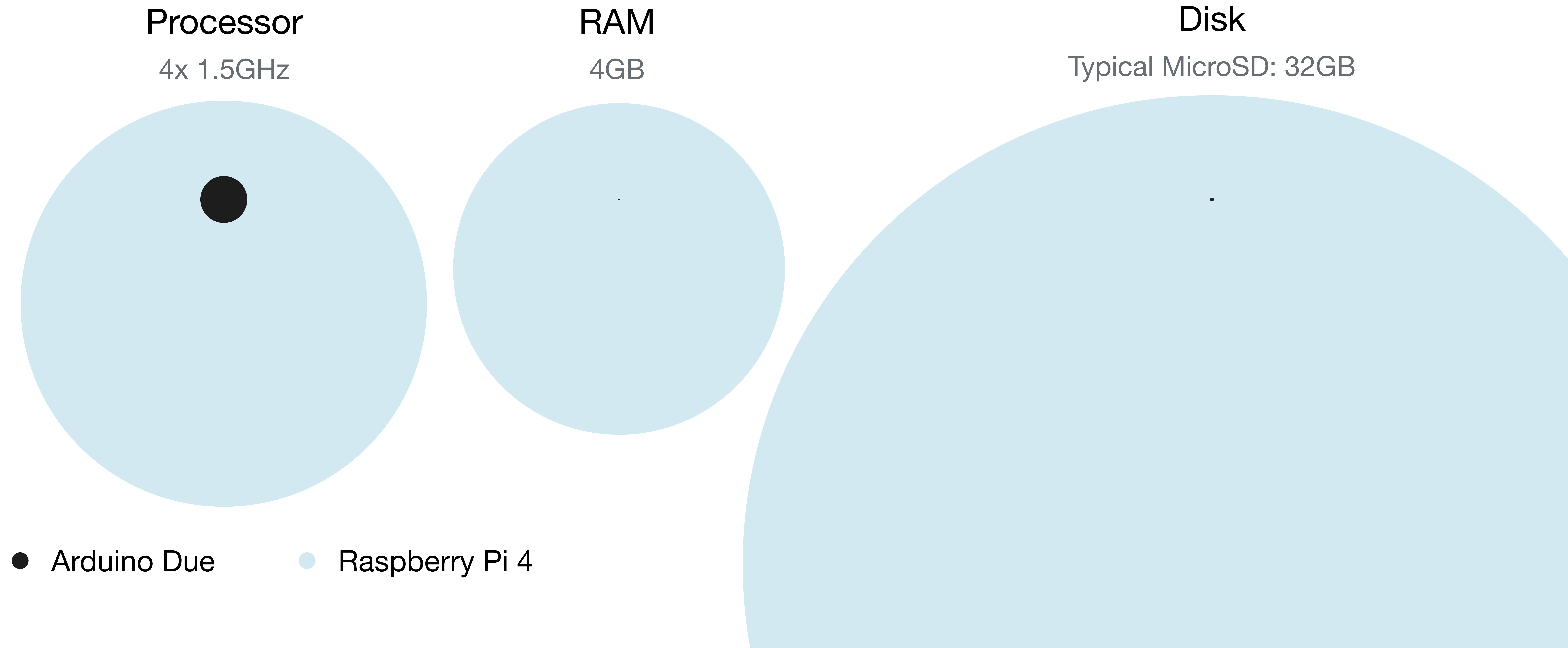


ez-clang firmware

single-threaded

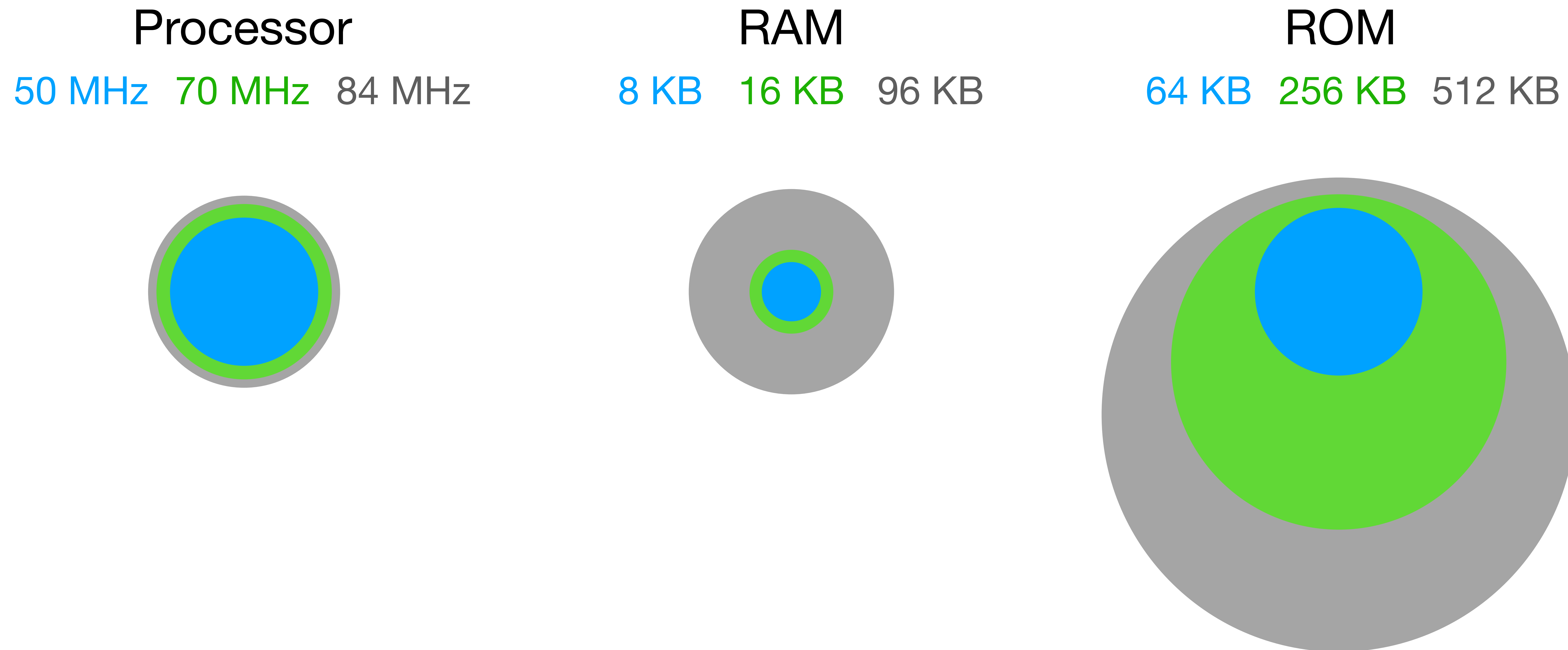
Hardware Dimensions

Raspberry Pi 4 vs. Bare Metal Microcontrollers



Hardware Dimensions

Bare Metal Microcontrollers

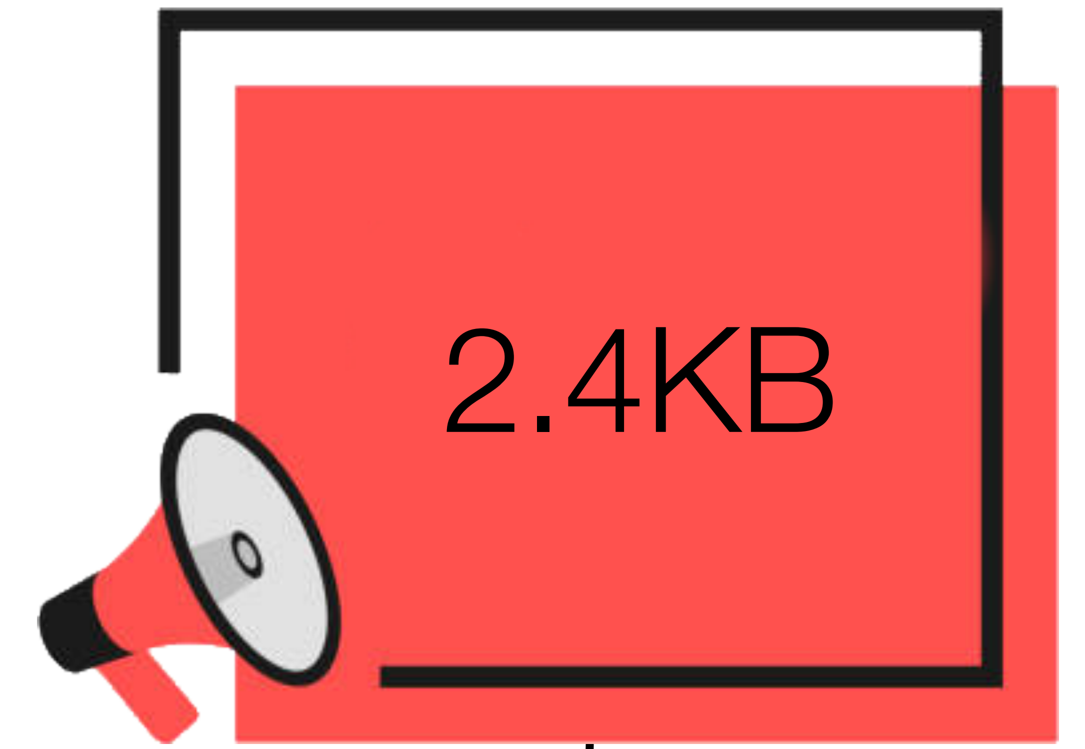


- ez-clang
- Stellaris lm3s811evb
- MicroPython (min. requirements*)
- Arduino Due

* 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”

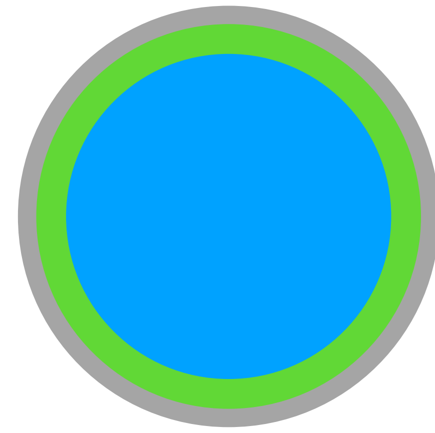
Hardware Dimensions

Bare Metal Microcontrollers



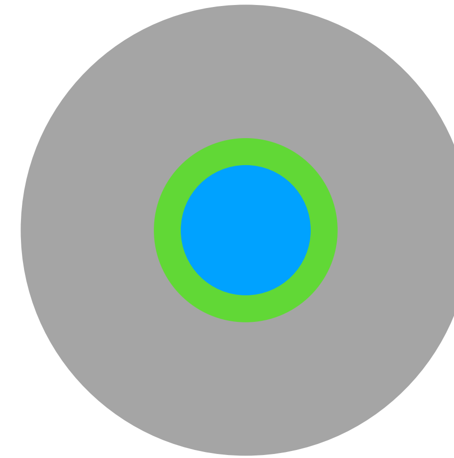
Processor

50 MHz 70 MHz 84 MHz



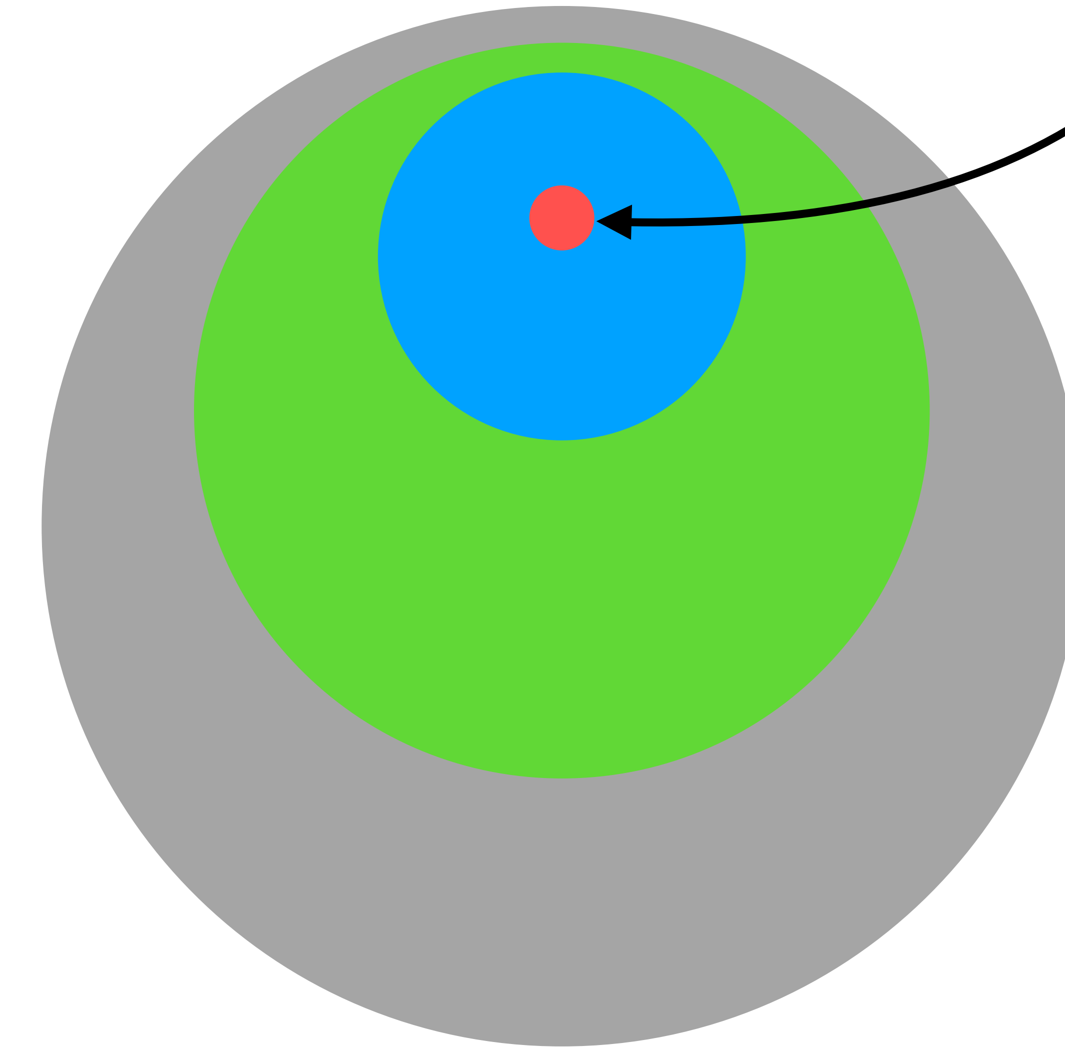
RAM

8 KB 16 KB 96 KB



ROM

64 KB 256 KB 512 KB

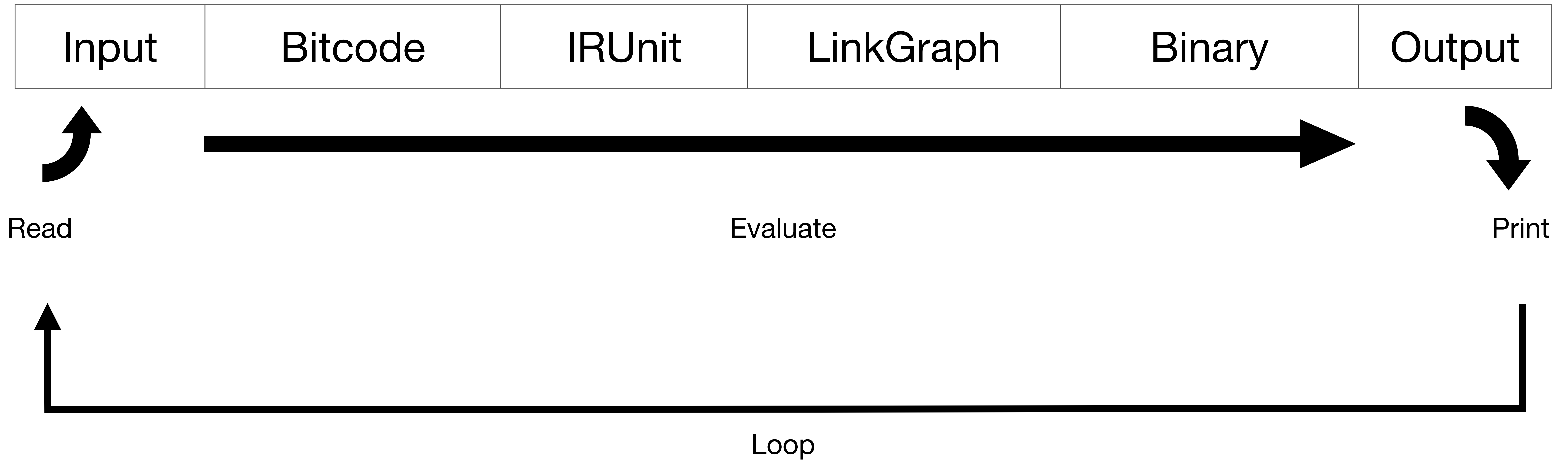


- ez-clang
- Stellaris lm3s811evb
- MicroPython (min. requirements)
- Arduino Due

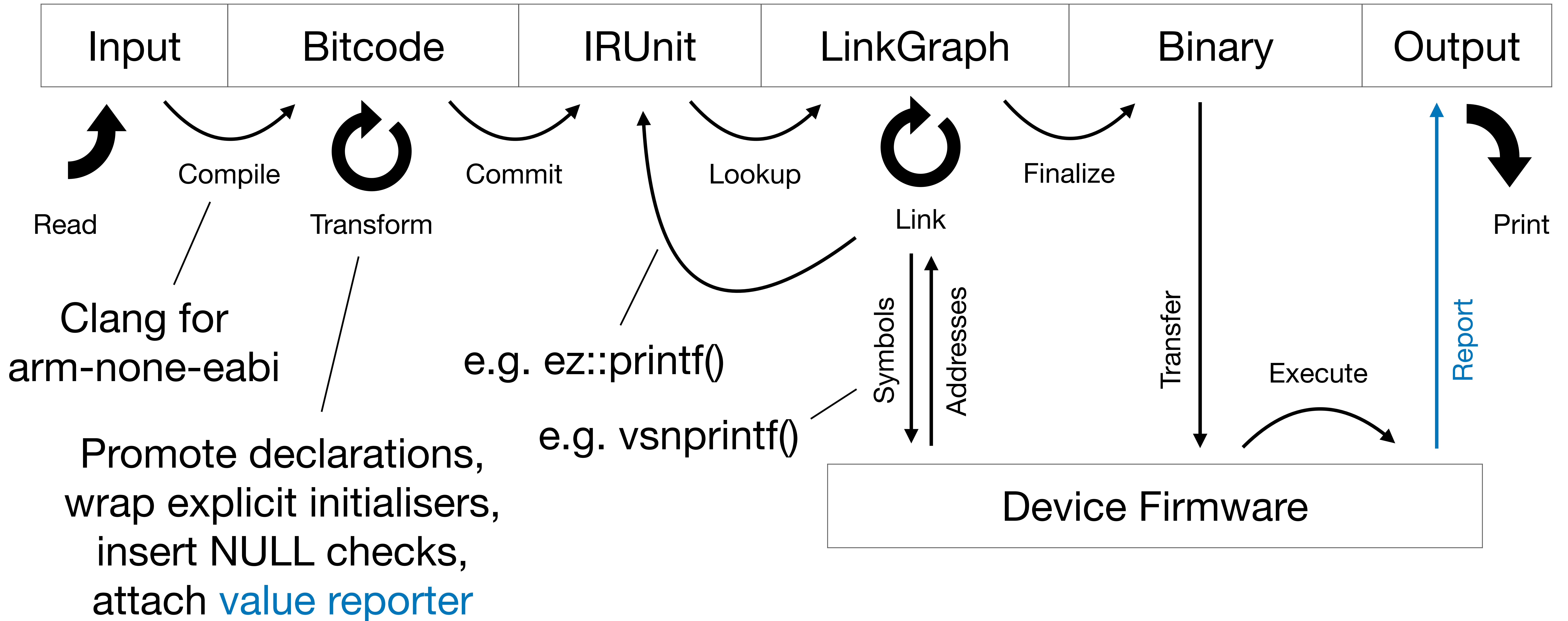
Competitors?

	MicroPython	**ez-clang**
Language	(Reduced) Python Dialect	Standard C++
Standard Libraries	Subset of Python Stdlib Feature-set depends on device capacity	Newlib instead of glibc STL adaptations like ETL
Execution Model	Interpreted, Interpreter on device	Compiled, Toolchain on host, Minimal stub on device

REPL Pipeline



REPL Pipeline



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

```
#0:  0x0000053cd3a4 cling`cling::printValue()
#1:  0x0000053d05b7 cling`executePrintValue<int>()
#2:  0x0000053ceecd cling`printUnpackedClingValue()
#3:  0x0000053cf73b cling`cling::valuePrinterInternal::printValueInternal()
#4:  0x0000053cc68e cling`cling::Value::print() const
#5:  0x0000053cc875 cling`cling::Value::dump() const
#6:  0x0000053c57fc cling`dumpIfNoStorage()
#7:  0x0000053c5922 cling`cling::runtime::internal::setValueNoAlloc()
#8:  0x7ffff7912042 JITed code
#9:  0x0000052cd821 cling`cling::IncrementalExecutor::executeWrapper() const
#10: 0x00000530d278 cling`cling::Interpreter::RunFunction()
...
```

- 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

```
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'
    |-IntegerLiteral 'unsigned int' 1
    |-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.

```
__ez_clang_rpc_lookup(array<string>) -> expected<array<addr>>
```

Input

Field	Bytes	Example	Interpretation
Count	8	`02 00 00 00 00 00 00 00`	Request for two symbols
Name 1	8 + N	`16 00 00 00 00 00 00 00` `5f 5f 65 7a 5f 63 6c 61 6e 67 5f 72 70 63 5f 65 78 65 63 75 74 65`	First symbol: __ez_clang_rpc_execute
Name 2	8 + N	`17 00 00 00 00 00 00 00` `5f 5f 65 7a 5f 63 6c 61 6e 67 5f 72 65 70 6f 72 74 5f 76 61 6c 75 65`	Second symbol: __ez_clang_report_value

Output

Field	Bytes	Example	Interpretation
Success Code	1	`00`	No errors during lookup
Count	8	`02 00 00 00 00 00 00 00`	Result with two addresses
Address 1	8	`01 1A 00 00 00 00 00 00`	First symbol @ `0x00001A01`
Address 2	8	`01 1B 00 00 00 00 00 00`	Second symbol @ `0x00001B01`

⇧ 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
- Runtime interface:
  release/0.0.5/docs/runtime.md
- Reference implementations:
  https://github.com/echtzeit-dev/ez-clang-arduino
  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

ez-clang

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