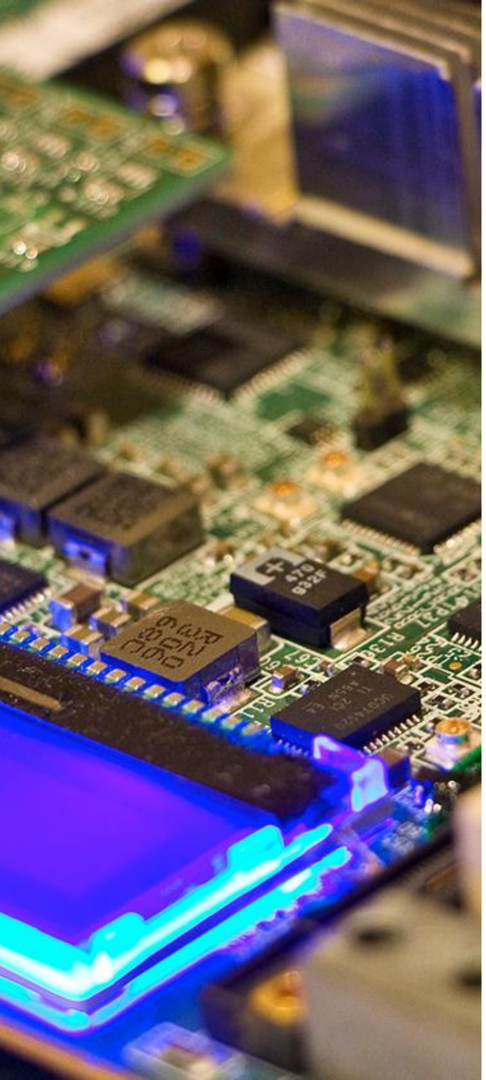
Android and modern toolchains - clang 3.6 and gcc 5.0, binutils 2.25

Presented by

Bernhard "Bero" Rosenkränzer

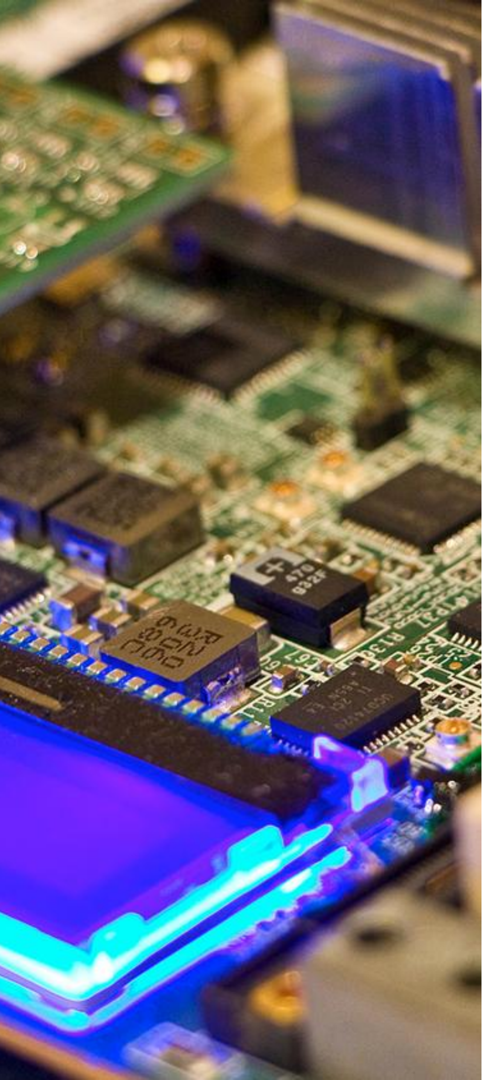
Date

Android Builders Summit 2015



Toolchains currently used by AOSP

- AOSP (both current master branch and the recent 5.1.0 release) defaults to using slightly modified versions of
  - gcc 4.8 for ARMv7, x86 and x86_64
  - gcc 4.9 for ARMv8 and MIPS
- binutils is 2.23.2 for gcc 4.8, 2.24 for gcc 4.9.



binutils 2.25

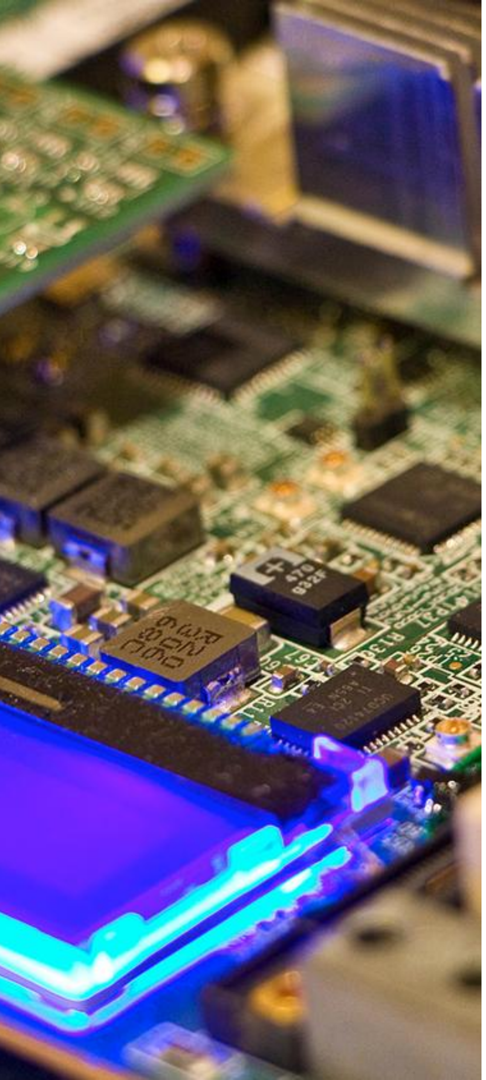
- Dropping in binutils 2.25 is painless - only change required: Make sure Nexus 10 blobs don't try to strip files that aren't target CPU binaries (should be fixed in the first place)
- Interesting addition in 2.25: gold linker for aarch64

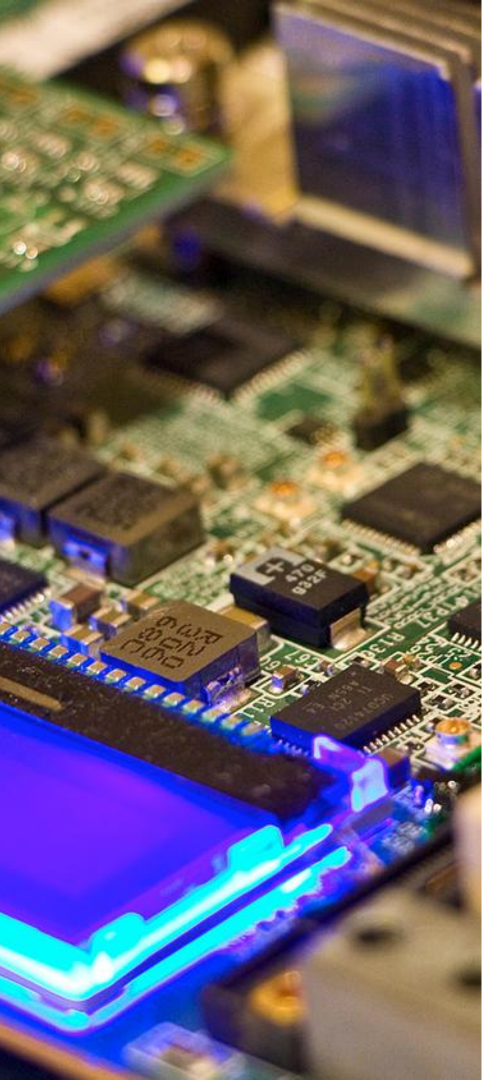
Status of CLANG support

After having submitted 117 related patches, AOSP master can be built with clang unmodified.
Set

```
USE_CLANG_PLATFORM_BUILD=true
```

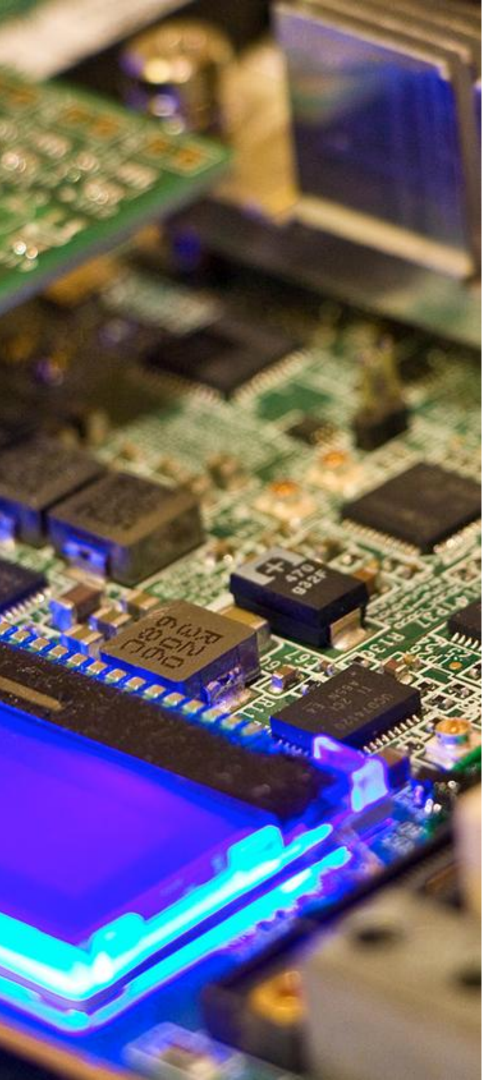
on the make command line to use it.





Status of CLANG support

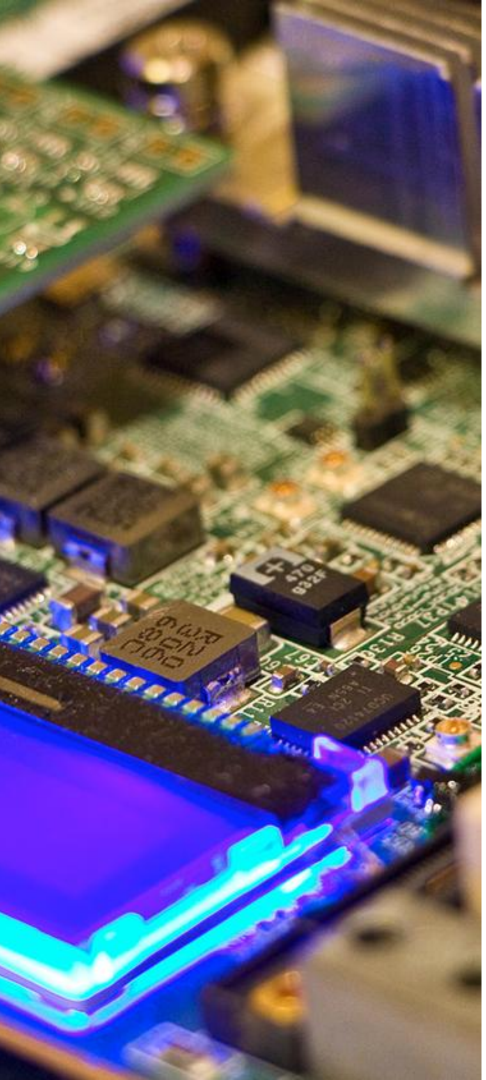
The supported version of clang comes with AOSP - it is a slightly modified version of a snapshot taken from svn shortly before the 3.6 release.



Status of CLANG support

AOSP 5.1.0 can *almost* be built with clang - it is missing 2 patches in Bionic, both oneliners.

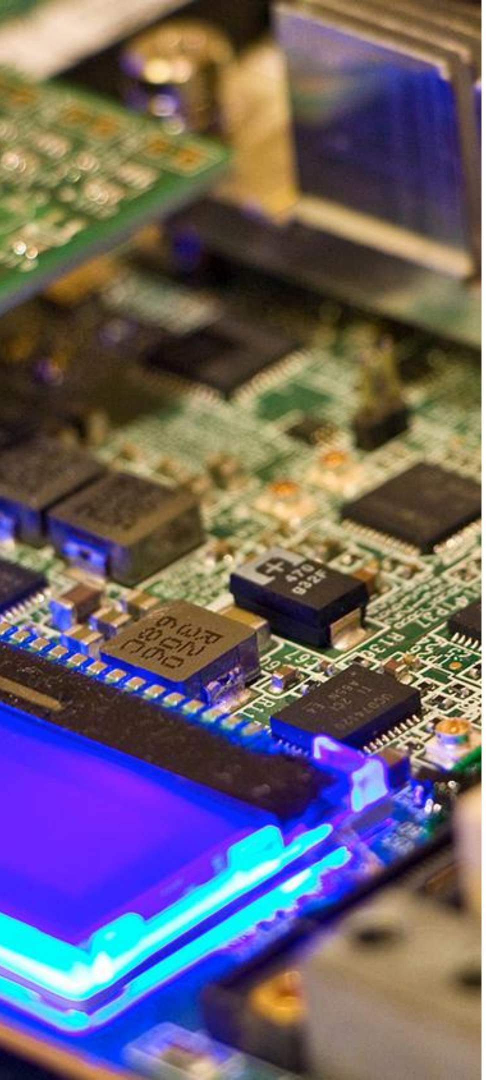
Backported patches are available.



Status of CLANG support

Clang based builds currently fall back to using gcc to build some subprojects:

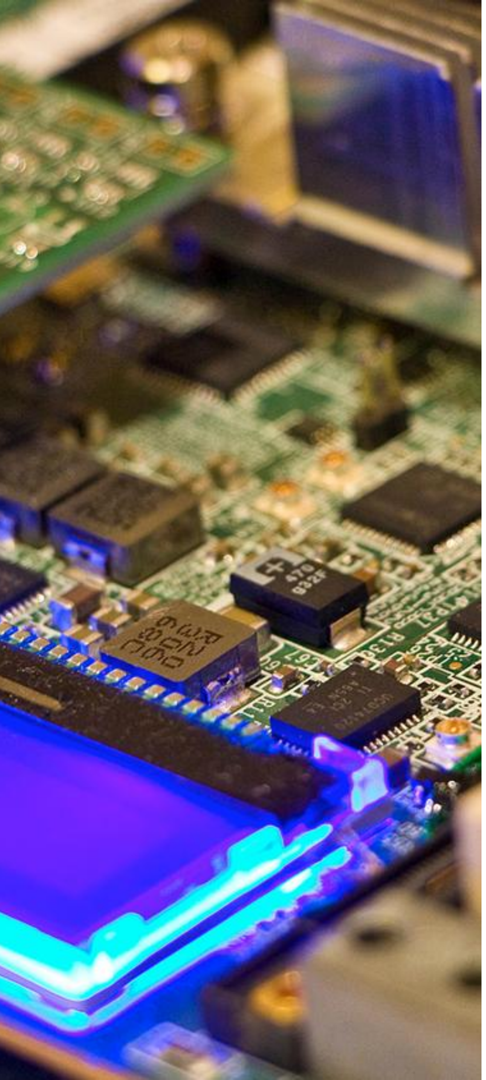
- perf
    - compile failure, real fix available



Status of CLANG support

Clang based builds currently fall back to using gcc to build some subprojects:

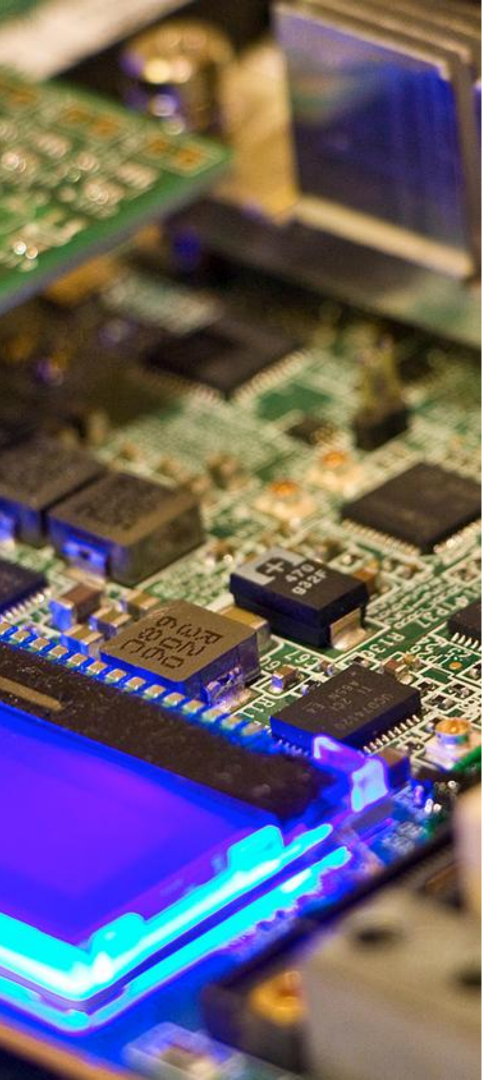
- elfutils
  - compile failures due to heavy use of nested functions, fix available, but will never be accepted upstream



Status of CLANG support

Clang based builds currently fall back to using gcc to build some subprojects:

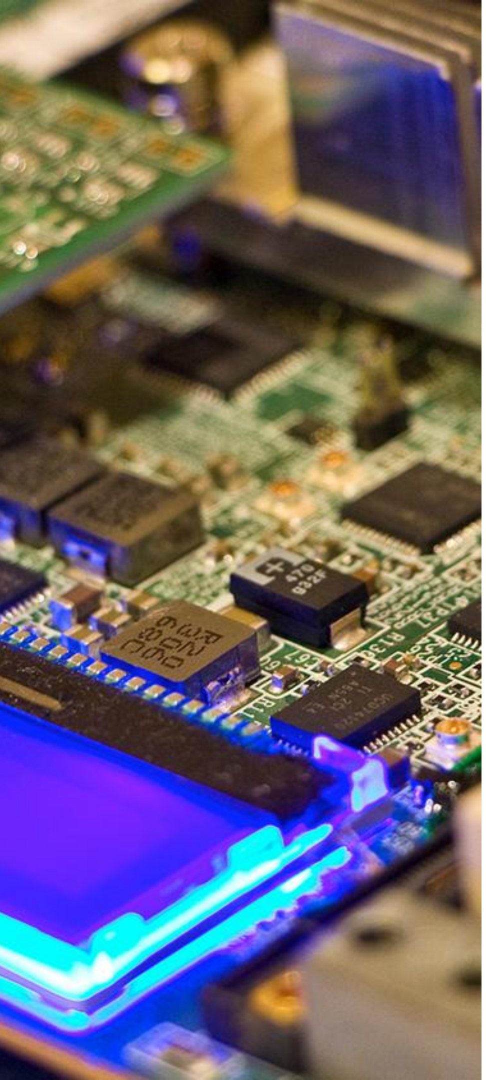
- bionic linker
  - reasoning unknown, it can be built with clang



Status of CLANG support

Some other subprojects are always built with clang, even in gcc builds:

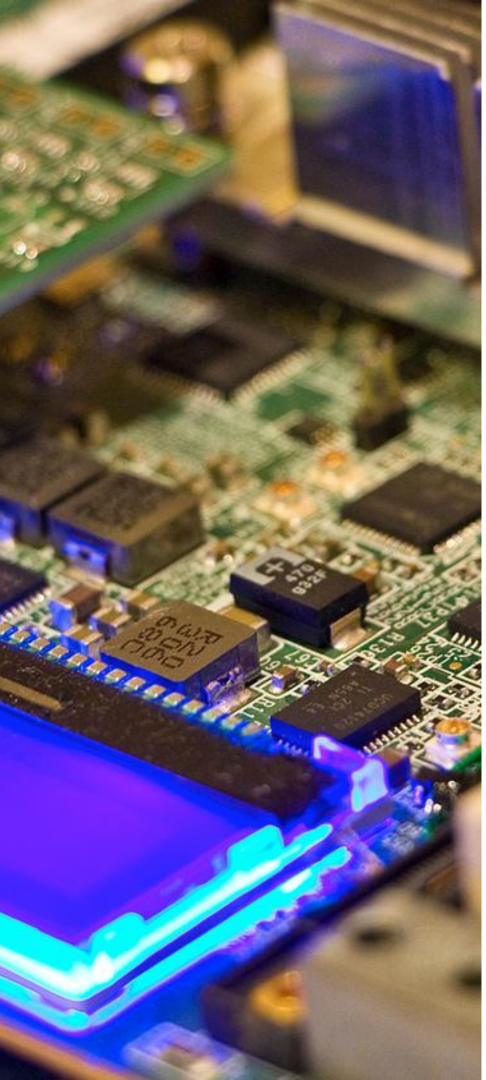
- parts of external/chromium_org
    - probably to reduce the potential for surprise -- Chromium uses clang for desktop OS builds too.
      gcc can build it.



Status of CLANG support

Some other subprojects are always built with clang, even in gcc builds:

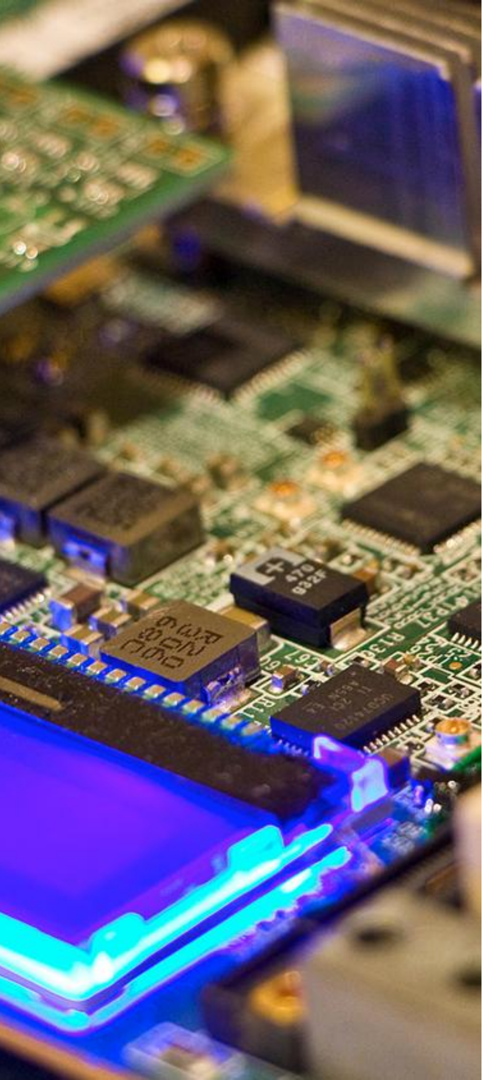
- libpng
  - AOSP's libpng makes use of clang's -ftrapv option to catch integer overflow errors



Status of CLANG support

Some other subprojects are always built with clang, even in gcc builds:

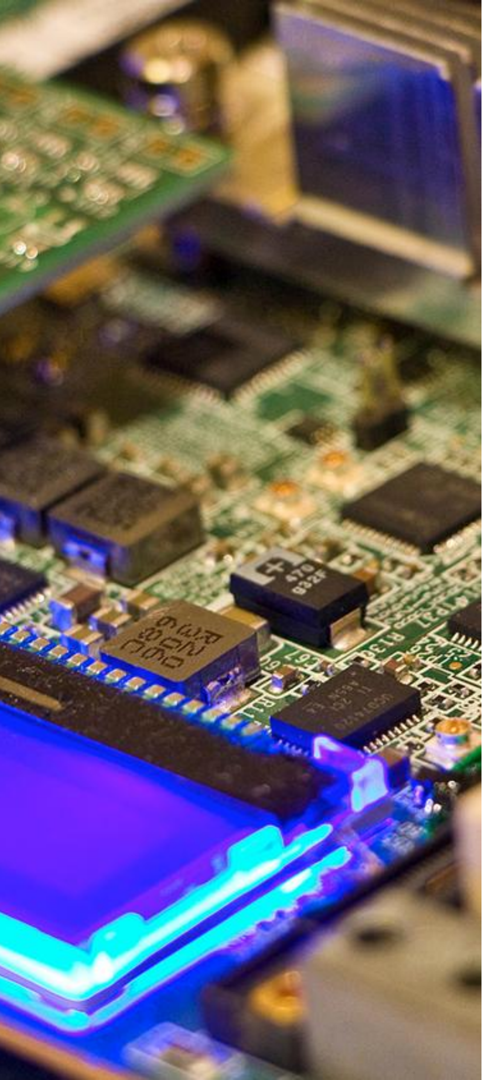
- llvm, compiler-rt, libcxx-abi, libcxx, mclinker
  - likely because they're clang sibling projects



Status of CLANG support

Some other subprojects are always built with clang, even in gcc builds:

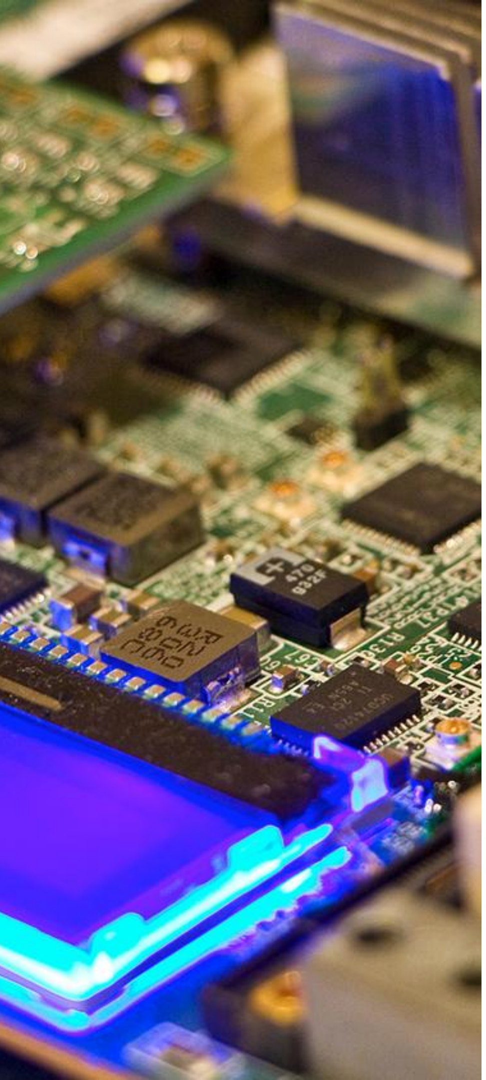
- scrypt
- srec
- libexif
- openssl



Status of CLANG support

Some other subprojects are always built with clang, even in gcc builds:

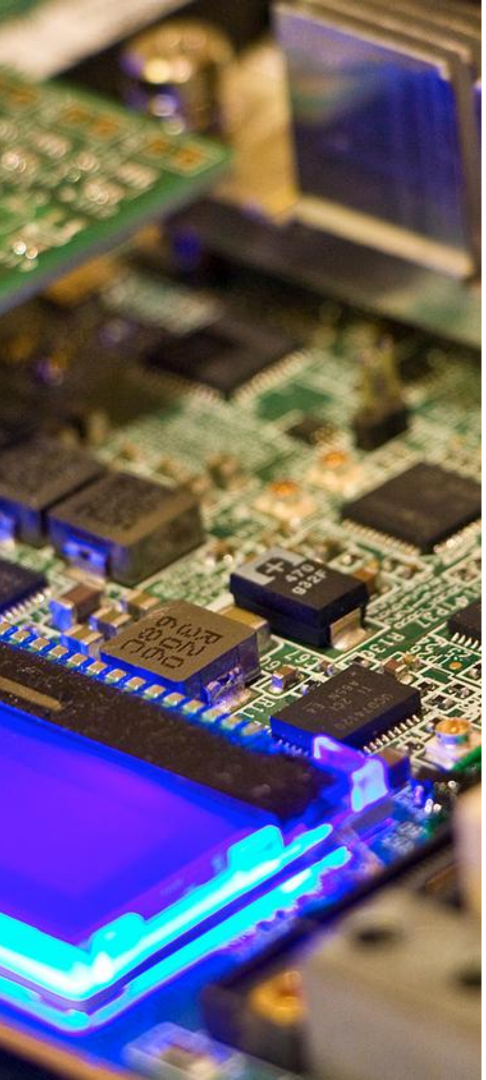
- vixl
- gtest
- conscrypt



Status of CLANG support

Some other subprojects are always built with clang, even in gcc builds:

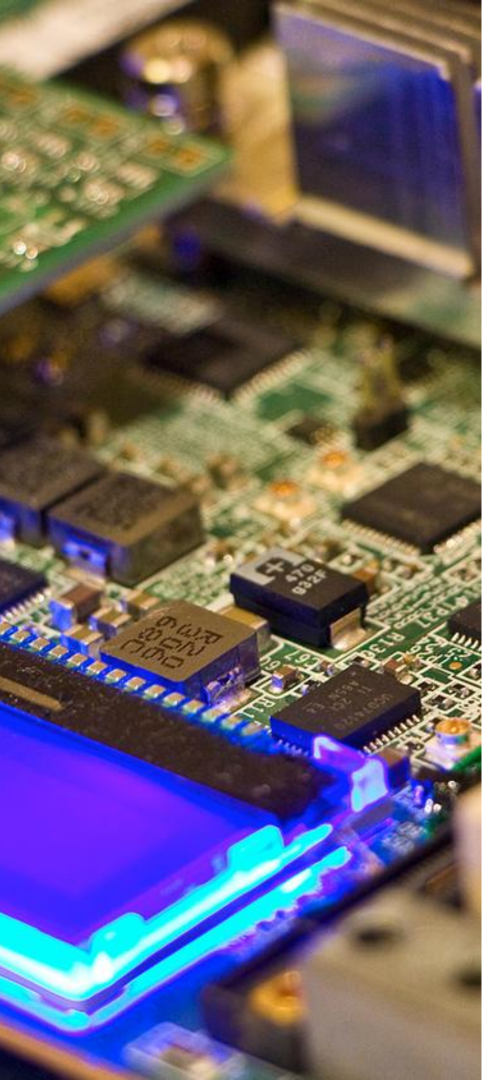
- libnativehelper
- libc_cxa in Bionic
- libnativebridge
- net
- dalvikvm
- LatinIME



Status of CLANG support

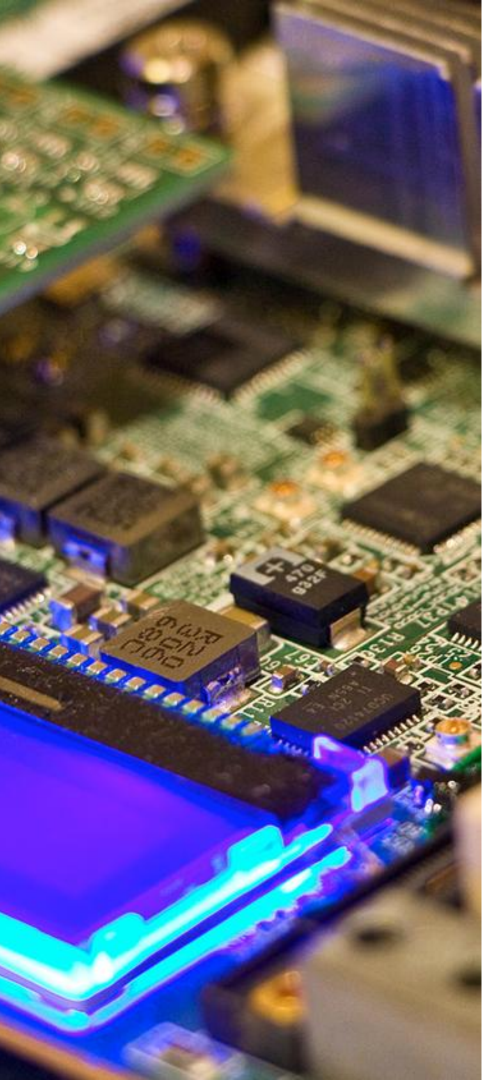
Some other subprojects are always built with clang, even in gcc builds:

- slang
- libbcc
- surfaceflinger
- renderscript
- libcore



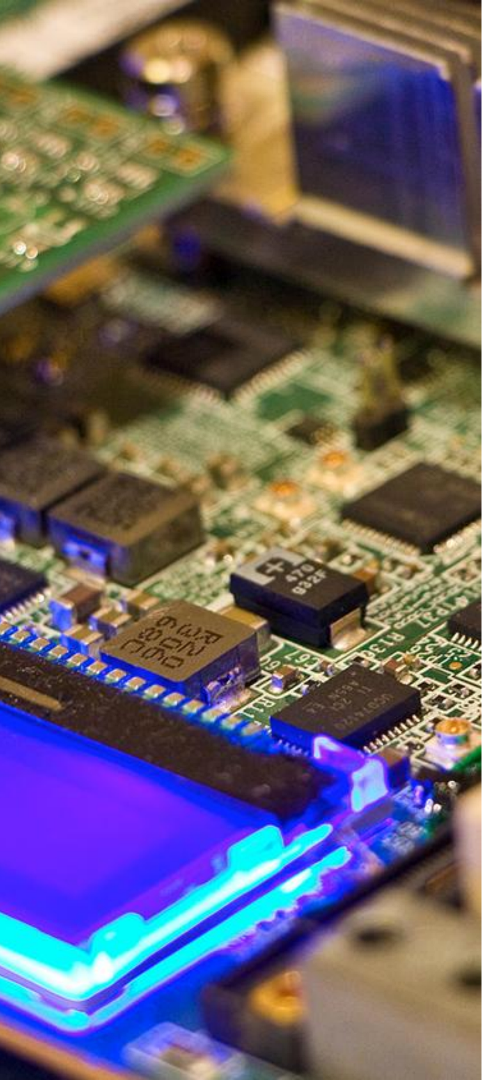
Status of gcc 5.0 support

- AOSP master can be built with gcc 5.0 snapshots with a few problems - most annoyingly, an autogenerated file in Chromium isn’t compatible with it



Status of gcc 5.0 support

- The resulting system boots and works for the most part, but the browser is unstable if built with gcc 5

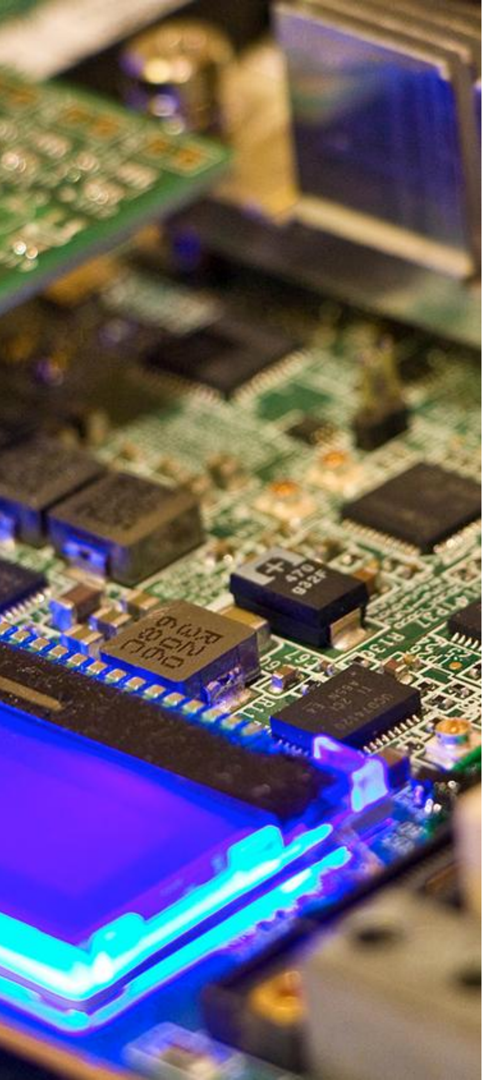


Stability

Both gcc and clang based builds run well and can pass the CTS (Compatibility Test Suite).

gcc 5.0 snapshot builds generally work fine, but have problems with the browser - this causes CTS failures as well.

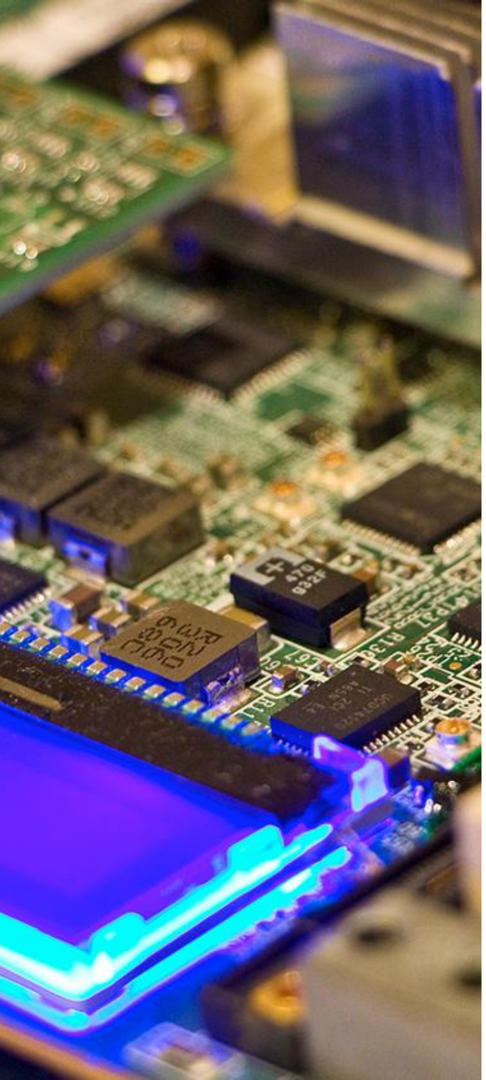
Tests with gcc 5.0 snapshots were done in late January, the situation may have improved with newer builds.



Code size

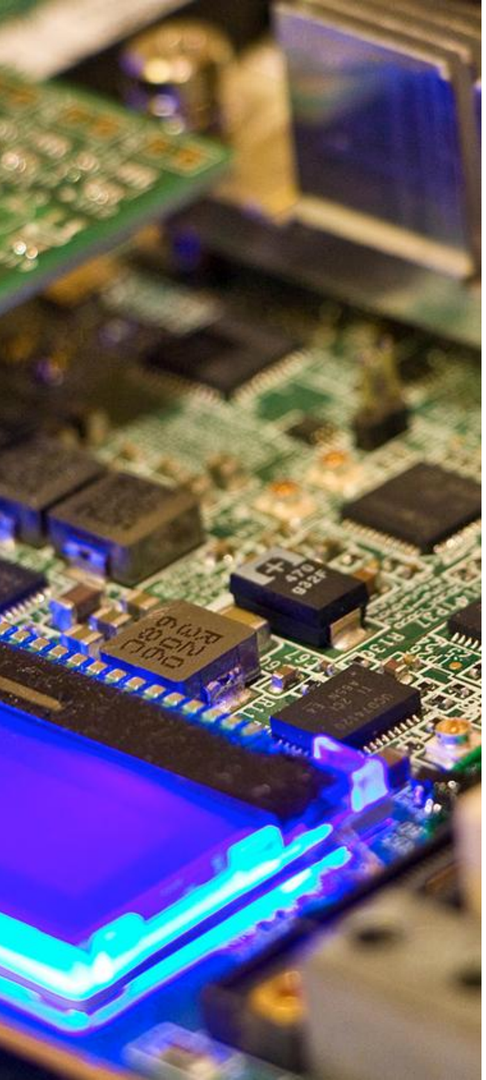
- On ARMv7, clang generates slightly larger binaries than gcc.
- On ARMv8, clang generates slightly smaller binaries than gcc.
- In both cases, the difference in code size is negligible (less than 2.5%)

Linaro



Code size

- The difference in code size between different current gcc versions (4.8 from AOSP, Linaro 4.9, 5.0 snapshot) is even smaller.

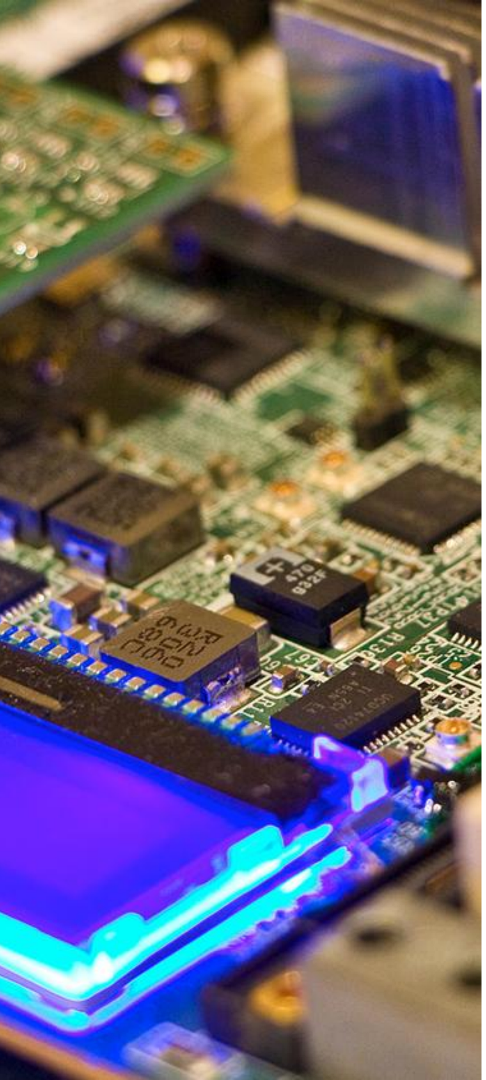


Benchmark results

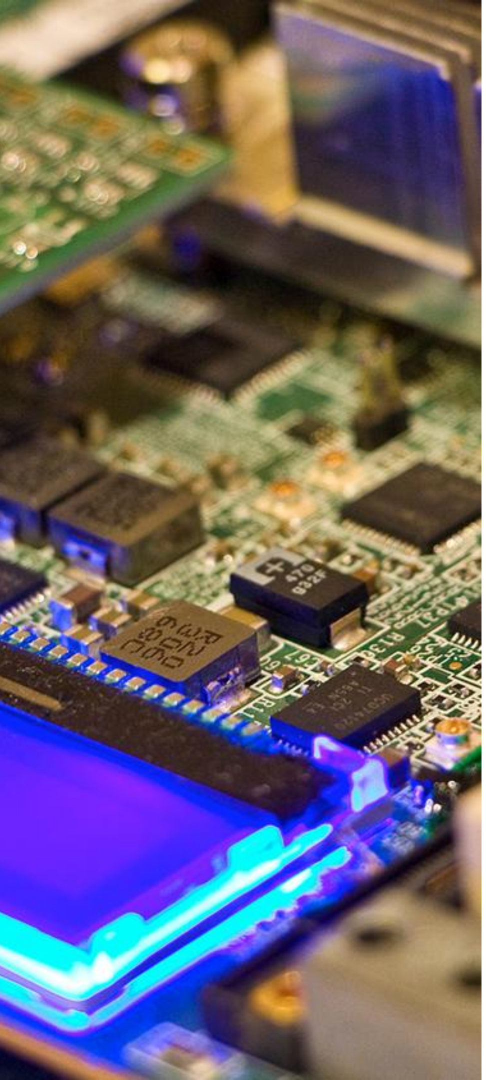
There is no clear winner. In most benchmarks, gcc is slightly ahead, in some, clang wins. gcc is generally better at multithreaded code.

Detailed results are on the Linaro wiki:
https://wiki.linaro.org/Platform/Android/GccClangBenchmark-2014-12
https://wiki.linaro.org/Platform/Android/GccClangBenchmark-2015-01

Benchmark results



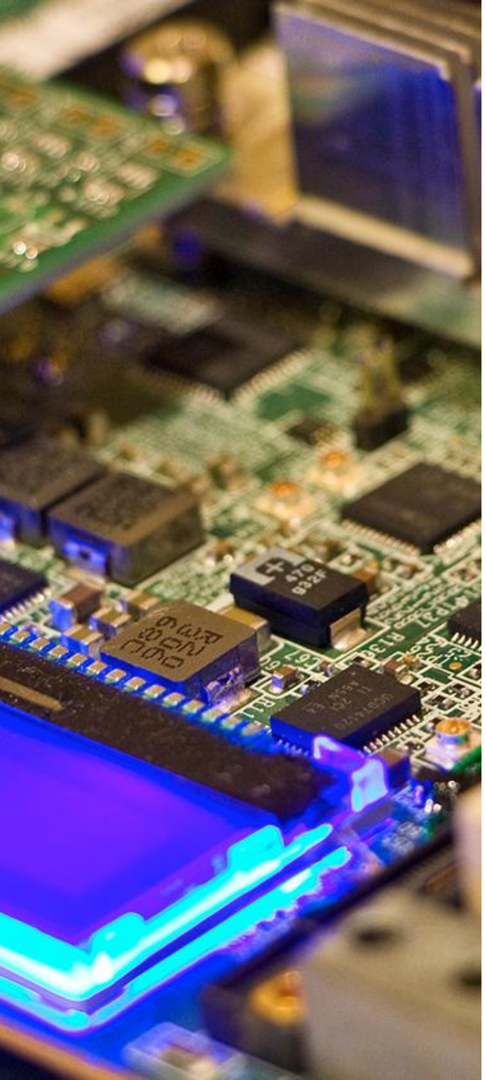
Since most common benchmarks are not Open Source and have not been updated with proper Aarch64 builds, some benchmark results from 64bit devices are actually in 32bit code -- not measuring the performance of the code generated by the 64bit compilers appropriately.



Benchmark results

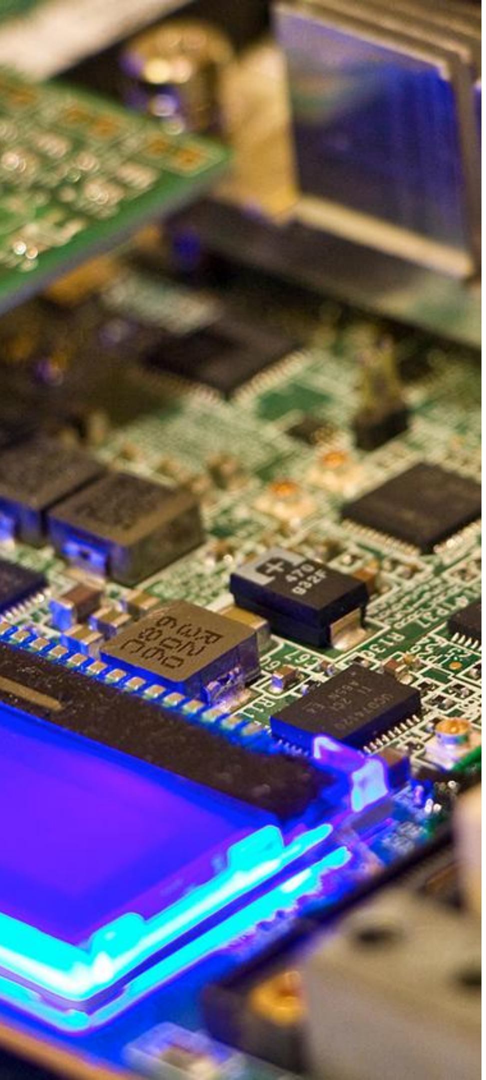
Proper 64 bit benchmarks will likely shift the results a little in Clang's favor - its 64bit code tends to be better than its 32bit code.

Also, tweaking compiler flags for clang builds may help -- people have had much more time to figure out the best flags for gcc than for clang.



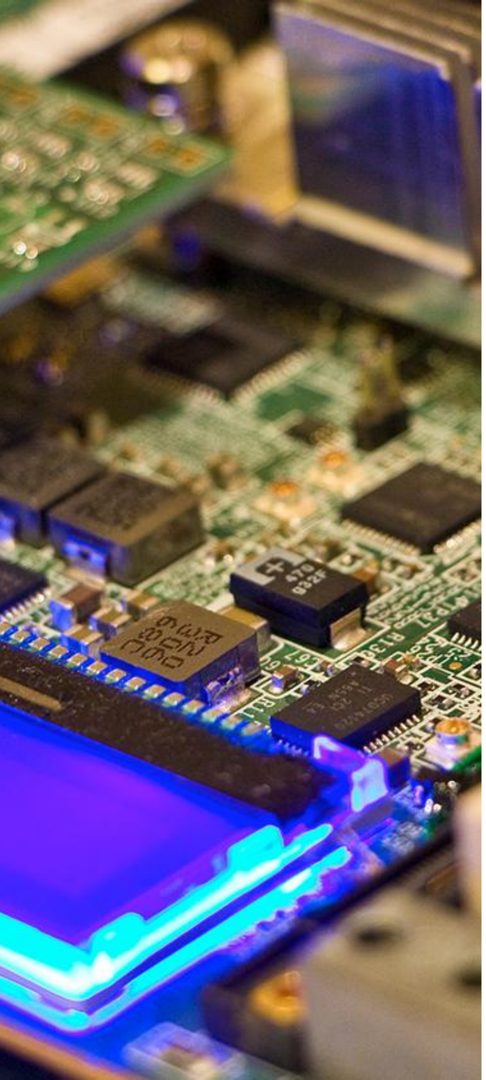
Benchmark results

AOSP currently makes some incorrect assumptions about clang -- e.g. “-mcpu=cortex-a15 has to be stripped from compiler flags” -- this used to be true, but clang added A15 support in 3.6. Clang has been updated, but the Makefiles adjusting compiler flags have not.



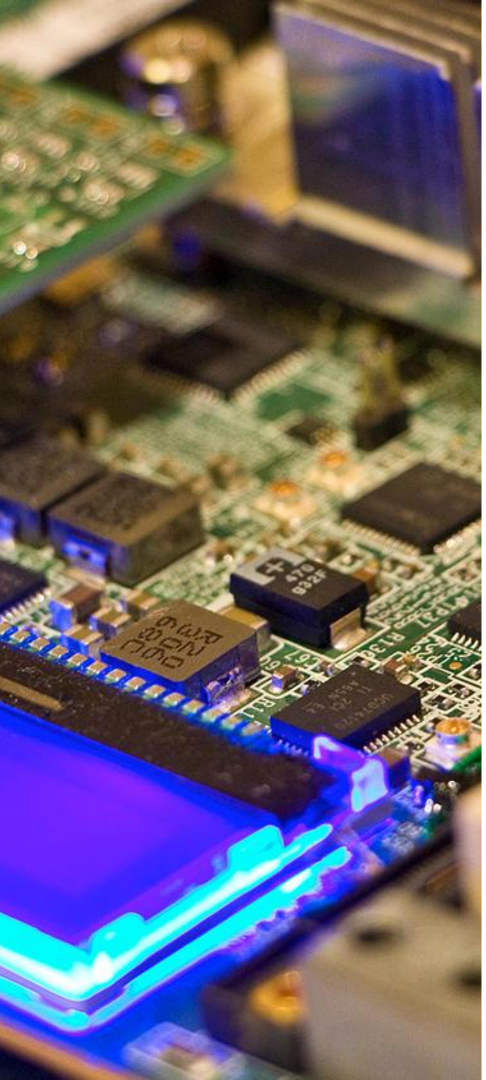
Getting a second opinion...

... is not just for medicine anymore -- asking another compiler for its opinion on code can turn up interesting bugs.



Interesting bugs found by clang

```
void something(char n[30]) {
  if(!memcmp(buffer, n, sizeof(n))) {
    …
  }
}
```



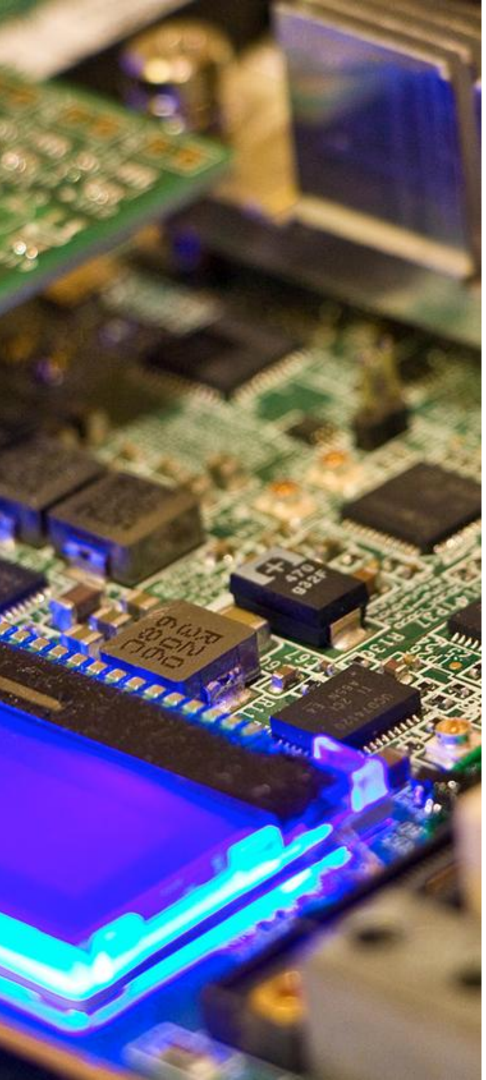
Interesting bugs found by clang

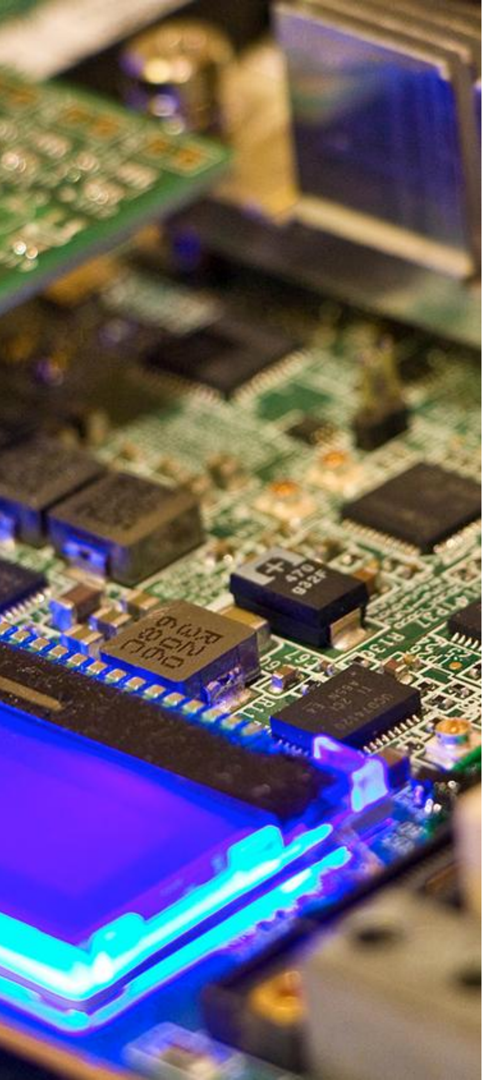
```
void something(char n[30]) {
   if(!memcmp(buffer, n, sizeof(n))) {
      …
   }
}
```

size of a pointer - not quite 30

Interesting bugs found by clang

```
unsigned char a[X];
for(int i=0; i<X; i++)
   b = a ? tagCpe++ : tagSce++;
```

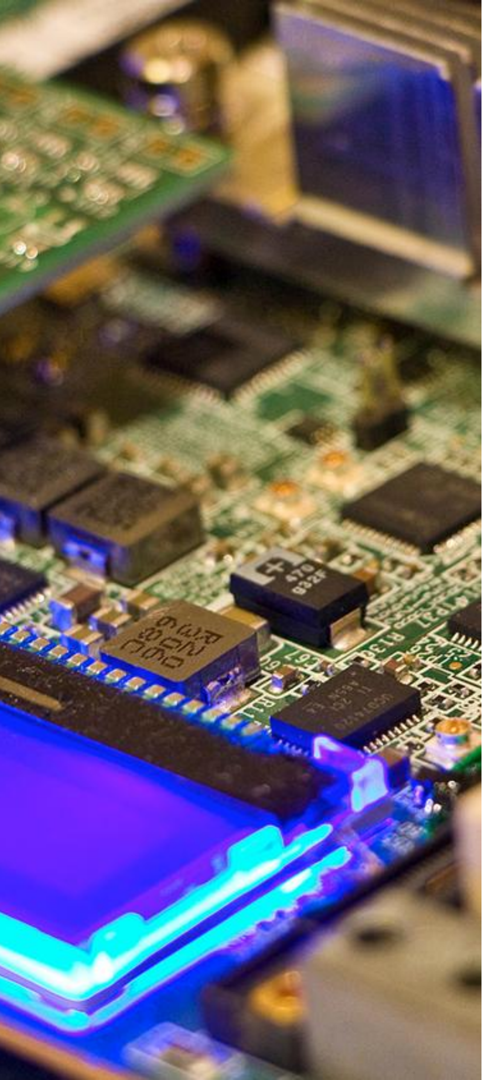




Interesting bugs found by clang

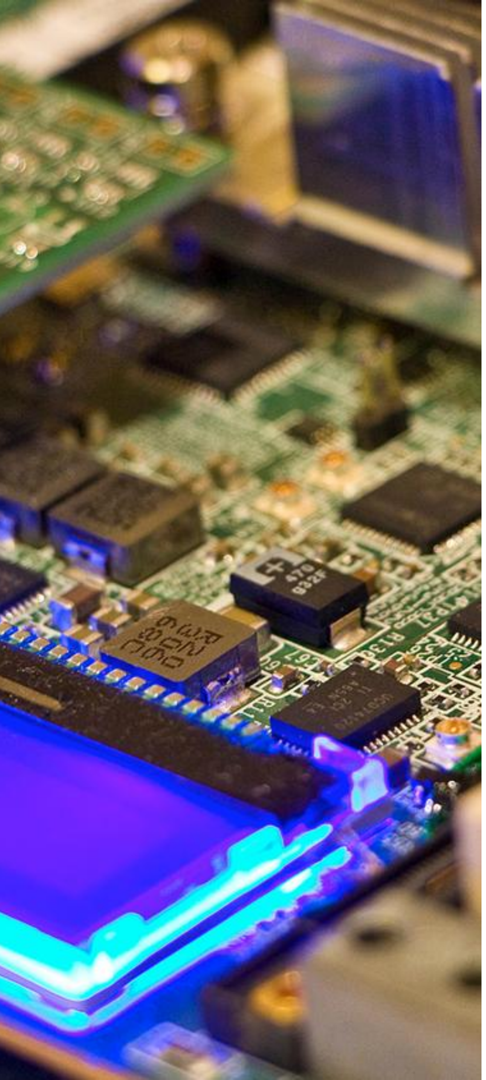
```
unsigned char a[X];
for(int i=0; i<X; i++)
   b = a ? tagCpe++ : tagSce++;
```

always true -- address of an array. This should have been a[i]



Things to avoid for compatibility

- Variable length arrays in structs
- Variable length arrays of non-POD types
- Empty structs
- Array subscripts of type “char” (`value['0']=0;`)
- `asm("add w0, w0, #-1");` (converted to `sub w0, w0, #1` by gas, but not by clang)



Things to avoid for compatibility

- Undefined internal functions and variables -- even if they aren't used:

```
static void a();
void b() {
    if (false)
        a();
}
```

Questions? Comments?