Rethinking the core OS in 2015

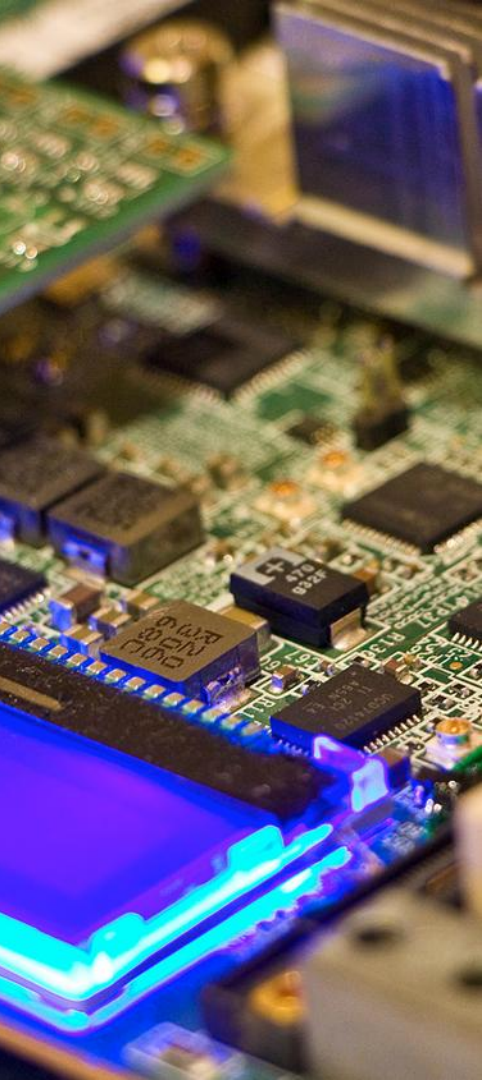
Are alternatives to gcc, libstdc++ and glibc viable yet? (And how do I use them?)

Presented by

Bernhard "Bero" Rosenkränzer

Date

Embedded Linux Conference Europe, 2015

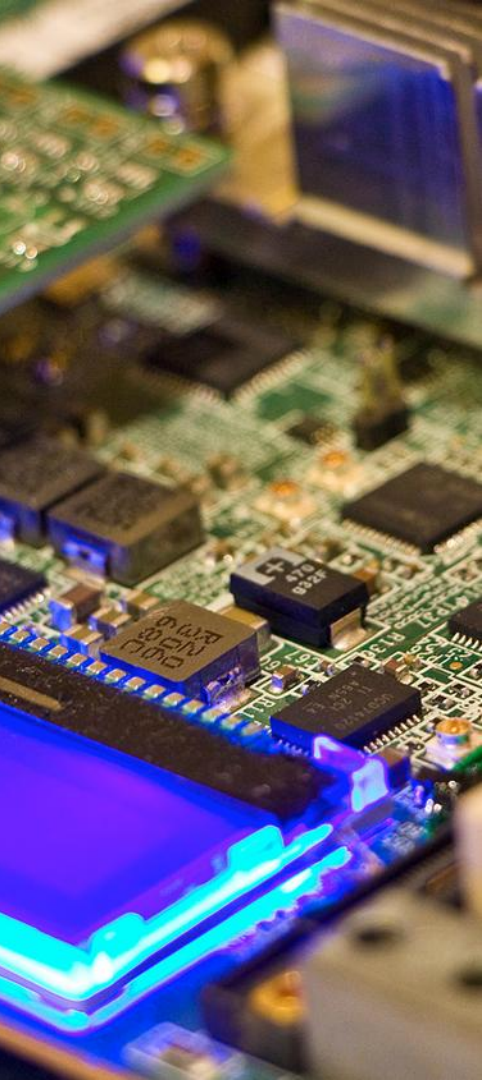


The traditional approach

Building a Linux system traditionally meant starting with building a core consisting of binutils, gcc and glibc (sometimes uClibc).

This is still a very viable approach, but now there are other options...

Linaro

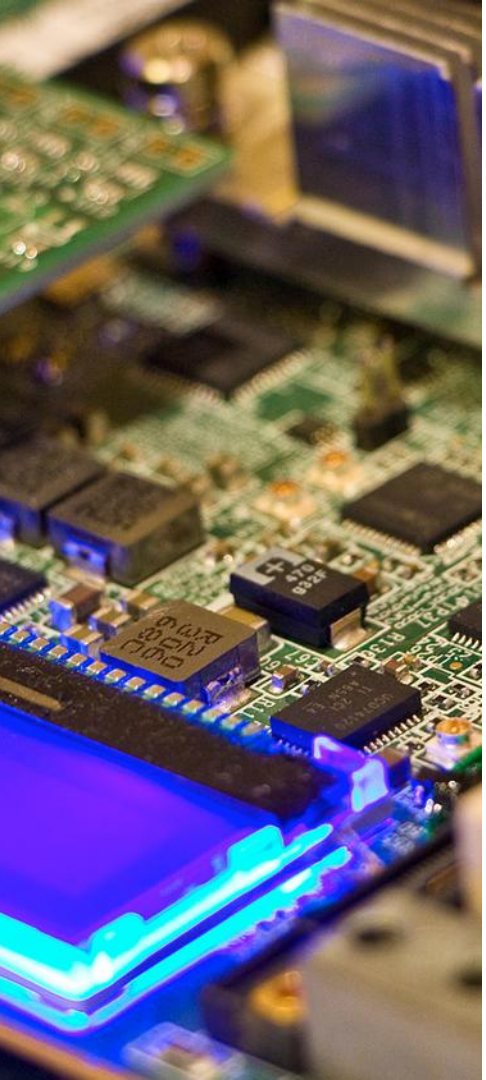


Binutils

Parts of binutils are still needed - in particular, a linker. (The traditional BFD ld can be replaced with gold, also part of binutils).

lld and mclinker are making some progress, but are not quite there yet.

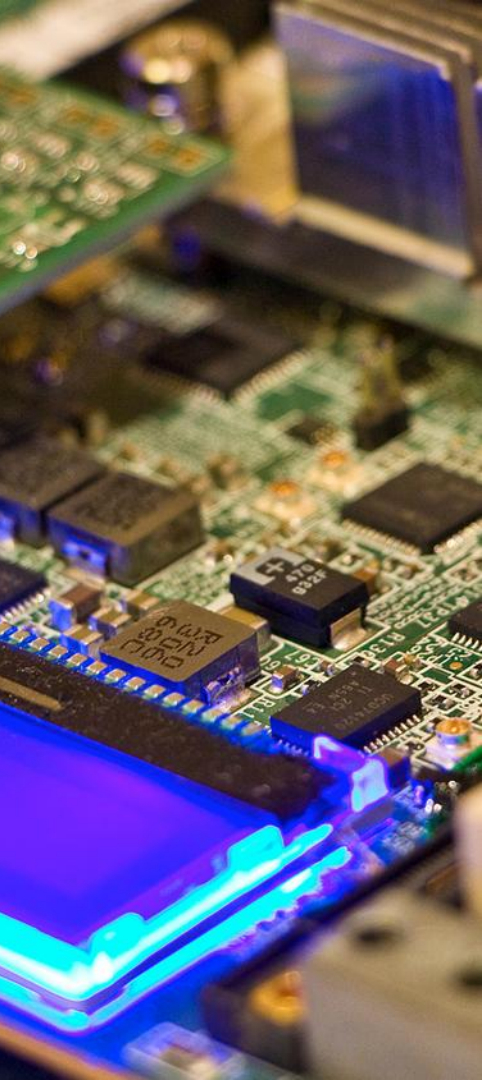
Linaro



Binutils

gas is sometimes needed because clang's integrated as doesn't support legacy constructs in common use (e. g. pre-unified syntax on ARM)

Linaro

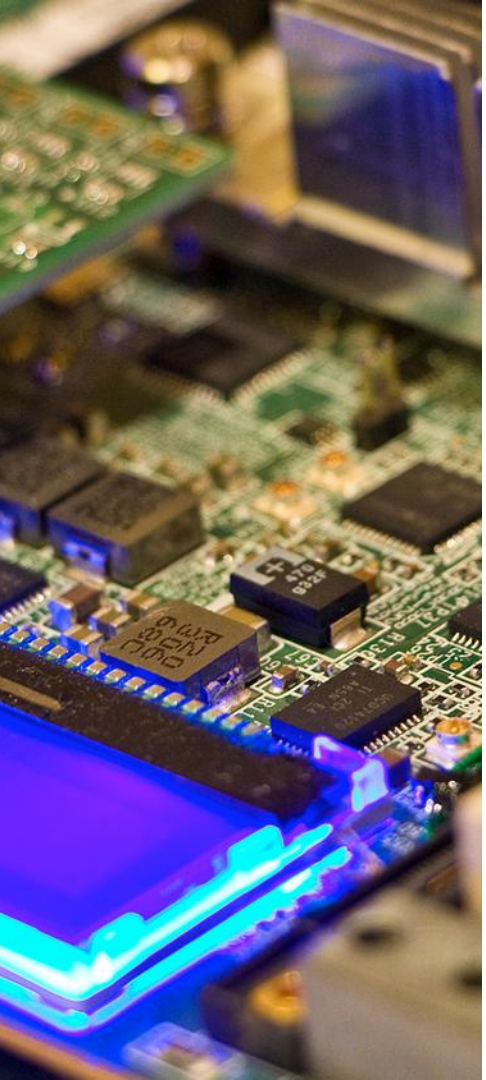


Binutils

Tools like nm need to get more complex: They should now deal with 3 types of input:

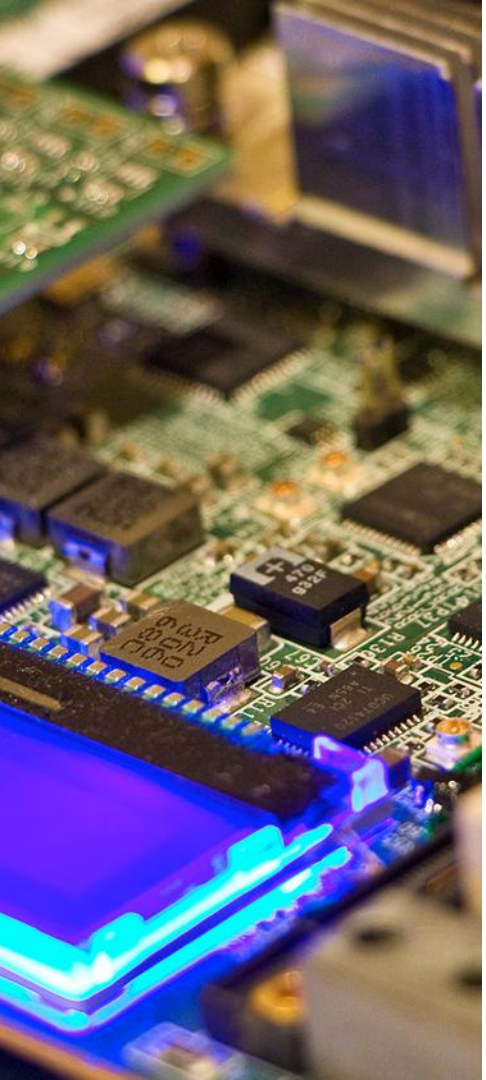
- regular object files
- LLVM bytecode (clang -flto)
- gcc interim code (gcc -flto)

Linaro



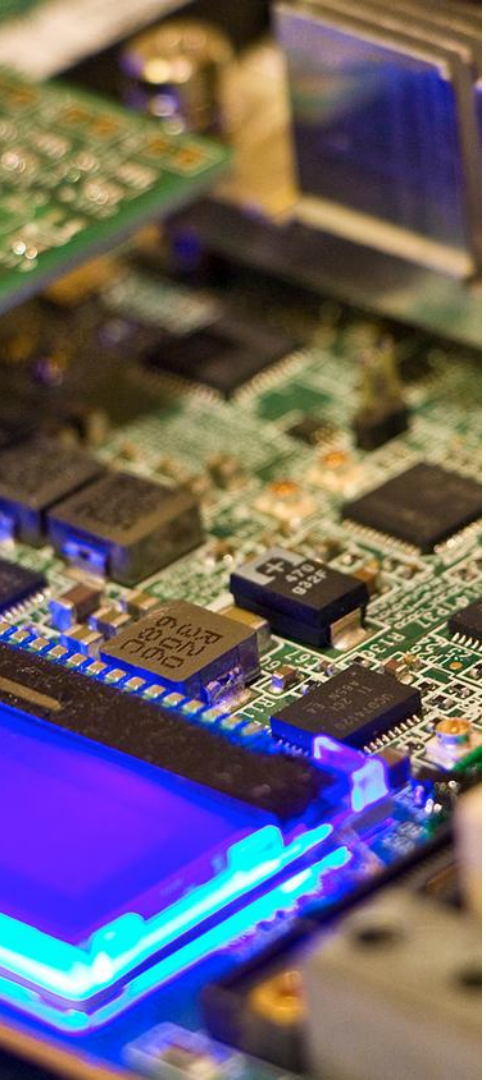
```
#!/bin/sh
REAL_NM=binutils-nm
PARENT="`readlink /proc/$PPID/exe`"
WRAPPED=false
# If /proc isn't mounted, let's do the least evil thing we can
if [ -z "$PARENT" ]; then
    WRAPPED=true
elif echo $PARENT |grep -qE -- '-nm$'; then
    # If we're being called by gcc-nm or llvm-nm, we're already
    # wrapped (and need to make sure we don't call ourselves recursively)
    WRAPPED=true
elif echo $PARENT |grep -qE -- 'qemu'; then
    # Fun... We're running inside qemu binfmt_misc emulation,
    # so we have to determine our parent the evil and less
    # reliable way...
    if grep -qP -- '-nm\x00' /proc/$PPID/cmdline; then
        WRAPPED=true
    fi
fi
```

Linaro



```
# If we're being called by gcc-nm or llvm-nm, we're
# already wrapped...
if ! $WRAPPED; then
    for i in "$@"; do
        [ "`echo $i |cut -b1`" = "-" ] && continue
        if echo $i |grep -qE '\.(o|a)$' && [ -e $i ]; then
            if LC_ALL=C gcc-nm $i 2>&1 |grep -q "File format not
recognized"; then
                which llvm-nm &>/dev/null && REAL_NM=llvm-nm
                break
            fi
        fi
    done
    if [ "$REAL_NM" = "binutils-nm" ] && which gcc-nm &>/dev/null; then
        REAL_NM=gcc-nm
    fi
fi
exec $REAL_NM "$@"
```

Linaro

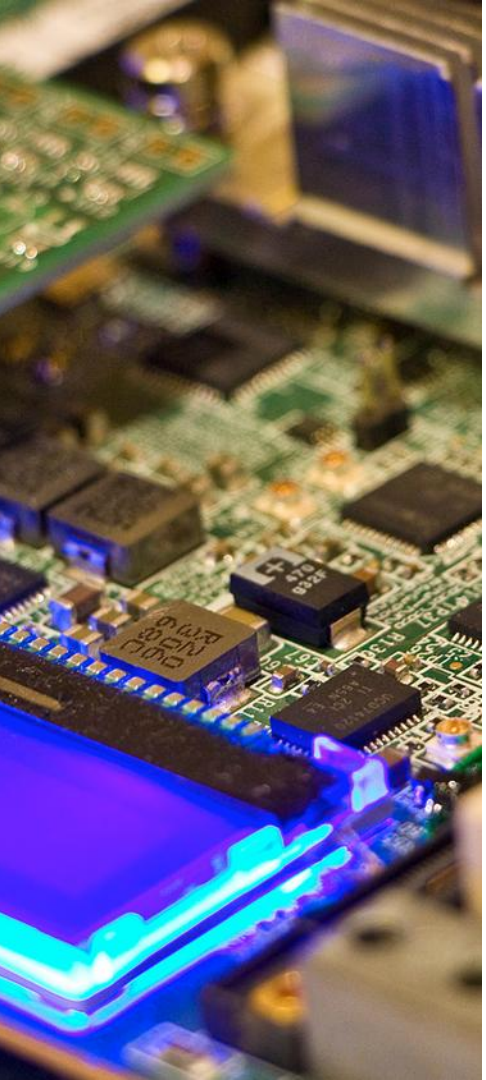


gcc

gcc can, for the most part, be replaced with clang these days.

OpenMandriva switched to clang as its primary compiler last year. OpenMandriva 3 (soon to be released) is almost fully built with clang 3.7.

Linaro

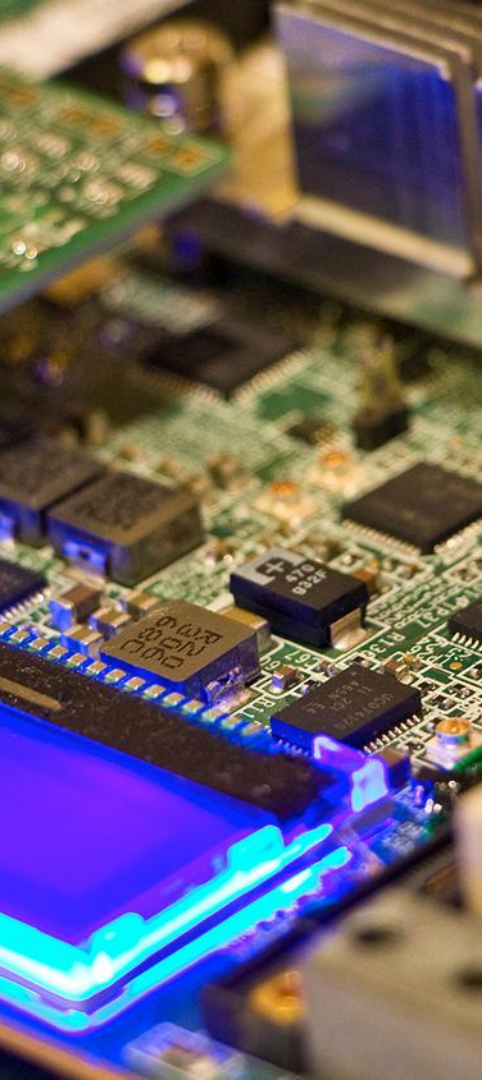


gcc

The transition was unproblematic, most packages that failed failed due to bad code or use of nonstandard gcc extensions.

We force some packages to build with CC=gcc CXX=g++.

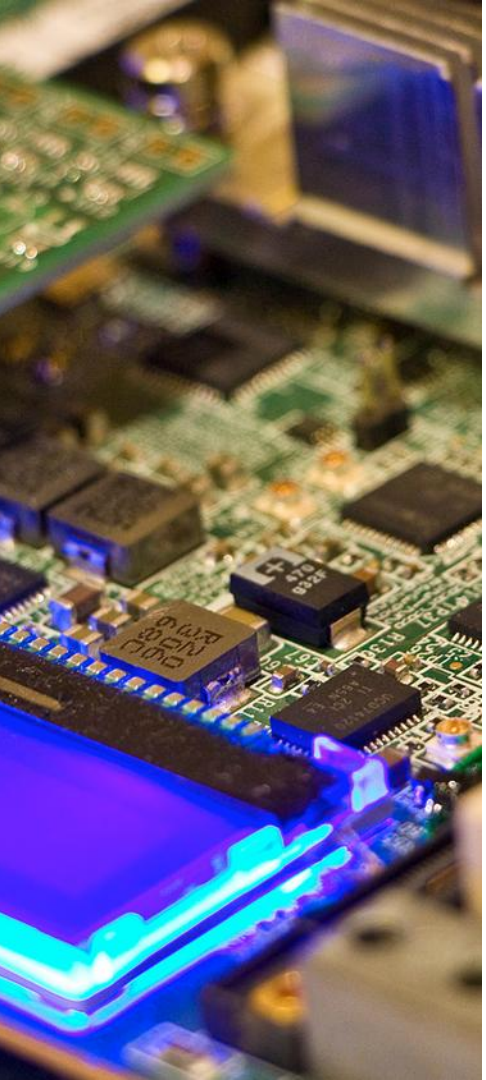
Linaro



gcc

We still need to build gcc even if we don’t want to use it as a compiler though: We need libgcc, libgcc_s, libatomic and friends (and potentially libstdc++)

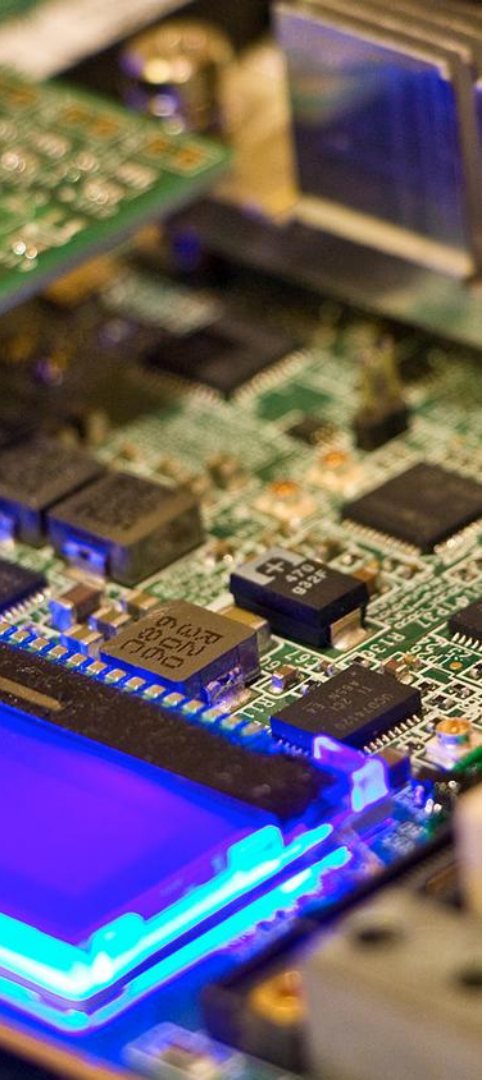
Linaro



gcc

clang's __GNUC__ macro definitions are too conservative, claiming to be gcc 4.2.1, causing code that checks __GNUC__ to leave out optimizations (sometimes, it will even fail to build because of assumptions about lack of standards compliance in what seems to be an old version of gcc)

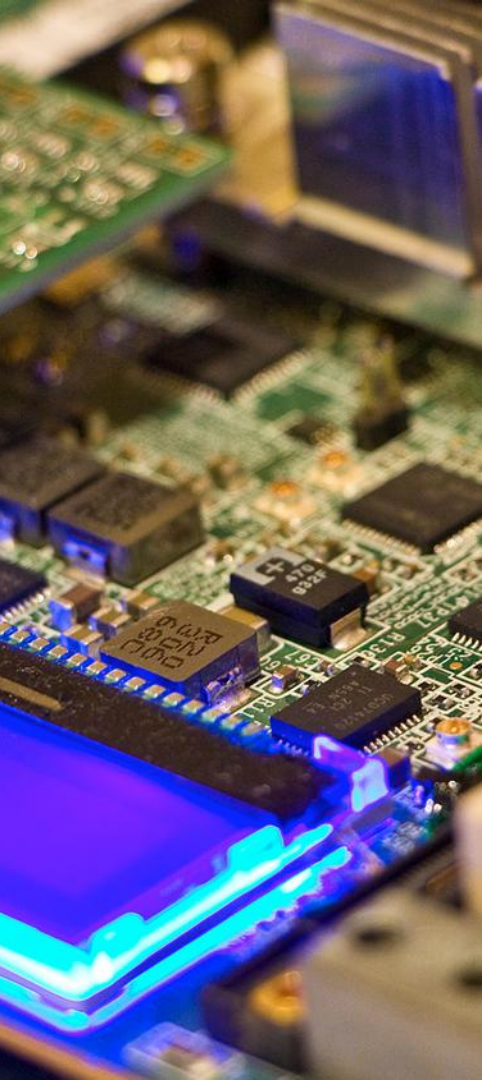
Patching it to say 4.9 produces better code.
(real fix is to check for features instead of compiler versions - but let's be realistic…)



Things to avoid for compatibility

- Nested functions
- Variable length arrays in structs
- Variable length arrays of non-POD types
- Empty structs
- Array subscripts of type “char” (`value['0']=0;`)
- Reserved words (“_Nullable” defined by both clang and Qt)

Linaro

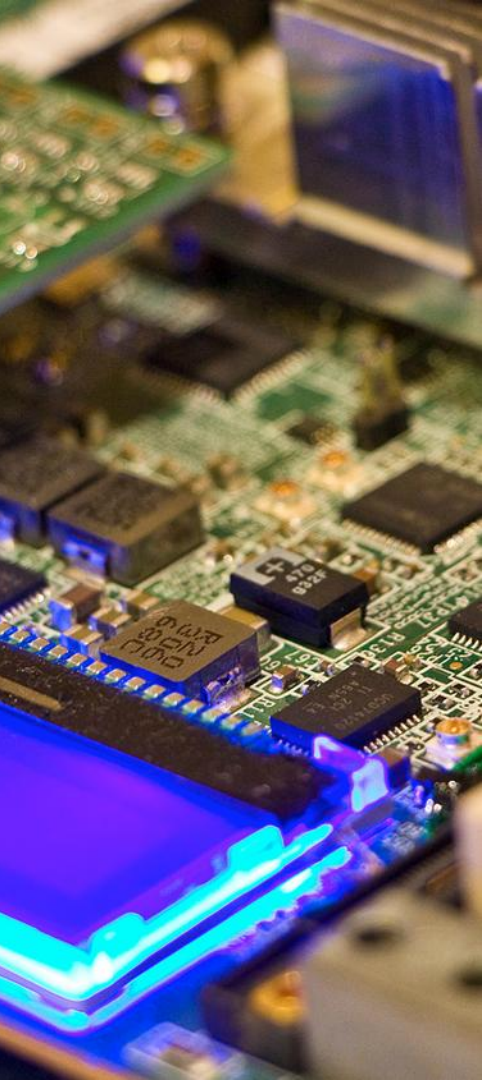


Things to avoid for compatibility

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

```
static void a();
void b() {
    if (0)
        a();
}
```

Linaro



Things to avoid for compatibility

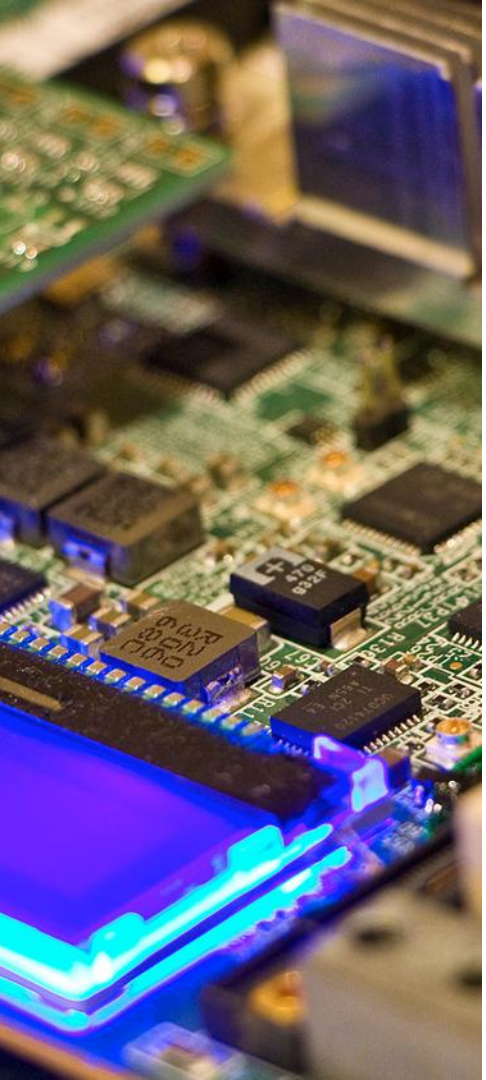
- gcc 5.x's changed libstdc++ ABI

  https://llvm.org/bugs/show_bug.cgi?id=23529

  - clang doesn't implement gcc's __attribute__ ((abi_tag)), needed by gcc 5.x's libstdc++ built in new ABI mode
  - build gcc with `--with-default-libstdcxx-abi=gcc4-compatible`

    for now if both compilers need to coexist (and you want libstdc++ instead of libc++)

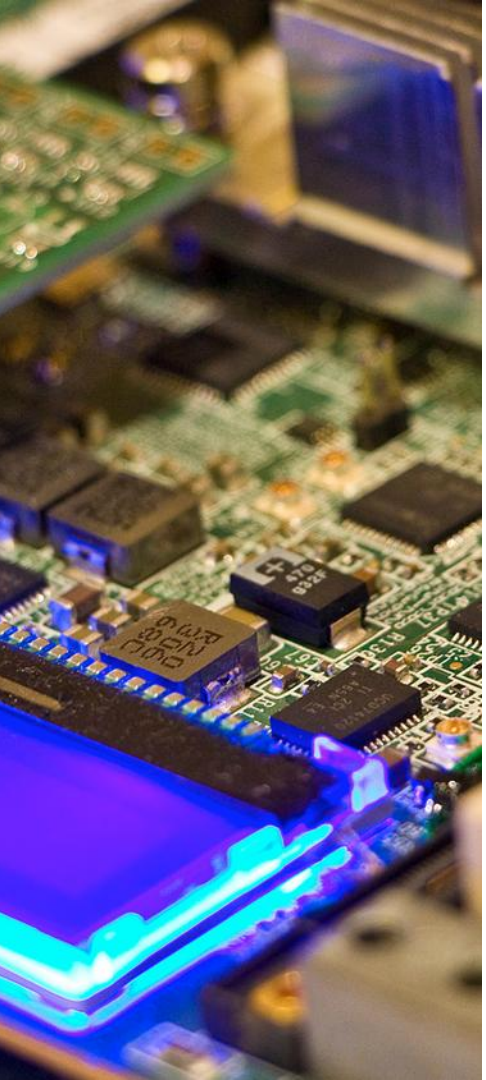
Linaro



Things to avoid for compatibility

- C89-isms and C++98-isms, e.g. changed meaning of “`extern inline`”

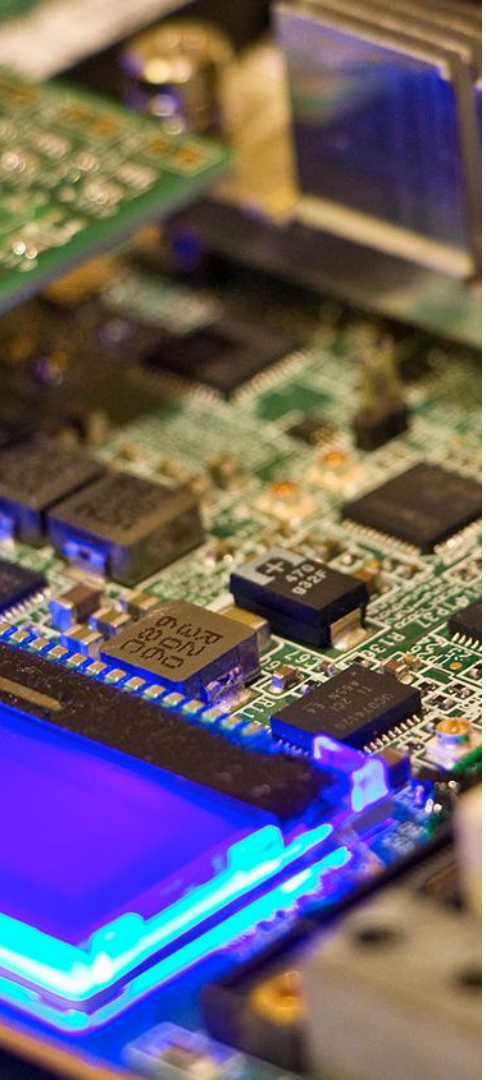
Linaro



Interesting bugs found by clang

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

Linaro

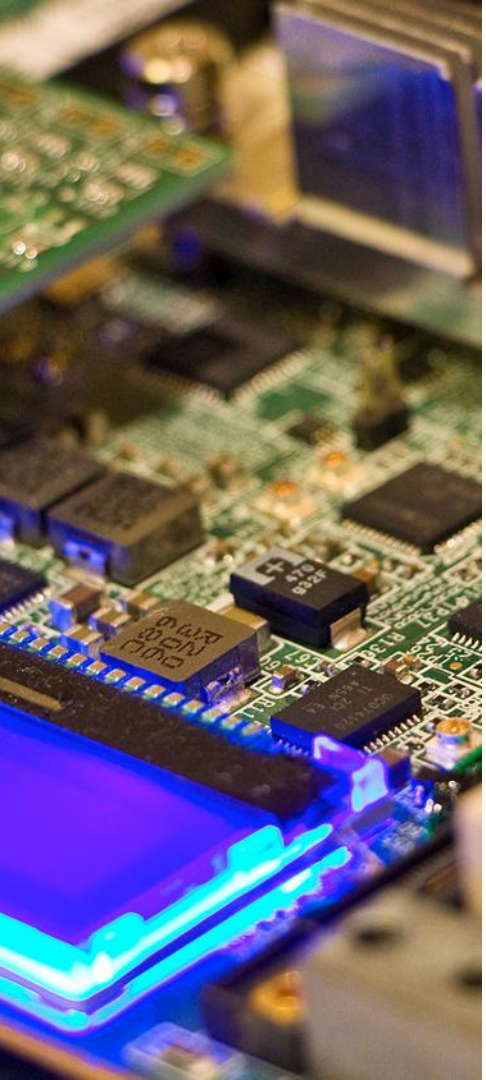


Interesting bugs found by clang

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

size of a pointer - not quite 30

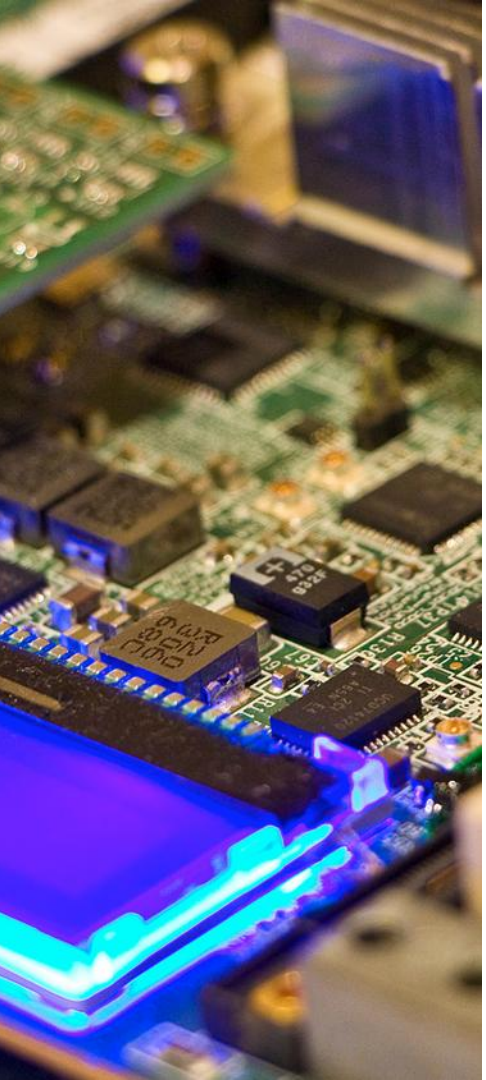
Linaro



Interesting bugs found by clang

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

Linaro

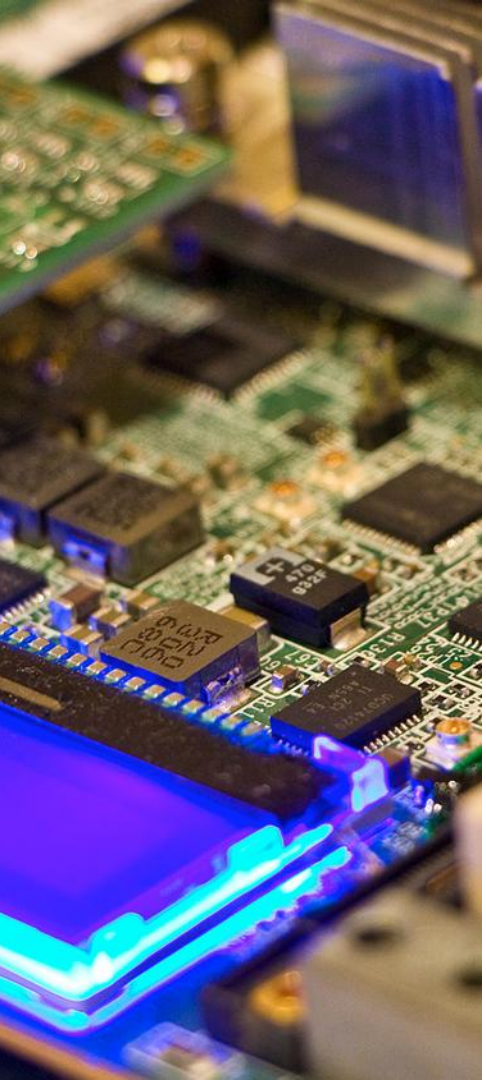


Interesting bugs found by clang

```
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]

Linaro



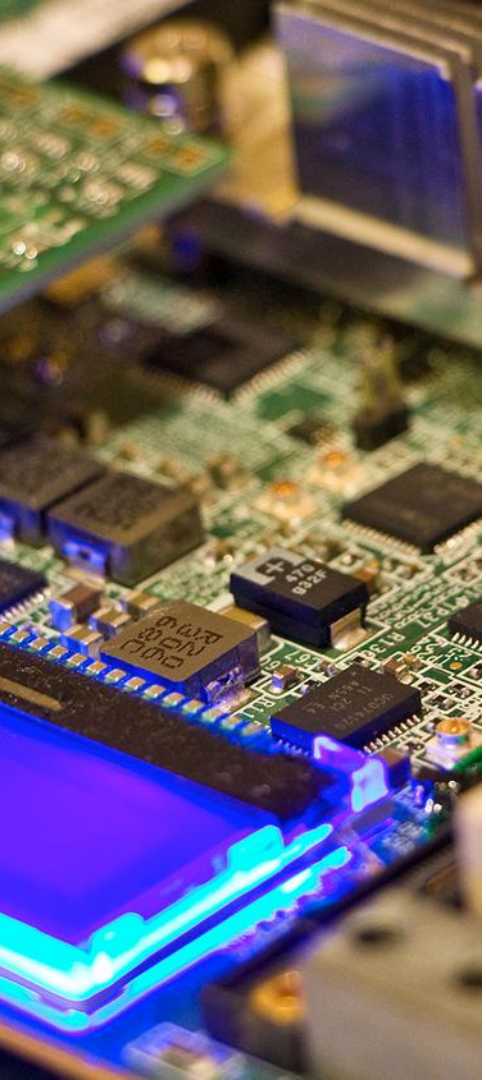
clang vs. gcc?

Both compilers are good. Performance of compiled code is similar.

clang:

- tends to be faster at compiling
- is easier to work on (more readable code)
- error messages tend to be more readable
- has an edge in targeting GPUs

Linaro

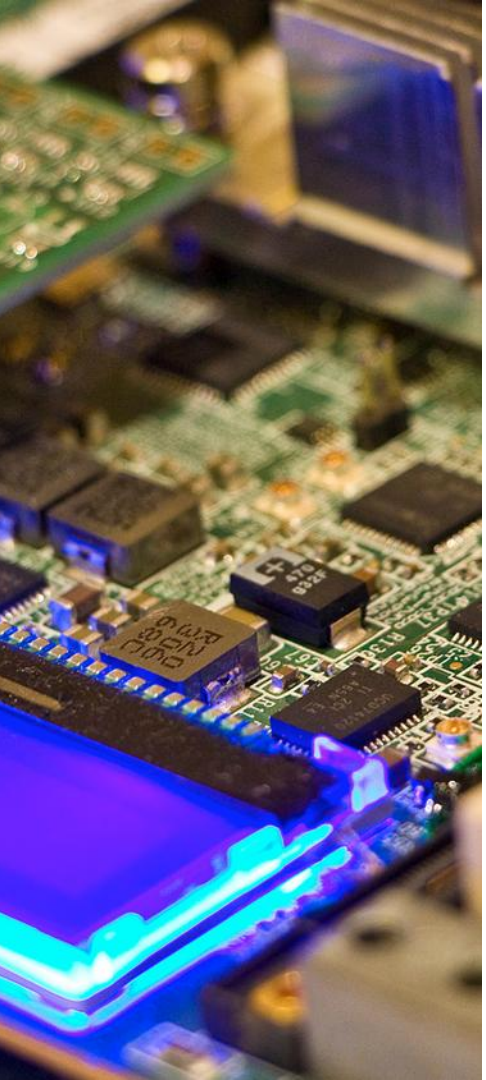


clang vs. gcc?

gcc:

- has been around longer - has had more time to learn about special cases and how to optimize them
- currently better at OpenMP
- supports more targets (most targets supported only by gcc are obsolete-ish though)

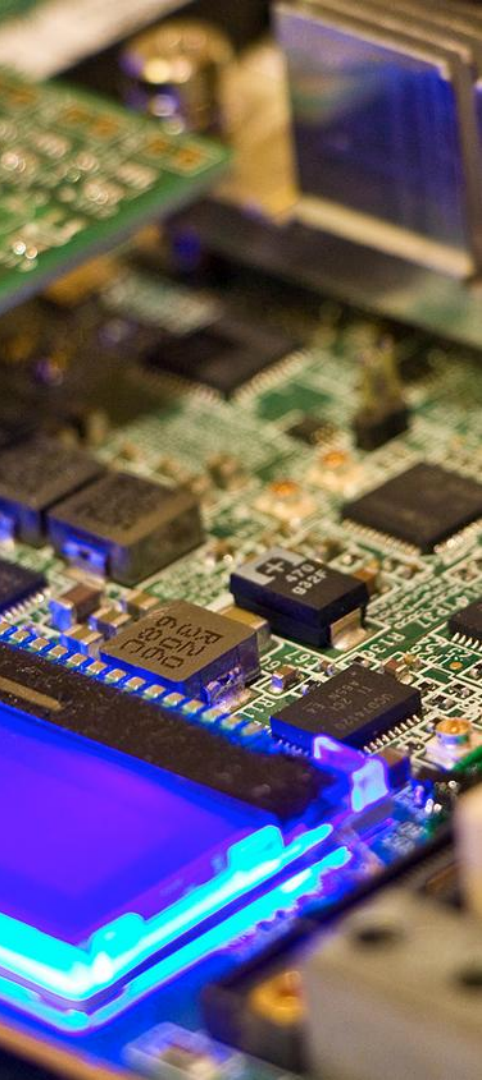
Linaro



clang vs. gcc?

It’s generally a good idea to try compiling your code with both compilers - either one may catch a bug the other didn’t see.

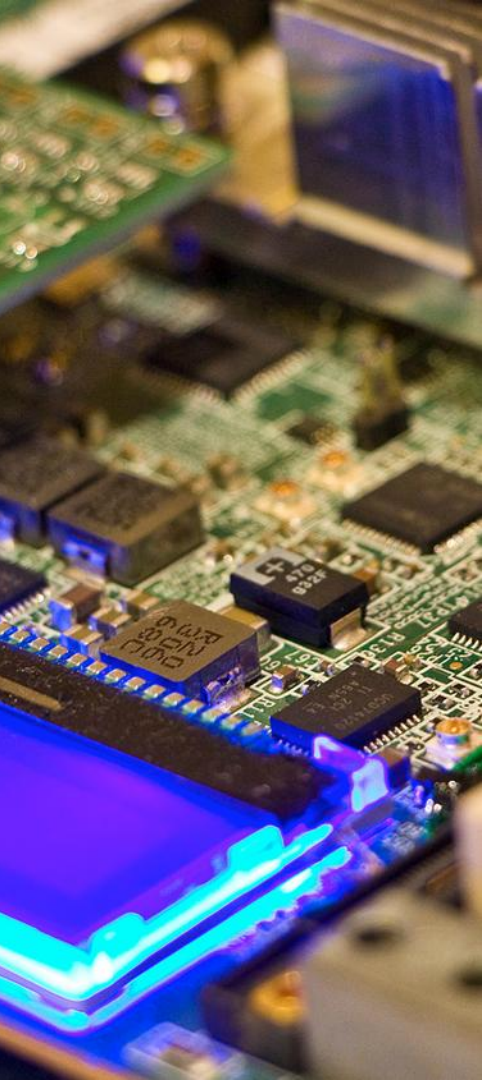
Linaro



glibc

musl is at a point where using it as the sole system libc is viable (if you don’t care about binary compatibility with other distributions).

Linaro



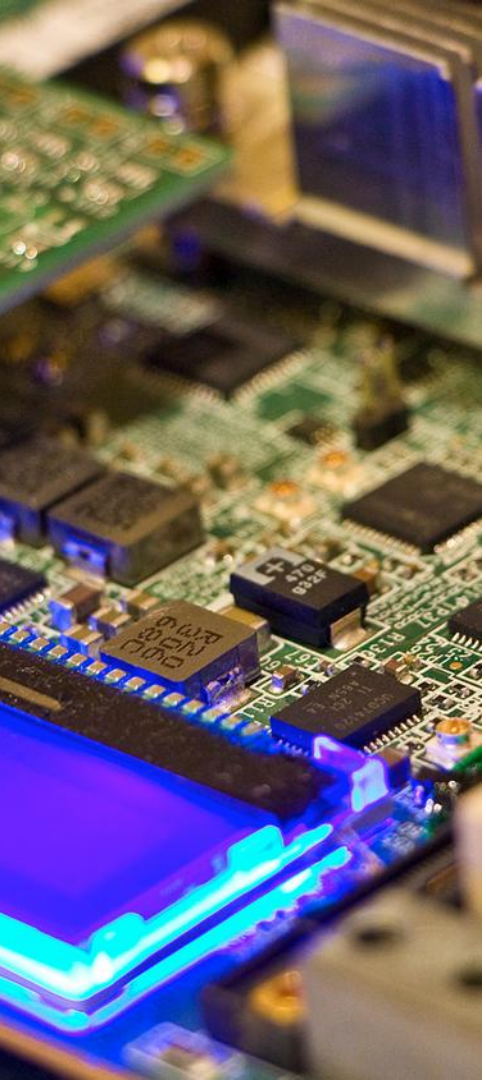
glibc

clang currently doesn’t support musl, but that’s fixable. Patches at https://abf.io/openmandriva/llvm

Patches are needed mostly to change the path to the dynamic linker.

gcc trunk supports musl.

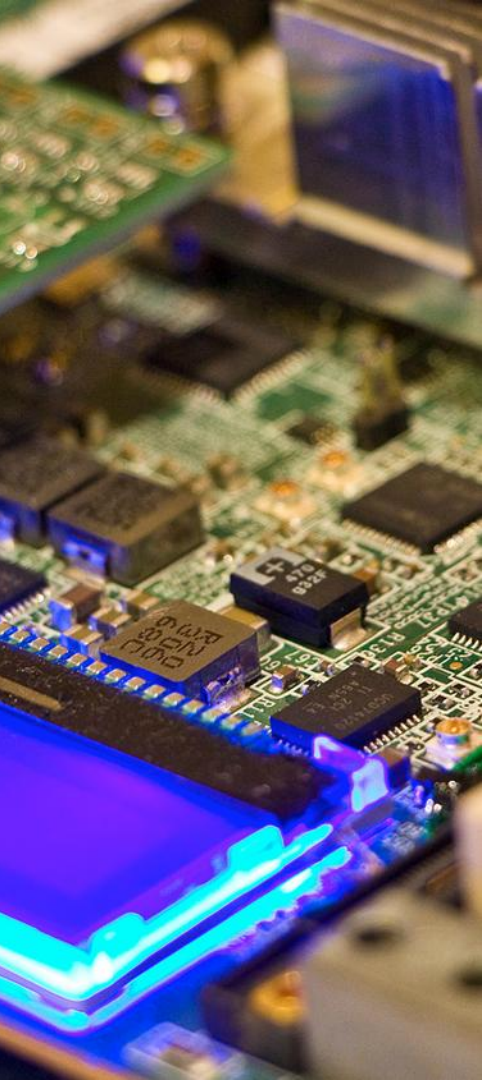
Linaro



glibc

Android’s Bionic is becoming a viable alternative as well - while it started its life as a small but not very optimized libc that does only what Android needs, it is highly optimized (especially for ARM) and nearly complete these days.
Still lacks SysV shared memory.

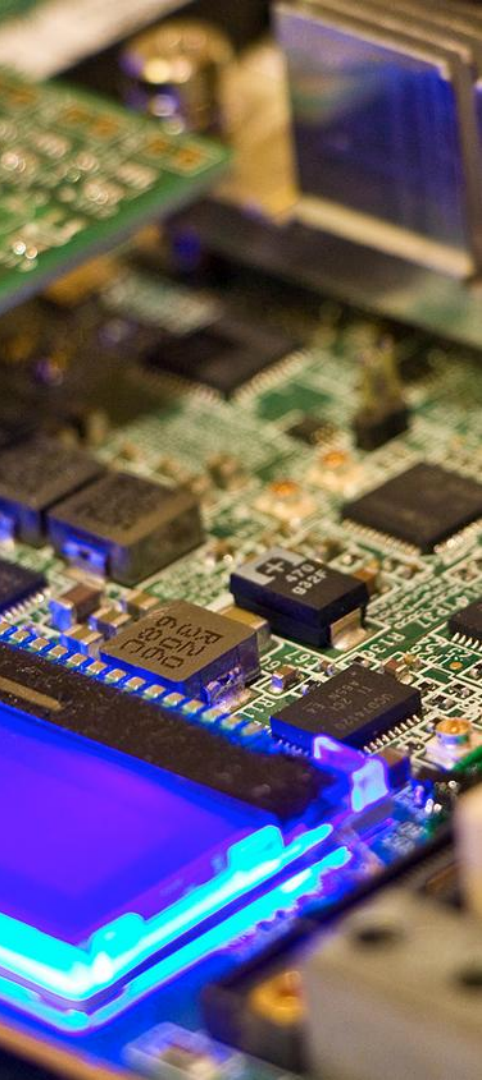
Linaro



Things to do for compatibility

- Make sure to #include the headers the code needs instead of cutting corners and e.g. omitting
  #include <string.h>
  because your favorite libc's stdlib.h happens to #include <string.h> and you're #including that

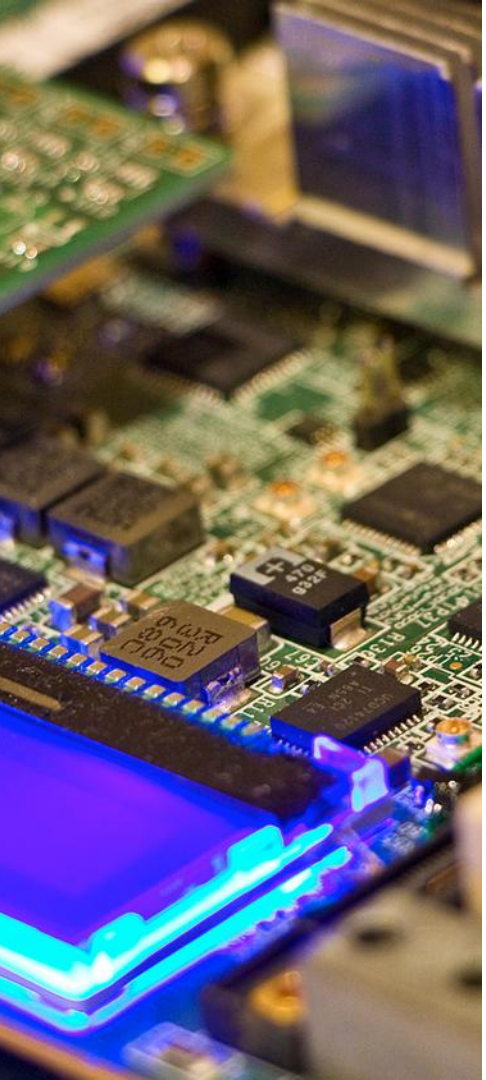
Linaro



Things to do for compatibility

- Avoid using deprecated APIs - they tend to be the lowest priority for new libcs
- Don’t assume _GNU_SOURCE, _BSD_SOURCE and friends default to what you’re used to
- Don’t assume __linux__ and __GLIBC__ are the same thing

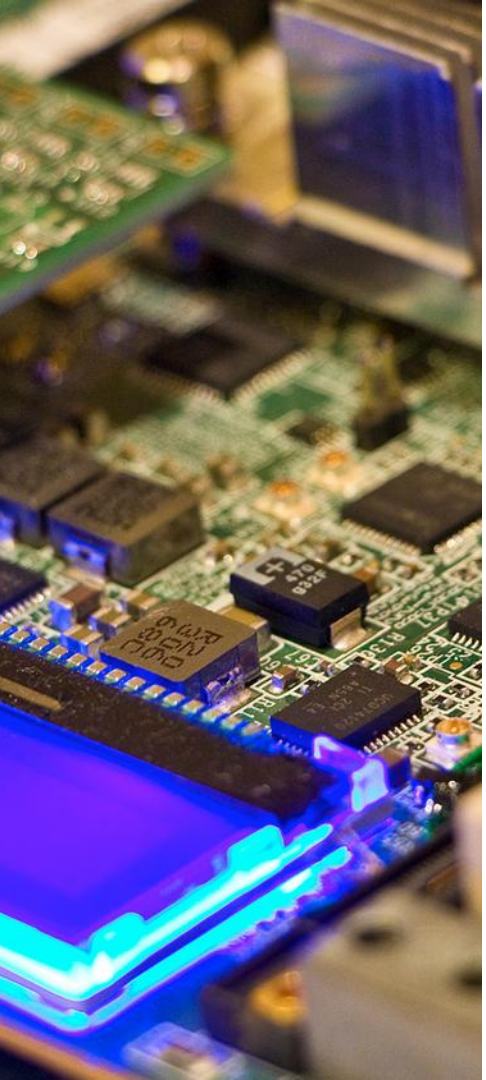
Linaro



Things to do for compatibility

- Some locale-aware variants of libc routines (isalnum_l etc.) may not exist (yet?).

Linaro

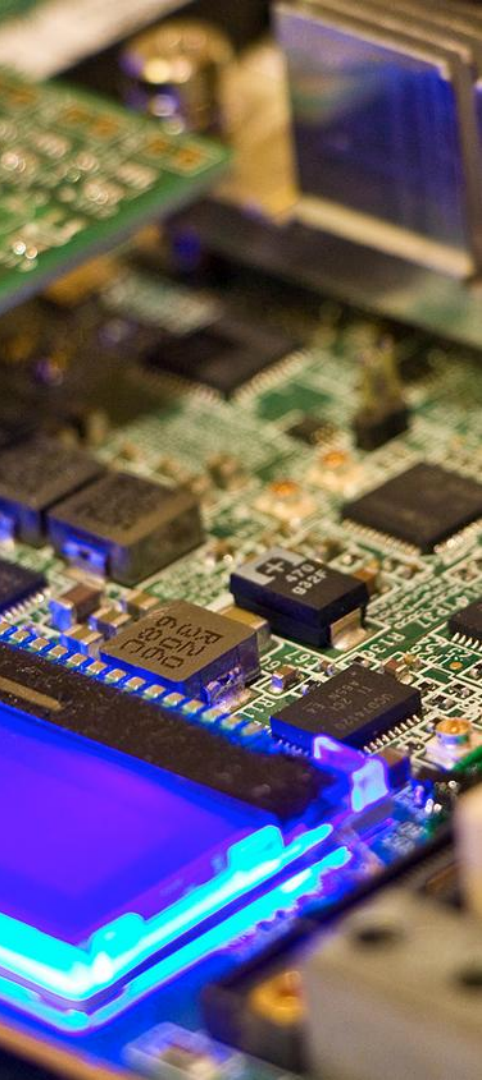


libc comparison

glibc:

- most standards compliant
- supports most targets
- binary compatibility with a wide range of systems

Linaro

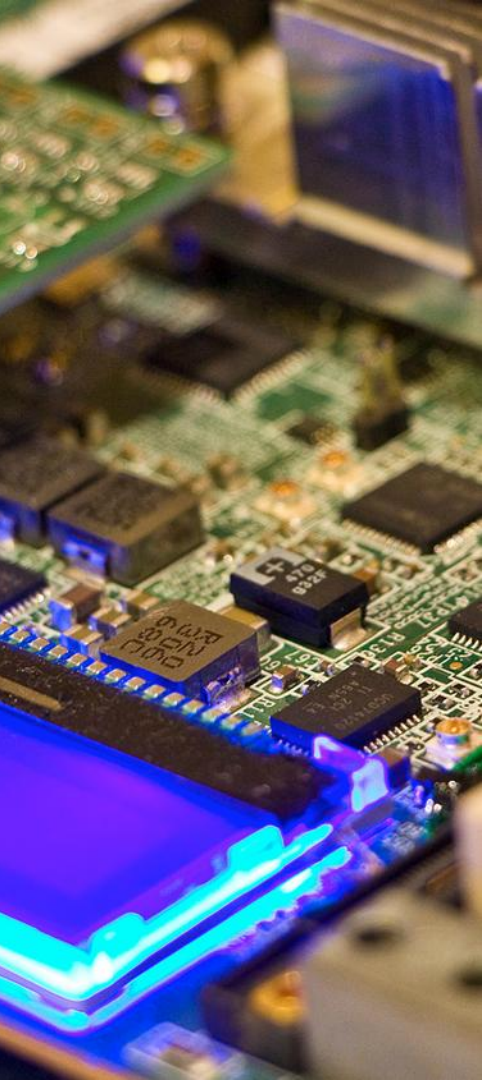


libc comparison

musl:

- small memory footprint
- fast
- complete enough for most uses
- doesn’t carry around a lot of cruft (which is both a good and a bad thing)

Linaro

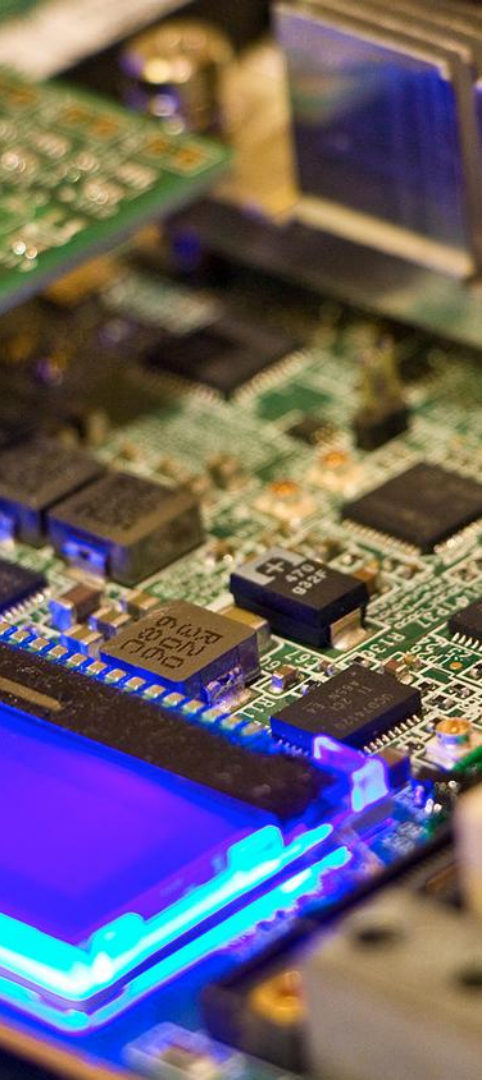


libc comparison

bionic:

- small memory footprint
- fast
- designed for Android’s needs, you may need to add some functions from another libc

Linaro

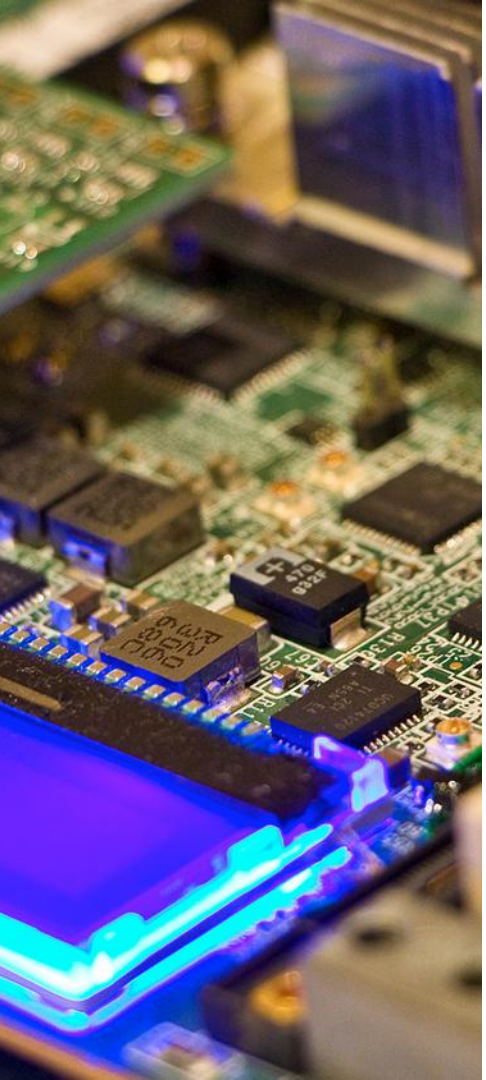


libc comparison

uclibc:

- small memory footprint
- highly customizable build system allows stripping out unneeded bits
- last official release in 2012 (may want to check uclibc-ng)
- supports many older CPU targets, but not aarch64

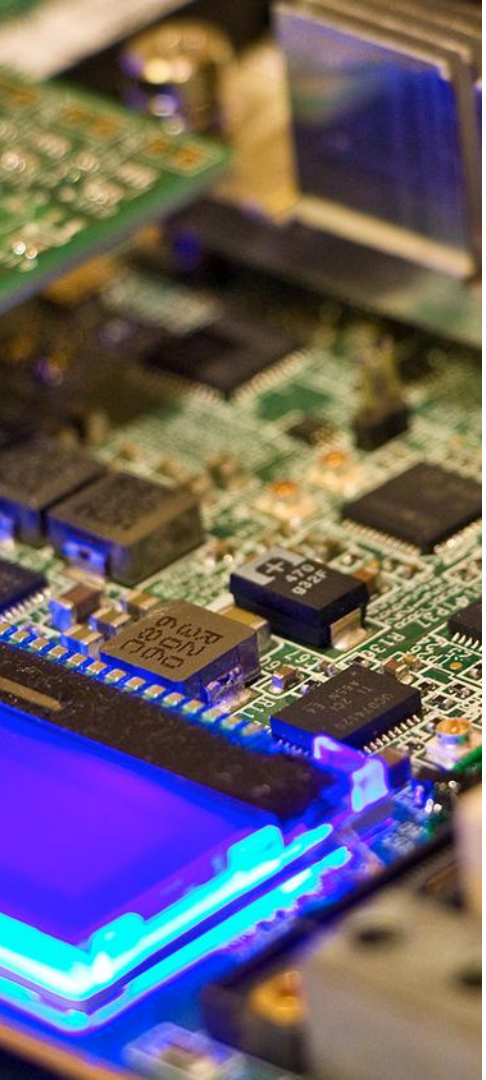
Linaro



libstdc++

LLVM’s libc++ is generally ready to replace libstdc++ where binary compatibility is not a concern.

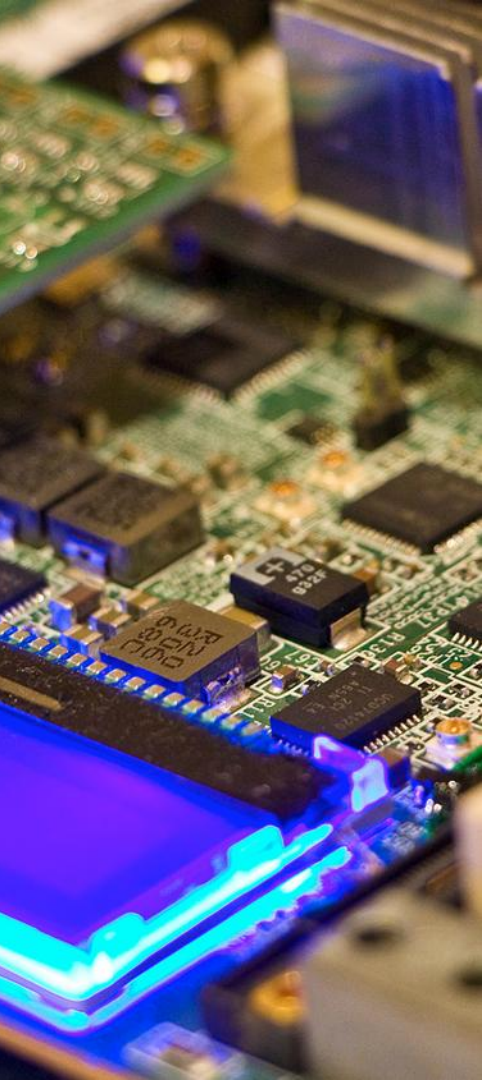
Linaro



libstdc++

Unfortunately, binary compatibility is a concern for many uses -- and while libstdc++ and libc++ can coexist, problems start showing up with other libraries (Qt linked to libc++, binary-only application uses Qt and links to libstdc++ → crash)

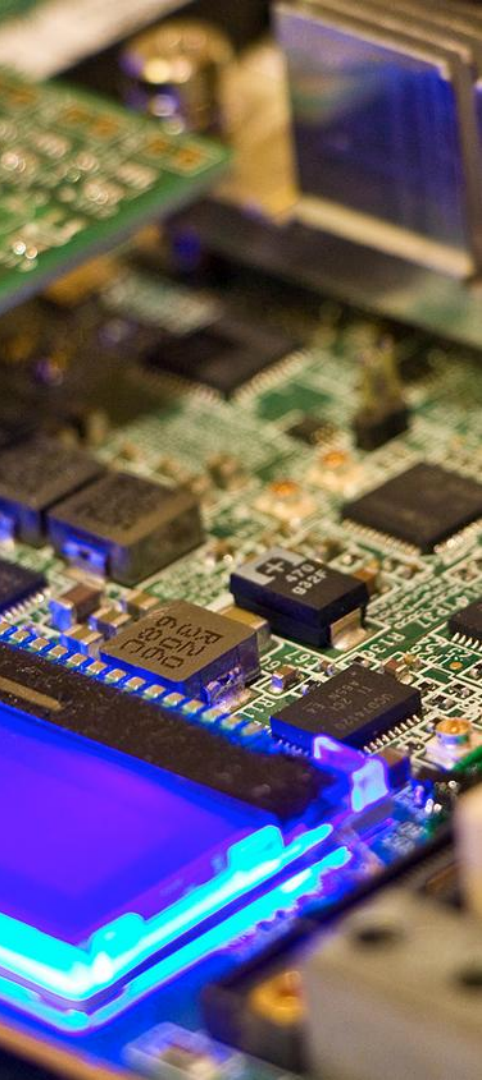
Linaro



Things to do for compatibility

- Code to the C++11, or better yet, C++14 and C++1z standards. libc++'s support for older standards is limited.
- Don't assume STL headers include other headers just because libstdc++ does.

Linaro

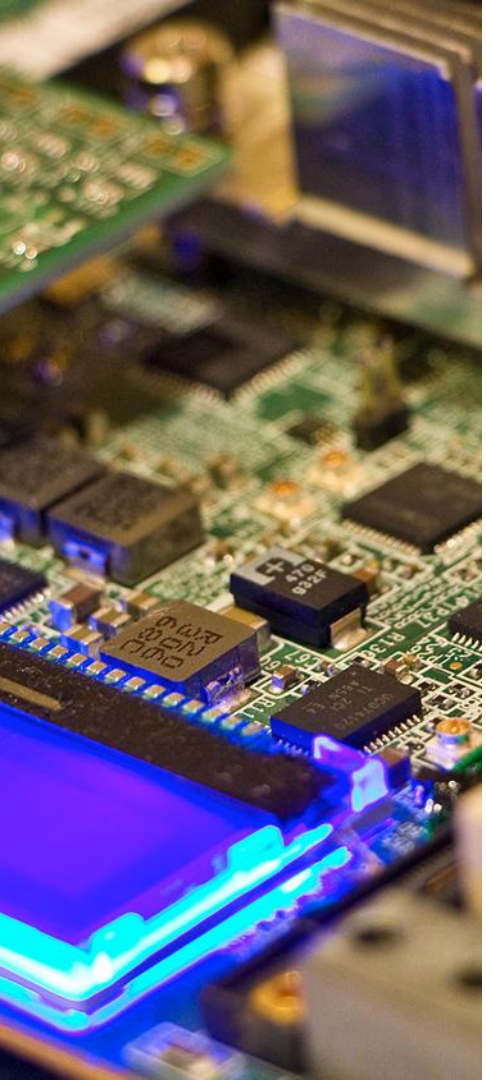


libstdc++

libc++ is often the better choice if binary compatibility is not a concern -- roughly 50% space saved, full C++14 support.

Android is doing the right thing by switching to libc++ (from STLport)

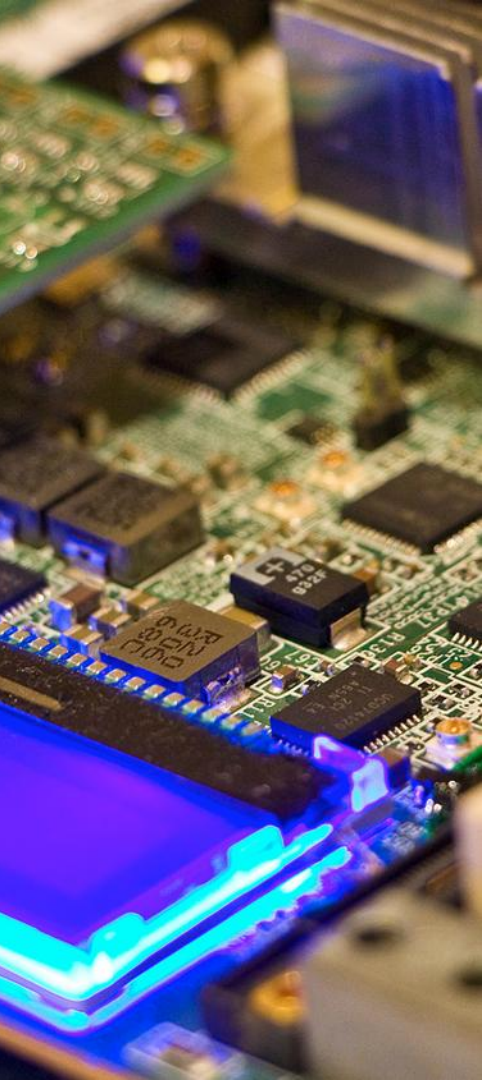
Linaro



libstdc++

libc++ is tested almost exclusively with clang - worth considering when picking the compiler or STL implementation

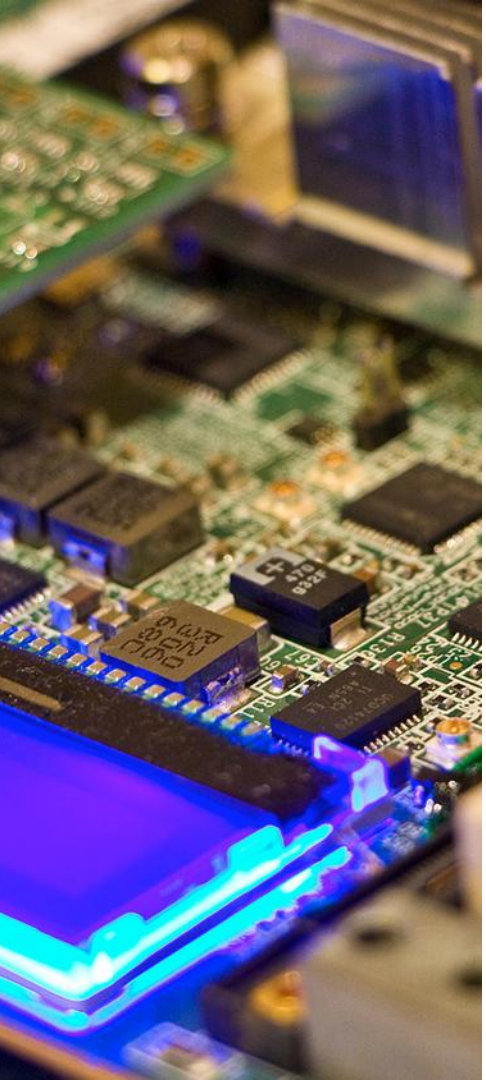
Linaro



crosscompiling

Switching to an LLVM/clang based toolchain is interesting for crosscompiling - a regular clang already has crosscompiling support built in, no need to build a fresh compiler for every new target

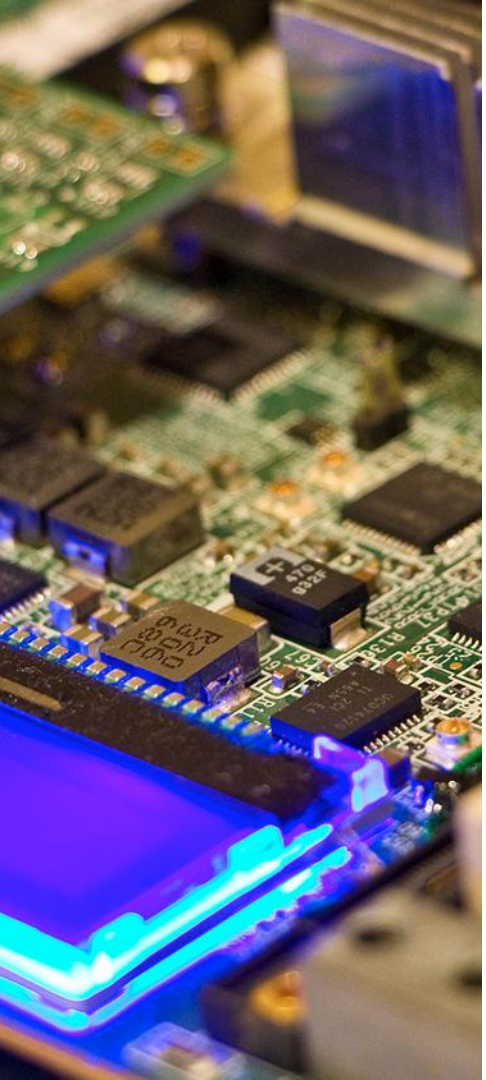
Linaro



crosscompiling

--sysroot in clang needs work: Still sees host system headers.
Wrapper scripts can be used to work around this.

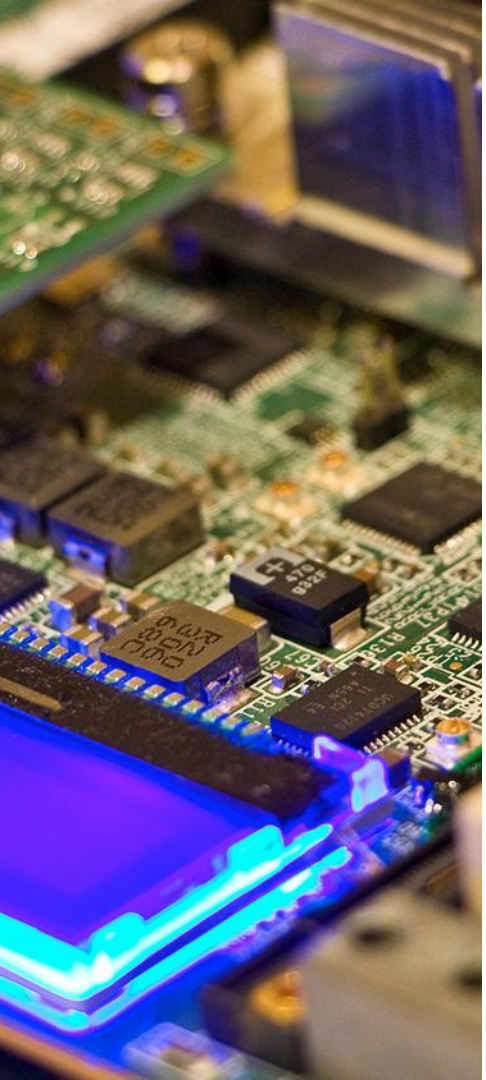
Linaro



crosscompiling

```
Lopts="-L$SYSROOT/usr/lib -L$SYSROOT/lib"
# Warnings like "argument unused during compilation"
# can break configure scripts
for i in "$@"; do
    if [ "$i" = "-E" -o "$i" = "-c" ]; then
        Lopts=""
        break
    fi
done
exec clang -target $TARGET \
    --sysroot=$SYSROOT -nostdinc \
    -isysroot $SYSROOT \
    -isystem $SYSROOT/usr/include \
    $Lopts \
    -ccc-gcc-name $TARGET-gcc "$@"
```

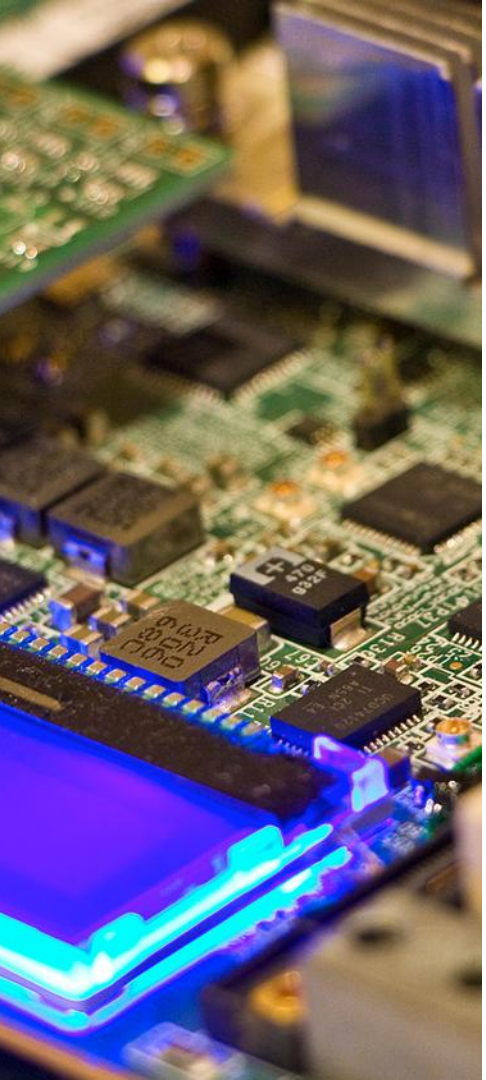
Linaro



crosscompiling

Automated toolchain and core system bootstrapping being worked on:
https://abf.io/openmandriva/crossbuild/blob/master/build-clang-musl.sh

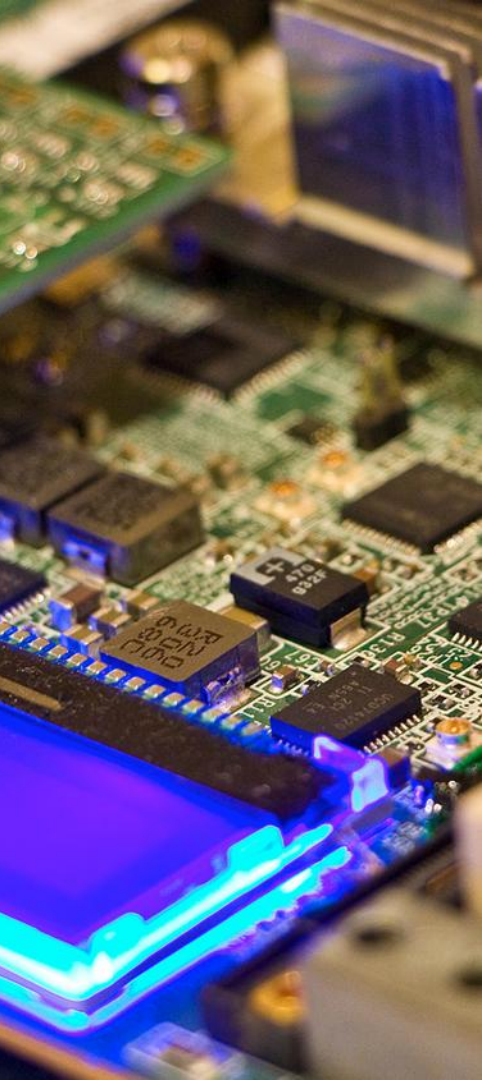
Linaro



crosscompiling

The idea: pass a target triplet, get a ready to use root filesystem that is ready to compile other code (e.g. applications with bogus Makefiles that aren’t ready for crosscompiling)

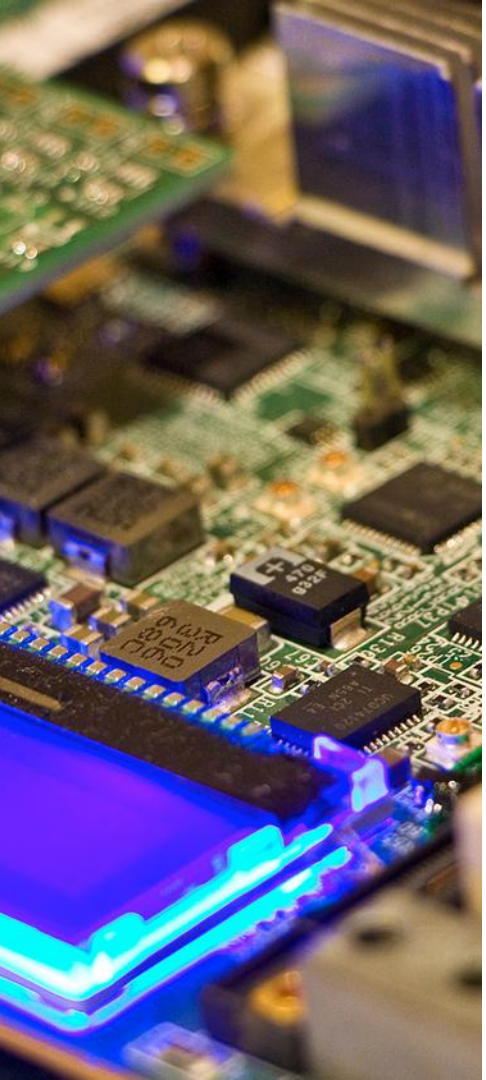
Linaro



crosscompiling

But of course… compiling on target devices may be slow, so can we crosscompile some more?

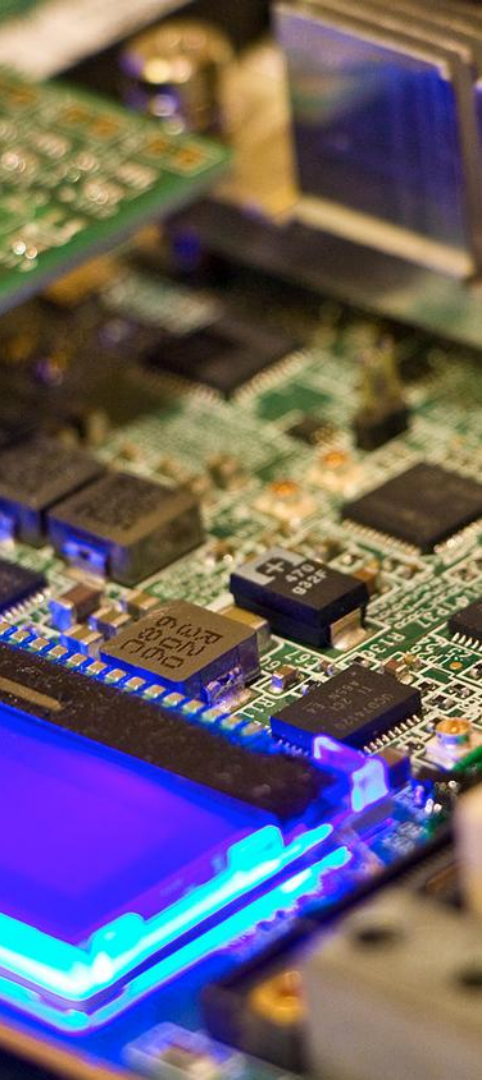
Linaro



reusing existing packages

Fortunately, well done rpm packages have been using macros for invoking autoconf-generated scripts and cmake for a while...
(%configure, %cmake)

Linaro



reusing existing packages

… so making packages ready for crosscompiling is often just a matter of making %configure and %cmake do the right thing:

- Add `--host=... --target=...` to %configure
- Add `-DCMAKE_TOOLCHAIN_FILE=... -DCMAKE_CROSS_COMPILING:BOOL=ON` to %cmake

Linaro

Questions? Comments?

bero@linaro.org