Report ACA Project: Linux on Patmos

Milos Golijanin, Stephan Felber

January 23, 2021

Abstract

The t-crest PATMOS processor is a quite mature time predictable processor. It already delivers a C toolchain and means of measuring the worst case execution time. Running Linux on PATMOS would further enhance the field of applicability and ease the start for new developers as they do not need to develop on the processor directly. Unfortunately, we believe that the current version of the toolchain does not support Linux as of yet. This report serves as a starting point for the next group once `patmos-clang` reaches version `>= 10.0.1`, so that they do not have to start from zero.

1 Introduction

We stopped working on this because we found that `patmos-clang v3.4` does not handle the `_Generic` statement correctly. The following program compiles on the recent `clang` and `gcc`, but not on `patmos-clang v3.4`. One can find more information about this here [1].

```
#include <stdio.h>
#define typestring(x)
_Generic(*(x),    \
    char:         "char",
\
    int:          "int")

int main(void) {
    char const a = 'a'; int i = 0;
    printf("%s, %s\n", \
      typestring(&a), typestring(&i));
}
```

This should be fixed once `patmos-clang` is bumped to a more recent version. Still, we strongly recommend to first implement GDB support for the patmos simulator before looking into this project again! This is also available as a project.

This section serves as a short overview how the porting process should work. In general, all the platform specific code for Linux lives in the folder `arch/patmos`. In our case, we also modified the top level `Makefile` to support llvm cross compilation.

Apart from the rather standard hardware abstraction layer implementation Linux features a so called *device tree* (DT). The DT was developed to avoid having to recompile the kernel for every slightly different hardware setup. Instead, there exists a device tree binary (DTB) for every desired hardware variant and the bootloader passes the kernel the correct DTB.

We set the first goal to be any sort of output via `printk` (the classic hello world). In order to achieve this, we chose to implement the very low level debug console.

2 Related Work

Despite Linux beeing ported to a multitude of architectures, the process is not very well documented. This might be due to the fact that not that many new architectures feature a completely new instruction set architecture, but rather implement some subset of an already existing ISA which makes it a lot easier to adapt and already existing architecture.

[1] `https://stackoverflow.com/questions/18857056/c11-generic-how-to-deal-with-string-literals/55392218#55392218`

Despite that, there are a couple of officially supported architectures with no MMU support in the mainline (`arch/c6x` and `arch/microblaze` for example).

3 Setup

We started working on the Linux mainline version `5.10-rc3`. We setup our build environment according to [2] with the following commands:

```
mkdir ~/t-crest
cd ~/t-crest
git clone https://github.com/t-crest/patmos-misc.git
cd patmos-misc
./build.sh
```

Then we added `~/t-crest/patmos-misc/local/bin` to our `$PATH` variable. In order to follow the coding style for cross compilation in Linux, we created links for a couple of patmos llvm binaries:

```
patmos-clang -> patmos-clang-3.4*
patmos-clang++ -> patmos-clang*
patmos-clang-cl -> patmos-clang*
patmos-ld.lld -> patmos-ld*
patmos-llvm-objcopy -> patmos-objcopy*
patmos-llvm-ranlib -> patmos-llvm-ar*
patmos-llvm-readelf -> patmos-readelf*
patmos-llvm-strip -> patmos-strip*
```

4 .config

Landley introduced [3] a very useful `allnoconfig` into the build system which produces a `.config` with all options turned off. As that is not very useful, we can supply it with a minimal configuration we require and let the build system only set the additional required options (because some options imply other options and so on). The `mini.conf` in our case only features five variables

```
CONFIG_PATMOS=y
CONFIG_SERIAL_CONSOLE=y
CONFIG_BIG_ENDIAN=y
CONFIG_ARM_APPEND_DTB=y
# CONFIG_EMBEDDED is not set
```

and can also be found in the repository. The last comment is actually needed as Landley himself explains in his very informative talk [4] . The first option tells the build environment that we want to build for PATMOS. The second enables the debug console and the third sets the endianness to big. The `CONFIG_ARM_APPEND_DTB` option specifies that the DTB is not passed to the kernel via a reference from the bootloader, but instead appended at the end of the kernel image in RAM. This option was introduced to ease the transition from older bootloaders which did not support DTB's. We think that this option makes it significantly easier to get the kernel to boot. The device tree binary is simply concatenated at the end of the kernel image via `cat arch/patmos/boot/zImage arch/patmos/boot/dts/patmos.dtb > myzImage` [5]

Also consider setting a kernel command line here. We never got to trying this option, but once the kernel is loaded it probably wants some sort of command line option in order to use the supplied debug console. Either teach the bootloader to add a command line to the supplied DTB (sounds hard), or set a fixed command line in the `.config` (sounds way easier). Refer to the .console section for more information on the console.

Running `make ARCH=patmos LLVM=1 V=1 KCONFIG_ALLCONFIG=mini.config allnoconfig` in the root folder of the kernel source, will produce a minimal `.config`. A consecutive `make ARCH=patmos LLVM=1 V=1` will start building the kernel. The extra `V=1` just enables a more verbose debugging output.

5 Device Tree

As already mentioned, the concept of DT's was introduced to allow the kernel to load drivers for non discoverable hardware. An example would be memory

[2] `https://github.com/t-crest/patmos`
[3] `https://lwn.net/Articles/160497/`
[4] `https://www.youtube.com/watch?v=Sk9TatW9ino`
[5] `https://events.static.linuxfound.org/sites/events/files/slides/petazzoni-device-tree-dummies.pdf`

mapped serial output (as present on PATMOS). The DT can become rather complicated and indeed useful, but we think that for the simple Linux on PATMOS implementation a very slim DT is sufficient. According to [6] a cpu block should be sufficient. Further, we require a serial block with the same name as the `console` driver for PATMOS in order for the kernel to load our driver. Petazzoni has more on DT's in general here [7], or on his slides [8] and in the general kernel docs [9].

Our device tree now looks like this (we think this is the minimal necessary implementation) and is located in `arch/patmos/boot/dts/patmos.dts`.

```
/ {
  #address-cells = <1>;
  #size-cells = <1>;

  cpus {
    #address-cells = <1>;
    #size-cells = <0>;

    cpu@0 {
      device_type = "cpu";
      model = "patmos";
      compatible = "t-crest,patmos";
      reg = <0>;
    };
  };

  serial@f0080000 {
    compatible = "ttyPATMOS";
  };
};
```

As mentioned in the .config section, the DTB should be appended to the kernel image and loaded into ram as a whole.

[6] https://elinux.org/Device_Tree_Usage

[7] https://www.youtube.com/watch?v=m_NyYEBxfn8

[8] https://events.static.linuxfound.org/sites/events/files/slides/petazzoni-device-tree-dummies.pdf

[9] https://www.kernel.org/doc/html/latest/devicetree/usage-model.html

6 Console

The console driver part is pretty much done (largely because it is very simple) and located in `arch/patmos/kernel/console.c`. It only implements busy writing and that's enough for now. The official kernel docs [10] describe how to set the `console=ttyPATMOS` kernel command line in order for that to work [11]. The command line can be pre-set in the kernel build configuration, but we did not get to that part.

Theoretically, the console driver could take arguments (like memory base addresses) from the DT (as it's intended), but for now we decided on hard coding everything.

7 General Hardware Abstraction

Most of the HAL in Linux actually works over `inline` functions defined in header files. All of the required files can be found under `include/asm-generic/` but not all files there are required. For example `arch/patmos/include/asm/cache.h` simply includes the corresponding generic `cache.h`. Looking into the generic version we see that it thus just sets some default cache line size. Most of the files work similar to this principle, where the generic version supplies the basic functionality and we can enhance or build on that. The file `arch/patmos/include/asm/irqflags.h` is also a good example. It just exports the headers of two required functions which are implemented in `arch/patmos/kernel/irqflags.c`. The rest is done in `include/asm-generic/irqflags.h`.

In `arch/patmos/kernel/head.c` we supply the starting symbol. After setting up the stack pointer and the empty exception handlers, we call `start_kernel`. Further information could be found

[10] https://www.kernel.org/doc/html/latest/core-api/printk-basics.html

[11] https://www.kernel.org/doc/html/latest/admin-guide/serial-console.html

here [12] or here [13] .

The directory `arch/patmos/include/generated` is auto populated by the build environment. If it doesn't exist, nothing works.

The directory `arch/patmos/include/uapi/asm` contains the following files with the following (assumed) semantics.

- `byteorder.h` sets the byteorder: big endian.
- `ptrace.h` defines the `struct pt_regs` used in context switching and everywhere registers need to be saved. No idea why in ptrace because we think that ptrace itself is actually optional. Maybe this can be moved to a different source file.
- `setup.h` just contains the maximum command line size.
- `sigcontext.h` defines the `struct sigcontext` allegedly used in signal contexts because it just contains the registers and the current interrupt flags.
- `unistd.h` no idea what is going on here. It seems to request features, so we turned them all of. This might break things. Maybe even remove this file.

In the other directories, namely `arch/patmos/include/asm`, we generally took the approach of deleting some file and if the build system complained we put it back piece by piece. We also heavily relied on the implementations of `arch/c6x` and `arch/microblaze` because both are mmu-less microprocessors.

Regarding makefiles we recommend reading the actual kernel documentation (its actually fairly readable) [14] because the Linux build system is something else entirely.

[12] `https://elinux.org/images/e/e3/Masters-PortingLinux.pdf`

[13] `https://www.embedded.com/linux-porting-guide/`

[14] `https://www.kernel.org/doc/html/latest/kbuild/makefiles.html`

8 Hacks

The hacks we employed along the way. Keep those in mind in case something really weird breaks.

- `include/linux/compiler-clang.h`: uncommented the version check. This should not be necessary once `patmos-clang` has been bumped to the most recent version!
- `scripts/Makefile.build`: added the option `-fpatmos-emit-reloc` in line 154 in order to produce an ELF output. Maybe this is better globally in the `$(c_flags)` variable?
- `scripts/Makefile.lib`: modified the sed expression to match the asm output. This might not be necessary anymore in the newest version!
- `scripts/Kconfig.include`: uncommented the check for the gold linker in line 43. Appearantly gold is not capable of linking the kernel properly.

9 Evaluation, Conclusion

We think this report provides a good starting point for future groups tackling this project. We documented all steps to get the build system to run, as debugging make is probably the most frustrating part. The project is at a state where one can employ 'compiler driven development': run make, fix errors as written. We also outlined the fastest way to some console output from the kernel, and identified potential pitfalls for the project. Still we recommend first implementing GDB support for `patmos-sim` before finishing this project.

10 Useful References

- **Landley, Simplest possible Linux System talk:** Youtube, LKML, github
- **Thomas Pettazzoni, Device Tree Talk:** Youtube, Slides, also Slides, Docs
- **bootlin.com, serial driver:** Serial Driver Presentation, Serial Driver PDF,

- **Porting, printk and nommu config:** General porting guide, Another general porting guide, Kernel nommu docs, Kernel printk docs, Linux Makefile docs(!)