naps - Relaxed SoC Design

Jaro Habiger* (anuejn@apertus.org), Robin Heinemann* (vup@apertus.org)
apertus° Association [*equal contributions]

Motivation: Rapid Prototyping of FPGA SoC Systems

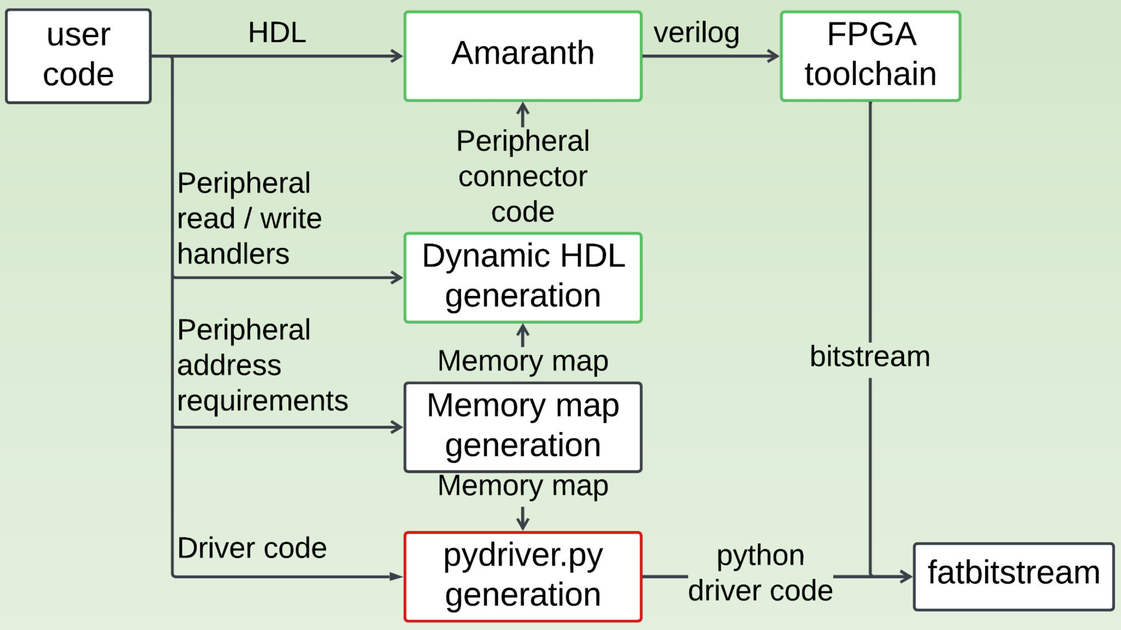
- FPGA-based systems often require **a mix of HDL and software running on CPUs** to tackle non-trivial problems. While FPGAs are great for high-speed interfaces and specialized computations, CPUs are more suited for control pane tasks.
- Designing such systems is often time consuming and tedious. Even though HDL and software components are interdependent, they are usually developed using separate languages and toolchains creating friction in the development process.
- With naps we want to improve developer tooling for building such integrated systems and enable **rapid prototyping** as especially enabled using the fast open source FPGA toolchains.

The naps Workflow: Python all the way!

- **naps uses Amaranth HDL.** This eDSL allows developers to describe gateware in Python leveraging metaprogramming to build high-level abstractions.
- naps provides building blocks for describing **memory mapped peripherals** accessible by the CPU. For example, any signal in the FPGA design can be marked as a **CSR** (Control and Status Register) which can be read / written from the CPU.
- naps automatically generates a memory map including all CSRs and other memory mapped peripherals and **automatically connects all peripherals to the SoC bus** by creating the necessary interconnects.

- **Software components of the system are also written in Python code** - in the same class hierarchy as the HDL. Functions that run on the CPU are annotated with special decorators. IDE features such as code completion and linting are directly available for the development of software.
- **Software can transparently access CSRs** or memory mapped peripherals using their Python identifiers. These are automatically resolved to addresses using the generated memory map.

- An **interactive Python shell** can be used to interact with the design. It includes auto-completion and allows for ad-hoc scripting, which helps with debugging designs.
- **HDL and software can be co-simulated** and unit-tested using standard python tooling.



Memory Mapped Peripherals: Abstracting Buses

- naps provides an interface to describe memory mapped peripherals in a **bus-agnostic** way using dynamic code generation.
- This interface is simplified to the bare minimum to reduce implementation complexity. It consists of only read and write operations of a fixed word size.
- To provide access to the peripherals, **naps automatically adds logic implementing a bus that the CPU can access**. Currently, naps supports **AXI-Lite** and **JTAG**.
- naps comes with a **rich library of pre-made peripherals**, such as host-accessible RAM, cores to read/write (packetized) streams, a bridge for the Xilinx PLL dynamic reconfiguration port, an Internal Logic Analyzer (ILA), etc.
- Memory mapped peripherals *can* be used to implement interfaces used by existing **linux kernel drivers**, which can automatically be configured and loaded using devicetree overlays. This can, for example, be used by a peripheral providing a memory-mapped HDMI framebuffer to configure the linux kernel with the correct address, resolution and data format.

Fatbitstream: Bundling Bitstreams and Software

The bitstream, accompanying (generated) software and support files are bundled into an executable ZIP file called “fatbitstream”. Fatbitstreams enhance the development experience by combining all interdependent components into a single file and initializing both hardware and software to the expected state.

entrypoint, unpacks the ZIP file, executes initialization commands (programming FPGA, loading devicetree overlays, etc.)

contains the Python software code snippets collected from the class hierarchy and support code for accessing the peripherals .

bitstream generated from the FPGA design

other generated files, for example devicetree overlays

Example: Mixing HDL and Software

The following code demonstrates the use of the naps framework on the simple example of measuring the frequency of a clock signal.

```
class ClockMeter(Elaboratable):
    def __init__(self):
        self.counter = StatusSignal(64)

    def elaborate(self, platform):
        m = Module()
        m.d.sync += self.counter.eq(self.counter + 1)
        return m

    @driver_property
    def mhz(self, t=0.1):
        from time import sleep
        initial_counter = self.counter
        sleep(t)
        return (self.counter - initial_counter) * (1 / t) / 1e6
```

Implemented in FPGA fabric

Running on CPU

Exploration: A Python Shell for FPGA Designs

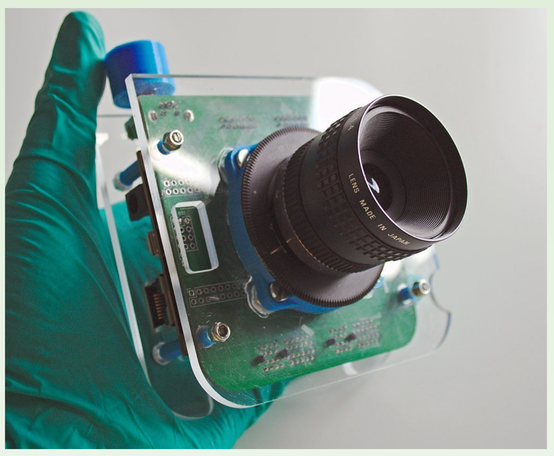
An active naps design can be interactively explored using an interactive Python shell. This python shell exposes all CSRs, memory-mapped peripherals and annotated driver functions with autocompletion.

```
Python 3.10.10 (main, Apr  5 2023, 14:58:08) [Clang 11.1.0 ] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> dut.clock_meter.mhz
100.0012345
```

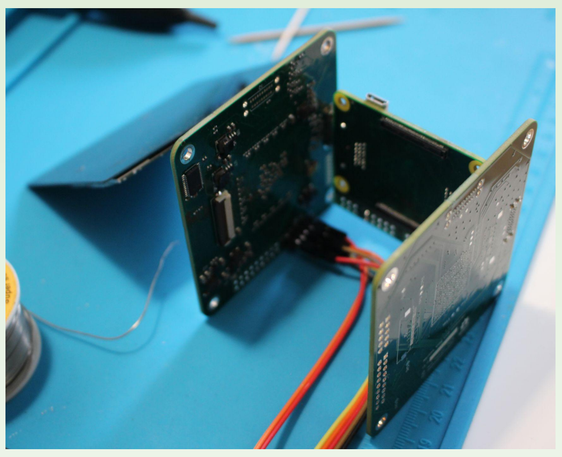
Case Studies: Hackable Video Processing

AXIOM micro open source video camera using a **Xilinx Zynq** 7010 as its main FPGA. Rapid prototyping and hackability of the code are top-priorities for this project. In this context, a bunch of open-source IP cores were developed and debugged using naps:

- HDMI output, including support for usage as a Linux framebuffer
- HiSPI input for receiving data from image sensors
- **Image convolution such as debayering and focus peeking (edge detection) leveraging Amaranth HDL metaprogramming**



AXIOM micro



Merlins Cube

Realization of the “Merlins Cube” art installation comprising a cube with touchscreens on all six sides. The installation was powered by a Raspberry-Pi compute module outputting two HDMI streams to ECP5 based connector boards each driving 3 MIPI-DSI screens. naps was used to develop the code running on the joint **Raspberry-Pi/ECP5** system. More open source IP cores were developed and debugged using naps:

- HDMI input
- MIPI-DSI output including an **interactive terminal for DCS (Display Command Set) commands**

References

whitequark. 2019. “Amaranth HDL (Previously NMigen): A Modern Hardware Definition Language and Toolchain Based on Python.” 2019. https://github.com/amaranth-lang/amaranth.

Apertus Open Source Cinema. 2019. “naps” 2019. https://github.com/apertus-open-source-cinema/naps.