Managing the material around your FPGA with Python/Amaranth

Time & Frequency department

N. Gallone

under the direction of J.-M. Friedt and G. Goavec-Merou

slides and references at

`https://github.com/oscimp/amaranth_twstft`

Outline

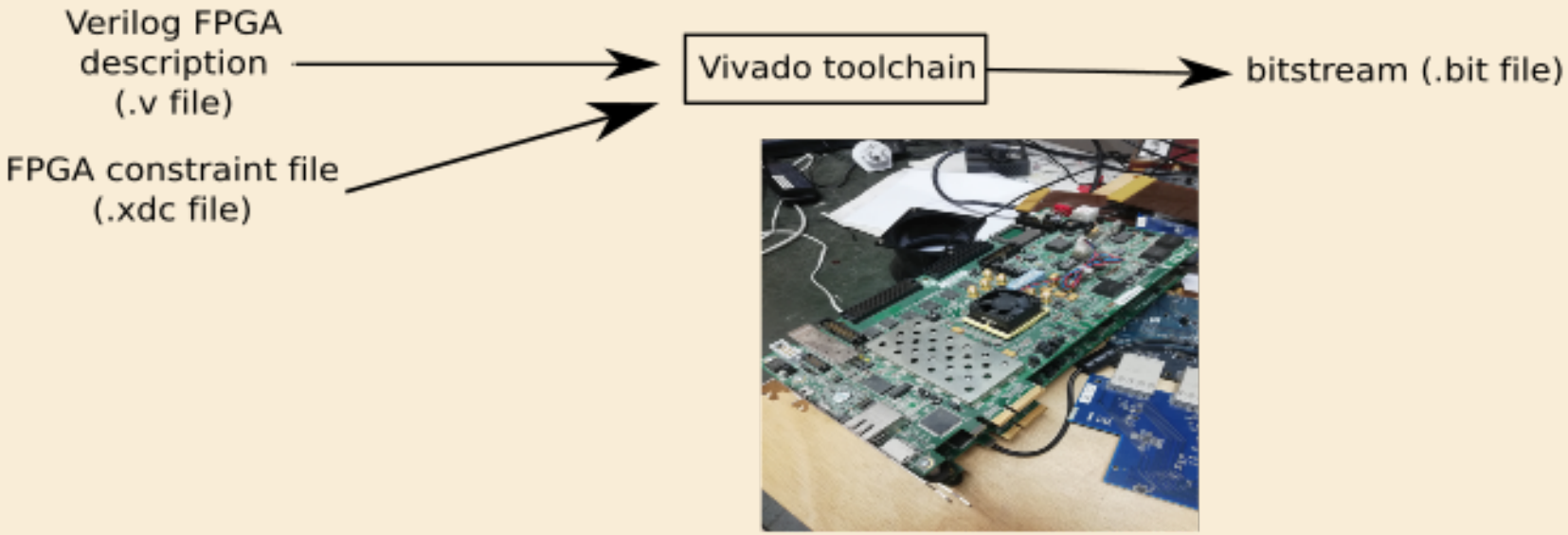
1. An overview of FPGA synthesis tools...
2. Accessing different resources with amaranth
3. Changing the clock rate of the FPGA with a PLL

Outline

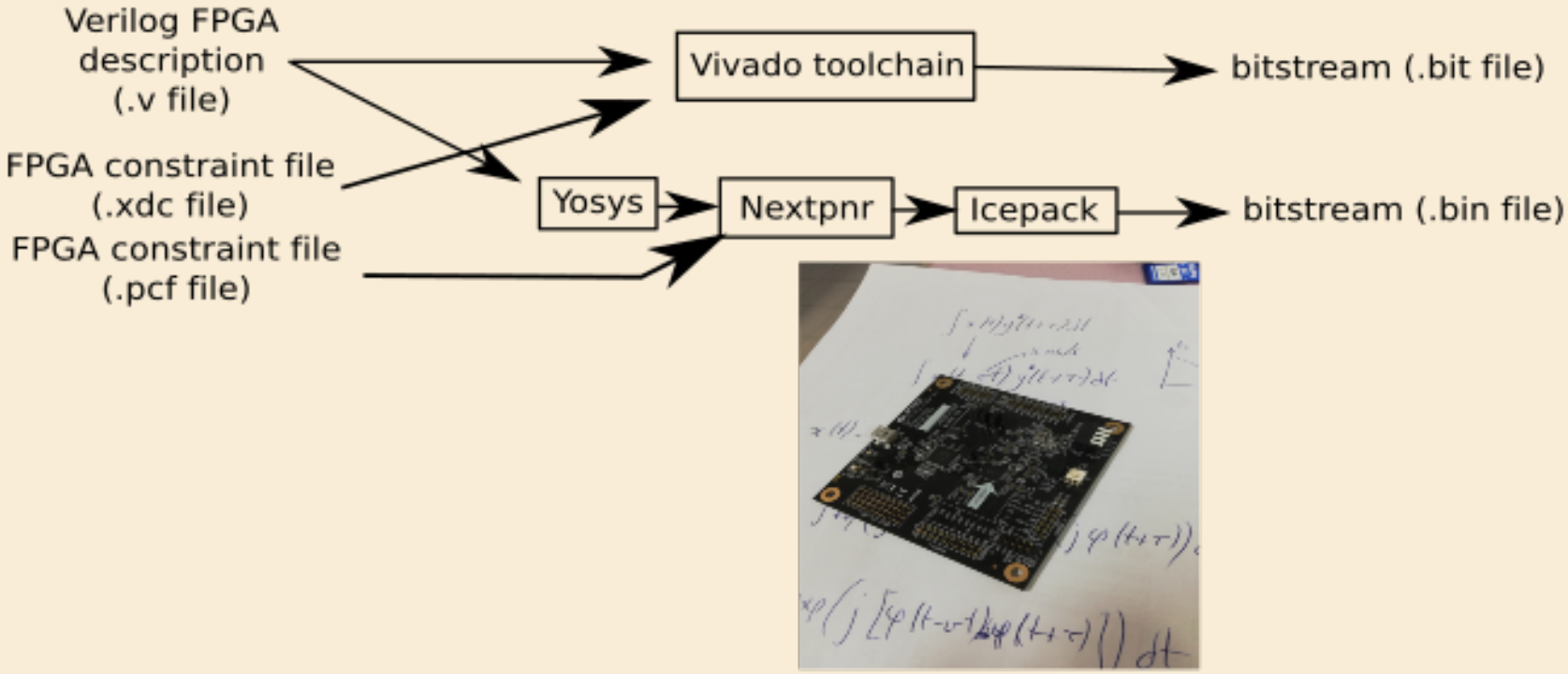
Current state of the art

- Plenty of different FPGA vendors
- Each one of them defines their own synthesis toolchain
- They only have the Hardware Description Language code in common (VHDL, Verilog or SystemVerilog)

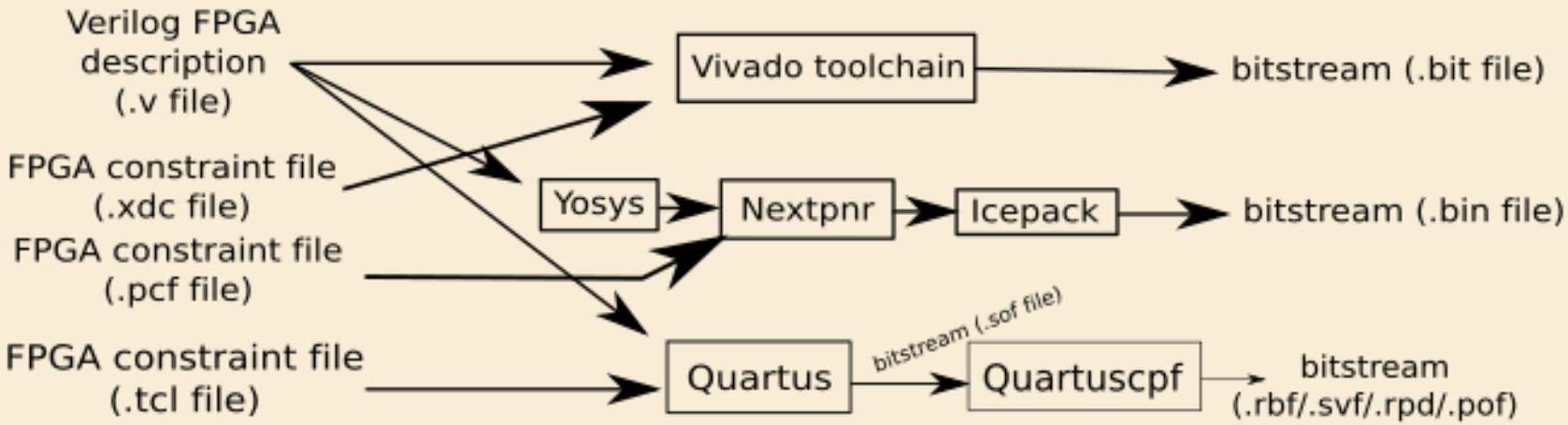
Very low portability



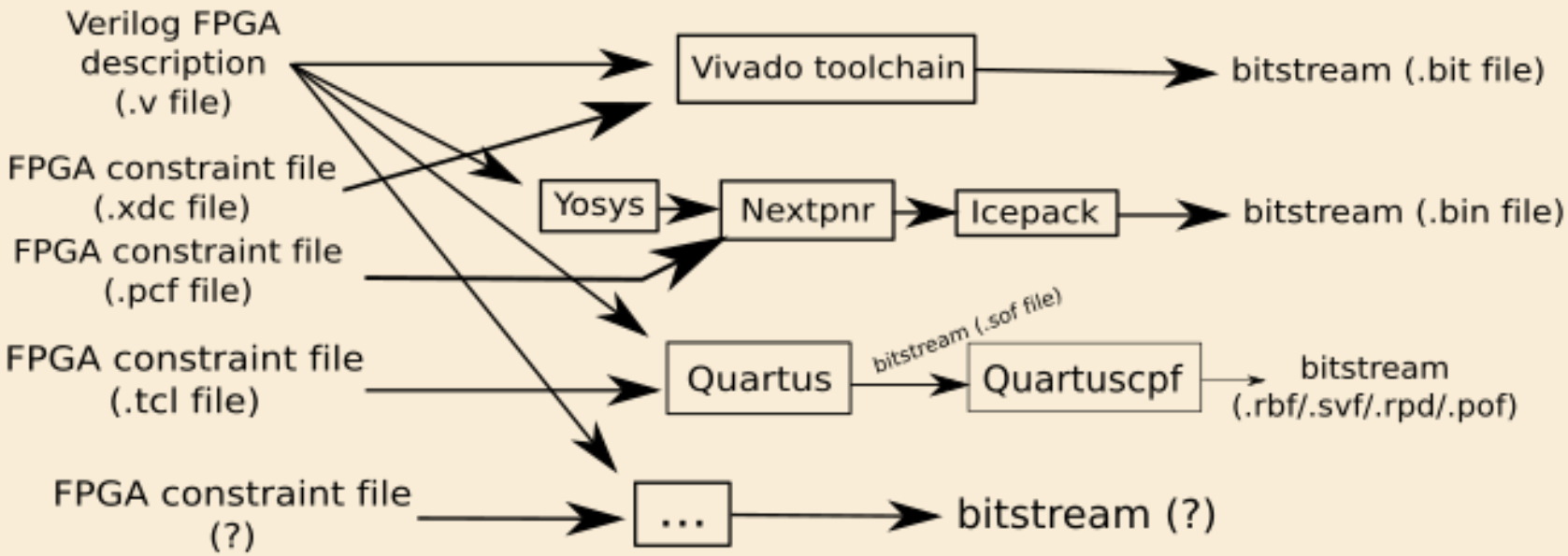
Very low portability



Very low portability



Very low portability



Amaranth : the redemption

- Simplified python based HDL
- Generates verilog files
- Manages the toolchain usage so that we only need to care about our FPGA description

Amaranth : the redemption

- Simplified python based HDL
- Generates verilog files
- Manages the toolchain usage so that we only need to care about our FPGA description

One code to rule them all

Just refer to the constraint file of your platform to make use of the different resources you need.

Outline

The resource system

The platform

Defined in the amaranth-boards directory and is made up of a platform class with interesting properties such as :

- device (the name of the platform)
- default_clk (the name of the default clock)
- resources (the list of all the materials on your platform)
- connectors (the list of the generic connectors (pmods/gpio/sma/...) available on your platform)

Accessing a resource

```
def elaborate(self, platform):
    m = Module()
    if platform is not None:
        if platform.device == platform_name:
            myresource = platform.request(resource_name, →
                ↪resource_index)
            # use myresource.i as a regular Signal
            # but only as input
            # buttons/switch buttons/clocks...
            # use myresource.o as a regular Signal
            # but only as output
            # led/Display7Seg/VGA...
    return m
```

Creating new kinds of resources

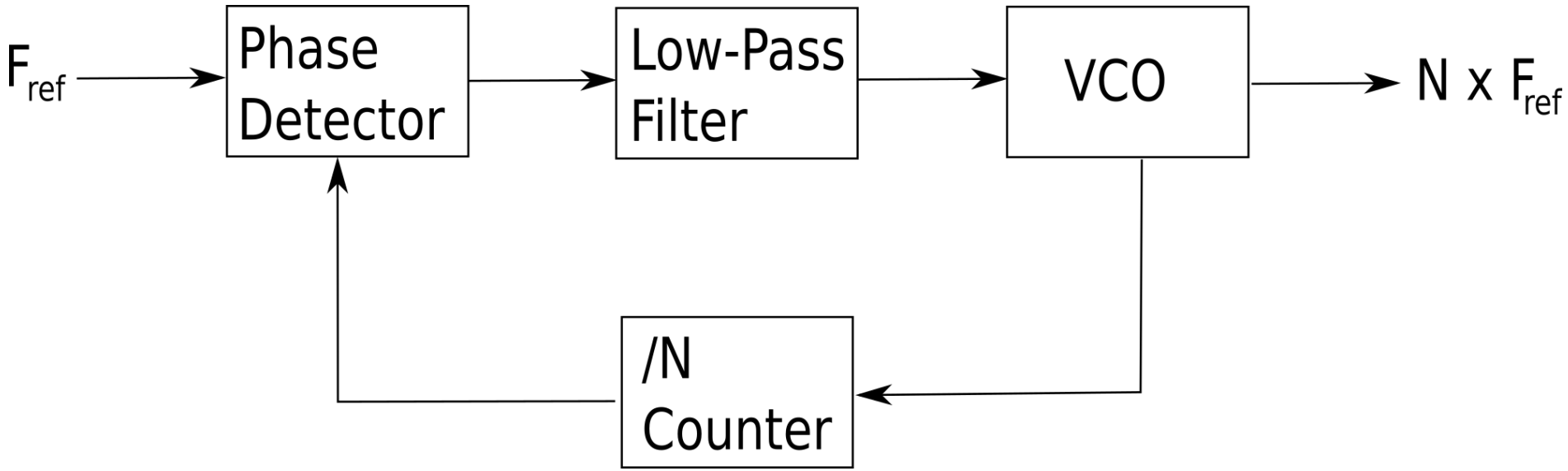
```
def elaborate(self, platform):
    m = Module()
    if platform is not None:
        if platform.device == platform_name:
            platform.add_resources([
                Resource("io_res", 0,
    Subsignal('input', Pins('7', conn = conn, dir='i')),
    Subsignal('output', Pins('15', conn = conn, dir='o')),
    Attrs(IO_STANDARD="SB_LVCMOS")
    )
            ])

            io_resource = platform.request("io_res", 0)


    return m
```

Example program

Outputing our LFSR Pseudo-Random Noise on an UP5K ICE40 (Lattice)

Outline

What is a PLL ?



In the end...

- Amaranth platform constraint files
- Access GPIOs described in such files
- Access and use a PLL to change the clockrate