Padframe Generator for open-source RTL2GDS Qflow

Philipp Gühring
Vienna, Austria
pg@futureware.at

***Abstract* — An opensource padframe generator was developed on the efabless platform for usage with the Open-Source Qflow Digital Synthesis Flow, for digital logic chips in the X-FAB XH018, 180nm process.**

Keywords—Qflow; padframe generator; efabless; X-Fab; 180nm; toplevel routing; OpenGalaxy; verilog

I. Introduction

Qflow[2] is a toolflow that currently supports synthesizing, placing, routing, and LVS and DRC checking a core. The padframe generation and toplevel routing wasn't automated by Qflow yet, and had to be done manually. This padframe generator automates the padframe generation, and leaves only the toplevel routing as the final manual step for now. During the webinar[1] about Qflow, where one can learn how to design chips on the efabless[3] platform, I found the manual way to build a padframe too tiresome, so I automated it.

II. Usage

To use the padframe generator, the first step is to synthesize the verilog source, place and route the core completely with Qflow in an OpenGalaxy project.

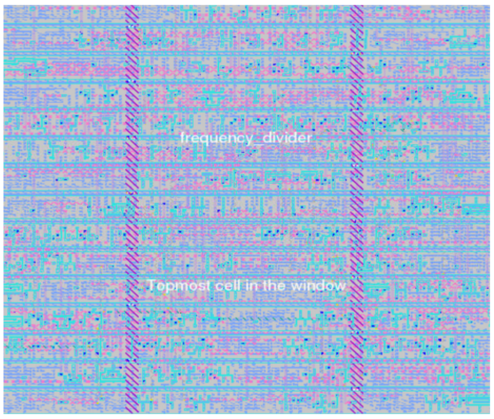


Image 1: Core generated by Qflow

The next step is to create a new OpenGalaxy project for the whole chip, edit the layout in magic, add a fully generated single core from the previously generated project beneath your home directory, and save it as a .mag file in the current directory. Then you run the generator, which generates a pad-frame for all the pins in the referenced design:

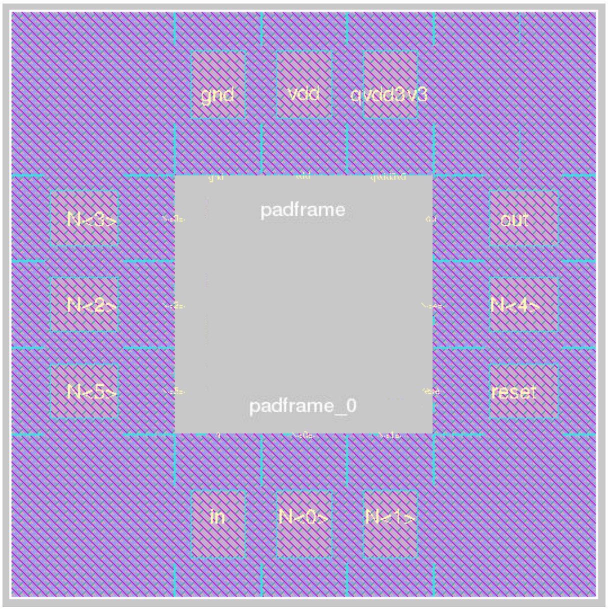


Image 2: generated Padframe

Open magic again, and include the generated padframe.mag, align them and do the toplevel routing:

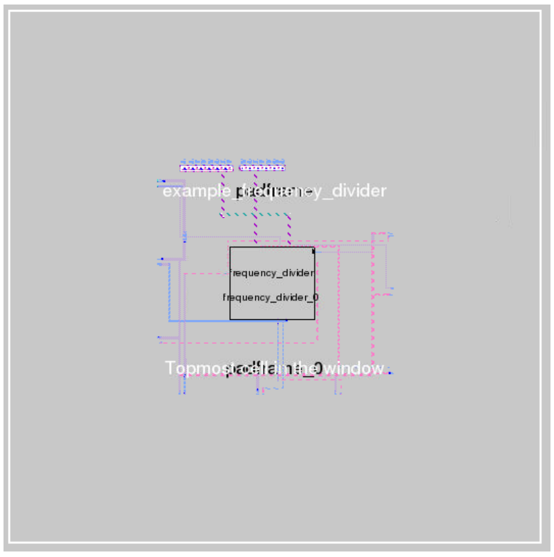
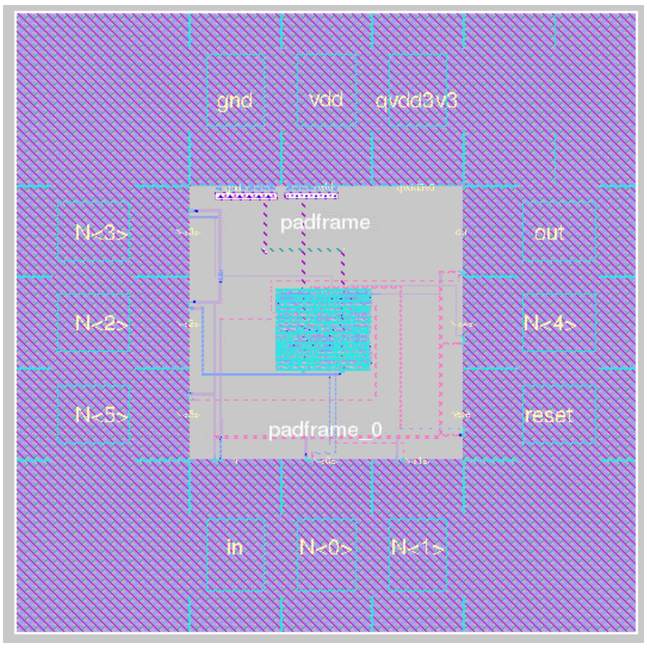
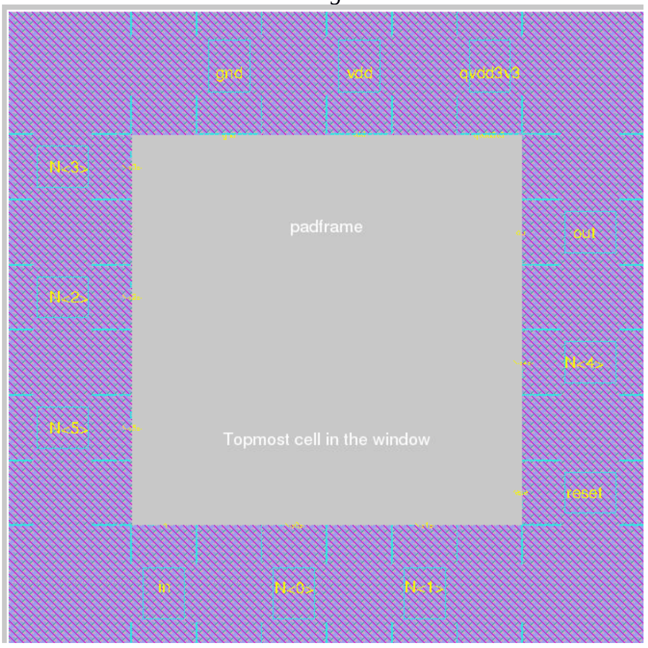


Image 3: manual Toplevel routing



Image



III. Internals

The first step of the padframe generator is to collect the requirements for the padframe. It extracts the inputs and outputs from the comp.json file that is generated by the Qflow tool and referenced in the current project. Based on the names of the signals, the generator guesses the function and uses appropriate pad frame cells from the cell library provided by X-FAB[4]:

TABLE I. Table Type Styles

signal name mapping to pad frame cells and verilog types			
Padname	***XH018***	***Verilog***	***Description***
QVDD3V3	VDDORPADF	input real	3.3V core voltage
VDD	VDDPADF	input real	1.8V core voltage
VCC	VDDIPADF	input real	Supply voltage
GND	GNDORPADF	input real	0V ground
RESET, RST	ICF	input	reset signal input
OUT	BT4F	output	Output signals
GPIO	BBCUD4F	inout	GPIOs

For example, “out” signals are using the BT4F cell, “gpio” signals are using the BBCUD4F cell, input signals are using the ICF cell. In case of an odd number of signals, the padframe generator automatically adds a FILLER84F cell to fill up the padframe and make it an even number of cells in the frame. For the supply voltages, both a 3.3V and 1.8V and a ground pad are needed and automatically added by the generator where necessary.

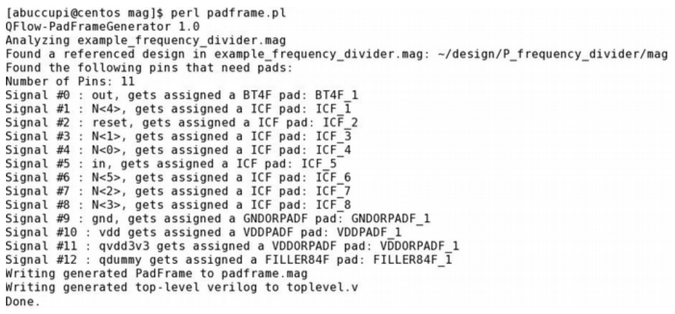
For designs with a large logic area and only few input output signals, the padframe generator can be parameterized to add a number of FILLER84F cells after every signal pad cell, to make sure that the resulting padframe is large enough to contain the generated die and top-level routes within the padframe (e.g. Image 5).

IV. Placement

The placement engine starts at the 0/0 coordinates, and whenever a quarter of the cells is placed, is places a corner cell and rotates by 90°. It places the cells, and additionally places large text labels naming the signal name in the center of the pads, which should help to get an overview, and smaller text labels at the center of the inside borders, which help to identify the signals for connecting them during toplevel routing. If the number of pads is divisable by 4, the resulting padframe will be quadratic, otherwise it will be rectangular.

V. Outputs

The padframe generator generates both a Magic padframe.mag file, which contains the cells with their position and the text labels, and a toplevel.v verilog netlist file, which should be usable for LVS checks.



```
[abuccupi@centos mag]$ perl padframe.pl
QFlow-PadFrameGenerator 1.0
Analyzing example_frequency_divider.mag
Found a referenced design in example_frequency_divider.mag: ~/design/P_frequency_divider/mag
Found the following pins that need pads:
Number of Pins: 11
Signal #0 : out, gets assigned a BT4F pad: BT4F_1
Signal #1 : N<4>, gets assigned a ICF pad: ICF_1
Signal #2 : reset, gets assigned a ICF pad: ICF_2
Signal #3 : N<1>, gets assigned a ICF pad: ICF_3
Signal #4 : N<0>, gets assigned a ICF pad: ICF_4
Signal #5 : in, gets assigned a ICF pad: ICF_5
Signal #6 : N<5>, gets assigned a ICF pad: ICF_6
Signal #7 : N<2>, gets assigned a ICF pad: ICF_7
Signal #8 : N<3>, gets assigned a ICF pad: ICF_8
Signal #9 : gnd, gets assigned a GNDORPADF pad: GNDORPADF_1
Signal #10 : vdd gets assigned a VDDPADF pad: VDDPADF_1
Signal #11 : qvdd3v3 gets assigned a VDDORPADF pad: VDDORPADF_1
Signal #12 : qdummy gets assigned a FILLER84F pad: FILLER84F_1
Writing generated PadFrame to padframe.mag
Writing generated top-level verilog to toplevel.v
Done.
```

Image 6: Output from padframe generator

As a future extension, it is planned to generate the netlist and configuration for a routing tool (currently preferred: qrouter[5]) to automate the toplevel routing afterwards.

VI. Room for improvements

- The first version of the padframe generator was developed in Perl due to timing constraints. For newer versions, Python would be preferred, for better integration into qflow.

- Enhancing the padframe generator to generate padframes for multiple cores inside the chip
- Capability to let the user influence the positioning/order of the pads
- Capability to let the user override the guesses which pads are necessary for the signals
- Automatically derive the amount of necessary filler pads for the size of the core so that there is enough space inside the padframe for the core
- Adaptation to other cell libraries / process nodes

References

[1] T. Edwards and K. Ghosh, “SoC Design of the PicoRV32 RISCV micro-processor“ Webinar for using Qflow: https://www.udemy.com/vsd-soc-design-of-the-picorv32-riscv-micro-processor/

[2] Qflow software: http://opencircuitdesign.com/qflow/index.html

[3] Efabless Design platform: https://efabless.com/

[4] XH018 Datasheet: https://www.xfab.com/fileadmin/X-FAB/Download_Center/Technology/Datasheet/XH018_Datasheet.pdf

[5] *qrouter software: http://opencircuitdesign.com/qrouter/*