The FPGA design process

Mario Freisinger

08.11.2023

FPGA design process Overview

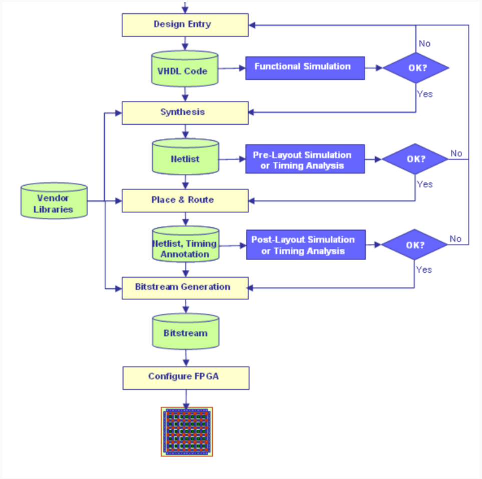


Figure 1: Steps to convert a design described with a HDL to a bitstream. [1]

Synthesis

Synthesis

- Convert RTL description to gate level description
- Translation of the design described with a HDL to a netlist according to the specification of the used hardware description language.
- For example in SystemVerilog the following construct should become a flip-flop.

```
always_ff @(posedge clk) begin
  if (~reset)
    reg_ff <= '0;
end
```

Synthesizer Example

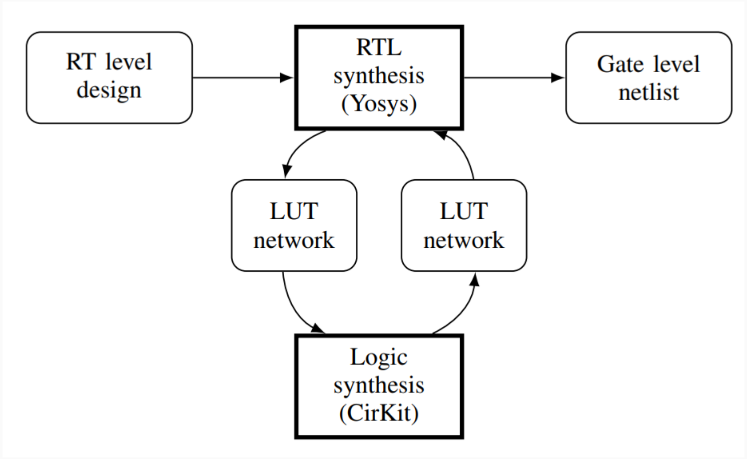


Figure 2: Intermediate steps of a HDL synthesizer like yosys. [2]

Netlist

- Describes how certain modules are connected
- Can be technology dependent or independent
- After the tech-mapping process it is technology dependent (A gate level netlist)
    - Only makes use of the Gates that are available on the targeted architecture
    - The gate level netlist can be simulated

[3]

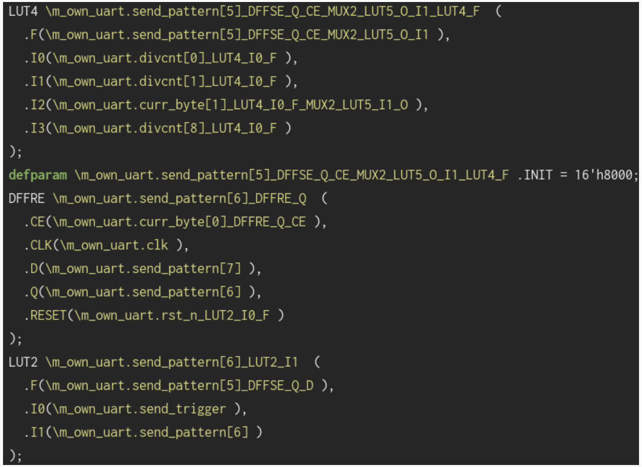
Gate level netlist

A post-synthesis gate level netlist consists of:

- Technology-dependent logic elements(std cells) or LUTs
- Technology-dependent storage elements: e.g. flip-flops
- Hardened IP Blocks like DSPs, Block Ram

[4]

Gate Level Netlist Example

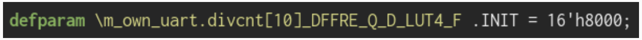


```
LUT4 \m_own_uart.send_pattern[5]_DFFSE_Q_CE_MUX2_LUT5_O_I1_LUT4_F  (
  .F(\m_own_uart.send_pattern[5]_DFFSE_Q_CE_MUX2_LUT5_O_I1 ),
  .I0(\m_own_uart.divcnt[0]_LUT4_I0_F ),
  .I1(\m_own_uart.divcnt[1]_LUT4_I0_F ),
  .I2(\m_own_uart.curr_byte[1]_LUT4_I0_F_MUX2_LUT5_I1_O ),
  .I3(\m_own_uart.divcnt[8]_LUT4_I0_F )
);
defparam \m_own_uart.send_pattern[5]_DFFSE_Q_CE_MUX2_LUT5_O_I1_LUT4_F .INIT = 16'h8000;
DFFRE \m_own_uart.send_pattern[6]_DFFRE_Q  (
  .CE(\m_own_uart.curr_byte[0]_DFFRE_Q_CE ),
  .CLK(\m_own_uart.clk ),
  .D(\m_own_uart.send_pattern[7] ),
  .Q(\m_own_uart.send_pattern[6] ),
  .RESET(\m_own_uart.rst_n_LUT2_I0_F )
);
LUT2 \m_own_uart.send_pattern[6]_LUT2_I1  (
  .F(\m_own_uart.send_pattern[5]_DFFSE_Q_D ),
  .I0(\m_own_uart.send_trigger ),
  .I1(\m_own_uart.send_pattern[6] )
);
```

Figure 3: Example of a gate-level-netlist after synthesizing - created with yosys

LUT configuration

- The contents of the LUTs are also stored in the gate level netlist



```
defparam \m_own_uart.divcnt[10]_DFFRE_Q_D_LUT4_F .INIT = 16'h8000;
```

Figure 4: LUT4 initialized with defparam in a gate level netlist created by yosys

I0	I1	I2	I3	O
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
.	.	.	.	.
1	1	1	1	1

Table 1: Look-up-table initialized with 16'h8000 = 16'b1000000000000000 and represents therefore an and of all four inputs

Timing analysis

- Dynamic Timing Analysis
    - Apply certain input vectors and estimate the timing behaviour of the circuit
    - Quality depends on input vectors
    - Hard to check every possible path of the design
    - Long run times [5]
- Static Timing Analysis
    - No input vectors are applied only statically analysed to determine if certain timing violations specified by vendor libraries occur
    - Define timing requirements
    - Check every topological path of the design but only for one cycle if timing requirements are fulfilled [5]

Static timing analysis(STA)

- Net delay: time it takes to charge the parasitic capacitance via the parasitic resistances
- Cell delay: depends on input transition time(slew), output capacitance, temperature
- Path delay = net delay + cell delay
- Timing constraints: Setup time, hold time, in-output delay specified by technology file

[5]

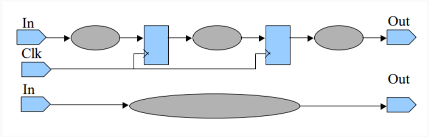


Figure 5: Block-diagram of an exemplary digital circuit [5]

The FPGA design process

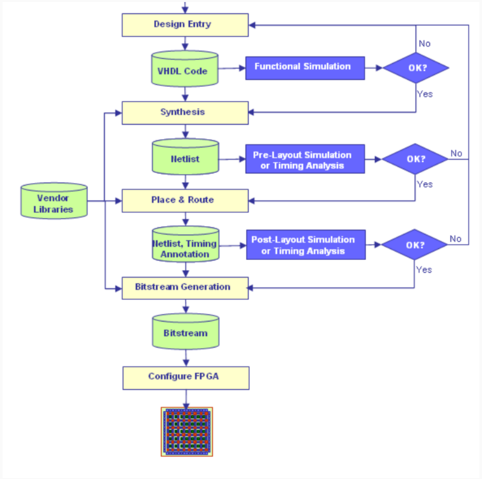


Figure 6: Steps to convert a design described with a HDL to a bitstream [1]

Place and Route (PNR)

Blocks of an FPGA

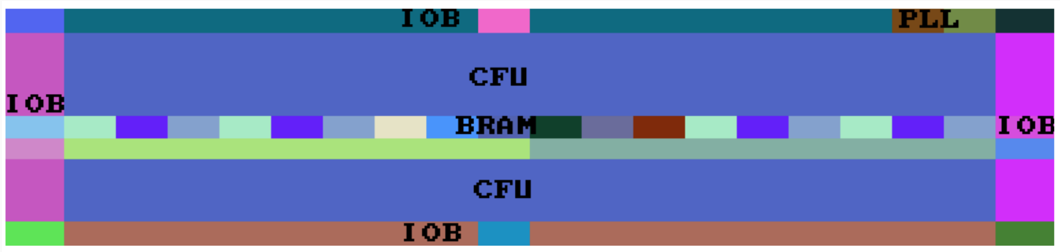


Figure 7: Basic FPGA Layout-Blocks(GOW1N-1 reverse engineered) [6]

- PLL for clock frequency scaling
- Most larger memory descriptions in HDL are mapped to Block RAM
- CFU (Configurable logic blocks)
- IOB IO Buffers

General FPGA layout with switch-boxes

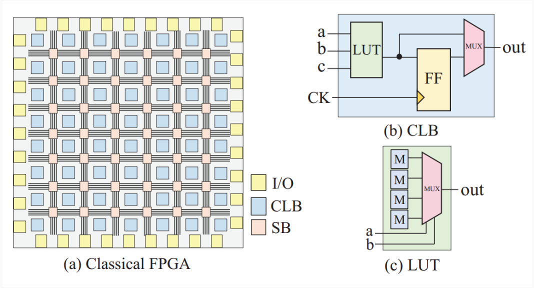


Figure 8: Basic FPGA Layout and the connection matrix via switch-boxes. [7]

Place vs Route

Place:

- Place modules such that wire length is minimized, given a netlist and a technology file of the FPGA
- Input: Netlist and technology file

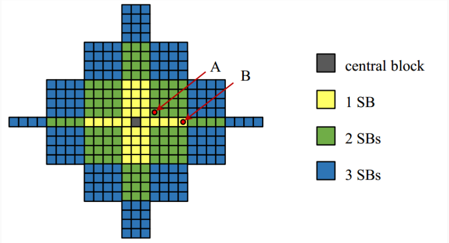
Route:

- Assign all module in- and outputs to the switch-boxes in such a way that the placed modules are connected as specified by the netlist
- Input: Netlist and placed circuit netlist

[7]

Analytical Placement

- Global placement - ignoring non-overlapping constraint
- Legalization - moving those modules that overlap
- Detailed Placement - improve solution further



Module Position	Distance	HPWL	# of SBs	Delay (ps)
A	(2,1)	3	2	405
B	(5,0)	5	1	309

Figure 9: Exemplary analytic placement algorithm [7]

Route

2 Major steps:

- Path searching
- Congestion Removal

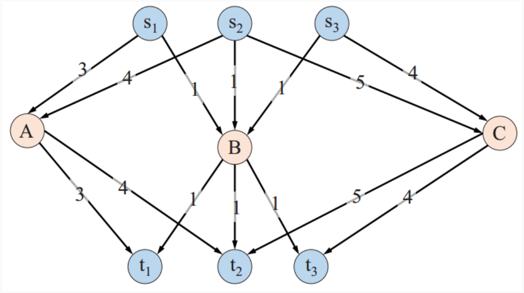
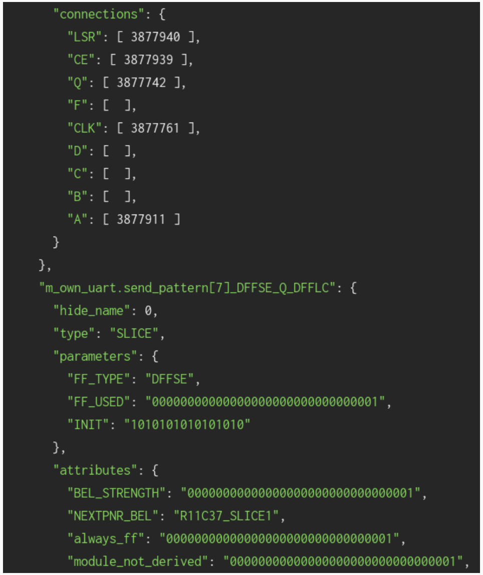


Figure 10: Exemplary Graph model: blue node is a CLB, sX are sources, tX are sinks and the orange nodes are switch-boxes [7]

Placed and routed design stored as JSON file by nextpnr



```
    "connections": {
      "LSR": [ 3877940 ],
      "CE": [ 3877939 ],
      "Q": [ 3877742 ],
      "F": [  ],
      "CLK": [ 3877761 ],
      "D": [  ],
      "C": [  ],
      "B": [  ],
      "A": [ 3877911 ]
    }
  },
  "m_own_uart.send_pattern[7]_DFFSE_Q_DFFLC": {
    "hide_name": 0,
    "type": "SLICE",
    "parameters": {
      "FF_TYPE": "DFFSE",
      "FF_USED": "00000000000000000000000000000001",
      "INIT": "1010101010101010"
    },
    "attributes": {
      "BEL_STRENGTH": "00000000000000000000000000000001",
      "NEXTPNR_BEL": "R11C37_SLICE1",
      "always_ff": "00000000000000000000000000000001",
      "module_not_derived": "00000000000000000000000000000001",
```

Bitstream

Contains:

- Logictables that are written into the LUTs of the CLUs
- Configuration of the switch-boxes
- BRAM content
- Other configurations for the PLL and other hardened blocks of the FPGA ...

Thank you for Listening!

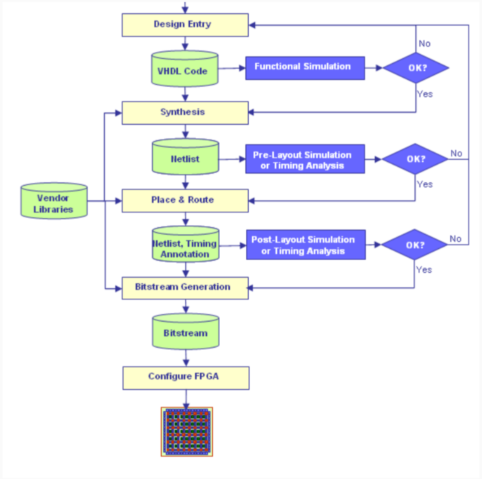


Figure 12: Steps to convert a design described with a HDL to a bitstream [1]

References i

References

[1] A. Shoufan and S. A. Huss, **A course on reconfigurable processors,** ACM Trans. Comput. Educ., vol. 10, no. 2, 7:1–7:20, 2010. DOI: 10.1145/1789934.1789937. [Online]. Available: https://doi.org/10.1145/1789934.1789937.

[2] H. Riener, M. Soeken, E. Testa, and G. D. Micheli, **Generic logic synthesis meets rtl synthesis,**, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:220747516.

[3] C. Wolf, **Yosys open synthesis suite presentation,**, 2016. [Online]. Available: http://yosyshq.net/yosys/files/yosys_presentation.pdf.

References ii

[4] B. L. Barzen, A. Reais-Parsi, E. Hung, *et al.*, **Narrowing the synthesis gap: Academic fpga synthesis is catching up with the industry,** in 2023 Design Automation und Test in Europe Conference und Exhibition, 2023, pp. 1–6. DOI: 10.23919/DATE56975.2023.10137310.

[5] C. Forzan, **Introduction to static timing analysis,**. [Online]. Available: http://www-micro.deis.unibo.it/~campi/Dida01/materiale/sta_overview.pdf.

[6] P. de Vos, M. Kirchhoff, and D. Ziener, **A complete open source design flow for gowin fpgas,** in International Conference on Field-Programmable Technology, (IC)FPT 2020, Maui, HI, USA, December 9-11, 2020, IEEE, 2020, pp. 182–189. DOI: 10.1109/ICFPT51103.2020.00033. [Online]. Available: https://doi.org/10.1109/ICFPT51103.2020.00033.

References iii

[7] S. Chen and Y. Chang, **FPGA placement and routing,** in 2017 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2017, Irvine, CA, USA, November 13-16, 2017, S. Parameswaran, Ed., IEEE, 2017, pp. 914–921. DOI: 10.1109/ICCAD.2017.8203878. [Online]. Available: https://doi.org/10.1109/ICCAD.2017.8203878.

[8] J. Serrano, **Introduction to fpga design,**, 2008. [Online]. Available: https://api.semanticscholar.org/CorpusID:19031067.

[9] X. Wang and J. You, **Summary of tools in fpga static timing analysis,** in 2020 7th International Conference on Dependable Systems and Their Applications (DSA), 2020, pp. 491–492. DOI: 10.1109/DSA51864.2020.00084.

References iv

[10] Y. Kukimoto, M. Berkelaar, and K. Sakallah, **Static timing analysis,** in Logic Synthesis and Verification, S. Hassoun and T. Sasao, Eds. Boston, MA: Springer US, 2002, pp. 373–401, ISBN: 978-1-4615-0817-5. DOI: 10.1007/978-1-4615-0817-5_14. [Online]. Available: https://doi.org/10.1007/978-1-4615-0817-5_14.