TECHNISCHE UNIVERSITÄT DRESDEN

DEFENCE: GROSSER BELEG

Graphical Support for the Design and Evaluation of Configurable Logic Blocks

Fredo Erxleben

Dresden, 4th June 2015

Section 1

Introduction to the Problem

01 Problem Statement

Aim: Have a tool(-chain) that allows to

- Create Schematics of CLBs
- Find the mappings of given boolean expressions onto these CLBs
- Represent the found mappings

(Restricted to combinatoric circuits.)

01 The Initial Plan

- Integration into the **Q**uite **U**niversal **C**ircuit **S**imulator [Erxl14][BrJa08]
- Use of intermediate formats (netlists, dimacs)
- Solve the resulting SAT-problem

Nice plan you have there. . .

. . . It would be a shame if something happened to it.

01 But. . .

Integration into *Qucs* [Tor+14] turned out to be infeasible.
No viable alternatives could be found.

I built my own tool . . .

. . . With a GUI and the features I required.

Bender is ©Matt Groening and David X. Cohen

Section 2

Theoretical Background

Note

CBD = **C**omponent **B**ehaviour **D**escriptor

02 Variables and pCBDs

Each component ϵ has

Configuration variables	C_ϵ
Node variables	N_ϵ
partial CBDs	$f_{\epsilon,i}(V_\epsilon) \mid V_\epsilon \subseteq (C_\epsilon \cup N_\epsilon),\ i \in [0, n) \subset \mathbb{N}$

In a module context there are

Input variables	X
Nodes superset	$N := \bigcup_\epsilon N_\epsilon$
Configurations superset	$C := \bigcup_\epsilon C_\epsilon$

(Module outputs are simple nodes.)

02 Describing a Component

For a component ϵ, one can put all pCBDs together into the CBD:

$$F_\epsilon := \bigcup_n f_{\epsilon,n}(V_\epsilon) \tag{1}$$

Everything together forms a components context

$$\square_\epsilon := (C_\epsilon, N_\epsilon, F_\epsilon) \tag{2}$$

02 Connecting Components

Let there be two components ϵ_a and ϵ_b, connected to the same node. This common node can be expressed as

$$\sigma_{a,b} = \{n_a \Leftrightarrow n_b | n_a \in N_{\epsilon_a} \wedge n_b \in N_{\epsilon_b} \wedge \ n_a \text{ and } n_b \text{ are connected}\} \tag{3}$$

Also, inputs need to be connected in a similar fashion

$$\sigma^* = \{x \Leftrightarrow n | x \in X \wedge n \in N \wedge \ x \text{ and } n \text{ are connected}\} \tag{4}$$

Note: Each σ is a set of boolean formulae.

02 The Global Context

Put all formulae into one set . . .

$$F := \bigcup_{\epsilon} F_{\epsilon} \cup \bigcup_{a,b} \sigma_{a,b} \cup \sigma^* \tag{5}$$

. . . and merge it into one formula.

$$F^* := \bigwedge_{f \in F} f \tag{6}$$

Finally, a global context that describes the whole module

$$\square := (C, N, X, F) \tag{7}$$

02 The Target Formulae

The functionality to be mapped can be expressed as a set of boolean formulae

$$T := \bigcup_i t_i \,|\, i \in \mathbb{N} \tag{8}$$

with

$$t_i\,(V) \,\Big|\, \begin{array}{l} V \subseteq (X \cup N) \\ i \in [0, n) \subset \mathbb{N} \end{array} \tag{9}$$

T can be merged into one formula

$$T^* := \bigwedge_{t \in T} t \tag{10}$$

02 Putting Everything Together

$$\exists \underline{c}.\forall \underline{x}.\exists \underline{n}.\,(F^* \wedge T^*) \;\Big|\; \underline{c} \in C^{|C|};\ \underline{x} \in X^{|X|};\ \underline{n} \in N^{|N|} \tag{11}$$

In plain text

There exists a configuration that for all input combinations, there is a node assignment so that $F^* \wedge T^*$ is satisfied.

This form is known as an *EAE*-problem,
which is a specialization of a *QBF*-problem,
which can be transformed into a *SAT*-problem [SaBa05].

02 Introducing Quantor

This is a QBF-problem → Use a dedicated QBF-solver[Nar+06].

Quantor

- Checks satisfiability of QBF-problems
- Additionally yields the variable assignment for the outermost ∃
- Open source and easy to integrate
- Uses *picosat* for internal SAT-Solving

Input to *Quantor* has to be provided in CNF.

Section 3

Design Decisions

03 Fundamentals

- Usage of `C++11` with 
- Separate visual representation from internal model
- Component information not hard-coded
  - Allow user-created component libraries

03 Using a Metamodel

Basic idea: Components with the same structure are of the same type (→ HDLs).

Component descriptors	abstract components with similar structure and behaviour
Component elements	serve as building blocks for component descriptors
Categories	aid with keeping an overview and forming hierarchies

Component descriptors can be created by the user with simple means.

03 A User-Defined Component Descriptor

```
1  {   "name":"lut2cnf",
2      "ports":
3      [   {"name":"x0",   "direction":"in"},
4          {"name":"x1",   "direction":"in"},
5          {"name":"y",    "direction":"out"}
6      ],
7      "configBits":
8      [   {"name":"c",    "size":4}   ],
9      "functions":
10     [   "[ x0,  x1, !c_0,  y]",
11         "[ x0,  x1,  c_0, !y]",
12         "[ x0, !x1, !c_1,  y]",
13         "[ x0, !x1,  c_1, !y]",
14         "[!x0,  x1, !c_2,  y]",
15         "[!x0,  x1,  c_2, !y]",
16         "[!x0, !x1, !c_3,  y]",
17         "[!x0, !x1,  c_3, !y]"
18     ]
19 }
```

03 User Interaction and Interface

- Disallow connecting output ports
- Draw connections in signal flow direction
- No mixed-direction (in-out) ports
- Visual semantics of ports as adapters/adaptees
- Automated generation of circuit symbols

03 Specifying Functional Behaviour

- Required for CBDs and target functions
- Different notation styles possible

Operator	Type	Symbol			Mnemonic
Negation	prefix	~	*or*	!	`not`
Conjunction	infix	&	*or*	*	`and`
Disjunction	infix	|	*or*	+	`or`
Negated Conjunction	infix		*(none)*		`nand`
Negated Disjunction	infix		*(none)*		`nor`
Exclusive Disjunction	infix		^		`xor`
Negated Exclusive Disjunction	infix		*(none)*		`xnor`

CNF is allowed as well.

03 Specifying Functional Behaviour (Examples)

Using mnemonics:

$$a = (b \text{ or } c) \text{ and not } d \tag{12}$$

Using symbolic operators:

$$(x + !y + z) * (x + y + !z) \tag{13}$$

Using CNF-notation:

$$\begin{matrix}[x,\ !y,\ z] \\ [x,\ y,\ !z]\end{matrix} \tag{14}$$

Note

Only non-CNF notation allows the use of parenthesis and the equality operator.

Section 4

Demonstration

04 The Example Task

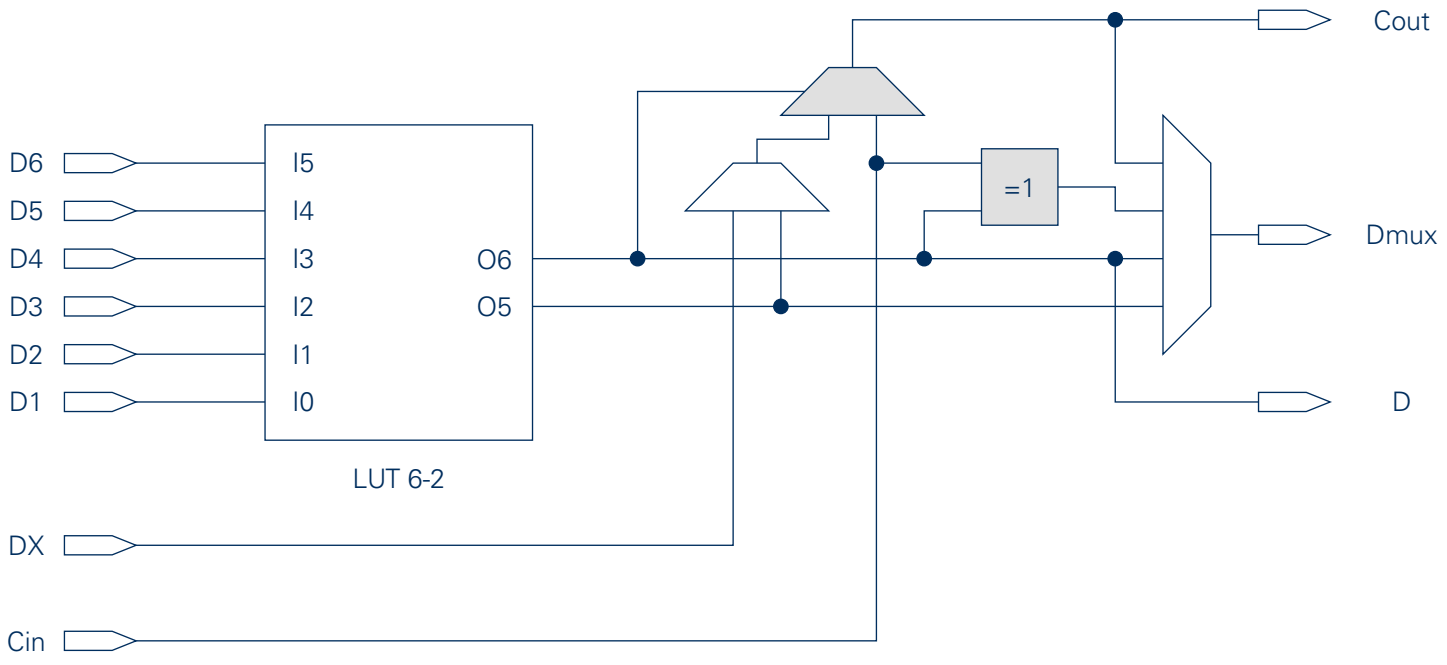
Task

Find the configuration to emulate a full adder on a *Xilinx Virtex 5* CLB slice.

Schematics follow on next slides (*c.f.* [Xili13][Xili12])

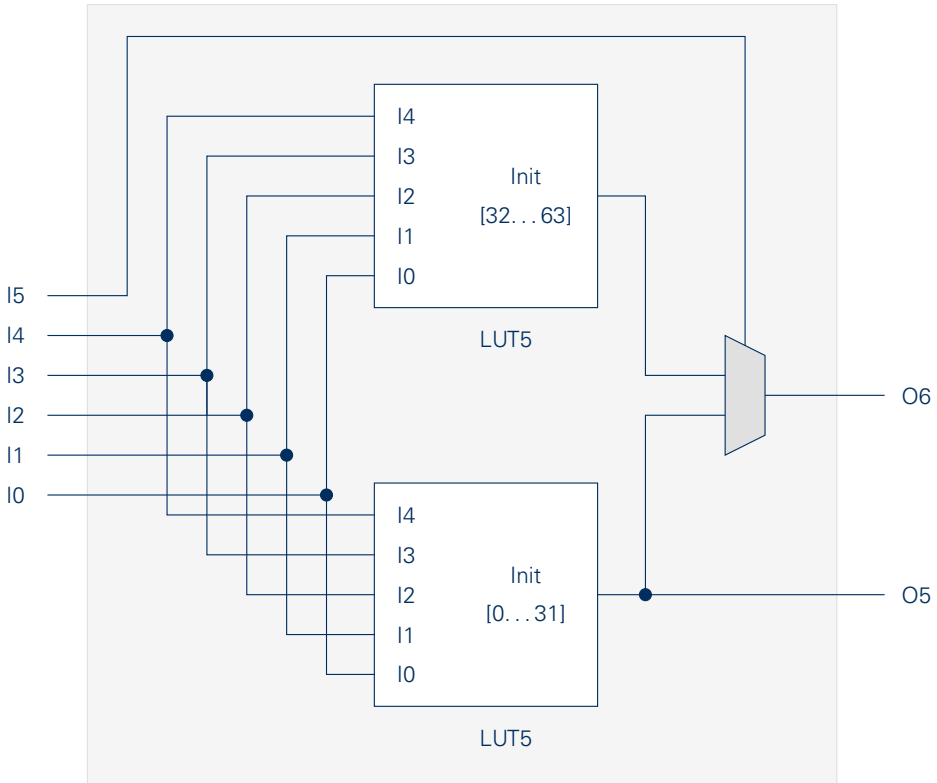
- Use the carry chain (`Cin` and `Cout` respectively)
- Use `D1` and `D2` for the summands
- Use `D` to expose the propagate bit
- Use `Dmux` for the sum

04 Xilinx Virtex 5 (Detail)



White components are configurable.

04 Xilinx LUT6-2



04 Almost There. . .

Note

For the design to work properly, input `D6` needs to be pinned to `true`.

The target formulae are:

```
dmux = (d1 xor d2) xor cin
cout = (d1 and d2) or ((d1 xor d2) and cin)
```

Section 5

Conclusion and Outlook

05 Summary

What was in this talk:

- Presented theoretical approach to the posed problem (c.f.[Lin+07][Saf+06])
- Spoke about decisions regarding the tools' user interaction design
- Provided a workflow example

What more is discussed in the written work:

- Software design
- Implementation details
- Employed visualization techniques

05 Further Ideas

- Allow schematics to be used as components in other schematics
- Support components with generic parameters
- Export configured designs as HDL

Most important

Let's see what the users desire most.

05 The (nearly) last slide

Git Repositories:

- `https://github.com/fer-rum/q2d`
- `https://github.com/fer-rum/q2d-components`

05 References I

Brinson, M.; Jahn, S.:

Qucs: A GPL software package for circuit simulation, compact device modeling and circuit macromodeling from DC to RF and beyond.
`http://www.mos-ak.org/eindhoven/papers/06_Qucs_MOS-AK_Eindhoven.pdf`, April/2008

Erxleben, F.:

The Hardware Design Toolchain - Approaches and State of the Art .
`users.ifsr.de/~fredo/writings/HardwareDesignToolchain.pdf`, 2014

Ling, A. C.; Singh, D. P.; Brown, S. D.:

FPGA PLB Architecture Evaluation and Area Optimization Techniques Using Boolean Satisfiability.
In: IEEE Trans. on CAD of Integrated Circuits and Systems,
Vol. 26 #7 (2007), S. 1196–1210

Narizzano, M.; Pulina, L.; Tacchella, A.:

Report of the Third QBF Solvers Evaluation.
In: JSAT,
Vol. 2 #1-4 (2006), S. 145–164

05 References II

Samulowitz, H.; Bacchus, F.:

Using SAT in QBF.
In: van Beek, P. (Hrsg.): *CP* Band 3709, Springer, 2005
Lecture Notes in Computer Science. –
ISBN 3–540–29238–1, S. 578–592

Safarpour, S.; Veneris, A.; Baeckler, G.; Yuan, R.:

Efficient SAT-based Boolean Matching for FPGA Technology Mapping.
In: *Proceedings of the 43rd Annual Design Automation Conference.*
New York, NY, USA: ACM, 2006
DAC '06. –
ISBN 1–59593–381–6, S. 466–471

Torri, G.; Brinson, M.; Schreuder, F.; Roucaries, B.; Novak, C.; Crozier, R.:

Building a second generation Qucs GPL circuit simulator: package structure, simulation features and compact device modelling capabilities.
`http://www.mos-ak.org/london_2014/presentations/09_Mike_Brinson_MOS-AK_London_2014.pdf`,
März/2014

05 References III


XILINX:
Virtex-5 FPGA User Guide, März/2012


XILINX:
Virtex-5 Libraries Guide for HDL Designs, Oktober/2013