Automatic Hard Block Inference on FPGAs

DSD 2013

Adrian Willenbücher, Klaus Schneider

Embedded Systems Group
University of Kaiserslautern

September 2013

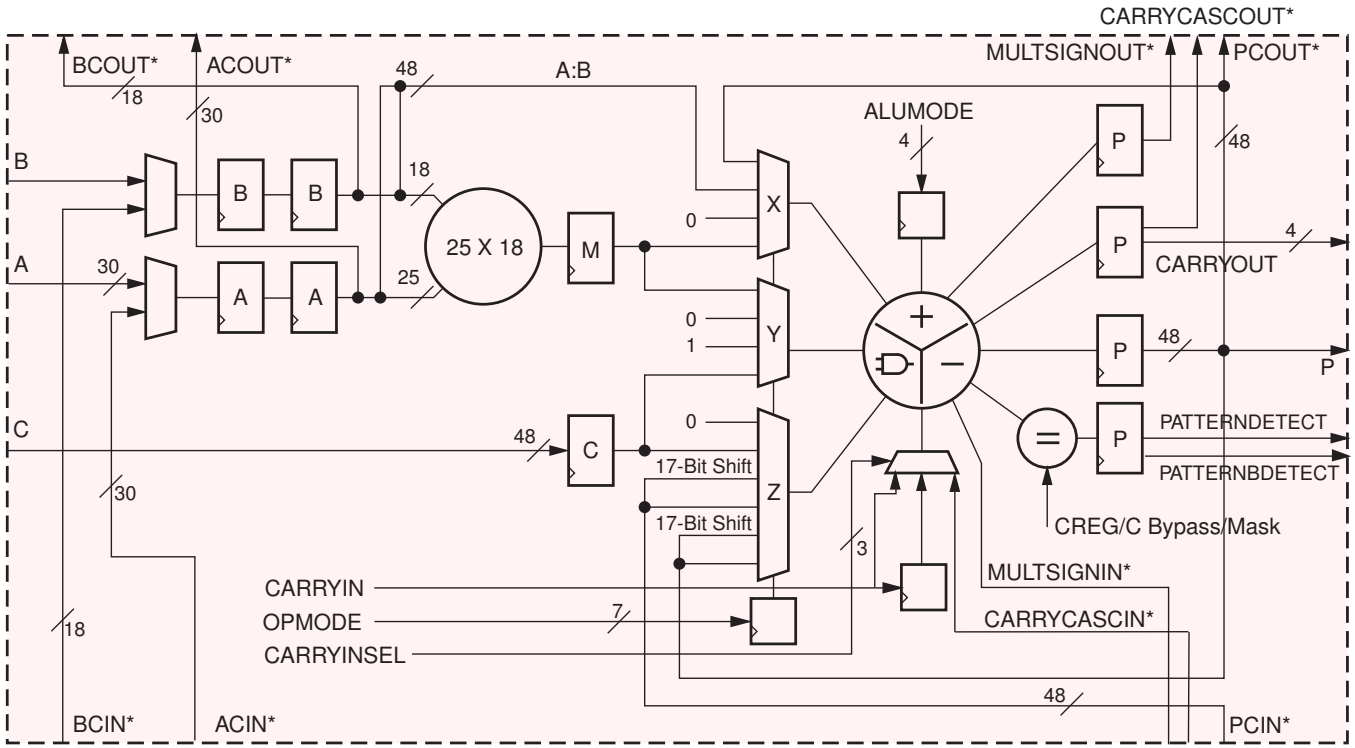
FPGAs

- mainly consist of LUTs and FFs $\Longrightarrow$ flexible
- optimize common operations
- $\Longrightarrow$ non-programmable *hard blocks*
- e.g., DSP elements, block RAM, Ethernet MACs, ...

Hard Blocks

Hard blocks ("hard IP cores"):

- flexible, fast, small, efficient



Problem

Problem:

How to use them?

Previous options:

- manual instantiation: complicated, time-consuming, tied to architecture
- wizards: limited, still need to instantiate and connect result
- automatic instantiation: very limited

Solution

Solution:

Automatically infer an instantiation where desired.

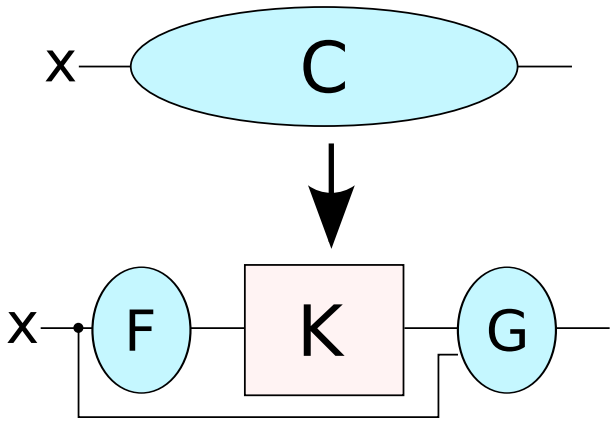
Steps:

1. developer marks part of design to be replaced (in HDL)
2. developer chooses suitable hard block
3. algorithm infers best instantiation
4. *replacement happens "behind the scenes"*

Restriction (for now): combinational circuits only

Approach

- given: circuit C, hard block K
- needed: circuits F and G



$$C(x) \equiv G(x, K(\ F(x)\ ))$$

Approach (cont.)

1. model the circuits symbolically as expressions/terms
2. use unification to find solutions for F and G

Unification:

- given terms u and v
- find substitution σ for variables in u and v
- such that $\sigma(u) \equiv \sigma(v)$

Unification

"Ordinary" unification:

- syntactical equality
- first-order variables

But we need:

- semantical equality: $a \cdot (b + c) \equiv a \cdot b + a \cdot c$
- higher-order variables: $G(x, K(\ F(x)\ ))$

Higher-Order Equational Unification

We need *higher-order equational unification*.

- semantical equality: modeled using set of equations E:
  - $x + (y + z) \equiv (x + y) + z$
  - $x + 0 \equiv x$
- find substitution σ with $\sigma(u) \equiv_E \sigma(v)$
- higher-order variables: required for G:
  - $G \mapsto \lambda\varphi, \psi \,.\, \varphi < a$
  - $G \mapsto \lambda\varphi, \psi \,.\, \varphi + \psi$

Application

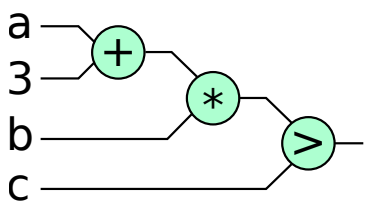
- treat inputs of C as "fresh" constant symbols
- $\Longrightarrow \quad \sigma(C) = C$

Unify $\langle C,\ G(\ K(y)\ )\rangle$

- G represents logic behind hard block
- (G is allowed to refer to inputs of C)
- y represents logic in front of hard block

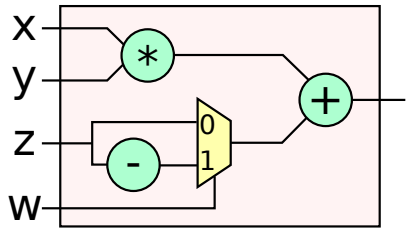
Example

Circuit:

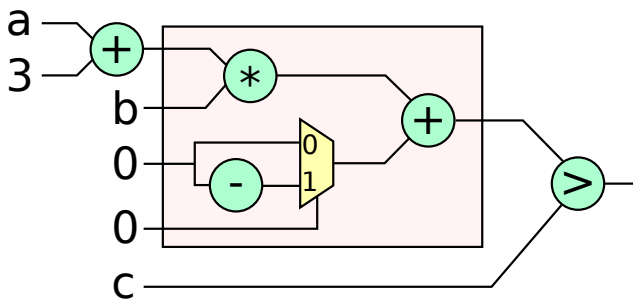


$(a + 3) \cdot b > c$

Hard block:



$G(x \cdot y + (w\ ?\ -z\ :\ z))$



$x \mapsto (a + 3) \quad \ldots \quad G \mapsto \lambda\varphi.(\varphi > c)$

Implementation

Traverse search tree:

- root: initial unification instance
- apply inference rules to nodes $\longrightarrow$ create child nodes
- leaves: solution candidates
- heuristics to trim search tree (see paper)
- automatic evaluation of solutions
- iteratively produces solutions

Setup

Experimental setup:

- hard block: combinational subset of Xilinx Virtex 5 DSP
- tested several circuits
- time budget: 10 seconds (determined empirically)
- measured time until optimal solution found

Results

Circuit	D=1, -Y	D=1, +Y	D=2, -Y	D=2, +Y
$b + c + 1$	0.2	0.2	0.2	0.2
$a\ ?\ (b \cdot c + d) : d$	5.1	5.5	3.5	3.6
$(b \cdot c) + (a\ ?\ d : 0)$	3.4	3.9	2.8	3.1
$a\ ?\ (d + b \cdot c) : (d + -(b \cdot c))$	—	4.6	—	—
$a\ ?\ (-b) \cdot c : b \cdot c$	—	—	—	—
$c + -((b \cdot c \cdot d) + a)$	1.9	2.2	1.8	2.2
$d < (b \cdot c)$	4.1	4.2	2.0	2.2
$b \cdot c \cdot d + e + f$	2.7	3.4	—	—
$a\ ?\ b - c : b + c$	3.2	3.4	—	—

Conclusion

- new method for solving hard block instantiation problem
- based on higher-order equational unification
- often finds best hard block instantiations
- fast enough for interactive use

Future Work

- improve heuristics for trimming search tree
- handle all IP cores, not just hard blocks
- replace combinational circuit with sequential IP core
- replace sequential circuit with sequential IP core

Questions?