TCONMAP: Technology Mapping for Parameterised FPGA Configurations

KAREL HEYSE, BRAHIM AL FARISI, KAREL BRUNEEL, and DIRK STROOBANDT,
Ghent University

Parameterised configurations are FPGA configuration bitstreams in which the bits are defined as functions of user-defined parameters. From a parameterised configuration, it is possible to quickly and efficiently derive specialised, regular configuration bitstreams by evaluating these functions. The specialised bitstreams have different properties and functionality depending on the chosen values of the parameters. The most important application of parameterised configurations is the generation of specialised configuration bitstreams for Dynamic Circuit Specialisation, a technique for optimising circuits at runtime using partial reconfiguration of the FPGA.

Generating and using parameterised configurations requires a new FPGA tool flow. In this article, we present a new technology mapping algorithm for parameterised designs, called TCONMAP, that can be used to produce parameterised configurations in which both the configuration of the logic blocks and routing is a function of the parameters. In our experiments, we demonstrate that in using TCONMAP, the depth and area of the mapped circuit is close to the minimal depth and area attainable. Both Dynamic Circuit Specialisation and fine-grained modular reconfiguration are extracted by TCONMAP from the HDL description of the design requiring only simple parameter annotations.

Categories and Subject Descriptors: B.6.3 [**Logic Design**]: Design Aids—*Automatic synthesis*; J.6 [**Computer Applications**]: Computer-Aided Engineering—*Computer-aided design (CAD)*

General Terms: Algorithms, Design

Additional Key Words and Phrases: FPGA, runtime reconfiguration, dynamic circuit specialisation, technology mapping

ACM Reference Format:
Karel Heyse, Brahim Al Farisi, Karel Bruneel, and Dirk Stroobandt. 2015. TCONMAP: Technology mapping for parameterised FPGA configurations. ACM Trans. Des. Autom. Electron. Syst. 20, 4, Article 48 (September 2015), 27 pages.
DOI: http://dx.doi.org/10.1145/2751558

1. INTRODUCTION

Parameterised configurations [Bruneel et al. 2011] for FPGAs are configuration bitstreams in which the bits can not only be '0' or '1' but also Boolean functions of user-defined parameters. By evaluating these Boolean functions using different parameter values, it is possible to efficiently derive specialised configuration bitstreams, with different properties and/or functionality, from a single parameterised configuration.

K. Heyse is supported by a Ph.D. grant of the Flemish Fund for Scientific Research (FWO – Vlaanderen). B. Al Farisi is supported by a Ph.D. grant of the Agency for Innovation through Science and Technology in Flanders (IWT). This work was partly supported by the European Commission in the context of the FP7 FASTER project (#287804).
Authors' addresses: K. Heyse (contact author), B. Al Farisi, K. Bruneel, and D. Stroobandt, Ghent University, ELIS Department, Sint-Pietersnieuwstraat 41, 9000 Gent, Belgium; contact author's email: karel.heyse@ugent.be.

1084-4309/2015/09-ART48 $15.00
DOI: http://dx.doi.org/10.1145/2751558

The main advantage of generating multiple configurations by specialising a parameterised configuration instead of compiling each of them from scratch using the conventional FPGA tool flow is the much lower compilation time cost per configuration. Specialising a parameterised configuration requires only the evaluation of its Boolean functions, instead of solving the computationally hard problems, such as placement and routing, that are part of the conventional tool flow.

Parameterised configurations have many potential applications. The most important application is the efficient generation of configuration bitstreams for Dynamic Circuit Specialisation (DCS). With DCS, partial runtime reconfiguration of an FPGA is used to optimise a circuit at runtime for the current conditions. This can improve its performance and cost and increase the functional density of the FPGA [Wirthlin and Hutchings 1997]. A good example is an encryption block that is optimised for a specific encryption key that changes only every few seconds. In Davidson et al. [2011], it is shown that the area of an AES encoder can be reduced by up to 20% by using DCS to specialise the circuit for the key in use.

Using the conventional FPGA tool flow to create the specialised configurations for DCS at runtime takes in, many cases, too much time (seconds to hours). An other option is to compute all specialised configurations in advance and store them in a database. However, this quickly becomes infeasible because of the immense number of possible configurations. For example, for a cryptographic block with a key of 128 bits, 2^{128} possible key-specialised configurations have to be prepared and stored. Using a parameterised configuration on the other hand, only one configuration needs to be prepared and stored while the different specialised configurations can be generated in milliseconds during run-time [Bruneel et al. 2011].

Other work such as the GoAhead [Beckhoff et al. 2012], RapidSmith [Lavin et al. 2011], and Torc [Steiner et al. 2011] tool flows can make partial-reconfiguration for DCS easier to perform and more efficient than using the conventional modular approach. They do not, however, make it possible to produce an optimised configuration for a design with many parameters as fast as the parameterised configurations method.

Other examples of possible uses of parameterised configurations are the “hard coding” of constants in a production process in which each produced device contains a different constant value (e.g., a MAC address, an encryption key or calibration data), or speeding up development by allowing the developer to tune coefficients (e.g., the address range of a bus peripheral or the coefficients of a DSP filter), without having to rerun the complete tool flow.

To use parameterised configurations and generate them from HDL, a special FPGA tool flow is needed [Bruneel et al. 2011]. This new tool flow contains, similar to the conventional tool flow, a synthesis, technology mapping, and placement and routing step. An overview of this tool flow is given in Section 3. The new placement and routing step is presented in Vansteenkiste et al. [2014]. In this work, we focus on the technology mapping step of the new FPGA tool flow.

At this time, a mapping algorithm, called TLUTMAP [Bruneel et al. 2011], exists that can be used to create parameterised configurations in which only the configuration of the lookup tables (LUT) of the FPGA can be expressed as functions of the parameters. The TLUTMAP method is limited, however, because it does not allow parameterisation of the configuration of the routing network of the FPGA.

A new algorithm for technology mapping, called TCONMAP, is presented in this article (Section 5) that is aware of the parameterisability of the routing network of the FPGA as well. This will allow the specialised configurations generated from a parameterised configuration to be better optimised for the values of the parameters. Moreover, TCONMAP uses the parameterisability of the routing network to make it possible to use the same LUT resources on the FPGA for different parts of the design between

which can be switched based on the values of the parameters. This is called *resource sharing*. It makes it possible to swap both small and larger parts of a design, such as is conventionally done using modular reconfiguration [Xilinx 2010], with little need for manual redesign. The technique to detect and implement the sharing of LUT resources is the main additional contribution of this work compared to our first TCONMAP algorithm [Heyse et al. 2012]. This makes the technique of parameterised configurations applicable to a whole new class of applications.

Computing a parameterised circuit, which can be used to generate all possible specialised circuits, requires only one pass of the TCONMAP tool. It does not require each specialised circuit to be mapped separately, which is often unfeasible since this requires a number of passes of the technology mapper that increases exponentially with the number of parameters.

Our experiments (Section 6) show parameterised mappings that use on average 46% fewer resources than their generic counterparts. In these experiments, the area and depth of the mapped circuits is reduced to almost the minimum attainable. This minimum is the largest area and depth encountered after constant propagation for any value of the parameters and mapping using the conventional FPGA tool flow. It is dependent on the design and the use of parameters therein.

Our experiments have been extended compared to previous work with larger, more representative applications and applications that demonstrate resource sharing. Moreover, a heuristic version of the TCONMAP algorithm was developed and evaluated for mapping to larger 6-LUTs as commonly found in current commercial FPGAs.

2. FPGA CONFIGURATION

For background, we give a basic overview of a typical FPGA architecture and how FPGAs can be configured and reconfigured during runtime. We discuss a simplified version of current commercial FPGAs.

2.1. FPGA Architecture

FPGAs consist of a grid of configurable logic blocks (CLB), input output blocks (IOB), and a routing network (Figure 1). Typically, an FPGA contains 10,000s of CLBs and hundreds of IOBs.

The input output blocks are connected to the external pins of the chip and thus allow communication with the outside world.

The configurable logic blocks contain a number of lookup tables (LUT) and flip-flops that can be combined to implement any digital circuit (Figure 2(a)). The LUTs implement arbitrary Boolean functions of their inputs using a truth table stored in configuration memory. The truth table contains the desired value for each combination of values of the inputs.

To implement sequential logic, the CLB's flip-flops can be used to store the output of a LUT. A configuration bit determines wether or not a flip-flop is used.

The inputs and outputs of the CLBs can be connected to each other and the IOBs using the FPGA's routing network. This consists of a large amount of wires laid out between the grid of CLBs. Connections between wires and between wires and CLBs or IOBs are made using multiplexers, which are grouped in switch blocks and connection blocks (Figure 2(b)). The CLBs contain crossbars to connect the inputs of the LUTs with the inputs of the CLB or outputs of other LUTs in the same CLB. The state of these multiplexers and crossbars are stored in configuration memory.

The FPGA has special infrastructure and ports to write (and read) this configuration memory. A configuration bitstream can for instance be loaded from an external flash memory or directly from the developers computer via the FPGA's debug port.

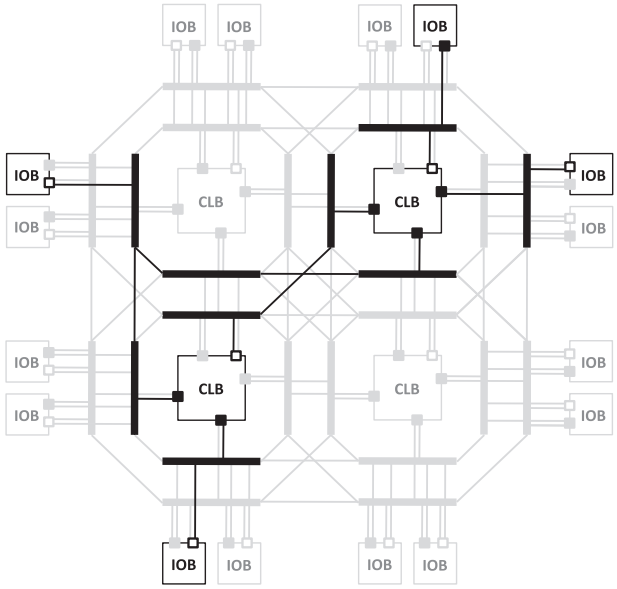
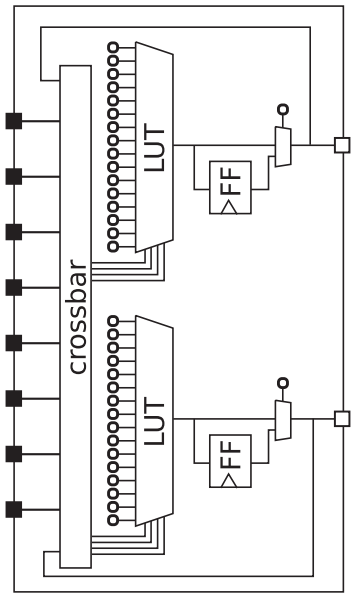
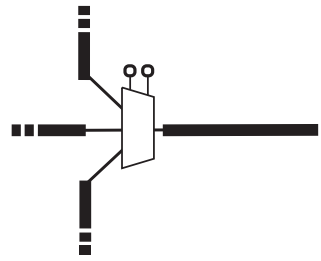


Fig. 1. Schematic of a stripped down FPGA with four CLBs, routing network, and IOBs. The small empty squares represent signal sources, the filled squares represent signal sinks. A small circuit using two CLBs is implemented on the FPGA. Commercial FPGAs can contain 10,000s of CLBs and hundreds of IOBs.



(a) Schematic of a CLB with two LUTs. Configuration memory is shown as small circles.



(b) Schematic of a connection point between wires of the routing network. Configuration memory is shown as small circles.

Fig. 2. Schematic of a CLB and detail of the routing network.

Commercial FPGAs additionally contain heterogeneous resources, such as RAM, hard multipliers, clock management resources, etc., and use more complex CLB architectures with additional features and layers of hierarchy than the simplified CLBs presented here.

2.2. Parameterised Configuration

In a parameterised configuration bitstream, the value of the bits in the configuration memory is described as Boolean functions of the user-defined parameters. The FPGA's configuration memory can only store a logic '1' or '0', so this Boolean function has to be evaluated before the FPGA can be configured.

Parameterised configurations can be used with any existing FPGA architecture and do not require any changes to the hardware.

2.3. Partial Reconfiguration

The configuration infrastructure of modern FPGAs makes it possible to overwrite the configuration memory of a small part of the FPGA, thus changing the functions of the LUTs and connections made by the routing network without affecting the operation of other regions of the FPGA. This is used to perform DCS using parameterised configurations. However, strict constraints apply to how partial reconfiguration can be performed during runtime because of transitional and side effects of overwriting the configuration memory [Xilinx 2010; Altera Corporation 2013].

On Xilinx FPGAs, it is possible to meet these constraints by implementing the DCS circuit as a whole in a modularly reconfigurable region of the FPGA [Xilinx 2010]. The modular reconfiguration method supported by Xilinx was designed to switch between multiple circuits by swapping the configuration of larger regions of the FPGA. Because the method is unaware of how the partial reconfiguration bitstreams are generated it can be used for DCS using parameterised configurations.

Open-source projects such as RapidSmith [Lavin et al. 2011] and Torc [Steiner et al. 2011; Soni et al. 2013] aim to build academic FPGA tool flows, including bitstream generation, for commercial FPGAs, while GoAhead [Beckhoff et al. 2012] provides a platform facilitating partial reconfiguration on Xilinx FPGAs. These can help to overcome the practical barriers which the closed-source nature of the commercial FPGA tool flows may form to the implementation of DCS circuits with TCONs on commercial FPGAs.

3. TOOL FLOW

To generate and use parameterised configurations, a new FPGA tool flow is needed [Bruneel et al. 2011]. The new tool flow, depicted in Figure 3, consists of two stages. The generic stage is performed offline at design time to create a parameterised configuration from an HDL description in which a number of inputs are annotated as parameters (Listing 1). The specialisation stage is performed every time a specialised configuration is needed to (re-)configure an FPGA.

During the specialisation stage, the Boolean functions contained in the parameterised configuration are evaluated to create a fully defined, regular configuration bitstream. This can efficiently be done using an embedded microprocessor, an application specific stack machine [Abouelella et al. 2013], or an external processor.

The generic stage of the new tool flow has to solve a more complex problem than the conventional tool flow. The increased execution time of this offline stage is the trade-off that is made to have an efficient and fast specialisation stage.

The generic stage consists of similar, but adapted, steps as the conventional tool flow: synthesis, technology mapping, packing, placement, and routing. We give a short description of the changes made to each step.

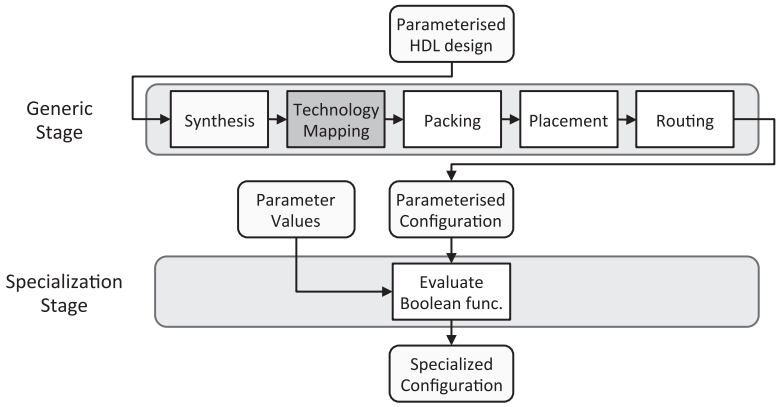


Fig. 3. Tool flow for parameterised configurations.

LISTING 1: Example of a Parameterised HDL Design for a Constant Adder

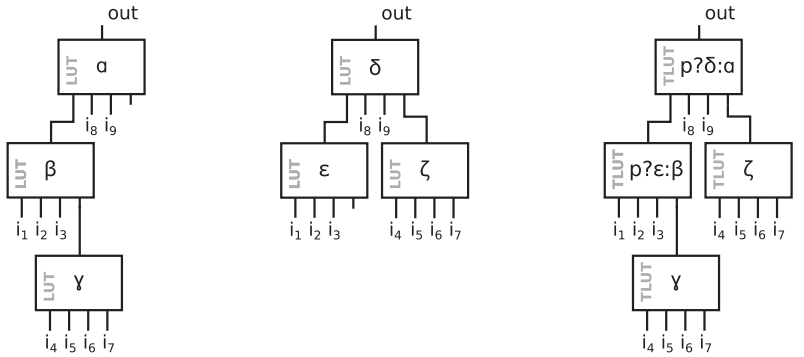
```
entity offset_adder is
generic(
  N : natural
);
port(
  --PARAM
  offset : in std_logic_vector(N-1 downto 0);
  --PARAM
  i : in std_logic_vector(N-1 downto 0);
  o : out std_logic_vector(N-1 downto 0)
);
end offset_adder;

architecture behaviour of offset_adder is
begin
  o <= i + offset;
end behaviour;
```

3.1. Generic Stage

3.1.1. Synthesis. The synthesis step converts an HDL description in which some inputs are annotated as parameters into a Boolean network of simple logic gates (AND, NOT, flip-flop, ...). No significant changes were made to this step because parameters can be synthesised just like regular inputs. Inputs annotated as parameters in the HDL description are annotated as parameters in the resulting Boolean network as well.

3.1.2. Technology Mapping. During technology mapping, the Boolean network generated by the synthesis step is mapped onto the resource primitives available in the target FPGA architecture (e.g., LUTs). The result of the technology mapping step is a netlist describing a circuit of these primitives. An example of a LUT circuit can be found in Figure 4.

The technology mapper tries to minimise the depth (i.e., the number of LUTs on the longest path from input to output) and area (i.e., the total number of LUTs) of this circuit.



(a) Optimal, static LUT circuit for parameter p=0

(b) Optimal, static LUT circuit for parameter p=1

(c) TLUT circuit generated by TLUTMAP

Fig. 4. A TLUT circuit with 4-input LUTs generated by TLUTMAP that uses more LUT resources than the optimal circuits for both values of the parameter p. The function of the LUTs are shown as greek letters, the functions of the TLUTs are a function of p as well. $p?\delta{:}\alpha$ is the ternary operator "δ if p is true, otherwise α."

A conventional mapper will map to static nets and LUTs, thus resulting in a static LUT netlist and configuration bitstream after placement and routing. To generate a parameterised configuration, however, we want to map to LUTs and nets that are parameterised, respectively, called tuneable LUTs (TLUT) and tuneable connections (TCON). The TCONMAP algorithm presented here was developed for this purpose. An overview of the differences of TCONMAP compared to the conventional technology mapper is given in Section 3.2 and an in-depth description of the algorithm is given in Section 5.

3.1.3. Packing, Placement, & Routing. During the packing step, LUT and TLUT primitives from the mapped netlist are clustered into CLBs according to their interconnectivity. During the placement step, physical CLBs on the FPGA are chosen to implement every instance of the CLB primitives in the packed netlist. During the subsequent routing step, routing resources are chosen to implement the nets and TCONs of the netlist.

Because the existing packing, placement, and routing algorithms do not work with the parameterised primitives, extensive work has been done to develop new algorithms that do. Recent research on this subject has been presented [Vansteenkiste et al. 2014]. The algorithms presented in this work do not yet support LUT resource sharing, which is introduced in this article and will need to be developed further.

3.2. Technology Mapping for Parameterised Configurations

The main difference between TCONMAP and the conventional technology mapper is the primitives it maps the Boolean network to. The conventional technology mapper only maps to static LUTs and nets, while TCONMAP maps to parameterised primitives.

3.2.1. Parameterised Primitives. The parameterised primitives used in TCONMAP are the following.

—*Tuneable Connection (TCON).* A TCON is a point-to-point connection which can be made or broken depending on the value of a Boolean function of parameters. It is implemented using only the FPGA's routing network (i.e., the reconfigurable switch blocks and connection blocks) and does not require special hardware support. The parameterised configuration of the used switch and connection blocks will be derived from this Boolean function.

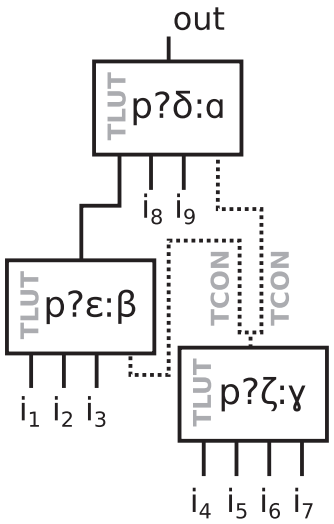


Fig. 5. An optimal TLUT+TCON circuit for Figure 4 generated by TCONMAP, using minimal number of LUT resources.

—*Tuneable LUT (TLUT)*. A TLUT is a lookup table of which the entries are defined as Boolean functions of parameters instead of ones and zeros.

In the TCONMAP algorithm, the output of a TLUT can also be unconnected and therefore unused for some parameter values. When this is the case, the TLUT does not have to be implemented using the FPGA's resources for those values of the parameters. This will allow TLUTs to share LUT resources.

A TLUT is implemented using one physical LUT of the FPGA and also does not require special hardware.

The result of the technology mapping step of the parameterised configurations tool flow is a netlist describing a circuit of LUTs, TLUTs, and TCONs. An example of such a circuit can be found in Figure 5.

The FPGA routing resources used to implement a TCON are only in use when the corresponding Boolean function of the parameters evaluates to true. This means that these resources can be used by the router for multiple TCONs as long as they do not have to be used at the same time.

The same is true for the TLUTs, which can be active or inactive depending on the value of a Boolean function. Two TLUTs that are not active for the same values of the parameters can be placed on the same physical LUT of the FPGA. This we call resource sharing. Because the connections to the physical LUT depend on which TLUT is implemented on it, resource sharing is only possible if the routing of the FPGA can be parameterised.

3.2.2. Comparison of TLUTMAP and TCONMAP. The current mapping algorithm for parameterised designs, TLUTMAP [Bruneel et al. 2011], can map a Boolean network with parameters onto TLUTs but not onto TCONs. Using this mapper, several designs (FIR filter with coefficients as parameters [Bruneel et al. 2011], TCAM with contents as parameters [Bruneel et al. 2011], AES encrypter/decrypter with key as parameter [Davidson et al. 2011], ...) achieved reductions in the number of physical LUTs needed for their implementation of up to 66% and clock speed improvements of up to 38% compared to generic implementations in which the parameters are treated as regular inputs.

The TLUTMAP method, however, is limited because no TCONs are used and the configuration of the routing network of the FPGA can thus not be parameterised. This means that the connections between the TLUTs in the TLUT circuit are constant for all possible values of the parameters and are therefore likely to be suboptimal for many values. This is illustrated in Figure 4 using a design with one parameter input p. Depending on the value of p, the design implements a different function. In this case, the optimal LUT circuit for each of the two possible values of the parameter p is different and the TLUT circuit constructed using TLUTMAP that can be used for both of them is larger than each circuit separately.

TCONMAP will recognise that two LUTs of this TLUT circuit are not active at the same time and will use *resource sharing*. Both TLUTs are then implemented using one LUT resource on the FPGA and the connections to this LUT become dependent on the value of the parameter (Figure 5).

Similarly, when a parameter is used to switch between two or more completely unrelated subcircuits, as is done with modular reconfiguration, TLUTMAP will implement these subcircuits next to each other. This is because the LUT circuits and the connections between the LUTs are almost never the same for unrelated circuits. Only by parameterising the routing network of the FPGA is possible to implement these subcircuits in the same region on the FPGA, sharing the same resources.

Moreover, the fact that TLUTMAP does not use TCONs means that some logic that can be implemented using only the routing network of the FPGA still requires LUT resources. A multiplexer of which the select inputs are parameters—as used in, for example, coarse-grained arrays—is a simple example of such a case. This multiplexer can be implemented very efficiently using TCONs connecting the inputs to the output of the multiplexer. Depending on the value of the select parameters, exactly one connection is closed at a time and implemented using the FPGA's routing network.

The new algorithm for technology mapping, TCONMAP, presented in this article can detect the previously described situations and use the parameterisability of the routing network of the FPGA to implement them as efficiently as possible. This will allow the specialised configurations generated from a parameterised configuration to be better optimised for the values of the parameters and reduce the amount of resources needed.

4. TECHNOLOGY MAPPING BACKGROUND

In this section, we will describe the conventional technology mapping algorithm that consists of three steps (cut enumeration, cut ranking, and node selection) [Chen and Cong 2004; Cong and Ding 1994] and explain a number of concepts used in technology mapping such as Boolean networks and cuts.

4.1. Definitions

4.1.1. Boolean Network. A Boolean network is a directed acyclic graph of which the nodes represent logic gates and the directed edges the wires that connect them. In this work, we use and-inverter graphs (AIG) to represent combinational circuits. AIGs are Boolean networks containing only 2-input AND gates and inverted or noninverted edges. AIGs can be used to represent any combinational circuit. Sequential circuits can also be handled by transforming them into combinational circuits by turning the registers into additional inputs and outputs of the circuit.

4.1.2. Primary Inputs and Outputs. Primary inputs (PI) and primary outputs (PO) are the inputs and outputs of the combinational circuit. A primary input can either be a parameter input or a regular input.

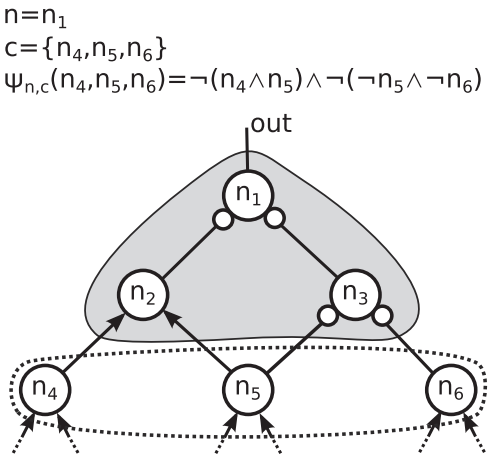


Fig. 6. A section of an AIG with a cut (dotted line) and cone (light grey shape) drawn for root n_1. Large circles represent and-nodes, small circles inverters.

4.1.3. Cut. A cut with a node n as its root is a set of nodes $c = \{n_1, n_2, \dots\}$ called leaves, so that every path from the PIs to the node n passes through at least one of the leaves (Figure 6). The trivial cut is the set $\{n\}$ containing only the root itself.

The local function of a cut $c = \{n_1, n_2, \dots\}$ with root n is the logic function that expresses the output of the root of the cut in function of the leaves: $\psi_{n,c}(n_1, n_2, \dots)$.

The cone corresponding to a cut with root n is the set of all the nodes on the paths from the leaves to the root, including the root and excluding the leaves.

4.2. Conventional Technology Mapping Algorithm

During technology mapping, the goal is to find a covering of the AIG using cones so that every cone can be implemented using one of the FPGA's primitives and the inputs of every cone are available as the outputs of other cones. The result of the conventional technology mapping algorithm is a netlist of LUTs that is functionally equivalent to the input AIG.

Because many different mappings are possible, extra objectives are added. The primary objective of the technology mapping algorithm is to minimise the depth and if possible also the area of the LUT netlist.

TCONMAP has been built based on a widely used conventional mapping algorithm [Chen and Cong 2004; Cong and Ding 1994] consisting of three main steps: cut enumeration, cut ranking, and node selection (Algorithm 2). Many variations of the algorithm exist but the main idea of the algorithm can be described as follows.

4.2.1. Cut Enumeration. A cut is called K-feasible if its local function can be implemented using a K-input LUT. In the first step, all K-feasible cuts $\Phi(n)$ of every node n in the AIG are computed using Equations (1), (2), and (3).

$$\Phi(n) = \begin{cases} \{\{n\}\}, & \text{if } n \in PI, \\ \{\{n_{in_1}\}\}, & \text{if } n \in PO, \\ \{\{n\}\} \cup M(n), & \text{otherwise}, \end{cases} \tag{1}$$

$$M(n) = \{c = c_1 \cup c_2 \mid c_i \in \Phi(n_{in_i}), \theta(n, c)\}, \tag{2}$$

$$\theta(n, c) = |c| \leq K. \tag{3}$$

Table I. Intermediate Results of the Conventional Technology Mapping of the AIG in Figure 7 to 3-input LUTs

Node n	K-feasible cuts with best cut in bold $\Phi(n)$	Depth of best cut $depth(n, bc)$
a_0	$\{\{a_0\}, \mathbf{\{S_0 S_1\}}\}$	1
a_1	$\{\{a_1\}, \mathbf{\{S_0 S_1\}}\}$	1
a_2	$\{\{a_2\}, \mathbf{\{S_0 S_1\}}\}$	1
a_3	$\{\{a_3\}, \mathbf{\{S_0 S_1\}}\}$	1
a_4	$\{\{a_4\}, \{a_0 I_0\}, \mathbf{\{I_0 S_0 S_1\}}\}$	1
a_5	$\{\{a_5\}, \{a_1 I_1\}, \mathbf{\{I_1 S_0 S_1\}}\}$	1
a_6	$\{\{a_6\}, \{a_2 I_2\}, \mathbf{\{I_2 S_0 S_1\}}\}$	1
a_7	$\{\{a_7\}, \{a_3 I_3\}, \mathbf{\{I_3 S_0 S_1\}}\}$	1
a_8	$\{\{a_8\}, \mathbf{\{a_4 a_5\}}, \{a_0 a_5 I_0\}, \{a_1 a_4 I_1\}\}$	2
a_9	$\{\{a_9\}, \mathbf{\{a_6 a_7\}}, \{a_2 a_7 I_2\}, \{a_3 a_6 I_3\}\}$	2
a_{10}	$\{\{a_{10}\}, \{a_8 a_9\}, \mathbf{\{a_4 a_5 a_9\}}, \{a_6 a_7 a_8\}\}$	3
O	$\{\mathbf{\{a_{10}\}}\}$	3

ALGORITHM 2: Overview of the Conventional Technology Mapping Algorithm

```
Input: AIG graph
Output: LUT netlist
for each node in aigGraph.nodesInTopologicalOrder() do
    node.enumerateAllCuts();
end
for each node in aigGraph.nodesInTopologicalOrder() do
    node.selectBestCut();
end
for each node in aigGraph.nodesInReverseTopologicalOrder() do
    node.determineIfSelected();
end
return aigGraph.netlistOfSelectedBestCuts();
```

The function $M(n)$ generates all possibly feasible cuts of n by combining a feasible cut of both its inputs n_{in_1} and n_{in_2}. The function $\theta(n, c)$ evaluates if a cut can be mapped to the K-LUT primitive. Cut enumeration is done for every node in topological order from the PIs to the POs.

In Table I you can find the results of this process applied to the AIG example in Figure 7.

In the cone enumeration process, redundant cuts are generated. A cut c of node n is redundant if some proper subset of c is also a cut of n. These cuts do not contribute to the quality of the mapping and can therefore be removed.

4.2.2. Cut Ranking. For each node in topological order, the cut with the lowest depth is selected as the best cut. This ensures an end result with minimal number of LUTs on the longest path from any output to any input. The depth of a mapped circuit is the maximum depth of all primary outputs.

The *delay* of a nontrivial and non-PO cut is 1, the delay of an LUT.

$$depth(n, c) = \begin{cases} 0, & \text{if } n \in PI, \\ \max\limits_{m \in c} depth(bestCut(m)) & \\ \quad + delay(n, c), & \text{otherwise.} \end{cases} \tag{4}$$

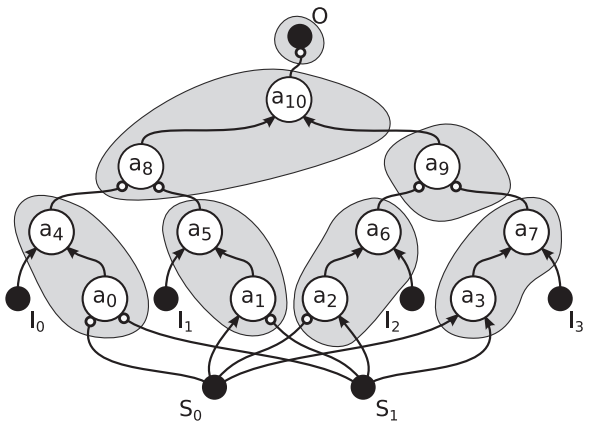


Fig. 7. AIG of a 4-to-1 multiplexer with the resulting cones of conventional technology mapping.

$$bestCut(n) = \begin{cases} \{n\}, & \text{if } n \in PI, \\ \underset{c \in \Phi(n)}{\operatorname{argmin}}\, depth(n, c), & \text{otherwise.} \end{cases} \tag{5}$$

With "$\operatorname{argmin}_a f(a)$" the operator that returns the argument a which minimises the function f.

The results of this process for the example in Figure 7 can also be found in Table I.

4.2.3. Node Selection. This step starts by selecting the POs and recursively selecting all nodes in the best cuts of the previously selected nodes until the PIs are reached (Equation (6)). The local function of the best cut of each selected node, except the POs, will be implemented using a LUT. After this step a depth-optimal mapping has been defined.

$$S = PO \cup \bigcup_{n \in S} bestCut(n). \tag{6}$$

Applied to the example in Table I, this gives $S = \{O, a_{10}, a_4, a_5, a_9, a_6, a_7, I_0, I_1, I_2, I_3, S_0, S_1\}$. The corresponding cones are also shown in Figure 7.

4.2.4. Additional Steps. Additional steps (e.g., based on area and area-flow) can reduce the number of LUTs without increasing the depth of the mapped circuit [Manohararajah et al. 2006].

5. TCONMAP TECHNOLOGY MAPPING ALGORITHM

In this section, we describe the TCONMAP algorithm for technology mapping of parameterised Boolean networks. TCONMAP maps to Tuneable LUTs (TLUT) and Tuneable Connections (TCON) and uses LUT resource sharing to exploit the parameterisability of the routing network of the FPGA. The TCONMAP algorithm is based on the previously described conventional mapping algorithm, but instead of a netlist of LUTs, a netlist of LUTs, TLUTs, and TCONs will be created (Figure 5). The objective of TCONMAP is the same as for the conventional mapping algorithm, that is, to minimise the depth cost (LUT levels) first and if possible also the area cost (LUT resources) of the netlist.In this section, we describe the novel parts of the TCONMAP algorithm.

First, we describe a new definition of feasibility and how to efficiently calculate the feasibility of a cut. Then, we introduce the concept of activation functions, which

are used to detect resource sharing and describe a heuristic to calculate maximum resource sharing. To conclude, we describe the optimisations applied to the algorithm and discuss its computational complexity.

5.1. Feasibility of Cuts with Parameters

Compared to conventional technology mapping, the primitives mapped to are different. Therefore, new definitions of feasibility of a cut and the feasibility evaluation function $\theta(n, c)$ are needed.

5.1.1. TLUT. Parameter signals of a circuit are not implemented as nets in hardware but become part of the Boolean functions of the parameterised configuration. Applied to the case of a cut with parameter inputs amongst its leaves, this means that we do not have to reserve an input for these parameter signals on the physical LUT that will implement the cut. Instead, the parameter inputs will become the variables of the Boolean functions that define the truth table of the Tuneable LUT.

Consequently, every cut with at most K regular nodes and any number of parameter inputs can be mapped to a TLUT and is TLUT-feasible (Equation (7)) [Bruneel et al. 2011].

$$\theta_{TLUT}(n, c) = |c_{regular}| \leq K. \tag{7}$$

The area and delay of a TLUT is the same as for a LUT, that is, 1.

5.1.2. TCON. In a conventional mapping algorithm, nets are not actively mapped to because their function is passive: connecting the LUTs that the mapping algorithm finds. When using TCONs, connections become active elements which can change the output of a circuit in function of the parameters. To exploit this capability we will actively map to TCONs as well.

Because TCONs should not cause shorts for any value of the parameters, a group of TCONs with the same sink can always be thought of as a multiplexer with a Boolean function of parameters at the select inputs, and vice versa. A cut whose local function is the function of such a multiplexer can thus be mapped to TCONs. Moreover, the inputs of the multiplexer can also be inverted or connected to constant '0' or '1'. These inversions or constants are implemented in the LUT or IO block to which the TCON-feasible cut is connected.

Cuts that meet these criteria have to be recognised as TCON-feasible. A method to do this is presented in Section 5.2.

The implementation of a cut using TCONs uses only routing resources and no LUTs, so a TCON-feasible cut does not contribute to the depth and area of the circuit.

5.1.3. TLC Supergate. The TLC (tuneable LUT and connections) is a virtual supergate composed of a TLUT with TCONs, or multiplexers controlled by parameters, attached to its inputs. Supergates are commonly used to reduce the influence of the structure of the Boolean network on the quality of the resulting mapping [Mishchenko et al. 2005]. An illustration of this is given in Figure 8, which contains an example of a circuit that is mapped more efficiently using a TLC than using TLUTs and TCONs separately.

A cut is TLC-feasible if for every combination of parameter values the local function of the cut is defined by at most K nonparameter variables, while the other inputs are don't-cares. This ensures that in every specialised configuration the physical LUT implementing the TLC will need at most K input signals.

Since the area and delay of TCONs is considered to be zero, the area and delay of a TLC is the same as a TLUT or LUT (i.e., 1).

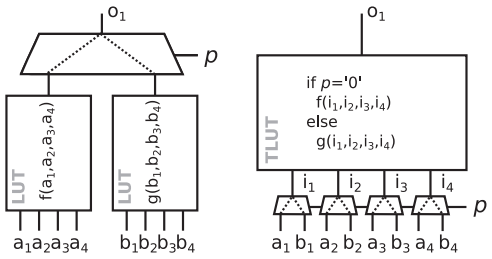


Fig. 8. The same circuit mapped to two TCONs (the multiplexer corresponds to one TCON-feasible cone) and two LUTs (left) vs. one TLC (right) (parameter p).

When a TLC-feasible cone is found, it is split into a TLUT and several TCONs so that the output of technology mapping does not contain TLC primitives and the new TLUTs can also be used for resource sharing.

5.2. Feasibility Calculation using BDD of Local Function

A method to compute the feasibility of a cut according to the previously defined criteria is needed for the cut enumeration step of the mapping algorithm. We propose an efficient way to evaluate the feasibility of a cut using the binary decision diagram (BDD) of its local function. A BDD is a Boolean network representation of a Boolean function using only 2-to-1 multiplexers with the variables of the Boolean function as select inputs, and the constant nodes '0' and '1' [Meinel and Theobald 1998].

To calculate the feasibility of a cut, the variables in the BDD of the local function are reordered [Java 2013] so that the multiplexers controlled by parameter inputs are closest to the root. The BDD is then cut in two so that all multiplexers controlled by parameter inputs lie at one side of the boundary while all other multiplexers lie at the other side. Every connection that crosses this boundary is the root of the (sub-)BDD of a partially evaluated or specialised local function for some values of the parameters.

Based on the criteria in Section 5.1, a cut can then be classified as follows.

—If every sub-BDD contains at most one variable, the cut is recognised as a multiplexer and is TCON-feasible.
—If the full local function depends on at most K different nonparameter variables in total, the cut is TLUT-feasible. This is largely equivalent to Equation (7).
—If every sub-BDD contains at most K different variables, then the cut is TLC-feasible.
—If any of the sub-BDDs contains more than K different variables, the cut is infeasible.

Example. Figure 9 contains the BDD of the local function, ψ, of a cut. The roots of the specialised local functions' sub-BDDs are marked with $\psi_{p=-,q=-}$. The specialised local functions $\psi_{p=0,q=0}$, $\psi_{p=0,q=1}$ and $\psi_{p=1,q=0}$ depend on variables a and b. Function $\psi_{p=1,q=1}$ depends on the variables a and c. The cut with local function ψ can thus be implemented using a tuneable 2-LUT with a at its first input and two TCONs connecting either b or c to its second input depending on the values of p and q.

Because a BDD is a compressed representation of the local function, the number of sub-BDDs that has to be checked is usually far less than the number of possible configurations (2^m with m the number of parameters of the local function). Moreover, classifying a cut can even be done in approximately linear time in the size of the BDD of the cut by further exploiting the compressed form of the sub-BDDs and the occurrence of subgraphs shared between sub-BDDs. This makes the binary decision diagrams very efficient for feasibility calculation with the newly defined criteria.

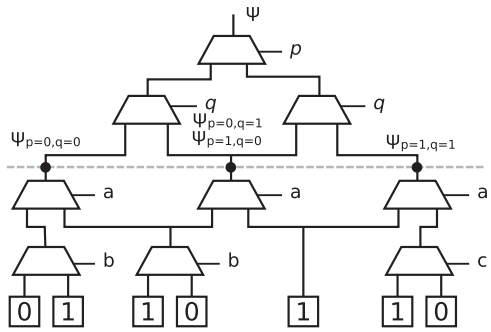


Fig. 9. BDD for the local function of a cut with the specialised sub-BDDs marked (parameters p, q).

5.3. Activation Function

Sometimes the output of a TLUT is not used anywhere in the specialised circuit for some values of the parameters. This happens for instance when the output of a TLUT is connected using a single TCON. The TLUT is then clearly only in use when the TCON connection is made. Consequently, the TLUT only has to be implemented on the FPGA when this condition is met.

The function of the parameters that is true for the values for which the output of a TLUT is used is called its *activation function*. The activation function of a node of an AIG is analogously defined.

5.3.1. Activation Function Uses. This activation function is used in several ways. First, it is used to detect pairs of TLUTs that are not active at the same time for any of the parameter values. This is done by performing the logic AND operation on their activation functions and comparing the result to the false-function. For such a pair of TLUTs, the same LUT resource on the FPGA can be shared to implement them. This can result in considerable area savings compared to TLUTMAP. The situation described in Section 3.2.2 and illustrated in Figure 4 is resolved in this way.

Second, the activation function of a node limits the values of the parameters for which the feasibility of the cuts have to be checked (e.g., $\psi_{p=1,q=0}$ can have more than K variables if the activation function is false for $p = 1, q = 0$). Using this information, cuts that were previously infeasible can become feasible and improve the mapping.

Third, some nodes may have an activation function equal to 'false'. The reason for this is that parameter inputs are constant during operation of the circuit, but the conventional synthesis tools used still treat them as signals whose values may change. As a result, some logic may be instantiated that is only used when there is a transition of one parameter value to another. During technology mapping, these nodes will get activation function 'false' and can be removed from the AIG.

5.3.2. Activation Function Calculation. The activation function, $\gamma_n(p_1, p_2, \dots)$, of a node, n, is a Boolean function of the parameters, $\{p_1, p_2, \dots\}$ that is true for the parameter values for which one of the outputs of the circuit depends on that node.

The calculation of the activation function starts with the calculation of the deactivation functions $\alpha_n(p_1, p_2, \dots)$ and $\beta_n(p_1, p_2, \dots)$. α_n and β_n are true for the values of the parameters for which the output of n is, respectively, constant '1' and '0' and thus does not depend on any regular input of the circuit. α_n and β_n are computed using Equations (8), (9), (10), and (11). The nodes are traversed in topological order.

$$\alpha_n = \begin{cases} n, & \text{if } n \text{ is parameter input,} \\ false, & \text{if } n \text{ is regular input,} \\ \bigwedge_{e \in \, fan\text{-}in(n)} \alpha'_e, & \text{otherwise.} \end{cases} \tag{8}$$

$$\alpha'_e = \begin{cases} \alpha_{source(e)}, & \text{if } e \text{ is non-inverted edge,} \\ \beta_{source(e)}, & \text{if } e \text{ is inverted edge.} \end{cases} \tag{9}$$

$$\beta_n = \begin{cases} \neg n, & \text{if } n \text{ is parameter input,} \\ false, & \text{if } n \text{ is regular input,} \\ \bigvee_{e \in \, fan\text{-}in(n)} \beta'_e, & \text{otherwise.} \end{cases} \tag{10}$$

$$\beta'_e = \begin{cases} \beta_{source(e)}, & \text{if } e \text{ is non-inverted edge,} \\ \alpha_{source(e)}, & \text{if } e \text{ is inverted edge.} \end{cases} \tag{11}$$

The activation function of a node is calculated using Equation (12). The nodes are traversed in reversed topological order.

$$\gamma_n = \begin{cases} true, & \text{if } n \text{ is output,} \\ \bigvee_{e \in \, fan\text{-}out(n)} \gamma_{sink(e)} \wedge \neg\alpha_n \wedge \neg\beta_n, & \text{otherwise.} \end{cases} \tag{12}$$

Because parameters are constant during operation of the circuit, the output of a latch is constant '0' or '1' for the same values of the parameters its input is constant '0' or '1'. The output, therefore, gets the same deactivation functions as the input of the latch. Analogously, the input of the latch gets the same activation function as the output. Because these rules update the activation and deactivation functions of nodes that have already been traversed, the node traversal has to be iterated until the activation and deactivation functions converge. Before the first iteration, the activation and deactivation functions of the latches are initialised to 'false'.

5.4. Resource Sharing Heuristic

The resource sharing heuristic is a step that runs after cone selection to determine which TLUTs with nonoverlapping activation functions to map to the same LUT resource. In this heuristic, we try to maximise resource sharing and therefore minimise the number of LUT resources needed. We use the restriction that every TLUT has to be assigned to a single LUT resource independent of the value of the parameters.

In the future, this step may be combined with the placement step of the FPGA tool flow, where it is possible to take the routability of the design after LUT sharing better into account.

In our current resource sharing heuristic, we first search for the minimal amount of resources needed to implement the design or, in other words, the largest subset of its LUTs that cannot use resource sharing among them. We then find a mapping of the other LUTs to this subset.

To do this, the TLUTs are first grouped into activation sets. Every activation set corresponds to an activation function and contains all the LUTs with that activation function. These activation sets form the nodes of a graph with links between activation sets that have overlapping activation functions and can therefore not share resources. Every clique, or fully connected set of nodes, in this graph corresponds to a group of TLUTs that cannot use resource sharing among them. They do therefore have to be implemented using different resources on the FPGA. The clique that maximises the total number of nodes contained in the selected activation sets gives the maximum number of LUT resources needed. This clique is computed using the Bron-Kerbosch algorithm [Bron and Kerbosch 1973].

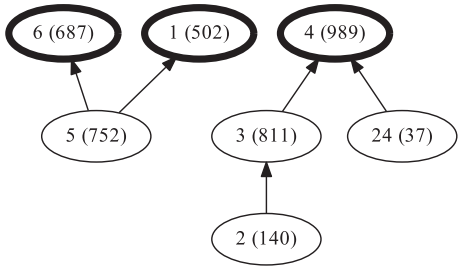


Fig. 10. Example solution of the resource sharing heuristic with the maximum clique sets in bold.

The actual calculation of which activation sets share resources is then done using a greedy algorithm that maps the LUTs of the other activation sets (indirectly) to the LUTs of the maximum clique activation sets. This algorithm is described in Algorithm 3 and an example solution is shown in Figure 10. This solution shows that set 5 with 752 LUTs reuses the resources of sets 6 and 1 that are part of the maximum clique, because either of them does not have 752 LUTs on their own. Sets 3 and 24 use respectively 811 and 37 LUTs of set 4, while set 2 uses 140 resources of set 3, which already reuses resources of set 4.

Although this simple algorithm is not always able to find a mapping that uses the minimal amount of LUTs, the graphs encountered in our experiments were mapped optimally. The development of better resource sharing heuristics is part of our future work.

5.5. Cut Enumeration Optimisation

Cut enumeration can be made significantly more efficient by removing cuts that contribute little or nothing to the quality of the mapping.

It has been proven that any cut containing a node with only parameter inputs in its transitive fan-in is redundant and does not contribute to the quality of the mapping [Bruneel et al. 2011]. If such a node is part of a feasible cut, another feasible cut exists in which the node is replaced by its transitive fan-in. The area and depth of this cut will never be larger. Consequently the final mapping will never be worse if these cuts are removed during enumeration.

Experimentally we have found that the nodes with a large number of feasible cuts are often part of large multiplexers. The reason for this is that the AIG of a multiplexer can easily be mapped to a single large TCON cut but also to many other possible combinations of smaller TCON cuts. Just as a large multiplexer can be constructed in many different ways using combinations of smaller multiplexers.

To avoid enumerating all these redundant TCON cuts, the heuristic assumption is made that every node that has at least one TCON cut and is not connected to a primary output is not to be used as a leaf of another cut. This means that the node should not be selected during the node selection step. This is not a problem because these TCON cuts will always result in more feasible cuts during cut enumeration of the fanout nodes. They will be merged with either the TLUT cuts of the outputs of the node, forming TLCs, or with the TCON cuts of the outputs of the node, forming larger TCONs.

To avoid enumerating cuts containing these described types of nodes (*nonLeafNode*), it is sufficient to prevent the trivial cut of these nodes from being added to the list of

ALGORITHM 3: Algorithm to Compute Maximum Resource Sharing between Activation Sets

```
Input: graph of activationsets
Output: resource sharing between activationsets defined
maximum_clique = computeMaximumClique(graph);
for each set in graph do
    set.LUTs_to_reuse = 0;
    set.LUTs_to_map = set.num_LUTs;
end
for each set in maximum_clique do
    set.LUTs_to_reuse = set.num_LUTs;
    set.LUTs_to_map = 0;
end
todo = sets of maximum_clique
while not todo.empty() do
    // Select parent and child activation sets
    parent = max(todo, key=LUTs_to_reuse);
    child = max(parent.can_reuse_resources_of_sets,
          key=LUTs_to_map);
    if child == None or child.LUTs_to_map == 0 then
        todo.remove(parent);
        continue;
    end
    // Child set reuses as many LUT resources from parent set as possible
    if parent.LUTs_to_reuse ≥ child.LUTs_to_map then
        parent.LUTs_to_reuse.subtract(child.LUTs_to_map);
        child.LUTs_to_map = 0;
        child.LUTs_to_reuse = child.num_LUTs;
        // Once child set is completely mapped, it can become a parent set for other activation
            sets
        todo.add(child);
    else
        child.LUTs_to_map.subtract(parent.LUTs_to_reuse);
        parent.LUTs_to_reuse = 0;
    end
    if parent.LUTs_to_reuse == 0 then
        todo.remove(parent);
    end
    // Grandchildren reuse resources of the child and indirectly of the parent too
    child.can_reuse_resources_of_sets =
          child.can_reuse_resources_of_sets ∩
          parent.can_reuse_resources_of_sets;
end
```

their feasible cuts (Equations (13)).

$$\Phi(n) = \begin{cases} \{\{n\}\}, & \text{if } n \in PI, \\ \{\{n_{in_1}\}\}, & \text{if } n \in PO, \\ \{\{n\}\} \cup M(n), & \text{if } n \notin nonLeafNode, \\ M(n), & \text{if } n \in nonLeafNode. \end{cases} \tag{13}$$

Additionally, the following experimentally defined heuristic optimisation was made: per node, a maximum of 2,000 cuts are considered and the 1,000 best feasible cuts are retained. This limit is only reached for a small number of nodes.

5.6. Computational Complexity

For the parts of the design that do not depend on parameters, the algorithm reduces to the conventional technology mapping algorithm enumerating cuts of at most K nodes.

For the parts that do depend on parameters, the new definition of feasibility allows for cuts with more than K nodes, depending on the local function of the cuts. It is known that enumerating *all* cuts with at most x nodes quickly becomes infeasible for large values of x. In this algorithm, however, we only enumerate some cuts with a larger number of nodes. The number of feasible cuts depends on the design and how parameters are used in the design. In addition to the longer runtime to enumerate the larger amount of feasible cuts, proportionally more memory is required to store the cuts. The memory requirement per cut is approximately proportional to the number of nodes in the cut.

The worst-case size of a BDD, and thus the worst-case execution time of the operations on the BDD, is exponential in the number of variables [Meinel and Theobald 1998]. State-of-the-art BDD packages do however perform optimisations, such as variable reordering, so that in most cases a better than worst-case solution is found. In our experiments with up to ca. 60,000 LUTs, the number of parameters per local function and activation function is in most cases less than a few tens, so they can be represented efficiently using BDDs.

The memory requirements for storing the BDDs is proportional to the sum of the sizes of the BDDs in use at any time. The memory consumption is reduced by freeing the BDDs during cut enumeration as soon as they are not needed anymore to produce cuts of output nodes.

Experimentally, we found that for the evaluated designs the size of the BDDs is on average 17 nodes, the average number of cuts increases with 62%, and the execution time increases fivefold (Section 6.6). This speed difference is the cost of TCONMAP's capability to produce mapped circuits for fully parameterised configurations that can very efficiently be turned into specialised configurations (at runtime).

6. EXPERIMENTAL RESULTS

In this section we will illustrate, using a number of experiments, that TCONMAP can be used to create parameterised configurations that take into account and exploit the reconfigurable properties of the routing network of the FPGA. We show that the resulting mapped designs have a lower depth and require less resources than their generic and TLUTMAPed counterparts.

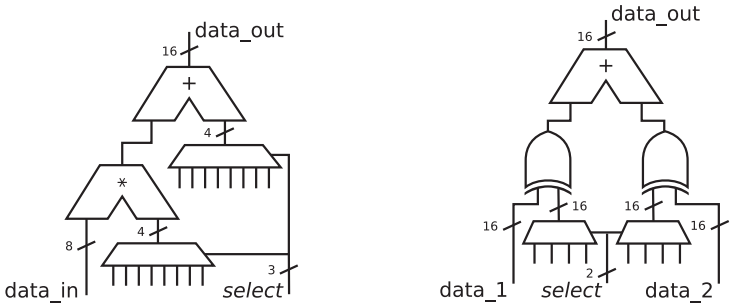
6.1. Implementation

An implementation of the conventional, TLUTMAP [Ghent University 2012] and TCONMAP technology mapping algorithms have been made in Java and are available at `https://github.com/UGent-HES/tcon_flow`. The implementations use additional area recovery steps after the initial mapping similar to the ones used in *fpga* in ABC [Berkeley Logic Synthesis and Verification Group 2007] and described in Manohararajah et al. [2006].

For the binary decision diagrams, the Java port of BuDDy [Buddy 2014] included in JavaBDD [Java 2013] is used.

6.2. Execution Environment

The experiments were performed on an Intel Core i7-3770 CPU (4 cores, clock speed 3.40GHz). A maximum of 1GB of memory was assigned to the Java processes. Java version "OpenJDK Runtime Environment (IcedTea6 1.13.4)" was used.



(a) Schematic of *multiplyAccumulate*.

(b) Schematic of *maskedAddition*.

Fig. 11. Schematics of *multiplyAccumulate* and *maskedAddition*. *select* is the only parameter of these designs. The other inputs of the multiplexers are random constants.

6.3. The Evaluated Designs

A 16-to-16 *crossbar* design was evaluated that can be implemented using only TCONs. The control signals of the crossbar were chosen as parameters.

To highlight the value of TLC supergates, two simple multimode datapaths are used in the experiments. The first design, *multiplyAccumulate*, performs a multiplication of an 8-bit input and adds an integer to this (Figure 11(a)). The multiplication coefficient and added integer are determined by the mode of the datapath. Three parameters define which of the eight modes is active. This module can be used in a larger design to perform offset and scaling adjustment of input signals based on a calibration procedure. The second design, *maskedAddition*, takes two 16-bit inputs and masks them using a bit pattern (Figure 11(b)). The resulting values are combined using an adder. The bit patterns are defined by the mode of the datapath. The active mode is selected from four possible modes using two parameters. This module or a similar one could be used as part of a partially reconfigurable application-specific instruction set processor [Koch et al. 2013].

Next to these examples three more complex real-world designs were used. The first design is a regular expression matching processing element to be used in a coarse-grained reconfigurable array (CGRA) [Heyse et al. 2013b]. One processing element can match a character and perform operations such as the union, negation, Kleene closure, etc. A regular expression matching CGRA contains large numbers of these processing elements which can be configured and connected to implement the desired regular expressions. The 25 configuration inputs of the processing element were chosen as parameters. A detailed description of this design can be found in Heyse et al. [2013a]. This design can be mapped to TCONs and TLCs but does not use resource sharing.

The *secretblaze* is an open-source soft-core processor compatible with the Xilinx MicroBlaze ISA [Barthe et al. 2011]. Using the many generics, the processor can be configured in different ways to optimise for area, clock frequency, or performance. This is done, for instance, by switching between a single cycle and pipelined version of the ALU's multiplier or disabling it altogether.

We believe that during development it can be very useful to be able to change the configuration of the generics of a design, such as the *secretblaze*, without having to recompile it. To make this possible, the majority of the generics of the design were turned into parameters. As a result, this design can be mapped to TCONs and TLCs and uses fine-grained resource sharing.

Lastly, we use a dual-mode *crypto* design. In the first mode, the input data is encrypted using the AES algorithm, while in the second, the 3DES algorithm is used

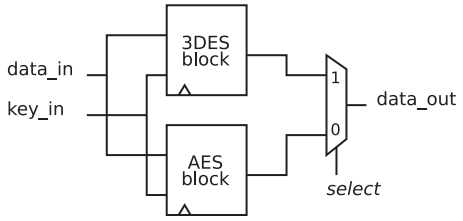


Fig. 12. Schematic of the *crypto* design. *select* is the single parameter of the design.

Table II. Number of 4-(T)LUTs used for Various Mapping Methods (Ratio Relative to LUTMAP in %)

	LUTMAP	TLUTMAP	TCONMAP limited	TCONMAP	TCONMAP sharing	lower bound
crossbar	160 (100)	80 (50)	0 (0)	0 (0)	0 (0)	0 (0)
multiplyAccumulate	92 (100)	49 (53)	44 (48)	32 (35)	24 (26)	22 (24)
maskedAddition	31 (100)	20 (67)	16 (52)	14 (45)	13 (42)	9 (29)
regex	42 (100)	26 (62)	24 (57)	24 (57)	24 (57)	27 (64)
secretblaze	65467 (100)	64676 (99)	64265 (98)	64203 (98)	60655 (93)	60597 (93)
crypto	64864 (100)	64633 (100)	64377 (99)	64375 (99)	60370 (93)	62916 (97)

Table III. Number of 4-(T)LUTs in Longest Path for Various Mapping Methods (Ratio Relative to LUTMAP in %)

	LUTMAP	TLUTMAP	TCONMAP limited	TCONMAP	TCONMAP sharing	lower bound
crossbar	4 (100)	2 (50)	0 (0)	0 (0)	0 (0)	0 (0)
multiplyAccumulate	7 (100)	7 (100)	6 (86)	5 (71)	5 (71)	5 (71)
maskedAddition	3 (100)	2 (67)	2 (67)	1 (33)	1 (33)	1 (33)
regex	5 (100)	4 (80)	4 (80)	4 (80)	4 (80)	4 (80)
secretblaze	60 (100)	59 (98)	59 (98)	59 (98)	59 (98)	59 (98)
crypto	10 (100)	10 (100)	10 (100)	10 (100)	10 (100)	10 (100)

(Figure 12). This design would typically be implemented using modular reconfiguration [Xilinx 2010]. In that case, two separate bitstreams for both modes are created. The bitstreams are then loaded into the configuration memory of that region of the FPGA on demand. In our case, however, one parameterised design is made that combines both modes.

6.4. Measurements for Various Mapping Methods

Table II contains the area and Table III the depth of the designs when mapped using the different mapping algorithms. Table IV contains the respective execution times.

We compared the mapping results of the following algorithms.

—LUTMAP. Conventional mapping to LUTs.
—TLUTMAP. Mapping to TLUTs only [Bruneel et al. 2011].
—TCONMAP limited. Mapping to TLUTs and TCONs but not to TLCs. The limited TCONMAP algorithm maps to TLUTs and TCONs only in order to allow us to evaluate the influence of the TLC supergate on the quality of the mapping.
—TCONMAP. Mapping to TLUTs, TCONs and TLCs.
—TCONMAP sharing. TCONMAP with maximum resource sharing between LUTs.

In the column 'lower bound', we also include the LUTMAP mapping results of the designs after replacing the parameters with constant values and running them through the complete, conventional FPGA tool flow. We chose the constant values that maximised the area and depth of the designs. This area and depth are ideally the best area

Table IV. Execution Time for Various Mapping Methods (Ratio Relative to LUTMAP)

(sec)	LUTMAP	TLUTMAP	TCONMAP limited	TCONMAP	TCONMAP sharing
crossbar	0.25 (1)	0.19 (0.8)	0.88 (3.6)	0.23 (0.9)	0.25 (1.0)
multiplyAccumulate	0.16 (1)	0.18 (1.0)	0.41 (2.2)	0.62 (3.8)	0.62 (3.8)
maskedAddition	0.064 (1)	0.045 (0.7)	0.114 (1.8)	0.112 (1.7)	0.117 (1.8)
regex	0.056 (1)	0.050 (0.9)	0.187 (3.3)	0.217 (3.9)	0.223 (4.0)
secretblaze	7.4 (1)	8.8 (1.2)	101 (13.8)	91 (12.4)	91 (12.4)
crypto	2.9 (1)	3.1 (1.0)	15.3 (5.2)	15.6 (5.3)	15.6 (5.3)

and depth that can be attained by applying TCONMAP to the parameterised design and are the same as would be attained using modular reconfiguration if this were practically feasible. Deviations from this lower bound and a complete analysis of the area and depth results are given in Section 6.5.

6.5. Area and Depth

In this experiment we show that a design that uses parameterised configurations is considerably faster and smaller than a generic implementation mapped using LUTMAP in which the parameters are implemented as regular signals.

For the *crossbar* design, TCONMAP is able to find the optimal mapping using no (T)LUTs at all and zero depth. Although this mapping is obvious, it is important to note that TCONMAP is, to our knowledge, the first mapper that can automatically map functionality to FPGA primitives that can be implemented using only the reconfigurable routing network of the FPGA.

For *multiplyAccumulate*, *maskedAddition*, and *regex*, we see that the area of the designs (TCONMAP sharing) is reduced by more than 43% and the depth of the circuit by more than 20% compared to the conventional implementation. The area is between 5 and 27 percentage points lower than what is attainable using TLUTMAP, and the depth is between 0 and 33 percentage points smaller. The depth of the circuits is equal to the lower bound depth and the area is at most 13 percentage points higher than the minimum area that is attainable using parameterised configurations. In the case of *regex*, the area is even a little bit lower than the largest constant-optimised design, which can in this case be attributed to structural bias of the AIG [Chatterjee et al. 2006].

If we interpret this result for the *regex* processing elements, we see that this means that a parameterised configuration that is aware of the reconfigurable routing network can implement about 8% more of these processing elements on a given FPGA than when only the LUTs can be parameterised. 75% more can be implemented than using a conventional implementation in which the configuration of the processing elements is stored in registers. The functional density [Wirthlin and Hutchings 1997] of the design—defined as $1/(AT)$ with A the area cost and T the total execution time—can increase by more than 75% if the lower depth is translated into a higher maximum clock frequency and, consequently, higher throughput.

In the cases of *multiplyAccumulate* and *maskedAddition*, respectively 13 and 7 percentage points of the area reduction and 15 and 33 percentage points of the depth reduction attained using TCONMAP is the result of the use of the TLC primitive and cannot be found using TCONMAP limited. The use of resource sharing further reduces the number of LUTs with 9 and 3 percentage points but has no influence on the depth. The results of *regex* are not affected by the use of the TLC primitive or resource sharing.

The *secretblaze* design in which the generics are turned into parameters and the *crypto* design benefit most from the sharing of LUT resources but less from the mapping to TCONs and TLCs. This is as expected since many of the generics of the *secretblaze* are

used to switch between different implementations of the components of the processor and the *crypto* design switches between two unrelated circuits. Note, however, that resource sharing is not possible without parameterising the configuration of the routing of the FPGA and will implicitly result in a mapped circuit with TCONs. Resource sharing is the main new contribution of this work compared to our previous work [Heyse et al. 2012]. It makes it possible to considerably reduce the area cost of the *secretblaze* and *crypto* designs.

Remarkable is the fact that the area and depth of *secretblaze* are extremely close to the best possible result. Even though, it is now possible to adapt the values of the generics without having to rerun the complete FPGA tool flow. The area of *crypto* is even smaller than the area of *AES*, the largest of its two modes (*AES*: 62437 LUTs, *3DES*: 4244 LUTs). The reason for this is that the dual-mode *crypto* design is mapped for a single optimal depth, which is equal to that of the design with largest depth: *3DES* (*AES*: 7 LUTs, *3DES*: 10 LUTs). Because this optimal depth is larger than that of the mode with largest area, *AES*, more area recovery can take place which results in a smaller circuit.

Before applying the various mapping algorithms, a tool was used to remove 876 latches from the *secretblaze* design that were only used during parameter transitions (Section 5.3.1).

Previous results that have shown great area and depth reduction using parameterised configurations with TLUTMAP can also be attained using TCONMAP. Examples are the AES encoder (20% fewer LUT resources) [Davidson et al. 2011], FIR filter (39% fewer LUT resources) and TCAM filter (66% fewer LUT resources) [Bruneel et al. 2011].

6.6. Execution Time

The TCONMAP algorithm is more complex and on average 4.7 times slower than the conventional LUTMAP algorithm. The reasons for this speed difference is that on average 9% more cuts per design have to be calculated and that the feasibility calculation using BDDs is significantly slower than the conventional method. The average size of the BDDs of the cuts is just 17.4 nodes. This speed difference is the cost of TCONMAP's capability to produce mapped circuits with minimal area overhead for fully parameterised configurations that can very efficiently be turned into specialised configurations (at run-time). The TCONMAP implementation is relatively new and could likely still be optimised for speed.

The additional execution time of the resource sharing heuristic in TCONMAP sharing is less than 1%.

TCONMAP without TLCs is about 7.9% slower than TCONMAP with TLCs. The reason for this is that the first has to compute and evaluate almost twice as many cuts because some of the optimisations described in Section 5.5 cannot be applied in this case. The average BDD size of the cuts is, however, only 13.2 nodes.

The number of cuts to enumerate and the size of the BDDs, and thus the execution time of the mapping algorithm, is dependent on the design that is being mapped and the use of parameters in it. For instance, when no parameters are used in a design, the number of cuts to enumerate will be the same as for the conventional mapping algorithm and the cuts will contain at most K nodes. On the other hand, a design of the same size with many parameters will have more feasible cuts and the cuts may be larger.

Because of this variability, the runtime increase is smaller for *crypto* (x5.3) than for *secretblaze* (x12.4). The reason for this is that the increase in number of considered cuts is 14% larger for the *secretblaze* design and the average BDD size is twice that of the *crypto* design.

The execution time of the smaller designs can not reliably be compared to that of the larger designs because a significant part of the execution time is spent on procedures that do not scale with the size of the design.

As previously mentioned, if a node has more than 1,000 feasible cuts or more than 2,000 possible cuts, not all cuts will be enumerated which can have a negative influence on the quality of the mapping but limits the execution time. In these experiments this happened for approximately one node in 1,000 of the AIG using the TCONMAP algorithm with TLC support, which is sufficiently low to have minimal impact on the quality of the mapping. However, if only TCONs are allowed and not TLCs, this happens for 2.5% of the nodes of the AIGs.

6.7. Heuristic Technology Mapping for 6-Input LUTs

While older architectures such as the Virtex 2 Pro and Virtex 4 use LUTs with four inputs each, current Xilinx FPGA architectures use LUTs with six inputs. The number of cuts that has to be enumerated to map to 6-LUTs is a lot larger than for 4-LUTs. This makes complete cut enumeration very computing and memory intensive for conventional mapping [Mishchenko and Brayton 2007] and infeasible for TCONMAP. To overcome this problem, heuristic versions of the conventional technology mapping algorithm, such as *priority cuts* and *cut pruning* [Mishchenko and Brayton 2007; Cong et al. 1999] have been developed to find good mappings without enumerating all cuts.

In this work, we have implemented heuristic versions of TCONMAP, TLUTMAP, and LUTMAP based on the *priority cuts* presented in Mishchenko and Brayton [2007]. With this technique, a limited number (8) of good feasible cuts are computed on the fly instead of enumerating all feasible cuts in advance. Several passes are performed to reduce the depth and the area of the mapped circuit, each time computing new feasible cuts that optimise the criteria that are important for that mapping pass. Cut computation is done by merging feasible cuts and checking their feasibility in the same way this was done for the standard technology mapping algorithms.

When mapping to 4-input LUTs using this heuristic algorithm, approximately the same number of resources are used and the average depth is almost equal to that of the standard mapping algorithms. In our implementation, the heuristic version of TCONMAP is on average 2.1 times slower than the non-heuristic algorithms for 4-LUTs. However, it does scale to larger LUTs whereas TCONMAP with complete cut enumeration does not.

The total execution time for mapping the designs to 6-input LUTs using heuristic TCONMAP is only 44% higher than for mapping to 4-input LUTs. Area and depth gains are similar for 6-input LUTs as for 4-input LUTs: The average area gain for the evaluated experiments is 46% for 6-LUTs and 48% for 4-LUTs (Table V). The depth is reduced on average by 39% for 6-LUTs and 36% for 4-LUTs (Table VI). Excluding the outliers *crossbar*, which does not use any resources, and *crypto*, which almost exclusively uses resource sharing, the average area gain for the evaluated experiments is 40% for 6-LUTs and 46% for 4-LUTs. The depth is reduced on average by 34% for 6-LUTs and 29% for 4-LUTs. When looking at the designs separately, we see that some designs, *maskedAddition* in particular, achieve lower area and depth gain using 6-LUTs than 4-LUTs. Others, such as *crypto*, have a larger gain. This variability is expected since the gain for switching from 4-LUTs to 6-LUTs with conventional technology mapping is also not constant for different designs.

The heuristic TCONMAP is on average 7.1 times slower than the heuristic LUTMAP algorithm for 6-input LUTs (Table VII). The reason for this is that LUTMAP does not use BDDs to compute cut feasibility. The average size of the BDDs computed for heuristic TCONMAP is 22.9. All experiments were performed using the same 1GB memory limit used for the 4-LUT experiments.

Table V. Number of 6-(T)LUTs used for Various Heuristic Mapping Methods (Ratio Relative to LUTMAP in %)

	heuristic LUTMAP	heuristic TLUTMAP	heuristic TCONMAP limited	heuristic TCONMAP	heuristic TCONMAP sharing	lower bound
crossbar	80 (100)	80 (100)	0 (0)	0 (0)	0 (0)	0 (0)
multiplyAccumulate	62 (100)	27 (44)	26 (42)	16 (26)	14 (23)	12 (19)
maskedAddition	20 (100)	16 (80)	14 (70)	14 (70)	14 (70)	9 (45)
regex	32 (100)	19 (59)	17 (53)	17 (53)	17 (53)	18 (56)
secretblaze	50491 (100)	50119 (99)	49844 (99)	49821 (99)	47093 (93)	47231 (94)
crypto	16152 (100)	15897 (98)	15629 (97)	15591 (97)	13506 (84)	14631 (91)

Table VI. Number of 6-(T)LUTs in Longest Path for Various Heuristic Mapping Methods (Ratio Relative to LUTMAP in %)

	heuristic LUTMAP	heuristic TLUTMAP	heuristic TCONMAP limited	heuristic TCONMAP	heuristic TCONMAP sharing	lower bound
crossbar	2 (100)	2 (100)	0 (0)	0 (0)	0 (0)	0 (0)
multiplyAccumulate	4 (100)	3 (75)	3 (75)	2 (50)	2 (50)	2 (50)
maskedAddition	2 (100)	1 (50)	1 (50)	1 (50)	1 (50)	1 (50)
regex	3 (100)	3 (100)	2 (67)	2 (67)	2 (67)	2 (67)
secretblaze	35 (100)	35 (98)	35 (98)	34 (97)	34 (97)	34 (97)
crypto	6 (100)	6 (100)	6 (100)	6 (100)	6 (100)	6 (100)

Table VII. Execution Time for Various Heuristic Mapping Methods for 6-LUTs (Ratio Relative to LUTMAP)

(sec)	heuristic LUTMAP	heuristic TLUTMAP	heuristic TCONMAP limited	heuristic TCONMAP	heuristic TCONMAP sharing
crossbar	0.36 (1)	0.46 (1.3)	1.38 (3.9)	0.52 (1.5)	0.53 (1.5)
multiplyAccumulate	0.25 (1)	0.28 (1.1)	0.70 (2.8)	0.64 (2.5)	0.64 (2.6)
maskedAddition	0.16 (1)	0.18 (1.2)	0.36 (2.3)	0.28 (1.8)	0.28 (1.8)
regex	0.14 (1)	0.14 (1.0)	0.50 (3.6)	0.44 (3.2)	0.44 (3.2)
secretblaze	12.6 (1)	12.3 (1.0)	258 (20.5)	329 (26.1)	329 (26.1)
crypto	8.7 (1)	8.6 (1.0)	63.5 (7.3)	63.7 (7.3)	63.8 (7.3)

When limiting cut enumeration for standard TLUTMAP and TCONMAP for 6-LUTs using a cutoff of 2,000 possible cuts per node, cut enumeration is incomplete for 16% to 20% of the nodes of the AIGs for all of the evaluated algorithms. This results in up to 3% higher area cost and 9% higher depth than with the heuristic versions of the algorithms. Moreover, the total execution time of the algorithms is 2.2 to 5.1 times higher, with the largest designs showing the largest speed differences. Clearly, the heuristic algorithms are faster and give better results for large LUTs.

6.8. Direct Synthesis

In some cases TCONs can easily be inferred from, for example, “if” or “case” statements in the HDL description of the design. It is important to note, however, that this is often not the case. The possibility to map parts of a design’s functionality to TCONs can then only be found by analysing the functionality of the design, such as is done by TCONMAP. As an example, the TCONs in the *maskedAddition* design are the result of the bit masking of the input signals and not of multiplexer-like statements in HDL that can be mapped directly to TCONs.

7. CONCLUSION AND FUTURE WORK

TCONMAP is a new technology mapping algorithm that is able to take automatically into account the reconfigurable properties of the routing network of the FPGA and map functionality of a parameterised design to it. The proposed TCONMAP algorithm maps a design to Tuneable LUTs, Tuneable Connections, and TLC supergates, and allows resource sharing of LUTs.

In our experiments, the area and depth of the mapped circuits is reduced to almost the minimum attainable. This results in parameterised mappings that use between 7% and 74% less resources than their generic counterparts and between 6% and 51% less than the existing TLUTMAP algorithm (Excluding outliers *crossbar* and *crypto*). TCONMAP can do this for designs in which the function of the parameters is complex and it is fully automatic and can thus be used in the new tool flow for parameterised configurations. The feasibility of the use of BDDs in the cut enumeration process, the advantage of the TLC supergate and LUT resource-sharing detection have been demonstrated in the experiments. Moreover, a heuristic version of TCONMAP is developed to enable technology mapping for large LUTs and is shown to give similar quality results as the standard TCONMAP algorithm.

Among others, the following improvement could still be made: currently the cost of TCONs is assumed to be zero because TCONs do not contribute to the area and depth of the circuit. However, for the purpose of dynamic circuit specialisation the (time) cost of reconfiguring routing is larger than zero. A more advanced mapping algorithm could try to take this into account. A better resource-sharing heuristic could also be developed that takes into account the routability of the design. This heuristic would preferentially be part of the placement step of the tool flow.

REFERENCES

Fatma Abouelella, Tom Davidson, Wim Meeus, Karel Bruneel, and Dirk Stroobandt. 2013. How to efficiently implement dynamic circuit specialization systems. *ACM Trans. Des. Autom. Electron. Syst.* 18, 3 (2013).

Altera. 2013. Design planning for partial reconfiguration. Altera Corporation.

L. Barthe, L.V. Cargnini, P. Benoit, and L. Torres. 2011. The SecretBlaze: A configurable and cost-effective open-source soft-core processor. In *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW'11)*. 310–313.

Christian Beckhoff, Dirk Koch, and Jim Torresen. 2012. GoAhead: A partial reconfiguration framework. In *Proceedings of the IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 37–44.

Berkeley Logic Synthesis and Verification Group. 2007. ABC: A system for sequential synthesis and verification. http://www.eecs.berkeley.edu/~alanmi/abc.

Coen Bron and Joep Kerbosch. 1973. Algorithm 457: Finding all cliques of an undirected graph. *Commun. ACM* 16, 9 (1973), 575–577.

Karel Bruneel, Wim Heirman, and Dirk Stroobandt. 2011. Dynamic data folding with parameterizable FPGA configurations. *ACM Trans. Des. Autom. Electron. Syst.* 16, 4 (2011), 43:1–43:29.

Buddy. 2014. Binary Decision Diagram Library. http://buddy.sourceforge.net/.

Satrajit Chatterjee, Alan Mishchenko, Robert Brayton, Xinning Wang, and Timothy Kam. 2006. Reducing structural bias in technology mapping. *IEEE Trans. Comput.-Aid. Des. Integ. Circ. Syst.* 25, 12 (2006), 2894–2903.

D. Chen and J. Cong. 2004. DAOmap: A depth-optimal area optimization mapping algorithm for FPGA designs. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*. 752–759.

J. Cong and Y. Ding. 1994. FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs. *IEEE Trans. Comput.-Aid. Des. Integ. Circ. Syst.* 13, 1 (1994), 1–12.

Jason Cong, Chang Wu, and Yuzheng Ding. 1999. Cut ranking and pruning: Enabling a general and efficient FPGA mapping solution. In *Proceedings of the ACM/SIGDA 7th International Symposium on Field Programmable Gate Arrays*. 29–35.

Tom Davidson, Fatma Abouelella, Karel Bruneel, and Dirk Stroobandt. 2011. Dynamic circuit specialisation for key-based encryption algorithms and DNA alignment. *Int. J. Reconfig. Comput.* 2012, Article 716984.

Ghent University. 2012. *The TLUT Tool Flow*. Hardware and Embedded Systems Group, Computer Systems Lab, ELIS Department. https://github.com/UGent-HES/tlut_flow.

Karel Heyse, Karel Bruneel, and Dirk Stroobandt. 2012. Mapping logic to reconfigurable FPGA routing. In *Proceedings of the 22nd International Conference on Field Programmable Logic and Applications*. 315–321.

Karel Heyse, Tom Davidson, Karel Bruneel, and Dirk Stroobandt. 2013a. (Virtual) coarse grained reconfigurable architecture for regular expression matching. Tech. Rep. http://users.elis.ugent.be/~kheyse/tech_report/vcgra_regex.

Karel Heyse, Tom Davidson, Elias Vansteenkiste, Karel Bruneel, and Dirk Stroobandt. 2013b. Efficient implementation of virtual coarse grained reconfigurable arrays on FPGAs. In *Proceedings of the 23rd International Conference on Field Programmable Logic and Applications (FPL'13)*. 1–8.

JavaBDD. 2013. Java Binary Decision Diagram Library, Release 1.0b2. http://javabdd.sourceforge.net.

Dirk Koch, Christian Beckhoff, and Guy G. F. Lemieux. 2013. An efficient FPGA overlay for portable custom instruction set extensions. In *Proceedings of the 23rd International Conference on Field Programmable Logic and Applications (FPL)*.

Christopher Lavin, Marc Padilla, Jaren Lamprecht, Philip Lundrigan, Brent Nelson, and Brad Hutchings. 2011. RapidSmith: Do-it-yourself CAD tools for Xilinx FPGAs. In *Proceedings of the 21st International Conference on Field Programmable Logic and Applications*. 349–355.

Valavan Manohararajah, Stephen D. Brown, and Zvonko G. Vranesic. 2006. Heuristics for area minimization in LUT-based FPGA technology mapping. *IEEE Trans. Comput.-Aid. Des. Integ. Circ. Syst.* 25, 11 (2006), 2331–2340.

C. Meinel and T. Theobald. 1998. *Algorithms and Data Structures in VLSI-Design: OBDD - Foundations and Applications*. Springer-Verlag, Berlin, Heidelberg.

Alan Mishchenko and Robert Brayton. 2007. Combinational and sequential mapping with priority cuts. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. 354–361.

Alan Mishchenko, Satrajit Chatterjee, Robert Brayton, Xinning Wang, and Timothy Kam. 2005. Technology mapping with Boolean matching, supergates and choices. ERL Tech. Rep. EECS Dept., University of California, Berkeley.

Ritesh Kumar Soni, Neil Steiner, and Matthew French. 2013. Open-source bitstream generation. In *Proceedings of the IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 105–112.

Neil Steiner, Aaron Wood, Hamid Shojaei, Jacob Couch, Peter Athanas, and Matthew French. 2011. Torc: Towards an open-source tool flow. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA'11)*. 41.

Elias Vansteenkiste, Brahim Al Farisi, Karel Bruneel, and Dirk Stroobandt. 2014. TPaR: Place and route tools for the dynamic reconfiguration of the FPGA's interconnect network. *IEEE Trans. Comput.-Aid. Des. Integ. Circ. Syst.* 33, 3 (2014), 370–383.

M. J. Wirthlin and B. L. Hutchings. 1997. Improving functional density through run-time constant propagation. In *Proceedings of the 5th ACM International Symposium on Field-Programmable Gate Arrays*. 86–92.

Xilinx. 2010. *Partial Reconfiguration User Guide* Xilinx.

Received October 2014; revised February 2015; accepted March 2015