Transistor Abstraction for the Functional Verification of FPGAs

Guy Dupenloup, Thierry Lemeunier, Roland Mayr
Altera Corporation
101 Innovation Drive
San Jose, CA 95134
1-408-544-8672

{gdupenlo, tlemeuni, rmayr}@altera.com

ABSTRACT

This paper discusses the use of transistor abstraction to enable the functional verification of FPGA fabrics with RTL models. It first describes the multiplexer structures that are used on a massive scale in FPGAs and the specific challenges that they pose to transistor abstraction tools. It then reviews previous approaches and shows that the cone model of the DESB system is particularly well suited to abstract FPGA logic because it makes pass-gate branches in multiplexer structures well apparent. Based on this model, methods are described to isolate multiplexer structures, take into account logic correlation between signals, and generate RTL models that are both simulation efficient and highly readable. Finally, Altera's ABX tool that implements these concepts is briefly described.

Categories and Subject Descriptors

B.5.2 [Register-Transfer-Level Implementation]: Design Aids, *Verification.*

General Terms

Algorithms, Design, Languages, Verification.

Keywords

Cone model, FPGA, functional verification, logic equivalence checking, multiplexer, Register Transfer Level, transistor abstraction.

1. INTRODUCTION

FPGA fabrics are created with building blocks that are assembled in regular patterns. Different types of blocks are used that encapsulate different types of logic, such as look-up tables and registers, dual-port memories, DSP units, etc. Although some of the blocks may use standard cells, the majority of them rely on a full-custom design approach to achieve the highest density possible and deliver as much logic as possible to users. For the most part, the design entry consists of transistor-level schematics and the layout of blocks is handcrafted. Chips are put together using tilers and interconnect is formed by block abutment.

DAC 2006, July 24–28, 2006, San Francisco, California, USA.

In standard-cell flows, the design entry consists of RTL models that are synthesized to obtain a gate-level implementation. Functional verification is primarily based on these RTL models and powerful means are available for that purpose, including high-performance simulators and advanced methodologies such as assertions. In FPGA designs, only switch-level models that are derived from transistor-level schematics are available. Due to their low level of abstraction, such models are inadequate for functional verification. The capacity of switch-level simulators is insufficient for full-chip simulation, run times are prohibitive, and debug is tedious. Therefore, RTL models that abstract the behavior of transistor-level circuits are written manually for functional verification, and are verified as follows:

- A transistor abstraction tool is used to convert a transistor-level netlist of the block, in formats such as SPICE or switch-level Verilog, to a gate-level netlist.

- A logic-equivalence-checking (LEC) tool is used to verify the RTL model against the gate-level netlist produced by the transistor abstraction tool that is used as the golden design.

This paper describes the concepts and techniques that have been implemented in Altera's ABX, a transistor abstraction tool the company uses to enable the functional verification of its FPGA products with RTL models. Multiplexer structures that are used on a massive scale in FPGAs are first presented, and the issues raised by the modeling of contention in such structures are discussed. Previous approaches to transistor abstraction are then reviewed, and the cone model that was first introduced in the DESB system is briefly described [7]. The sections that follow present the modifications and enhancements made to the cone model to isolate multiplexer structures, take into account logic correlation between signals, and generate abstracted models. Finally, the ABX abstraction tool is briefly described, together with directions for future work.

2. MULTIPLEXER STRUCTURES IN FPGAs

FPGAs make massive use of multiplexers to both route signals and implement programmable logic elements such as look-up tables [1]. These multiplexers are usually designed with pass-gate devices, using either NMOS pass-transistors or CMOS pass-gates. For die size optimization purposes, SRAM bits in the configuration memory of the FPGA directly control pass-gate devices with no decoding logic in between.

The multiplexer structures that are used range from simple one-hot multiplexers consisting of a single stage of pass-gates to complex multi-stage structures. Figure 1 shows an example of multiplexer used in routing logic. It has 14 inputs, and uses two pass-gate stages with some sharing of control signals in the first

stage and two sneaky inputs that directly enter the second stage. The output connects to a driver that restores degraded voltage levels when passing a one through the NMOS pass-transistors.

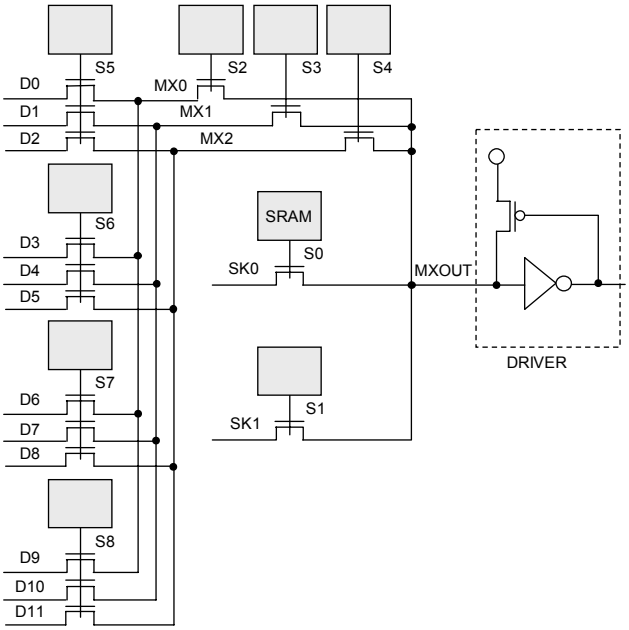

Figure 1. Example of multiplexer used in routing logic

Two key factors must be considered when designing a transistor abstraction tool for FPGAs:

a) An FPGA typically uses a large variety of multiplexer flavors, with different numbers of inputs and different structures. In addition, from one family of products to another and from one product generation to another, different types of structures are used depending on density and performance objectives, and characteristics of the technology used. Therefore, attempting to rely on a catalog of all multiplexer structures that may be used would be unrealistic.

b) Because no decoding logic is used to control pass-gates, multiplexers in FPGAs can take both a high-impedance state and a contention state. Typically, an error in the software that configures the FPGA may produce incorrect SRAM bit settings, eventually resulting in a multiplexer being incorrectly controlled. Therefore, it is essential that high-impedance and contention conditions are properly modeled for all multiplexer structures.

3. MODELING CONTENTION

Let us consider again the multiplexer in Figure 1, and let us examine the condition for input SK0 to conflict with another input (the exclamation mark is used for the NOT operator):

```
X(SK0) =   S0.S1.(SK0.!SK1 + !SK0.SK1)
         + S0.S2.(SK0.!MX0 + !SK0.MX0)
         + S0.S3.(SK0.!MX1 + !SK0.MX1)
         + S0.S4.(SK0.!MX2 + !SK0.MX2)
```

The expression of net MX0 is as follows, with the expressions of nets MX1 and MX2 being similar:

```
MX0  =   ( S5.!S6.!S7.!S8).D0
       + (!S5. S6.!S7.!S8).D3
       + (!S5.!S6. S7.!S8).D6
       + (!S5.!S6.!S7. S8).D9
       + ( S5. S6.!S7.!S8).D0.D3
       + ( S5.!S6. S7.!S8).D0.D6
       + ( S5.!S6.!S7. S8).D0.D9
       + (!S5. S6. S7.!S8).D3.D6
       + (!S5. S6.!S7. S8).D3.D9
       + (!S5.!S6. S7. S8).D6.D9
       + ( S5. S6. S7.!S8).D0.D3.D6
       + ( S5. S6.!S7. S8).D0.D3.D9
       + ( S5.!S6. S7. S8).D0.D6.D9
       + (!S5. S6. S7. S8).D3.D6.D9
       + ( S5. S6. S7. S8).D0.D3.D6.D9
```

As it can be seen from the expressions above, the condition for contention to occur when input SK0 is selected is far from being trivial. Clearly, the global expression of contention that involves all inputs to the multiplexer in Figure 1 is a complex function.

Therefore, modeling contention in multi-stage multiplexer structures in an exact manner would be unrealistic in RTL models. Instead, it is preferable to use simplified models such as the Verilog-HDL model in Figure 2 that relies on a 'default' clause in a 'case' statement to capture all contention conditions. Such a model is highly readable and efficient from a simulation perspective, and represents contention in a conservative manner. As soon as the values of the control inputs of the multiplexer create the conditions for contention to occur, the model generates an 'X' value even if the data inputs that are selected don't have conflicting values. Such pessimism is not an issue in multiplexer structures that are controlled by SRAM bits because all settings that create the conditions for contention to occur are erroneous, and must all be detected in functional verification.

```
casex(s[8:0])
   // High-impedance
   9'b????_00000 : mxout = 1'bz;
   9'b0000_????? : mxout = 1'bz;

   // Sneaky inputs to second stage
   9'b????_00001 : mxout = sk[0];
   9'b0000_???01 : mxout = sk[0];
   9'b????_00010 : mxout = sk[1];
   9'b0000_???10 : mxout = sk[1];

   // First-stage inputs
   9'b0001_00100 : mxout = d[0];
   9'b0001_01000 : mxout = d[1];
   9'b0001_10000 : mxout = d[2];
   9'b0010_00100 : mxout = d[3];
   9'b0010_01000 : mxout = d[4];
   9'b0010_10000 : mxout = d[5];
   9'b0100_00100 : mxout = d[6];
   9'b0100_01000 : mxout = d[7];
   9'b0100_10000 : mxout = d[8];
   9'b1000_00100 : mxout = d[9];
   9'b1000_01000 : mxout = d[10];
   9'b1000_10000 : mxout = d[11];

   // Contention
   default       : mxout = 1'bx;
endcase
```

Figure 2. RTL model for the multiplexer in Figure 1

As a consequence of the approach used in RTL models, the transistor abstraction tool must be able to simplify the expression of contention consistently. This is required to enable the LEC tool to match the abstracted models and the RTL models.

4. PRIOR WORK, THE CONE MODEL

Transistor abstraction used to be popular before standard cells became the dominant methodology, and both pattern matching and generic approaches have been published. In pattern matching methods, transistor structures in the design are matched with predefined patterns and are replaced with library models [2][3]. In generic approaches, the design is first partitioned in sub-circuits using a set of rules, and Boolean expressions that model sub-circuits are then generated [4][5][6][7]. Pattern matching approaches have the advantage of enabling the abstraction of any type of circuits, including analog circuits. However, they require that all transistor structures be inventoried in advance. Generic techniques do not suffer from the same limitation but often apply to certain classes of digital logic only, like static CMOS.

Among all the approaches that have been published, the cone model of the DESB system appears as ideally suited to abstract FPGA logic [7]. In DESB, branches are expanded through transistor channels starting from nets that control the gate of one or several transistors and until power nets are reached. The obtained set of branches is referred to as a "cone" and the net that they influence as a "cone root". The Boolean expressions that model the behavior of the cone root net can be generated from the conditions to turn each branch in the cone on and off. The strengths of such an approach to extract FPGA logic are obvious: the method is generic and does not require that multiplexer structures be inventoried in advance, and the different ways of multiplexers that are built with pass-gates explicitly appear as cone branches or sections of cone branches.

5. ISOLATING MULTIPLEXER STRUCTURES

Before RTL models like the example in Figure 2 can be generated, the different pass-gate branches that build a multiplexer structure must be identified.

This task is complicated by the fact that logic gates may drive the inputs of pass-gate branches. The example in Figure 3 illustrates such a situation. A NAND gate that is built with transistors T2, T3, T4 and T5 drives the pass-gate branch {T0, T1} in the multiplexer structure that is rooted at net MXOUT.

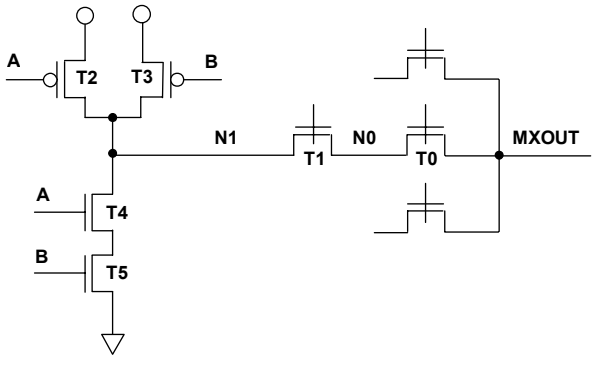

Figure 3. Pass-gate branch driven by a logic gate

When creating cone MXOUT, branches {T0, T1, T2}, {T0, T1, T3} and {T0, T1, T4, T5} will be constructed. Because pass-gates T0 and T1 must be traversed before reaching the output of the NAND gate, they appear at the beginning of all the VCC and VSS branches that include the transistors in the NAND gate.

This observation provides a simple method to identify pass-gate branches in multiplexer structures. VCC and VSS branches are searched for common sections that start from the cone root. Branches are then cut where common sections end, and the trimmed sections are placed in new cones. In the example of Figure 3, branches are cut at net N1. The common section {T0, T1} remains in cone MXOUT, and the trimmed sections {T2}, {T3} and {T4, T5} are placed in a new cone that is rooted at net N1.

6. CONE VARIABLES AND BRANCHES

Let us consider a pass-gate branch in a cone. The input of the branch and the control variables of the cone can be either independent from each other or logically correlated. In the absence of correlation, the input of the pass-gate branch can be set to 0 or 1 regardless of the values of the control variables. In the presence of correlation, this may not be the case. Turning on another branch in the cone may result in the input of the pass-gate branch being set to 0 or 1.

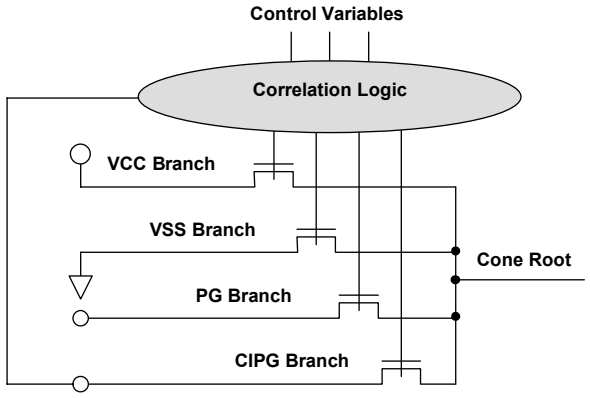

Figure 4. Types of branches in a cone

Therefore, as illustrated in Figure 4, a cone can have four types of branches: VCC branches that end at a logic-one power net, VSS branches that end at a logic-zero power net, pass-gate branches with an input that is logically independent of the control variables of the cone (noted 'PG branches'), and pass-gate branches with an input that is logically correlated to the control variables (noted 'CIPG branches').

The control variables of the cone can be identified as in the YAGLE system [8]. The correlation logic that connects to the gates of transistors in branches is represented by a directed graph, and the "frontier nodes" of the graph are identified. A node is a frontier node if there is no path that goes from one of its ancestors to the root of the graph without passing through it. The "primary nodes", which are the closest frontier nodes to the graph root, are then used as the control variables of the cone.

Note that some PG-branch inputs can be logically correlated with each other. For example, a PG-branch input may be the logical NOT of another PG-branch input. Because a key objective in our approach is to isolate multiplexer structures, such correlation is left outside of our cone model. This potentially results in false contention states being extracted in cones with PG-branch inputs that are correlated with each other due to external logic. In order

to pass LEC, RTL models then have to include these contention states that are unreachable. However, such situations are unlikely to occur in FPGAs where multiplexer structures are mainly used to route logically independent signals and select constant values in look-up tables.

7. CONE EXPRESSIONS, MODEL GENERATION

For each branch in a cone, the conditions ON(Ci) and OFF(Ci) to turn the branch on and off are computed as functions of the control variables Ci. For a CIPG branch, the branch input IN(C_i) depends on the values of the control variables, and the conditions to pass a 0 and a 1 through the branch are computed as follows:

$$P0(Ci) = !IN(Ci) \,.\, ON(Ci)$$

$$P1(Ci) = IN(Ci) \,.\, ON(Ci)$$

Note that IN(Ci) expressions are by-products of the correlation graph that is used to identify the control variables. If the input of a branch is logically correlated to the control variables, it appears as a node in the correlation graph.

Once branch conditions are available, the expressions that model the logic behavior of the cone root net are generated:

- Contention expression (root set to 'X')
- High-impedance expression (root set to 'Z')
- Up expression (root set to 1)
- Down expression (root set to 0)
- Data-input expressions (root set to a PG-branch input).

The contention expression is obtained by forming all pairs of branches in the cone that potentially conflict. Two PG branches can always conflict because their inputs can be independently set to 0 and 1. Because their conflict expression is computed as a function of the control variables only, the type of simplified representation of contention that is used in RTL models is directly obtained. Note that in contrast to PG branches, two CIPG branches only conflict when one of them passes a 0 (P0 condition) and the other one passes a 1 (P1 condition).

To generate a model, the contention and high-impedance expressions are computed first. If they are both identical to 0, the cone is a logic gate such as a NAND gate or a fully decoded multiplexer. In this case, all PG-branch inputs are added to the list of control variables of the cone, and all PG branches are converted to CIPG branches. The 'Up' expression is then computed and is used to model the cone.

If the contention expression is identical to 0 but the high-impedance expression is not, the cone may be a tristate buffer. If it is neither a gate nor a tristate buffer, a model that is similar to the RTL model in Figure 2 is generated, which is a trivial task given the cone expressions that are available.

8. ALTERA's ABX

The concepts and techniques described above have been implemented in Altera's ABX tool that is being used in conjunction with a commercially available LEC tool to develop FPGA families in 65nm technology. ABX is a strategic tool for Altera as it enables the functional verification of the company's FPGA products at the RT level with all associated benefits.

ABX is used to abstract the combinational logic in all building blocks that use a full-custom design approach, including memory blocks. Sequential elements are handled as library cells and don't have to be extracted. Memory blocks in FPGAs are highly configurable using SRAM bits that control the word size and port access modes. ABX is used to abstract the address decoding logic, data multiplexing logic and port control logic, leaving out only the memory core that consists in an array of SRAM cells. Memory cores are verified separately using ad-hoc means and are modeled behaviorally for functional verification.

Building blocks consist in a mix of logic gates and transistors. ABX only extracts transistor-level structures and leaves gate-level structures unchanged. Typically, 3,000 to 15,000 transistors are extracted from a block in a single ABX run. CPU times are in the 2-5 minutes range on Pentium-4 PCs running at 3 GHz under Linux. A large percentage of the time is consumed by logic minimization of the various Boolean expressions involved.

In its current form, ABX only extracts static logic, which has not been a limitation as FPGAs seldom use dynamic logic. The simplified representation of contention as a function of control variables only has caused no issue. ABX could easily be enhanced to provide users with means to request the tool to compute the exact representation of contention for a given list of cones. So far, the need for such an enhancement has not arisen. Similarly, no occurrence of false contention states due to correlation between inputs of PG-branches has been encountered.

ABX is being further developed to raise the level of abstraction of the models it creates. The netlist of cone modules that the tool produces is further processed to identify sets of signals and cells that can be vectorized, and logic elements that can be grouped together in functional units. In the long run, it is expected that ABX will create models that are abstract enough to be used for functional verification, thus eliminating the need for writing RTL models manually.

9. REFERENCES

[1] V. Betz, J. Rose, A. Marquardt, "Architecture and CAD for Deep-Submicron FPGAs", *Kluwer Academic Publishers*.

[2] F. Luellau, T. Hoepken, E. Barke, "A Technology Independent Block Extraction Algorithm", *Design Automation Conference,* 1984.

[3] M. Ohlrich, C. Ebeling, E. Ginting, L. Sather, "SubGemini: Identifying SubCircuits Using a Fast Subgraph Isomorphism Algorithm", *Design Automation Conference*, 1993.

[4] G. Ditlow, W. Donath, A. Ruehli, "Logic Equations for MOSFET Circuits", *International Symposium on Circuits and Systems*, 1983.

[5] R. Bryant, "Boolean Analysis of MOS Circuits", *Transactions on Computer-Aided Design of Integrated Circuits*, Vol. 6, No 4, 1987.

[6] M. Boehner, "LOGEX – An Automatic Logic Extractor from Transistor to Gate Level for CMOS Technology", *Design Automation Conference*, 1988.

[7] M. Laurentin, A. Greiner, R. Marbot, "DESB, a Functional Abstractor for CMOS VLSI Circuits", *European Design Automation Conference*, 1992.

[8] A. Lester, P. Bazargan-Sabet, A. Greiner, "YAGLE, a Second Generation Functional Abstractor for CMOS VLSI Circuits", *International Conf. on Microelectronics*, 1998.

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
http://www.altera.com