A Novel Approach to Minimizing Reconfiguration Cost for LUT-Based FPGAs

Krishna Prasad Raghuraman
Southern Illinois University Carbondale

Haibo Wang
Southern Illinois University Carbondale, haibo@engr.siu.edu

Spyros Tragoudas
Southern Illinois University Carbondale

A Novel Approach to Minimizing Reconfiguration Cost for LUT-Based FPGAs

Krishna Prasad Raghuraman, Haibo Wang, and Spyros Tragoudas
Department of Electrical and Computer Engineering
Southern Illinois University Carbondale, IL 62901

Abstract

This paper proposes a novel approach to reducing the size of FPGA reconfiguration bitstreams by fixing appropriate orders for LUT inputs. With such LUT input orders, memory locations that need to be altered during partial reconfiguration are relocated into common frames. We present a novel problem formulation that relates the number of frames (that need to be downloaded into FPGAs) to the number of minterms of a specially constructed logic function. A heuristic procedure is developed to solve the formulated problem in polynomial time. The proposed methodology is validated by experiments conducted on Xilinx Virtex FPGA platform. Considerable reduction on the size of reconfiguration bitstreams have been observed from our experimental results.

1 Introduction

Implementing reconfigurable hardware using FPGAs is a very active research direction. Quite a few FPGA reconfigurable systems have been developed for real applications. One important concern in FPGA reconfigurable systems is reconfiguration cost, which is normally proportional to the size of reconfiguration bitstreams. Previously, numerous techniques [1, 2, 3, 4, 5, 6, 7] have been presented to reduce FPGA reconfiguration cost at high level. In this work, we address the problem of minimizing reconfiguration data size at logic level. Techniques developed in this work can be combined with previous high level approaches to more efficiently reduce the size of FPGA reconfiguration data.

In many LUT-based FPGAs, configuration data are normally partitioned into frames [8, 9]. A frame is the minimum size of configuration data that can be read or written into FPGAs. This work proposes to permute LUT input orders such that memory bits that need to be altered during a reconfiguration are relocated into some common frames. Consequently, the number of reconfiguration frames is reduced. A novel problem formulation for this optimization problem is proposed in this paper. In addition, an efficient algorithm is developed to solve the formulated problem.

The platform used in our experiment is Xilinx Virtex architecture. Reconfiguration frames for Xilinx Virtex FPGAs are explained in the example depicted in Figure 1. More details can be found in Virtex manuals [8, 9]. As shown in Figure 1, a vertical column of FPGA real estate contains N LUTs, which belong to different CLBs. Because it has four address inputs, each LUT has 16 memory locations. These 16 memory locations of any LUT in the column belong to 16 different frames. In addition, each frame contains N bits, corresponding to the same memory locations in the N LUTs of the column. Since a frame is the smallest portion of configuration data that can be accessed by reconfiguration commands, the entire frame has to be written into the FPGA even if we just want to change a single bit of a LUT during partial reconfiguration. Although this arrangement seems to increase the size of bitstreams during partial reconfiguration, it actually lessens the burden of addressing each memory location. Consequently, it simplifies hardware design and reduces the size of reconfiguration bitstreams.

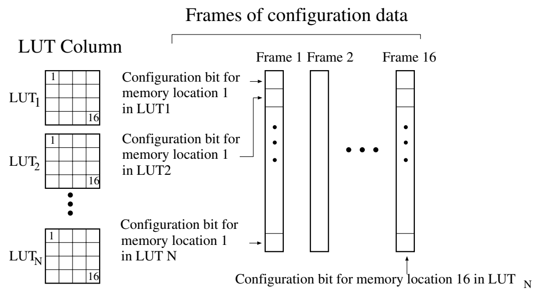
Frames of configuration data
LUT Column
Frame 1
Frame 2
Frame 16
LUT1
LUT2
LUTN
1
16
Configuration bit for memory location 1 in LUT1
Configuration bit for memory location 1 in LUT2
Configuration bit for memory location 1 in LUT N
Configuration bit for memory location 16 in LUT N

Figure 1. Virtex configuration frames.

The rest of the paper is organized as follows. Section 2 explains the proposed approach and describes the problem formulation. Section 3 presents a heuristic algorithm to solve the formulated problem. Section 4 discusses experimental results, and the paper is concluded in Section 5.

2 Preliminaries

The basic idea of the proposed approach is illustrated in the following example. Assume that a column of LUTs contains two LUTs, which are denoted as LUT_1 and LUT_2. Also, we assume that functions implemented in both LUTs are altered during reconfiguration. The original and final functions of LUT_1 are $A \cdot (B+C)$ and $A+B$, respectively. Meanwhile, $A \cdot B + C$ and $(A + B) \cdot C$ are the original and final functions of LUT_2. We use three-input LUTs in this example for a simple and clear demonstration. Also, the example assumes for simplicity in the explanation that the original and final functions of an LUT depend on the same set of variables.

In the first scenario, we assume input orders for both LUTs are $\{A,\ B,\ C\}$. Consequently, contents stored in both LUTs before and after reconfiguration are shown in Figure 2. Labels C_1 and C_2 are used to indicate LUT data before and after reconfiguration. In addition, we use *asterisks* to mark memory locations whose contents are changed during reconfiguration. In this scenario, five frames need to be downloaded into the FPGA. However, if we change the input order for LUT_2 to $\{C,\ A,\ B\}$, we need download only three frames as illustrated in Figure 3.

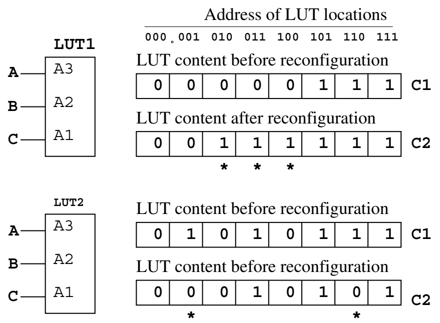

Figure 2. Reconfiguration data before permutation.

Note that the output function of an LUT can be expressed either in terms of its logic input signals ($A,\ B,\ C$ in Figures 2 and 3) or in terms of its address inputs ($A_3,\ A_2,\ A_1$). For the convenience of description, we refer to the function defined in terms of logic input signals as the logic function of the LUT. In addition, we name the function expressed in terms of LUT address inputs as LUT mapping function. For example, LUT_1 in Figure 2 before reconfiguration has logic function $A \cdot (B + C)$ and mapping function $A_3 \cdot (A_2 + A_1)$. The difference between logic function and mapping function is the following. The logic function of an LUT represents a cover of all the minterms of the function it implements. However, the mapping function of an LUT is a cover of all the memory locations that store logic 1. When permuting LUT inputs, we change LUT mapping functions but keep logic functions untouched. For example, LUT_2 has the same logic function in Figure 2 and 3. However, it has different mapping functions ($A_3 \cdot A_2 + A_1$ in Figure 2 and $A_1 \cdot A_2 + A_3$ in Figure 3, both before reconfiguration).

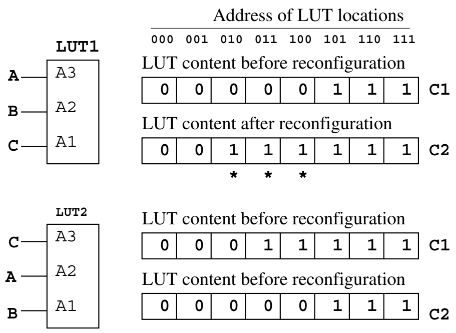

Figure 3. Reconfiguration data after permutation.

The following present formally the studied problem which relates the number of frames (that need to be downloaded into FPGAs during partial reconfiguration) to the number of minterms of a specially constructed logic function. This formulation allows us to take advantage of well-developed function manipulation procedures when tackling the problem of minimizing FPGA reconfiguration bitstreams. The following notations are used in our discussion.

- N is the number of LUTs in one column
- LUT_i indicate the ith LUT in the selected column and $1 \leq i \leq N$
- f_i^1 and f_i^2 represent the mapping functions of LUT_i before and after reconfiguration.

Functions f_i^1 and f_i^2 can be obtained as follows. Assume that two circuits $\mathcal{C}_1$ and $\mathcal{C}_2$ will be implemented on an FPGA. $\mathcal{C}_1$ is the original circuit and $\mathcal{C}_2$ is the circuit derived from $\mathcal{C}_1$ by performing partial reconfiguration. Using any available FPGA design tool, circuits $\mathcal{C}_1$ and $\mathcal{C}_2$ can be separately mapped into the same area of the FPGA layout. For any LUT, *e.g.* LUT_i, located in the mapped area, two logic functions, denoted by $f^1_{LUT_i}$ and $f^2_{LUT_i}$, will be assigned. $f^1_{LUT_i}$ is used in the implementation of $\mathcal{C}_1$ and $f^2_{LUT_i}$ is for $\mathcal{C}_2$. Such assignments build one-to-one relations between LUT address inputs and logic variables on which $f^1_{LUT_i}$ and $f^2_{LUT_i}$ depend. To obtain f^1_i and f^2_i, we can simply substitute the logic variables by their corresponding LUT input addresses in their logic function expressions.

The difference function between f^1_i and f^2_i is expressed as:

$$F_i = f^1_i \oplus f^2_i \tag{1}$$

Note that the minterms of F_i represent memory locations in LUT_i that need to be altered to change the function implemented on LUT_i from $f^1_{LUT_i}$ to $f^2_{LUT_i}$. From the previous discussion, we know that each minterm of F_i will require a frame of configuration data. For a given LUT column, configuration frames that cover all the LUT locations that need to be altered can be calculated by performing union operation for all the difference functions of LUTs in the given column. This is:

$$\mathcal{F} = \bigcup_{i=1}^{N} F_i \tag{2}$$

When performing union operations, address inputs with the same name but located in different LUTs (e.g. A_1 of LUT_i and LUT_j) will be treated as the same variable. This is valid because address inputs in LUT mapping functions serve the same purpose: they are used as coordinates to indicate memory locations that contain logic 1. The final function obtained from the union operation will depend on only four variables $A_4,\ A_3,\ A_2,\ A_1$. Hence, the problem of minimizing reconfiguration data is translated into a problem of finding proper input permutation orders for a set of logic functions (F_i) such that the number of minterms of function $\mathcal{F}$ is minimized.

3 Proposed solution

Using exhaustive enumeration method to solve the above formulated problem will be very time consuming since there are 24^{N-1} possible combinations (assume that a column contains N LUTs; each LUT has four inputs and consequently results in 4! input permutations). To efficiently search optimal LUT input orders, this section presents a search procedure based on greedy algorithm. Its major steps are described in Figure 4. It first constructs LUT difference functions (line 3) and, concurrently, finds the LUT that requires the least number of reconfiguration frames (lines $4 \sim 8$). The input order of the selected LUT will not be permuted, and is used as a reference when permuting other LUT input orders. Also, function *MintermCount* used in line 4 counts the number of minterms of its operand function. After the reference LUT is selected, the algorithm sequentially picks an unprocessed LUT and permutes its inputs. The permutation procedure is sketched from line 12 to 22. It exhaustively tries all the possible permutations and picks the one that results in the smallest increase on the number of minterms of the newly constructed union function ($\mathcal{F}^{tmp}$). The time complexity of the proposed procedure is $24 \cdot (N-1)$, which is significantly smaller than the time complexity of the exhaustive enumeration method.

```
1   min_tmp = 16
2   for i = 1 to N
3       F[i] = $f^1_i \oplus f^2_i$
4       min = MintermCount(F[i])
5       if min < min_tmp)
6           min_tmp = min
7           min_index = i
8           $\mathcal{F}$ = F[i]
9   for i = 1 to N
10      if i $\neq$ min_index)
11          $\mathcal{F}$ = permute($\mathcal{F}$, F[i])

12  permute( $\mathcal{F}$, F[i] ) {
13      min_tmp = 16
14      for each permutation order of $LUT_i$
15          derive new function $F'[i]$ according
            to the new input order
16          $\mathcal{F}^{tmp}$ = $\mathcal{F} \bigcup F'[i]$
17          min = MintermCount( $\mathcal{F}^{tmp}$ )
18          if min < min_tmp)
19              min_tmp = min
20              Order[$LUT_i$] = current permut. order
21              $\mathcal{F}^{min}$ = $\mathcal{F}^{tmp}$
22      return $\mathcal{F}^{min}$ }
```

Figure 4. The proposed search procedure.

4 Experimental results

The proposed search procedure has been implemented on the top of a Binary Decision Diagram (BDD) package [10]. In the experimental flow, we use ISCAS85 circuits as the initial circuits that are implemented on FPGAs before reconfiguration. The hardware platform used in our experiments is Xilinx Virtex 1000 device. In addition, Xilinx ISE design tool is used to map example circuits and generate configuration data.

Due to the lack of suitable partial reconfiguration benchmark circuits, we derive FPGA final circuits, which are to be implemented after partial reconfiguration, by performing random function modification on original circuits. In this process, we first define a set of functions, denoted by $f_1, f_2, \cdots f_i$, which depend on variables $A_4,\ A_3,\ A_2,\ A_1$. Then, we derive the final logic function for a selected LUT by performing either *COMPOSE* or *INTERSECT* operation with using the original LUT function and one function selected from $f_1, f_2, \cdots f_i$ as operands. The *COMPOSE* and *INTERSECT* are function manipulation operations defined in the BDD package. The selection on operation (*COMPOSE* or *INTERSECT*) and operand function ($f_1, f_2, \cdots f_i$) is totally randomized.

For the formed FPGA circuits we generate reconfiguration data using Xilinx ISE design automation tool. Initially, we simply follow the traditional design flow without permuting LUT inputs. Then, we re-generate reconfiguration bits by using the proposed approach. We find the optimal LUT input orders and force the Xilinx design tool to keep such orders during the generation of reconfiguration data. The sizes of reconfiguration bitstreams obtained using the above two approaches are compared in Table 1. It shows that around 15% reduction on the number of frames can be achieved by using the proposed method. In the experiment, we also vary the size of LUT column to provide more case studies.

Table 1. Comparison of Reconfiguration frames.

Circuit Name	No.of LUTs per column	No. of *Frm.* without Permutation	No. of *Frm.* with Permutation	savings (%)
C432	3	274	244	11%
	4	233	212	10%
	8	166	136	18%
C1355	3	142	137	4%
	6	124	111	10%
	9	106	95	10%
C1908	3	255	239	10%
	6	198	175	12%
	9	172	141	18%
C2670	3	430	389	10%
	6	334	286	14%
	9	276	232	16%
C3540	6	771	659	15%
	9	632	506	20%
	12	567	409	28%
C5315	9	769	617	21%
	12	626	574	8%
	15	542	440	19%
C6288	12	1168	964	17%
	15	986	826	16%
	18	852	712	16%
C7552	12	967	780	9%
	15	814	660	19%
	18	693	570	18%

5 Concluding remarks

In this paper, we presented a methodology to reduce the size of reconfiguration bitstreams for LUT-based FPGAs. This is achieved by properly ordering LUT inputs when mapping circuits into FPGAs. Furthermore, the problem of finding such proper orders is formulated and solved using a heuristic algorithm based greedy method. Our approach tackles the problem of minimizing FPGA reconfiguration cost at logic level. To the best of our knowledge, this type of problem was rarely addressed at logic level. Our work demonstrates a new dimension on minimizing FPGA reconfiguration cost. Experimental results show that the size of reconfiguration data can be reduced around 15% by the proposed method alone. In addition, without any compromise, the proposed method can be combined with other techniques that reduce FPGA reconfiguration cost through high-level optimization to further reduce FPGA reconfiguration cost.

References

[1] Zhining Huang and Sharad Malik, "Managing dynamic reconfiguration overhead in systems-on-a-chip design using reconfigurable datapaths and optimized interconnection networks," in *Proceeding of Conference of Design, Automation and Test in Europe*, pp. 13–16, 2001.

[2] Daler Rakhmatov and Sarma B.K. Vrudhula, "Minimizing routing configuration cost in dynamically reconfigurable FPGAs," in *Proceedings of Parallel and Distributed Processing Symposium*, pp. 1481–1488, 2001.

[3] K.Compton, J.Cooley and S.Knol, "Configuration relocation and defragmentation for reconfigurable computing," in *Proc. of FPCCM*, pp. 79–80, 2000.

[4] K.Compton, Z.Li,S.Knol and S.Hauck, "Configuration relocation and defragmentation for reconfigurable computing," vol. 10, pp. 209–220, 2002.

[5] Anna Antola, Vincenzo Piuri, Mariagiovanna Sami, "Online Diagnosis and Reconfiguration of FPGA Systems," in *Proceeding of Electronic Design, Test and Applications*, pp. 291–296, 2002.

[6] Douglas Chang and Malgorzata Marek-Sadowska, "Partitioning Sequential Circuits on Dynamically Reconfigurable FPGAs," *IEEE Transaction on Computers*, vol. 48, no. 6, pp. 565–578, 1999.

[7] S.Dueck and W.Kinsner, "Netlist Partitioning for FPGA-Based Run-Time Reconfiguration," in *Proceedings of 2002 IEEE Canadian Conference on Electrical and Computer Engineering*, pp. 584–590, 2002.

[8] XILINX Inc., *Virtex Series Configuration Architecture User Guide*, 2003.

[9] XILINX Inc., *Two Flows for Partial Reconfiguration:Module Based or Small Bit Manipulations*, 2002.

[10] Fabio Somenzi, "Cudd package." http://vlsi.colorado.edu/ fabio/CUDD/cuddIntro.html.