ELectronics EXpress

LETTER

IEICE Electronics Express, Vol.11, No.4, 1–7

A LUT manipulation based intrinsic evolvable system

Kaifeng Zhang[a)], **Huanzhang Lu, Weidong Hu, and Jian Wang**

ATR Key Lab, National University of Defense Technology, Changsha, Hunan, 410073, P.R.China

a) *zkf0100007@163.com*

Abstract: The paper presents an evolvable system for intrinsic hardware evolution based on look-up-table (LUT) manipulation. We also introduce dynamic routing using multiplexer to improve the flexibility of the system. The proposed approach is implemented on Xilinx ML403 Evaluation Platform, and an evolution of 3-bit multiplier is employed for verification. The experimental results show that more than three orders of evolution speed enhancement over JBits and one order of evolution speed enhancement over bitstream reverse engineering (BRE) based methods is achieved.

Keywords: evolvable hardware, LUT, virtual reconfigurable circuits, bitstream reverse engineering

Classification: Integrated circuits

References

[1] P. Kaufmann, K. Glette, T. Gruber, M. Platzner, J. Torresen and B. Sick: IEEE Trans. Evol. Comput. **17** [1] (2013) 46.

[2] J. Lohn, G. Larchev and R. DeMara: 2003 International Conference on Evolvable System (ICES) (2003) 47.

[3] R. S. Oreifej and R. F. DeMara: Appl. Soft Comput. **12** [8] (2012) 2470.

[4] F. Cancare, M. D. Santambrogio and D. Sciuto: Proc. of 2010 IEEE International Symposium on Circuits and Systems (ISCAS 2010) (2010) 853.

[5] J. B. Note and E. Rannaud: Proc. of 16th International ACM/SIGDA Symposium on Field Programmable Gate Arrays (FPGA 08) (2008) 264.

[6] J. Wang and C. H. Lee: IEICE Electron. Express **6** [3] (2009) 141.

1 Introduction

Evolvable hardware (EHW) refers to hardware that can change its architecture and behavior dynamically and autonomously by interacting with its environment. EHW can be classified into two major categories, extrinsic EHW and intrinsic EHW. The intrinsic EHW principle allows a system to adapt to a changing environment, recover from faulty states, and react to new resource requirements at runtime [1].

An efficient and flexible evolvable platform plays an important role in the research domain of intrinsic EHW. Due to the fact that no commercial evolvable platforms are available, many researchers have been focused on evolvable platform since the birth of EHW. In early times, many intrinsic

DOI: 10.1587/elex.11.20131003
Received December 18, 2013
Accepted January 23, 2014
Publicized January 31, 2014
Copyedited February 25, 2014

evolutions were carried out using JBits [2], which is a Java based bitstream manipulation tool provided by Xilinx. Unfortunately, new devices after Virtex II are not supported by JBits. Moreover, JBits running on Java virtual machine which has the disadvantage of great complexity and high computation cost are not suitable for embedded systems. Thus, a bitstream reverse engineering (BRE) based direct bitstream manipulation approach was proposed in [3, 4]. In comparison with JBits method, the BRE method achieves lower computation cost and higher reconfiguration speed. However, to perform the BRE operation, the bitstream format should be obtained by reverse engineering the bitstream through repetitive trial-and-error experiments. The problem of reverse-engineering the bitstream is that most of these file formats are proprietary. Furthermore, the most important of these processing steps, place and route (PAR), is very slow and not deterministic, making it difficult to reliably generate slightly altered bitstream and observe incremental changes in order to infer the function of the altered locations [5]. A well-known alternative is using virtual reconfigurable circuits (VRC) [6], a reconfigurable layer built on top of the reconfigurable fabric that reduces the complexity of the reconfiguration process, creating a kind of application specific programmable elements. Drawbacks of VRCs are complained about area and delay overheads, as well as high power consumption.

In this paper, we propose a look-up-table (LUT) manipulation based evolvable platform. The LUT manipulation is performed by the ICAP (Internal Configuration Access Port) API provided by Xilinx, where no detailed information of bitstream is needed. Further, All FPGAs of entire Virtex families are fully supported. Unlike that in VRC all possible logic functions are implemented at design time, our method can implement different logic function by changing the LUT contents dynamically at run time. As result, a great reduction of logic resources could be achieved. In addition, dynamic routing is not supported by the conventional BRE based methods. To overcome this limitation, a multiplexer based dynamic routing is employed in our system. In other words, our system tries to take advantage of both the direct bitstream manipulation systems and the VRC-based systems. Experiments of evolving a 3-bit multiplier were conducted to verify the effectiveness of the proposed method. The experimental results show that the proposed method outperforms the existing methods in terms of logic utilization and evolution speed.

2 The proposed LUT based evolvable platform

2.1 LUT based logic function implementation

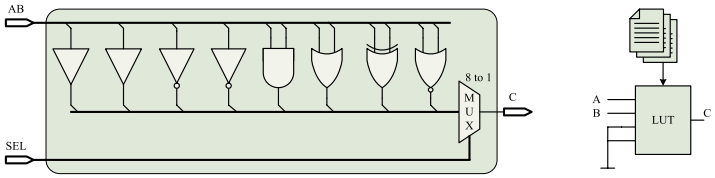
The intrinsic hardware evolution relies on a reconfigurable architecture, such as FPGA or a VRC built on top of FPGA. The reconfigurable architecture can be modeled as a programmable function element (PFE) array and interconnections. The hardware implementation of PFE requires a lot of logic resources. In this paper, we employ a LUT based logic function implementation method, which brings very small area occupation. The logic functions that are widely used in hardware evolution are shown in Table I.

Take a 2-input and 1-output PFE for example, 4 LUTs are required to implement 8 logic functions for the VRC based method. In the case of LUT

DOI: 10.1587/elex.11.20131003
Received December 18, 2013
Accepted January 23, 2014
Publicized January 31, 2014
Copyedited February 25, 2014

Table I. Logic functions of PFEs

Index	Function	LUT contents	Index	Function	LUT contents
0	0	0000	8	$\overline{A}\cdot B$	0010
1	1	1111	9	$\overline{A}\cdot\overline{B}$	0001
2	A	1100	10	$A\oplus B$	0110
3	B	1010	11	$\overline{A\oplus B}$	1001
4	$\overline{A}$	0011	12	$A+B$	1110
5	$\overline{B}$	0101	13	$A+\overline{B}$	1101
6	$A\cdot B$	1000	14	$\overline{A}+B$	1011
7	$A\cdot\overline{B}$	0100	15	$\overline{A}+\overline{B}$	0111



(a) (b)

Fig. 1. Two logic function implementation methods. (a) VRC based method (b) LUT based method

based method, only one LUT is needed. The two logic function implementation methods are shown in Fig. 1.

By changing the contents of LUT, the LUT can realize any 4-input logic functions. As in our work, the PFE has only 2-input, the other two inputs are connected to the ground.

As depicted in Fig. 2, with the growth of the number of logic functions, the LUT utilization of VRC increases drastically. On the contrary, the number of LUTs of the proposed method remains the same.

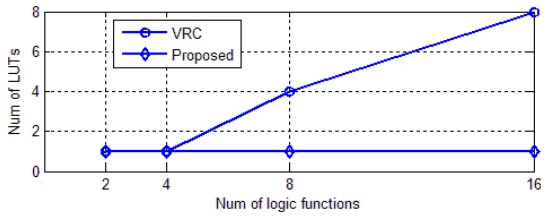


Fig. 2. LUT utilization comparison between VRC and the proposed method

2.2 Runtime modification of LUT and FF

In order to perform runtime modification, the location information of FFs and LUTs are needed. The location of the FFs and LUTs could be assigned manually or automatically. If the manual method is employed, the location information is included in the user constraint file. Otherwise, the PAR tool will assign the locations automatically. The location information is stored in the *.ncd* file, which has a closed binary format. The *-ncd2xdl* command can be used to translate the *.ncd* file into the *.xdl* file. The location information can be easily obtained from the *.xdl* file. The two methods are as shown in Fig. 3.

Once the location information of LUT/FF is obtained, the LUT/FF can be modified during work phase. The coordinates are represented as (X, Y)

DOI: 10.1587/elex.11.20131003
Received December 18, 2013
Accepted January 23, 2014
Publicized January 31, 2014
Copyedited February 25, 2014

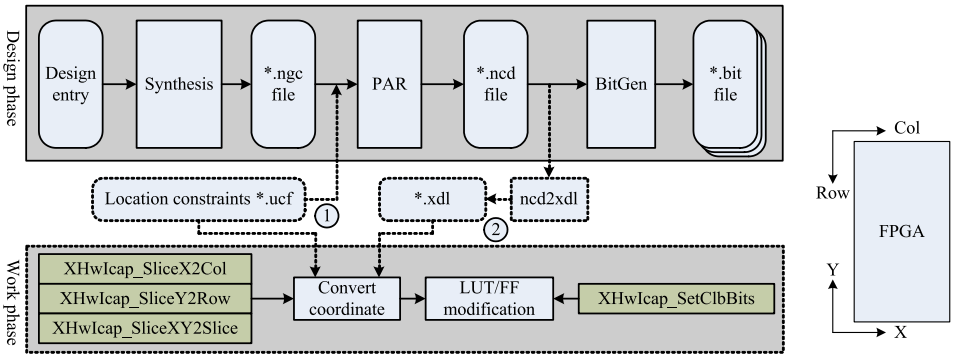


Fig. 3. LUT/FF manipulation flow

in the *.ucf or *.xdl file. While in the ICAP API, the coordinates are represented as (X, Y). As a result, the (X, Y) coordinates are converted to (Row, Col) coordinates used by the ICAP API. After that, LUT/FF could be modified using XHwIcap_SetClbBits. These four XHwIcap functions used during the LUT/FF manipulation flows are ICAP APIs of xps_hwicap IP provided by Xilinx. These APIs are fully supported by Xilinx embedded develop kit.

2.3 Proposed LUT based evolvable platform

An intrinsic evolvable platform is designed based on the proposed LUT based logic function implementation method. The proposed evolvable platform has been designed as a full on-chip hardware subsystem. The system includes a PowerPC core as the control unit, on-chip Block RAMs and external peripherals such as the DDR SDRAM and ICAP configuration interface. The PFE array is connected to the ICAP interface. The LUT contents can be dynamically configured at runtime through the ICAP interface. The detailed schematic view of the LUT manipulation based evolvable platform is shown in Fig. 4.

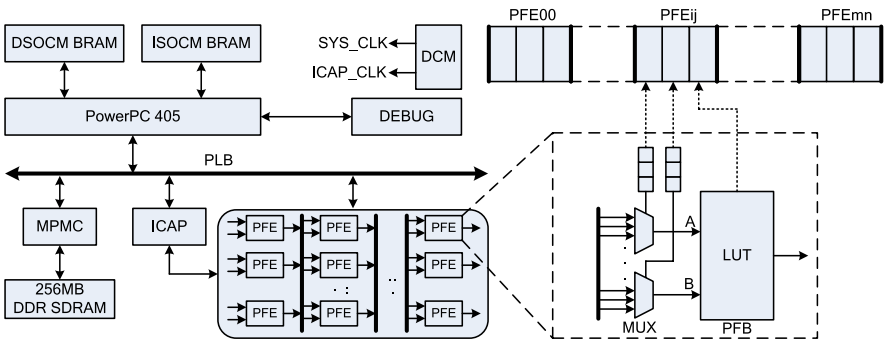


Fig. 4. LUT manipulation based evolvable platform

The chromosome is composed of two parts such as the interconnection strings and the function strings. The function of the LUT is defined by the function strings. The interconnection strings determine the control signals of the multiplexers. The multiplexers are also implemented using LUTs. The PFE is limited to connect to the PFE outputs from its immediate preceding column. In the case of VRC, the interconnection strings and the function strings are both stored in flip flops (FF). Since in our work, only the interconnection strings are stored in FFs, no FFs are needed for the function strings. The function strings which represent the LUT contents

DOI: 10.1587/elex.11.20131003
Received December 18, 2013
Accepted January 23, 2014
Publicized January 31, 2014
Copyedited February 25, 2014

are stored in LUTs directly. The LUT contents for each logic function are as shown in Table I.

The details of mapping from chromosome to bitstream are as shown in Fig. 5. The coordinates of LUTs and FFs can be attained by manual or automatic constraints during the design phase. The relationship between the genes and the LUTs and FFs are stored in LUT/FF coordinate library. The ICAP API maps the LUTs and FFs to the correct offset in the bitstream file. The mapping process are carried out by the ICAP API, no detailed information of bitstream is needed.

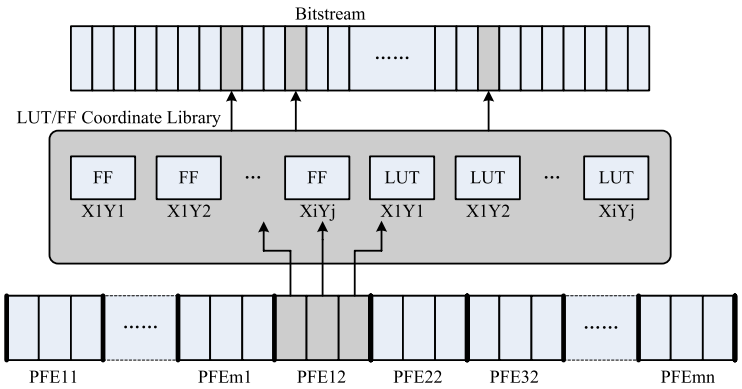


Fig. 5. Mapping from chromosome to bitstream

In this paper, our goal is to design a fully functional circuit using evolutionary algorithm (EA). The fitness function is:

$$fitness = \sum_{i=0}^{2^n-1} \sum_{j=0}^{m-1} (w_{ij} \odot v_{ij}) \tag{1}$$

In Eq. (1), m and n represent the number of bits of outputs and inputs of a circuit, respectively; w_{ij} is the jth bit of the output for the ith input pattern; v_{ij} is the desired output bit defined by truth table [6].

Fig. 6 shows the details of the PFE array configuration process. To evaluate the individuals during the evolution, the chromosomes are decoded to attain the corresponding contents of FFs and LUTs. The contents are used to configure the FFs and LUTs. The XHwIcap_SetClbBits API is employed to perform the configuration of FFs and LUTs. Take PFE_{ij} for example to illustrate the configuration process. The piece of chromosome represents PFE_{ij} is *chrom= {A_SEL[3:0], B_SEL[3:0], FUNC}*, where *A_SEL* is PFE's A input, *B_SEL* is PFE's B input, and *FUNC* is PFE's logic function. *A_SEL* and *B_SEL* are stored in FFs,

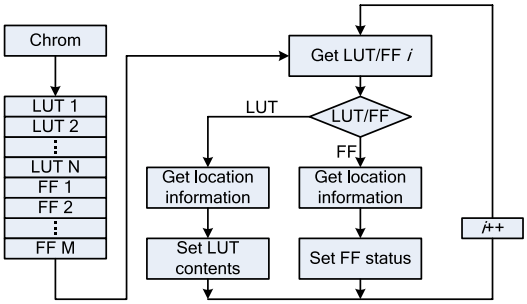


Fig. 6. PFE array configuration flow diagram

DOI: 10.1587/elex.11.20131003
Received December 18, 2013
Accepted January 23, 2014
Publicized January 31, 2014
Copyedited February 25, 2014

A_SEL[i] or B_SEL[i] is used to set the corresponding FF's status. *FUNC* represents LUT's logic function, which can be used to set LUT's content directly. The coordinates can be read from the aforementioned LUT/FF coordinate library.

By combining EA and reconfigurable architecture, an evolvable system can be constructed. The basic process steps of the hardware evolution are listed below:

Step 1: Randomly generates a set of initial population.

Step 2: Configure the PFE array. The configuration information is downloaded to the PFE array using ICAP API.

Step 3: Evaluates all individuals. If the circuit meets the requirements or the specified iteration is exhausted, then go to step 5, otherwise go to step 4.

Step 4: Create a new offspring using genetic operators (including selection, mutation), go to step 2.

Step 5: Evolution terminated.

3 Experimental results

The proposed evolvable platform is implemented on a Virtex-4 FX12 FPGA contained in ML403 Evaluation Platform. The PFE array contains 12 rows and 6 columns. The employed EA is $(1+\lambda)$ evolutionary strategy (ES), where λ=4. The genetic operator of the EA includes selection and mutation. No crossover is applied. The mutation rate is 0.03, and the maximum generation number is 200000. The ES is implemented in the C programming language which running on the PowerPC processor. The PowerPC runs at 400 MHz and the ICAP_CLK is 100 MHz. A commonly used 3-bit multiplier evolution is employed to verify the proposed platform. The optimal evolved 3-bit multiplier is shown in Fig. 7. The optimal 3-bit multiplier evolved by us has a comparative performance comparing with its competitors [6].

The performance comparisons are summarized in Table II. To make a fair comparison, the VRC is also synthesized to Virtex-4 FX12 FPGA. It can be seen from Table II that the proposed method achieves 50% logic

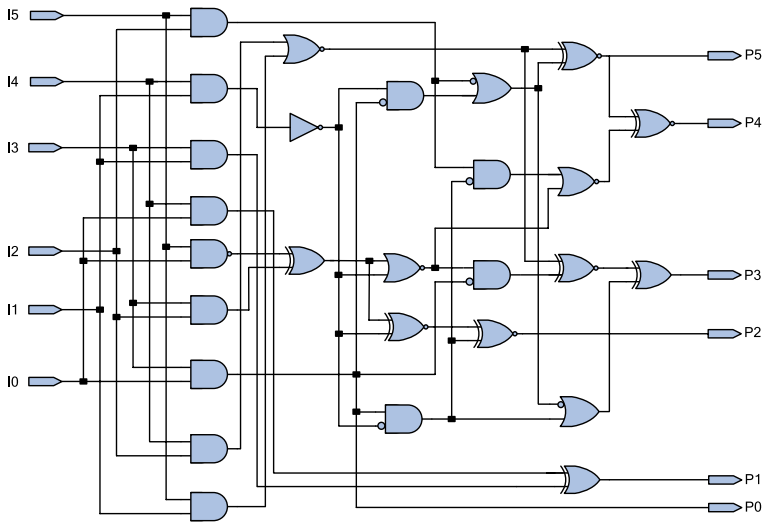


Fig. 7. An optimal evolved 3-bit multiplier (27 gates, 6 propagation gate delays)

DOI: 10.1587/elex.11.20131003
Received December 18, 2013
Accepted January 23, 2014
Publicized January 31, 2014
Copyedited February 25, 2014

Table II. Performance comparison with previous works

method	Logic utilization		Configuration time cost (s)	Dynamic routing support	Newer devices support
	#LUT	#FF			
VRC [6]	2100	988	3.88×10^{-6}	Yes	Yes
JBits [2]	-	-	6	Yes	No
BRE [4]	-	-	2.06×10^{-1}	No	Yes
BRE [3]	-	-	3.9×10^{-2}	No	Yes
Proposed	**1080**	**648**	$\mathbf{1.3\times10^{-3}}$	**Yes**	**Yes**

utilization saving over the VRC. In view of the varieties of EAs and benchmarks employed in the existing methods, configuration time cost is chosen as the criteria for evolution speed. Naturally, the VRC achieves the highest evolution speed. It can be seen from the results that more than three orders of evolution speed enhancement over JBits and one order of evolution speed enhancement over BRE based methods is achieved by the proposed platform. In theory, the BRE method supports newer devices. However, with the increase of the bitstream length, the BRE becomes more and more difficult. Although JBits supports dynamic routing, the supports for newer devices are not continued.

4 Conclusions

To the best of our knowledge, this paper for the first time reports a LUT manipulation based evolvable system without BRE. By taking advantage of both the direct bitstream manipulation systems and the VRC-based systems, our platform can achieve dynamic routing, which brings more flexibility comparing with the BRE based methods. The evolution of 3-bit multiplier is employed to demonstrate the efficiency and feasibility of our method. Experimental results show that more than three orders of evolution speed enhancement over JBits and one order of evolution speed enhancement over BRE based methods is achieved by the developed platform. Furthermore, with an acceptable speed sacrifice, the proposed method achieves 50% logic utilization saving over the VRC.

Acknowledgments

The authors acknowledge support from National Natural Science Foundation of China (NO.61302140). The first author acknowledges the support of Dr. Tao Xu.

DOI: 10.1587/elex.11.20131003
Received December 18, 2013
Accepted January 23, 2014
Publicized January 31, 2014
Copyedited February 25, 2014