Embedding the Digital Watermarks into FPGA-Projects Containing the Adaptive Logic Modules

Kostiantyn Zashcholkin[1], Oleksandr Drozd[1], Olena Ivanova[1],
Ruslan Shaporin[1], Olga Veselska[2], Hanna Stepova[1]
[1] Odessa National Polytechnic University, Odessa, Ukraine, const-z@te.net.ua, drozd@ukr.net,
en.ivanova.ua@gmail.com, rshaporin@gmail.com, hanna.suhak@gmail.com
[2] University of Bielsko-Biała, Bielsko-Biała, Poland, oveselska@ath.bielsko.pl

I. Introduction

Nowadays the FPGA chips are being widely used for building the computer systems of high performance. The reason for FPGA chips usage is that on the one hand they are programmable devices (like microprocessors), and on the other hand they provide rather high rate of performance, which cannot be guaranteed by microprocessors.

As for any programmable devices, the program code integrity provision for FPGA is problematical. This problem is traditionally solved in two ways: a) access restriction to the program code; b) usage of program code integrity monitoring. However, for FPGA-based systems in the course of their usage there is the need to modify their components program code in order to: a) eliminate the detected faults in functioning; b) upgrade and optimize a program code. The technical capability of such legitimate modification creates vulnerability, which can be used for malicious integrity violation of a program code. Because of this vulnerability the illegitimate modification of program code can be masked and executed with a single or several legitimate modifications.

To minimize the possibility of exploiting the mentioned vulnerability, the program code integrity monitoring is used. However, the usage of integrity monitoring does not fully solve this problem. This is due to the fact that the integrity monitoring process has some disadvantages itself, which potentially allows to bypass it.

To execute the integrity monitoring the monitoring hash sums of program code are used. These hash sums are either stored together with an information object of program code or join it by concatenating. As a result, it is not difficult to separate a hash sum from an information object, for which the integrity monitoring is carried out. The encrypting can be used to make the hash sum reading impossible. But all the same this does not hide from an outside surveillance the fact that the program code integrity is monitored. All these factors make possible to spoof a hash sum to hide the informational object integrity violation. For such kind of spoofing a range of attacks is used beginning from the brute-force search and ending with social engineering methods.

We suppose an approach, which carries out the FPGA program code integrity monitoring and where the monitoring data (including the informational object hash sum of program code) are embedded in the program code in the form of digital watermark, to be perspective. Embedding is executed by the equivalent transformation, which does not change both the program code size and FPGA-based system functioning. Such approach hides both a hash sum and the fact that the integrity monitoring is executed with respect to the program code.

II. Literature review and goal of the paper

As a target place of embedding a digital watermark in FPGA a program code of LUT (Look Up Table) units is used. The usage exactly of LUT units for the solution of embedding problem is conditioned by the two reasons:

- LUT units are the most mass elementary parts in FPGA structure. The presence of the large number of LUT units allows to vary the embedded digital watermark size in wide range;
- for program codes of LUT units there is an equivalent transformation, which permits to implement the embedding of digital watermark bits into these codes. The usage of this equivalent transformation modifies neither the LUT-circuit structure of FPGA-based device nor its power and performance characteristics.

Both method of equivalent transformation of LUT unit program codes and the ones of digital watermarks based on it are focused on the classical FPGA architecture. Within the framework of this architecture the FPGA basic logic element contains a programmable calculating LUT unit and programmable flip-flop. A LUT unit possesses n inputs (for the classical FPGA architecture n is normally equal to 4) and implements a single logical function from n arguments. The LUT unit programming is executed with the help of 2^n-bit program code. The result of the logical function calculation can be passed to the basic logic element output of FPGA through the flip-flop (which is a part of the basic logic element) or bypassing it (flip-flop).

For embedding a single bit of the digital watermark, a pair of series-connected LUT units is used. In coinciding the value of target bit of LUT_1 unit program code $code_1$ with the embedding bit of digital watermark the transformation is not realized. In the case of value inequality of these bits a bitwise inversion of LUT_1 unit program code $code_1$ is carried out as well as the bits rearrangement of LUT_2 unit program code $code_2$ compensating this inversion:

$$code^*_1 = I(code_1); \quad code^*_2 = P(v, code_2),$$

where $code_1$ and $code_2$ – program codes before transformation; $code^*_1$ and $code^*_2$ – program codes after the transformation; $I(code_1)$ – function of inverting the code bits; $P(v, code_2)$ – function of compensating rearrangement of the code bits; v – parameter defining the binary weight of LUT_2 unit input, which is connected to the LUT_1 unit output.

The manufacturers are trying to improve FPGA architecture to achieve the main goal – to increase the FPGA-based system performance. One of the aspects of such kind of architecture improvement is selecting the functions most frequently used in projects, which (functions) are not preferably implemented in programmable way but in the form of hard logic. As a result of this the classical FPGA architecture development is leading to the complication of basic logic element structure and including dedicated hard (non-programmable) units in its structure.

For example, FPGA of series Intel (Altera) Stratix, Arria and the latest generations of series FPGA Cyclone (Cyclone V, Cyclone 10GX), in which the basic logic element (ALM – Adaptive Logic Module) composition contains two dedicated adders with hard logic. LUT units including in ALM composition can pass their output values to the inputs of dedicated adders or the other components inside ALM, bypassing the dedicated adders.

The existing methods of embedding the digital watermarks in the FPGA program code are oriented to the classical FPGA architecture. These methods can be applied for FPGA with basic logic ALM element. However, in this case for embedding the digital watermark bits the LUT units, whose outputs are connected to the inputs of dedicated adders, cannot be used. This is conditioned by the fact that the methods applied for embedding requires either direct LUT unit connection or the ones (connections) through the flip-flop, which is included in the basic logic element composition of FPGA. The adder position between LUT units prevents the digital watermark bits from the possibility to be embedded into these units. This is the reason of using not in full the resources, which provide the LUT units for embedding the digital watermark bits. On this basis it should be stated that there exists the need to modify the methods of digital watermarks embedding for adaptation them (methods) to FPGA architecture with basic logic ALM elements.

The goal of the given paper is to make possible the digital watermark bits embedding into the program code of LUT unit, whose outputs are connected to the inputs of dedicated adders (which are included in the composition of basic logic ALM elements).

III. The proposed method adapted to FPGA with basic ALM elements

In the methods of embedding the program code of LUT unit (in which the digital watermark bit m_i, is directly imbedding), fits (if necessary) to the value m_i by inverting. If the LUT unit output is connected to one of the inputs of the dedicated adders then the possible code inversion makes changes in addition process logic. Table I shows the influence of the dedicated adder input inversions on its output values.

TABLE I. The influence of input inversions on the values of dedicated adder outputs

Inputs inversion	Outputs value changes
Inversion of any single input	~*S*, **Cout*
Inversion of any two inputs	*S*, **Cout*
Inversion of all three inputs	~*S*, ~*Cout*

The value at sum output is inverted (~*S*) in inverting one or three adder inputs and is not changed in inverting two inputs. The value at carry output is inverted (~*Cout*) in inverting three adder inputs. In the other two cases the carry has the values (**Cout*), which are not associated with any simple functional dependence to its original value.

Units LUT_1 and LUT_2 form the operands for the dedicated adder. A value from sum output *S* passes through the flip-flop or bypassing it to the output of ALM. The sum value is sent to the inputs of other ALM elements, which contain units LUT_3 … LUT_k. The dedicated adder receives the input value *Cin* (Carry in) and forms the output value *Cout* (Carry out). To achieve the goal of the given paper the possibility of embedding the digital watermark bits into program codes of units LUT_1 and LUT_2 should be provided.

A method of embedding the digital watermark bits into FPGA program code, which allows the usage of LUT units forming the operands for dedicated adders, is offered. The main statements of the offered method are presented below.

The first statement of the method is that for embedding only those LUT units are taken, which are connected to the adders with an unused carry output. They can be either the last adders in the carry chain or the ones, which calculate only the sum value and do not participate in carry formation.

The second statement of the method is that for embedding both of units LUT_1 and LUT_2, which are connected to the dedicated adder inputs, can be chosen or only one of them.

The third statement of the method is that if for embedding a digital watermark bit only one of the units LUT_1 or LUT_2 is used, and wherein the program code of this unit is inverted (wherein the sum value is also inverted) there are two possible ways of compensation of this inversion:

- the compensating rearrangement $P(v, code)$ of bits in program code of the LUT unit, which receive on their input the sum value ($LUT_3 \dots LUT_k$);
- the compensating inversion of the program code of the second LUT unit connected to the adder input. In this case the values inversion at two inputs of the adder creates a changeless value at sum output.

The fourth statement of the method is that if for embedding a digital watermark bit both of the units LUT_1 and LUT_2 are used and wherein the program code of these units is inverted, the compensation of these inversions should not be executed because the sum does not change its value.

The fifth statement of the method is that for the decrease of regularity of embedding the digital watermark bits by a pseudo-random way should choose, which of the LUT units in ALM composition are used – LUT_1 or LUT_2, or both. In addition, if only one of the LUT units is used a method of the inversion compensation is chosen in a pseudo-random way: by inverting the second LUT unit code or by compensating rearrangement $P(v, code)$ of units $LUT_3 \dots LUT_k$ program code bits. Wherein the parameters of pseudo-random number generator are the components of steganographic key, which is used for extracting the digital watermark.

The proposed method takes the following as input data: a) the sequence of bits of the digital watermark $M = <m_1, m_2, \dots, m_p>$; b) the steganographic key, which determines the rules and parameters of embedding and extracting the digital watermark; c) FPGA-project, in which embedding the digital watermark is executed. As a result of applying the method is a FPGA-project, in the program code of which the digital watermark is embedded.

The method offered in the paper is based on the method, which carries out embedding of bits $m_i \in M$ into FPGA with the classical architecture of basic logic elements. In the framework of this method the LUT unit enumeration occurs in the order determined by the steganographic key. In the course of enumerating the next bit m_i of the digital watermark is embedded in each of the LUT units. If the next LUT unit does not have connection to the dedicated adder, the base method is applied, otherwise the one offered in the given paper is used.

The method consists of the following cyclic sequence of stages.

Stage 1. The initialization of pseudo-random number generator is executed. The generator parameters are set by the steganographic key. The generator is aimed at generating the six possible values (e.g. 0…5).

Stage 2. The generation of pseudo-random number $RGen \in \{0 \dots 5\}$ is performed. This number determines what LUT units are to be used as the target ones for embedding the digital watermark bits. The correspondence of value *RGen* and target LUT units are shown in Table II. With values *RGen* 0 or 1 the unit LUT_1 is used for embedding the digital watermark. Wherein each of these values *RGen* determines a separate way of compensating the LUT_1 unit inversion: by inverting the code of the LUT_2 unit or by rearranging the $LUT_3 \dots LUT_k$ unit bits. Similarly values *RGen* 2 or 3 correspond to the two variants of inversion compensation when LUT_2 unit is used to embedding. With values *RGen* 4 or 5 both units (LUT_1 and LUT_2) are applied for embedding the digital watermark bits.

Stage 3. Depending on the condition if the value of current embedded bit m_i of the digital watermark and the value of the target bit of LUT unit program code (d_{LUT1} for unit LUT_1 and d_{LUT2} for unit LUT_2) coincide, the inversion (I) of LUT_1 and LUT_2 program code bits is executed according to table II. The rearrangement (P) of $LUT_3 \dots LUT_k$ units program code bits compensating this inversion is also carried out.

TABLE II. DEPENDENCE OF THE WAY OF EMBEDDING ON THE TARGET LUT UNIT

RGen	Target LUTs	Match	LUT_1	LUT_2	$LUT_3 \dots LUT_k$
0	LUT_1	$m_i = d_{LUT1}$	—	—	—
		$m_i \neq d_{LUT1}$	*I*	—	*P*
1	LUT_1	$m_i = d_{LUT1}$	—	—	—
		$m_i \neq d_{LUT1}$	*I*	*I*	—
2	LUT_2	$m_i = d_{LUT2}$	—	—	—
		$m_i \neq d_{LUT2}$	—	*I*	*P*
3	LUT_2	$m_i = d_{LUT2}$	—	—	—
		$m_i \neq d_{LUT2}$	*I*	*I*	—
4	LUT_1, LUT_2	$m_i = d_{LUT1}$ $m_{i+1} = d_{LUT2}$	—	—	—
		$m_i \neq d_{LUT1}$ $m_{i+1} = d_{LUT2}$	*I*	—	*P*
		$m_i = d_{LUT1}$ $m_{i+1} \neq d_{LUT2}$	—	*I*	*P*

		$m_i \neq d_{LUT1}$ $m_{i+1} \neq d_{LUT2}$	*I*	*I*	—
5	LUT_1, LUT_2	$m_i = d_{LUT1}$ $m_{i+1} = d_{LUT2}$	—	—	—
		$m_i \neq d_{LUT1}$ $m_{i+1} = d_{LUT2}$	*I*	*I*	—
		$m_i = d_{LUT1}$ $m_{i+1} \neq d_{LUT2}$	*I*	*I*	—
		$m_i \neq d_{LUT1}$ $m_{i+1} \neq d_{LUT2}$	*I*	*I*	—

After the completion of Stage 3 if all digital watermark bits have been embedded the method is considered to be completed. Otherwise, the transition to stage 2.

The method proposed in the given paper has been implemented in the form of software application. This application was used for the experimental research of the method. The goal of the experiment was as follows: to indicate on the basis of the specific FPGA-projects how much the size of the digital watermark, which can be embedded into FPGA program code, increases as compared to the methods existing previously.

To make the experiment the six FPGA-projects of different values and various design missions (calculating devices; control devices; the devices, which possess both calculating and control modules) were used. The synthesis of the projects was performed with the help of CAD Intel Quartus Prime. As target FPGA chips Intel Cyclone V were applied.

In the course of making the experiment the digital watermark bits were embedded into LUT unit program code of the experimental FPGA-projects. This embedding was executed in two ways:

- with the help of the methods oriented at the classical architecture, which used only LUT units not connected to the dedicated adders;
- with the help of the method, which is offered in the given paper.

In the process of embedding the digital watermark the maximal number of bits, which can be embedded using the first and the second ways under the condition of the same components of the steganographic key, was indicated. The results obtained during the experiment are shown in Table III.

TABLE III. EXPERIMENT RESULTS

Project No	**Mission of the project**	**The maximum possible number of bits of a digital watermark**		
		Method oriented on classic FPGA architecture	***Proposed method***	***Difference***
1	Calculations	8265	8330	65
2	Calculations	1436	1448	12
3	Control	1182	1182	0
4	Control	1281	1284	3
5	Mix	4589	4619	30
6	Mix	7403	7452	49

One can see from Table III that the proposed method permits to increase the number of the digital watermark bits for projects, in which some number of ALM elements function in the arithmetic mode. E.g. for the projects, which realize the arithmetic calculations or contain some calculating components, the method increases the possible number of bits of the digital watermark. For the projects, in the framework of which the arithmetical calculations are not made in the large scale (e.g. the projects of control devices No 3 and No 4) the method does not help increase the number of digital watermark bits.

IV. CONCLUSION

The method proposed in the paper is a part of the integrity monitoring technology of FPGA-based devices program code. The method provides the embedding of digital watermark into the LUT unit program code. To carry out the integrity monitoring the watermark possesses a hash sum in itself. The method is different from the existing ones because it permits to use for embedding the LUT units, connected to inputs of the dedicated adders, into the ALM elements composition.

The experimental comparison of the proposed method to the existing ones has shown a high efficiency of its application. This permits to increase the possible number of the digital watermark bits due to the usage of LUT units connected to the inputs of dedicated adders. Increasing digital watermark size gives the possibility to use a broader set of hash functions to monitoring the integrity of FPGA program code. This allows using the hash functions possessing a big cryptographically strong in the process of the integrity monitoring.