Integrated Intelligent Research (IIR)

International Journal of Web Technology
Volume: 05 Issue: 01, June 2016, Pages: 69-74
ISSN: 2278-2389

Hardware Trojan Detection and Prevention Using Pattern Matching Technique Along With Dummy Logic

G.Alamelu[1], P. Sridevi[2]
[1]Student, ECE Department, Sri Ramanujar Engineering College
[2] Asst. Professor, ECE Department, Sri Ramanujar Engineering College.

Abstract—Hardware Trojan horses (HTH) have recently emerged as a major security threat for field-programmable gate arrays (FPGAs). Previous studies to protect FPGAs against HTHs show that a considerable amount of logic resources are left unused to be misused by malicious attacks. A low-level protection and detection scheme for FPGAs is proposed by filling the unused resources with dummy logic. The unused resources at the device layout-level are identified and replaced by dummy logic cells for different resources. An intrusion detection system is used to detect the presence of Trojans using pattern matching techniques. The proposed HTH detection and protection scheme has been applied on Xilinx Virtex devices implementing a set of IWLS benchmarks. The results show that the chance of logic abuse can be significantly reduced by employing the proposed HTH detection and protection scheme. Experimental results also show that as compared to unprotected designs, the proposed HTH scheme imposes no performance and power penalties.

Keywords— field-programmable gate arrays (FPGAs), hardware trojan horse, intrusion detection system, pattern matching, dummy logic.

I. INTRODUCTION

Field-Programmable Gate Arrays (FPGAs) with lower *nonrecurring engineering* (NRE) cost and less *time-to-market* (TTM) compared to *application-specific integrated circuits* (ASICs) provide a promising single chip solution for implementing embedded systems [2]. The reconfiguration property of FPGAs provides designers the flexibility of implementing a wide range of applications. This property also enables the embedded system designers to apply design updates even after circuit implementation and shipment. In contrast to ASICs, however, the reconfigurability of FPGAs makes them more susceptible to design modification even after chip being manufactured [7]. Therefore, the secure implementation of an embedded system using FPGAs not only requires employing protection schemes during design and manufacturing process similar to ASICs, but also requires secured protection mechanisms against malicious attacks after device shipment.

Recently, *hardware trojan horses* (HTH) insertion and detection have received significant attention in the literature [6], [9], [17], mainly because they can maliciously disturb the normal functionality of a circuit or change its characteristics. This, in turn, can affect the reliability and security of a chip. Although a great deal of research has been conducted towards insertion of HTHs and/or detection/prevention against them in ASICs, there have been a limited amount of studies investigating HTH insertion and detection in FPGAs, where not only silicon HTH can be inserted in the device, application space HTH can also be inserted in the design [8], [23]. Such studies mostly focus on insertion and detection of a HTH or *Design For Hardware Trust* (DFHT).

Previous HTH studies on FPGAs can be classified into either high-level techniques using *hardware description languages* or low-level techniques. While high-level techniques [15] aim at higher levels of design process, low-level ones [7] perform insertions, detections, or DFHT at bitstream or device levels. In [18], the authors present a technique, called BISA, to fill the unused space in layout of a design with functional standard cells instead of filler cells. These cells form a circuit with specific I/O signature; hence, any modification in BISA will alter the signature. This can make fabrication-stage HTH insertion difficult and detectable, unless the inserted HTH only affects the original design cells, where BISA cannot detect the inserted HTH. The main aim of the method proposed in [13] is to utilize all synthesized resources on all clock cycles, so there will be no room within the hardware for HTH inclusion. This requires accurate and fully specified design in all levels of abstraction. While it is possible to check whether a design meets the mentioned criteria, it would be difficult to specify a design such that all of its resources are utilized all the time, particularly in such a way that their operation will be essential to appropriate operation of the whole circuit. An approach to implement various HTHs has been proposed in [14]. This approach cannot be detected by ring oscillator-based protection mechanisms. A low-level technique to manipulate the configuration bitstream has been proposed in [7]. In this technique, the unused parts of bitstreams are detected and a properly presynthesized, preplaced, and prerouted bitstream of a HTH is inserted to the original one. Such HTH insertion scheme cannot be applied to a design which is evenly distributed throughout the FPGA fabric.

In this letter, we propose a low-level HTH detection and prevention scheme by running the logic through an intrusion detection system and filling the unused resources of the FPGA with *low-level dummy logic* (LLDL). The proposed FPGA based intrusion detection system uses pattern matching detection to check specific matching criteria in every incoming packet. Ethernet output is sent to the FPGA where the IP packet is parsed and filtered. Incoming packets are stored in SDRAM and Bloom Filter is maintained in SRAM. Patterns are compared against the payload of a packet or values within the header of a packet. A match is alerted a human analyst to determine the next course of action. The unused resources

International Journal of Web Technology
Volume: 05 Issue: 01, June 2016, Pages: 69-74
ISSN: 2278-2389

within the FPGA device are identified and dummy logic cells are proposed for different resources of FPGAs. The proposed scheme significantly reduces the chance of application space HTH attacks by giving no free configurable resource for HTHs insertion in the design's bitstream. The malicious inclusion can either disturb the functionality of the original design, incur device fluctuations such as warming up or transistor aging as in [7]. This is in contrast to high-level protection techniques that may leave considerable amount of logic resources to be misused by attackers. Additionally, by employing the proposed low-level technique, the bitstream reverse engineering becomes much more difficult for the purpose of leaking design specifications, e.g., number, location, and configuration of utilized resources, which can facilitate subsequent power analysis attacks [8].

The proposed protection scheme could be automatically applied to the designs after placement and routing stages. In addition, the proposed LLDL scheme does not alter the placement and routing of the original design and only exploits the unused configurable resources in the FPGA. We have applied the proposed scheme on Xilinx Virtex devices implementing a set of IWLS 2005 benchmarks [1]. The experimental results reveal that our proposed scheme does not impose any power or performance overheads as compared to the original nonprotected designs. Our results also show that the proposed scheme increases the utilization of FPGAs up to 15X without having any power or performance overheads. The rest of this letter is organized as follows. Section II describes the proposed LLDL scheme. Section III presents experimental setup and results. Finally, Section IV concludes the letter.

II. THE PROPOSED SCHEME

In this section, we first discuss low-level configurable resources available in a typical FPGA. Then, the insertion of dummy logic and dummy routing resources are presented in the subsequent subsections.

A. Intrusion Detection System

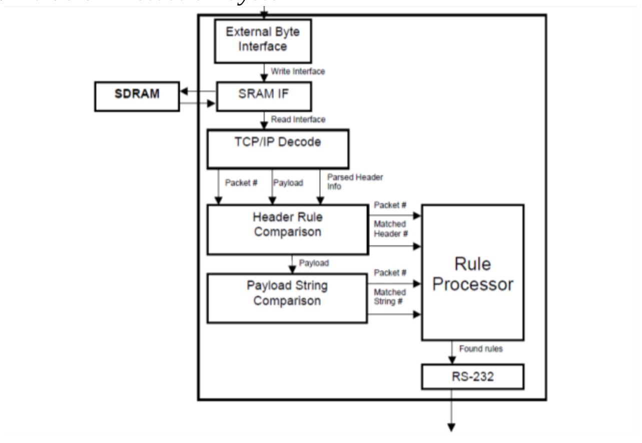

Fig.1. Block Diagram of IDS FPGA architecture

This function is performed in three stages. Processing begins by parsing the TCP/IP header. All packets have their TCP/IP header decoded. The header information is matched against rules in the Snort rule set for matches. If any packet matches any rule, it is sent along to string detection engines for payload inspection and the packet number is passed onto the final processing stage. If there is no match, then the packet is discarded and no further processing is performed on that packet. The payload inspection block takes each payload that meets the header criteria and compares it against payload strings found within the rule set. If a payload does not match any strings, it is discarded and no further processing is performed on the packet. However, if the payload does contain one or more strings from the rule set, the payload inspection block sends which strings have been matched, along with the packet number to the final processing stage. In the final processing stage, the matched headers are correlated with the matched payload strings to see if a complete rule has been matched. This block can either check for false positive results from the string comparison stage or it can pass results on for further processing. This block passes its results to the output interface of the FPGA.

B. Low-Level Configurable Resources in FPGA

LLDLs are added to the original circuit after placement and routing stage. At this stage, the *native circuit description* (NCD) file of the original circuit could be converted to the human readable description at the physical-level of device. Some vendors offer *layout-level hardware description language* (LHDL) to directly describe a device functionality using device resources. As an example, Xilinx provides a language, called *Xilinx description language* (XDL) [3], which can be employed to efficiently design a circuit with available device resources. Such vendors offer toolsets to convert the NCD file to its LHDL equivalent format. The produced LHDL format similar to the NCD file contains all of the low-level configuration state of logic and routing resources in the FPGA. Having the LHDL format of the original design, the proposed LLDL could be inserted by exploiting low-level FPGA resources.

Although the device resources in LHDLs vary among different series of FPGAs, they could be categorized under the following two groups: 1) *programmable logic points* (PLP); and 2) *programmable interconnect points* (PIPs). PLPs include the resources implementing a special logic which are typically grouped in a *cluster*. State-of-the-art FPGAs offer various types of PLPs for a single device type. The most common PLP uses *look-up tables* (LUTs), called *PLPL*. This type of PLP is composed of several LUTs, *Flip-Flops* (FFs), and Multiplexers and can implement both combinational and sequential circuits. Other types of PLPs may implement memory blocks, hard logic multipliers, or even *digital aignal processing* (DSP) blocks. For instance, Xilinx Virtex-II series have more than twenty PLP types such as *block random access memories* (BRAM), SLICE, BSCAN, IOB, ICAP, MULT18 X18 , PCILOGIC, STARTUP, and VCC [20]. PIPs, on the other hand are programmable interconnects which provide the connectivity between PLPs. In the subsequent subsections, we will discuss the LLDL insertion in PLP and PIP resources.

C. Dummy PLPL Insertion

After the placement and routing of the original design, FPGA clusters will be in either partially utilized or unutilized states. In order to protect partially utilized PLPLs against HTH insertion, the unused resources such as FFs, LUTs, and Multiplexers should be instantiated in the LHDL file in a way that it does not disturb the functionality of the original design. However, for the unutilized PLPL, the entire PLPL could be filled with a random dummy PLPL. We intentionally use random configuration for unused resources in order to avoid any distinguishable pattern in the dummy logic. However, if the original design contains limited or specific type of configurations, we can fill the dummy logic with similar configurations. A PLPL consists of a LUT-4 cell, a FF, multiple multiplexers, and hard logic cells, as shown in Fig. 1. Configuration data of a PLP determines its functionality. It is notable to mention that the intra-PLP connections are described in the LHDL file after the placement and routing stages. Listing 1 illustrates the description of a PLPL in the XDL format. As shown in Listing 1, the configuration of each resource in the PLP is presented in the XDL format. For instance, the function of a LUT is described as a boolean function of the four inputs. If a LUT has not been utilized, its corresponding state in the XDL format will be demonstrated as *OFF*.

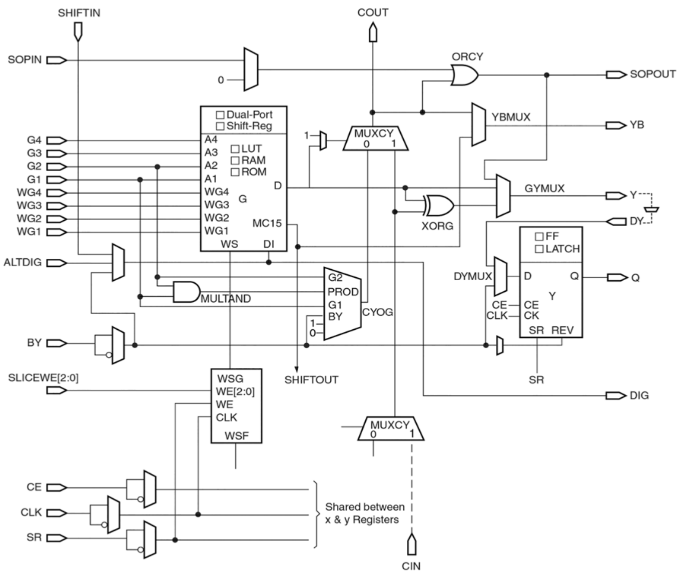

Fig. 2. Virtex-II Slice (Top Half) [21].

```
inst "N_106" "SLICE",placed R11C4 SLICE_X6Y11 ,
  cfg " BXINV::BX BXOUTUSED::#OFF BYINV::BY
BYINVOUTUSED::#OFF BYOUTUSED::#OFF CEINV::#OFF CLKINV::#OFF
COUTUSED::#OFF CY0F::#OFF CY0G::#OFF CYINIT::#OFF
CYSELF::#OFF CYSELG::#OFF DIF_MUX::#OFF DIGUSED::#OFF
DIG_MUX::#OFF DXMUX::#OFF DYMUX::#OFF
F::#LUT:D=((~A3*(~A1+(~A2+A4)))+(A3*(~A1*(A2*~A4))))
F5USED::0 FFX::#OFF FFX_INIT_ATTR::#OFF FFX_SR_ATTR::#OFF
FFY::#OFF FFY_INIT_ATTR::#OFF FFY_SR_ATTR::#OFF FXMUX::#OFF
FXUSED::0 F_ATTR::#OFF
G:#LUT:D=((~A3*(~A2*(A4*~A1)))+(A3*(~A2+(~A4+A1))))
GYMUX::#OFF G_ATTR::#OFF REVUSED::#OFF SHIFTOUTUSED::#OFF
SLICEWE0USED::#OFF SLICEWE1USED::#OFF SLICEWE2USED::#OFF
SOPEXTSEL::#OFF SOPOUTUSED::#OFF SRFFMUX::#OFF SRINV::#OFF
SYNC_ATTR::#OFF WF1USED::#OFF WF2USED::#OFF WF3USED::#OFF
WF4USED::#OFF WG1USED::#OFF WG2USED::#OFF WG3USED::#OFF
WG4USED::#OFF XBMUX::#OFF XUSED::#OFF YBMUX::#OFF
YBUSED::#OFF YUSED::#OFF ";
```

Listing 1. Cluster description in a sample XDL file.

The proposed LLDL protection scheme first automatically detects the unused clusters and then fills them with dummy PLPLs. Next, the unused resources in partially utilized clusters can be detected and filled up with dummy logic. Note that the inputs of any instantiated PLP should be driven by a net, and also its outputs should drive a net; otherwise, this PLP will be removed automatically during bitstream generation. Therefore, we insert a net per each output of a dummy PLP and also instantiate a single net to drive all inputs of the dummy PLP. Listing 2 shows a dummy net that drives all used inputs of a dummy PLP. By filling up partially used and unused clusters in the FPGA, there will be no freer cluster in the FPGA for possible low-level HTH insertions. Nevertheless, since a dummy logic does not implement any specific function, one can replace a HTH instead of the dummy logic without making any impact on the functionality of the original system. However, differentiating between the dummy logic and the original design especially after bitstream generation is a very challenging task. This is because the configuration bits are not distinguishable due to the confidentiality of the bitstream format.

```
net "NET0_SLICE_X0Y0" ,
  outpin "_SLICE_X0Y0_" COUT ,
  inpin "_SLICE_X0Y0_" G4 ,
  inpin "_SLICE_X0Y0_" BY ,
  inpin "_SLICE_X0Y0_" F4 ,
  inpin "_SLICE_X0Y0_" CLK ,
  ...
```

Listing 2. Net description in a sample XDL file.

```
inst "_RAMB16_X0Y0_" "RAMB16",placed place RAMB16_X0Y0 ,
cfg " CLKAINV::CLKA CLKBINV::#OFF ENAINV::ENA ENBINV::#OFF
PORTA_ATTR::1024X18 PORTB_ATTR::#OFF SAVEDATA::#OFF
SSRAINV::SSRA SSRBINV::#OFF WEAINV::WEA WEBINV::#OFF
WRITEMODEA::WRITE_FIRST WRITEMODEB::#OFF
INITP_00::4551b7...[RANDOM_DATA]
INITP_01::4eeee6...[RANDOM_DATA]
...
INIT_3f::e5fca42...[RANDOM_DATA]
INIT_A::00000 SRVAL_A::00000 ";
```

Listing 3. Filling an unused BRAM with random data in an XDL file

D. Other Dummy PLP Insertion

Except the PLPL which could be partially utilized, other PLPs have only two states: 1) used; and 2) unused. To protect a design against HTH insertion, unused PLPs are configured with a random configuration data. Clearly, the used PLPs will not be manipulated during the dummy logic insertion. As an example, if an unused BRAM is detected in the LHDL file, it will be filled up with a bunch of random data. This has been illustrated in Listing 3.

E. Dummy PIP Insertion

FPGAs comprise numerous wires spread out among logic blocks to provide appropriate routability. Each wire segment may drive or be driven by several other segments. For any wire segment, a PIP determines which of its endpoint segments be activated. Hence, the definition of a PIP is as "*pip tileName srcName-> destName*," in which, *tileName* refers to the encompassing tile (or cluster), *srcName* refers to the name of the source wire, and *destName* indicates the name of the destination wire. PIPs compose a major part of bitstream bits. For example, a Virtex2 XC2V80 device contains 720 752 PIPs which occupy about 48% of the device bitstream. However, PIPs that drive a unique destination within a cluster cannot be used simultaneously. Knowing this fact, we define a dummy net in the LHDL file and add a PIP "*pip TA->B* ", if it meets the following conditions:

1 there should not be any "*pip TX->B* ," neither in the original nor in the currently added PIPs;
2 there should not be any "*pip TA->Y* ," in the primary PIPs.

The first rule is necessary to avoid the cases that multiple wire segments drive a unique segment, and the second rule prevents adding extra load to the nets of the original design to avoid performance overheads.

F. Discussion: Possible Attacks and Configuration Encryption

Although brute-forcing the device logic to discriminate the dummies based on monitoring the design's functionality is theoretically possible, due to large number of resources inside a FPGA, it is extremely difficult to change the logic configuration one by one and give an essentially complete test pattern to verify the correct functionality of the design, particularly for complex and sequential circuits. Another way is to arbitrary place a HTH (without finding dummy logic), which requires no overlapping between resources of the original and HTH circuits. This is also infeasible, because for example if we consider half of the device resources to be empty and the HTH is composed of only 50 LUTs (which is less than 5% of total LUTs in smallest devices), the probability of having no overlap is approximately). Furthermore, since we have not left any room for adding new PIPs, the attacker cannot introduce new interconnect between the hypothetical HTH resources. A full reverse engineering, which requires the complete database between bitstream bits and corresponding resources can make it possible to generate the modified design XDL, but yet not the original design. Moreover, the available tools [12], [5], [4] have very limited resource database and only are available to extract a few set of PIPs as a plain text, so they are entirely unable to reconstruct the interconnect which comprises most of the configuration bits. Note it is also possible to distinguish a chunk of dummy LUTs by accurately monitoring the signal activities, if a considerable portion of the original device is unutilized. This issue can be resolved by choosing a near-to-minimum size device or evenly distributing the original design logic within the FPGA using simple synthesis constraints.

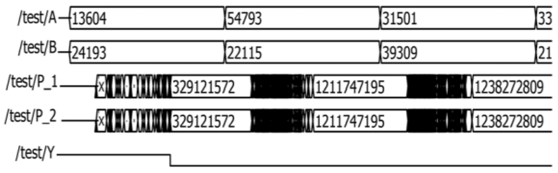

Fig. 3. Functional verification.

```
'timescale 10 ns/1 ps
module stimuli();
 reg [15:0] A, B;
 wire [31:0] P_1;
 wire [31:0] P_2;
 integer i;
 TopLevel6288 ins1(.A(A), .B(B), .P(P_1));
 TopLevel6288_full ins2(.A(A), .B(B), .P(P_2));
 assign Y = |(P_1 ^ P_2);
 initial begin
  for(i = 0; i < 1000; i=i+1) begin
   #4 A = $random; B = $random;
  end
  $stop;
 end
endmodule
```

Listing 4. Stimuli module for functional verification.

Another argument regarding the proposed scheme is that configuration encryption is sufficient to ensure the device security. We should note that not all of device families support bitstream encryption; even if they do, there are several successful works, usually based on power analysis attacks that break bitstream encryption [11]. Therefore, our scheme can be used on top of other protection mechanisms as configuration encryption. The proposed scheme protects the design against insertion of malicious objects in free resources of the device and is irrelevant to the modification made on the base design

III. EXPERIMENTAL SETUP AND RESULTS

Here, we will first verify the functionality of a design after applying the proposed HTH protection scheme. Then, the original and modified designs will be compared in terms of performance and power consumption. Finally, the resource utilization of the original and modified designs will be discussed. In these experiments, Xilinx ISE Design Suite 10.1 is used to implement a set of IWLS benchmark circuits on the Xilinx Virtex-II device. Then, performance reports are extracted by post placement and routing simulations in Xilinx Timing Analyzer [19]. In addition, power reports are extracted by Xilinx XPower tool [22]. Utilization reports are also adopted from Xilinx ISE utilization reports.

A. Functional Verification

In order to perform functional verification, post placement and routing simulation models are generated for both original and protected designs using Xilinx *netgen* command. This command produces a Verilog file along with the corresponding *standard delay format* (SDF) file from the post placement and routing NCD file. Then, as it is shown in Listing 4, the top modules of both the original and protected designs are instantiated and wrapped up in an another Verilog module named *stimuli*. The instances of the original and protected designs share the same inputs. In order to verify the functionality of the protected design against the original design, the equivalent outputs are first XORed. Then, the output of all XOR gates are ORed to examine the exact match between the outputs of both designs. Note that the generated HDL files contain Xilinx primitives as LUT modules and BRAMs. To this end, the SIMPRIM library [16] is added to the Mentor Modelsim [10] simulator. As it is shown in Fig. 2, simulation results reveal that the output of two designs are exactly identical. This proves that the proposed protection scheme does not disturb the functionality of the original design. To ensure whether all added instances exist in the final bitstream, we convert back the post placement and routing NCD to XDL file to remove possible dangling instances. The results show that dummy logic still exists in the XDL file, which proves that they have been added to the design appropriately.

Table I Power and Performance of Protected Versus Unprotected Designs

Circuit	Critical Path (ns)		Power (Watt)	
	Unprotected	Protected	Unprotected	Protected
ac97_ctrl	4.672	4.672	0.161	0.161
aes_core	7.140	7.140	1.075	1.075
des	6.811	6.811	0.837	0.837
mem_ctrl	12.763	12.763	0.356	0.356
spi	9.848	9.848	0.269	0.269
tv80	15.287	15.287	0.180	0.180
usb_phy	3.201	3.201	0.078	0.078

B. Performance

Critical path delay of both original and protected designs are obtained from the Xilinx Timing Analyzer reports. This tool gets an NCD file and analyzes the timing constraints of the design. As shown in Table I, the pad-to-pad critical path delay of both designs are the same. Hence, our scheme does not impose any performance overhead to the original design. This is one of the main advantage of the proposed LLDL scheme as compared to the HLDL schemes which can negatively affect the performance of the original design.

C. Power Consumption

To measure the power consumption of the mapped designs, Xilinx XPower Analyzer is used. This tool gets the NCD file beside an optional activity file (VCD or SAIF) to estimate the power consumption. The activity file is produced by ModelSim simulations described earlier in Section III-A. Table I illustrates the power consumption of the original (unprotected) and protected designs. As it is shown, although the protected design utilizes more resources than the original design, it consumes almost the same power as the original design. This is due to the fact that static power of a FPGA is constant, regardless of the implemented design. That is whether a resource is utilized or not, it consumes almost the same static power in the FPGA. In addition, the dummy resources in the protected design are not triggered by any input, i.e., although their input nets are configured, we choose to not drive them with any active signal (by the second criteria in Section II-B) to cause any switching. Therefore, a protected design does not consume further dynamic power as compared to the original design.

Table Ii Plp and Pip Utilization: Protected Versus Unprotected Designs

Circuit	PLP utilization		PIP utilization	
	Unprotected	Protected	Unprotected	Protected
ac97_ctrl	60.3%	100%	1.52%	7.65%
aes_core	7.3%	100%	0.25%	8.02%
des	21.3%	100%	0.46%	7.78%
mem_ctrl	19.1%	100%	0.51%	7.93%
spi	78.2%	100%	1.53%	7.48%
tv80	52.6%	100%	1.07%	7.55%
usb_phy	16.6%	100%	0.36%	8.03%

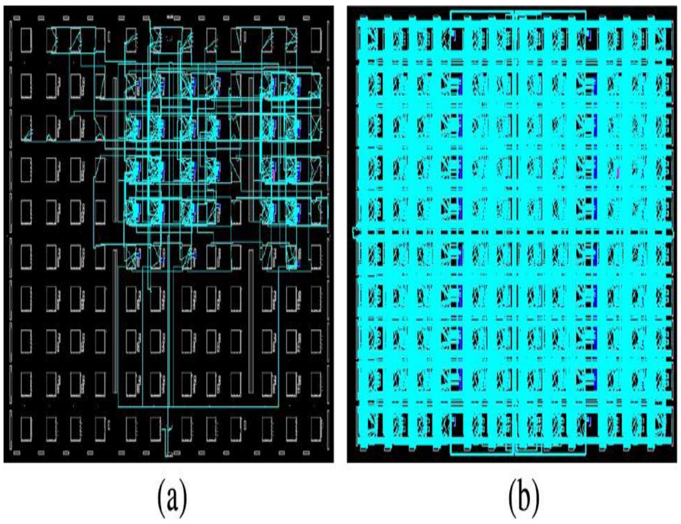

Fig. 3. Protected versus unprotected design. (a) Unprotected usb_phy. (b) Protected usb_phy.

D. Resource Utilization

Unused resources are potential candidates to be used by low-level HTHs. Therefore, resource utilization of a design in FPGAs can be used as a measure of susceptibility to the HTH insertion. High resource utilization reduces the chance of a HTH insertion. In the experiments, the resource utilization is measured before and after applying the protection scheme. The proposed protection scheme instantiates all unused PLPs in the target FPGA. However, as mentioned before in Section II, it cannot utilize all of the available PIPs. To evaluate the resource utilization, a set of IWLS benchmark circuits is implemented on a minimum sized FPGA. As shown in Table II, on average, the number of PIPs is increased by 15X in the protected design as compared to the original unprotected one. In addition, the PLP utilization is increased to 100% for all of protected designs. As a result, the resource utilization is significantly increased after applying the proposed protection scheme as compared to the original design. Note that the provided numbers for unprotected PLPs account only for the LUTs, i.e., several other primitives such as multipliers and BRAMs are entirely unused and significantly reduce the overall PLP utilization. Fig. 3 illustrates the resource utilization of a sample unprotected circuit (usb_phy) from IWLS circuits vs. the same circuit after being protected by applying the proposed LLDL scheme. Note that the high resource utilization is achieved in the proposed scheme while no power and performance penalties are imposed to the system

IV. CONCLUSION

In this letter, we proposed a low-level protection scheme against HTH insertion in the bitstream of FPGAs. This scheme is based on utilizing unused resources of an FPGA. Functional verification has been performed in order to assure the correctness of the functionality of the circuit after applying the protection scheme. Experimental results show that the proposed scheme increases the utilization of FPGAs up to 15X without having any impact on power or performance.

References

[1] C. Albrecht, “Iwls 2005benchmarks,” *Int. Workshop for Logic Synthesis (IWLS)*, 2005 [Online]. Available: http://www.iwls.org

[2] H. Asadi and M. B. Tahoori, “Analytical techniques for soft error rate modeling and mitigation of FPGA-based designs,” *IEEE Trans. Very Large Scale Integr.*, vol. 15, no. 12, pp. 1320–1331, Dec. 2007.

[3] C. Beckhoff, D. Koch, and J. Torresen, “The xilinx design language (xdl): Tutorial and use cases,” in *IEEE Int. Workshop Reconfigurable Commu.-Centric Systems-on-Chip (ReCoSoC)*, pp. 1–8.

[4] F. Benz, A. Seffrin, and S. A. Huss, “Bil: A tool-chain for bitstream reverse-engineering,” in *IEEE FPL.*, 2012, pp. 735–738.

[5] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, J. H. Anderson, S. Brown, and T. Czajkowski, “Legup: High-level synthesis for fpga-based processor/accelerator systems,” *ACM/SIGDA FPGA.*, pp. 33–36, 2011.

[6] R. S. Chakraborty, S. Narasimhan, and S. Bhunia, “Hardware trojan: Threats and emerging solutions,” in *Proc. IEEE Int. High Level Design Validation Test Workshop (HLDVT)*, 2009, pp. 166–171, IEEE.

[7] R. Chakraborty, I. Saha, A. Palchaudhuri, and G. Naik, “Hardware trojan insertion by direct modification of fpga configuration bitstream,” *IEEE Des. Test*, vol. 30, no. 2, pp. 45–54, Apr. 2013.

[8] S. Drimer, “Security for Volatile FPGAs,” University of Cambridge, Computer Laboratory, Rapport technique UCAM-CLTR-763, 2009.

[9] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor, “Trustworthy hardware: Identifying and classifying hardware trojans,” *Computer*, vol. 43, no. 10, pp. 39–46, 2010.

[10] Modelsim [Online]. Available: http://www.mentor.com/products/fv/modelsim/

[11] A. Moradi, A. Barenghi, T. Kasper, and C. Paar, “On the vulnerability of fpga bitstream encryption against power analysis attacks: Extracting keys from xilinx virtex-ii fpgas,” in *Proc. 18th ACM Conf. Comput.Commun. Security*, 2011, pp. 111–124.

[12] J.-B. Note and E. Rannaud, “From the bitstream to the netlist,” *FPGA*, vol. 8, pp. 264–264, 2008.

[13] M. Potkonjak, “Synthesis of trustable ics using untrusted cad tools,” in *Proc. ACM DAC*, 2010, pp. 633–634.

[14] J. Rilling, D. Graziano, J. Hitchcock, T. Meyer, X. Wang, P. Jones, and J. Zambreno, “Circumventing a ring oscillator approach to fpga-based hardware trojan detection,” in *IEEE ICCD.* , 2011, pp. 289–292.

[15] H. Salmani and M. Tehranipoor, “Analyzing circuit vulnerability to hardware trojan insertion at the behavioral level,” in *Proc. IEEE DFT*, 2013, pp. 190–195.

[16] Simulation Libraries. [Online]. Available: http://www.xilinx.com/itp/xilinx10/isehelp/ise_c_simulation_libraries.htm

[17] M. Tehranipoor and F. Koushanfar, “A survey of hardware trojan taxonomy and detection,” *IEEE Des. Test*, vol. 27, no. 1, pp. 10–25, Jan.- Feb. 2010.

[18] K. Xiao and M. Tehranipoor, “Bisa: Built-in self-authentication for preventing hardware trojan insertion,” *IEEE HOST*, pp. 45–50, 2013, IEEE.User Guide, Xilinx, “Timing Analyzer Guide - 2.1i”.

[19] User Guide, Xilinx, “Virtex-II platform FPGA User Guide,” Mar. 2005.

[20] Data Sheet, Xilinx, “Virtex-II Platform FPGAs: Complete Data Sheet,” Nov. 2007.

[21] “Xpower Analyzer,” [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_6/isehelp_start.htm

[22] J. X. Zheng, E. Chen, and M. Potkonjak, “A benign hardware trojan on fpga-based embedded systems,” in *Proc. IEEE FPL*, 2012, pp. 464–470.

[23] M. Attig and J. Lockwood. *A Framework for Rule Processing in Reconfigurable Network Systems.* IEEE Symposium on Field-Programmable Custom Computing Machines, April 18-20, 2005.

[24] M. Attig and J. Lockwood. *SIFT: Snort Intrusion Filter for TCP*. IEEE Symposium on High Speed Interconnects, August 17-19, 2005, Stanford, CA.

[25] Z. K. Baker and V. K. Prassanna. *A Computationally Efficient Engine for Flexible Intrusion Detection.* IEEE transactions on very large scale integration (vlsi) systems, vol. 13, no. 10, October 2005.

[26] R. Bejtlich. *Extrusion Detection: Security Monitoring for Internal Intrusions*. Addison Wesley. 2006.

[27] B. Bloom. *Space/Time Trade-offs in Hash Coding with Allowable Errors.* Communications of the ACM, July, 1970, pp. 422-426.

[28] S. Dharmapurikar, P. Krishnamurthy, T. Sproull, J. Lockwood. *Deep Packet Inspection using Parallel Bloom Filters Bloom Filters : A Tutorial, Analysis, and Survey by James Blustein and Amal El-Maazawi*, Faculty of Computer Science, Dalhousie University, B3H 1W5, Canada.

[29] J. W. Lockwood, N. Naufel, J. S. Turner, and D. E. Taylor. *Reprogrammable Network PacketProcessing on the Field Programmable Port Extender (FPX)*. In *ACM International Symposium on Field Programmable Gate Arrays (FPGA'2001)*, pages 87–93, Monterey, CA, USA, Feb. 2001.

[30] M. Mitzenmacher. *Compressed Bloom Filters.* IEEE/ACM transactions on networking, vol. 10,no. 5, October 2002.

[31] M.V. Ramakrishna. A *Simple Perfect Hashing Method for Static Sets*. Proceedings of the Fourth

[32] International Conference on Computing and Information, 1992, pp. 401-404.

[33] H. Song and J. Lockwood. *Multi-pattern Signature Matching for Hardware Network Intrusion Detection Systems*. IEEE Globecom 2005, St. Louis, MO, Nov. 28, 2005, pp. CN-02-3.