Security Enhancement of Remote FPGA Devices By a Low Cost Embedded Network Processor

Qutaiba I. Ali* Sahar Lazim
Computer Engineering Department/College of Engineering/Mosul University/Iraq
***E mail: Qutaibaali@uomosul.edu.iq**

Abstract

The incredible growth of FPGA capabilities in recent years and the new included features have made them more and more attractive for numerous embedded systems. There is however an important shortcoming concerning security of data and design. Data security implies the protection of the FPGA application in the sense that the data inside the circuit and the data transferred to/from the peripheral circuits during the communication are protected. This paper suggests a new method to support the security of any FPGA platform using network processor technology. Low cost IP2022 UBICOM network processor was used as a security shield in front of any FPGA device. It was supplied with the necessary security methods such as AES ciphering engine, SHA-1, HMAC and an embedded firewall to provide confidentiality, integrity, authenticity, and packets filtering features.

Keywords: FPGA Security, Network processor, Remote reconfiguration, AES algorithm, SHA-1 algorithm, HMAC algorithm, Firewall.

تحسين أمن أجهزة مصفوفة البوابات المبرمجة حقليا عن بعد بواسطة المعالج الشبكي المضمن المنخفض التكلفة

قتيبة ابراهيم علي سحر لازم قدوري
جامعة الموصل/ كلية الهندسة/ العراق

الخلاصة

النمو هائل في إمكانيات أجهزة مصفوفة البوابات المبرمجة حقليا في السنوات الأخيرة والميزات الجديدة المضمنة جعلته أكثر جاذبية للعديد من النظم المطمورة. مع ذلك هناك نقاط الضعف الهامة المتعلقة بأمن البيانات والتصميم. أمن البيانات ينطوي على حماية تطبيق أجهزة مصفوفة البوابات المبرمجة حقليا بمعنى أن البيانات داخل الدائرة والبيانات المنقولة إليه خلال عملية الاتصال عن بعد يجب أن تكون محمية. هذا البحث يقترح طريقة جديدة لدعم أمن أي جهاز مصفوفة البوابات المبرمجة حقليا باستخدام تقنية معالج الشبكة. تم استخدام معالج شبكة المنخفضة التكلفة UBICOM IP2022 كدرع الأمن أمام أي جهاز مصفوفة البوابات المبرمجة حقليا. حيث تم تجهيزه بالأساليب الأمنية اللازمة مثل خوارزمية التشفير الشهيرة AES ، خوارزمية SHA−1، وخوارزمية HMAC بالاضافة الى جدار الحماية المضمن لتوفير السرية والنزاهة والأصالة وميزات تصفية الحزم.

1. Introduction

Reconfigurable platforms cleverly partitioned between software and reprogrammable hardware will allow the designer through its flexibility to change designs at any stage even in the field. A reconfigurable platform such as Field Programmable Gate Array (FPGA) technology provides a cost-attractive alternative to ASIC implementation for small- to mid-volume applications. The number of designs which use reconfigurable hardware is exploding, with applications ranging from embedded systems to super-computers [1],[2]. FPGA becomes one of the most successful of today's technologies for developing the systems which require a real time operation. Nowadays, FPGAs are not only used as fast prototyping tools, but they also become active players as components in embedded systems. FPGA is a reconfigurable integrated circuit that consists of two dimensional arrays of logic blocks and flip-flops with an electrically programmable interconnection between logic blocks [3],[4]. The reconfiguration property enables fast prototyping and updates for hardware devices even after market launch[5],[6].

Remote configuration is an interesting FPGA feature that allows system upgrade, provide conditional functionality, and shorten time-to-market by selling products with limited functionality. But this feature must be strongly secured because it gives many possibilities to attackers. The first related threat is undesired reconfiguration, where, the design could be remotely changed by attacker without user permission. The second is "man in the middle" attacks, where, the user wants to upgrade his programmable device securely, but an attacker intercepts his request and replies with a fake configuration. Therefore, the connection must be secured with authentication and integrity checking engine in order to prevent interception of the configuration data and prevent unauthorized changes of the configuration data [7], [8].

The three essential principles on which security is based to guarantee the correct execution of a program and the correct management of the communications are [7]:

- Confidentiality: only the entities involved in the execution or the communication can have access to the data.
- Integrity: the message must not be damaged during the transfer or the program must not be altered before its execution.
- Authenticity: the entity must be sure that the message comes from the right entity or the system must trust the program source code.

In this paper, we suggest a security solution so that any FPGA platform will guarantee the convenience of all the above points in order to make it intellectually and costly effective. The outline of this paper is as follows: Section 2 displays the most recent and most efficient architecture for secure remote reconfiguration through the related works. Section 3 consists of the proposed system for secure remote reconfiguration protocol and illustrates the overview of UBICOM Network Processor that is used to protect FPGA platform on the insecure network. Section 4 contains the techniques for achieving configuration security. Section 5 presents the interfacing between UBICOM Network Processor and FPGA platform. Also, the testing for transmit real configuration bitstream file are displayed in section 6. As, section 7 shows the results to transmit some of bitstream files. Finally, the conclusions of this paper are declared in section 8.

2. Related Works

The research to improve the security level of FPGAs platform and particularly the improvement of remote configuration is necessary today. Some works give efficient solutions to secure the remote configuration of FPGA, nevertheless, there are some drawbacks and it is possible to be improved

in this paper. In 2008, Benoît Badrignans et al. [9] proposed a new architecture to encrypt and authenticate a bitstream at power-up, when the configuration is loaded from an external memory, and up on a remote system update performed by the SD (System Designer), in order to prevent any tampering with the FPGA configuration(including a replay). Also, Krzysztof Kępa et al. [10] proposed a Secure Reconfiguration Controller (SeReCon) which provides secure runtime management of designs downloaded to the Dynamic Partially Reconfigurable FPGA system and protects the design Intellectual Property (IP). SeReCon requires minor modification to the FPGA fabric. In 2009, Saar Drimer et al. [11] presented a security protocol for the remote update of volatile FPGA configurations stored in non-volatile memory. Their protocol provided for remote attestation of the running configuration and the status of the upload process. It authenticated the uploading party both before initiating the upload and before completing it, to both limit a denial-of-service attack and protect the integrity of the bitstream. In 2010, Ted Huffmire et al. [12] proposed security primitives using ideas centered around the notion of "moats and drawbridges". The primitives encompass four design properties: logical isolation, interconnect traceability, secure reconfigurable broadcast, and configuration scrubbing. Each of these is a fundamental operation with easily understood formal properties, yet they mapped cleanly and efficiently to a wide variety of reconfigurable devices. In the same year, Laurent Sauvage et al. [13] experimentally gave evidence that differential placement and routing of an FPGA implementation can be done with a granularity fine enough to improve the security gain. However, so far, this gain turned out to be lower for FPGAs than for ASICs. However, they expected that an in-depth analysis of routing resources power consumption could still help reduce the interconnected differential leakage. Also, Nele Mentens et al. [14] presented a solution for secure remote reconfiguration of FPGAs. Communicating the bitstream was done in a secure manner to prevent an attacker from reading or altering the bitstream. The result is an FPGA architecture that is divided into a static and a dynamic region. The static region holds the communication, security and reconfiguration facilities, while the dynamic region contains the targeted application. In 2011, Yannick Verbelen et al. [15] proposed a server architecture as a functioning solution against interception of reconfiguration bitstreams for embedded systems to allow secure bidirectional communication over TCP/IP. Elliptic curve key agreement, AES, and SHA-256 cryptographic routines have proven to be successful to implement a secure connection between server and FPGA client to exchange both data and reconfiguration bitstreams. In 2012, Lubos Gaspar et al. [16] proposed two novel protection schemes for IP bitstreams implemented on multi-FPGA systems. The first scheme is targeting low-cost FPGAs and provides a license scheme for IP owners to offer their products to system integrators in a secure way. The second unique scheme is provided to high-end partially reconfigurable FPGAs and enables IP owners to remotely install their IPs in an untrusted FPGA environment without having any pre-stored secret. In the same year, Binu K. Mathew et al. [17] proposed a new technique that can be used to provide security for circuits implemented of an FPGA. This technique not only provides security, but also makes the system run time reconfigurable, which is not possible with the encryption based systems. Bitstream encryption is mainly employed to avoid copying of bit streams when they are loaded into the FPGAs.

3. Problem Statements

The recent works have efficient solutions to progress the configuration FPGA security, but, they have some drawbacks from another side. Firstly, the hardware of the security techniques is

embedded inside the FPGA chip and consumes FPGA silicon area normally reserved for the developed applications. So, the total application dedicated-area is reduced by these solutions. Secondly, a new hardware chips are added to the FPGA platform to perform the security techniques for the configuration file. Thus, it is a lack of flexibility for the system, it will be difficult to upgrade it with a new encryption algorithm for example. Thirdly, partial and dynamic reconfiguration is difficult because it needs specific software and hardware facilities to manage it and can't be done on any FPGA families. Fourthly, the recent works didn't provide all the security solutions to cover the mentioned principles of security. Finally, recent solutions were built for specific FPGA families not for all.

In this paper, UBICOM IP2022 platform is adopted to implement security system to protect the configuration bit stream file of FPGA platform transferred through an insecure network. UBICOM IP2022 is a network processor produced by UBICOM Company; UBICOM provides the whole solution as a fully integrated platform - the Real Time Operating System (RTOS), the protocol stack, and the necessary hardware. UBICOM's IP2022 chip embeds some basic hardware, but it permits combining it with on-chip software to support the most prevalent protocols. The key to this approach is Software System on Chip (SOC) technology [18].

UBICOM's hardware includes the following components as shown in Figure 1 [18].

- 64KB (32K x 16) Flash program memory
- 16KB (8K x 16) SRAM data/program memory
- 4KB (4K x 8) SRAM data memory
- Can be provided with external flash memory (2M x 8)
- Two SerDes communication blocks supporting common PHYs (Ethernet, USB, UARTs, Bluetooth, Wireless IEEE 802.11, and so on.) and bridging applications
- 120MHz RISC processor
- Supports software implementation of traditional hardware functions
- In-system reprogrammable and run time self-programmable

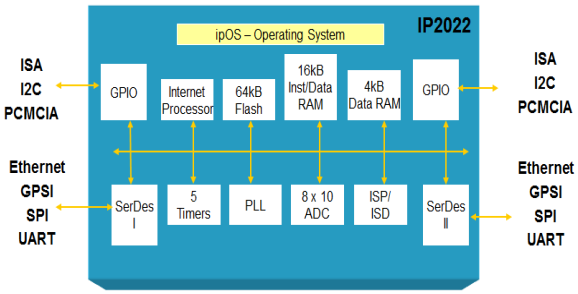

Figure 1 IP2022 Block Diagram

The basic components for UBICOM's complete development environment can separate to three essential components [19]:

1) UBICOM's Software Development Kit (SDK): UBICOM supports many software packages, like ipOS™, ipStack™, ipHAL™, ipModule™ and etc.
2) Red Hat GNUPro tools which consist of GCC ANSI C compiler, Assembler, Linker, and GNU debugger.
3) UBICOM's Configuration Tool Integrated tool to support rapid development efforts.

UBICOM's Unity Integrated Development Environment (IDE) contains Editor, project manager, graphic user interface to GNU debugger, device programmer.

This paper will exceed the previous drawbacks by adopting UBICOM Platform to handle the functions of the security system. Hence, it is an external component that can be considered portable and can be connected by a specific port with any FPGA platform, thus, there is no need to add a new hardware inside or outside FPGA by the manufacturer. As, it is a reprogrammable platform, it offers flexibility for the system by modifying the security techniques.

A security system is proposed to perform the secure remote configuration bit stream file of FPGA platform. The main

components of the proposed system are as follows (see Figure 2):

- Update Server:
- UBICOM Network Processor:
- FPGA Platform:

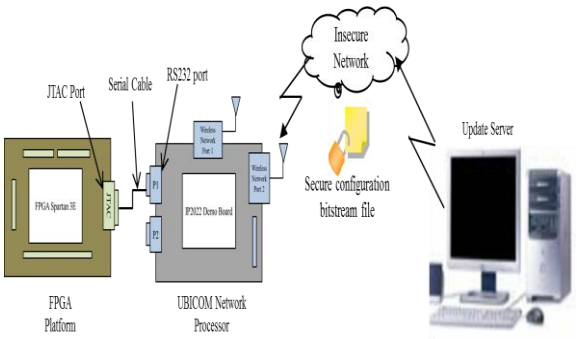

Figure 2 The Secure Remote Configuration Bitstream Protocol

Initially, a new configuration bit stream file is created on the update server, which is a specific PC provided with FPGA configuration tools and the security techniques tools. This file is processed by the security techniques (will be explained in section 4) such as AES encryption algorithm, creating the digest, and encrypting this digest. Afterward, the handshaking is established in order to ensure that the configuration file is suitable for the FPGA connected with UBICOM platform. If this operation completes, the processed configuration file is ready to be transmitted over the insecure network using TCP protocol for more reliability. UBICOM Network processor receives it through Ethernet or wireless network port, then, the file is processed by the reversed security techniques such as decryption operation and saved in its 2 MByte external flash memory. Thus, the file is prepared to be transmitted through serial port to the FPGA platform which received it via JTAG port.

4. Security Techniques for Achieving Remote Configuration

In the "man-in-the-middle" attacks, an attacker is assumed to be placed in the network between the update server and the UBICOM platform. He is able to retrieve and modify all the communications transmitted through this network. Considering such an attacker, it will focus on two attacks:

- ❖ Spoofing: first the adversary intercepts a communication or a bitstream then replaces it by his own.
- ❖ Replay: in a first step the adversary intercepts and records a communication or a bitstream and he can at any time "replay" the recorded data transfer through the network.

In order to guarantee that all security principles are available in the proposed system, some security techniques are chosen as follows:

- AES algorithm (will be explained in section 4.2) which is chosen to provide the privacy or confidentiality mechanisms to protect the bitstream against unauthorized reading.
- SHA-1 algorithm (will be explained in section 4.3) is chosen to guarantees the integrity of a message. It guarantees that the message has not been changed.
- HMAC algorithm (will be explained in section 4.3) is chosen to provide message authentication.

In this paper, it is assumed that the update server is trusted and that UBICOM platform is initialized in a trusted area. The configuration stream is encrypted with symmetric keys (the symmetric keys of AES and HMAC algorithms) shared between the UBCIOM platform and the Update Server. This mechanism permits the protection of the designer's IP against cloning. It also precludes reverse engineering of the configuration stream that might lead to IP disclosure.

However, to ensure the freshness of these bitstreams and to prevent replay attacks, a tag called the bitstream Version Tag (BVT) must be taken as an extra input. The bitstream Version Tag is a crucial parameter, this tag must be a NONCE (Number used ONCE), which is used to determine the number of FPGA configuration updates by the update server. An easy way to avoid the overflow, it is

chosen a large tag, for instance 64-bit. With such a tag, the counter that generates the BVT will never have the time to roll over during the FPGA lifetime. The update server and UBICOM platform also keep a copy of the FPGA chip's unique identifier, the platform ID in a trusted database.

Figure 3 illustrates the security operations which are done in update server, while, the security operations at UBICOM platform side are explained through Figure4. Figure 5 shows the security techniques which are executed at update sever and UBICOM Platform.

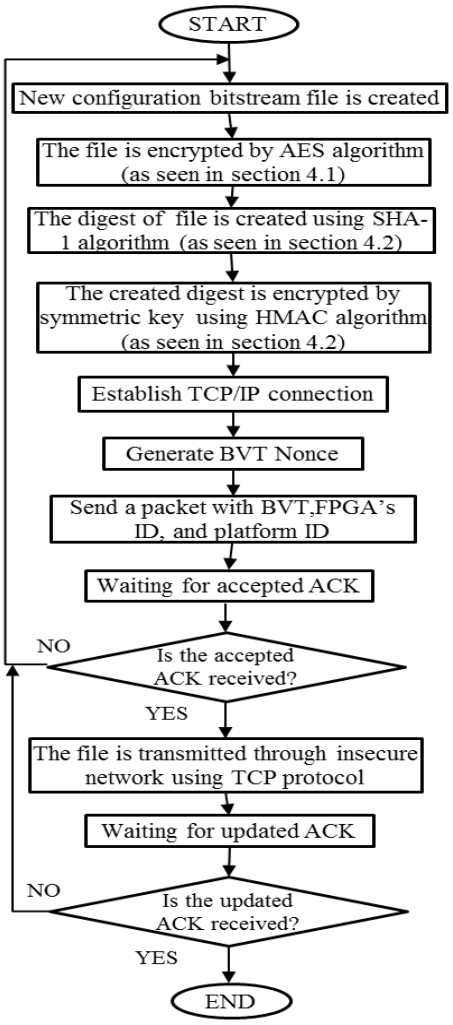

Figure 3 Security Operations at Update Server Side

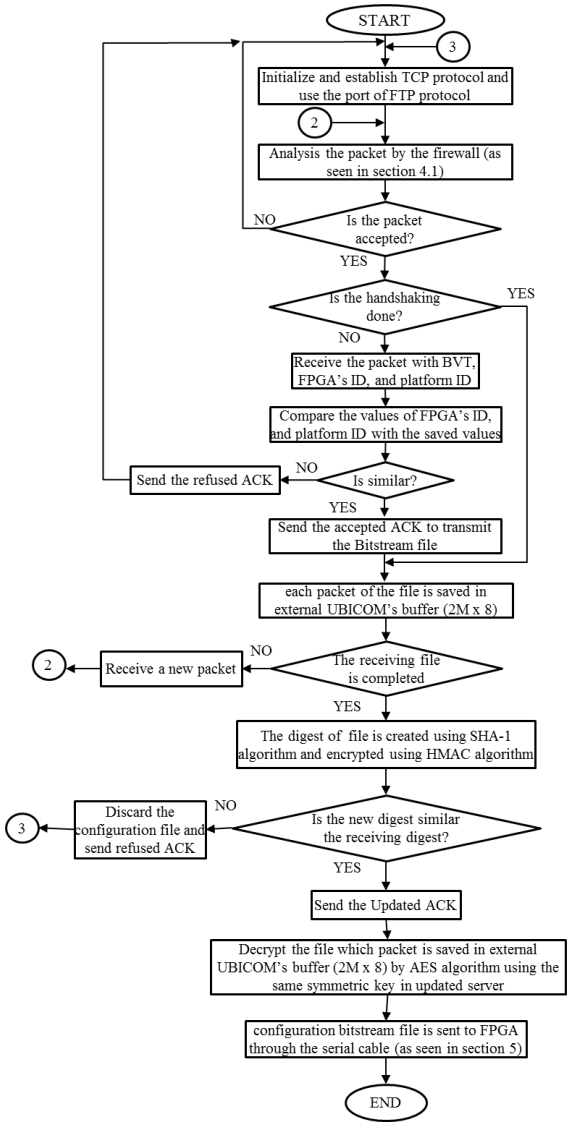

Figure 4 Security Operations at UBICOM Platform Side

4.1 Firewall

The suggested Ubicom security system was also armed with an embedded firewall. A data packet that matches a rule in the rule base is allowed to pass through the firewall into the system; the remaining packets are considered unsafe and hence, are discarded by the firewall. A packet filtering rule specifies the filtering policy for a data packet, i.e. whether to admit the packet into the system or to discard it at the firewall. The operation of a rule is based on the data packet's source and destination IP addresses the source and destination port numbers, and the transport protocol of the packet.

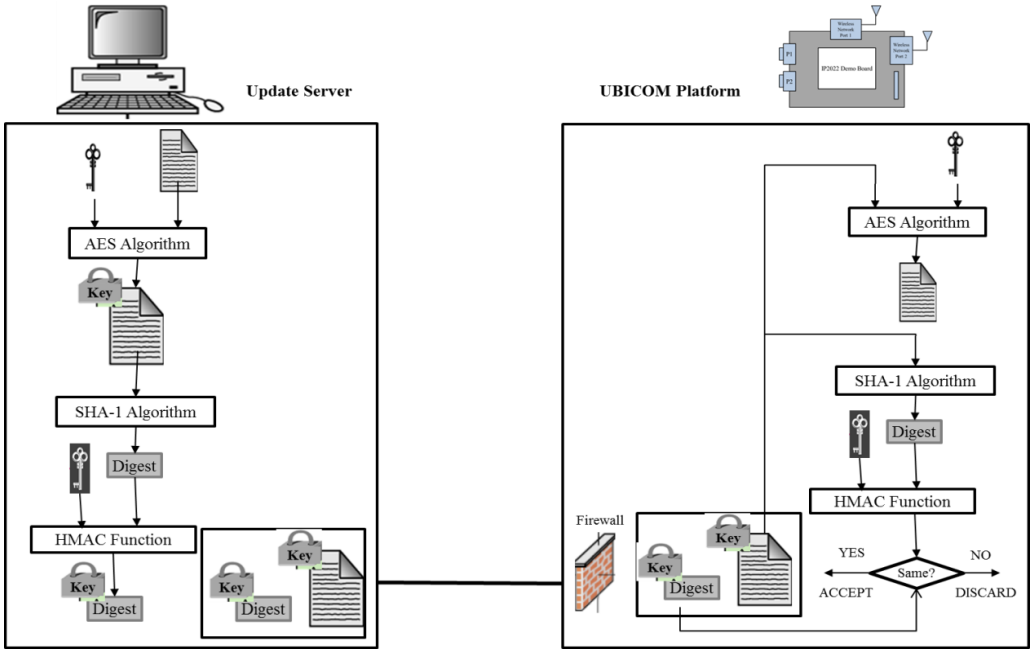

Figure 5 the Executed Security Techniques at Update Server and UBICOM Platform

Examples: IP Access List
access-list 101 permit tcp any any established
access-list 101 permit tcp any host 144.254.1.4 eq ftp
access-list 101 permit tcp any host 144.254.1.4 eq ftp-data
access-list 101 deny tcp any host 144.254.1.3 eq www
access-list 101 deny ip any any log

The below code represents the C code for the simple firewall, this code was uploaded to UBICOM platform and it will perform as shown in Figure6.

```
if (protocol==0x6) {
 if (dest_port==0x50) {
  if (dest_addr==0xa000001) {
   if (src_addr==0xa000000) {
     /*Action for "tcp 10.0.0.0 any -> 10.0.0.1..." */
     return;
  }}}}
```

4.2 AES Algorithm

In 2002 the Advanced Encryption Standard (AES) was approved by the National Institute of Standards and Technology for federal use in the United States. The AES algorithm is a symmetric block cipher that can encrypt (encipher) and decrypt (decipher) information where, Encryption converts data to an unintelligible form called cipher text; decrypting the cipher text converts the data back into its original form, called plaintext [19].The AES algorithm is capable of using cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt data in blocks of 128 bits. UBICOM platform supports key size of 128 bits only, so that this length of key will be used.

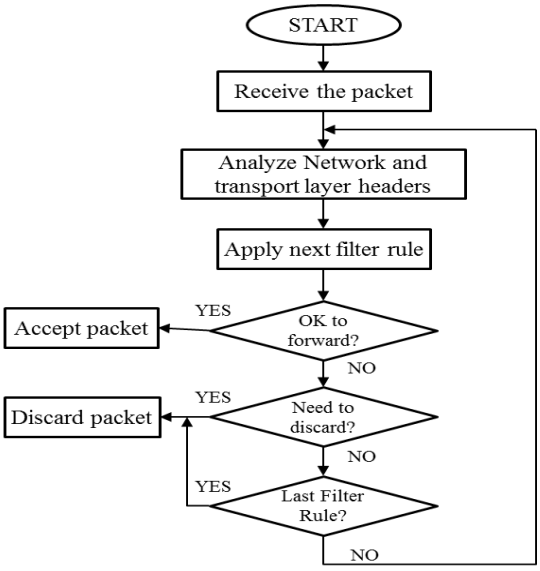

Figure 6 IP2022 Firewall operation flowchart

The functions that are used to execute the AES algorithm in order to decrypt configuration bitstream file are as follow:

- The function that expands the AES key into the round keys array which must be passed to the encryption and decryption functions, not the key itself, is as follows:

void aes_compute_round_keys(aes_block k, aes_round_keysrk);
aes_block k: *a pointer to the 16 byte AES key.*
aes_round_keysrk: *a pointer to a 176 byte array that will contain the round keys.*

- The following function decrypts the 16 bytes of ciphertext pointed to by a, leaving the resulting plaintext in the same array. The round key array must be derived from the 128 bit key using the first function.

void aes_decrypt(aes_block a, aes_round_keysrk);
aes_block a: *a pointer to an array of 16 bytes of ciphertext to decrypt.*
aes_round_keysrk*: a pointer to a 176 byte array that contains the round keys.*

The speed and space requirements of this function can be adjusted using the configuration tool of ipAES package that is specified for this algorithm on UBICOM platform.

4.3 SHA-1 and HMAC Algorithms

The Secure Hash Algorithm is a cryptographic hash functions. SHA-1(Secure Hash Algorithm 1) is a revised version of SHA designed by the National Institute of Standards and Technology (NIST). A very interesting point about this algorithm and others is that they all follow the same concept. Each creates a digest of length N from a multiple-block message, each block is 512 bits in length [19].

A hash function guarantees the integrity of a message. It guarantees that the message has not been changed. A hash function, however, does not authenticate the sender of the message. The digest created by a hash function is normally called a modification detection code (MDC). The code can detect any modification in the message. To provide message authentication, a modification detection code must be changed to a message authentication code (MAC). This idea is a hashed MAC, called HMAC that can use any standard keyless hash function such as SHA-l or MD5. HMAC creates a nested MAC by applying a keyless hash function to the concatenation of the message and a symmetric key [19].

The Ubicom's function that is executed to computes the HMAC-SHA1 message digest of the 64 bytes data is follows:

void hmac_sha1_create_digest(u32_t *digest, u8_t *input, addr_tinput_len, u8_t *key, addr_tkey_len);
u32_t *digest*: Pointer to the block of data to which the digest will be written.*
u8_t *input*: Pointer to the block of data over which the digest will be calculated.*
addr_tinput_len*: Amount of data to be used to generate the digest value.*
u8_t *key*: Pointer to the key to be used in generating the digest.*
addr_tkey_len*: Length of the key in bytes.*

5. Experimental Work

In order to validate the suggested method practically, a simple example of full adder logic circuits was built at update server and will be sent to the specific FPGA platform (Spartan 3E) through the insecure wireless network, see Figure7.

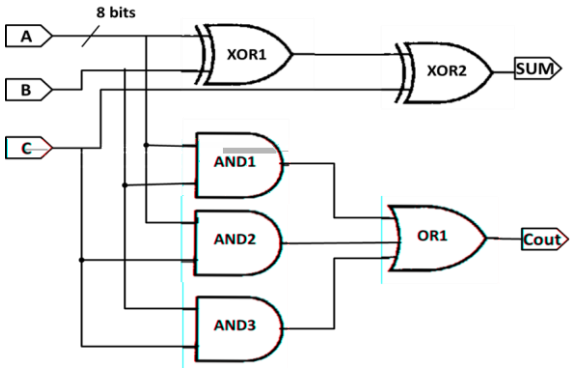

Figure7 Full Adder Logic Circuit

The VHDL code for the full adder schematic is written by the administrator at update server as follows:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity full_adder is
   Port ( A : in  STD_LOGIC_VECTOR (7 downto 0);
   B:in STD_LOGIC_VECTOR (7 downto 0);
  C :in STD_LOGIC_VECTOR (7 downto 0);
  SUM: out STD_LOGIC_VECTOR (7 downto 0);
 Cout: out  STD_LOGIC_VECTOR (7 downto 0));
end full_adder;
architecture Behavioral of full_adder is
begin
   SUM <= A XOR B XOR C;
  Cout<= (A AND B)OR(A AND C)OR(B AND C);
end Behavioral;
```

Subsequently, the VHDL file is converted to a bitstream file. As a demo, the first 64 bytes of the bitstream file (small part of bitstream file is used for this test) are encrypted by AES algorithm as follows:

```
Unencrypted_BitStream(64)=
0X4F79AE2D7F3316A62BD834CD9EAEF4
CCBFD092BF34005E875C6A19905EF4006
623B7D22810C682EEFFD9201414EB40C9
BB331729532EEEB72D1C9CD0B9127C21
```

```
AES_key(16)=
0XA2450237B488247DE8C1DA0589F611FE
```

```
Encrypted_BitStream(64)=
0XDC1699EA27DD896C8BA5A0888F85CF
5023778B9BF5C64E9D63FDA1CDA717253
795FF4A99B8A7E44530D3F0F6FCBA1233
CCD8BBB025EF5EE9882C4E8DBC8F8D64
```

Afterward, the encrypted bitstream Bytes are inputted to SHA-1 algorithm to compute the digest for the bitstream file and this digest is encrypted by HMAC function. The SHA-1 algorithm uses 5 state variables, each one of them is a 32 bit integer (an unsigned long on most systems). The variables are initialized as follows:

```
h0(8) = 0x67452301;
h1(8) = 0xEFCDAB89;
h2(8) = 0x98BADCFE;
h3(8) = 0x10325476;
h4(8) = 0xC3D2E1F0;
```

The key for the HMAC function can be in any length, the following key was chosen:

```
HMAC_Key(20)=
0X2F95F3EE73BABDBCCFD6FDEE2633B
E3A DEF24159
```

After the digest is encrypted by the key of HMAC function, it will be as follows:

```
Encrypted_Digest(20)=
0XD9F3738DCDBB469F0200A11D5ACE04
B1F6DFEA94
```

The encrypted Bitstream file and encrypted digest will be sent through the network to UBICOM platform to perform the previous operations in the reverse order. After performing all the necessary security actions mentioned earlier, the file which is saved in external flash memory of UBICOM platform will be sent to FPGA platform through RS232 - JTAG download cable. The configuration pins of RS-232 port are shown in Table 1.

Table 1 Pins of RS232 port

Pin	***Description***
DCD	Data Carrier Detect
RxD	Receive Data
TxD	Transmit Data
DTR	Data Terminal
GND	Signal Ground Data Set Ready
DSR	Data Set Ready
RTS	Request To Send
CTS	Clear To Send
RI	Ring Indicator

The JTAG standard itself defines instructions that can be used to perform functional and interconnect tests as well as built-in self-test procedures. Vendor-specific extensions to the standard allow execution of maintenance and diagnostic

applications as well as permit programming algorithms for reconfigurable parts. The heart of any JTAG test is the Test Access Port (TAP) controller. JTAG boundary scan testing is accomplished with only four external signals: Test Mode Select (TMS), Test Clock (TCK), Test Data Input (TDI) and Test Data Output (TDO). When parts are interconnected with a JTAG boundary scan chain, their various TMS, TCK, TDI and TDO signals are attached in various configurations [20]. The function of each JTAG pin is described in Table 2. While, Figure 8 declared the connection pins for download cable between RS-232 and JTAG port.

Table 2 The four pins of JTAG description

Pin	*Description*
TDI	Test Data Input is the serial data input to all JTAG instruction and data registers.
TDO	Test Data Out is the serial data output for all JTAG instruction and data registers.
TMS	Test Mode Select is the mode input signal to the TAP Controller.
TCK	JTAG Test Clock sequences the TAP controller as well as all JTAG registers

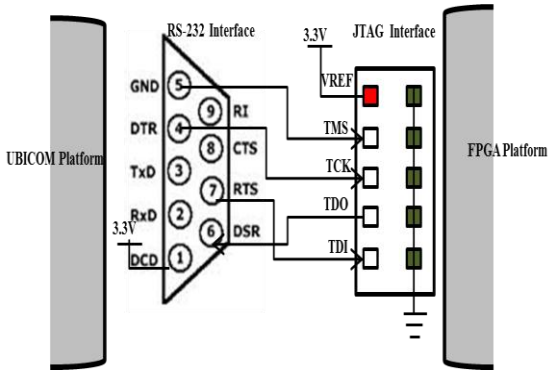

Figure 8 The Connection Pins for The Download Cable

In JTAG configuration, the status pin (DONE) functions the same as in the other configuration modes. The DONE pin can be monitored to determine if a bitstream has been completely loaded into the device. If DONE is Low, the entire bitstream has not been sent or the start-up sequence is not finished. If DONE is High, the entire bitstream has been received correctly. DONE pin is connected to an input pin as flag bit on UBICOM platform to ensure that the FPGA platform is updated or not. This flag bit is sent to update server to inform it that the bitstream file is damaged.

The TAP controller responds to control sequences supplied through the test access port (TAP) and generates the clocks and control signals required by the other circuit blocks. Table 3 describes the TAP controller commands required to configure FPGA platform [21].

6. Results

In this section, the times and delays to transfer a configuration file with any size from the update server to UBICOM platform. Figure 9 displays the experimental environment components. Where, update server is shown with Corei3CPU type and 3GHZ speed of processor which includes the configuration tool (Xilinx ISE 9.2i) and the software security techniques programs (AES algorithm, SHA-1 and HMAC algorithm, and firewall programs). As, UBICOM platform is connected to access point through Ethernet cable, in the other side, it is connected to JTAG loader through Serial to USB cable. JTAG loader is linked to FPGA kit by the four pins of JTAG interface.

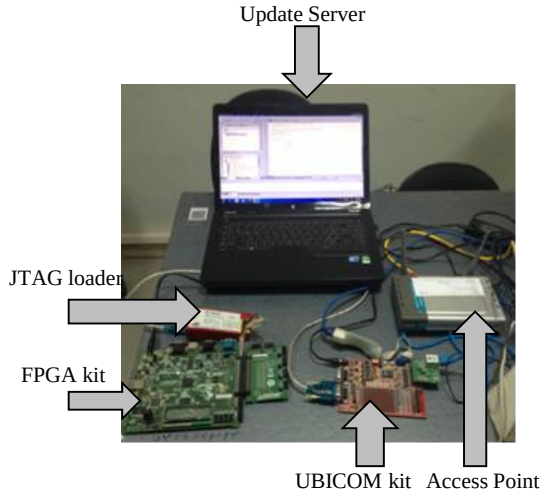

Figure 9 The experimental environment

Table 3 Single device configuration sequence

TAP Controller Step Description		Set and Hold Number of		Set and Hold Number of
		TDI	**TMS**	**TCK**
1	On power-up, place a "1" on the TMS and clock the TCK five times. (This ensures starting in the TLR (Test-Logic-Reset) state.)	X	1	5
2	Move into the RTI state.	X	0	1
3	Move into the SELECT-IR state.	X	1	2
4	Enter the SHIFT-IR state.	X	0	2
5	Start loading the CFG_IN instruction .(1)	0101	0	4
6	Load the last bit of CFG_IN instruction when exiting SHIFT-IR (defined in the IEEE standard).	0	1	1
7	Enter the SELECT-DR state.	X	1	2
8	Enter the SHIFT-DR state.	X	0	2
9	Shift in the Virtex bitstream. (bitN (MSB) is the first bit in the bitstream (1))	bit1… bitN	0	(Number of bits in bitstream) -1
10	Shift in the last bit of the bitstream. (bit0 (LSB) is shifted on the transition to EXIT1-DR)	bit0	1	1
11	Enter UPDATE-DR state.	X	1	1
12	Enter the SELECT-IR state.	X	1	2
13	Move to the SHIFT-IR state.	X	0	2
14	Start loading the JSTART instruction.(1) (The JSTART instruction initializes the startup sequence.)	1100	0	4
15	Load the last bit of the JSTART instruction.	0	1	1
16	Move to the SELECT-DR state.	X	1	2
17	Move to SHIFT-DR and clock the STARTUP sequence. (by applying a minimum of 12 clock cycles to the TCK).	X	0	≥14
18	Move to the UPDATE-DR state	X	1	2
19	Return to the RTI state. (The device is now functional).	X	0	1

Figure 10 shows how the wireless connections were achieved between the update server and UBICOM platform to transfer the configuration file.

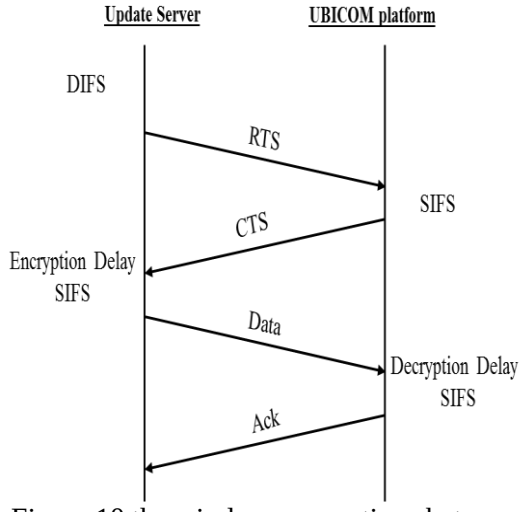

Figure 10 the wireless connections between Update Server and UBICOM Platform

From Figure 10, Encryption Delay includes the summation of the time to execute AES, SHA-1 and HMAC algorithms for any transferred configuration file on the Update server, while, Decryption Delay considers the summation of the time to execute AES, SHA-1 and HMAC algorithms on any received configuration file on the UBICOM platform.

Depending on the settings given in Table 4, the latency for different size of configuration file was measured (Firewall time is neglected because it is very small by comparing with other times). Figure 11 shows the additional latency variation resulting from different size of file with the effect of the different security methods.

Table 4 Experimental Settings

Variables	Value
Data Rate for 802.11g	54 Mbps
Packet processing rate for Update server	10000packet/ sec.
Packet processing rate for UBICOM platform	2000 packet/ sec.
Max. Packet length for wireless network	2312 bytes
Packet lengths for RTS and ACK	20 bytes
Packet lengths for CTS	14 bytes
Executing time for AES algorithm on Update Server	0.73 mSec
Executing time for AES algorithm on UBICOM Platform	18.4 mSec
Executing time for SHA-1 and HMAC algorithms on Update Server	2.66 mSec
Executing time for SHA-1 and HMAC algorithms on UBICOM Platform	66.7 mSec
DIFS	100 µ sec
SIFS	10 µ sec

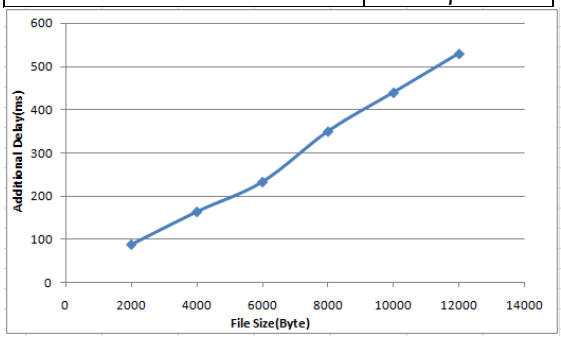

Figure11 Additional Delay Variation vs. File Size

Figure 11 shows the variation in the additional delay value with respect to different file sizes (payload size). It is noted that the additional delay values could be effective when transferring relatively large files due to the further processing needed to perform the different security tasks in various locations in the system. However, the final judgment on the system delay could be made according to the real time demands of the different FPGA applications.

7. Conclusions

FPGA is nowadays commonly used in applications requiring regular upgrade of the hardware. Remote update/upgrade is attractive in such systems to offer new multimedia features or to repair eventual security vulnerabilities. However, remote update requires transmitting the hardware Intellectual Property (IP) over insecure communication channels and thus introduces new security issues In this paper, we suggest the adoption of low cost embedded network processor to support the security of remote FPGA platforms. We believe that the current method has several advantages which make it as an attractive, flexible and strong security solution. This method can work on any FPGA platform, offloading security tasks from FPGA, supported with full security solutions and can be upgraded at any time easily.

8. References

[1] K. Kepa, et al., "Design Assurance Strategy and Toolset for Partially Reconfigurable FPGA Systems", ACM Transactions on Reconfigurable Technology and Systems, Vol.2, No.3, 2009.

[2] D. Nath, et al., "Customization of Arm Processor in Mixed Signal FPGA", International Journal of Engineering Research & Technology (IJERT), Vol. 1, Issue 6, Aug. 2012.

[3] Z. Obaid, et al., "FPGA-based Implementation of Digital Logic Design using Altera DE2 Board", International Journal of Computer Science and Network Security (IJCSNS), Vol. 9, No. 8, July 2009.

[4] Beyrouthy, and L. Fesquet, "An Asynchronous FPGA Block with Its Tech-Mapping Algorithm Dedicated to Security Applications", International Journal of Reconfigurable Computing, Vol. 2013, 2013.

[5] S. Drimer, "Security for volatile FPGAs", PhD thesis, University of Cambridge, 2009.

[6] D. Merli, et al., "Improving the Quality of Ring Oscillator PUFs on FPGAs", ACM Transactions on Reconfigurable Technology and Systems, Vol.3, No.2, 2010.

[7] E. Wanderley, et al., "Security and FPGA: Analysis and Trends", Montpellier Laboratory publications, 2011.

[8] N. Mentens, "Secure Remote Reconfiguration of FPGAs", Dynamically Reconfigurable Architectures Journal, Katholieke University, Leuven, 11-16 July 2010.

[9] B. Benoˆıt, et al., "Secure FPGA Configuration Architecture Preventing System Downgrade", IEEE Field Programmable Logic conference, Sep. 2008.

[10] K. Kępa, et al., "SeReCon: a Secure Dynamic Partial Reconfiguration Controller", IEEE Computer Society Annual Symposium on VLSI, pp. 292-297, 2008.

[11] S. Drimer, and M. Kuhn, "A Protocol for Secure Remote Updates of FPGA Configurations", Computer Laboratory publication, University of Cambridge, 2009.

[12] T. Huffmire, et al., "Security Primitives for Reconfigurable Hardware-based Systems", ACM Transactions on Reconfigurable Technology and Systems, Vol.3, No.3, 2010.

[13] L. Sauvage, et al., "Exploiting Dual-Output Programmable Blocks to Balance Secure Dual-Rail Logics", International Journal of Reconfigurable Computing, 2010.

[14] N. Mentens, et al., "Secure Remote Reconfiguration of FPGAs", Dagstuhl Seminar Proceedings 10281, Dynamically Reconfigurable Architectures, 2010.

[15] Y. Verbelen, et al., "Implementation of a Server Architecture for Secure Reconfiguration of Embedded Systems", Journal of Systems and Software (ARPN), Vol. 1, No. 9, Dec. 2011.

[16] L. Gaspar, et al., "Two IP Protection Schemes for Multi-FPGA Systems", ReConFig Conference, Mexico, 2012.

[17] B. Mathew, and K. Zachariah, "New techniques to Enhance FPGA based System Security", International Journal of Advanced Research in Computer Engineering & Technology, Vol. 1, Issue 5, July 2012.

[18] "IP2022 Wireless Network Processor Features and Performance Optimized for Network Connectivity IP2022 Data Sheet", UBICOM, Inc., 22 Jan. 2009, Web Site: http//www.ubicom.com.

[19] B. Forouzan, "Data Communication and Networking", 4th edition, McGRAW - HILL, ISBN 978-007-125442-7, 2007.

[20] Xilinx Corporation, "The Tagalyzer - A JTAG Boundary Scan Debug Tool", Application Note, XAPP 103(Version 1.1), 1 Mar. 2007.

[21] Xilinx Corporation, "Configuration and Readback of Virtex FPGAs Using JTAG Boundary-Scan", Application Note: Virtex Series, XAPP139 (v1.7), 14 Feb. 2007.