SYSTEM FOR ENCRYPTED COMMUNICATIONS THROUGH PUBLIC SWITCHED TELEPHONE NETWORK

Valeriu IONESCU[1], Ion SIMA[2], Emil SOFRON[3]
[1,3]University of Pitesti, Faculty of Electronics, Communications and Computer Science, Romania
[2]LUMINA-The University of South-East Europe, Bucharest, Colentina str., No 64B, Romania
[1]valeriu.ionescu@upit.ro, [2]ion.sima@lumina.org, [3]emil.sofron@upit.ro

Keywords: AES, SHA, MELP, OpenSSL, voice and data communications

Abstract: *Classic Public Switched Telephone Networks (PSTN) lines offer very low data bandwidth, but the evolution of encryption and encoding techniques makes possible to easily implement secure transmissions over this medium. This paper presents the design and implementation problems for a secure voice communication system using PSTN lines. The developed system uses software components mixed with FPGA hardware in order to offer design flexibility and the processing speed.*

1. INTRODUCTION

Voice and data communications are key components in today's information exchange, and users want to access them securely even in remote locations, with limited bandwidth.

This paper presents the architecture and the challenges in the implementation of an encrypted voice communication system that uses low bandwidth PTSN lines.

The advantage of the proposed architecture is that it can be easily constructed with minimal implementation costs due to the use of many readymade development boards and open source software solutions.

The system's components can be integrated into a mobile unit that can connect at any location to the switched telephone lines, similar to the Secure Telephone Terminals STU-III [1, 2].

In order to simplify the implementation of the communication system, both hardware and software components were used because:

- The modular design is allowed by the presence of the three main computational units: the cryptographic accelerator with FPGA, the TMS320C6713DSK platform [3] and the MELP voice processing central unit;

- The activities can be grouped (algorithms for signal processing, cryptographic algorithms and system management) based on the particularities of the computational unit;

- The specific need for computational power for the cryptographic operations and the operations of signal processing needed by the MELP vocoder can be met by hardware components.

In the development of the hardware architecture the interconnection methods had to be taken into account.

2. HARDWARE ARCHITECTURE

In order to reduce the implementation costs, the embedded components were grouped resulting in a system with two main components:

- The embedded equipment: built around the TMS320C6713 processor that connects the human interface elements (LCD, keyboard, microphone and speaker) and the telephonic line (modem);

- The PC sub-system that runs the software components and integrates the PCI Express (PCIe) development board with Virtex 5 FPGA.

The Figure 1 presents the functional block diagram of the implemented system.

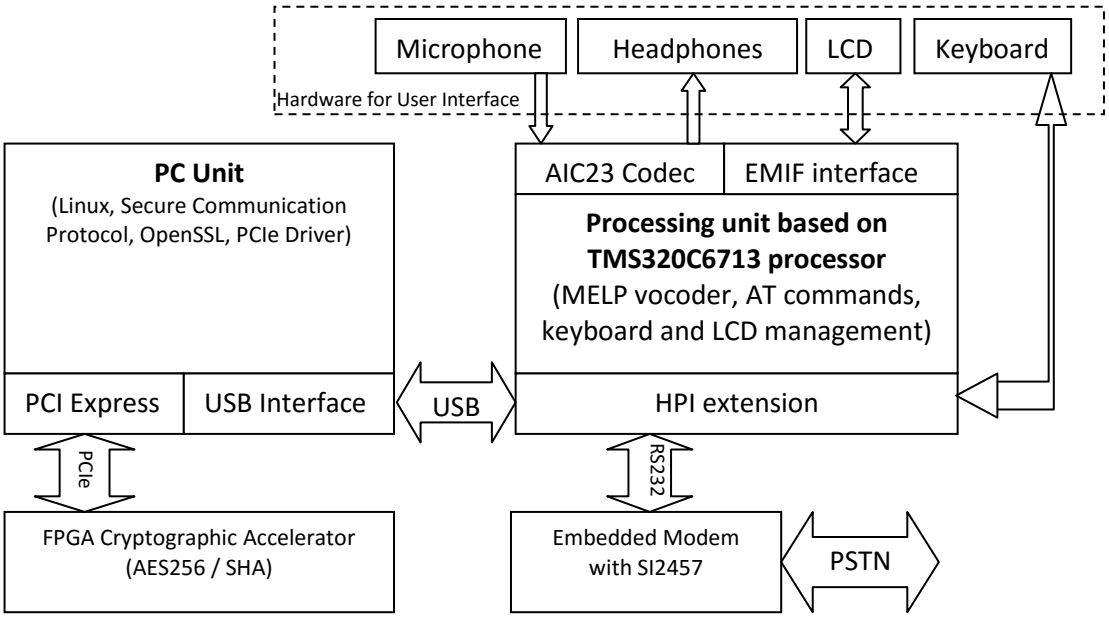

Fig. 1 Architecture of the designed system [4]

The development board TMS320C6713DSK with the TMS320C6713 processor implements the following software functions [4]:

-MELP coding and decoding at 2400bps of the voice signal. The speaker and the microphone are connected to the AIC23 hardware codec from the TMS320C6713DSK;

-the AT commands for PTSN communication using the embedded modem SI2457, commands sent through the RS-232 interface of the HPI (Host Port Int) module. The HPI is a board that allows connections to the development board using a RS232 line, a Universal Serial Bus (USB) line and digital IO signals;

-keyboard and display management, with the keyboard connected to the digital IO lines of the HPI extension using its BD25 connector, and the LCD is connected to the TMS320C6713 processor using the EMIF interface (External Memory Interface);

-the communication with the PC is using the USB interface on the HPI interface.

The following figure presents the two developed systems that make a minimal two node communication network that was used in testing.

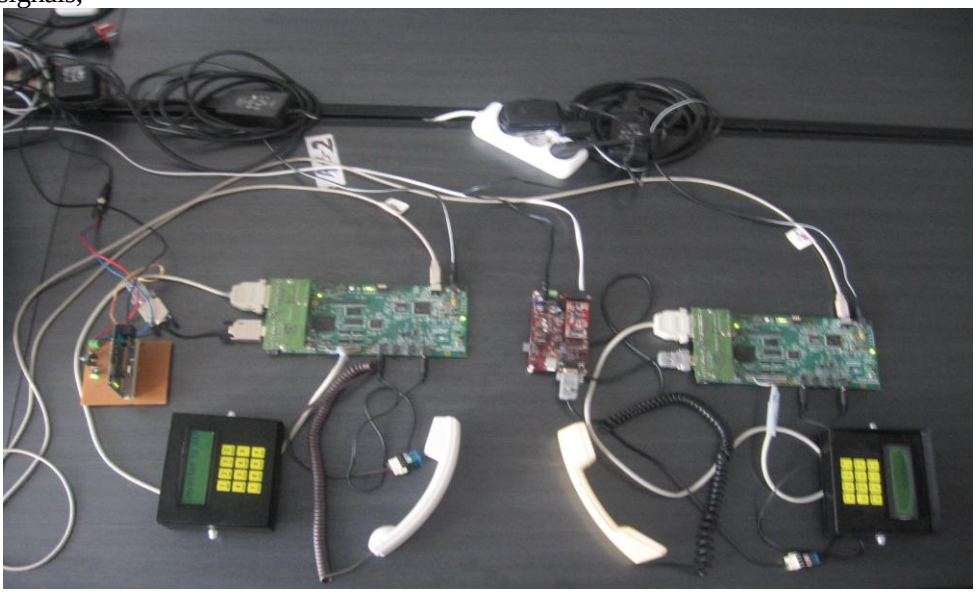

Fig.2 The implemented communication system has two test stations

The PC's CPU used in this implementation does not need high performance because it is aided in its computations by the connected development board (through the USB interface) and the FPGA board (through the PCI Express interface) [5].

The PC has a Linux Suse 10.1 operating system installed using the OpenSSL library and the driver to access the cryptographic accelerator through the PCIe interface.

The cryptographic accelerator using the FPGA circuit implements the cryptographic AES algorithm with keys on 128, 192 and 256 bits in ECB and CBC modes, as well as the Secure Hash Algorithm (SHA) to compute a texts hash. The FPGA also implements the block for PCIe communication.

In order to implement the cryptographic algorithms using programmable logic, the ModelSim and ISE Foundation software tools were used. The results generated by these tools showed that an estimated minimum of 70000 logic cells were needed for the FPGA implementation (in order to satisfy the speed requirements) therefore a Virtex 5 device was used.

The FPGA used gives the possibility to connect the cryptographic accelerator directly to the PCIe interface without the need to acquire a special license because of the Endpoint Block Plus for PCIe [7] offered by Xilinx. The use of the PCIe interface offers the following advantages:

- It is standardized and allows the use of the dedicated cryptographic accelerator with other devices besides those used in this development system.

-It insures a high transfer speed (1-4Gbps) depending on the number of serial channels used (1, 4 or 8);

-It is integrated in the FPGA as a distinct block that needs only to be activated therefore relieving the user from the complex details specific to the PCIe architecture.

Fig. 3 presents the cryptographic accelerator inside the PC connected to the PCIe interface.

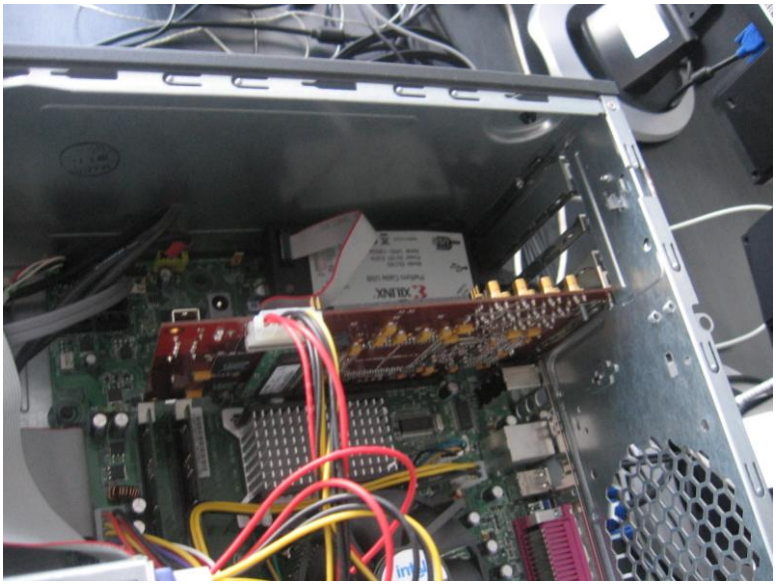

Fig. 3 Virtex 5 board with JTAG programming interface inside the PC

The processor TMS320C6713 has the main role in the implementation of the MELP vocoder with two computational blocks associated to the two main operations:

-encoding: performs the compression of the voice signal and generates the bit sequence that will be sent to the receiver. The computational block receives from the integrated hardware codec AIC23 a set of 180 samples on 16 bits computed by the A/D convertor at 8000Hz, resulting in a 22.5ms length according to the MELP standard. The sampling set is completely introduced in the computational block as a floating point vector. When the computations are over, the block generates a structure that contains the output parameters of the codec that can be sent as 54 bits to the receiver in order to perform the decoding;

-decoding: it generates the initial signal with a quality specific to the MELP vocoder starting from the 54 bits receiver from the encoder block. The block generates a set of 180 samples of 16 bits that are transmitted uniformly in 22.5ms to the hardware codec AIC23 and from it to the D/A convertor to the receiver headset.

The TMS320C6713DSK also implements the keyboard management, the display and also the management of the SI2457 modem.

Figure 4 presents the connectivity of the TMS320C6713DSK board:

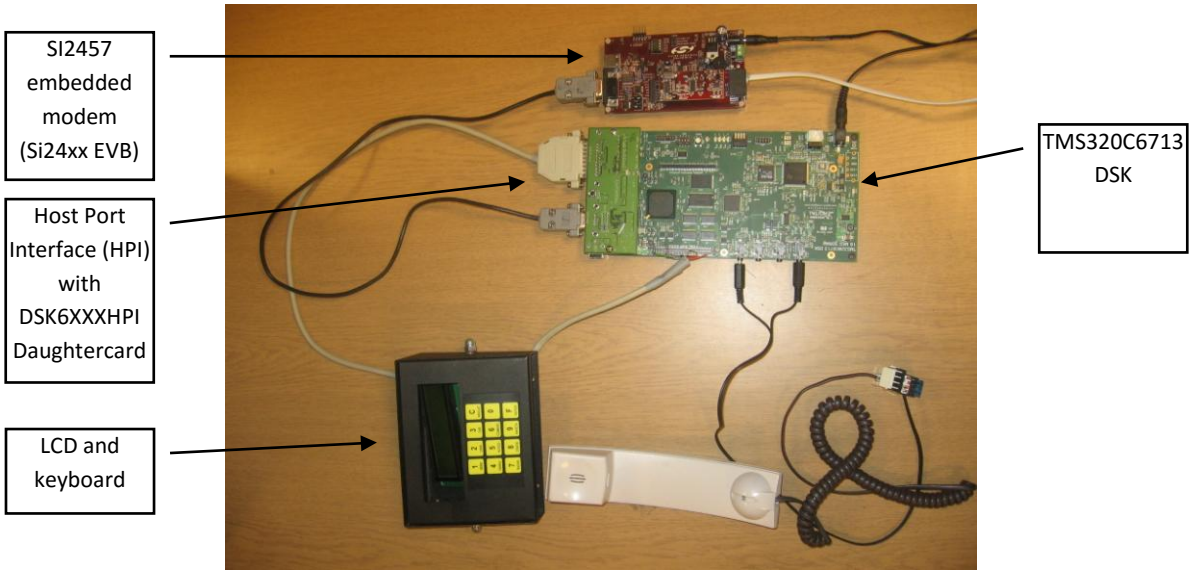

Fig. 4 Connectivity of the TMS320C6713DSK to the phone line is made using the embedded SI2457 modem [6] connected to the HPI module with DSK6XXXHPI Daughtercard

The SI2457 modem connects the system to the PSTN and connects to the RS-232 interface of the TMS320C6713DSK.

The modem is controlled by the vocoder application through AT commands transmitted through the RS232 channel of the HPI extension. The communication through the HPI extension uses a command set available from the extension manufacturer.

3. THE SECURE HASH ALGORITHM BLOCK

The software architecture was developed on top of the the hardware architecture so that the requirements of performance, timing and interconnection are met.

The PC unit has the purpose of implementing the cryptographic functions. The PC manages the operations in the cryptographic accelerator because the functions are performed using the hardware implementations in the FPGA device. The connectivity to the TMS320C6713DSK is made through the USB interface.

The SECURE HASH ALGORITHM (SHA) block presented in Figure 5 uses the SHA-1 algorithm according to the FIPS180-2 standard. The hash generated by this module has 160 bits. Before the message is sent to the sha1 block it is prepared in advance by adding the length, padding, etc. according to the FIPS180-2 standard, resulting a message that has a length multiple of 512. These 512 bit blocks are sent to the sha1 block. Each message is processed, resulting in a 160 bit block. After the last message is processed, the last 160 bits are the final result.

The maximum frequency that the sha1 block can operate at is 87MHz and was determined in the synthesis process.

Fig. 5 SHA block structure

The results obtained following the testing of the SHA1 and SHA256 algorithms regarding the cycle numbers for processing a 512 bit block are presented in the following table.

Table 1. Execution cycle count for the SHA block

Algorithm	Cycle count
SHA1	80
SHA256	65

Considering that the determined working frequency of the hash algorithm of 87MHz, the estimated resulting speed is for each *sha* block is:

V_{sha1}=512Mbps

V_{sha256}=630Mbps

The speed is multiplied with the number of *SHA* blocks instantiated in the design. Figure 6 presents the results of the ModelSim 6.3c simulation for the sequence "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq" (from the FIPS180-2 standard) sent to the inputs of the sha1 and the sha256 blocks.

(a)

(b)

Fig. 6 SHA 1 (a) and SHA256 (b) operation for the FIPS180-2 test input

The output of the SHA1 block is according to the standard: 84983e44 1c3bd26e baae4aa1 f95129e5 e54670f1 and so is the output of the SHA256 block: 248d6a61 d20638b8 e5c02693 0c3e6039 a33ce459 64ff2167 f6ecedd4 19db06c1.

4. THE ADVANCED ENCRYPTION STANDARD BLOCK

The Advanced Encryption Standard (AES) block was described in VHDL language and can encrypt data using the Rijndael algorithm with keys on 128, 192 and 256 bits.

The block gets the key first then the 128 data blocks that will be encrypted.

Next the key expansion is performed for keys used in each round.

Afterwards blocks of 128 bit will be sent to be encrypted using the key expanded in the previous step.

At the end of the encryption process the data will be written as blocks of 128 bits.

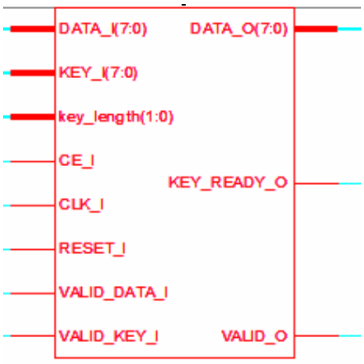

Fig. 7 AES block structure

In order to obtain the desired speed of decryption and encryption it was necessary to use the multiplication capabilities in the FPGA for the AES computations and it was necessary to implement a special dispatcher block for the data management at the entry of the AES blocks.

The early tests performed showed that the transfers between the test PC unit and the encryption module using programmed transfer instructions on the PCIe x1 interface have a top speed of 10Mbits/s which is not enough from the perspective of the encryption module.

This low speed was due to the software/driver/operating system but also because each TLP packet used for these transfers has in its structure only 4 usable bytes of data [10]. These constraints are generated by the firmware in the complex root (chipset) for all transfers initiated by the CPU and the driver/system is not able to intervene in order to optimize the transfer.

In order to connect the cryptographic implementation in the FPGA to the PC, the Endpoint Block Plus for PCIe from Xilinx was used. The block implements the PCIe 1.1 standard and ensures the connectivity using 1, 4 or 8 serial lanes.

To increase the speed it was necessary to use the Direct Memory Access (DMA) mechanism. This was possible because the interface block Endpoint Block Plus for PCIe from Xilinx is capable to behave like a master on the PCIe interface and can initiate transfers between the FPGA and the PC memory without the processor intervention, at transfer rates specific to the PCIe 1.1 standard.

The advantage of this approach is that the PC processor doesn't have to handle this time consuming activity, and that the transfers are performed rapidly using the optimized firmware present in the FPGA interface block (endpoint) and the motherboard (root complex).

The implemented module functions as follows: in order to make a transfer to or from the FPGA to the PC, the PC application programmatically loads the configuration registers needed for the transfer (PC memory address, transfer length, etc.), then the transfer is initiated. After the transfer has ended an interrupt is generated by the FPGA and captured by the PC in the driver/application.

From the perspective of the PC unit, in order to connect the cryptographic accelerator to the CPU using the PCIe interface, it was necessary to create a kernel driver that implements the direct access operations to the hardware resources (memory and interrupts) and other operations required by a multitasking environment. Even if the Linux operation system is open-source and already has drivers for the PCIe interface, the special driver was necessary because of the specific functions of the cryptographic accelerator.

The communication with the module at application level is performed through a device module driver [9]. The driver implements the base functions: open, close, write (to the module), read (from the module). At operating system level the module is seen as a special type of device, created once the kernel is loaded. This is not a usual file, but a device type that has a physical device attached.

In the Linux operating system the drivers, depending of the way the data is transferred from the driver to the device, can be of character type or block type, with each mode having its advantages and disadvantages. In this case it is necessary to transfer data in variable quantities, therefore the solution of character type drivers was chosen. Block type drivers are recommended in the file system transfers, where large quantities of data are transferred in fixed size data blocks. For the implemented character driver the functions needed to be implemented in order to insure the compatibility and interoperability with the other kernel components are:

a) Functions for the interface with the user applications;

b) Interface functions to the operating system kernel;

c) Functions called by the operating system at the driver load or removal from the memory.

5. THE CRYPTOGRAPHY BLOCK IMPLEMENTATION

The cryptographic operations block implements the AES and SHA algorithms. The communication with this block is performed through two 32 bit ports. The block signals through the ports: RX_wr_en, Tx_rd_en, TX_empty, RX_full.

The cryptographic block receives commands and data on the input interface (RX_din(31:0)) and writes the results on the output interface (TX_dout(31:0).

The block waits on the input interface for a succession of packets composed from header and payload.

The cryptographic block will write, in order, processed data to the output port after receiving the respective command.

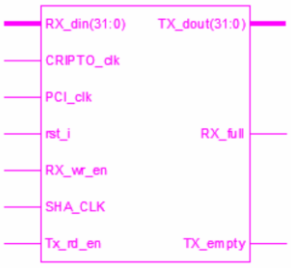

Fig. 8 Cryptographic operations block structure

The figure 9 presents the components of the cryptographic module and their hierarchy.

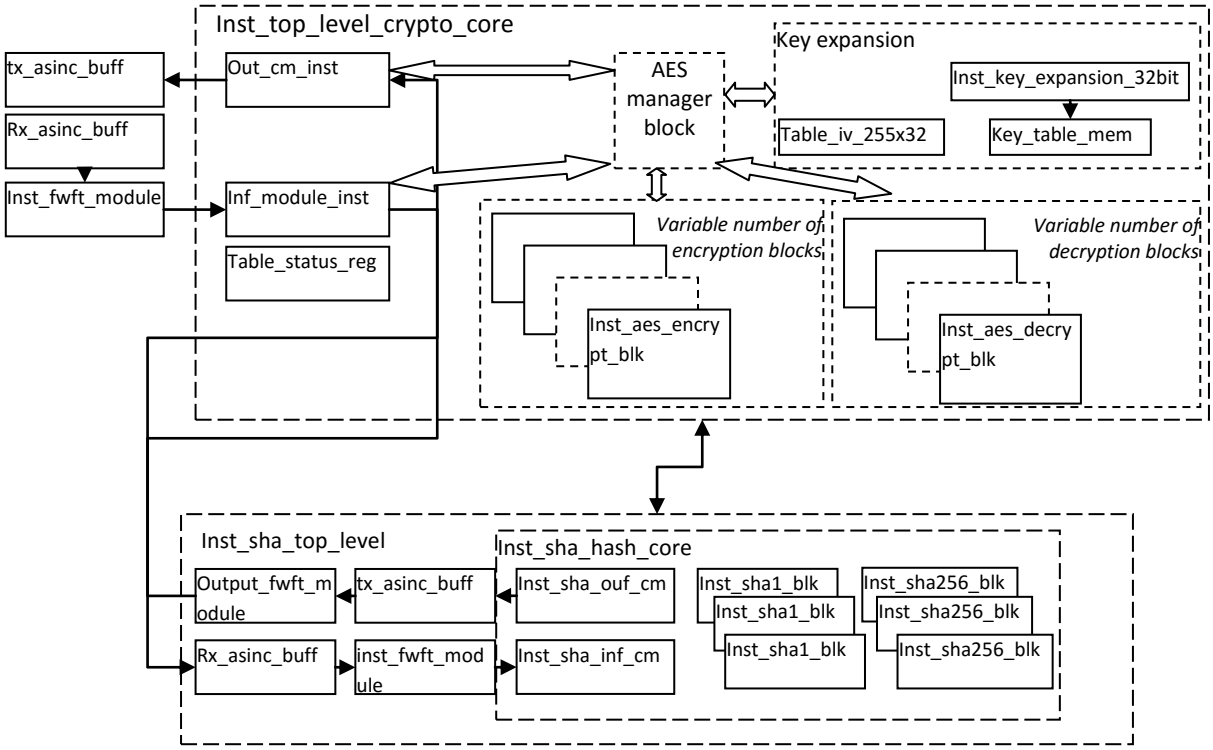

Fig. 9 Structure of the designed and implemented Cripto_design block [5]

In the presented architecture, the maximum number of parallel encryption and decryption blocks varies based on the number of logic cells present in the used FPGA circuit.

The implemented module supports a maximum of 256 simultaneous cryptographic contexts, each context having a unique identifier.

The access logic to these parallel modules is controlled and optimized by a control module.

After the testing of the AES algorithm in the FPGA, the following execution times resulted depending on the operation type (key expansion, encryption, and decryption) and key length.

The length is expressed in the number of cycles necessary for the execution, and the execution time is computed based on the frequency that the cryptographic block can operate at, as reported by the synthesis process.

Table 2. Execution Cycles for different cryptographic operations

Operation \ Key Size	128 bits	192 bits	256 bits
key expansion	133	123	151
16 octet block encryption	99	119	139
16 octet block decryption	91	109	131

Considering that the frequency reported by the synthesis software of approximately 200MHz, and the average length of 100 clock cycles for each encryption/decryption operation, this will result in an approximate duration for processing each of the 16 bytes blocks of:

T = 1 / ((200*1.000.000) / 100) = 1 / (2* 1.000.000) = 0, 5 μs.

This gives the encryption/decryption speed for each AES block of:

V=32 MB/sec=512 Mbps

Because the solution allows the simultaneous functioning of up to 256 AES computational blocks, the number being limited only by the FPGA cell count, we obtain a maximum possible speed of:

V_{max}=256*512Mbps=131172Mbps=131Gbps

This speed concerns only the performance of the AES computational block and does not take into account the data transfer delays from the FPGA to the PC that depends on the used PCIe interface.

6. INTEGRATION OF THE OPENSSL LIBRARY

Because the PC would implement the cryptographic algorithm directly through a library, several aspects were studied regarding the operations with asymmetric algorithms in order to have a secure communications protocol:

-public key cryptography using X.509 digital certificates;

-establishing the keys using the protocol ECMQV based on operations with elliptic curves;

-digital signatures using the DSA algorithm Elliptic Curve Digital Signature Algorithm (ECDSA).

Starting from these considerations, there are many cryptographic libraries that offer support for a secure communication protocol, but after being analyzed [8], the best suited for this operation was the OpenSSL library. This is because it has the following characteristics: it is open source and it is maintained and updated by an international community; offers support in the area of public key cryptography using elliptic curves; it is implemented and tested on Linux operating systems in many commercial applications; it can be easily integrated with the FPGA cryptographic accelerator in order to implement the AES and SHA algorithms.

The figure 10 presents the n-tier architecture with the software packages used for the cryptographic accelerator integration.

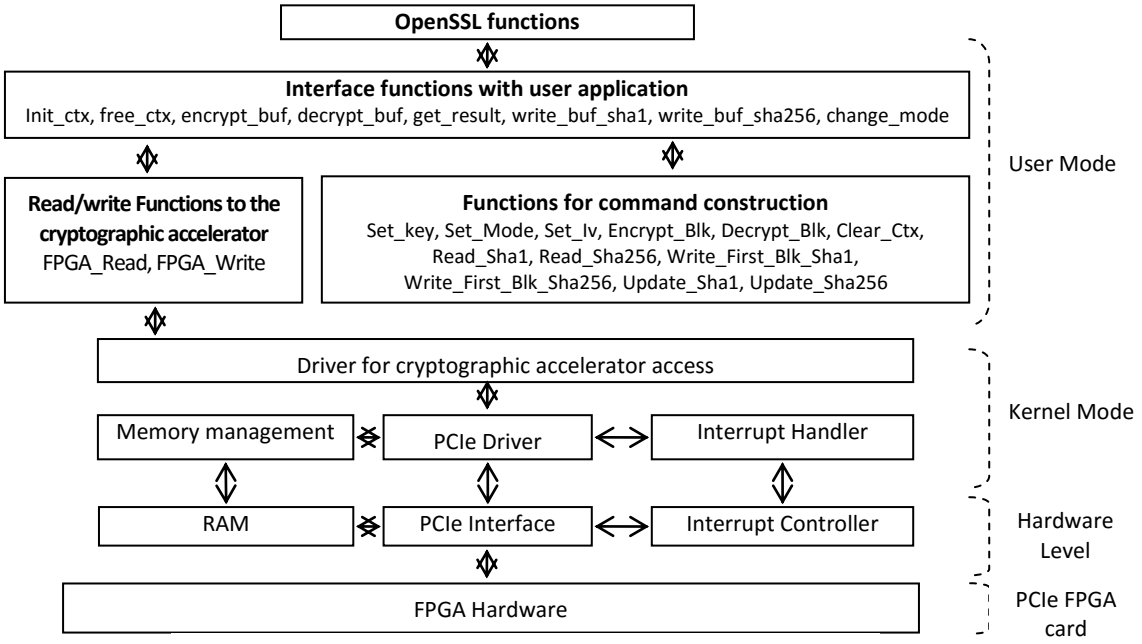

Fig. 10 Integration of the OpenSSL library with the hardware

For the integration of the OpenSSL library with the cryptographic hardware it was necessary to make modifications in many the library files so that external hardware will be called instead of software routines [8]. For this purpose an API was developed and was integrated by intercepting function calls passed between software components [9]. The intercepting is done by modifying the low level OpenSSL functions so that they call functions from the developed library (which in turn will access hardware functions) instead of the default ones.

The CSVDT_MODULE symbol was defined that will control the compilation of the OpenSSL packet with or without the support for the developed platform depending on the compilation parameters. In order to compile for the developed platform, the CSVDT_MODULE directive was added to compile the sources in the OpenSSL package.

If the CSVDT_MODULE directive is used, the library will used the modified software routines instead of the default ones.

The file aes.c was modified by adding the following sequence:

```
/// for using the module CSVDT with its library
#ifdef CSVDT_MODULE
#include "fpga_cmd.h"
extern CSVDT_AES_CTX ctx[MAX_CTX];
int fd = -1; /// file descriptor for the special
device file (/dev/fpga-driver)
#endif /// CSVDT_MODULE
```

For example the file *aes_cbc.c* was modified in order to encrypt the input buffer in the CBC encryption mode.

Each block will be sent to the cryptographic accelerator using the function: Encrypt_Buf(fd,ctx_id, in, 16), and the result will be obtained using the function: Get_Result(fd, &ctx_id,&seq_id, &payload_length, &error_flag,&instr_code, &instr_type, out); the modifications at function level are presented as follows:

```
#ifdef CSVDT_MODULE
unsigned int ctx_id = 1; /// the id 1 is used
int ret;
unsigned int seq_id;
unsigned int payload_length;
unsigned int error_flag;
unsigned int instr_code;
unsigned int instr_type;
assert((AES_ENCRYPT == enc)||(AES_DECRYPT ==
enc));
printf ("AES_cbc_encrypt - with CSVDT module...\n");
if (AES_ENCRYPT == enc)
{
assert (fd != -1); /// open  device
///change to CBC mode
ret = Change_Mode (fd, ctx_id, CBC_MODE);
assert (ret == CSVDT_SUCCESS);
while (len >= AES_BLOCK_SIZE)
{
/// send block for encryption to device
ret = Encrypt_Buf(fd, ctx_id, in, 16);
assert (ret == CSVDT_SUCCESS);
/// get the result
ret = Get_Result(fd, &ctx_id, &seq_id, &payload_length,
&error_flag, &instr_code,
&instr_type, out);
assert (ret == CSVDT_SUCCESS);
len -= AES_BLOCK_SIZE;
in += AES_BLOCK_SIZE;
out += AES_BLOCK_SIZE;
}
if (len) { /// encrypt the final bloc (with padding)
for(n=0; n < len; ++n) out[n] = in[n];
for(n=len; n < AES_BLOCK_SIZE; ++n) out[n] = iv[n];
/// send block to device for encryption
ret = Encrypt_Buf(fd, ctx_id, in, 16);
assert (ret == CSVDT_SUCCESS);
/// get result
ret = Get_Result(fd, &ctx_id, &seq_id, &payload_length,
&error_flag, &instr_code,
&instr_type, out);
    assert (ret == CSVDT_SUCCESS);
    }
    }
    #endif CSVDT_MODULE
```

7. CONCLUSIONS

The integration of all the components of the system allowed the verification of the unit functionality after solving the problems introduced by the used communication interfaces.

Communication protocols had to be implemented between the components that would allow bringing the equipment back to functional status in case of errors would appear.

The tests performed showed that the communication sequence introduces large delays and in order to optimize the average

communication speed, larger packets are recommended to be used.

The advantage of the solution is the ease of implementation due to commercial hardware boards and open source software however the level of integration for the final system is reduced. The solution allows a professional security level system that can connect to PSTN communication lines in conditions of low bit rates (2400bps – minimum requirement of the MELP vocoder).

In order to increase the level of integration of the solution a single hardware module can be constructed based on the Virtex and TMS320C6713 DSK boards with the software components integrated into the FPGA. This would increase the level of integration however the complexity of the designed system would increase as there are no longer pre-configured components to be used.

If the secure communications protocol follows the Security Communications Interoperability Protocol (SCIP) standard (the hardware and software components used for the implementation allow this compatibility) the solution can be used for professional secure military communications.

8. REFERENCES

[1]. Department of Defense Security Institute - "STU-III HANDBOOK FOR INDUSTRY", http://www.tscm.com/STUIIIhandbook.html, accessed 20.10.2011

[2]. NSA - "Secure Telephone Unit Third Generation (STU-III)/ Secure Terminal Equipment (STE)", http://www.fas.org/irp/program/security/_work/stu3.html, accessed 20.10.2011

[3]. SPECTRUM DIGITAL, INC - "TMS320C6713 DSK: Technical Reference, Start Guide", http://c6000.spectrumdigital.com/dsk6713/V2/docs/dsk6713_TechRef.pdf,

[4]. V. Togan, L. Apostol – "Integrarea Elementelor Componente In Cadrul Echipamentului CSVDT", University of Pitesti Scientific Bulletin, Series: Electronics and computers science, Special Issue: Tehnologii şi echipamente de comunicaţii securizate de voce şi de date pe reţele telefonice comutate, ISSN - 1453 – 1119

[5]. M. Togan, A. Floarea – "Accelerator Criptografic Cu Circuit FPGA", University of Pitesti Scientific Bulletin, Series: Electronics and computers science, Special Issue: Tehnologii şi echipamente de comunicaţii securizate de voce şi de date pe reţele telefonice comutate, ISSN - 1453 – 1119

[6]. Silicon Laboratories, "Embedded Modem Development Kit User's Guide", http://www.silabs.com/Support%20Documents/TechnicalDocs/MODEM-DK.pdf, accessed 20.10.2011

[7] Xilinx Inc - "Virtex-5 LogiCORE Endpoint Block Plus for PCI Express Designs User Guide", http://www.xilinx.com/support/documentation/ip_documentation/pcie_blk_plus_ug341.pdf, accessed 20.10.2011

[8] V. Togan, A. Fejer - Operatii Criptografice Cu Chei Publice Specifice Protocolului SCIP, University of Pitesti Scientific Bulletin, Series: Electronics and computers science, Special Issue: Tehnologii şi echipamente de comunicaţii securizate de voce şi de date pe reţele telefonice comutate, ISSN - 1453 – 1119

[9] Jonathan Corbet, Greg Kroah-Hartman, Alessandro Rubini, "Linux Device Drivers, 3rd Edition", Ed. O'Reilly, 2005

[10] Ravi Budruk, Don Anderson, Tom Shanely, "PCI Express System Architecture", Ed. Addison Wesley, 2003,ISBN-13: 978-0321156303