Hardware / Software Co-Design using LEON3 Processor: AES as Case Study

Priti S. Chimankar
Research Scholar
Department of Electronics Engineering,
Shri Ramdeobaba College of Engineering and Management,
Nagpur, India

Meghana A. Hasamnis
Associate Professor
Department of Electronics Engineering,
Shri Ramdeobaba College of Engineering and Management,
Nagpur, India

S. S. Limaye, Ph.D
Principal
Jhulelal Institute of Technology,
Nagpur, India

ABSTRACT
Nowadays many powerful public domain IP cores are available for complicated component like 32 bit processor i.e. LEON3. It needs considerable expertise and pain taking experimentation to implement a hardware/software co-design project. This paper presents step-by-step description for AES algorithm implementation on LEON3 processor. This will prove to be valuable to researchers working in this area and save their valuable time.

The concept of GPIO (General Purpose I/O Port) is introduced; through which any custom hardware i.e. own designed hardware or IP core can be interfaced with the open source processor. AES encryption algorithm is selected as an IP core to be interfaced with LEON3 processor. AES is implemented in VHDL, while the control of the algorithm is in software. AES algorithm partitioned in hardware and software. The complete algorithm in hardware and control of algorithm in software. The part of algorithm in hardware is interfaced with the system designed using processor as a custom hardware and performance parameters studied. AES implemented using Codesign approach. AES is the latest encryption standard used to protect confidential information like financial data for government and commercial use. The LEON3 is a synthesizable VHDL model of a 32-bit processor available under the GNU GPL license. The design is implemented on Cyclone II FPGA from Altera Corporation.

Keywords
Advanced Encryption Standard (AES), LEON3 Processor, GPIO (General Purpose I/O Port), Cyclone II FPGA.

1. INTRODUCTION
An embedded system is a combination of processor along with software specifically designed for a particular function. With increasing complexity of the systems, sub-micron technologies are used significantly. Device density and complexity of embedded applications also increases, with requirements for high performance, miniaturization, long battery life etc. Advanced ASIC & FPGA technologies allow to integrate complex systems on a single chip. For high processing performance, reducing power consumption SoC (System on chip) technology is used [1, 2].

Different commercial packages are available for System on Chip design like Xilinx EDK or Altera Nios II IDE, for the design of embedded system. They provide proprietary soft cores and the tools needed for implementing them in the manufacturer's FPGAs. These commercial solutions have certain limitations. The most important is that the implemented soft cores are dependent on the manufacturer's specific hardware. These soft cores are also closed source, so modifications or enhancements to these soft cores are impossible. For low budget projects, the cost of these software packages is unaffordable. Therefore, it is preferable to use open source cores [3] which are freely available from open source communities, for example open source core LEON3 by Gaisler Research [4, 5] and Open RISC 1200 from open cores.

2. AES ALGORITHM
The National Institute of Standards and Technology, (NIST), solicited proposals for the Advanced Encryption Standard (AES). The AES is a Federal Information Processing Standard, (FIPS), which is a cryptographic algorithm that is used to protect electronic data [6]. The AES algorithm is a symmetric block cipher that encrypt (encipher), and decrypt (decipher), information. Encryption converts data to an unintelligible form called cipher-text. Decryption of the cipher-text converts the data back into its original form, which is called plaintext. 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 [7]. The Rijndael algorithm was developed by Joan Daemen of Proton World International and Vincent Rijmen of Katholieke University at Leuven.

The structure of AES-128 Encryption is shown below in "Figure 1". The initial 128-bit key is fed into the Key Expansion function which produces separate keys for each of the 10 required rounds. These rounds combine their scheduled keys with a two dimensional representation of the input (the State") using various transformations:

- Addroundkey –The Addroundkey transformation Addroundkey ( ), adds the round key to the State using a bitwise XOR operation.
- Subbytes - The bytes substitution transformation Subbyte ( ) is a non-linear substitution of bytes. It operates independently on each byte of the State using a substitution table (S-box).
- Shiftrows - In the Shift Rows transformation Shiftrows ( ), the bytes in the last three rows of the State are cyclically shifted over different numbers of bytes (offsets). The first row, $r = 0$, is not shifted.
- Mixcolumns –The mix column transformation Mixcolumns ( ), separately modifies each column of the state using a matrix multiplication operation [8].

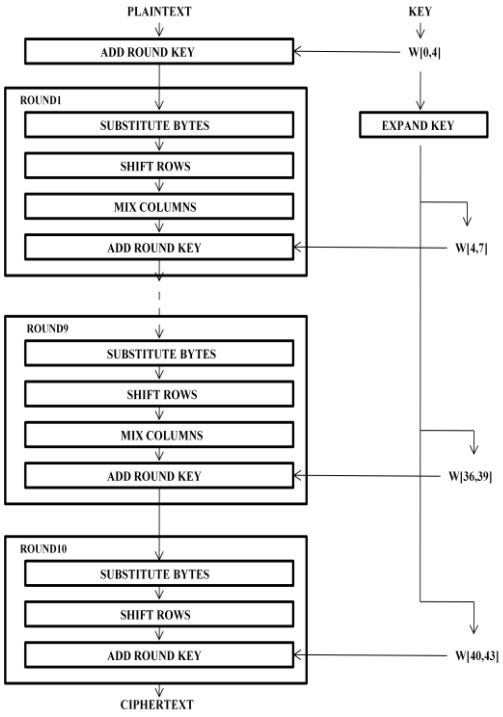

Fig 1: AES Encryption [7]

3. DESIGN ENVIRONMENT

The design environment, consists of LEON3 processor, and is developed on CentOS-5.4 operating system, the Linux platform.

LEON3 is a 32-bit CPU microprocessor core conforming to the IEEE-1754 (SPARC V8) architecture and instruction set and is a part of GRLIB IP library. The LEON3 processor is designed for embedded applications, combining high performance with low complexity and low power consumption. The model is highly configurable, support power-down mode and clock gating with robust and fully synchronous single-edge clock design [4].

LEON3 processor uses the AMBA-2.0 AHB bus to connect the main processor with high-speed controller like cache, memory and other optional units like the on chip RAM or PCI or Ethernet interfaces. Another AMBA-2.0 APB bus is used to access most on chip peripherals. It is optimized for simple operation and low-power consumption. It is connected to the AHB and LEON3 via the AHB/APB Bridge, which is master of that bus. LEON3 external memory access is provided by a programmable memory controller with interfaces to PROM, SRAM, SDRAM and memory mapped I/O peripherals. The controller can decode a map of up to 2 Gbytes. The processor is extensively configurable and can be efficiently implemented on both FPGAs and ASIC technologies [3].

The architectural block diagram of LEON3 processor is shown in "Figure 2".

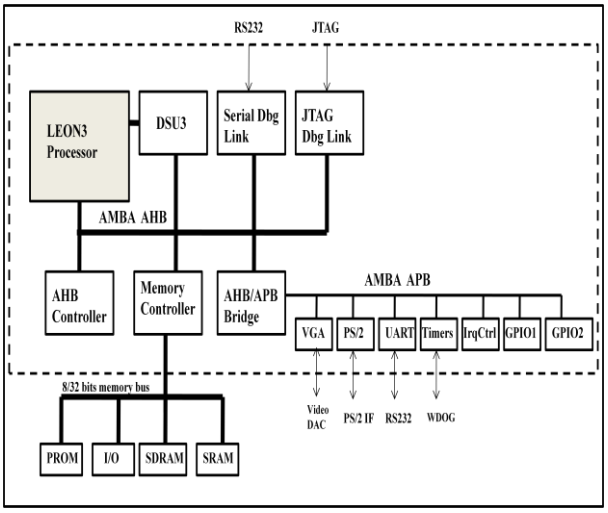

Fig 2: LEON3 Template Design Block Diagram [5]

4. INTERFACING AES AS CUSTOM HARDWARE WITH LEON3 PROCESSOR

To interface AES as custom hardware with LEON3 Processor, there are two possible ways,

- Interfacing AES as custom hardware to AMBA AHB bus or AMBA APB bus.
- Interfacing AES as custom hardware to AMBA APB through GPIO's.

Since, the design which we have selected i.e. Leon3-altera-de2-ep2c35, from GRLIB IP Library, has two General Purpose I/O Port (GPIO), interfaced with LEON3 Processor, it is more flexible and relatively easy, to interface AES through GPIO's. The detailed description of GPIO and how it is used to interface AES with LEON3 processor is given below.

4.1 GPIO – General Purpose I/O Port

The general purpose input output port core provided by Gaisler Research, under the template design, is scalable. It provides optional interrupt support. The port width can be set to 2 - 32 bits through the n bits VHDL generic (i.e. n bits = 16). Each bit in the general purpose input output port can be individually set to input or output, and can optionally generate an interrupt. It is possible to share GPIO pins with other signals. The output register can then be bypassed through the bypass register. The "Figure 3" shows a diagram for one I/O line.

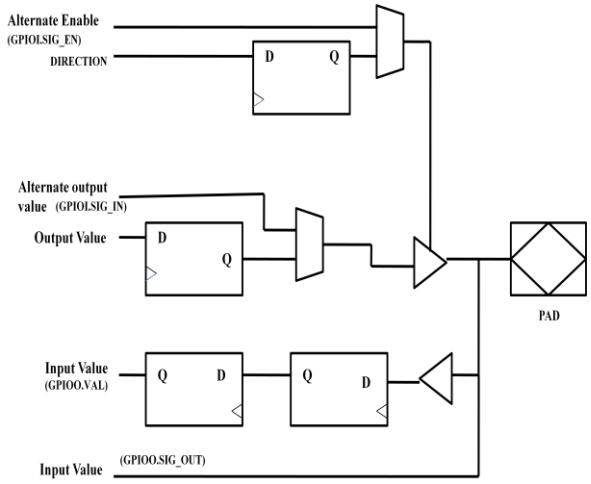

Fig 3: General Purpose I/O Port Diagram [4]

The I/O ports are implemented as bi-directional buffers with programmable output enable. The input from each buffer is synchronized by two flip-flops in series to remove potential meta-stability. The synchronized values can be read-out from the I/O port data register. They are also available on the GPIOO.VAL signals. The output enable is controlled by the I/O port direction register. A '1' in a bit position will enable the output buffer for the corresponding I/O line. The output value driven is taken from the I/O port output register [4].

A GPIO pin can be shared with other signals. The ports that should have the capability to be shared are specified with the bypass generic (the corresponding bit in the generic must be 1).The unfiltered inputs are available through GPIOO.SIG_OUT and the alternate output value must be provided in GPIOI.SIG_IN. The bypass register then controls whether the alternate output is chosen. The direction of the GPIO pin can also be shared, if the corresponding bit is set in the bpdir generic. In such case, the output buffer is enabled when GPIOI.SIG_EN is active.

4.2 AES Interfaced with LEON3 Processor

"Figure 4" shows the LEON3 processor architecture, where AES block is connected through GPIO's to LEON3 processor.

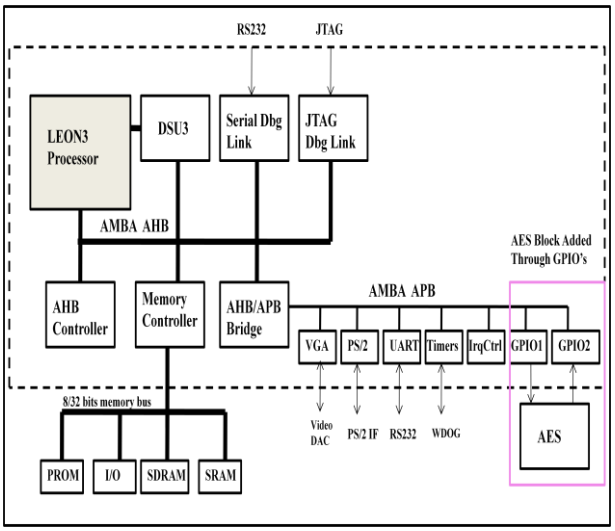

Fig 4: AES Core Interfaced With LEON3 Processor

Since the GPIO is of 32 bits, but the plaintext and key input that we have to provide to the AES block are of 128 bits, 32 bits of GPIO port are not sufficient and also some of the pins are also required for the control signals. So to solve this problem, we have placed one demultiplexer before AES core and one multiplexer after AES core. The diagram depicting this is shown in "Figure 5" below.

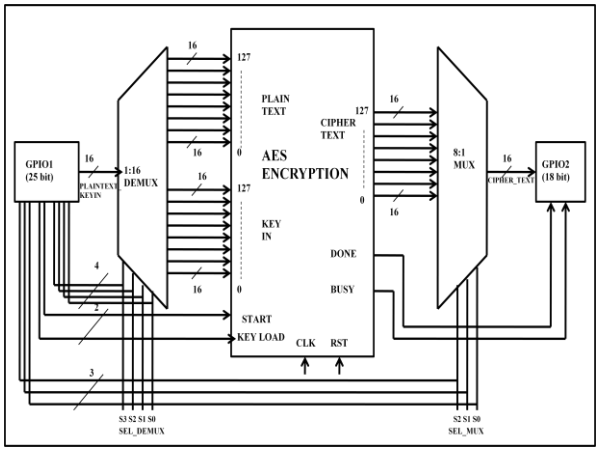

Fig 5: Block Diagram of AES Interfaced with GPIO

The template design selected, has two GPIO's, out of which one is used to provide input to the AES core and other is used to take output from AES core. GPIO1 is configured for 25 bits, out of which 16 LSB line are used to provide plaintext and key input through demultiplexer and remaining 9 lines are used to provide control signals. GPIO2 is configured for 18 bits, out of which 16 LSB are used to take output i.e. cipher text , and remaining two lines are to provide indication of whether the encryption is complete or not.

5. DESIGN STEPS

5.1 Configuration of LEON3

The first step in design process is the configuration of LEON3 processor according to our requirements. For the configuration of LEON3 processor, first enter into the directory grlib-gpl-1.1.0-b4113/designs/LEON3-altera-de2-ep2c35 and then give the command, *make xconfig*

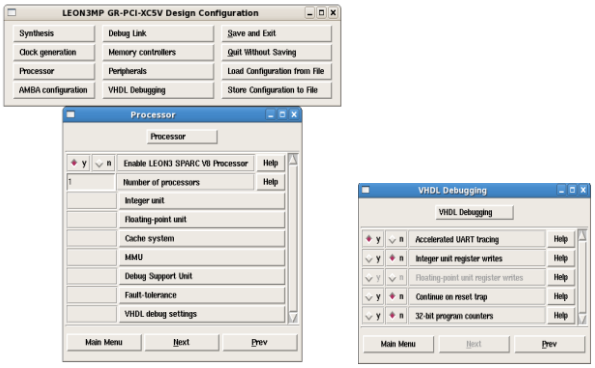

Fig 6: Configuration of LEON3 Processor

Using GUI interface configure various aspects of LEON3 processor. Click the 'Processor' button and it will give various options for configuring integer unit, floating-point unit, cache

system, memory management unit, debug support unit etc. For this project we have disabled the floating-point unit and memory management unit and enabled debug support unit and Accelerated UART tracing. To copy the configuration to the LEON3-altera-de2-ep2c35/config.vhd file click on 'save and exit'.

5.2 Compilation of control program written in C

To compile the control program written in C, we have to first copy the program in systest.c of directory grlib-gpl-1.1.0-b4113/designs/LEON3-altera-de2-ep2c35 and then give command, *make soft*

The program in systest.c get compiled into an sdram.srec file which is loaded into the memory of the processor while simulation [8].

5.3 Compilation of AES Encryption core

The design can be compiled by giving the following command in grlib-gpl-1.1.0-b4113/designs/Leon3-altera-de2-ep2c35 directory, *make vsim.* It compiles all .vhd files [9].

5.4 Simulation

To simulate the design, first enter into the grlib-gpl-1.1.0-b4113/designs/LEON3-altera-de2-ep2c35 directory and then give command, *vsim testbench*

The subdirectory 'software' contains all the test files for the processor. Each test has been described in a separate file. These tests are compiled into an sdram.srec file which is loaded into the memory of the processor while simulation. To view any signals, add the desired signals from the right hand side pane to the waveforms. Finally to start the simulation give command, *run - all*

It runs the simulation completely. Simulation is halted by generating a failure. Simulation report is shown below "Figure 7".

5.5 Synthesis

The template design can be synthesized with either Synplify, Quartus or ISE/XST. To synthesize the design using Quartus II, enter into the grlib-gpl1.1.b4113/designs/LEON3-altera-de2-ep2c35 directory and then give command , *make quartus*

It generates leon3mp.qpf and leon3mp.sof in the same directory [8].

```
VSIM 6> run -all
# ** Note: Cyclone II PLL is enabled
#    Time: 0 ps  Iteration: 2  Instance: /testbench/d3/clkgen0/sden/altpll0/cycloneii_altpll/m3
# LEON3 Altera DE2-EP2C35 Demonstration design
# GRLIB Version 1.1.0, build 4113
# Target technology: stratixii ,  memory library: stratixii
# ahbctrl: AHB arbiter/multiplexer rev 1
# ahbctrl: Common I/O area at 0xfff00000, 1 Mbyte
# ahbctrl: AHB masters: 4, AHB slaves: 8
# ahbctrl: Configuration area at 0xfffff000, 4 kbyte
# ahbctrl: mst0: Gaisler Research      LEON3 SPARC V8 Processor
# ahbctrl: mst1: Gaisler Research      AHB Debug UART
# ahbctrl: mst2: Gaisler Research      JTAG Debug Link
# ahbctrl: mst3: Gaisler Research      SVGA frame buffer
# ahbctrl: slv0: European Space Agency  LEON2 Memory Controller
# ahbctrl:      memory at 0x00000000, size 512 Mbyte, cacheable, prefetch
# ahbctrl: slv1: Gaisler Research      AHB/APB Bridge
# ahbctrl:      memory at 0x80000000, size 1 Mbyte
# ahbctrl: slv2: Gaisler Research      LEON3 Debug Support Unit
# ahbctrl:      memory at 0x90000000, size 256 Mbyte
# ahbctrl: slv3: Gaisler Research      PC133 SDRAM Controller
# ahbctrl:      memory at 0x40000000, size 8 Mbyte, cacheable, prefetch
# ahbctrl:      I/O port at 0xfff00100, size 256 byte
# ahbctrl: slv7: Gaisler Research      Test report module
# ahbctrl:      memory at 0x20000000, size 1 Mbyte
# apbctrl: APB Bridge at 0x80000000 rev 1
# apbctrl: slv0: European Space Agency  LEON2 Memory Controller
# apbctrl:      I/O ports at 0x80000000, size 256 byte
# apbctrl: slv1: Gaisler Research      Generic UART
# apbctrl:      I/O ports at 0x80000100, size 256 byte
# apbctrl: slv2: Gaisler Research      Multi-processor Interrupt Ctrl.
# apbctrl:      I/O ports at 0x80000200, size 256 byte
# apbctrl: slv3: Gaisler Research      Modular Timer Unit
# apbctrl:      I/O ports at 0x80000300, size 256 byte
# apbctrl: slv4: Gaisler Research      LCD Controller
# apbctrl:      I/O ports at 0x80000400, size 256 byte
# apbctrl: slv5: Gaisler Research      PS2 interface
# apbctrl:      I/O ports at 0x80000500, size 256 byte
# apbctrl: slv6: Gaisler Research      SVGA frame buffer
# apbctrl:      I/O ports at 0x80000600, size 256 byte
# apbctrl: slv7: Gaisler Research      AHB Debug UART
# apbctrl:      I/O ports at 0x80000700, size 256 byte
# apbctrl: slv9: Gaisler Research      General Purpose I/O port
# apbctrl:      I/O ports at 0x80000900, size 256 byte
# apbctrl: slv10: Gaisler Research      General Purpose I/O port
# apbctrl:      I/O ports at 0x80000a00, size 256 byte
# apbctrl: slv11: Gaisler Research      General Purpose I/O port
# apbctrl:      I/O ports at 0x80000b00, size 256 byte
# apbctrl: slv10: Gaisler Research      General Purpose I/O port
# apbctrl:      I/O ports at 0x80000a00, size 256 byte
# apbctrl: slv11: Gaisler Research      General Purpose I/O port
# apbctrl:      I/O ports at 0x80000b00, size 256 byte
# apbctrl: slv15: Gaisler Research      AHB Status Register
# apbctrl:      I/O ports at 0x80000f00, size 256 byte
# testmod7: Test report module
# svgactrl6: SVGA controller rev 0, FIFO length: 384, FIFO part length: 128, FIFO address bits: 9, AHB access size: 32 bits
# apbps2_5: APB PS2 interface rev 2, irq 5
# ahbstat15: AHB status unit rev 0, irq 1
# grgpio11: 32-bit GPIO Unit rev 1
# grgpio10: 18-bit GPIO Unit rev 1
# grgpio9: 25-bit GPIO Unit rev 1
# gptimer3: GR Timer Unit rev 0, 16-bit scaler, 2 32-bit timers, irq 8
# irqmp: Multi-processor Interrupt Controller rev 3, #cpu 1, eirq 0
# apbuart1: Generic UART rev 1, fifo 4, irq 2, scaler bits 12
# apblcd4 : APB LCD module rev 0
# sdctrl163: PC133 SDRAM controller rev 1
# ahbjtag AHB Debug JTAG rev 1
# ahbuart7: AHB Debug UART rev 0
# dsu3_2: LEON3 Debug support unit + AHB Trace Buffer, 2 kbytes
# leon3_0: LEON3 SPARC V8 processor rev 0
# leon3_0: icache 2*4 kbyte, dcache 1*4 kbyte
# ** Note: Cyclone II PLL locked to incoming clock
#    Time: 110 ns  Iteration: 4  Instance: /testbench/d3/clkgen0/sden/altpll0/cycloneii_altpll/m3
#
#
#  PLAINTEXT FOR ENCRYPTION
#
#  32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34
#
#  KEY FOR ENCRYPTION
#
#  2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c
#
#
#
#
#
#  TEXT AFTER ENCRYPTION
#
#  39  25  84  1d  02  dc  09  fb  dc  11  85  97  19  6a  0b  32
#
# ** Failure: *** IU in error mode, simulation halted ***
#    Time: 2345192 ns  Iteration: 0  Process: /testbench/iuerr File: testbench.vhd
# Break in Process iuerr at testbench.vhd line 206

VSIM 6>
```

Fig 7: Simulation result on console window

6. RESULT

Simulation result of AES Encryption core interfaced with LEON3 processor on Modelsim is shown in "Figure 8" below:

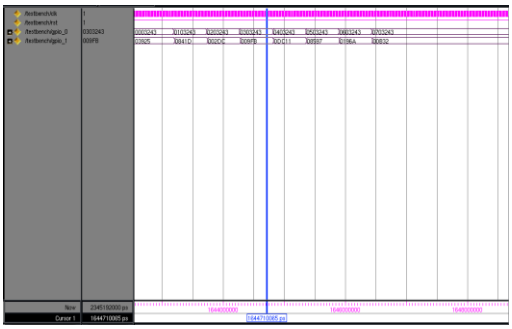

Fig 8: Simulation result on Modelsim

AES encryption IP core interfaced with LEON3 processor using GPIO's. Time, area and power reports are shown below in "Table 1",

Table 1. Synthesis report

AES Algorithm		**Total CPU clock cycles required for encryption**	**Area (Logic elements)**	**Power Dissipat-ion**
Har-dware	**Soft-ware**			
AES as custom hardware	Control part	1685	14,149/33,216 (43 %)	420.75 mW

7. CONCLUSION

Step-by-step implementation of AES encryption using LEON3 processor is given in this paper. AES is connected with the system designed around LEON3 processor as custom hardware block and the speed and area required is calculated. The interface between hardware and software is done through GPIO's. This paper will prove to be valuable to researchers working in this area and save their valuable time.

8. REFERENCES

[1] Specification and Modeling of HW/SW CO-Design for Heterogeneous Embedded Systems Adnan Shaout, Ali H. El-Mousa., and Khalid Mattar, Proceedings of the World Congress on Engineering 2009 Vol I, WCE 2009, July 1 - 3, 2009, London, U.K.

[2] Giovanni De Micheli, fellow, IEEE, and Rajesh K. Gupta, member, IEEE, "Hardware/Software Co-Design", proceedings of the IEEE, Vol. 85, No. 3, March 1997

[3] Declan Staunton, "Successful use of an open source processor in a commercial ASIC", D&R Industry articles.

[4] Gaisler Research, "GRLIB IP Library User's Manual", Version 1.1.0 B4113 January 2012.

[5] Gaisler Research, "GRLIB IP Core User's Manual" ,Version 1.1.0 - B4113, January 2012.

[6] FIPS PUB 197, Advanced Encryption Standard (AES), National Institute of Standards and Technology, U.S. Department of Commerce,November2001(http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf).

[7] J. Daemen and V. Rijmen, "The design of AES-The Advance Encryption Standard" Springer-Verlag, 2002.

[8] X. Zhang and K. K. Parhi, "High Speed VLSI Architectures for the AES Algorithm," *IEEE* transactions on VLSI system , vol. 12, no. 9, September 2004.

[9] Jiri Gaisler, "BCC - Bare-C Cross-Compiler User's Manual", Version 1.0.36, April 2011.

[10] Gaisler Research, "TSIM2 Simulator User's Manual", Version 2.0.18, October 2010.

[11] Gaisler Research, *"GRMON User's Manual",* Version 1.1.47, November 2010.

[12] Altera Corporation, "Cyclone II FPGA Starter Development Kit User Guide", version 1.0.0 October 2006.