A Proof of Concept on Defending Cold Boot Attack

Joo Guan Ooi[1], Kok Horng Kam[2]
[1]Intel Microelectronics Sdn Bhd, Malaysia, [2] Intel Microelectronics Sdn Bhd, Malaysia
joo.guan.ooi@intel.com

Abstract

DRAM is an essential memory of a modern computer. Microprocessor loads the data which the user requested into DRAM before processing the data. Hence, DRAM contains important information in a computer. Recently, security researchers disclosed that DRAM is vulnerable to attack. Through Cold Boot Attack, DRAM contents can be recovered even after the computer has been powered off for several minutes [1]. The information obtained can be used to circumvent popular disk encryption system such as FileVault and Bit Locker. In this paper, we proposed an enhanced memory architecture which adds a data scrambling / descrambling layer between the microprocessor and DRAM controller to prevent the original data to be stored as cleartext in the DRAM. The original data will be scrambled before writing to DRAM and hence preventing the Cold Boot Attack. This new layer consists of XOR circuit, Galois Field Multiplication of order 128 (GF128) and a Pseudo Random Number Generator (PRNG). The scrambling scheme was selected in this proposal due to its simplicity for proof of concept. Any other cryptography scheme can replace the scrambling / descrambling blocks according to the required level of data protection. The designed blocks were implemented and tested on the Altera DE2 FPGA board using Nios II system. The results confirm that the use of the scrambling / descrambling blocks provides an easy solution with additional level of protection to secure the contents in the DRAM.

1. Introduction

DRAM is used by most of the computer nowadays and many people including experts think that the contents in the DRAM will be lost almost immediately after power is being cut off. In reality, the contents in the DRAM will still persist for a finite moment after the power was removed. If the temperature of DRAM is kept low, the data will persist for minutes or even for hours [1]. This phenomenon will give a great opportunity to attacker with physical access to obtain the full memory image of targeted system. Some of the popular disk encryption systems including the popular BitLocker used by Vista, True Crypt and also FileVault used by Mac OS store their "key" information in DRAM after the user login [1]. The attacker is able to use cold reboots to harvest the memory contents and circumvent popular disk encryption systems using no special device or material. By having the full memory image, the attacker can extract the "key" from the DRAM through simple pattern matching method in a matter of seconds. Once the attacker got the "key", he/she gains full access to the protected contents or system [1].

The slowly discharging capacitor in DRAM causes the computer to be vulnerable to Cold Boot Attacks. In order to solve the problems, we proposed to add scrambling / descrambling blocks between the microprocessor and DRAM controller. By doing this, the information in the DRAM is scrambled. And, it is no longer a trivial job, as before, for the attacker to recover the secure key(s) or password(s) although he/she has full access to the memory image of the targeted system. Therefore, it prevents the Cold Boot Attack from extracting the important information such as the key of encryption software from DRAM.

This proposal applies bit scrambling technique using XOR circuit and GF128. The GF128 block will take the address to the DRAM and a session key as the inputs to perform arithmetic operation. The GF128's output will XOR with the original data to form the scrambled data when writing to the DRAM. For descrambling process, the output from the GF128 will be XOR'ed together with the scrambled data to reconstruct the original data.

A session key is the snapshot value of the entropy source whenever the system come out of reset, i.e. reset signal is de-asserted. In order to minimize the predictability of the session key, a PRNG is used as the entropy source instead of a simple monotonic counter. The session key is software invisible and kept in hardware register until the next reset. Therefore, the contents in the DRAM are rendered useless at the next system boot up unless the previous session key is known.

This proposal was implemented in Altera DE2 FPGA board using Verilog. We connected the designed scrambling / descrambling blocks to the Altera Nios II system on Avalon Bus. The functionality of the design was tested using a C program.

The implementation gives an idea to protect a computer system from Cold Boot Attack. The original data is prevented from exposure in the DRAM through scrambling process. The session key is changed each time the system reset or reboot. This proposal prevents the Cold Boot Attacker to recover the contents in the system memory.

2. Previous Work

Cold Boot Attack has become a security issue that many people concerning about since last year [1]. Some of the researchers have proposed some possible solutions to prevent the Cold Boot Attack.

Frozen Cache blog presents a general-purpose solution to solve the Cold Boot Attack issues on Full Encryption software. The blog is proposing the use of the CPU cache for thwarting Cold Boot Attacks (at least on most X86 systems) [4]. Frozen Cache's proposal aims to store the encryption key in CPU Cache instead of the vulnerable RAM. The CPU cache is switched into a special mode that one can force that

data remains in the cache and is not written back to RAM. This technique calls Cache-as-RAM in LinuxBIOS/CoreBoot. The "Frozen Cache" concept provides an easy and convenient solution to protect the encryption key in the RAM. The major issue on "Frozen Cache" concept is the performance. Thus, the author suggests that you only do this when the screen is locked. However, there may be some programs that run in the background when the screen is locked. Activate the special cache mode may slow down the performance of the computer. The protection from the proposed solution is only limited to certain encryption key and other sensitive data in the RAM remains vulnerable to be attacked.

There are some researchers that focus on creating a secure execution environment. The examples of the implementation included the "SP Architecture" and "IBM4758 Secure Coprocessor".

SP (Secret Protected) Architecture is an architecture proposed to protect a user's critical secrets [2]. The architecture implements a Concealed Execution mode to launch the protection to a sensitive data such as secret key. In the Concealed Execution mode, a Trusted Software Module is used to protect the computations on user's secret key, the computation of secret keys and all the intermediate state are protected from observation and tempering by adversaries. Secure registers and cryptographic engines are added to the general purpose processor to support concealed execution mode.

IBM4758 Secure Coprocessor is a temper-responding device that provides a secure software and hardware execution environment [3]. The architecture subdivides the trust level into three levels. The hardware has higher trust level compared to the firmware, while the firmware has higher trust level compared to the software. The IBM 4758 Secure Coprocessor uses a faster hardware temper response instead of slower software temper response. The architecture implies a well-sealed package and careful design of the communication paths between the 4758 and the outside world.

The SP architecture and IBM4758 Coprocessor provide secure execution environment and ensure the secret key did not expose to the outside world without proper authentication. However, SP architecture and IBM4758 Coprocessor need some changes on the current system and the architectures. This increases the manufacture cost of the current product. Generally, the hardware level protection is safer than a software-only solution which is easier to be attacked by the attacker. All of the solutions listed above are not able to help to solve the cold boot attack problems fully. Although the encryption keys are hidden carefully, the other important contents in the DRAM still vulnerable to attack by using Cold Boot Attack.

3. Methodology

The main objective of this project is to implement the proof of concept on defending the Cold Boot Attack using an enhanced memory system. In order to achieve the goal, we need to design the enhanced memory system, implement and test it in the hardware. The enhanced memory system is designed using Verilog and simulated using Altera Quartus II Software. Besides that, we also created a complete computer system by using Altera SOPC Builder. The designed blocks will be inserted into the created Nios II computer system. After that, the whole Nios II system will be programmed into the Altera DE2 Development board.

Generally, the project development process can be divided into five phases. They are:

i. Planning ,
ii. Design,
iii. Software Simulation,
iv. Hardware Implementation,
v. Testing and Debugging.

3.1. Planning

Planning is the first stage and one of the most important stages in the project. In this stage, we planned and determined which are the best algorithm and method to implement the design. In this project, we will use Galois Field Multiplication as our randomizer for scrambling / descrambling operations. Basically, there are five main logic blocks to be designed and implemented in this project. They are the GF128 block, scrambling block, descrambling block, a PRNG and an Avalon Bus compliance transparent bridge. We designed the blocks and validated them separately before we put them into the Nios II system.

3.2. Design

The overall design of the enhanced memory architecture is shown in Figure 3.1. The key components of the design are the scrambling and descrambling blocks. The first logic which was designed and developed was the GF128. The GF128 block is the core of the scrambling and descrambling blocks which provides randomization for the scrambling and descrambling operations. The scrambling and descrambling operations are just simple XOR operation between the data and the GF128 output. Then, a PRNG was implemented as the entropy source of this system. The GF128 block takes in the snapshot of the PRNG output as the session key for current boot cycle. Next, in order to connect all the designed blocks into the Nios II system, we build a transparent bridge as the interface connecting the memory and the Avalon Fabric. Once all the building blocks were developed and tested, the PRNG, Scrambling and Descrambling Logics are integrated into the transparent bridge to make up the enhanced memory system. The following paragraphs will elaborate on each and every building block.

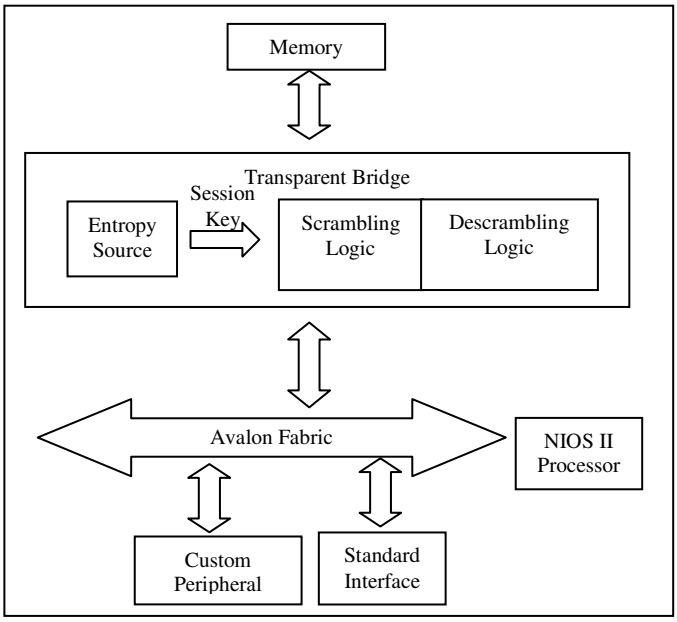

Figure 3.1: The Structure of Enhanced Memory Architecture

3.2.1. Galois Field Multiplication Block

In this project, we implemented Galois Field Multiplication of order 128. The GF128 block is the mathematical algorithm that we used to provide randomization for the scrambling and descrambling operations in this project. There are three inputs to the GF128 block; they are the address to the DRAM, the session key and system clock.

The GF128 will provide randomized output based on the address to the DRAM and the session key. The output from the Galois Field Multiplication is totally different with the inputs.

3.2.2. Pseudo Random Number Generator

In this project, we implemented a 32-bit Galois Linear Feedback Shift Register (LFSR) as the PRNG. The polynomial implemented in this project is $x^{32} + x^{31} + x^{29} + x + 1$. LFSR was chosen because it is simple to implement and easily expandable to higher order to provide better entropy to the system. A session key is created whenever the system reset signal is de-asserted. This session key is stored in hardware register and valid for the current boot cycle until the next reset event. Software is not accessible to this key in current implementation.

3.2.3. Scrambling Block

The structure of the scrambling block is shown in Figure 3.2. The session key and the address to the DRAM will be used by GF128 to provide its randomized output. The output of GF128 is connected to XOR circuit through a multiplexer. The multiplexer allows scrambling to be enabled or disabled by a switch. When scrambling is enabled, GF128's output will appear at the output of the multiplexer, otherwise the multiplexer's output will be zeros. The Data to be written to the DRAM will be XOR'ed with the output of the multiplexer. The result of the XOR circuit will then be written to the DRAM. If the encryption is disabled, the output from the XOR will be the same as the Data.

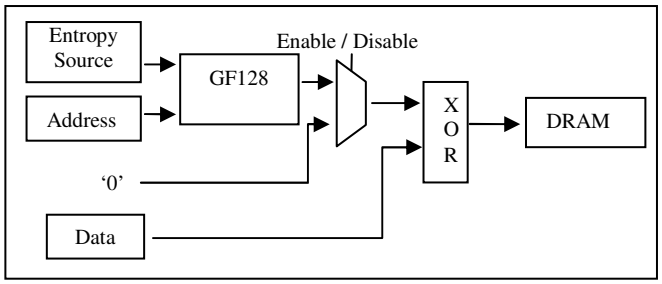

Figure 3.2: The Design of the Scrambling Block

3.2.4. Descrambling Block

The structure of the descrambling block is shown at the Figure 3.3. From Figure 3.3, we can notice that the structure of the descrambling block is similar to the scrambling block. The position of the DRAM and the Data is swapped. Contrary to the memory write cycle as performed in the scrambling block, the descrambling block performs a memory read from DRAM. Therefore, the DRAM will provide the input data to the XOR circuit for a given address. The data from the DRAM will be XOR'ed with the output from GF128 through the multiplexer to recover the original Data.

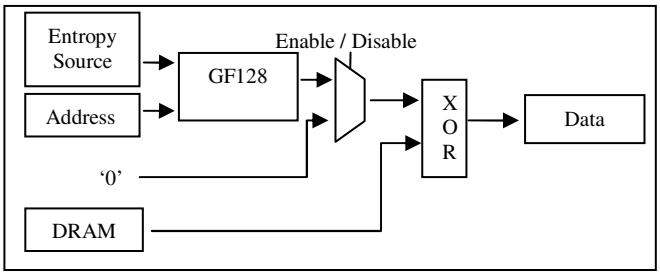

Figure 3.3: The Design of the Descrambling Block

3.2.5. The Transparent Bridge

All our design blocks are integrated into the transparent bridge which acts as the interface to Avalon Bus and DRAM controller. The interface of the Transparent Bridge is shown at the Figure 3.4.

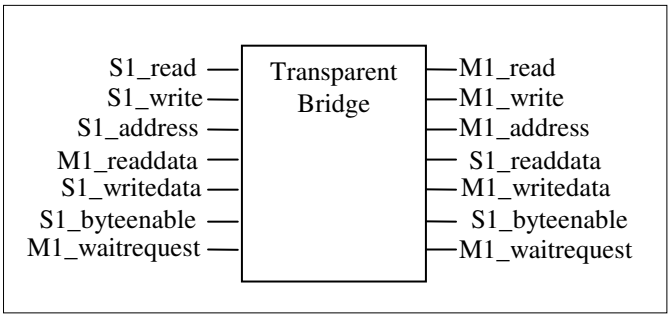

Figure 3.4: The Interface of Transparent Bridge

There are some signals that are required to establish the connection between the Avalon Bus and DRAM controller. The description of the signals required to connect to the Avalon Bus and DRAM controller are listed at Table 3.1 and Table 3.2, respectively [11].

Table 3.1: Signals required connecting to the Avalon Bus

Signals	Description
S1_read	Asserted to indicate a read transfer.
S1_write	Asserted to indicate a write transfer.
S1_address	The address of the SDRAM
M1_readdata	The readdata provided by the slave in response to a read transfer.
S1_writedata	Data from the system interconnect fabric for write transfers.
S1_byteenable	Enables specific byte lane(s) during transfers.
M1_watirequest	Asserted by the slave when it is unable to respond to a read or write request.

Table 3.2: Signals required connecting to the SDRAM Controller

Signals	Description
M1_read	Read request signals from master.
M1_write	Write request signals from master.
M1_address	The address of the SDRAM.
S1_readdata	Data signal for read transfers.
M1_writedata	Data signal from the master for write transfers.
M1_byteenable	Enables specific byte lane(s) during transfers.
S1_watirequest	Asserted by the slave when it is unable to respond to a read or write request.

3.3 Software Simulation

Software simulation is an essential part in the design process. The simulation can be done by using Altera Quartus II Design software. The result of simulation on the current design will be examined in this stage. The design will be changed if the simulation results could not meet the functional or timing requirements of the project.

3.4 Hardware Implementation

After the software simulation, we are going to download the design to Altera DE2 FPGA. Pin assignments and arrangements are required before we can download the design to Altera DE2 FPGA board.

3.5 Testing and Debugging

Testing and debugging are a very important process in the project. Most of the development time will be used to do testing and debugging on the software and also the hardware. If the system failed to meet the requirements, we will need to re-design and re-test again.

In our project, a program was written in C language for the purpose of testing. The program has the capabilities to check the scrambling / descrambling status, read 4 bytes from DRAM, write 4 bytes to DRAM, read the contents of the DRAM as string and write a character string to the DRAM. With the help of the program, we can perform the validation and testing stage effectively.

4. Results and Discussions

In this section, we will discuss on the results of the enhanced memory architecture from our proposal. We included the output from read and write to DRAM, the output from read and write string to DRAM and the performance comparison among the Nios II system with scrambling enabled, scrambling disable.

4.1. Read and Write to DRAM

The read and write to DRAM functionality can be tested by using the written C program. Firstly, we will write some random integers to the DRAM at specified addresses with scrambling enabled. In this experiment, we wrote “1” into the address of 0x000000, “2” to the address 0x000004, “3” to the address 0x000008, “4” to the address 0x00000c, and “5” to the address 0x000010. The values we wrote to DRAM are shown clearly in Figure 4.1.

```
Write to SDRAM
************************************************************
Please enter the SDRAM offset (0x000000 - 0x7fffff)     > 0

Please enter the value to write or 'q' to quit
WRITE_SDRAM   0x000000 > 1
WRITE_SDRAM   0x000004 > 2
WRITE_SDRAM   0x000008 > 3
WRITE_SDRAM   0x00000c > 4
WRITE_SDRAM   0x000010 > 5
WRITE_SDRAM   0x000014 > q
```

Figure 4.1: Write to DRAM with Scrambling Enabled

After the write process, we tried to read back the data from the DRAM with scrambling enabled. The read DRAM results are shown in Figure 4.2. From Figure 4.2, we can notice that the values are match with the values we wrote to DRAM previously. This indicates that the logic works correctly and the scrambling and descrambling operations are performed without any problems.

We repeat the read process but disable the scrambling function this time. This is a negative testing to show that the original data is not recoverable as well as to examine the actual data in the DRAM. Figure 4.3 shows the actual data in the DRAM. The data in the DRAM are scrambled successfully and leaving no visible trace of any correlation with the original data.

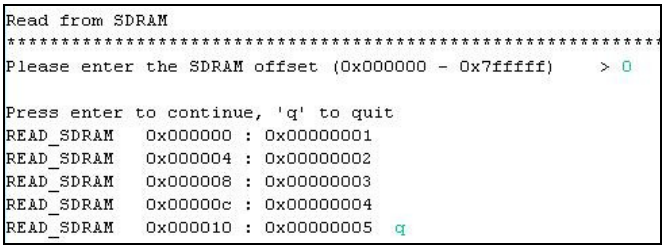

Figure 4.2: Read from SDRAM with Scrambling Enabled

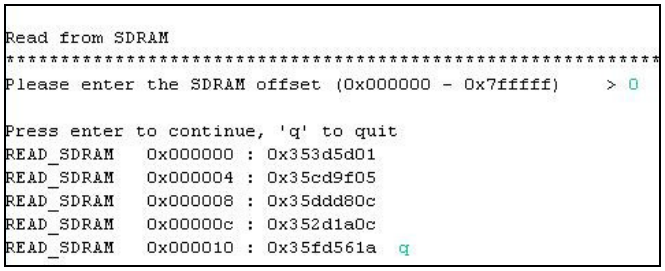

Figure 4.3: Read from SDRAM with Scrambling disabled

4.2. Read and Write string to DRAM

The experiment in section 4.1 is repeated here but this time we read and write strings to the DRAM instead of integers. We tried to write an ASCII picture of a monster with a size of 1978 bytes into the DRAM. The write string processes are recorded in Figure 4.4.

Figure 4.4: Write String to DRAM with Scrambling Enabled

After that, we read back the contents from the DRAM and the result is shown in Figure 4.5. We could successfully recover the same ASCII picture from the DRAM.

Figure 4.5: Read String from DRAM with Scrambling Enabled

Next, we disabled the scrambling/descrambling function and try to read again from the DRAM. The contents in DRAM are shown in Figure 4.6. This time, the monster picture is gone! This implies that the data in the DRAM are scrambled successfully.

Figure 4.6: Read String from DRAM with Scrambling Disabled

4.4. Performance Comparison

In this section, the performance of the enhanced memory system when scrambling / descrambling enabled and scrambling / descrambling disabled are compared. The range of the data sizes used for the write and read operations is from 10k bytes to 320k bytes. The read and write speed of the enhanced memory system are shown in Figure 4.7 and Figure 4.8 respectively.

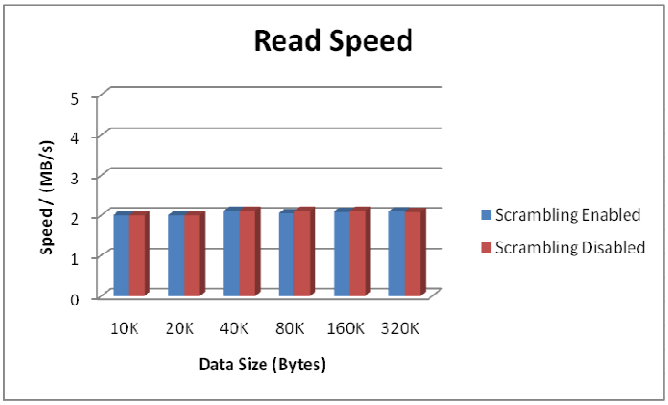

Figure 4.7: Read Speed of the Enhanced Memory System

From Figure 4.7, the read speed of the system when scrambling / descrambling enabled and disabled are approximately the same. The throughput of the system for read operation is around 2MB/s and increase slightly when the data size increase.

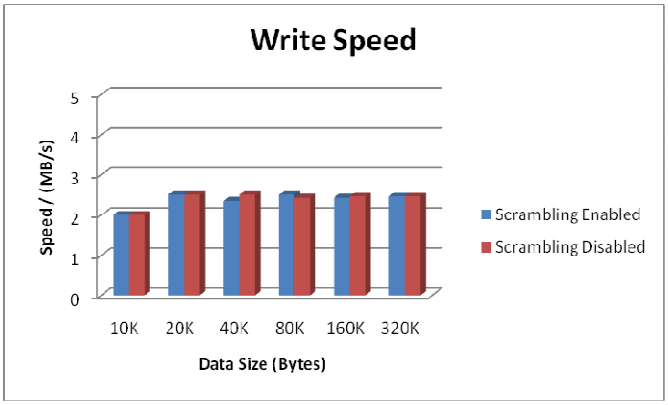

Figure 4.8: Write Speed of the Enhanced Memory System

The write performance of the system is almost the same when the scrambling and descrambling blocks are enabled or disabled as shown in Figure 4.8. The throughput of the system for write operation is around 2MB/s when data size is 10k bytes and increase to around 2.5MB/s when data size increase.

Theoretically, the scrambling and descrambling blocks will consume additional time compared to the normal system. The delay may affect the performance of a system. However, in our experiments, the scrambling and descrambling blocks doesn't have significant effect on the performance of Altera DE2 Board which is clocked at 50MHz. One of the possible reasons is the latency for scrambling and descrambling operations are lower than the DRAM access time for our experiment setup. Therefore, the additional delay from scrambling and descrambling operations are insignificant to the overall system performance.

5. Advantages and Disadvantages

Our proposal prevents the data to be stored in the DRAM as cleartext. This proposal does not only protect the secure key but it also protects other sensitive data in the DRAM. The session key is always updated once the system reset. The data in DRAM will be unrecoverable in the following system boot up without the correct session key. Hence, it can prevent the Cold Boot Attacker to get any information from the DRAM even though a perfect memory dump could be obtained.

The drawbacks of the proposal are reduced performance of the memory system and additional cost for implementing the scrambling and descrambling logics.

6. Conclusion

As a conclusion, our proposal provides a good idea on defending against Cold Boot Attack. The design is a hardware implementation which provides the hardware level protection that is safer than a software-only solution. The contents of the DRAM are scrambled and it prevents any attacker to recover sensitive information in the DRAM. Besides that, it also solved the current full disk encryption problems that store the key in the memory without requiring any change on the full disk encryption software.

7. References

[1] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W.,Felten. "Lest We Remember: Cold Boot Attack on Encryption Keys", Princeton University, April 2008.

[2] R. Lee, P. Kwan, J.P. McGregor, J. Dwoskin, Z. Wang, "Architecture for Protecting Critical Secrets in Microprocessors", Proceedings of the 32nd International Symposium on Computer Architecture (ISCA 2005), pp. 2-13, June 2005.

[3] J. G. Dyer, M. Lindemann, R. Perez, R. Sailer, L. van Doorn, S. W. Smith, and S. Weingart, "Building the IBM 4758 secure coprocessor", IEEE Computer, v.34 n.10 p.57–66, 2001.

[4] ACME Security, "Frozen Cache", http://frozencache.blogspot.com/, Jan 2009.

[5] Altera Corporation, Nios II Hardware Development Tutorial, 2008.

[6] Altera Corporation, Quartus II Version 8.0 Handbook, 2008.

[7] Altera Corporation, Introduction to the Altera Nios® II Soft Processor, 2008.

[8] Altera Corporation, Introduction to the SOPC Builder, 2008.

[9] Altera Corporation, Using the SDRAM Memory on Altera's DE2 Board, 2008.

[10] Altera Corporation, Using C with Altera DE2 Board, 2008.

[11] Altera Corporation, Avalon Interface Specification, 2008.