CYENG 225 – SP 23: Microcontroller Essentials for Cyber Applications
Instructor: Dr. Shayan (Sean) Taheri
Gannon University (GU)

Assignment 6: Chapter 6 – Memory Protections

Total Points: 100**; and Deadline:** May/05/2023, 11:59 PM.

Note – Cheating and Plagiarism: Cheating and plagiarism are not permitted in any form and they cause certain penalties. The instructor reserves the right to fail culprits.

Deliverable: All of your responses to the questions of assignment should be included in a single compressed file to be uploaded to the Gannon University (GU) – Blackboard Learn environment.

Question 1. Provide short answers (i.e., no more than five lines on average with the font size of 12) for the following items. The grade for each item is **10 points**.

1. Specify different **types of attacks** that cause **memory vulnerability**. Mention major techniques that can provide **protection for memories**.
2. Discuss how the **Standard Merkle Tree** and the **Bonsai Merkel Tree** provide memory protection. Determine their similarities and differences.
3. Explain how **encryption** can provide **confidentiality protection** for memories. Illustrate using a figure how communications between processor and memory can be **efficiently** encrypted and decrypted.

Question 2. Identify the security requirements of the memory hierarchy, especially for cache memory and main memory (i.e., random-access memory or RAM); assess acceptability of secure hash algorithm (SHA) as a protective technology in satisfying the identified needs and being a solution for related security problems; Determine the technical specifications according to your analysis and consider them in your hardware implementation of cache memory, RAM, and SHA modules; and conduct experiments to evaluate and test their operations, and analyze their functionalities and results:

Cache Memory owes its introduction to Wilkes back in 1965. At that time, Wilkes distinguished between two types of main memory: the conventional and the slave memory. In Wilkes terminology, a slave memory is a second level of unconventional high-speed memory, which nowadays corresponds to what is called a cache memory (i.e., the term cache means a safe place for hiding or storing things). The idea behind using a cache as the first level of the memory hierarchy is to keep the information expected to be used more frequently by the central processing unit (CPU) in the cache (i.e., a small high-speed memory that is very near the CPU). The end result is that at any given time certain active portion of the main memory is duplicated in the cache. **Main Memory** is a physical device that holds the data and the instructions that the CPU needs. There are two types of main memory that includes read-only memory (ROM) and RAM. It is located close to the CPU on the computer motherboard, enabling the CPU to read data from main memory very quickly. It has a variable content, also it is generally used to store the variable data. RAM can also be used to store frequently changed programs and other information. The term “random” means that any memory location can be accessed in the same amount of time, regardless of its position in the memory. RAM allows the computer to store information quickly for later references. **Hash Algorithms** are used as components by other cryptographic algorithms and processes to provide information security services. Hash functions are often utilized with digital signature algorithms, keyed-hash message authentication codes, key derivation functions, and random number generators. A hash algorithm converts a variable length message into a condensed representation of the

electronic data in the message. **Secure Hash Algorithm** is the most popular standard of Hash functions. Several security protocols use SHA to provide message integrity, authentication and digital signature. Every cryptographic hash function is a hash function. However, not every hash function is a cryptographic hash. A cryptographic hash function aims to guarantee a number of security properties. Most importantly that it's hard to find collisions or pre-images for SHA and that its output appears random. Non-cryptographic hash functions just try to avoid collisions for non-malicious input. Complete the following Steps. The grade for this question is **70 points**.

A. Study the articles under the following section of "**Resources for Step A**" to fully understand the functionalities of the Cache Memory, the RAM, and the SHA modules.
B. Implement the Cache Memory, the RAM, and the SHA modules using **Verilog hardware description language (HDL)**. Sample implementations of them in VHDL are provided in the following section of "**Resources for Step B**" for your kind reference, guidance, and understanding of the implementation processes. **Hint**: You can find a working code in VHDL and translate it to Verilog HDL.
C. Perform the procedure from the "**EXPERIMENT #1: Introduction to Xilinx's FPGA Vivado HLx Software**" laboratory assignment for your implementations.
D. Include the following items in your submitting package:
   - Provision of the achieved results from your implementations in the **Steps B and C**.
   - All files of your implementations in the **Steps B and C**.
   - Provide a report that includes: (**1**) your overall understanding, identifications, high-level assessments, experimental evaluations, analyses, and conclusions from completing the experiments; (**2**) the interesting points and the challenges that you faced in this laboratory; and (**3**) the screenshots for all of the major steps in your experiments.

Resources for Step A – Cache Memory:

1. Cache (computing) - Wikipedia
2. CPU cache - Wikipedia
3. What Is Cache Memory in My Computer | HP® Tech Takes
4. Principles of Cache Design - Technical Articles (allaboutcircuits.com)
5. Cache Memory in Computer Organization - GeeksforGeeks
6. Computer Architecture: Cache Cheatsheet | Codecademy
7. Cache Memory (Computer Organization) - javatpoint
8. Basics of Cache Memory – Computer Architecture (umd.edu)
9. Cache Associativity - Algorithmica

Resources for Step A – RAM:

1. Random-access memory - Wikipedia
2. Memory Hierarchy Design – Basics – Computer Architecture (umd.edu)
3. 901320_Main Memory.ppt (live.com)
4. Watson (latech.edu)
5. Random Access Memory (RAM) and Read Only Memory (ROM) - GeeksforGeeks
6. Different Types of RAM (Random Access Memory) - GeeksforGeeks
7. What is RAM (Random Access Memory)? - Definition from SearchStorage (techtarget.com)
8. Designing of RAM in VHDL using ModelSim (circuitdigest.com)

Resources for Step A – SHA:

1. Secure Hash Algorithms - Wikipedia
2. SHA-1 - Wikipedia
3. SHA-2 - Wikipedia
4. SHA-3 - Wikipedia
5. What is SHA? What is SHA used for? | Encryption Consulting
6. Secure Hash Algorithms | Brilliant Math & Science Wiki
7. What Is SHA-256 Algorithm: How it Works and Applications [2022 Edition] | Simplilearn
8. Back to Basics: Secure Hash Algorithms | Analog Devices
9. What Is the Most Secure Hashing Algorithm? (codesigningstore.com)
10. SHA-1 Hash - GeeksforGeeks
11. SHA-256 Cryptographic Hash Algorithm implemented in JavaScript | Movable Type Scripts (movable-type.co.uk)
12. https://xilinx.github.io/Vitis_Libraries/security/2019.2/guide_L1/internals/sha2.html

Resources for Step B – Cache Memory:

1. aminrashidbeigi/SAYEH-Cache: implementing SAYEH cache using VHDL (github.com)
2. 4_way_set-associative_cache/src_m at master · avina5hkr/4_way_set-associative_cache (github.com)
3. VHDL/Cache.vhd at master · Harshita-rs/VHDL (github.com)
4. josh-jacobson/cache: VHDL implementation of a basic memory hierarchy with L1 and L2 caches. (github.com)
5. VHDL-MIPS-processor/CACHE.vhd at master · Daniel-BG/VHDL-MIPS-processor (github.com)
6. MIPS/cache.vhd at master · ocervell/MIPS (github.com)
7. godraadam/2waycache: 2-way set associative cache implemented in VHDL (github.com)
8. Cache-Simulation-in-VHDL/cache.vhd at master · Ayush9719/Cache-Simulation-in-VHDL (github.com)
9. VHDLMipsCache/CacheL1wb.vhd at master · marcelomlinck/VHDLMipsCache (github.com)
10. Cache/cache.vhd at master · Tabrizian/Cache (github.com)
11. sidjos/Cache_Design_Project: Simple Cache Design Implementation in VHDL (github.com)
12. dqkt/32-byte-cache: VHDL implementation of two-way, set associative 32-byte cache simulation (github.com)
13. cache/Modules at master · mmsamiei/cache (github.com)
14. prtyspt/Cache_VHDL: Multi-level set-Associative cache with replacement - implemented in VHDL (github.com)
15. Direct-mapped-Associative-mapped-cache-controller/Four-way Set Associative mapped/VHD Files/Top at main · 2331sakshi/Direct-mapped-Associative-mapped-cache-controller (github.com)
16. mips-complex/cache.vhd at master · g-gaston/mips-complex (github.com)
17. marcelomlinck/VHDLMipsCache: This repository contains the vhdl files describing a mips processor with two levels of cache implementing both write-through and write-back. (github.com)
18. AniketBadhan/Cache: VHDL code for 32-bit Cache (github.com)
19. 8_block_cache/cache_cell.vhd at main · bcain150/8_block_cache (github.com)
20. sandino/cache.vhd at master · vasigavr1/sandino (github.com)

Resources for Step B – RAM:

1. JuniorTrojilio/RamMemory_32x4: A sample implementation of Ram Circuit with bus 32 x 4 in VHDL. (github.com)
2. Kanishk-K-U/32x8-RAM: Implementation of 32x8 RAM in VHDL (github.com)
3. JuniorTrojilio/RamMemory: A sample memory created with VHDL (github.com)
4. vlsicad/fifo_single_port_ram_with_all_control_logic: this is fifo with ram and all required control logic like fifo full, empty, ready etc. code is in vhdl. hope you find this is useful, can be useful in cdc design. (github.com)
5. SysAlloc-VHDL/ram.vhd at master · Hilx/SysAlloc-VHDL (github.com)
6. VHDL_processor_incomplete/RAM.vhd at master · kbickham/VHDL_processor_incomplete (github.com)
7. Angiaraji/Implementation-of-generic-FIFO-using-RAM-in-VHDL (github.com)
8. vm/ram32x4.vhd at master · Razzaz/vm (github.com)
9. 16-bit-Computer-Architecture-Simulation-in-VHDL/256x16RAM.vhd at master · polatbilek/16-bit-Computer-Architecture-Simulation-in-VHDL (github.com)
10. Basic_VHDL_Utilities/ram.vhd at master · QDucasse/Basic_VHDL_Utilities (github.com)
11. simon-game-vhdl/ram_datos.vhd at master · santifs/simon-game-vhdl (github.com)
12. vhdl-collection/ram.vhdl at master · dominiksalvet/vhdl-collection (github.com)
13. Cache/RAM.vhd at master · arminkz/Cache (github.com)
14. cache/ram.vhd at master · mmsamiei/cache (github.com)
15. VHDL_ram/RAM at master · matticsphi/VHDL_ram (github.com)
16. ram/ram.vhd at master · Sivaranjith1/ram (github.com)
17. DeimosHall/memRAM: VHDL program (github.com)

Resources for Step B – SHA:

1. ikwzm/SECURE_HASH: SHA-1,SHA-256,SHA-512 Secure Hash Generator written in VHDL(RTL) for FPGA(Xilinx and Altera). (github.com)
2. lostpfg/SHA-256-HDL: An implementation of original SHA-256 hash function in (RTL) VHDL (github.com)
3. sha256/src at master · skordal/sha256 (github.com)
4. martinafogliato/Sha256_Hw_Accelerator: SHA256 hardware accelerator, synthesized for and mapped on the Zynq core of the Zybo board by Digilent (github.com)
5. dsaves/SHA-256: An SHA-256 module implementation in VHDL. Based on NIST FIPS 180-4. (github.com)
6. dsaves/SHA-512: SHA-512 hardware implementation in VHDL. Based on NIST FIPS 180-4. (github.com)
7. adspirop/SHA-1-VHDL: Implementation of SHA-1 core in VHDL for FPGA Using Pipeline (github.com)
8. fbv81bp/VHDL_SHA2-384: SHA-2 versions working on 64 bits of data: 384 and 512. (github.com)
9. FPGA-SHA1/sha1.vhd at master · Unallocated/FPGA-SHA1 (github.com)
10. Destfolk/SHA-256: SHA-256 Hash Function Algorithm (github.com)
11. tpm-cryptoprocessor/sha256.vhd at master · ryanmcclure4/tpm-cryptoprocessor (github.com)
12. SHA-256-HDL/sha256-core.vhd at master · lostpfg/SHA-256-HDL (github.com)
13. fbv81bp/VHDL_SHA2-256: VHDL Secure Hash Algorithm 2 cores (github.com)
14. Processor_AES_SHA/SHA_256.vhd at main · jukepper99/Processor_AES_SHA (github.com)

15. DuyTran62529/SHA256-Hardware-Implementation: VHDL RTL design of SHA256 Hash function (github.com)
16. sha2/vhdl at master · sbates130272/sha2 (github.com)
17. SECURE_HASH/src/main/vhdl at master · ikwzm/SECURE_HASH (github.com)
18. SHA-256/SHA_256.vhd at main · Destfolk/SHA-256 (github.com)
19. Xoodyak/SHA-256 VHDL at main · adeirman46/Xoodyak (github.com)