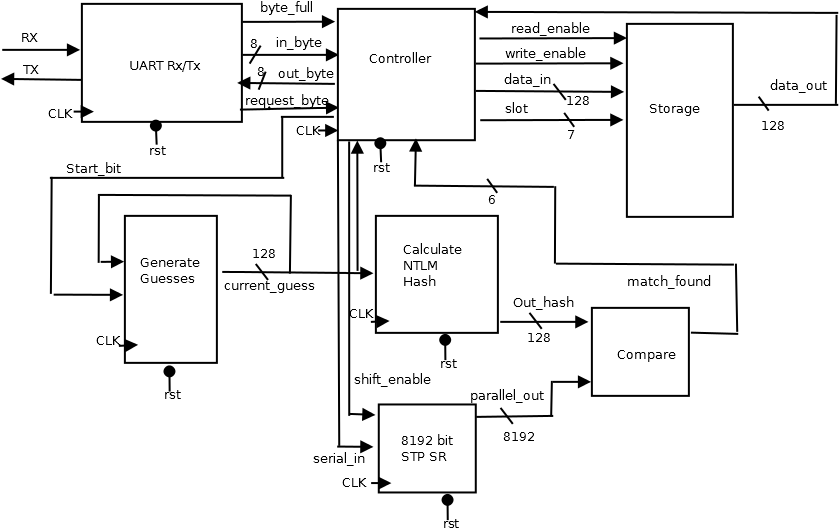
NTLM Brute-Force Hash Cracker Verification Plan

Team Members: Clayton Brandt, Darius Elliott, Adit Gosh, Tom Yurek

**Architecture**



**Calculate NTLM Hash**:

This block takes an input of an ASCII string given as a sequence of binary values which represents the current guess of the “Generate Guesses” block. It then implements the MD4 message-digest algorithm to create a 16-byte output hash. The output hash is then compared to the stored hashes that were initially inputted to the system.

**Generate Guesses:**

This sequential block will generate potential passwords that could be recreated using a hash algorithm to compare with our inputted hashes. The block takes the previous guess (an ASCII string) and outputs the next guess after a clock cycle.

**Compare:**

This combinational block takes each input hash and compares it with the output of the NTLM block. This serves to compare the current password guess to any of the hashes that have been inputted to the system. If there is a match, it indicates which hash matches and outputs that hash’s number so that the system can store the current password with the correct corresponding hash.

**Storage:**

This combinational block will first take the input from the UART via the Controller and store each inputted hash. After the initial loading phase, this block can store and retrieve hashes along with any cracked values the system has found. The desired data is fed back to the Controller and the Compare block. The SRAM, this block will have 128 addresses, each containing 128 bits of information. This is to allow for the storage of 64 128-bit hashes and 64 128-bit cracked values.

**UART:**

The entry and exit point for data in our system. When outputting data, it takes the calculated passwords from the storage block. This block will also communicate to the Controller signaling when the device is being used for input or output.

**Controller:**

The controller will send out a start bit to the Generate Guesses block to start generating passwords. The Controller will also receive a match found signal from the compare block and it will decide whether or not to save the current guess for a password to the current hash being compared by sending this data to the Storage block. If the controller is being used for output, meaning it receives the request byte signal, then the Controller will begin reading data from Storage, and then send this data on the out byte line back to the UART, noting that the output is the saved passwords for the inputted hashes.

**Fixed Success Criteria**

1. Test benches exist for all top level components and the entire design. The test benches for the entire design can be demonstrated or documented to cover all of the functional requirements given in the design specific success criteria. (2 pts)
2. Entire design synthesizes completely, without any inferred latches, timing arcs, and sensitivity list warnings. (4 pts)
3. Source and mapped version of the complete design behave the same for all test cases. The mapped version simulates without timing errors except at time zero. (2 pts)
4. A complete IC layout is produced that passes all geometry and connectivity checks. (2 pts)
5. The entire design complies with targets for area, pin count, throughput (if applicable), and clock rate. The final targets for these parameters will be determined by course staff based on your design review. Failure to reach any of the targets will result a score of 1 out of 2 provided that you are within 50% on area, 10% on pin count, and 25% on throughput. Doing worse in any category will result in a score of 0 out of 2. (2 pts)
   1. Area: 34,986,239 um2
   2. Pin Count: 5
   3. Clock Period: 100 MHz

**Design specific criteria**:

1. Demonstrate by simulation of Verilog test benches that the complete design is able to

successfully match a generated password to that of an inputted hash. (2 pts)

2. Demonstrate by simulation of Verilog test benches that the complete design is able to

store multiple hashes inputted on a UART line. (2 pts)

3. Demonstrate by simulation of Verilog test benches that the complete design is able to

output multiple matched passwords. (1 pt)

4. Demonstrate by simulation of Verilog test benches that the complete design is able to

ignore invalid input and will not produce an output for the given input. (1 pt)

5. Demonstrate by simulation of Verilog test benches that the complete design is able to

generate a valid NTLM hash. (1 pt)

6. Demonstrate by simulation of Verilog test benches that the design functions correctly

for different inputted character sets. (1 pt)

**Verification Plan Summary**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **What to Verify** | **Design Module(s) involved** | **Verification Procedure Summary** | **DSSC(s) Proved** | **Use in Final Demo** | **Comments** |
| **Correctness of chip NTLM Hash Generation** | Top Level | Compare with alternate implementation. | DSSC 5 | Yes | use <http://www.tobtu.com/lmntlm.php> as a reference for correctness. |
| **Correctness of chip output** | Top Level | Compare results to an alternate implementation | DSSC 1, 3, 6 | Yes | use <http://www.tobtu.com/lmntlm.php> as a reference for correctness. |
| **Valid data stored within Shift Register** | Top Level | Compare the generated values to those stored in SRAM | DSSC 2 | Yes | Use temporary registers in test bench to capture Shift Register data |
| **Ignores Invalid Input** | Top Level | Compare results and see if the input did not produce an output | DSSC 4 | Yes | Use testbench to test whether the invalid input was simply ignored and did not output an invalid password |
| **Correctness of Generate Guess** | Generate Guess | Compare generated values to check for correctness | None | Only if can’t show using top level | Use testbench to test the validity of incrementing from one character to the next |
| **Correctness of Controller** | Controller | Check correctness of different functionalities | None | Only if can’t show using top level | Use temporary register in test bench to capture controller outputs |
| **Validate data stored in SRAM** | SRAM | Compare test values for SRAM with test values | None | Only if can’t show using top level | Use temporary registers to store data from SRAM and compare those values to the target values |
| **Correctness of chip NTLM Hash Generation** | Calculate NTLM Hash Block | Compare with alternate implementation | DSSC 5 | Only if can’t show using top level | use <http://www.tobtu.com/lmntlm.php> as a reference for correctness. |
| **Correctness of Compare** | Compare | Compare test inputs to outputs | None | Only if can’t show using top level | Use temporary registers in a testbench to compare the output of the compare block to the expected value |
| **Correctness of UART transmission** | UART | Send in test data and compare outputs to expected targets | None | Only if can’t show using top level | Use temporary registers in a testbench to compare the output of the UART to the expected values for the different functionality |

**Detailed Verification Test Breakouts**

**Demo Tests**

**Correctness of chip NTLM Hash Generation**

* Shown in Demo: Yes
* DSSC(s) Proved: 5
* Highest Level of Design Module(s) involved:
  + Total Design/Chip
* Test bench Expectations/Requirements:
  + Have temporary registers to store hash generation results.
* Use <http://www.tobtu.com/lmntlm.php> as a reference for NTLM Hash correctness
* No pre/post processing is needed
* Main Verification Test Steps:
  + Test each/random generated hash with the above website with NTLM hash implementation

**Correctness of Chip Output**

* Shown in Demo: Yes
* DSSC(s) Proved: 1, 3, 6
* Highest Level of Design Module(s) involved:
  + Total Design/Chip
* Test bench Expectations/Requirements:
  + Have temporary registers to store SRAM results.
* Use <http://www.tobtu.com/lmntlm.php> as a reference for NTLM Hash correctness
* No pre/post processing is needed
* Main Verification Test Steps:
  + Test each/random generated hash stored in SRAM to the inputs given using the above website with the NTLM hash implementation

**Valid Data Store in Shift Register**

* Shown in Demo: Yes
* DSSC(s) Proved: 2
* Highest Level of Design Module(s) involved:
  + Total Design/Chip
* Test bench Expectations/Requirements:
  + Have temporary registers to store data held in the shift register
* Use <http://www.tobtu.com/lmntlm.php> as a reference for NTLM Hash correctness
* No pre/post processing is needed
* Main Verification Test Steps:
  + Compare the data within the shift register to that inputted by the UART line.

**Ignores Invalid Input**

* Shown in Demo: Yes
* DSSC(s) Proved: 5
* Highest Level of Design Module(s) involved:
  + Total Design/Chip
* Test bench Expectations/Requirements:
  + Have temporary registers to store output results
* Use <http://www.tobtu.com/lmntlm.php> as a reference for NTLM Hash correctness
* No pre/post processing is needed
* Main Verification Test Steps:
  + Check chip output to that of the inputs and a given invalid hash will not generate a given valid password.

**Backup and Sub-Module Tests**

**Correctness of chip NTLM Hash Generation**

* Shown in Demo: Only if can’t show using top level
* DSSC(s) Proved: 5
* Highest Level of Design Module(s) involved:
  + NTLM Hash Generation Block
* Test bench Expectations/Requirements:
  + Random ascii character generator for raw data generation
* Use <http://www.tobtu.com/lmntlm.php> as a reference for NTLM Hash correctness
* No pre/post processing is needed
* Main Verification Test Steps:
  + Randomly generate an ascii value to generate a corresponding hash
  + Feed ascii value to generate hash block as if coming from Generate Guess block output line
  + Check hash value to the above reference for correctness
  + Repeat above steps for multiple data values.

**Correctness of Generate Guess**

* Shown in Demo: Only if can’t show using top level
* DSSC(s) Proved: -
* Highest Level of Design Module(s) involved:
  + Generate Guess
* Test bench Expectations/Requirements:
  + Have temporary registers to store results from one generated guess to the next.
* Use <http://www.tobtu.com/lmntlm.php> as a reference for NTLM Hash correctness
* No pre/post processing is needed
* Main Verification Test Steps:
  + Test each generated guess and check that the the block is incrementing correctly based on special characters (eg. zz -> aaa or zz-> zA etc)

**Correctness of Storage Block:**

* Shown in Demo: Only if can’t show using top level
* DSSC(s) Proved: -
* Highest Level of Design Module(s) involved:
  + Storage
* Test bench Expectations/Requirements:
  + Send in test signals as if coming from the Controller Block
* Use <http://www.tobtu.com/lmntlm.php> as a reference for NTLM Hash correctness
* No pre/post processing is needed
* Main Verification Test Steps:
  + Send in test data to be stored by SRAM.
  + Then, generate test signals to retrieve the stored data
  + Test the outputted/retrieved data for correctness against the initial inputted data
  + Repeat above steps for multiple values and test cases

**Correctness of Controller**

* Shown in Demo: Only if can’t show using top level
* DSSC(s) Proved: 5
* Highest Level of Design Module(s) involved:
  + Controller Block
* Test bench Expectations/Requirements:
  + Demonstrate successful loading of input hashes into shift register
  + Demonstrate successful storage of matched passwords to SRAM
  + Demonstrate successful loading of matched passwords to UART TX
  + Demonstrate successful propagation of user input character set
* No external or pre-made references are needed
* No pre/post processing is needed
* Main Verification Test Steps:
  + Simulate initial loading phase. Ensure relevant information would be propagated to the PTS Shift Register and Generate Guesses block.
  + Simulate the matching of a password and ensure that the correct ascii string would be stored in the correct address in SRAM.
  + Simulate the receiving of a user request for progress and ensure that the passwords are output correctly

**Correctness of Compare Block:**

* Shown in Demo: Only if can’t show using top level
* DSSC(s) Proved: 1
* Highest Level of Design Module(s) involved:
  + Compare
* Test bench Expectations/Requirements:
  + Have temporary register to store compared value
  + Send in data as if coming from Hash Generation and shift register
* No external or pre-made references are needed
* No pre/post processing is needed
* Main Verification Test Steps:
  + Check if the generated hash matches any of the hashes stored in the shift register.
  + Ensure that the correct match (or lack of match) is output to the controller

**Correctness of UART Transmission:**

* Shown in Demo: Only if can’t show using top level
* DSSC(s) Proved: 2, 3
* Highest Level of Design Module(s) involved:
  + UART Block
* Test bench Expectations/Requirements:
  + Send in test signals as if coming from the Controller Block and user
* No references
* No pre/post processing is needed
* Main Verification Test Steps:
  + Send in data from both ends, Controller and user
  + Check that outputted data corresponds to inputted data via registers