[[1]](#footnote-1)

Sequential String Detection, Hash, and Data Transmission in Verilog(May 2021)

Long H. Nguyen, *Student, IEEE*.

*Abstract*—This paper documents the ideation, design, implementation, and testing of a sequential circuit project in Verilog Hardware Description Language. The background and motivation of the project will be explored, along with the specifications of implementation. The final section of this paper performs data analysis on results and concludes with relevant engineering concepts applied in the project.

*Index Terms*—Sequential, Hashing, Encryption, Verilog, Pattern Detection, ASCII, ModelSim.

# INTRODUCTION AND BACKGROUND

D

URING World War I and II, allied forces would intercept German military messages sent through telegraph or radio. Many of these messages are often encrypted with ciphers, the most notorious being the Enigma machine. In a similar spirit, the purpose of this project was to create a device that could track suspicious messages and relay them in the context of security. In today’s Age of Information, ciphers are not the primary encryption method since they are inherently reversible. Many modern methods involve hashing the string into a fixed sized value. Ideally, cryptographic hashing is deterministic (same input always hashes to same output), easily computable, collision free (no two different input produces the same output) and are impractical to reverse. As the project is an individual effort, all work submitted except the hashing algorithm pseudocode is the student’s original work. The waveform simulation software used for this project is ModelSim – Intel FPGA Starter Edition.

# GOALS AND SPECIFICATIONS

The goal of this project is to create a device that listens in on a serial 8-bit ASCII data stream to detect preconfigured string patterns. If a pattern is detected, then the device will perform a hashing function on the pattern and transmit it to the receiver. Finally, the receiver would compare the transmitted data with precomputed hashes to identify the pattern. Thus, this device is intended to capture suspicious words and transmit them over a potentially insecure channel which could be intercepted. Since hashing is intended to be irreversible, it serves as the encryption method over this open channel. For the device to be practical, the finite machine states must be independent from the pattern it is trying to detect. This allows the users to easily modify the Verilog design for different ASCII patterns. Thus, it was necessary to separate the data processing away from the control state machine.

# DESIGN AND IMPLEMETATION PROCESS

1. *Control Circuit Overhead*

Many designs of the final circuit were considered. Initially, incorporating the specific characters to detect into the controls like binary sequence detectors yielded an extremely complex state diagram and table. Therefore, a more modular approach was chosen as the design plan. Counters were an extremely essential component of this project, as they were utilized to keep track of the character detected so far, as well as the hashing and transmission loops. Fig. 1 depicts the state diagram of the overhead controls. The circuit was designed as a Moore machine, with the desirable number of four states (being a power of 2).

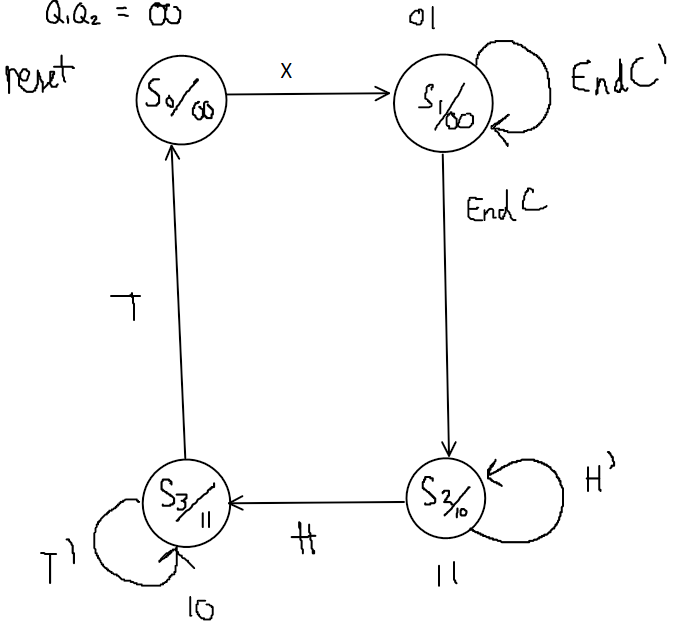
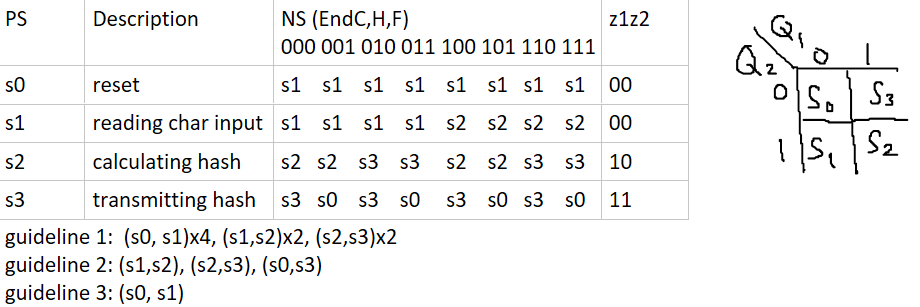


Fig. 1. State diagram of the control circuit. Although it appears simple, the complexity is merely shifted into the inner data processing modules.

S0 indicates the reset state of the device, which always leads to S1, the pattern detection state. An external module will detect if any patterns from a given list is found. If so, it will set the EndCheck signal, signaling the controls to transition to state S2. Upon entering S2, another module will start hashing the pattern and generate a high HashDone signal when complete. When receiving the H signal, the controls will transition to state S3, which will begin transmitting the hash value to the receiver. Upon completion, the device will go into its reset stage and the cycle again. There are two Moore outputs, Z1 and Z2 to indicate to the user which part of the process the machine is currently in. If the user observes 00, then it is in the reset state. If Z1 is HIGH, then the user knows that the machine has detected a pattern and must prepare to start receiving the hash. The moment Z2 is HIGH signals to the user to start receiving the hash serially every clock edge.



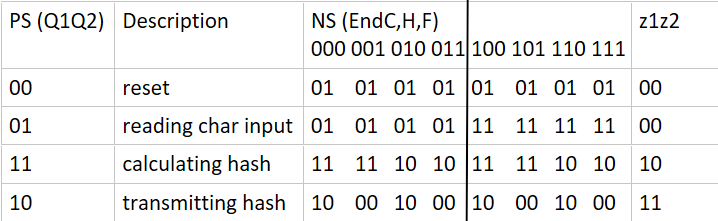


Fig. 2. State table of the controls circuit and assignment optimization.

Fig. 2 shows enumerating the states into a table. No further state reductions are possible because for S0 and S1 to be equivalent requires S1 and S2 to be equivalent (impossible as they both have different outputs). Since the circuit does not have any don’t care outputs, there is no possibility of compatible states. Next, the state assignments were enumerated according to the three optimization guidelines: states with the same next states are given highest adjacency priority, next states within the same row are given next priority, and finally states with the same outputs are given priority if possible. Thus, the table to the top right of Fig. 2 is the optimal solution that incorporates all three guidelines. Upon performing two   
5-variable Karnaugh maps and analysis of the outputs, the equations yield:

1. *Pattern Detection Module*

The pattern detection module is first of three inner modules essential to the functionality of the device. It takes in the   
8-bit ASCII character input, clock, and reset signals and outputs the End and k-bit output specifying which pattern from a list of k elements it has detected. For this particular project, k = 3 and the patterns are arbitrarily chosen to be “APPLES”, “BANANAS”, and “ORANGES”. Fig. 3 shows the block diagrams and nested inner modules of this aspect of the machine.

Diagram, schematic

Description automatically generated

Fig. 3. Pattern detection module consists of various nested sequential and combinational logic.

This inner device consists of three parallel sequence detectors all for with serial input aligned synchronously with the clock. The counter modules are simple synchronous counters that takes in a 4-bit number N, then counts to N at every rising edge if the Up signal is high. There is also a synchronous reset for the counter, and it also displays the current count value as well as a count done signal (HIGH if the counter counted to N). All counters reset to zero and N is assigned the length of the pattern to detect. Initially, the current count (being zero) is fed into a pattern lookup table, which outputs the ASCII character of the string at the current position (i.e. pattern[n]). The input X is then compared with this value and signals the counter to increment if they are both equal, else reset. Since the input X changes serially at each clock edge, the counters here are synchronous as well. Thus, if any of the counters have finished counting, it means that a pattern is detected because N successful character comparisons has occurred (with N being the length of the pattern to check). The count done signal is then merged into a k-bit value (k = 3 in this case), and the EndCheck signal turns HIGH whenever the value is greater than zero. The merged 3-bit signal will be 000 unless a pattern is detected, which will flip any of the bits to a 1. The purpose of this is for the machine to identify which pattern has been detected in a pseudo one-hot like implementation. The Reset signal fed into this pattern detection module propagates to all inner counters so pattern detection is disabled if the outside Reset signal is HIGH. This detection module will always be reset unless the control circuit is in state S1.

1. Hashing Module

While Many hash algorithms were considered for the project, the DJB2 Hash by Dan Bernstein was chosen to keep an efficient time and space complexity within the scope of hardware.

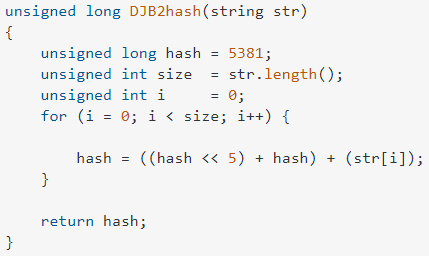


Fig. 4. Dan Bernstein’s DJB2 Hash C++ code courtesy of Matt Chaplin [1].

The hashing function shown above was implemented in hardware, shown by the following the block diagram of the hashing module.

Diagram, schematic

Description automatically generated

Fig. 5. Hashing module block diagram.

Eight bytes were determined to be large enough to store the hashes of all English words and phrases. The initialization of pattern length and hash value of 5381 occurs whenever this module is reset. Since the input is the 3-bit data from the pattern detection module, a 3-bit register is necessary to save this information when the control circuit enters state S2. This is because the pattern detection output gets reset to 000 whenever it resets outside of state S1. Here, the 3-bit word location is used to load the counter N with the length of that pattern from the list. Then, the counter iterates through every character of the pattern, and adds it to the current hash value and the hash value shifted to the left 5 bits. This is then stored into the 64-bit hash register at every rising edge. An additional 64-bit shift register is required to perform the left shift, which is shifted at every falling edge to avoid timing errors. Once the counter completes the for loop (iterates from 0 to length of the pattern), then the hash module will output the HashDone signal to be HIGH. This indicates to the controls that the current 64-bit hash value at this moment in time is the correct hash value that should be read and transmitted at state S3. Upon leaving S2, the hash module will always be reset.

1. Transmission Module

The transceiver module takes the 64-bit hash value from the previous module, stores it inside a shift register, and outputs the hash in 8-bit bytes at every rising edge.

Diagram, schematic

Description automatically generated

Fig. 6. Transmitter module block diagram.

Fig. 6 shows the internal block diagram of this module. When the Reset signal is on, the shift register will always load the input hash value. When the reset signal is LOW, the module will extract the smallest 8 bits from the register via masking, then output it at every rising edge. Then, the shift register will right shift to transmit the next byte. Since there are eight bytes in a 64-bit value, the counter facilitates a for loop that iterates eight times on every rising edge. Once it is done counting, the TransmitDone signal output will turn on to let the control circuit know that hash transmission has been completed.

1. *Controls Circuit Overhead Revisited*

The following figure represents the block diagram of the main driver controls of the program, which combines and synchronizes the three previous modules into one functional unit. In addition to the Boolean equations derived in part *A*, further combinational logic was required for the each reset input of the smaller modules. Let Reset be the universal reset signal given into the device from outside, then the internal reset signal Boolean expressions determined by state table are:

Diagram, schematic

Description automatically generated

Fig. 7. Transmitter module block diagram.

# WAVEFORM RESULTS AND ANALYSIS

Since the nature of the project is solely communication and data based, ModelSim waveform outputs of preconfigured test benches serve as excellent testing and debugging tools for the project. If a module did not function correctly, the Verilog code was analyzed for syntax and logic errors. Thus, through cycles of generating code logic, debugging, and refining, the end product coalesced into the digital circuit design elaborated in the previous section.

1. *Pattern Detection Data and Analysis*

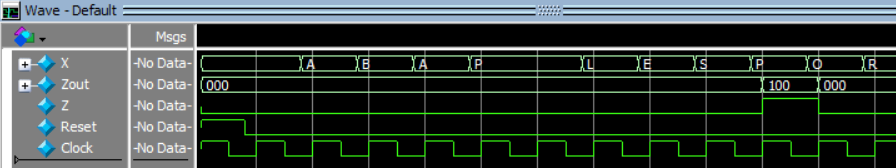


Fig. 8. Pattern detection module ModelSim waveform.

Fig. 8 displays the test bench for the pattern detection module. Here, the reset signal is initially HIGH, then set to LOW. The input changes a few nano seconds before the triggering clock edge to ensure proper functionality. The sequence “ABAPPLESPOR…” was used as the test input. Here, the combined signal Zout and Z was equal to 0 until “APPLES” was detected. Immediately at the next clock edge, Zout becomes 100 (indicates that this is the first pattern in the list), and Z becomes 1, which signals to the main control circuit to transition into hashing. Due to the nature of the device, the major disadvantage of the circuit is that it cannot detect any patterns that occur during the hashing or transmission process since the pattern detection module will be reset during that period. The main device will simply ignore any new signals generated here. Parallel threading the detection process was not considered because of variable lengths that the patterns could have, as well as adding needless complexity to the circuit to save and dispense the patterns found during the other processes.

1. *Hash Function Data and Analysis*

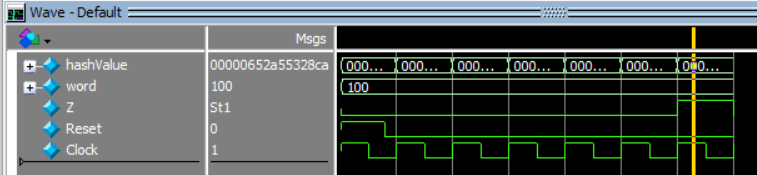


Fig. 9. Hash function ModelSim waveform.

Fig. 9 shows the hash value being updated at every clock edge after the Reset is LOW. The test pattern here is “APPLES” so the HashDone signal (Z in this case) turns HIGH after the sixth clock edge since the pattern has six characters. Furthermore, the hash value is 652a55328ca, which matches the computed hash shown in Fig 10. Further testing shows that the hash module correctly hashes the other patterns as well. As with the pattern detection module, the hash value output gets reset immediately when the control circuit leaves the hashing state, which means that this value must be stored inside of a register for the transmitter module to access during the transmission state.

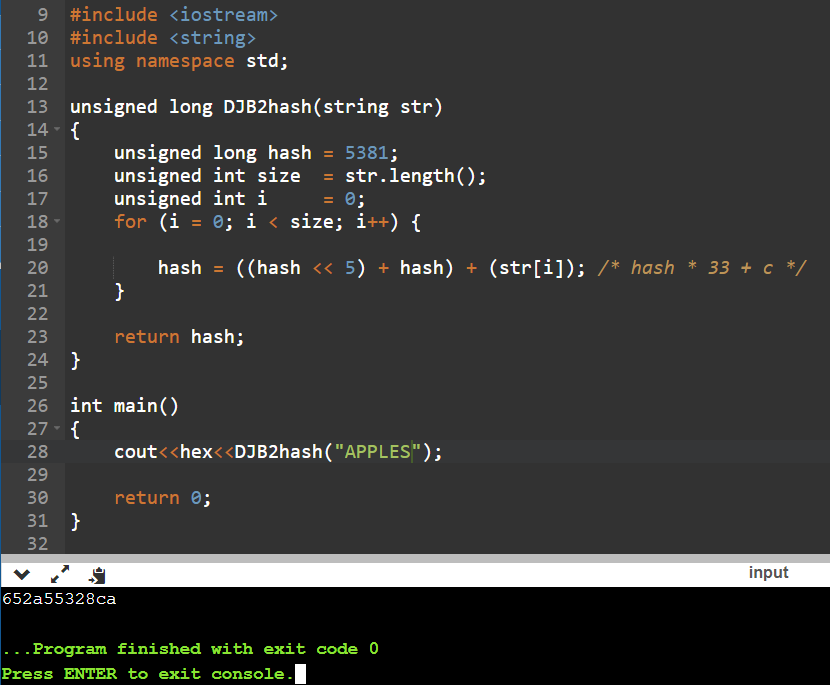


Fig. 10. C++ computation of the hash on “APPLES”.

1. *Transmitter Data and Analysis*

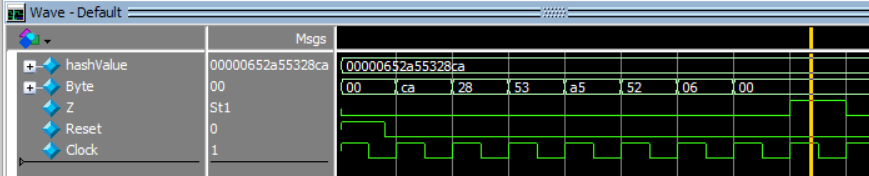


Fig. 11. Serial byte transmission of the hash value.

Fig. 11 shows that the TransmittDone (Z) output is HIGH at the eighth clock cycle after Reset has been turned off. The byte patterns received by the user are in little endian format since the program will output the least significant byte first. Otherwise, the default output of this module is hex 00 when it is reset. Furthermore, since the counter always counts to eight, there will be some leading zeros if the hash value does not take up the entire 64 bits.

1. *Control Circuit Data and Analysis*

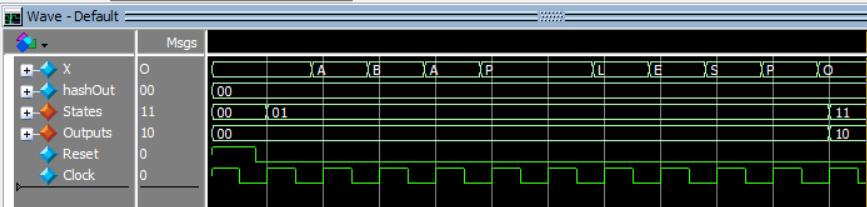


Fig. 12. Controls circuit showing the reset and pattern detection states.

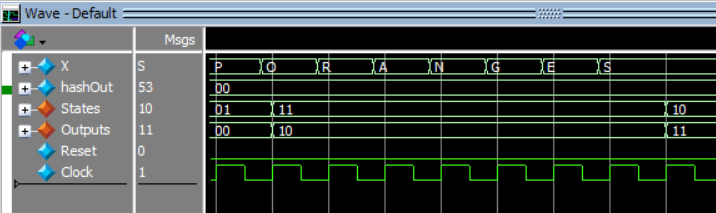


Fig. 13. Controls circuit showing the hash computation state.

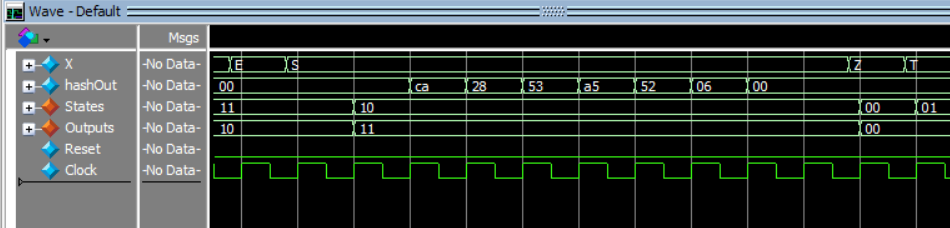


Fig. 14. Controls circuit showing the hash transmission and reset states.

Initially, the test bench resets the entire device to S0, then starts transmitting serial ASCII data into X. The device transitions to S1 and begins pattern detection for the three words from the internal storage. Once it has detected “APPLES”, the device transitions into state S2 and outputs 10 after two clock cycles. The first delay here occurs because the EndCheck signal takes one cycle to synchronize from the inner pattern detection module to the control and another cycle for the control circuit to synchronize with the hash function module. After this point, the device will not consider any new inputs until it is reset. After the hash function module computes the hash value, a delay of one clock cycle is needed to synchronize with the controls circuit before transitioning to S3. Upon entering S3, the output will become 11 and the receiver will have one clock cycle to prepare for data coming in from the device. The hash output of Fig. 14 exactly matches the hash generated in figure 9, 10, and 11. Immediately after ending the eight bytes of the hash, the device will reset back to S0 and begin its cycle of detection, hash, and transmission again.

# CONCULSION

As with most things, there are certainly positive and negative aspects of the project. First, my knowledge of Verilog syntax and procedure beyond a doubt improved. While state assignments and optimization played a significant role at the beginning of the project ideation, its impact on the final product may not be as prevalent as I had envisioned. This is because two approaches were considered initially: either frontload the control circuit with all the data management and intricacies, or offload the complexity into smaller components for easier overhead facilitation. I opted for the latter since I had a lot more experience with software abstraction, separating the controls from the data. To implement a for loop, a topic taught in the advanced course ECE 176, a non-standard method involved synchronous counters nested inside the different modules to sync up the individual processes. In retrospect, I believe this project may have been a little too ambitious for ECE 106. Perhaps the individual pattern detection, hashing, and transmission modules are smaller mini projects rolled into one. There are many improvements that could be made, such as optimizing the two-cycle delay between detection and hash, streamlining the inner modules, and even utilize a better hashing algorithm. Despite the shortcomings, I still believe the project is a great success. From an engineering perspective, the device accomplishes every single feature specified in its goals and meets the sequential state machine project requirement.

# References

|  |  |
| --- | --- |
| [1] | "Stack Overflow," [Online]. Available: https://stackoverflow.com/questions/19892609/djb2-by-dan-bernstein-for-c. [Accessed 9 May 2021]. |

1. This manuscript was submitted on May 10, 2021 as part of the ECE 106 Final Design Project for Dr. Stillmaker at CSU Fresno. All design files, block diagram schematics, and Verilog files were submitted. [↑](#footnote-ref-1)