

**Princess Sumaya University for Technology**

**King Abdullah II School of Engineering**

**Computer Engineering Department**

**Computer Architecture (2)**

Secure Hash Algorithm 2 (SHA-256) **Pipelined Implementation**

**Authors: Supervisor:**

Omar Hamada 20191016 Dr. Awos Kanan

Taima’a Khallad 20200862

https://github.com/ScribblerCoder/ComputerArch2\_proj

May 27, 2023

**Abstract**

SHA-256 (Secure Hash Algorithm 2) is a set of cryptographic hash functions designed by the United States National Security Agency (NSA) and first published in 2001. [1] A hash function is a mathematical function that converts any digital data into an output string with a fixed number of characters. It is a widely used cryptographic hash function that produces a 256-bit message digest. This project presents a pipelined architecture hardware implementation for calculating the SHA-256 hash of any input by unrolling the repeated iterations in a systematic way, the project promises to achieve a higher throughput compared with the non-pipelined architecture. [2]

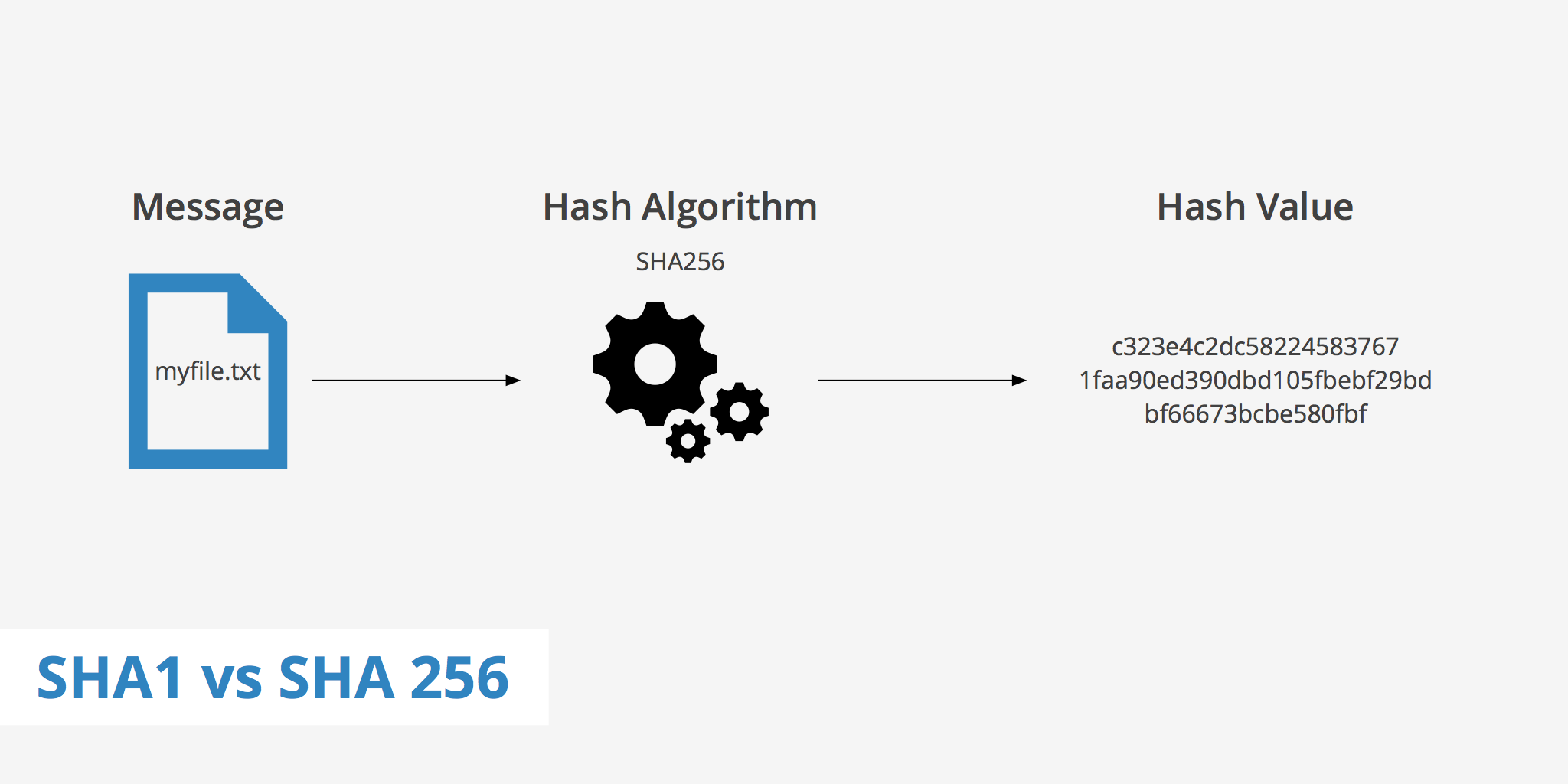


Figure 1 Hash Algorithm Explained [3]

**Introduction**

The pipelined architecture offers a highly efficient and parallelized approach to computing the SHA-256 hash. Designed by breaking it down into 64 stages (iterations), where each stage is a single SHA-256 round. The same operation is repeated 64 times which allows us to unroll this loop by repeating the same block 64 times to allow new input while still processing the old input.

To ensure the efficient processing of multiple messages in parallel, it splits each message into m blocks, then the pipelined architecture employs a strategy where the next block from a next message is assigned to a stage while the previous block within the same message is still being processed.

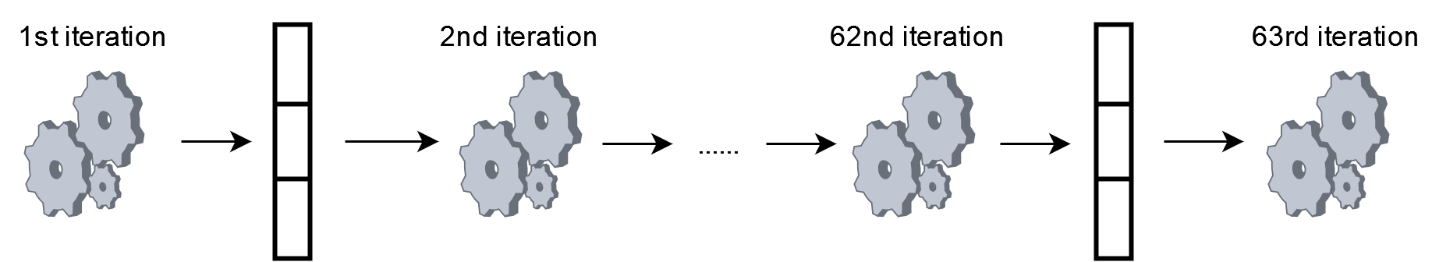


Figure 2 Diagram of the pipeline

**Proposed Design**

The following description provides a detailed explanation of each module and its functionality in the Verilog code. It specifically focuses on the ten stages (as a proof of concept) involved in the SHA-256 algorithm implementation.

sha256\_single\_iteration: This module represents a single iteration of the SHA-256 algorithm. It takes as input a message schedule word W, a constant K, and the current state variables a, b, c, d, e, f, g, h. It performs various logical and arithmetic operations to update the state variables based on the SHA-256 algorithm. The output of this module is the updated state variables a\_out, b\_out, c\_out, d\_out, e\_out, f\_out, g\_out, and h\_out.

The following figure represents the process of calculating a single iteration/round SHA-256 of an input.

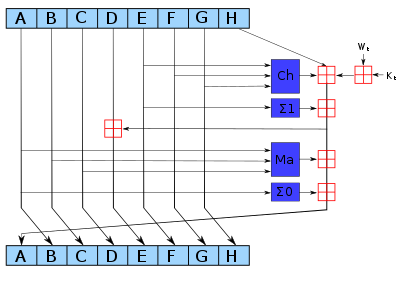


Figure 3 SHA-256 Single Round Block diagram [4]

## Input

The input blocks At through Ht are 32-bit words. They represent the current iteration’s input

W is a sequence of 32-bit words called the message schedule. Its generation is explained in the next section

K is a sequence of 64 constant 32-bit words (always fixed), K0, K1, K2, K3 … K63. In hex, these constant words are (from left to right): [7]

428a2f98 71374491 b5c0fbcf e9b5dba5 3956c25b 59f111f1 923f82a4 ab1c5ed5  
d807aa98 12835b01 243185be 550c7dc3 72be5d74 80deb1fe 9bdc06a7 c19bf174  
e49b69c1 efbe4786 0fc19dc6 240ca1cc 2de92c6f 4a7484aa 5cb0a9dc 76f988da  
983e5152 a831c66d b00327c8 bf597fc7 c6e00bf3 d5a79147 06ca6351 14292967  
27b70a85 2e1b2138 4d2c6dfc 53380d13 650a7354 766a0abb 81c2c92e 92722c85  
a2bfe8a1 a81a664b c24b8b70 c76c51a3 d192e819 d6990624 f40e3585 106aa070  
19a4c116 1e376c08 2748774c 34b0bcb5 391c0cb3 4ed8aa4a 5b9cca4f 682e6ff3  
748f82ee 78a5636f 84c87814 8cc70208 90befffa a4506ceb bef9a3f7 c67178f2

## Functions

Wt is calculated according to the following equation:

where M is the input to be hashed and , are described as follows

The blue blocks are the following functions:

The red block  is an addition operation modulo 232 (a 32-bit adder that allows overflow). [5]

## Output

The output blocks At+1 through Ht+1 are 32-bit words. These outputs are the next iteration’s input.

TenStagePipeline: This module represents a 10-stage pipeline implementation of the SHA-256 algorithm. It takes as input a clock signal (clk), a reset signal (reset), and an array of messages (message). The output of this module is the digest of the SHA-256 algorithm (digest).

Inside the TenStagePipeline module, there are several pipeline stages represented by registers (stage\_1\_a to stage\_10\_h). Each stage represents the state variables of a specific iteration of the SHA-256 algorithm. These stages are connected in a pipeline fashion, where the output of one stage becomes the input of the next stage.

Additionally, there are wire connections (stage\_X\_a\_out to stage\_X\_h\_out) between the output of one stage and the input of the next stage. These wires carry the updated state variables from one stage to the next.

The sha256\_single\_iteration module is instantiated 10 times, once for each stage, with the appropriate inputs and outputs connected. This ensures that each stage performs the necessary computations to update the state variables according to the SHA-256 algorithm.

The message array represents the order of the blocks of the messages, the blocks are represented in a sequential order, starting with the first block of the first message, followed by the first block of the second message, and so on, until the first block of the tenth message. Then, the second block of the first message would be processed, followed by the second block of the second message, and so on. This pattern continues for all of the 10 messages. The input message is processed by the SHA-256 algorithm, where each stage of the pipeline receives a portion of the message. The initial hash values (H) and constant values (K) are defined as parameters within the module.

Overall, the TenStagePipeline module implements the SHA-256 algorithm by dividing the computation into multiple pipeline stages, allowing for parallel processing and improved performance.

**Design block diagram**

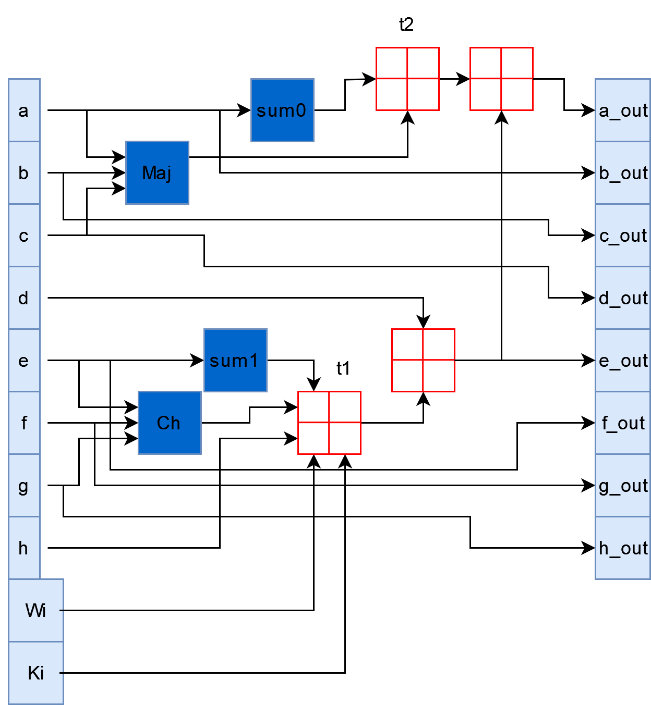


Figure 4 SHA-256 Single Iteration Block diagram

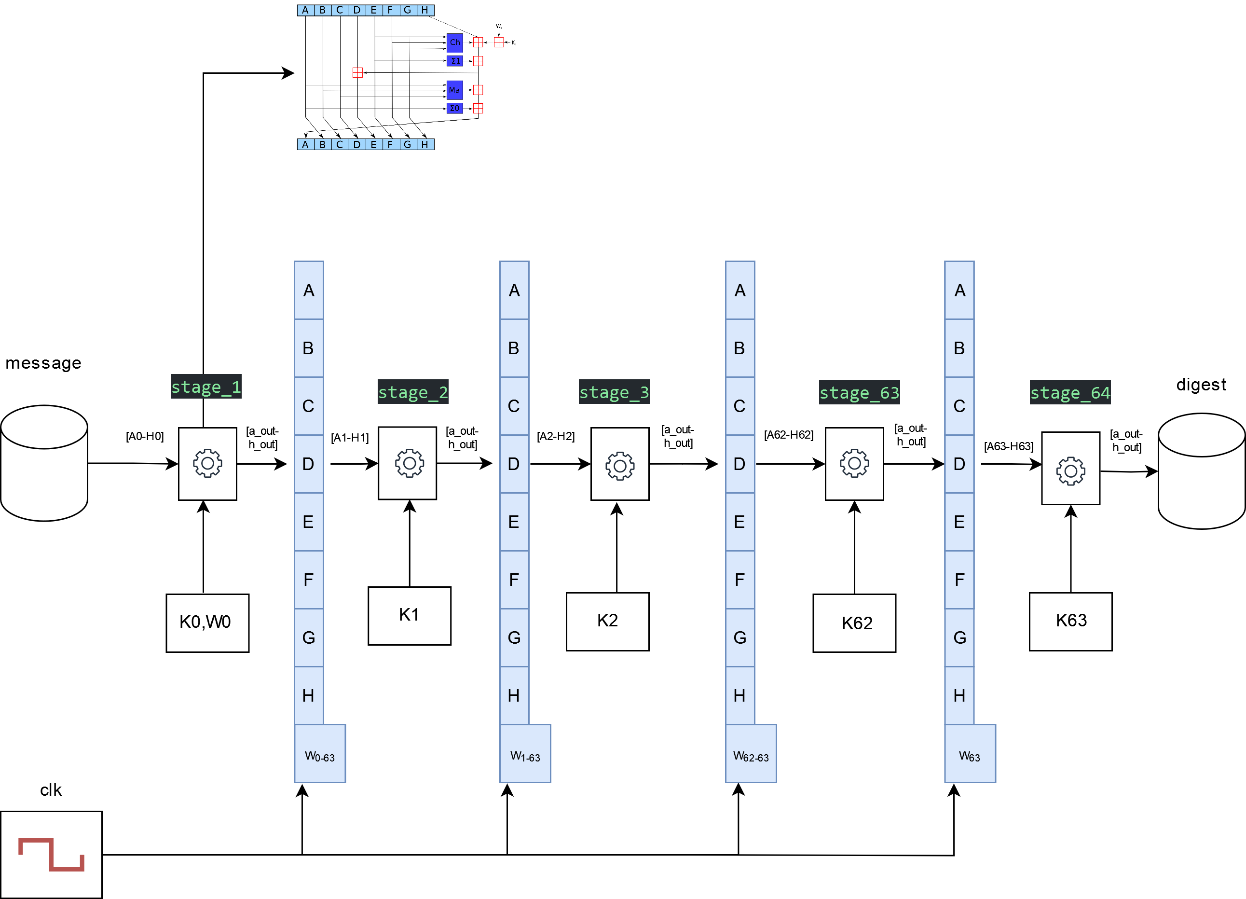


Figure 5 SHA-256 Pipelined Block diagram

**Design analysis**

The pipelined design allows for 64 messages simultaneously, the processing of each message is divided into m blocks with size 64-bytes as shown in Figure -6-, and each block depends on the previous block within the same message. This dependency arises from the nature of the SHA-256 algorithm, where the hash computation for each block is influenced by the hash result of the previous block.

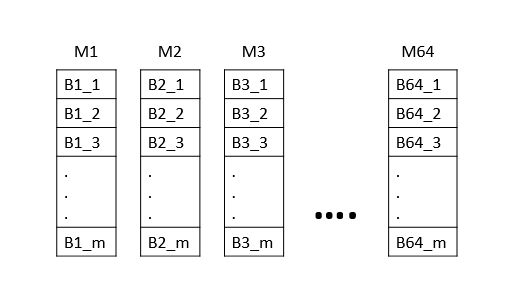


Figure 6 64 messages split into m blocks

The dependency between blocks within a message is managed by organizing the flow of data through the pipeline. As each stage completes the processing of a block, it passes the intermediate results to the subsequent stage while simultaneously receiving the next block from the following message in the pipeline. This mechanism ensures a continuous utilization of the pipeline, where blocks from different messages are processed concurrently, exploiting the parallel nature of the pipelined architecture, the following table shows the complete flow of blocks inside the pipeline :

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Cycles | Iter1 | Iter2 | Iter3 | Iter4 | Iter5 | Iter6 | Iter7 | Iter8 | Iter9 | Iter10 |
| 0. | B11 |  |  |  |  |  |  |  |  |  |
|  | B21 | B11 |  |  |  |  |  |  |  |  |
|  | B31 | B21 | B11 |  |  |  |  |  |  |  |
|  | B41 | B31 | B21 | B11 |  |  |  |  |  |  |
|  | B51 | B41 | B31 | B21 | B11 |  |  |  |  |  |
|  | B61 | B51 | B41 | B31 | B21 | B11 |  |  |  |  |
|  | B71 | B61 | B51 | B41 | B31 | B21 | B11 |  |  |  |
|  | B81 | B71 | B61 | B51 | B41 | B31 | B21 | B11 |  |  |
|  | B91 | B81 | B71 | B61 | B51 | B41 | B31 | B21 | B11 |  |
|  | B101 | B91 | B81 | B71 | B61 | B51 | B41 | B31 | B21 | B11 |
|  | B12 | B101 | B91 | B81 | B71 | B61 | B51 | B41 | B31 | B21 |
|  | B22 | B12 | B101 | B91 | B81 | B71 | B61 | B51 | B41 | B31 |
|  | B32 | B22 | B12 | B101 | B91 | B81 | B71 | B61 | B51 | B41 |
|  | B42 | B32 | B22 | B12 | B101 | B91 | B81 | B71 | B61 | B51 |
|  | B52 | B42 | B32 | B22 | B12 | B101 | B91 | B81 | B71 | B61 |
|  | B62 | B52 | B42 | B32 | B22 | B12 | B101 | B91 | B81 | B71 |
|  | B72 | B62 | B52 | B42 | B32 | B22 | B12 | B101 | B91 | B81 |
|  | B82 | B72 | B62 | B52 | B42 | B32 | B22 | B12 | B101 | B91 |
|  | B92 | B82 | B72 | B62 | B52 | B42 | B32 | B22 | B12 | B101 |
|  | B102 | B92 | B82 | B72 | B62 | B52 | B42 | B32 | B22 | B12 |
|  | B13 | B102 | B92 | B82 | B72 | B62 | B52 | B42 | B32 | B22 |
|  | B23 | B13 | B102 | B92 | B82 | B72 | B62 | B52 | B42 | B32 |
|  | B33 | B23 | B13 | B102 | B92 | B82 | B72 | B62 | B52 | B42 |
|  | B43 | B33 | B23 | B13 | B102 | B92 | B82 | B72 | B62 | B52 |
|  | B53 | B43 | B33 | B23 | B13 | B102 | B92 | B82 | B72 | B62 |
|  | B63 | B53 | B43 | B33 | B23 | B13 | B102 | B92 | B82 | B72 |
|  | B73 | B63 | B53 | B43 | B33 | B23 | B13 | B102 | B92 | B82 |
|  | B83 | B73 | B63 | B53 | B43 | B33 | B23 | B13 | B102 | B92 |
|  | B93 | B83 | B73 | B63 | B53 | B43 | B33 | B23 | B13 | B102 |
|  | B103 | B93 | B83 | B73 | B63 | B53 | B43 | B33 | B23 | B13 |
|  |  | B103 | B93 | B83 | B73 | B63 | B53 | B43 | B33 | B23 |
|  |  |  | B103 | B93 | B83 | B73 | B63 | B53 | B43 | B33 |
|  |  |  |  | B103 | B93 | B83 | B73 | B63 | B53 | B43 |
|  |  |  |  |  | B103 | B93 | B83 | B73 | B63 | B53 |
|  |  |  |  |  |  | B103 | B93 | B83 | B73 | B63 |
|  |  |  |  |  |  |  | B103 | B93 | B83 | B73 |
|  |  |  |  |  |  |  |  | B103 | B93 | B83 |
|  |  |  |  |  |  |  |  |  | B103 | B93 |
|  |  |  |  |  |  |  |  |  |  | B103 |

Table 1 data flow of blocks

Each block **Bx,y** requires N iterations, assuming Bx,y starts its execution at cycle t, it completes its processing at cycle **t+N**. The subsequent block, Bx,y+1, relies on the completion of Bx,y before it can begin its execution. Therefore, Bx,y+1 can start at cycle t+N, ensuring a smooth transition in the pipeline. The cycle number t (the starting time of a Bx,y), can be calculated as:

***(1.5)***

Where y represents the block number ranging from 1 to m within a message, and x represents the message number ranging from 1 to 64.

To handle more than 64 messages, the pipelined architecture employs a mechanism where the messages are split into chunks of 64 blocks. Each chunk is processed in the pipeline, the subsequent chunks are temporarily held in a buffer, awaiting their turn in the pipeline.\

**Test runs and discussion**

A single iteration test run:

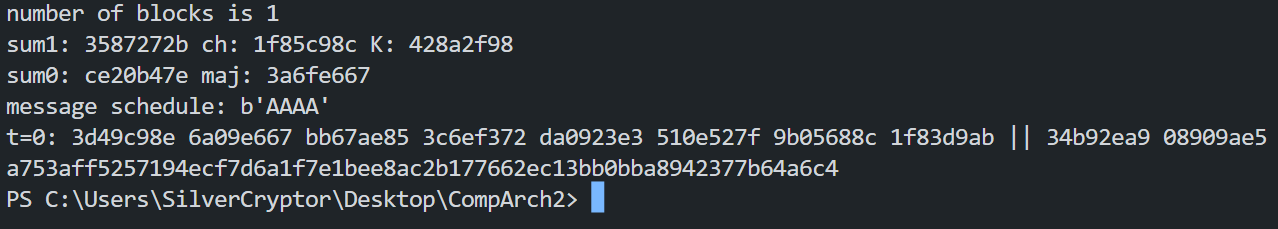


Figure 7 a software run of a single iteration

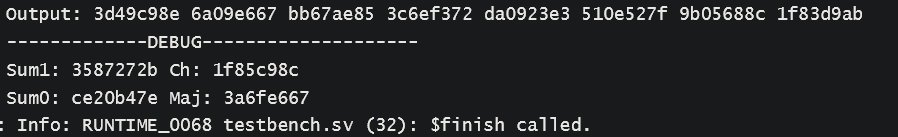


Figure 8 verilog test run for a single iteration

Pipelined 10 iterations test run:

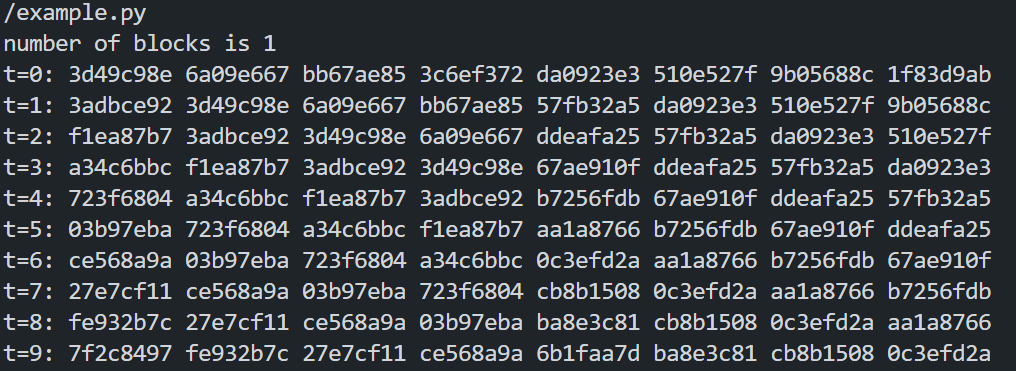


Figure 9 software test run for a pipelined 10 stages

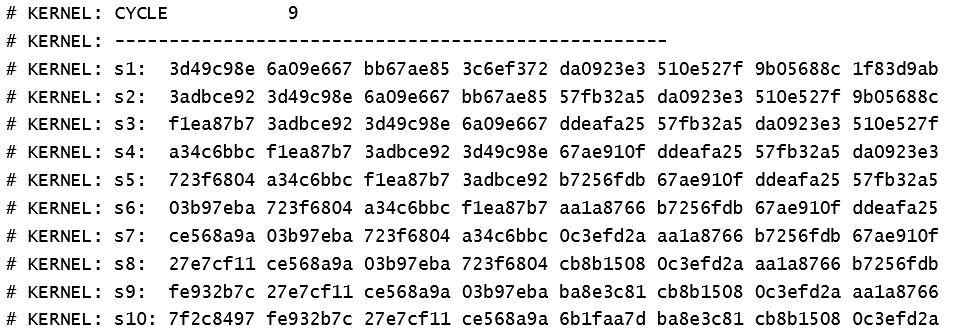


Figure 10 verilog test run for pipelined 10 stages

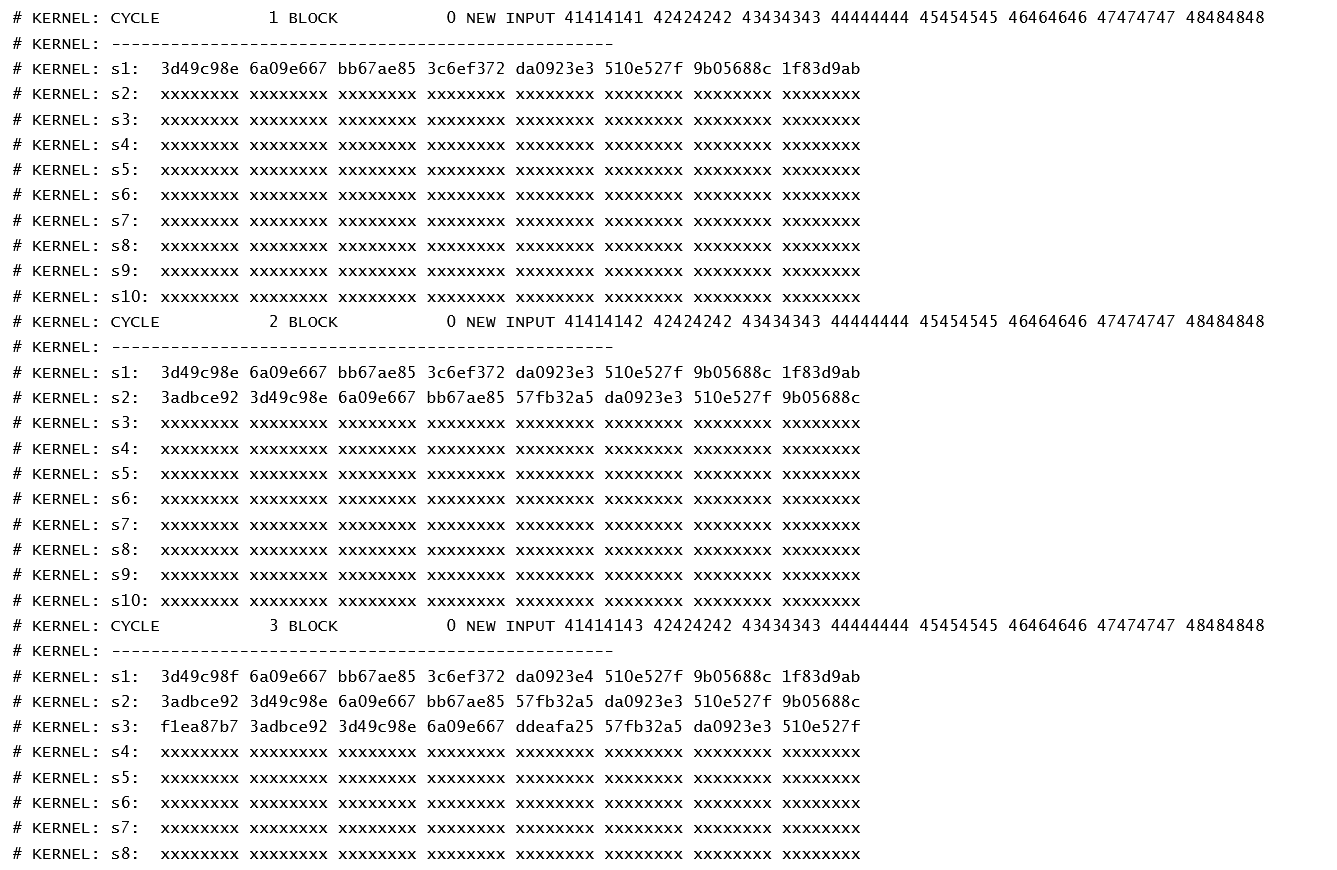


Figure pipelined stages cycle by cycle

A comparison between the non-pipelined and the pipelined implementation:

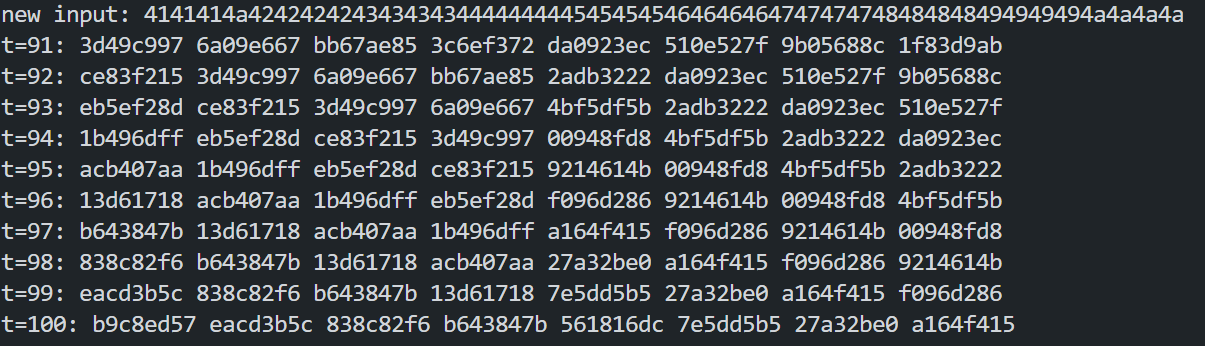


Figure 12 non-pipelined software test run

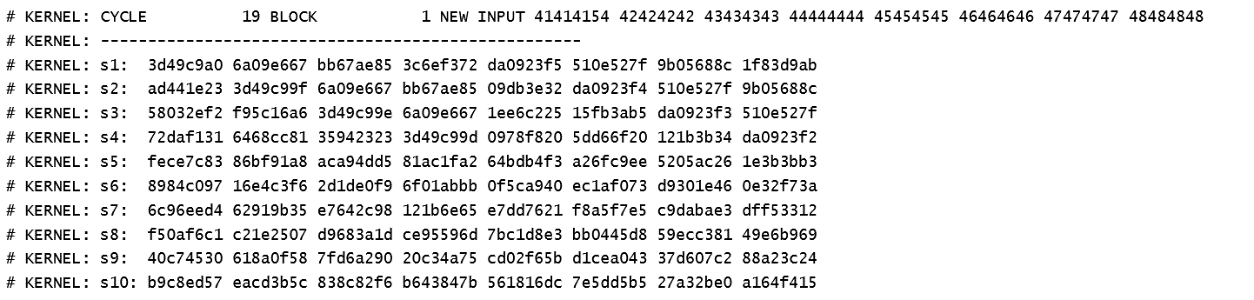


Figure 13 pipelined test run

For the same number of blocks (10 blocks) in non-pipelining it takes 100 cycles while in pipelining it takes 19 cycles to finish, which is ~ 5 times less than the non-pipelining.

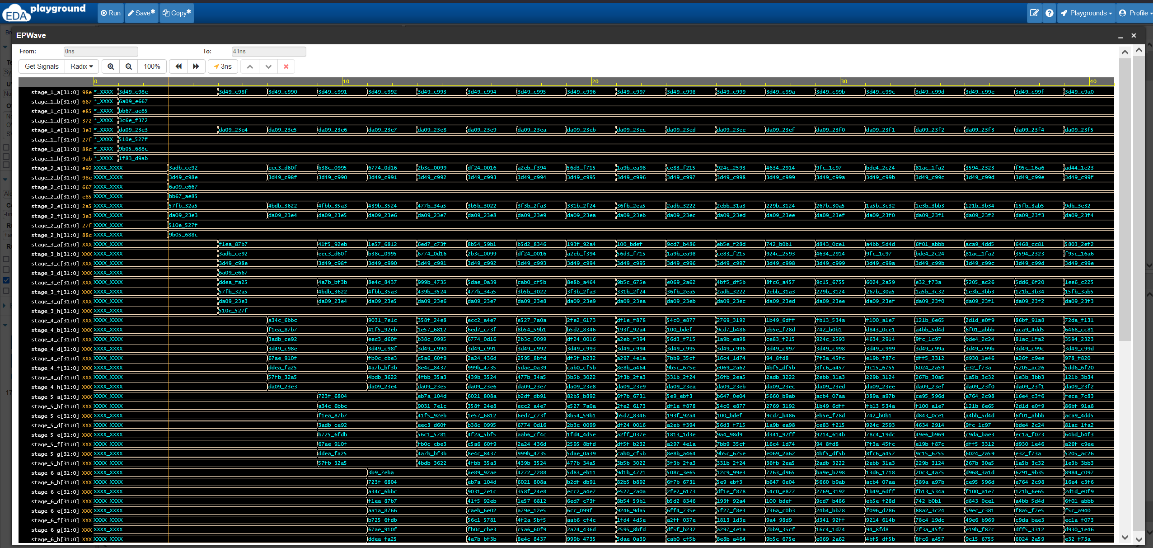


Figure pipelined waveforms

**Performance analysis**

In a non-pipelined architecture, instructions are executed sequentially, one after another, without overlapping stages. Each instruction must complete its execution before the next instruction can be fetched and executed. While the pipelined architecture offers several advantages in terms of performance, by overlapping different stages of instruction execution, the pipelined architecture can achieve higher instruction throughput and reduce the overall cycles needed:

**(1.6)**

**(1.7)**

According to the equations mentioned above. The pipelined architecture typically has a lower number of cycles needed compared to the non-pipelined architecture. This reduction is due to the overlap of instruction execution in different pipeline stages. This results in faster program execution and improved system efficiency.

**Conclusion and Future Work**

The pipelined architecture of SHA-256 has shown improvements in terms of performance and throughput compared to non-pipelined implementations. The pipelined architecture allows for a continuous flow of data through the different stages of the algorithm and enables overlapping computations, this results in improved efficiency and achieving a higher throughput.

In this project only 10 stages were implemented as a proof of concept. Future work will include the full pipeline of 64 stages and the ability to handle more than 64 message, we can make multiple instances of the proposed architecture shown in figure--, that work simultaneously, while each instance handles up to 64 message.

**References**

* [1] <https://en.wikipedia.org/wiki/SHA-2>
* [2] ULTRA HIGH SPEED SHA-256 HASHING CRYPTOGRAPHIC MODULE FOR IPSEC HARDWARE/SOFTWARE CODESIGN Harris Michail 1, 2, 3, George Athanasiou1 , Angeliki Kritikakou1 ,Costas Goutis
* [3] <https://github.com/cian2009/SHA-256>
* [4] [File:SHA-2.svg - Wikimedia Commons](https://commons.wikimedia.org/wiki/File:SHA-2.svg)
* [5] "Announcing the SECURE HASH STANDARD" ([PDF](https://csrc.nist.gov/csrc/media/publications/fips/180/2/archive/2002-08-01/documents/fips180-2.pdf)). Federal Information Processing Standards Publication 180-2. 1 August 2002.