ECE 111: Advanced Digital Design Project
Prof. Yatish Turakhia

Final Project Report

June 10, 2023

Conner Hsu (A16665092)
Haozhang Chu (A16484292)
Kirtan Shah (A16227067)

Contents

1 Simplified SHA-256 **2**
1.1 Introduction 2
1.2 Algorithm 2
1.3 Simulation Results 3

2 Bitcoin Hashing **4**
2.1 Introduction 4
2.2 Algorithm 4
2.3 Resource Usage 4
2.4 Simulation Results 4

1 Simplified SHA-256

1.1 Introduction

Hash functions are powerful tools that play a vital role in various areas of computer science and information security. At their core, hash functions are mathematical algorithms that transform data of any size into a fixed-size string of characters, known as a hash value or hash code. This transformation is designed to have a few important properties:

- compression; the output hash value is fixed in size regardless of the size of the input.
- avalanche effect: a small change in the input results in a huge change in the output.
- determinism: the same input must always generate the same output.
- pre-image resistant: an inverse hash function should not exist and it should be very difficult to determine the input that generated a given output.
- collision resistance: the hash function should be nearly completely injective and thus should almost never have two inputs map to the same output.

For cryptographic applications, avalance effect, noninvertibility and injectiveness are very important properties. These properties all together make hash functions invaluable for tasks such as data retrieval, data integrity verification, password storage, and digital signatures. By producing unique and irreversible hash values, hash functions enable efficient data indexing, quick data comparison, and secure authentication. Whether it's ensuring data integrity, enhancing search performance, or providing robust security measures, hash functions are essential components in modern computing systems.

In this report, we explore the implementation of a particular cryptographic hash function called Secure Hashing Algorithm - 256 or SHA-256 for short. We will build this using System Verilog and use the Arria-II FPGA to realize the implementation on hardware.

1.2 Algorithm

Before running the SHA-256 algorithm, the input message must be broken up into blocks of 512 bits. The final block must contain at least 66 extra bits; a 1 followed by at least one 0, and 64 bits equal to the size of the original input message. The number of zeros padding between the first 1 and the message size bits depends on how many bits are needed to make the last block 512 bits long.

For example, suppose the message is 640 bits long. This means that the first block will contain the first 512 bits of the message, and the second block will contain the last 128 bits of the message. Next, a 1 will be appended to the last block making it 129 bits long. Then, 319 zeros will be appended to make the last block now 448 bits long. Last, 64 bits will be appended to store the message size and the last block will not be 512 bits long.

As another example, suppose that the input message divides evenly into 512 bit blocks meaning that the message size is some multiple of 512. In this case, the final block will have a 1 in the beginning, then 448 zeros, then end off 64 bits for the message size. The number of zeros is increased to 448 to ensure that this final block is 512 bits long.

From here, the state diagram of the SHA-256 can be described.

- IDLE: A state the waits for a signal to begin computation

- READ:
- BLOCK:
- COMPUTE:
- WAIT:

1.3 Simulation Results

2 Bitcoin Hashing

2.1 Introduction

2.2 Algorithm

2.3 Resource Usage

2.4 Simulation Results