Mini Project Report On

IMPLEMENTATION OF SECURE HASH ALGORITHM (SHA-1) USING VERILOG

Group – 14

Submitted for Mini project of **EC332-DIGITAL SYNTHESIS LABORATORY** as partialfulfillment of B.Tech Course.

By 1) 21EEB0A41- Satya Pramod Yadav

2)21EEB0A42- Manish Srivastava

3)21EEB0A43-Akash Chowdary

Under Supervision of:

1) Assistant Professor

Dr. Ekta Goel

AIM: To implement Secure Hash Algorithm using Verilog HDL

Software used : Xilinx Vivado

Design Suite is a software suite produced by Xilinx for synthesis and analysis of hardware description language (HDL) designs, superseding Xilinx ISE with additional features for system on a chip development and high-level synthesis. Vivado was introduced in April 2012, and is an integrated design environment (IDE) with system-to-IC level tools built on a shared scalable data model and a common debug environment. Vivado includes electronic system level (ESL) design tools for synthesizing and verifying C-based algorithmic IP; standards based packaging of both algorithmic and RTL IP for reuse; standards based IP stitching and systems integration of all types of system building blocks; and the verification of blocks and systems.

THEORY:

Introduction:

Cryptography is one of the most useful fields in the wireless communication area and personal communication systems, where information security has become more and more importantarea of interest . Cryptographic algorithms take care of specific information on security requirements such as data integrity, confidentiality And data origin authentication. To assure that a communication is authentic, the authentication service is of much concern. The function of authentication services is to assure recipient that the message is from the source it claims to be. In computer security, the process of attempting to verify the digital identity of the sender of a piece of information is known as authentication.

In order to make a very secure cryptographic portable electronic device, the selected well-known algorithm must be trusted, time-tested and widely peer-reviewed in the global cryptographic community. A one-way hash function is an algorithm that takes input data and irreversibly creates a digest of that data. One of the trusted one-way hash function are SHA-1 (Secure Hash Algorithm)

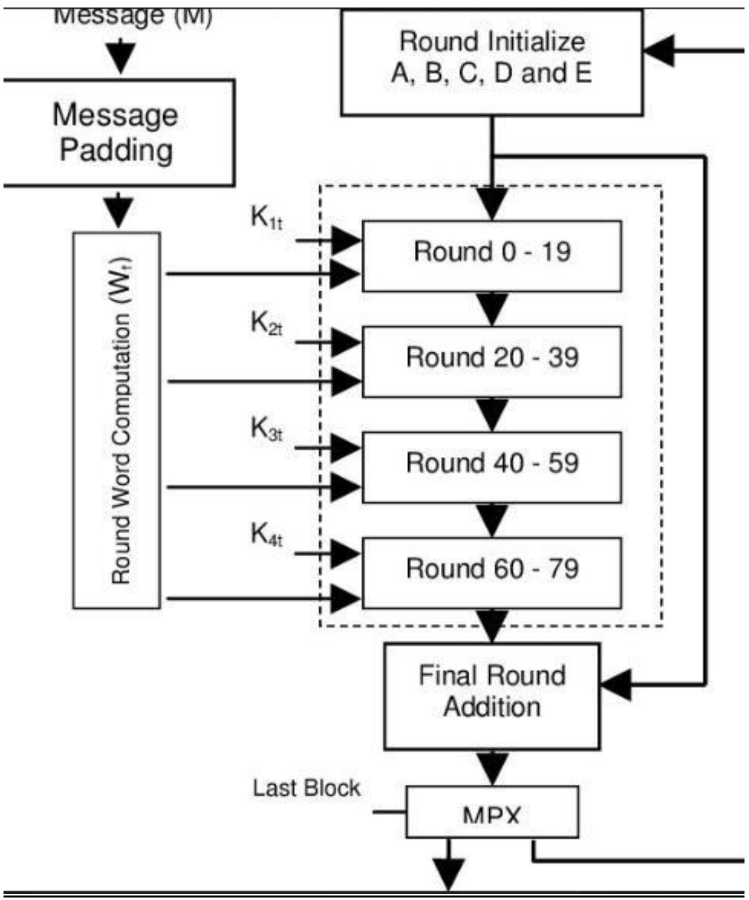
SHA algorithms are called secure because, for a given algorithm,it is computationally infeasible

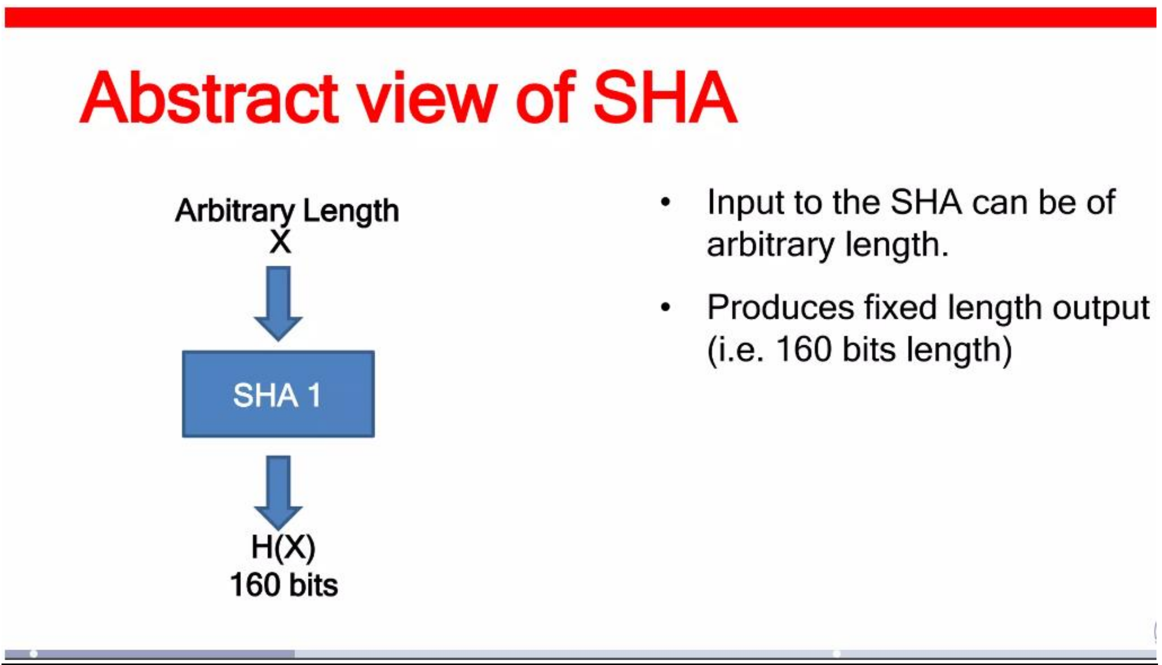
1) to find a message that corresponds to a given message digest or

2)to find two different messages that produce the same message digest. Any change to a message will, with a very high probability results in a different message digest. This will result in a verification failure when the secure hash algorithm is used with a digital signature algorithm or a keyed-hash message authentication algorithm.

Topics

Block Diagram

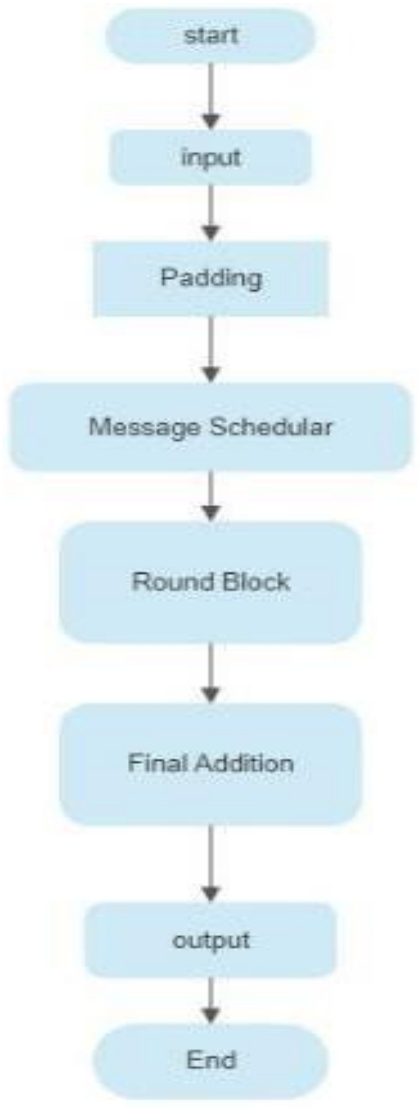




Abstract view of SHA

- Input to the SHA can be of arbitrary length.
- Produces fixed length output (i.e. 160 bits length)

FLOW CHART:


start
input
Padding
Message Schedular
Round Block
Final Addition
output
End

METHODOLOGY(WORKING OF SHA ALGORITHM):

SHA depends on and shares the similar building blocks as the MD4 algorithm. The design of SHA introduced a new process which develop the 16-word message block input to the compression function to an 80-word block between other things.

The processing of SHA works as follows –

- **Padding**
- **Compression function**
- **Final addition**

Message Padding process:

The SHA-1 is used to compute a message digest for a message or data file that is provided as input. The message or data file should be considered to be a bit string. The length of the message is the number of bits in the message (the empty message has length 0). If the number of bits in a message is a multiple of 8, for compactness we can represent the message in hex. The purpose of message padding is to make the total length of a padded message a multiple of 512. The SHA-1 sequentially processes blocks of 512 bits when computing the message digest. The following specifies how this padding shall be performed. As a summary, a "1" followed by m "0"s followed by a 64-bit integer are appended to the end of the message to produce a padded message of length 512 * n. The 64-bit integer is l, the length of the original message. The padded message is then processed by the SHA-1 as n 512-bit blocks.

Suppose a message has length $l < 2^{64}$. Before it is input to the SHA-1, the message is padded on the right as follows:

a. "1" is appended. **Example:** if the original message is "01010000", this is padded to "010100001".

b. "0"s are appended. The number of "0"s will depend on the original length of the message. The last 64 bits of the last 512-bit block are reserved for the length l of the original message.

Example: Suppose the original message is the bit string
01100001 01100010 01100011 01100100 01100101.

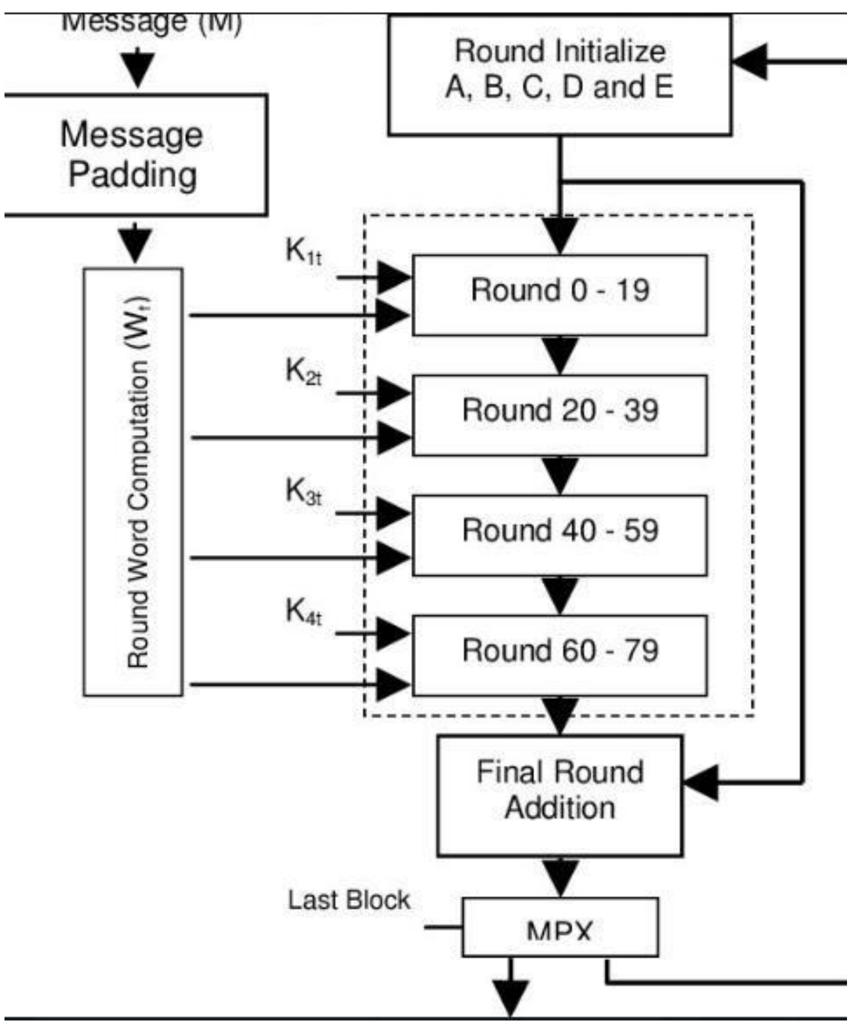
After step (a) this gives
01100001 01100010 01100011 01100100 01100101 1.

Since l = 40, the number of bits in the above is 41 and 407 "0"s are appended, making the total now 448. This gives (in hex)

```
61626364  65800000  00000000  00000000
00000000  00000000  00000000  00000000
00000000  00000000  00000000  00000000
00000000 00000000.
```

COMPRESSION FUNCTION :

Now let’s take a deeper look at the compression function in the above image. The compression function in itself has a total of 80 rounds in it and keep it in mind that these 80 rounds take place for each part of the image that when padded sums up to the 512 bits each.



Now, the next part is that these 512 bits are further divided into 16 parts each of 32 bits and these aremarked from M1 to M16.

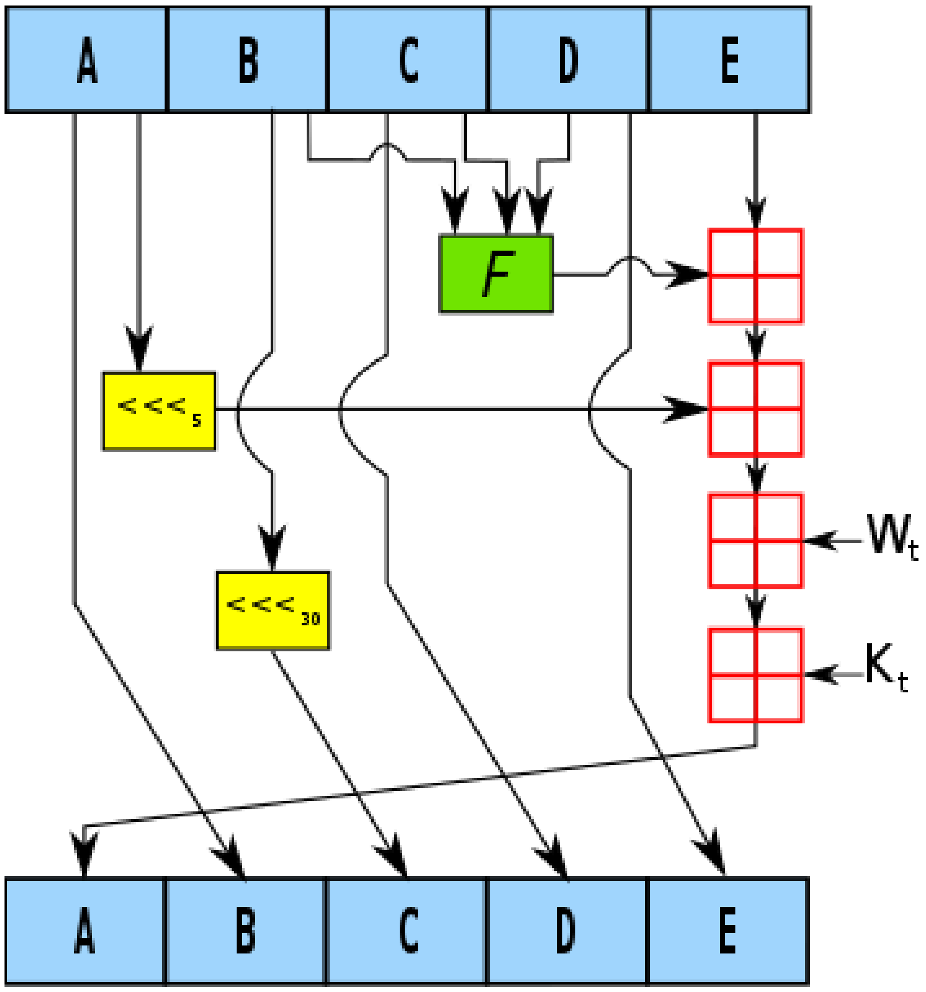
So, currently, we have 80 rounds of computation left to be carried out, with the 512 bits message that we have divided into 16 parts each of 32 bits.

The problem is that we have only 16 distinct parts of the message and 80 rounds of computation hence we repeat these 16 parts over and over for five times in the exact sequence they are in M1 to M16.

In the beginning, we have 160 bits input, we break it down into 5 parts which we name to be A, B, C, D & E. The initial values for A, B, C, D & E are mentioned below

```
A = 0x67452301
B = 0xEFCDAB89
C = 0x98BADCFE
D = 0x10325476
E = 0xC3D2E1F0
```

Once, we break them down into these 5 parts then we again have a set of complex procedures that we carry out in each of these rounds. That is the base core and reason for the complexity of the SHA-1 algorithm.



We can see that I have mentioned F(t), W(t) & K(t). We already know about W(t), these are the set of 32 bits, part of the 512 bits message but we are still unaware of F(t) and K(t). These are unique set of functions and values that are used to compute the hash and they remain constant for every 20 rounds i.e. they change 4 times while 80 rounds of computations take place. For every 20 rounds, F(t) and K(t) are constant they have a set of predefined values and function description which remains common depicted below.

FUNCTIONS USED:

A sequence of logical functions $f_0, f_1, \ldots, f_{79}$ is used in the SHA-1. Each f_t, $0 <= t <= 79$, operates on three 32-bit words

B, C, D and produces a 32-bit word as output. f_t(B,C,D) is defined as follows: for words B, C, D,
f_t(B,C,D) = (B AND C) OR ((NOT B) AND D) (0 <= t <= 19)

f_t(B,C,D) = B XOR C XOR D (20 <= t <= 39)

f_t(B,C,D) = (B AND C) OR (B AND D) OR (C AND D) (40 <= t <= 59)

f_t(B,C,D) = B XOR C XOR D (60 <= t <= 79).

CONSTANTS USED:

A sequence of constant words K(0), K(1), ... , K(79) is used in the SHA-1. In hex these are given by
K = 5A827999 (0 <= t <= 19)

K_t = 6ED9EBA1 (20 <= t <= 39)

K_t = 8F1BBCDC (40 <= t <= 59)

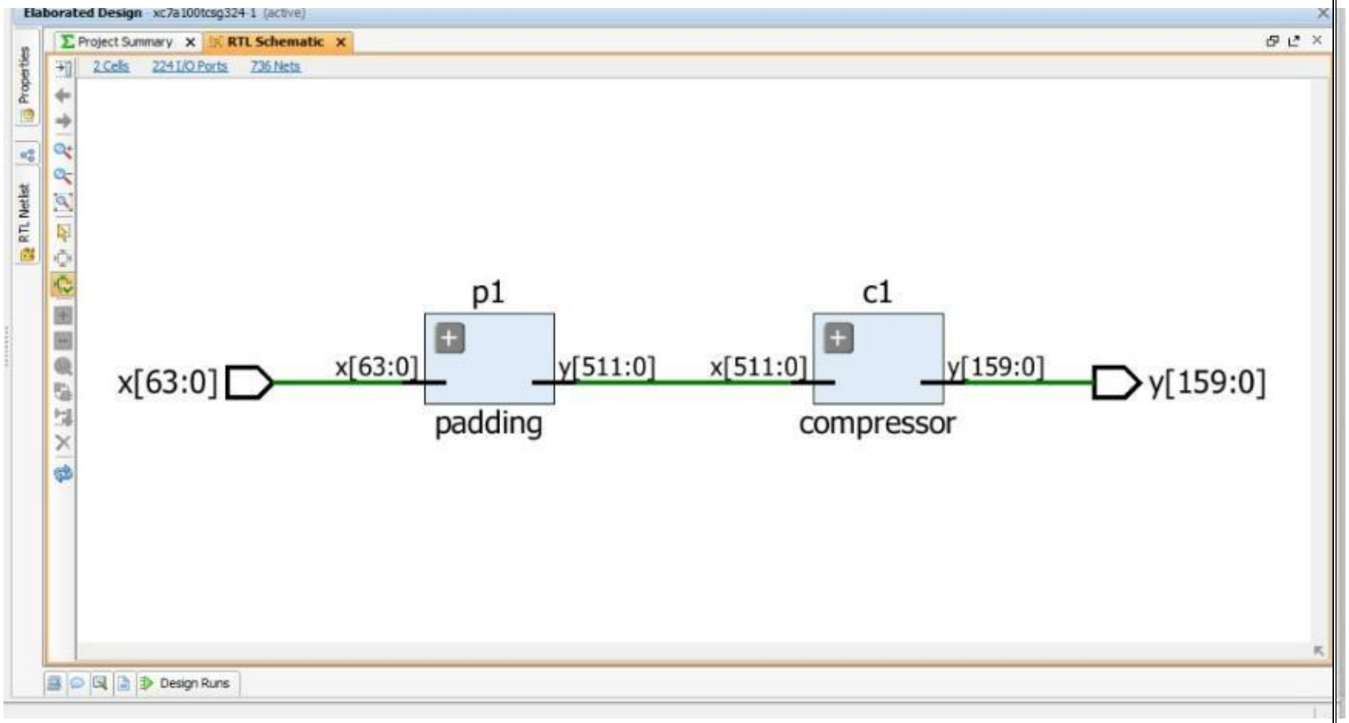
K_t = CA62C1D6 (60 <= t <= 79).

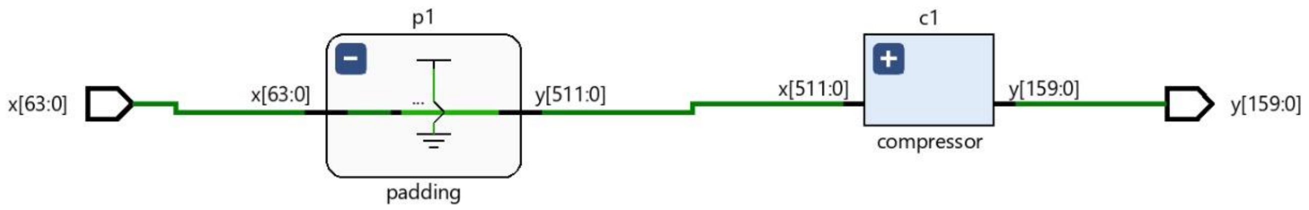
Output :

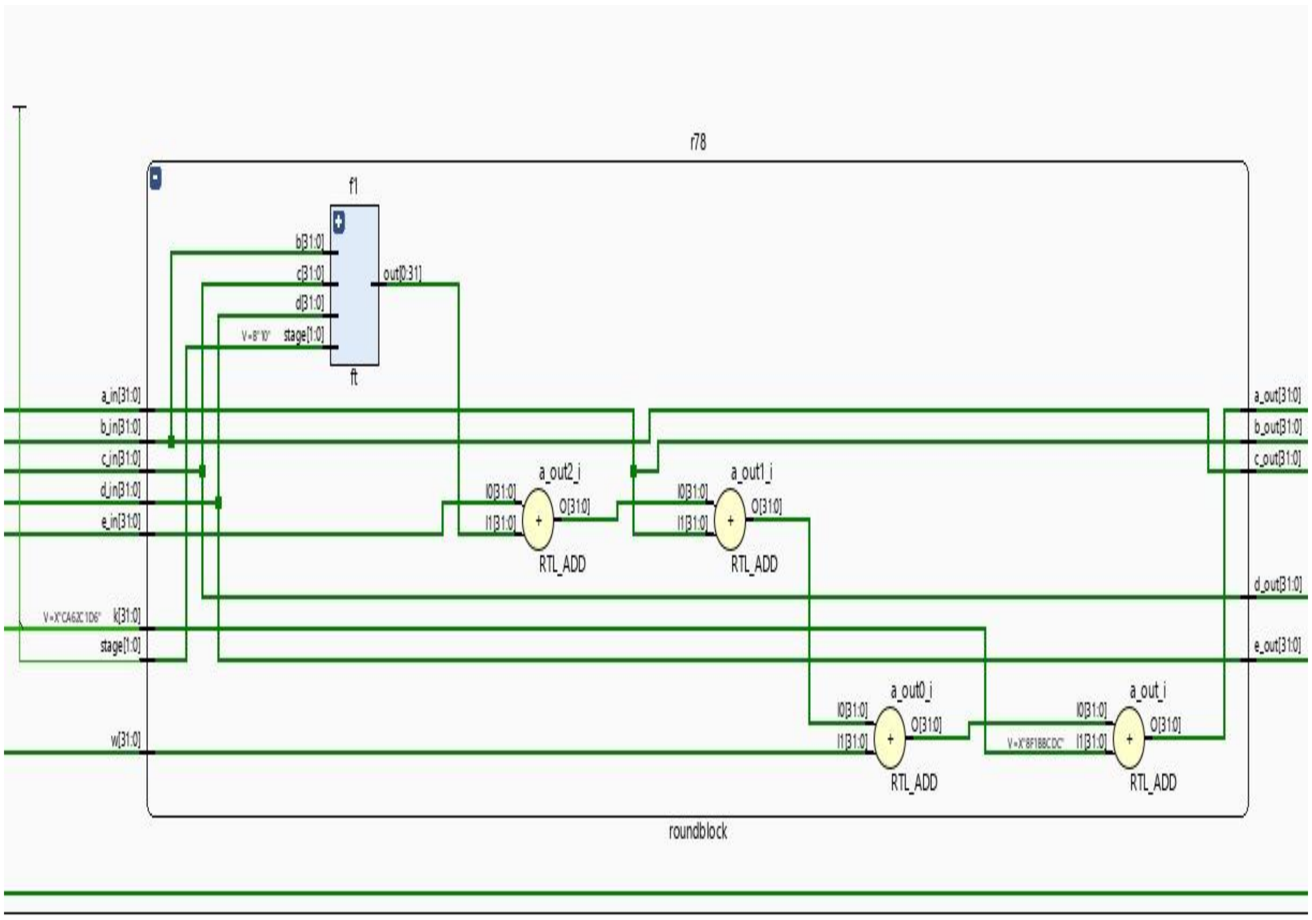
After processing the final 512-bit message block t (considering that the message is divided into t 512-bit blocks), and it can obtain a 160-bit message digest.

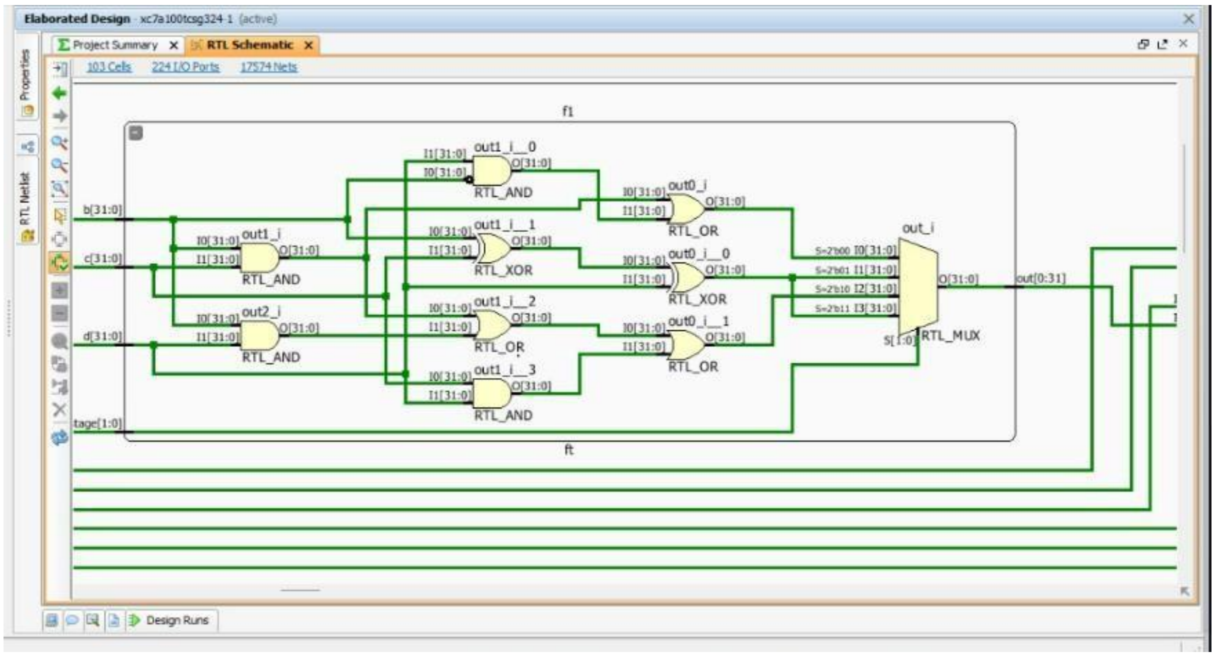
SIMULATIONS:

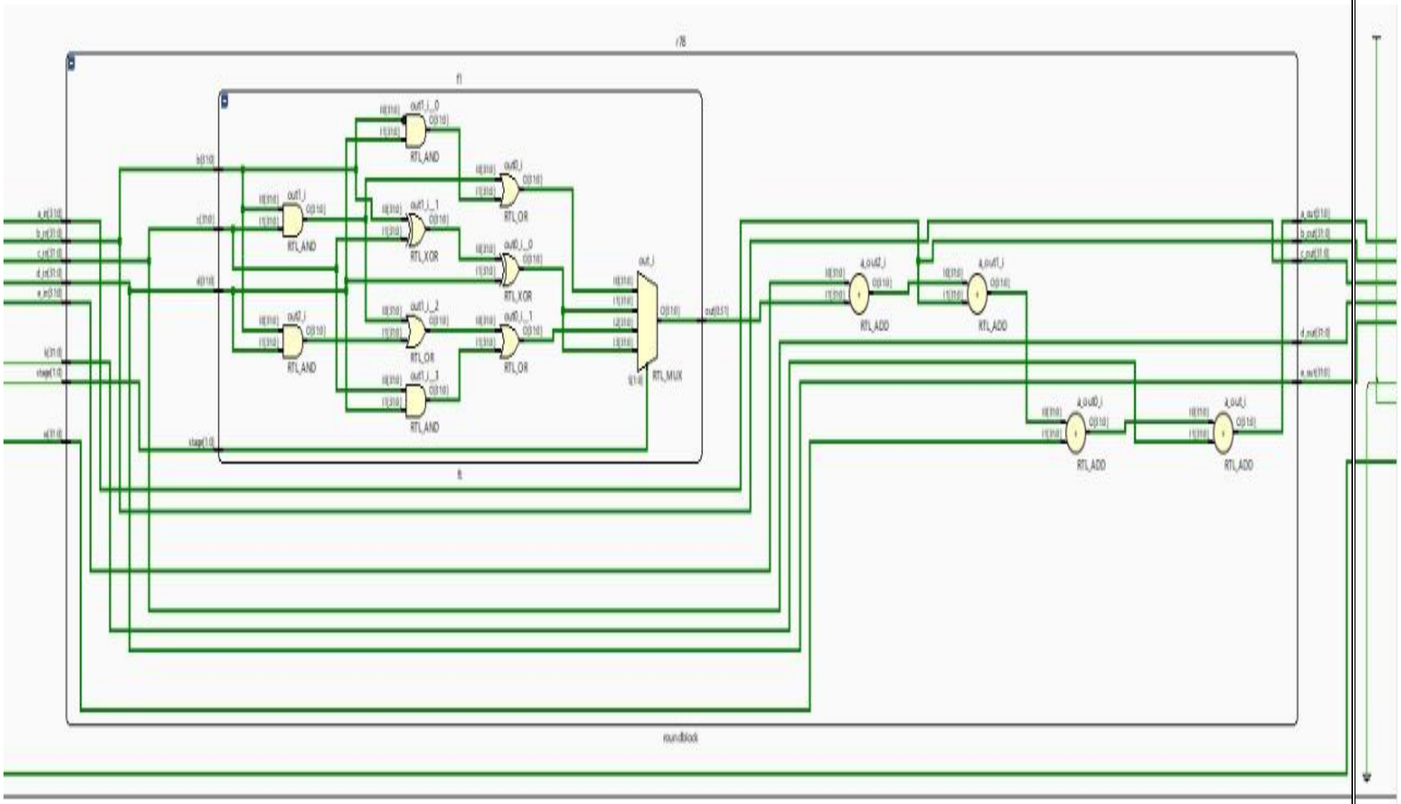
RTL SCHEMATIC



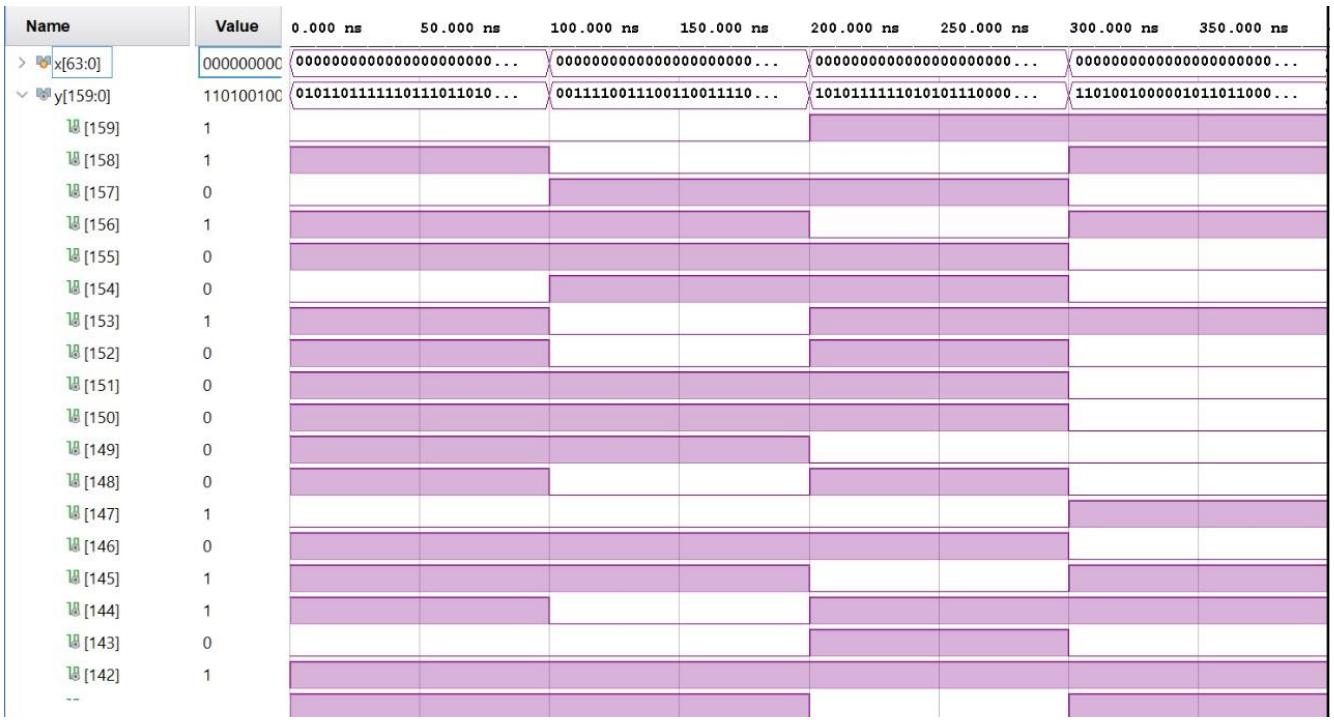



r78
f1
b[31:0]
c[31:0]
d[31:0]
V=B"10"
stage[1:0]
out[0:31]
ft
a_in[31:0]
b_in[31:0]
c_in[31:0]
d_in[31:0]
e_in[31:0]
V=X"CA62C1D6"
k[31:0]
stage[1:0]
w[31:0]
a_out2_i
I0[31:0]
I1[31:0]
O[31:0]
RTL_ADD
a_out1_i
I0[31:0]
I1[31:0]
O[31:0]
RTL_ADD
a_out0_i
I0[31:0]
I1[31:0]
O[31:0]
RTL_ADD
a_out_i
I0[31:0]
V=X"8F1BBCDC"
I1[31:0]
O[31:0]
RTL_ADD
a_out[31:0]
b_out[31:0]
c_out[31:0]
d_out[31:0]
e_out[31:0]
roundblock


Elaborated Design - xc7a100tcsg324-1 (active)
Project Summary
RTL Schematic
103 Cells
224 I/O Ports
17574 Nets
Properties
RTL Netlist
f1
b[31:0]
c[31:0]
d[31:0]
tage[1:0]
out1_i
RTL_AND
out2_i
RTL_AND
out1_i__0
RTL_AND
out1_i__1
RTL_XOR
out1_i__2
RTL_OR
out1_i__3
RTL_AND
out0_i
RTL_OR
out0_i__0
RTL_XOR
out0_i__1
RTL_OR
out_i
RTL_MUX
S=2'b00 I0[31:0]
S=2'b01 I1[31:0]
S=2'b10 I2[31:0]
S=2'b11 I3[31:0]
S[1:0]
O[31:0]
out[0:31]
ft
Design Runs


r78
f1
ft
RTL_AND
RTL_OR
RTL_XOR
RTL_MUX
RTL_ADD
roundblock

Output Waveform

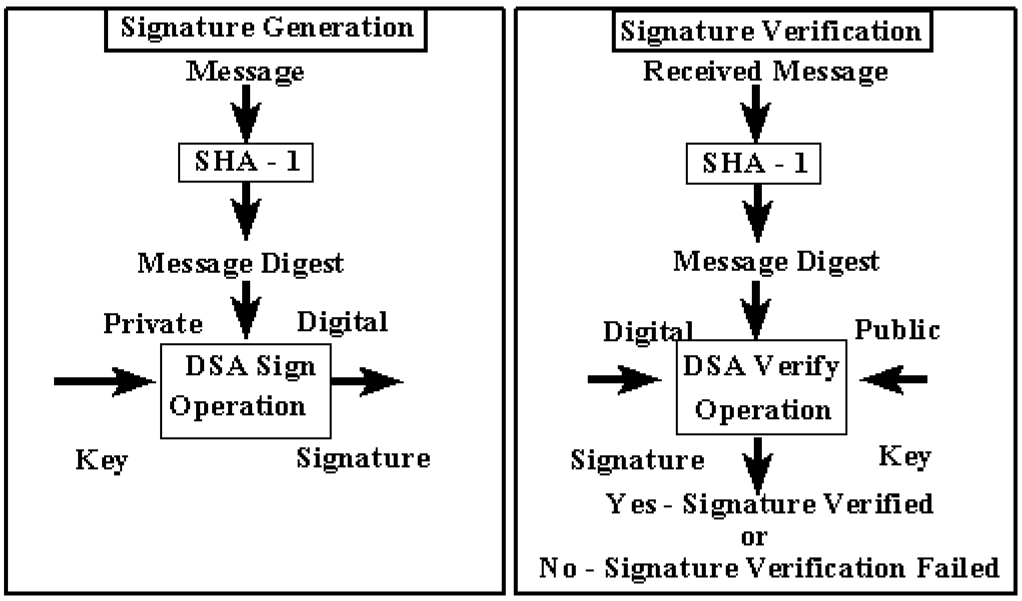


Result

SHA-1 REAL TIME USES:

One real-world example where SHA-1 may be used is when you're entering your password into a website's login page. Although it happens in the background without your knowledge, it may be the method a website uses to securely verify that your password is authentic.

If the website uses the SHA-1 cryptographic hash function, it means your password is turned into a checksum after you enter it in. That checksum is then compared with the checksum that's stored on the website that relates to your current password, whether you haven't changed your password since you signed up or if you just changed it moments ago. If the two match, you're granted access; if they don't, you're told the password is incorrect.



Another example where this hash function may be used is for file verification. Some websites will provide the SHA-1 checksum of the file on the download page so that when you download the file, you can check the checksum for yourself to ensure that the downloaded file is the same as the one you intended to download.

APPLICATIONS:

- Message Digest
- Password Verification
- Data Structures(Programming Languages)
- Compiler Operation
- Rabin-Karp Algorithm
- Linking File name and path together
- Game Boards
- Graphics

CONCLUSION:

The proposed design is verified using Xilinx Vivado 2021.1 tool on software by timing simulation. In this study, Verilog HDL is usedto design and capture SHA-1 in Xilinx Vivado software environment.

SHA or secured hash algorithm aims to provide an additional security level to the increasing and massive data you have to deal with. Hackers and attackers will keep finding a vulnerability in all the newer forms of hashing techniques being used.