CHAPTER 3

DESIGN AND DEVELOPMENT OF RECONFIGURABLE ARCHITECTURE FOR DATA INTEGRITY UNIT

3.1 INTRODUCTION

The commercial implementation of next generation wireless devices based on Software Defined Radio (SDR) technology have vital security problems in software downloading for reconfiguration. Downloading of all the relevant software is performed via a public channel and accordingly the security has become major issue in downloading. Similarly commercial end users will need the safety and integrity of their private information and transactions secure against unauthorized access or disruption.

Cryptographic hash function plays an important role in verifying the information that received is exactly the same as the information sent. They are employed in many applications for digital signatures, message authentication data integrity and key derivation. SHA 1 specifies Secure Hash Algorithm which generates condensed of message called message digest. Hash functions takes a message of variable length as input and produce a fixed length string as output referred to as hash code or simply hash of the input message. The hash functions are classified into keyed and unkeyed hash function (Bruce Schneier ,1996, William Stallings ,2003). The keyed hash functions are used in the Message Authentication Code (MAC) whose specification are dictates two distinct inputs a message and a secret key. The unkeyed hash functions have their categories hash function based on block
ciphers, modular arithmetic and customized hash function. The hash functions have one-way property; given \( n \) and an input \( M \), computing \( H(M) = n \), must be easy and given \( n \) it is hard to compute \( M \) such that \( H(M) = n \). The type of attacks\([1]\) are the collision attack (find two message \( M = M' \) with \( H(M) = H(M') \)), the preimage attacks (given a random value \( Y \), find a message \( M \) with \( H(M) = y \)) and the second preimage attack (given a message \( M \), find a message \( M - M' \) with \( H(M) = H(M') \)).

The SHA1 is required for use with the digital signature algorithm as specified in Digital Signature Standard (DSS) and whenever a secure hash algorithm is required. Both the transmitter and intended receiver of a message in computing and verifying a digital signature uses the SHA-1. It is necessary to ensure the security of digital signature algorithm, when a message of any length is input, the SHA produces \( m \) bits output called Message Digest. The MD is then used in the digital signature algorithm. Signing the MD using the private key rather than the message often improved efficiency of the process because the MD is usually much smaller than the message. The same MD should be obtained by the verifier using the user public key when the received version of the message is used as input to SHA.

In the recent years much progress has been made in the design of practical one-way hashing algorithm which is efficient for implementation by both hardware and software. Noteworthy work includes the MD family which consist of three algorithm MD2,MD4,MD5 (Bruce Schneier ,1996, William Stallings ,2003), the federal information processing standards for secure hash proposed by NIST\([5]\) for the past few years NIST designed the SHA-1 family which produce 160 bits, SHA-2 which produces 256 bits and 384 bits and SHA-3 which produces 512 bits of message digest. SHA-1 which produces message digest of 160 bits long was the best established of existing SHA hash functions and employed in several widely used security application and
protocols. Security flaws have been identified in SHA-1 (Xiaoyun, 2005), possible mathematical weakness might exist indicating that stronger hash function would be desirable.

Therefore the primary objectives of this chapter are:

- To propose a secure hash algorithm SHA-192 with stronger message digest than SHA-1.

- To design hardware architecture for verifying data integrity in SDR using proposed hash algorithm SHA-192 and MD5.

The aim of this research is to design a secure one-way hashing algorithm of 192 bit to enhance the security and resist to advanced attacks such as preimage, second preimage and collision attacks. Certain modifications are introduced in the existing SHA-1 algorithm to improve the strength of security. The maximum security depends on the length of message digest generated by the hash functions which is limited by the size of input to the algorithm. It also shows how the modification is done with satisfying the properties like compression, preimage resistance, and collision resistance. The simulation results show that proposed scheme provides better security than the existing one.

For the purpose of security in downloading software in SDR it is necessary to ensure information integrity of the data being transmitted. Hence in this research work an attempt is made to propose an efficient reconfigurable hardware architecture which supports data integrity for the secured downloading of software in reconfigurable receivers.

In recent past, quite a lot of research papers have been published on the hardware architecture of hash algorithms. It is found that these methods
are aiming to increase one or more features such as speed and throughput or reducing the hardware utilization and power consumption.

The performances of reported architecture are well appreciated, but most of the published architecture fails to determine the power consumption. Therefore attempts have been made to develop architecture with low power consumption and high throughput.

3.2 PROPOSED HASHING ALGORITHM – SHA-192

Information integrity ensures that information received (e.g. as part of a download) or stored at some earlier point has not been changed either as a result of transmission/storage media errors or intentional modification. For example, software to be downloaded to user terminals as part of a software upgrade might need to be stored in multiple locations in the network operator’s network so that it can be downloaded to all the affected terminals.

One method of providing this service is to perform a form of mathematical calculation using all of the information and then transmit that result along with the information. The calculation result would be encrypted or otherwise protected by a suitable authentication mechanism to ensure that the parametric result hasn’t also been altered. This method can also be used to verify that software already installed on a terminal hasn’t been modified or tampered with while the terminal was in a power down condition. Protection for such a parameter has to be tamper resistant. This SDR Security mechanisms need to be standardized on a global basis, if possible, since they are core security measures essential to SDR deployment.
The SHA-1 (NSA 1995) is based on design aspects and mathematical principles similar to the applied MD4 and MD5. SHA-1 produces a 160-bit message digest, longer than the generated 128-bit hash value by MD5. Some mathematical flaws have been identified (Rijmen and Oswald 2005 and Xiaoyun Wang et al 2005) in SHA-1 and cryptanalytic attack proved that by 2011 SHA-1 algorithm can be broken. Hence a new algorithm has been developed for data integrity Secure Hash Algorithm (SHA-192).

3.2.1 Algorithm Description

A “word” is a 32-bit quantity and a “byte” is a eight bit quantity. A sequence of bits can be interpreted in a natural manner as a sequence of bytes, where each consecutive group of eight bits is interpreted as a byte with high-order (most significant) bit of each byte listed first. Similarly a sequence of bytes can be interpreted as sequence of 32-bit words, where each consecutive group of four bytes is interpreted as a word with low-order (least-significant) byte given first.

Let the symbol “+” denote addition of words (i.e. modulo-2^32 additions). Let A<< s denote 32-bit obtained by circularly shifting A left by s bit position. Let not (A) denote the bit-wise complement of A and let A V B denotes bit-wise OR of A and B. Let A XOR B denotes bit-wise XOR of A and B, and let AB denote the bit –wise AND of A and B.

The proposed SHA-192 is another improved version in SHA -1 family. The compression diagram of SHA-192 is shown in Figure 3.1. It is used to hash message, M having a length of “l” bits, where 0<l<2^64. The algorithm uses six working variables of 32 bits each and generates final result of 192 bit message digest. The message digest is used during generation of a
signature for the message. The SHA-192 is also used to compute the message
digest for the received version of the message during the process of verifying
the signature. Any change to the message in transmission will result in a
different message digest, and the signature will fail to verify.

The SHA-192 is designed to have the following properties:

- It is computationally feasible to find a message which
  corresponds to a given message digest
- It is computationally feasible to find two different messages
  which produce the same message digest

3.2.2 Mathematical Model

The words of the message schedule are labeled $W_0$, $W_1$, $W_2$…$W_{79}$.
The six working variables are labeled $A$, $B$, $C$, $D$, $E$ and $F$. The words of the
hash value are labeled $H_0^{(i)}$,…,which will hold the initial hash value, and is
replaced by each successive intermediate hash value(after each message block
is processed)and ending with final hash value $H_{(N)}$.

![SHA-192 Compression Function](image)

**Figure 3.1 SHA-192 Compression Function**
3.2.3 Operations on Words

The following logical operators will be applied to words:

\[ A \lor B \] denotes bit-wise logical OR of A and B
\[ A \land B \] denotes bit-wise logical “and” of A and B
\[ \neg (A) \] denote the bit-wise logical complement of A
\[ A \oplus B \] denotes bit-wise logical exclusive OR of A and B,

3.2.4 Message Padding

The SHA-192 is used to compute message digest of length 192 for a message or a 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. If the number of bits in a message is a multiple of 8, for compactness it can be represented 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-192 sequentially processes block 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 number of “0” followed by 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 1, the length of the original message. The padded message is then processed by SHA-192 as an 512 bit blocks.

3.2.5 SHA-192 Hash Computation

A sequence of logical functions \( f_0, f_1, \ldots, f_{79} \) is used in the SHA-192. Each \( f_t \), \( 0 \leq t \leq 79 \), 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,
\[ f_t(B, C, D) = (B \text{ AND } C) \text{ OR } ((\text{NOT } B) \text{ AND } D) \quad \text{for } 0 \leq t \leq 19 \]
\[ f_t(B, C, D) = B \text{ XOR } C \text{ XOR } D \quad \text{for } 20 \leq t \leq 39 \]
\[ f_t(B, C, D) = (B \text{ AND } C) \text{ OR } (B \text{ AND } D) \text{ OR } (C \text{ AND } D) \quad \text{for } 40 \leq t \leq 59 \]
\[ f_t(B, C, D) = B \text{ XOR } (C \text{ XOR } D), \quad \text{for } 60 \leq t \leq 79 \]

A sequence of constant words \( K_0, K_1, \ldots, K_{79} \) is used in the SHA-192. These constants are defined in hexadecimal values as follows:

\[ K_t = 5A827999 \quad (0 \leq t \leq 19) \]
\[ K_t = 6ED9EBA1 \quad (20 \leq t \leq 39) \]
\[ K_t = 8F1BBCDC \quad (40 \leq t \leq 59) \]
\[ K_t = CA62C1D6 \quad (60 \leq t \leq 79) \]

### 3.2.6 Computing Message Digest

The message digest is computed using the final padded message. The computation uses two buffers each consisting of six 32-bit words and a sequence of eighty 32-bit words. The words of first 50-word buffer are labeled as A, B, C, D, E, F. The words of second buffer are labeled as \( H_0, H_1, H_2, H_3, H_4, H_5 \). The words of the 80-word sequence are labeled as \( W_0, W_1, \ldots, W_{15} \). The single word buffer TEMP is also employed to generate message digest, the 16-word blocks \( M_1, M_2, \ldots, M_n \) is processed. The processing of each \( M_i \) involves 80 steps. Before processing any blocks the \( \{ H_i \} \) are initialized as follows

\[ H_0 = 67452301 \]
\[ H_1 = EFCDB89 \]
\[ H_2 = 98BADCFE \]
\[ H_3 = 10325476 \]
\[ H_4 = \text{C3D2E1F0} \]
\[ H_5 = \text{F9B2D834} \]

Now, \( M_1, M_2 \ldots M_n \) are processed using following steps:

Step 1 : Divide \( M_i \) into 16 words \( W_0, W_1 \ldots W_{15} \), where \( W_0 \) is the left-most word.

Step 2 : For \( t = 0 \) to \( 15 \) \( W_t = M_i \)

For \( t = 16 \) to \( 79 \) let \( W_t = S_1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \).

Step 3 : Let \( A = H_0, B = H_1, C = H_2, D = H_3, E = H_4, F = H_5. \) \quad (3.1)

Step 4 : For \( t = 0 \) to \( 79 \) do

\[
\text{TEMP1} = S_5(A) + f_t(B,C,D) + E + W_t + K_t \] \quad (3.2)
\[
\text{TEMP2} = S_5(A) + A + f_t(B,C,D) + E + W_t + K_t+F \] \quad (3.3)
\[
E = D \] \quad (3.4)
\[
D = C \] \quad (3.5)
\[
C = S_30(B) \] \quad (3.6)
\[
B = S_{15}(A) \] \quad (3.7)
\[
F = \text{TEMP1} \]
\[
A = \text{TEMP2} \]

Step 5 : Let \( H_0 = H_0 + A \), \( H_1 = H_1 + B \), \( H_2 = H_2 + C \), \( H_3 = H_3 + D \), \( H_4 = H_4 + E \), \( H_5 = H_5 + F. \) \quad (3.8)
After processing $M_n$, the message digest is the 192-bit string represented by the 6 words $H_0 \ H_1 \ H_2 \ H_3 \ H_4$ and $H_5$.

### 3.2.7 Security Analysis

Good hash functions have a strong ability to withstand all kinds of cryptanalysis and attacks that try to break the system such as brute-force analysis, statistical analysis and collision attacks. Cryptanalysis of secure hash algorithm focuses on the internal structure of the iterated compression function and is based on attempts to find efficient techniques for producing collisions for a single execution of the compression function. The best way to find a collision pair is by using the birthday attack. To measure the security strength of the SHA with 192-bit results, finding preimage or second old preimage attacks requires about bit operations and for collisions attack. The preimage and second image of proposed algorithm occurs at different rate which is greater than both SHA-1. Standard attacks on various hash functions are tabulated in Table 3.1.

#### Table 3.1 Attacks on Standard Hash Function

<table>
<thead>
<tr>
<th>Hash Algorithm</th>
<th>Author</th>
<th>Type</th>
<th>Complexity</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA-0</td>
<td>Chabaud and Joux</td>
<td>Collision</td>
<td>$2^{61}$ (theory)</td>
<td>1998</td>
</tr>
<tr>
<td></td>
<td>Biham and Chen</td>
<td>Near-collision</td>
<td>$2^{40}$</td>
<td>2004</td>
</tr>
<tr>
<td></td>
<td>Biham et al</td>
<td>Collision</td>
<td>$2^{51}$</td>
<td>2005</td>
</tr>
<tr>
<td></td>
<td>Wang et al</td>
<td>Collision</td>
<td>$2^{39}$</td>
<td>2005</td>
</tr>
<tr>
<td>SHA-1</td>
<td>Biham et al</td>
<td>Collision</td>
<td>$2^{75}$</td>
<td>2005</td>
</tr>
<tr>
<td></td>
<td>Wang et al</td>
<td>Collision</td>
<td>$2^{63}$</td>
<td>2005</td>
</tr>
<tr>
<td>SHA-192 ( Developed )</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>
3.3 MD-5 ALGORITHMS

MD5 algorithm developed by Ronald Rivest (1992) accessed 512-bit message blocks and calculates a 128-bit message digest for an arbitrary \( l \)-bit message. It is an enhanced version of its predecessor MD4 but a more complex design. In MD5 architecture a fourth round has been added, while each transformation step has unique additive constants. Each step now adds the result of previous transformation step. This promotes a faster avalanche effect compared with MD4. Furthermore the order of the processed message sub-blocks is changed in transformation rounds 2 and 3. In spite of these differences both MD4 and MD5 produce a 128-bit message digest.

The algorithm could be described in two stages: Preprocessing and hash computation. Preprocessing involves padding a message, parsing the padded message into \( m \)-bit blocks, and setting initialization values to be used in hash computation. The final hash value generated by the hash computation is used to determine the message digest.

3.3.1 Algorithm Description

The algorithm takes as input message of arbitrary length and produces as output a 128-bit fingerprint or message digest of the input. It is conjectured that it is computationally infeasible to produce two messages having the same message digest or to produce any message having a given pre-specified target message digest.

Let a \( b \)-bit message is input for which message digest is to be generated. Here \( b \) is an arbitrary nonnegative integers; \( b \) may be zero, it need not to be multiples of eight and it may be arbitrarily large. Imagine the bits of message written down as follows:

\[ m_0, m_1 ... m_{b-1} \]
The following five steps are performed to compute the message digest of the message.

**Step1. Append Padding Bits:** The b-bit message is padded (extended) so that its length (in bits) is congruent to 448 modulo 512 (i.e.) the message is extended so that it is just 64 bits by being a multiple of 152 bits long. Padding is always performed, even if the length of the message is already congruent to 448, modulo 512. Padding is performed as follows: a single “1” bit is appended to the end of the message, and then 0 bits are appended until the length of the message becomes congruent to 448 modulo 512. In all, at least one bit and at a most 512 bits are appended.

**Step2. Append Length:** A 64-bit representation of b (the length of the message before padding bits were added) is appended to the result of the padding. In the likely event that is b is greater than $2^{64}$, then only the low-order 64 bits of b are used. These bits are appended as two 32-bits words and appended low-order word first in accordance with the previous conventions. At this point the resulting message (after padding with bits and with b) has length that is an exact multiple of 512 bits. Equivalently, this message has a length that is an exact multiple of 16(32-bits) words. This message is denoted here as Y.

**Step3. Initialize MD Buffer:** A four word buffer is used to compute the message digest. Let A, B, C, D be 32-bit registers. These registers are initialized to the following values in hexadecimal, low-order bytes first:

- Word A: 01234567
- Word B: 89abcdef
- Word C: fedcba98
- Word D: 76543210
**Step 4. Process Message in 16-Word Blocks:** This is the heart of the algorithm, which includes four rounds of processing. It is represented

![Diagram of MD5 Compression Function](image)

**Figure 3.2 Compression Function MD5**

in Figure 3.2 and its single step logical operation is given in Figure 3.3.

![Diagram of Single Step of MD5](image)

**Figure 3.3 Operation of Single Step of MD5**
The four rounds have similar structure but each uses different auxiliary functions F, G, H and I.

\[
F (B, C, D) = (B \text{ AND } C) \text{ OR } ((\neg B) \text{ AND } C)
\]
\[
G (B, C, D) = (B \text{ AND } D \text{ OR } (C \text{ AND } (\neg D))
\]
\[
H (B, C, D) = B \text{ XOR } C \text{ XOR } D
\]
\[
I (B, C, D) = C \text{ XOR } (B \text{ OR } (\neg D))
\]

Each round consists of 16 steps and each step uses a 64-element table \( T[1 \ldots 64] \) constructed from the sine function. Let \( T[i] \) denote the i-th element of the table, which is equal to the integer part of \( 2^{32} \) times \( \text{abs}(\sin(i)) \), where \( i \) is in radians. Each round also takes as input the current 512-bit block \( Y_q \) and the 128-bit chaining variable \( CV_q \). An array \( X \) of 32-bit words holds the current 512-bit \( Y \). For the first round the words are used in their original order. The following permutations of the words are defined for rounds 2 through 4:

\[
\rho_2 (i) = (1 + 5i) \text{ mod } 16
\]
\[
\rho_3 (i) = (5 + 39) \text{ mod } 16
\]
\[
\rho_4 (i) = 7i \text{ mod } 16
\]

The output of the fourth round is added to the input of the first round \( (CV, ) \) to produce \( CV_{q+1} \).

**Step5. Output:** After all \( L \) 512-bit blocks have been processed, the output from \( L^{\text{th}} \) stage is the 128-bit message digests.

The additions are modulo \( 2^{32} \). Four different circular shift amounts \( S \) is used each round and are different from round to round. Each step is of the following form,
A -> D
B -> B + ((A + F(B, C, D) + X[K1 + T[I]]) << s)
C -> B
D -> C

3.4 HARDWARE IMPLEMENTATION OF MD5

The developed architecture for MD5 is shown in Figure 3.4. The structure of MD5 algorithm allows both iterative and pipelined implementation, because the MD5 steps of equations in step 5 is performed 64 times. The padding and length appending operations are left outside of the design, because they are easy and fast to perform and hence hardware acceleration is not needed for those steps. The architecture shown in Figure 3.4 performs steps 3-5 of MD5 algorithm.

![Figure 3.4 Hardware Architecture of MD5](image)
The inputs to the architecture are as following: the message is given for the design with data-in, width of the data-in can be chosen freely. Processing of the algorithm is started with either start_new or continues signals. Start_new is used when a derivation of a new message digest is started i.e. when \( x_0 \) is processed, and continue is used when processing a new \( x_j \) for which \( j \geq 1 \), is started. For example if a 1024 bit message is processed start_new is used for the first 512 bits and when the second 512 bits are processed, continue is used. The counter in the architecture counts from 0 to 63 and it is reset to zero when new derivation begins. When a new derivation of a new message digest is started, A, B, C, D registers are initialized to the values given in step 3. Otherwise; values from previous derivation are used. the second multiplexer is used in iterative architecture where the same hash core is utilized several times in a processing of the algorithm. The adder performs the addition step 4. the hash core computes the step 2 values.

The heart of the architecture is the hash core block which calculates the process in step 2. The internal diagram of hash core block is given in Figure 3.5. This block is implemented so that it is performed in one clock cycle and it forms the critical paths of MD5 architecture in Figure 3.4. From step 2 onwards four 32-bit addition are required and also the following non linear functions are implemented.

\[
\begin{align*}
F (B, C, D) &= \text{B AND C OR } (\text{NOT B} \text{ AND C}) \\
G (B, C, D) &= \text{B AND D OR } (C \text{ AND } \text{NOT D}) \\
H (B, C, D) &= B \text{ XOR C XOR D} \\
I (B, C, D) &= C \text{ XOR } (B \text{ OR } \text{NOT D})
\end{align*}
\]

These function are implemented using simple bitwise logical operations. The shifting operation is the most logic resource demanding operation in the MD5. Also significant part of the delay of the block consists of the delay of the shifter.
3.5 UNIFIED ARCHITECTURE OF MD-5 AND SHA-192

The developed architecture for implementation of integrity unit in is shown in Figure 3.6. The heart of the integrity unit is hash core function, which implements both MD5 and SHA-192 hash functions; the hardware architecture for proposed SHA-192 is implemented with minor modifications in MD 5 logic. The proposed Integrity unit is reconfigurable in the sense that operates for two different modes: for MD5 and for SHA-192 security scheme. The selection of the operation mode is upon to the user needs it’s time and it is not pre defined by the integrity unit operation scenario. The developed architecture operates efficiently as an original MD5 and also for developed SHA-192 hash function.
3.5.1 Hash Function Core

The critical component of the integrity unit is the hash function core. The hash core operates for both MD5 and SHA-192 hash functions, alternatively. The padding data unit pads the input data and convert them to 512 bits blocks (padded data). This procedure is the same for both the hash functions. It is characterized of simplicity and it is well defined by both MD5 and sha-192. The purposed of padding is to make the total length of an input message a multiple of 512-bits. Both the hash function process blocks of 512-bits, when they compute the message digest. The padded data are generated according to following process: a logic “1”, followed by m”0” as followed by 64 bit integer are appended to the end of the input data, to produce a padded data block with length equal to 512*n. The 64-bit integer is equal to the length of the input data message. Every produced padded data
block is stored in registers. The appropriate data transformation is performed in the pipeline data transformation unit. The specified constants for the MD5 and SHA-192 are stored in constants units. The transformed data are finally modified by modulo adders unit and, in this way the message digest is produced.

The pipelined data transformation unit is the critical component for the data transformation of the hash core function. The architecture for this unit designed with pipelined technique. This consists of four different data transformation rounds. The pipelined applied design technique demands four registers, one for each data transformation round. Every round operates on 6 inputs (A,B,C,D,E,F) plus the message input m_i and the constant input K_i, all equal to 32-bits.

In the case of the MD5 operation the Data Transformation Unit uses only the four inputs/outputs B, C, D, E of each one of the four Data Transformation Rounds. The input/output named A, F is not used, for this hash function operation (MD5). This is due to the fact that MD5 processes on 128-bit blocks (4x32-bit) transformation blocks, instead of the 192-bit blocks that are used in SHA-192. The specified constants for are stored in constant unit. Four 32-bit constant (initial values have been defined. These values are called as chaining variables. In addition every one of the four data transformation rounds demands 16X32 bit constants. The four Data Transformation Rounds are similar, but its one performs a different operation. The integrated architecture for data transformation round is presented in detail.

Modulo addition (MA) components in the architecture indicate modulo addition $2^{32}$, while the shifters components define left shift rotations of the input data block. The Data Transformation Round i operation is based on a Nonlinear Function i transformation of the three of BIn, CIn, and DIn,
inputs. Then, this result is added to the fourth input EIn with the input data block and the constant. That result is rotated to the right and the rotated output data are added with the input DIn. There are four different nonlinear functions, one used for each Data Transformation Round, which perform the digital logic transformation according to following equations (3.9) to (3.12)

MD5 Round Block 1:

\[
F (B, C, D) = (B \text{ AND } C) \text{ OR } ((\text{NOT } B) \text{ AND } C ) \quad (3.9)
\]

MD5 Round Block 2:

\[
G (B, C, D) = (B \text{ AND } D \text{ OR } (C \text{ AND } (\text{NOT } D)) \quad (3.10)
\]

MD5 Round Block 3:

\[
H (B ,C , D) = B \text{ XOR } C \text{ XOR } D \quad (3.11)
\]

MD5 Round Block 4:

\[
I (B, C, D) = C \text{ XOR } ( B \text{ OR } (\text{NOT } D)) \quad (3.12)
\]

where B, C, D are 32-bit variables.

The Hash Function Core can be used alternatively for the operation SHA-192 hash function also. In this case, the Data Transformation Unit and the Data Transformation Rounds process the data in a different way, compared with MD5 operation mode, in order the Hash Function Core to perform efficiently as SHA-192.

\[
A = S_5(A) + A + f_t(B,C,D) + E + W_t + K_t + F; \quad (3.13)
\]
\[
B = S_{15}(A); \quad (3.14)
\]
\[
C = S_{30}(B); \quad (3.15)
\]
\[
D = C; \quad (3.16)
\]
\[
E = D; \quad (3.17)
\]
\[
F = TEMP1; \text{ TEMP1} = S_5(A) + f_t(B,C,D) + E + W_t + K_t; \quad (3.18)
\]
The non-linear function $F_i$ of each data transformation performs a different digital logic transformation this time. The next four equations define the non-linear functions operations in each transformation round:

**SHA-192 Round Block 1:**

$$F_1 (B, C, D) = (B \text{ AND } C) \text{ OR } ((\text{NOT } B) \text{ AND } C) \quad (3.19)$$

**SHA-192 Round Block 2:**

$$F_2 (B, C, D) = (B \text{ AND } D \text{ OR } (C \text{ AND } \text{NOT } D)) \quad (3.20)$$

**SHA-192 Round Block 3:**

$$F_3 (B, C, D) = B \text{ XOR } C \text{ XOR } D \quad (3.21)$$

**SHA-192 Round Block 4:**

$$F_4 (B, C, D) = C \text{ XOR } (B \text{ OR } \text{NOT } D) \quad (3.22)$$

where $B, C, D$ are 32-bit variables.

---

**Figure 3.7:** Data Transformation for Combined Hash Computation
In addition four 32-bit constants are used, one for each round of the pipelined data transformation unit. It has to be mentioned that both initial values and round constants have been specified by SHA-192 standard. The initial values are refreshed each time; a new message digest is produced. The data are transformed 20 times in each round, totally 80 times and finally 192-bit transformed data block is produced. The constant unit in the case of SHA-192 operation initializes the input of pipeline data transformation with six 32-bit different initial values compared with MD5 mode.

The design criteria of both pipeline data transformation and data round transformation unit have been derived as a combination of both MD5 and SHA-192. Both the hash function uses four basic iterations of the data transformation according to four equations (3.9) to (3.12) for MD5 and equations (3.19) to (3.22) for SHA-192. For each of the iteration loop, which executes the calculation of an equation for many times, a pipeline stages has been designed. So totally there are four stages figure. Each one of the equations for MD5 and one for SHA-192 is calculated by the data transformation round i, the non linear f functions i operates in two different modes for MD5 and SHA-192. Especially for NF1 operates in one common mode for both MD5 and SHA-192 since the equations (3.9) for MD5 is equal to equation (3.19) for SHA-192. NF2 calculates equation (3.10) for MD5 and equation (3.20) for SHA-192. NF3 function calculates equation (3.11) for MD5 and equation (3.21) for SHA-192. Finally NF4 calculates equation (3.12) for MD5 and Equation (3.22) for SHA-192.

The operation of each round is supported by 16 x32 constants which are stored registers. The pipeline data transformation round generates a data block equal to 128-bit for MD5 and 192-bit for SHA-192 operation
MODES respectively. The modulo adder unit consists of six modulo adders that perform modulo\(^2^{32}\) addition between input data and constant. In this way message digest is generated finally. Four data blocks are processed in MD5 and 128-bit message digest is generated. The six data blocks are processed by SHA-192 and generate output of 192-bit message digest. With the pipeline design architecture of hash core function, 128-bit message digest of MD5 is generated for every 16+1 clock cycles and a 192-bit message digest of SHA-192 is generated in SHA-192 for every 20+1 clock cycles. Thus the combined architecture of MD-5 and SHA-192 results in reduced hardware utilization compared to the individual implementation of MD-5 and SHA-192.

3.6 RESULTS AND DISCUSSION

The developed integrity unit hardware architecture is implemented by using Verilog. After generating verilog code, the synthesis, place and route has been done using Xilinx ISE 10.1i. The power analysis is done using Synopsys-Design vision tool.

3.6.1 Synthesis Output

The first stage of the synthesis is to analyze the generated code to check the compatibility for synthesizing. After analyzing the source code, the target device has been synthesized and the net list has been created. The device utilization summary for the target architecture of 2V4000BF957-6 has been provided in the Table 3.2.
Table 3.2  Device Utilization Summary of MD5, SHA-192 and Combined Architecture

<table>
<thead>
<tr>
<th>FPGA device: 2v4000bf957-6</th>
<th>Allocated area</th>
<th>Used/Available for MD5</th>
<th>Used/Available For SHA 192</th>
<th>Used/Available for Combined MD5 and SHA-192</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of I/Os</td>
<td>162/684</td>
<td>194/684</td>
<td>195/684</td>
<td></td>
</tr>
<tr>
<td>Number of Function Generators</td>
<td>724/46080</td>
<td>2349/46080</td>
<td>1275/46080</td>
<td></td>
</tr>
<tr>
<td>Number of CLB Slices</td>
<td>406/23040</td>
<td>1333/23040</td>
<td>757/23040</td>
<td></td>
</tr>
<tr>
<td>Number of D FlipFlops and Latches</td>
<td>298/46080</td>
<td>1257/46080</td>
<td>1033/46080</td>
<td></td>
</tr>
<tr>
<td>Frequency</td>
<td>57.36MHz</td>
<td>83.801 MHz</td>
<td>105.67MHz</td>
<td></td>
</tr>
<tr>
<td>Power Consumption</td>
<td>4.55mW</td>
<td>15.49 mW</td>
<td>7.092mW</td>
<td></td>
</tr>
<tr>
<td>Throughput</td>
<td>458Mbps</td>
<td>536Mbps</td>
<td>845Mbps (MD5)</td>
<td>676Mbps (SHA-192)</td>
</tr>
</tbody>
</table>

It is seen from the Table 3.2 that the developed model for data integrity utilizes 23% of bonded IOBs for MD5, 28% for SHA-192 and 28% for unified architecture. It indicates that minimum of 162 IOBs for MD5, 194 for SHA-192 and 195 IOBs for unified architecture out of available 684 IOBs. It is also be seen that developed architecture utilizes 1% of bonded CLBs for MD5, 3% for SHA-192 and 5% for unified architecture. It indicates that minimum of 406 CLBs for MD5, 1333 CLBs for SHA-192 and 757 CLBs for unified architecture out of available 23040 C Lbs. the device utilization is found that the selected device is well suited for implementing developed architecture.
The achieved operating frequency is equal to 57.36 MHz for MD5, 83.801 MHz for SHA-192 and 105.6 MHz for unified architecture. The comparative study shows that unified architecture utilize less area than individual structure. The power analysis is done using Synopsys-Design vision and found that unified architecture consumes less that individual architecture. From the above tabulation, it could be inferred that the device utilization of developed architecture is less in unified architecture compared with the individual implementation of MD-5 and SHA-192. The unified architecture of MD-5 and SHA-192 proved to consume less power and also efficient in computing the hash values.

### 3.6.2 Area Delay Product Analysis

The area delay product comparison of individual architectures and the unified architecture shown in Figure 3.8. Area delay product is obtained from the product of number of slices utilized and delay. From which it could be inferred that the developed combined architecture has less area delay product.

---

**Figure 3.8 Area Delay Product Comparisons**
3.6.3 Throughput Analysis

The efficiency of the system can be measured using throughput and analysis carried out for number of test vectors. Throughput is calculated using formulae,

\[
\text{Throughput} = \frac{\text{blocksize} \cdot \text{clockfrequency}}{\text{latency}}
\]  

(3.23)

where block size is the number of bits, clock frequency is operating frequency and latency is time taken to process one block. Throughput of developed architecture is tabulated in Table 3.3 and comparison is shown in Figure 3.9.

Table 3.3 Throughput Analysis MD5, SHA-192 and Combined Architecture

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>FPGA device : 2v4000bf957-6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>MD5</td>
</tr>
<tr>
<td>Throughput</td>
<td>458Mbps</td>
</tr>
</tbody>
</table>

Figure 3.9 Throughput Comparisons of Proposed Design
From the analysis it is found that throughput of the unified architecture is also comparatively high.

The technology level schematic is found to have higher complexity and it is shown in Figure 3.10.

![Figure 3.10 RTL Schematic of Combined Architecture](image)

3.7 OUTCOME OF PLACING AND ROUTING

The synthesized net list describes the interconnection of blocks, the logic cell within the blocks and logic cell connections. The net list is fed for floor planning before placing and routing process. The floor planning process that maps the logical description with physical description. The main goal of floor planning is to arrange the blocks on a selected chip, decide the location of I/O pads and clock distribution. The schematic diagram of developed
architecture has been shown in Figure 3.11. The floor planning for the developed architecture has been generated and top view after same is illustrated in Figure 3.12.

Figure 3.11 Schematic Block Diagram of Integrity Unit

Figure 3.12 View of Integrity Unit after Routing
It is observed from the Figure 3.11 all the circuit elements are placed properly on the FPGA for the developed data integrity unit. After floor planning, the logic cells within the flexible blocks have been placed and the necessary connection has been made by routing chip. The component organization for the developed scheme after routing is shown in Figure 3.10. The input output port and other related blocks as well as logic cells are interconnected using global routing method.

3.8 POWER CONSUMPTION

The requirement of portability in emerging reconfigurable receivers like software radio, mobile devices and other handheld devices places severe restrictions on power consumption. In order to satisfy the requirements of such wireless devices special care has been taken in design of proposed architecture. The low power design technique architecture is applied in both system and architecture level. Energy consumption in hardware design is proportional to the active capacitance. By minimizing the total capacitance in the design power consumption can be reduced. In the developed architecture use of on chip resources reduces significantly the active capacitance. The power required for execution of data integrity unit using Target device has been computed and listed in Table 3.4 for individual components. The total dynamic power has been calculated using the relation,

\[
P = \sum_{i=1}^{n} P_{ci} + \sum_{j=1}^{k} P_{ioj} + P_{clk}
\]  

(3.24)

where \( P \) is total consumed power, \( n \) is number of CLBs utilized, \( P_{ci} \) is power of \( i^{th} \) component, \( k \) is number IOs utilized, \( P_{ioj} \) is the power of \( j^{th} \) IO pad and \( P_{clk} \) is clock power.
Table 3.4 Power Consumed by Target Device

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>MD5</th>
<th>SHA-192</th>
<th>Unified Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total quiescent power</td>
<td>2.75</td>
<td>10.81</td>
<td>5.07</td>
</tr>
<tr>
<td>Total dynamic power</td>
<td>1.8</td>
<td>4.68</td>
<td>2.02</td>
</tr>
<tr>
<td>Total Power</td>
<td>4.55mW</td>
<td>15.49mW</td>
<td>7.092mW</td>
</tr>
</tbody>
</table>

The low power design technique reduces total power consumption of developed architecture. The power consumption of proposed design is analyzed by using synopsis tool. The total power of the whole design is 7.09 mw this shows that the developed unified architecture consumes less power than the individual architecture.

3.9 VERIFICATION AND VALIDATION OF THE RECONFIGURABLE HARDWARE

3.9.1 Verification of the Hardware

The implemented reconfigurable data integrity architecture has been verified and validated for different inputs. For example the generated model has been verified for given input and the output is shown in Figures 3.13 and 3.14 respectively. The simulation result generates 128 bit output for MD5 algorithm and 192 bit message digest for SHA -192 algorithms. The output is tested for both individual architecture and unified architecture.
The verification and validation of the developed model has been performed for different inputs and genuiness of the implemented hardware has been analyzed.
3.9.2 Testing of Reconfigurable Hardware

At the initial condition, the state of the FPGA has been maintained in predetermined position and whenever the hashing technique is changed in received signal, the FPGA is reconfigured by downloading the appropriate bit stream file. Also observed that if the hashing technique changes from one scheme to another, it has been measured that the time required to respond the developed hardware is 70ns. Various bit files have been generated for different input and different hashing schemes MD5 and SHA192. It is also ensured that the reconfiguration takes place only at the end of computation of message digest to avoid the misfiring of the hardware.

3.10 PERFORMANCE COMPARISON

The developed architecture is simulated in Xilinx devices and their performances are evaluated. The synthesis result shows that the proposed design performs better than the conventional architectures available previously. Hardware utility of proposed architecture in Virtex 2v4000 is found to be less when compared with other devices. Similarly uses less number of bonded IOs and also less number of slices. Efficiency of the developed architecture has been found to be better with others. The performance of developed architecture is compared with existing architectures and the analysis shows that developed architecture performs better by obtaining high throughput than the architectures developed by Deepakumara et al 2001, Dominikus et al 2002, Kimmo Järvinen 2005 and Khan et al 2007). The comparison with previously published implementation of the same hash functions are presented in the Table 3.5.
### Table 3.5 Performance Comparison of Proposed Unified Architecture with Previous Work

<table>
<thead>
<tr>
<th>Device and Architecture</th>
<th>CLB Slices</th>
<th>Area Utilization</th>
<th>Operating Frequency in MHz</th>
<th>Throughput in Mbps</th>
<th>Power Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>V100FG680 Deepakumara et al (2001)</td>
<td>880/12288 (iterative) 4763/12288 (Full loop Unrolling)</td>
<td>7.1% 38.8%</td>
<td>21 71.4</td>
<td>165 (MD5) 354 (MD5)</td>
<td>NA</td>
</tr>
<tr>
<td>VIRTEX-E XCV300E Dominikus et al 2002</td>
<td>1004/1536</td>
<td>65%</td>
<td>42.9</td>
<td>107 (MD5) 86 (SHA-1)</td>
<td>NA</td>
</tr>
<tr>
<td>V150BG352 (MD5 and SHA-1) (Nicolas Sklavos et al 2003)</td>
<td>1518 / 1728</td>
<td>87.85%</td>
<td>71</td>
<td>1.7 Gbps</td>
<td>32mW</td>
</tr>
<tr>
<td>VIRTEX-II XC2V4000-6 (Kimmo Järvinen 2005)</td>
<td>7997/23040</td>
<td>34.7%</td>
<td>93.4</td>
<td>725 (MD5)</td>
<td>NA</td>
</tr>
<tr>
<td>DESIGN COMPILER(0.18µM) Jun Han et al 2007</td>
<td>9kgates</td>
<td>-</td>
<td>104MHz</td>
<td>16 (SHA-1) 28 (MD5)</td>
<td>NA</td>
</tr>
<tr>
<td>VIRTEX-II XC2V4000-6 Khan et al 2007</td>
<td>14911/23040</td>
<td>64.71%</td>
<td>43.47</td>
<td>171.2 (MD5) 137.4 (SHA-1)</td>
<td>NA</td>
</tr>
<tr>
<td>XC2V4000BF957-6 Proposed Design</td>
<td>757/23040</td>
<td>3.23%</td>
<td>105.67</td>
<td>845 (MD5) 676 (SHA-192)</td>
<td>7.09mW</td>
</tr>
</tbody>
</table>

### 3.11 SUMMARY

This chapter proposed a secure hash algorithm SHA-192 to improve the strength of existing hash algorithms SHA-1. The performances of proposed algorithm are evaluated with different data sets. It is proved that the proposed algorithm outperforms the other variants in quality metrics at higher security.

The reconfigurable hardware architecture was also developed FPGA. The implementation of proposed SHA-192 and MD5 algorithm the
can be utilized for data integrity unit for software defined radio receiver. The developed has been implemented using verilog and implemented using Xilinx virtex device. The implemented hardware has been verified and validated for different hash algorithm MD5 and SHA-192 and the results show that the module can works efficiently for both the algorithms and achieves high throughput. The proposed architecture operation is mainly based in two reconfigurable designed units. With this applied technique the allocated area resources have been minimized by a great factor, compared with other conventional implementations. The developed architecture is found to be better in area utilization, power consumption and operates at high speed. The Reconfigurable Integrity Unit has better performance than other conventional architectures for both MD5 and SHA-192 operation modes.