Hardware Implementation of
The Message Digest Procedure MDP-384

Rabie Mahmoud1, Magdy Saeb2

1. Department of Mathematics, Faculty of Science, Cairo University, Cairo, Egypt.
2. Computer Engineering Department, Arab Academy for Science, Tech. & Maritime Transport (AAST), Alexandria, Egypt. mail@magdysaeb.net

ABSTRACT: The message digest procedure MDP-384 is a cryptographic hash function improvement of the hash function MDP-192 which is based on the principles provided by Markel’s work, Rivest MD-5, SHA-1 and RIPEMD. MDP-384 accepts a variable-size input message and returns a fixed-size string as 384-bit long hash that utilizes twelve variables for the round function with cascaded XOR operations and deliberate asymmetry in the design structure to provide higher security with negligible increase in execution time and memory requirement. In this work, we provide a Field Programmable Gate Array (FPGA) hardware implementation of this hash function.

Keywords: FPGA, Hash function, MDP-384, Cryptography, Hardware.

1. Introduction

The message digest procedure MDP-384 is a one-way function having the attributes of cryptographic hash function and can be applied in many applications of computer communication security as message tampering detection, message authentication codes, digital signatures, user authentication when used with a secret key, code recognition for protecting original codes, malware identification, commitment schemes, key update and derivation, cryptographic primitive for block and stream ciphers, detection of random errors, and finally random number generation which is used in stone metamorphic cipher. The details of the stone metamorphic cipher and its FPGA implementation can be found in [1] and [2] respectively. MDP-384 is modified and is based on the principles of the hash function MDP-192 where twelve variables instead of six variables in MDP-192 are used for the round function. The principles similar to those used by SHA-1 of the Secure Hash Standard (SHS) of the US Federal Information Processing Standard Publications (FIPS PUB 180-3) [3], [4], [5], the design objectives of MD-2, MD-4, and MD-5 [6], [5], [7] developed by Ron Rivest, RIPEMD-160 [8], and Merkle, in his dissertation [9] are adopted in MDP-192 and MDP-384.

The main improvements which are included in MDP-384 to be more secure than MDP-192 are:

● The increased size of the hash; that is 384 bits compared to 192, 128 and 160 bits for the MDP-192, MD-5 and SHA-1 schemes respectively. The security bits have been increased from 64, 80 and 96 to 192 bits in MDP-384.
● The other improvement than MDP-192 is based on processing the message blocks employing twelve variables rather than six variables where this contributes to better security and faster avalanche effect.

Also the hash function MDP-384 inherits all improvements of MDP-192 which are:

● The message block size is increased to 1024 bits providing faster execution times.
● The message words in the different rounds are not only permuted but computed by XOR and addition with the previous message words. This renders it harder for local changes to be confined to a few bits. In other words, individual message bits influence the computations at a large number of places. This, in turn, provides faster avalanche effect and added security.
● Moreover, adding two nonlinear functions and one of the variables to compute another variable, not only eliminates the possibility of certain attacks but also provides faster data diffusion.
● The XOR and addition operations do not cause appreciable execution delays for today’s processors. Nevertheless, the number of rotation operations, in each branch, has been optimized to provide fast avalanche with minimum overall execution delays [10]. In the following sections, we provide the conventional method of using a hash as message authentication code, the formal description of MDP-384 algorithm, the details of our circuit design, discussion of the results of the FPGA
2. Description of The Procedure

The procedure can be summarized as follows:

1. Read a file as binary file, we call it the message (m).
2. Divide this file into a number of 1024-bit blocks ($M_0, M_1, \ldots, M_n$).
3. Last block will, in general, need padding to complete it to 1024-bit block where the padding can be achieved by adding a “1” followed by as many as needed “0” then a 64-bit integer representing the original length of the message.
4. Use the algorithm of MDP-192 for all $k = 0, 1, 2, \ldots, n$.
5. Append hash to original message, encrypt and send to destination.

3. The Algorithm

The formal description of the MDP-192 algorithm is shown in [10], and we provide a summary of message digest procedure MDP-384 as follows:

The symbols in the algorithm which illustrates different actions during a process are:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Mnemonic</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ll m$</td>
<td>ROTL m</td>
<td>Rotate to the left m times</td>
</tr>
<tr>
<td>+</td>
<td>ADD</td>
<td>Addition</td>
</tr>
<tr>
<td>$\oplus$</td>
<td>XOR</td>
<td>Bitwise XOR</td>
</tr>
<tr>
<td>$\leftarrow$</td>
<td>ASG</td>
<td>Assigned to</td>
</tr>
<tr>
<td>$\land$</td>
<td>AND</td>
<td>Bitwise AND</td>
</tr>
<tr>
<td>$\lor$</td>
<td>OR</td>
<td>Bitwise OR</td>
</tr>
<tr>
<td>$\neg$</td>
<td>INV</td>
<td>Complement</td>
</tr>
</tbody>
</table>

Algorithm: Message Digest Procedure MDP-384

INPUT: A given set of 1024-bit blocks ($M_0, M_1, \ldots, M_n$) where each block is 32 32-bit words; this set of blocks represents the message to be hashed.

OUTPUT: A 384-bit hash function that is representing the original message.

Begin

Repeat Begin
  [For all $M_k$ for $k=1, 2, \ldots, n$]
  [Within each block $M_k$, process each word $W_i$ as follows:]
  for $i = 0$ to $383$
    [That is the reason we need to expand $W_i$ from 32 values to 384 since each 1024-bit message $M_i$ is only 32 32-bit words]
    \[
    A_{i+1} \leftarrow (P_i \ll m_i) + \Phi_i (P_i, Q_i, R_i) + \Phi_i (R_i, S_i, T_i) + U_i + W_i + K_i;
    \]
    \[
    B_{i+1} \leftarrow T_i \ll m_5;
    \]
    \[
    C_{i+1} \leftarrow P_i \ll m_1;
    \]
    \[
    D_{i+1} \leftarrow Q_i \ll m_2 \oplus R_i \ll m_1;
    \]
    \[
    E_{i+1} \leftarrow R_i \ll m_3 \oplus Q_i \ll m_2;
    \]
    \[
    F_{i+1} \leftarrow S_i \ll m_4 \oplus R_i \ll m_3;
    \]
    \[
    G_{i+1} \leftarrow (A_i \ll m_5) + \Phi_i (A_i, B_i, C_i) + \Phi_i (C_i, D_i, E_i) + F_i + W_i + K_i;
    \]
    \[
    Q_{i+1} \leftarrow E_i \ll m_5;
    \]
    \[
    R_{i+1} \leftarrow A_i \ll m_1;
    \]
    \[
    S_{i+1} \leftarrow B_i \ll m_2 \oplus A_i \ll m_1;
    \]
    \[
    T_{i+1} \leftarrow C_i \ll m_3 \oplus B_i \ll m_2;
    \]
    \[
    U_{i+1} \leftarrow D_i \ll m_4 \oplus C_i \ll m_3;
    \]
end;
  \{The number of rotations for each branch $m_i$ is optimized for fast avalanche effect; taken equal to 00100 for each side\}
Repeat this iteration loop until end-of-message;
  \{That is Repeat for all blocks $M_k$ for $k=1, 2, \ldots, n$, until end-of-message. After processing the message, the message digest is computed by concatenating the final values of the twelve variables: $A_f, B_f, C_f, D_f, E_f, F_f, G_f, Q_f, R_f, S_f, T_f, U_f$. This is a 384-bit message digest where the final values of each variable are computed as follows: \}
Repeat End;

End.

The functions $\Phi_i (X, Y, Z)$:

$\Phi_i (X, Y, Z) = X \oplus (-Y) \oplus Z$

for $i = 0$ to $31$;

$\Phi_i (X, Y, Z) = \Phi_i ((-X) \land Z) \lor ((Y \land (-Z)) \lor (-X))$

for $i = 32$ to $63$;

$\Phi_i (X, Y, Z) = X \ll m_i$;

for $i = 64$ to $95$;

$\Phi_i (X, Y, Z) = X \ll m_i$;

for $i = 96$ to $127$;

$\Phi_i (X, Y, Z) = X \ll m_i$;

for $i = 128$ to $159$;

$\Phi_i (X, Y, Z) = X \ll m_i$;

for $i = 160$ to $191$;

$\Phi_i (X, Y, Z) = X \ll m_i$;

for $i = 192$ to $223$;
The block function 

\[ \Phi_i(X, Y, Z) = X \oplus Y \oplus Z \]

for \( i = 128 \) to 159;

for \( i = 320 \) to 351;

\[ \text{for } i = 160 \text{ to } 191; \]

and

\[ \text{for } i = 352 \text{ to } 383. \]

The values of \( W_i \):

\[ W_i \leftarrow W_i \quad \text{for } i = 0 \text{ to } 31; \]

\[ \text{for } i = 32 \text{ to } 383. \]

Also, this algorithm is summarized in Figure 1.

![Message Digest Procedure MDP-1](image)

**Figure 1.** Operation of MDP-384 hash function

The constants \( K_i \):

\[ K_i \leftarrow (6071498F)_{16} \quad \text{for } i = 0 \text{ to } 31; \]

\[ K_i \leftarrow (A205B064)_{16} \quad \text{for } i = 32 \text{ to } 63; \]

\[ K_i \leftarrow (BB40E64E)_{16} \quad \text{for } i = 64 \text{ to } 95; \]

\[ K_i \leftarrow (4E1560F1)_{16} \quad \text{for } i = 96 \text{ to } 127; \]

\[ K_i \leftarrow (36C2F808)_{16} \quad \text{for } i = 128 \text{ to } 159; \]

\[ K_i \leftarrow (EFC23920)_{16} \quad \text{for } i = 160 \text{ to } 191; \]

\[ K_i \leftarrow (60714990)_{16} \quad \text{for } i = 192 \text{ to } 223; \]

\[ K_i \leftarrow (A205B065)_{16} \quad \text{for } i = 224 \text{ to } 255; \]

\[ K_i \leftarrow (BB40E64F)_{16} \quad \text{for } i = 256 \text{ to } 287; \]

\[ K_i \leftarrow (4E1560F2)_{16} \quad \text{for } i = 288 \text{ to } 319; \]

\[ K_i \leftarrow (36C2F809)_{16} \quad \text{for } i = 320 \text{ to } 351; \]

\[ K_i \leftarrow (EFC23921)_{16} \quad \text{for } i = 352 \text{ to } 383. \]

To initialize the iteration process, we used the following randomly chosen Initialization Values (IV) based on some natural unrelated constants, rather than a mathematical function:

\[ A_0 \leftarrow (5F7F45CC)_{16} \quad \text{Based on Electron Charge}; \]

\[ B_0 \leftarrow (364BD04C)_{16} \quad \text{Based on Electron Mass}; \]

\[ C_0 \leftarrow (23E50E70)_{16} \quad \text{Based on Avogadro’s number}; \]

\[ D_0 \leftarrow (4C081C80)_{16} \quad \text{Based on Earth’s Diameter}; \]

\[ E_0 \leftarrow (239BE7E9)_{16} \quad \text{Based on Earth’s Mass}; \]

\[ F_0 \leftarrow (14B7F480)_{16} \quad \text{Based on Moon’s Diameter}; \]

\[ G_0 \leftarrow (11DE784A)_{16} \quad \text{Based on Speed of Light}; \]

\[ H_0 \leftarrow (22679CB1)_{16} \quad \text{Based on Euler’s Constant}; \]

\[ I_0 \leftarrow (39827EB7)_{16} \quad \text{Based on Faraday’s Constant}; \]

\[ J_0 \leftarrow (63B22E45)_{16} \quad \text{Based on Proton’s Mass}; \]

\[ K_0 \leftarrow (63D55C6B)_{16} \quad \text{Based on Neutron’s Mass}; \]

\[ L_0 \leftarrow (277E9430)_{16} \quad \text{Based on Planck’s Constant}. \]

### 4. FPGA Implementation

The lucidity of the message digest procedure MDP-384 lead to a relatively easy-to-design FPGA-based implementation. We have implemented the MDP-384 using VHDL hardware description language [11], [12], [13] and Quartus II 9.1 Service Pack 2 Web Edition [14] and utilizing Altera design environment. The hash function implementation is performed by dividing the procedure into three parts in order to use three FPGA devices or chips connected sequentially on the board. The first chip (chip1) is responsible for padding the 256-bit input to 1024-bit block. Subsequently, The block is divided into 32-bit words \( W_i \); starting from the twelve initial values \( A_0, B_0, C_0, D_0, E_0, F_0, G_0, H_0, I_0, J_0, K_0, L_0 \) to produce \( A_{127}, B_{127}, C_{127}, D_{127}, E_{127}, F_{127}, G_{127}, H_{127}, I_{127}, J_{127}, K_{127}, L_{127} \). The third chip (chip3) is used to compute the values \( A_{255}, B_{255}, C_{255}, D_{255}, E_{255}, F_{255}, G_{255}, H_{255}, I_{255}, J_{255}, K_{255}, L_{255} \) and then applying the last step in the algorithm by XORing the initial values with resulting values and concatenating the result values together to have a 384-bit hash block as output. The schematics for demonstrative parts of MDP-384 algorithm are shown in Figures 2, 3, and 4. The design was implemented using three EP2C70F896C6, Cyclone II family devices. Based on this device characteristic, the worst case pin-to-pin delay in chip1 was found to be equal to 1089.363 ns. The worst case pin-to-pin delay in chip2 was found to be equal to 1159.366 ns. The worst case pin-to-pin delay in
chip3 was found to be equal to 1203.756 ns. A series of screen-captures of the different implementation results are shown in Figures 5 to 16. For example, Figures 5, 6, and 7 provide indication of successful compilation for chip1, chip2 and chip3 of MDP-384 respectively. In addition, parts of RTL for chip1, chip2 and chip3 of MDP-384 are shown in Figures 8, 9, and 10 respectively. Figure 11 displays chip1 simulation result showing the output $A_{127}$, $B_{127}$, $C_{127}$, $D_{127}$, $E_{127}$, $F_{127}$, $P_{127}$, $Q_{127}$, $R_{127}$, $S_{127}$, $T_{127}$, $U_{127}$ values bits. Figure 12 displays chip2 simulation showing the output $A_{255}$, $B_{255}$, $C_{255}$, $D_{255}$, $E_{255}$, $F_{255}$, $P_{255}$, $Q_{255}$, $R_{255}$, $S_{255}$, $T_{255}$, $U_{255}$ bit values. Figure 13 displays the chip3 simulation showing the output of the hash function procedure which is highlighted in this screen capture. Figures 14, 15 and 16 demonstrate the floor plan for chip1, chip2 and chip3 respectively. The details of the analysis and synthesis report, and the implementation delays in Balanced, Area and Speed optimization techniques are shown in appendixes A, B and C for chip1, chip2 and chip3 respectively. Figure 17 shows a comparison chart between various implementation delays.
Figure 7. Compiler tool screen showing correct implementation of chip3 of MDP-384

Figure 8. RTL screen for part of chip1 of MDP-384 implementation

Figure 9. RTL screen for part of chip2 of MDP-384 implementation

Figure 10. RTL screen for part of chip3 of MDP-384 implementation

Figure 11. Simulator screen showing the outputs of chip1 of MDP-384

Figure 12. Simulator screen showing the outputs of chip2 of MDP-384
Summary & Conclusion

We have furnished a brief discussion of the hardware implementation of the Message Digest Procedure MDP-384. Various modules, using three Altera Cyclone II family devices, are connected sequentially to allow for the FPGA implementation relatively high gate count. The resulting circuit provides a proof-of-concept FPGA implementation. It was shown that the worst case pin-to-pin delay in chip1 is equal to 1089.363 ns. The worst case pin-to-pin delay in chip2 is equal to 1159.366 ns. The worst case pin-to-pin delay in chip3 is equal to 1203.756 ns. The reports also indicate 87%, 89% and 94% total logic elements utilization in chip1, 2 and 3 respectively. A comparison with other implementations is not applicable since this is the first time this hash function is FPGA- implemented. This and other related issues will be dealt with in future development of the device.
Appendix A: The analysis and synthesis report details of chip1 of MDP-384:

Family: Cyclone II
Device: EP2C70F896C6
Total logic elements: 59,571 out of 68,416 (87 %)
Total combinational functions: 59,571
Logic element usage by number of LUT inputs
- 4 input functions: 13,696
- 3 input functions: 33,868

Table A1: A comparison between optimization technique implementations of chip1 of MDP-384

<table>
<thead>
<tr>
<th>Balance</th>
<th>Area</th>
<th>Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total logic elements</td>
<td>59571</td>
<td>58892</td>
</tr>
<tr>
<td>Total combinational</td>
<td>59571</td>
<td>58892</td>
</tr>
<tr>
<td>functions:</td>
<td>4 input functions</td>
<td>13696</td>
</tr>
<tr>
<td></td>
<td>3 input functions</td>
<td>33868</td>
</tr>
<tr>
<td>&lt;= 2 input functions</td>
<td>12007</td>
<td>11928</td>
</tr>
<tr>
<td>Maximum fan-out</td>
<td>39</td>
<td>35</td>
</tr>
<tr>
<td>Total fan-out</td>
<td>180399</td>
<td>175780</td>
</tr>
<tr>
<td>Average fan-out</td>
<td>3.00</td>
<td>2.96</td>
</tr>
<tr>
<td>Block interconnects</td>
<td>98509</td>
<td>96734</td>
</tr>
<tr>
<td>C16 interconnects</td>
<td>1308</td>
<td>1459</td>
</tr>
</tbody>
</table>
Appendix B: The analysis and synthesis report details of chip2 of MDP-384:

Family: Cyclone II
Device: EP2C70F896C6
Total logic elements: 60,740 out of 68,416 (89 %)
Total combinational functions: 60,740
Logic element usage by number of LUT inputs
- 4 input functions: 14,143
- 3 input functions: 35,108
- <=2 input functions: 11,489
Total memory bits: 0 out of 1,152,000 (0 %)
Embedded Multiplier 9-bit elements: 0 out of 300 (0 %)
Total PLLs: 0 out of 4 (0 %)
Optimization Technique: Balanced
Maximum fan-out: 28
Total fan-out: 184776
Average fan-out: 3.02

<table>
<thead>
<tr>
<th>Fitter Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block interconnects: 99,603 out of 197,592 (50 %)</td>
</tr>
<tr>
<td>C16 interconnects: 942 out of 6,270 (15 %)</td>
</tr>
<tr>
<td>C4 interconnects: 57,565 out of 123,120 (47 %)</td>
</tr>
<tr>
<td>Direct links: 15,988 out of 197,592 (8 %)</td>
</tr>
<tr>
<td>Global clocks: 0 out of 16 (0 %)</td>
</tr>
<tr>
<td>Local interconnects: 883 out of 68,416 (1 %)</td>
</tr>
<tr>
<td>R24 interconnects: 1,339 out of 5,926 (23 %)</td>
</tr>
<tr>
<td>R4 interconnects: 70,041 out of 167,484 (42 %)</td>
</tr>
</tbody>
</table>

Table B1 shows the number of usage logic elements and the interconnections between them in Area, Speed, and Balanced optimization technique. Also, the comparison between optimization techniques was extracted from the timing reports of implementing Area, Speed, and Balanced optimization.

- In Balanced optimization, Longest pin-to-pin delay from source pin "Input[32]" to destination pin "A[31]" was 1159.366 ns. Total cell delay was 680.616 ns. Total interconnect delay was 478.750 ns.
- In Area optimization, Longest pin-to-pin delay from source pin "Input[36]" to destination pin "P[30]" was 1150.398 ns. Total cell delay was 677.351 ns. Total interconnect delay was 473.047 ns.
- In Speed optimization, Longest pin-to-pin delay from source pin "Input[32]" to destination pin "P[20]" was 1171.155 ns. Total cell delay was 671.717 ns. Total interconnect delay was 499.438 ns.

Table B1: A comparison between optimization technique implementations of chip2 of MDP-384

| C4 interconnects | 59229 | 58464 | 63564 |
| Direct links | 15113 | 15402 | 14993 |
| Global clocks | 0 | 0 | 0 |
| Local interconnects | 904 | 627 | 822 |
| R24 interconnects | 1281 | 1227 | 1789 |
| R4 interconnects | 67646 | 66031 | 77125 |

Appendix C: The analysis and synthesis report details of chip3 of MDP-384:

Family: Cyclone II
Device: EP2C70F896C6
Total logic elements: 64,479 out of 68,416 (94 %)
Total combinational functions: 64,479
Logic element usage by number of LUT inputs
- 4 input functions: 14,583
- 3 input functions: 34,544
- <=2 input functions: 15,352
Total memory bits: 0 out of 1,152,000 (0 %)
Embedded Multiplier 9-bit elements: 0 out of 300 (0 %)
Total PLLs: 0 out of 4 (0 %)
Optimization Technique: Balanced
Maximum fan-out: 25
Total fan-out: 192436
Average fan-out: 2.97

<table>
<thead>
<tr>
<th>Fitter Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block interconnects: 103,679 out of 197,592 (52 %)</td>
</tr>
<tr>
<td>C16 interconnects: 1,164 out of 6,270 (19 %)</td>
</tr>
<tr>
<td>C4 interconnects: 60,876 out of 123,120 (49 %)</td>
</tr>
<tr>
<td>Direct links: 18,265 out of 197,592 (9 %)</td>
</tr>
<tr>
<td>Global clocks: 0 out of 16 (0 %)</td>
</tr>
<tr>
<td>Local interconnects: 688 out of 68,416 (1 %)</td>
</tr>
<tr>
<td>R24 interconnects: 1,305 out of 5,926 (22 %)</td>
</tr>
<tr>
<td>R4 interconnects: 70,453 out of 167,484 (42 %)</td>
</tr>
<tr>
<td>Nominal Core Voltage: 1.20 V</td>
</tr>
<tr>
<td>Low Junction Temperature: 0 °C</td>
</tr>
<tr>
<td>High Junction Temperature: 85 °C</td>
</tr>
</tbody>
</table>

Table C1 shows the number of usage logic elements and the interconnections between them in Area, Speed, and Balanced optimization technique. Also, the comparison between optimization techniques was extracted from the
timing reports of implementing Area, Speed, and Balanced optimization.

- In Balanced optimization, Longest pin-to-pin delay from source pin "Input[96]" to destination pin "Output[254]" was 1203.756 ns. Total cell delay was 686.548 ns. Total interconnect delay was 517.208 ns.
- In Area optimization, Longest pin-to-pin delay from source pin "Input[34]" to destination pin "Output[254]" was 1191.374 ns. Total cell delay was 683.596 ns. Total interconnect delay was 507.778 ns.
- In Speed optimization, Longest pin-to-pin delay from source pin "Input[98]" to destination pin "Output[254]" was 1213.668 ns. Total cell delay was 658.809 ns. Total interconnect delay was 554.859 ns.

Table C1: A comparison between optimization technique implementations of chip 3 of MDP-384

<table>
<thead>
<tr>
<th></th>
<th>Balance</th>
<th>Area</th>
<th>Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total logic elements</td>
<td>64479</td>
<td>63413</td>
<td>65627</td>
</tr>
<tr>
<td>Total combinational</td>
<td>64479</td>
<td>63413</td>
<td>65627</td>
</tr>
<tr>
<td>functions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 input functions</td>
<td>14583</td>
<td>10125</td>
<td>19945</td>
</tr>
<tr>
<td>3 input functions</td>
<td>34544</td>
<td>38108</td>
<td>29930</td>
</tr>
<tr>
<td>&lt;=2 input functions</td>
<td>15352</td>
<td>15180</td>
<td>15752</td>
</tr>
<tr>
<td>Maximum fan-out</td>
<td>25</td>
<td>26</td>
<td>29</td>
</tr>
<tr>
<td>Total fan-out</td>
<td>192436</td>
<td>184952</td>
<td>200842</td>
</tr>
<tr>
<td>Average fan-out</td>
<td>2.97</td>
<td>2.90</td>
<td>3.04</td>
</tr>
<tr>
<td>Block interconnects</td>
<td>103679</td>
<td>100898</td>
<td>108563</td>
</tr>
<tr>
<td>C16 interconnects</td>
<td>1164</td>
<td>726</td>
<td>1420</td>
</tr>
<tr>
<td>C4 interconnects</td>
<td>60876</td>
<td>55432</td>
<td>66769</td>
</tr>
<tr>
<td>Direct links</td>
<td>18265</td>
<td>18173</td>
<td>17526</td>
</tr>
<tr>
<td>Global clocks</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Local interconnects</td>
<td>688</td>
<td>324</td>
<td>979</td>
</tr>
<tr>
<td>R24 interconnects</td>
<td>1305</td>
<td>1235</td>
<td>1556</td>
</tr>
<tr>
<td>R4 interconnects</td>
<td>70453</td>
<td>67713</td>
<td>78926</td>
</tr>
</tbody>
</table>

Appendix D:

Sample VHDL code for a MDP-1 module of hash function

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.all;

ENTITY MDP_1 IS
Port (input_to_hash : std_logic_vector(1023 downto 0);
M_D_P_1 : out std_logic_vector(383 downto 0));
END MDP_1;

ARCHITECTURE behavioral OF MDP_1 IS

signal W0 : std_logic_vector (31 downto 0);
signal K0 : std_logic_vector(31 downto 0) :=
"011000000011000011010000001111111111111';
signal A0 : std_logic_vector(31 downto 0) :=
"0100111111111110101011000001100000';
signal B0 : std_logic_vector(31 downto 0) :=
"00110100101111010110010000000000';
signal C0 : std_logic_vector(31 downto 0) :=
"01000011110011101110001111111011001';
signal D0 : std_logic_vector(31 downto 0) :=
"01001100000010000000111111100100000';
signal E0 : std_logic_vector(31 downto 0) :=
"00100011110011101110110101000101';
signal F0 : std_logic_vector(31 downto 0) :=
"0001010101011111111111010101000000';
signal P0 : std_logic_vector(31 downto 0) :=
"0000101100000011111111111100101010';
signal Q0 : std_logic_vector(31 downto 0) :=
"0010001111111110111111111101000011';
signal R0 : std_logic_vector(31 downto 0) :=
"0011001100000011111111111110000111';
signal S0 : std_logic_vector(31 downto 0) :=
"0110001111011000110110001111000111';
signal T0 : std_logic_vector(31 downto 0) :=
"0010001111011011111011010101000011';
signal U0 : std_logic_vector(31 downto 0) :=
"0011001111111111111000000000000000';
signal A1, B1, C1, D1, E1, F1, P1, Q1, R1, S1, T1, U1 : std_logic_vector(31 downto 0);

BEGIN

A1 <= (P0(30 downto 0) & P0(31)) + ((P0 and Q0) or ((not P0) and R0)) + ((R0 and S0) or ((not R0) and T0)) + U0 + W0 + K0;
B1 <= (T0(26 downto 0) & T0(31) & T0(30) & T0(29) & T0(28) & T0(27));
C1 <= (P0(30 downto 0) & P0(31));
D1 <= (Q0(29 downto 0) & Q0(31) & Q0(30)) xor (P0(30 downto 0) & P0(31));
E1 <= (R0(28 downto 0) & R0(31) & R0(30) & R0(29)) xor (Q0(29 downto 0) & Q0(31) & Q0(30));
F1 <= (S0(27 downto 0) & S0(31) & S0(30) & S0(29) & S0(28)) xor (R0(28 downto 0) & R0(31) & R0(30) & R0(29));
P1 <= (A0(30 downto 0) & A0(31)) + ((A0 and B0) or ((not A0) and C0)) + ((C0 and D0) or ((not C0) and E0)) + F0 + W0 + K0;
Q1 <= (E0(26 downto 0) & E0(31) & E0(30) & E0(29) & E0(28) & E0(27));
R1 <= (A0(30 downto 0) & A0(31));
S1 <= (B0(29 downto 0) & B0(31) & B0(30)) xor (A0(30 downto 0) & A0(31));
Sample VHDL code for a MDP-352 module of hash function

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.all;

ENTITY MDP_352 IS
Port (A351, B351, C351, D351, E351, F351, P351, Q351, R351, S351, T351, U351 :
       in std_logic_vector (31 downto 0);
W319 : in std_logic_vector (31 downto 0);
M_D_P_352: out std_logic_vector (383 downto 0));
END MDP_352;

ARCHITECTURE behavioral OF MDP_352 IS
signal K11 : std_logic_vector (31 downto 0) :=
"1110111110000100011100100001";
signal A352, B352, C352, D352, E352, F352, P352, Q352, R352, S352, T352, U352:
       std_logic_vector (31 downto 0);

BEGIN
A352 <= (P351(30 downto 0) & P351(31)) + (P351 xor Q351 xor R351) + (R351 xor S351 xor T351) + U351 + W351 + K11;
B352 <= (T351(26 downto 0) & T351(31) & T351(30) & T351(29) & T351(28) & T351(27));
C352 <= (P351(30 downto 0) & P351(31));
D352 <= (Q351(29 downto 0) & Q351(31) & Q351(30)) xor (P351(30 downto 0) & P351(31));
E352 <= (R351(28 downto 0) & R351(31) & R351(30) & R351(29)) xor (Q351(29 downto 0) & Q351(31) & Q351(30));
F352 <= (S351(27 downto 0) & S351(31) & S351(30) & S351(29) & S351(28)) xor (R351(28 downto 0) & R351(31) & R351(30) & R351(29));
P352 <= (A351(30 downto 0) & A351(31)) + (A351 xor B351 xor C351) + (C351 xor D351 xor E351) + F351 + W351 + K11;
Q352 <= (E351(26 downto 0) & E351(31) & E351(30) & E351(29) & E351(28) & E351(27));
R352 <= (A351(30 downto 0) & A351(31));
S352 <= (B351(29 downto 0) & B351(31) & B351(30)) xor (A351(30 downto 0) & A351(31));
T352 <= (C351(28 downt to 0) & C351(31) & C351(30) & C351(29)) xor (B351(29 downto 0) & B351(31) & B351(30));
U352 <= (D351(27 downto 0) & D351(31) & D351(30) & D351(29) & D351(28)) xor (C351(28 downto 0) & C351(31) & C351(30) & C351(29));

END behavioral;

Magdy Saeb received the BSEE, School of Engineering, Cairo University, in 1974, the MSEE, and Ph.D. degrees in Electrical & Computer Engineering, University of California, Irvine, in 1981 and 1985, respectively. He was with Kaiser Aerospace and Electronics, Irvine California, and The Atomic Energy Establishment, Anshas, Egypt. Currently, he is a professor in the Department of Computer Engineering, Arab Academy for Science, Technology & Maritime Transport, Alexandria, Egypt; He was on-leave working as a principal researcher in the Malaysian Institute of MicroElectronic Systems (MIMOS). His current research interests include Cryptography, FPGA Implementations of Cryptography and Steganography Data Security Techniques, Encryption Processors, Mobile Agent Security. www.magdysaeb.net

Rabie Mahmoud received the BS Degree, Faculty of Science, Tishreen University, Syria, in 2001, the MS, in Computational Science, Faculty of Science, Cairo University, Egypt, in 2007. Currently, he is working on his Ph.D. Dissertation in the Department of Mathematics, Faculty of Science, Cairo University, His current interests include Image processing, Cryptography, FPGA Implementations of Cryptography and Data Security Techniques. rabiemah@yahoo.com