FPGA-based Implementation of SHA-256 with Improvement of Throughput using Unfolding Transformation - Explained

Original invention by Shamsiah Suhaili and Norhuzaimin Julai.

Original documentation https://doi.org/10.47836/pjst.30.1.32

Author of this document: Juha-Pekka Varjonen, Geeky Gadgets Online © 2024, Finland

PREFACE

Original presentation is so horribly badly documented and hard to understand. Also it contains many errors. So I made this new clear and straightforward documentation about it after a days of development and research... Now everybody can understand how cleverly it works with these simple steps. Original authors did not even answer my e-mails so they just don’t care. Here it begins, be happy, stay fluffy and enjoy! 😁

ABSTRACT

This paper proposes the fast implementation of the SHA-256 hash function. It uses unfolding method that increases SHA-256 output performance. This unfolding method decreases the number of clock cycles required for traditional architecture by a factor of 4. From 64 to 16 clock cycles.

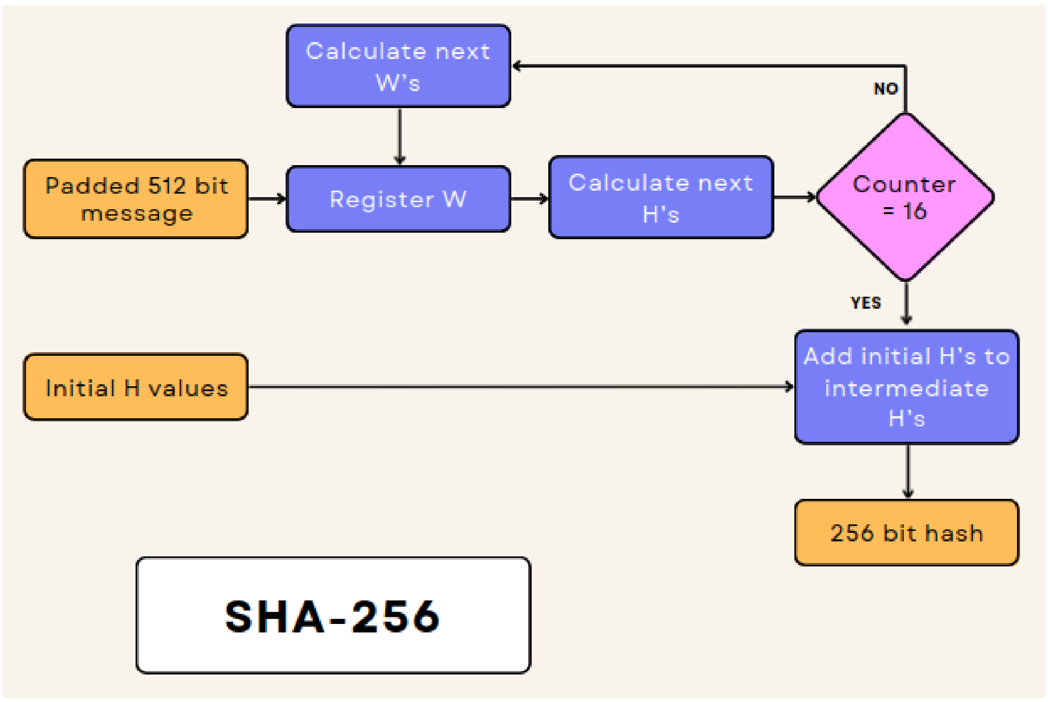
Throughput is around 4196 Mbps with this technique. It is 58% improvement from the conventional design.

MATERIALS AND METHODS

Unfolding means parallel calculations instead of one after another. This unfolding by four is possible because only every fourth line is actually different in SHA-256 algorithm output. Here is simple example of beginning of output register H content of clock cycles from zero to seven. Data content is irrelevant at this point.

```
           A        B        C        D        E        F        G        H
t= 0: 5D6AEBCD 6A09E667 BB67AE85 3C6EF372 FA2A4622 510E527F 9B05688C 1F83D9AB
t= 1: 5A6AD9AD 5D6AEBCD 6A09E667 BB67AE85 78CE7989 FA2A4622 510E527F 9B05688C
t= 2: C8C347A7 5A6AD9AD 5D6AEBCD 6A09E667 F92939EB 78CE7989 FA2A4622 510E527F
t= 3: D550F666 C8C347A7 5A6AD9AD 5D6AEBCD 24E00850 F92939EB 78CE7989 FA2A4622
t= 4: 04409A6A D550F666 C8C347A7 5A6AD9AD 43ADA245 24E00850 F92939EB 78CE7989
t= 5: 2B4209F5 04409A6A D550F666 C8C347A7 714260AD 43ADA245 24E00850 F92939EB
t= 6: E5030380 2B4209F5 04409A6A D550F666 9B27A401 714260AD 43ADA245 24E00850
t= 7: 85A07B5F E5030380 2B4209F5 04409A6A 0C657A79 9B27A401 714260AD 43ADA245
```

It looks very repetitive and can easily be faster. Imagine all those eight lines in *two* clock cycles! Below is simplified flowchart of what we are going to do.



First we need six functions. Same than in SHA-256 standard. I repeat them here for your convenience. I changed letters more understandable ones.

$$Ch(e,f,g)=(e\wedge f)\oplus(\neg e\wedge g)$$

$$Maj(a,b,c)=(a\wedge b)\oplus(a\wedge c)\oplus(b\wedge c)$$

$$\Sigma_0(a)=ROTR^2(a)\oplus ROTR^{13}(a)\oplus ROTR^{22}(a)$$

$$\Sigma_1(e)=ROTR^6(e)\oplus ROTR^{11}(e)\oplus ROTR^{25}(e)$$

$$\sigma_0(w)=ROTR^7(w)\oplus ROTR^{18}(w)\oplus SHR^3(w)$$

$$\sigma_1(w)=ROTR^{17}(w)\oplus ROTR^{19}(w)\oplus SHR^{10}(w)$$

Hint:
You can make *ROTR*^{x} function with this code where *n* is rotated number and *x* is amount of rotation.

```
(n >> x) | (n << (32 – x)) % 2 ** 32
```

We also need constants H and K from SHA-256 standard. That PDF is freely available from internet and it is good to read before going any further. I made example program with Python and simulation with free open source program called Digital. Both codes are freely downloadable from GitHub. Digital is downloadable from https://github.com/hneemann/Digital. It is better than Logisim.

Finally two T function is needed as in standard. I removed and cleared all unnecessary numbers from function declarations for your convenience.

$$T_1=h+\Sigma_1(e)+Ch(e,f,g)+K+W$$

$$T_2=\Sigma_0(a)+Maj(a,b,c)$$

Initialize constants K and H to appropriate registers and padded 512 bit message to register W and we are ready for coding or simulation, either one is your intention. Our intention is to calculate next four W register values and all eight H register values during one clock cycle. There is total of 16 clock cycles needed. Next I will present a content of a first clock cycle. Rest 15 are simply repeating it. All additions (+) are performed with modulo 2^{32}.

Calculating next W values

$\sigma_{00}=\sigma_0(W_1)$
$\sigma_{10}=\sigma_1(W_{14})$
$next_{W0}=(\sigma_{00}+W_0+\sigma_{10}+W_9)\,mod\,2^{32}$
$\sigma_{01}=\sigma_0(W_2)$
$\sigma_{11}=\sigma_1(W_{15})$
$next_{W1}=(\sigma_{01}+W_1+\sigma_{11}+W_{10})\,mod\,2^{32}$
$\sigma_{02}=\sigma_0(W_3)$
$\sigma_{12}=\sigma_1(next_{W0})$
$next_{W2}=(\sigma_{02}+W_2+\sigma_{12}+W_{11})\,mod\,2^{32}$
$\sigma_{03}=\sigma_0(W_4)$
$\sigma_{13}=\sigma_1(next_{W1})$
$next_{W3}=(\sigma_{03}+W_3+\sigma_{13}+W_{12})\,mod\,2^{32}$

Calculating next E values

$T_{10}=\Sigma_1(H_4)+Ch(H_4,H_5,H_6)+H_7+K_{counter\cdot 4}+W_0$
$next_{E0}=(H_3+T_{10})\,mod\,2^{32}$
$T_{11}=\Sigma_1(next_{E0})+Ch(next_{E0},H_4,H_5)+H_6+K_{counter\cdot 4+1}+W_1$
$next_{E1}=(H_2+T_{11})\,mod\,2^{32}$
$T_{12}=\Sigma_1(next_{E1})+Ch(next_{E1},next_{E0},H_4)+H_5+K_{counter\cdot 4+2}+W_2$
$next_{E2}=(H_1+T_{12})\,mod\,2^{32}$
$T_{13}=\Sigma_1(next_{E2})+Ch(next_{E2},next_{E1},next_{E0})+H_4+K_{counter\cdot 4+3}+W_3$
$next_{E3}=(H_0+T_{13})\,mod\,2^{32}$

Calculating next A values

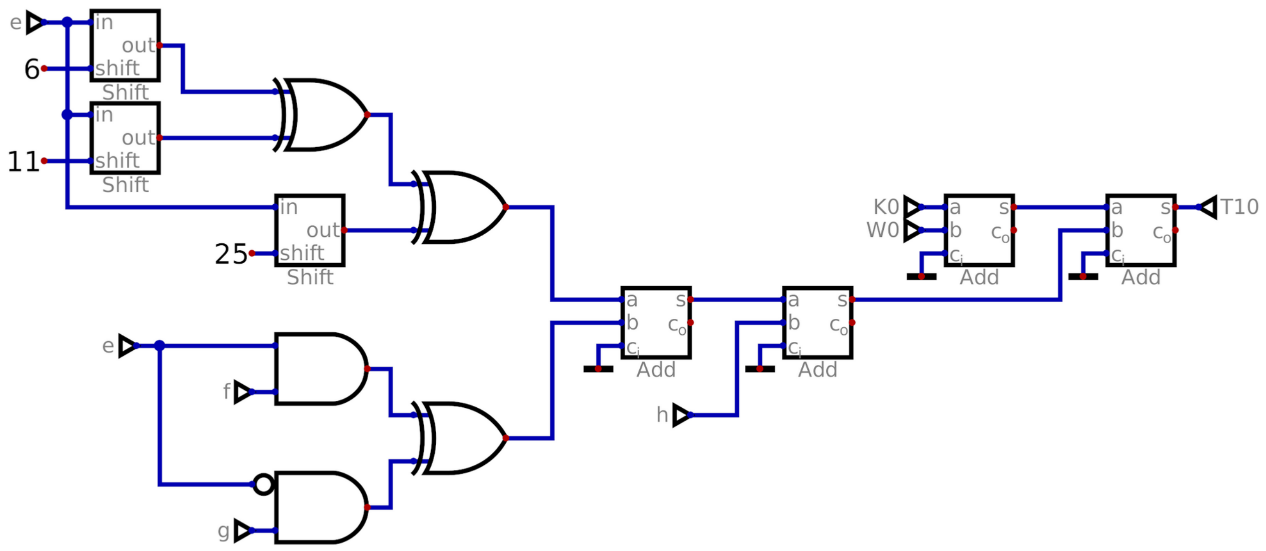
$T_{20}=\Sigma_2(H_0)+Maj(H_0,H_1,H_2)$
$next_{A0}=(T_{20}+T_{10})\,mod\,2^{32}$
$T_{21}=\Sigma_2(next_{A0})+Maj(next_{A0},H_0,H_1)$
$next_{A1}=(T_{21}+T_{11})\,mod\,2^{32}$
$T_{22}=\Sigma_2(next_{A1})+Maj(next_{A1},next_{A0},H_0)$
$next_{A2}=(T_{22}+T_{12})\,mod\,2^{32}$
$T_{23}=\Sigma_2(next_{A2})+Maj(next_{A2},next_{A1},next_{A0})$
$next_{A2}=(T_{23}+T_{13})\,mod\,2^{32}$

Update H and W registers

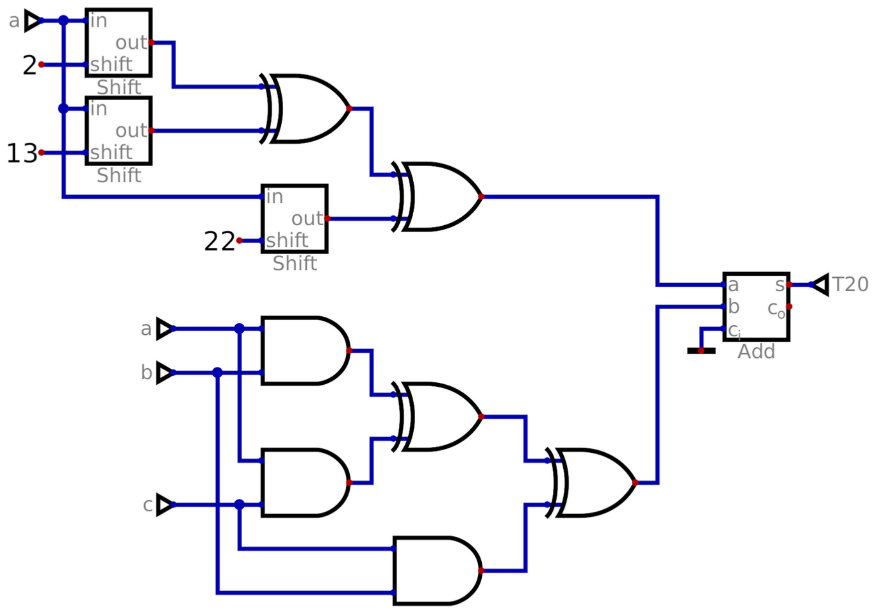
$H_0 = next_{A3}$
$H_1 = next_{A2}$
$H_2 = next_{A1}$
$H_3 = next_{A0}$
$H_4 = next_{E3}$
$H_5 = next_{E2}$
$H_6 = next_{E1}$
$H_7 = next_{E0}$
(shift left by four and add to the end)
$W = W[4:] + [next_{W0}, next_{W1}, next_{W2}, next_{W3}]$

After all 16 clock cycles register H contains intermediate SHA-256 hash value. Add initial value to every register position and then output is new hash. It’s that simple! If message W is longer than 512 bit then initial H value is previous hash.

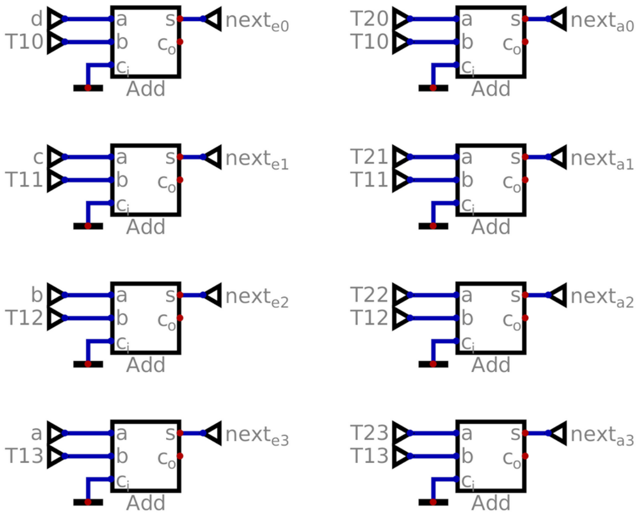
So, we are ready to make simulation and VHDL code with Digital or Logisim or any other suitable software. Below is each different blocks and how many of them is needed.



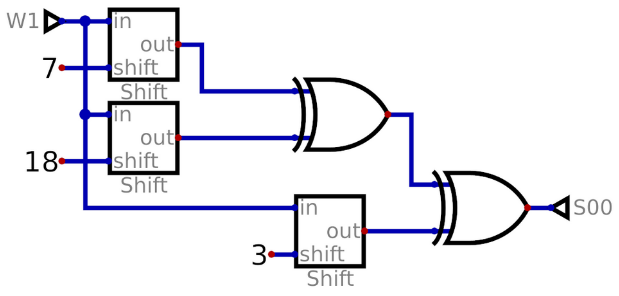
Above block contains Σ_1 and *Ch* functions which are summed together with 32 bit full adders. This block outputs T_{10} so we need them a total of four pieces. See connection from example file or function declarations earlier.



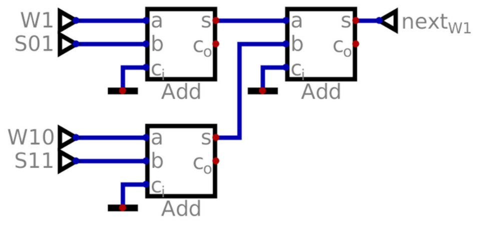
This block contains Σ_0 and *Maj* functions which are then summed together with 32 bit full adder. It outputs T_{20} and we need four of these. See again connections from file or above equations.



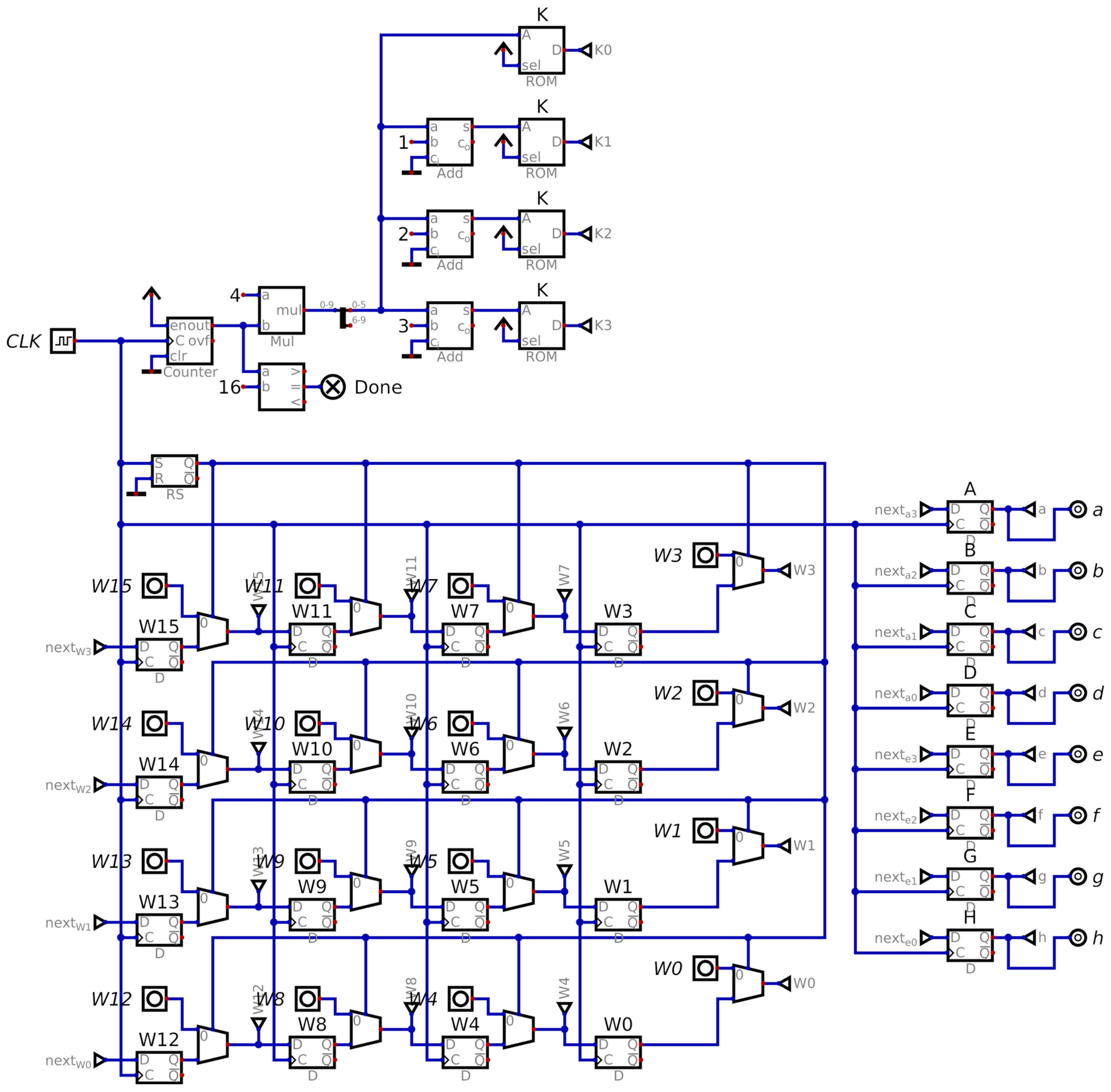
These are all final 32 bit full adders for H register update.



These are S_{00} and S_{10} blocks. We need four of both of them. Please notice that lowest shift is different than any other shift blocks. It is *x >> 3* and *x >> 10* respectively. So no rotate right like all others have.



Final summation blocks for W register. We need four of them. See correct connections from documentation earlier from this document.



This final block is application specific and it's here only for example. There is system clock and counter and registers K, H and W. This doesn't add initial H values to result because it is done by host processor in my case and not the FPGA.

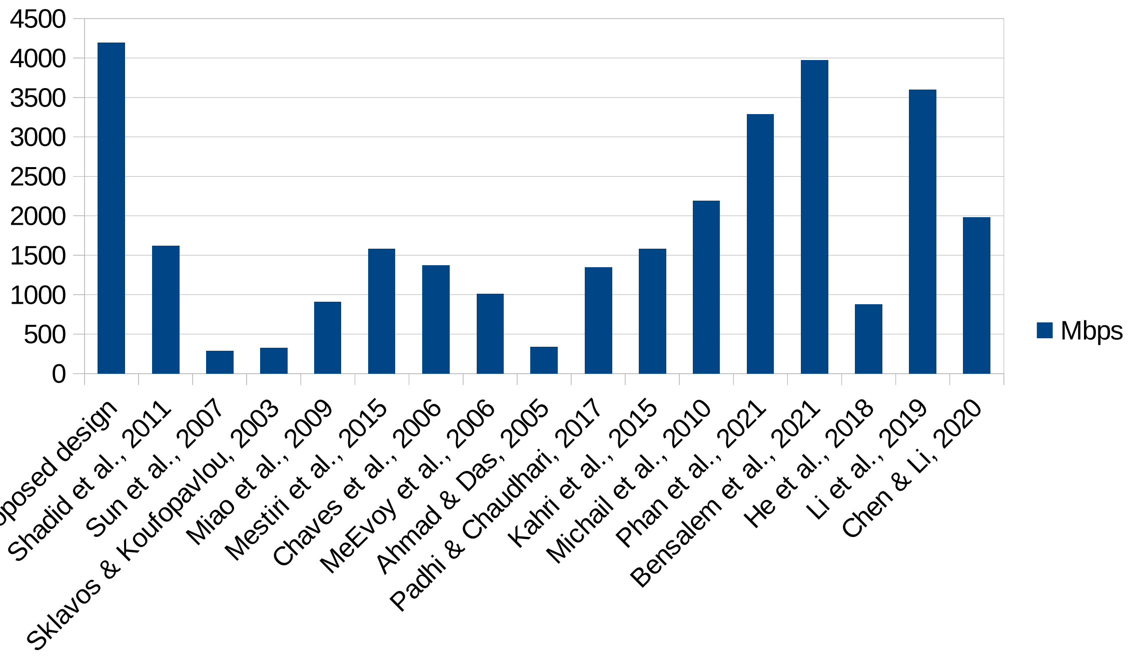
RESULTS AND DISCUSSION

Throughput is around 4196 Mbps with this technique. It is 58% improvement from the conventional design. It is calculated with this equation:

$$Throughput = \frac{512 \cdot FMax}{Number\ of\ Cycle}$$

Most of the previous SHA-256 algorithms were applied the iterative and pipelining method to design the hash function. Therefore the area implementation of the designs was smaller than the unfolding method.

Comparison of different SHA-256 design results.



According to the results, the proposed design had the highest throughput. Compared to traditional architecture, this technique enhanced the design by eliminating round cycles.

CONCLUSION

By applying this unfolding method with factor four to the SHA-256 hash function, the throughput of the design increase significantly. It is the best solution to improve the performance of the hash function.