High-Speed Parallel Architecture of the Whirlpool Hash Function

Deen Kotturi[1] and Seong-Moo Yoo[2]

[1]*Cadence Design Systems, 6400 Int'l Pkwy Suite 1500, Plano, Texas 75093 USA*
[2]*ECE Dept, The Univ. of Alabama in Huntsville, Huntsville, Alabama 35899 USA*
yoos@eng.uah.edu

Abstract

Whirlpool hash function should be capable of processing the input data streams at high speeds. We propose a fully synchronous parallel pipelined architecture with ten stages of pipelining between rounds, and internal pipelining within each round stage. The proposed architecture can tremendously improve the performance of Whirlpool hash function. Our final implementation can encrypt continuous bit streams seamlessly, achieving a throughput of 56.89 G bps, which is considerably high compared to existing implementations in literature.

Keywords: *pipelining, throughput, Whirlpool hash function.*

1. Introduction

Data security is an ever changing, rapidly developing field where new security architectures are developed to overcome security attacks on the existing encryption methodologies. Various hash algorithms were developed to serve the purpose of enhancing data security [1]. Among them, Whirlpool [2] is a relatively new hash function that is now a member of the New European Schemes for Signatures Integrity, and Encryption (NESSIE) [3]. The International Organization for Standardization has adopted Whirlpool in ISO specification [4].

There are several reports on Whirlpool hardware architectures [5, 6, 7, 8]. Kitsos and Koufopavlou [5] presented area-efficient architectures as well as high-speed designs, the fastest of which runs at 4.48 Gbps. McLoone et al. [6] presented a full look-up table based design and reported a high throughput of 4.9 Gbps.

In this paper, we propose a fully synchronous parallel pipelined architecture with ten stages of pipelining between rounds and internal pipelining within each round stage. Our final implementation can encrypt continuous bit streams seamlessly achieving a throughput of 56.89 Gbps in CMOS 130nm technology based Xilinx Virtex-II Pro FPGA. Our throughput is very high compared to those of earlier reports.

2. Proposed high-speed hardware architecture

Fig. 1 shows ten identical blocks with pipelining between each block. These blocks are instantiations of the round block that produces the intermediate cipher state for each of the rounds. The output of WHPL_RND10 is passed to GSO (Generate Stream Out) block that creates the output of Whirlpool hash function. At the top-level, therefore, the Whirlpool hash function can be simply implemented with:

(a) ten instantiations of round block that process the intermediate cipher state,

(b) key schedule block that creates all the ten round keys, and
(c) an additional block to perform XOR operations on the output of ten-round transformation on intermediate cipher state with the input state to produce the Whirlpool hash output.

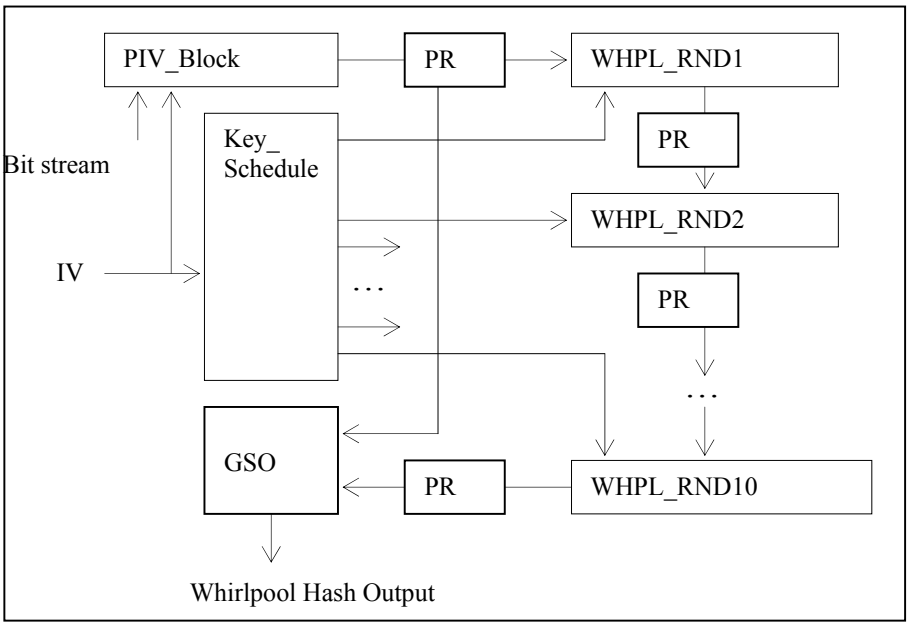

Figure 1. Top level Whirlpool representation. All primary and intermediate signals are synchronous. (PR: pipeline register)

Fig. 2 shows our implementation of the round function in WHPL_RND block which contains the following operations.
(a) Perform non-linear substitution on 512-bit state and divide the 512-bit data into eight 64-bit blocks.
(b) Perform shift column operation on the result of (a).
(c) Perform circulant transformation on the result of (b).
(d) Add round key to result of circulant transformation from (c).

These operations are implemented within each of the WHPL_RND1 to WHPL_RND10 instances of the round function block with internal pipelining between Steps (a), (b), (c) and (d). Each instance WHPL_RND, of the round function block implements the round function $\rho(k)$ [2]. Functions $\sigma(k)$, θ, π and γ [2] are implemented as key_add_Row, circulant, shiftCols and makeRows_alr blocks, respectively.

Fig. 3 shows the pseudo code representing our Verilog based netlist used to implement makeRows_alr block. The outputs of the makeRows_alr block are eight 64-bit wide rows. To output each of the 64-bit rows, the makeRows_alr uses 4 DPRAMs configured as 4 ROMs readily available in the Xilinx Virtex-II pro platform. Therefore, to output the result after a single round transformation, our implementation requires 32 DPRAMs.

The shiftCols block takes the output from the makeRows_alr block through an internal pipeline register and shifts each column cyclically. This block is implemented using 512 slices flip-flops by rearranging the 512-bit state to perform the π function. The circulant block implements the linear diffusion layer described as

the θ function. This is a pure combinational block performing linear diffusion on the 8 × 8 state matrix using the circulant matrix. Multiplication of the 8 × 8 state matrix elements with circulant matrix elements 01, 02, 04, 05, 08 and 09 are implemented as functions that represent multiplications in GF (2^8) using the primitive polynomial $x^8 + x^4 + x^3 + x^2 + 1$. Fig. 4 shows Verilog based pseudo-code for multiplication of state byte with the primitive polynomial.

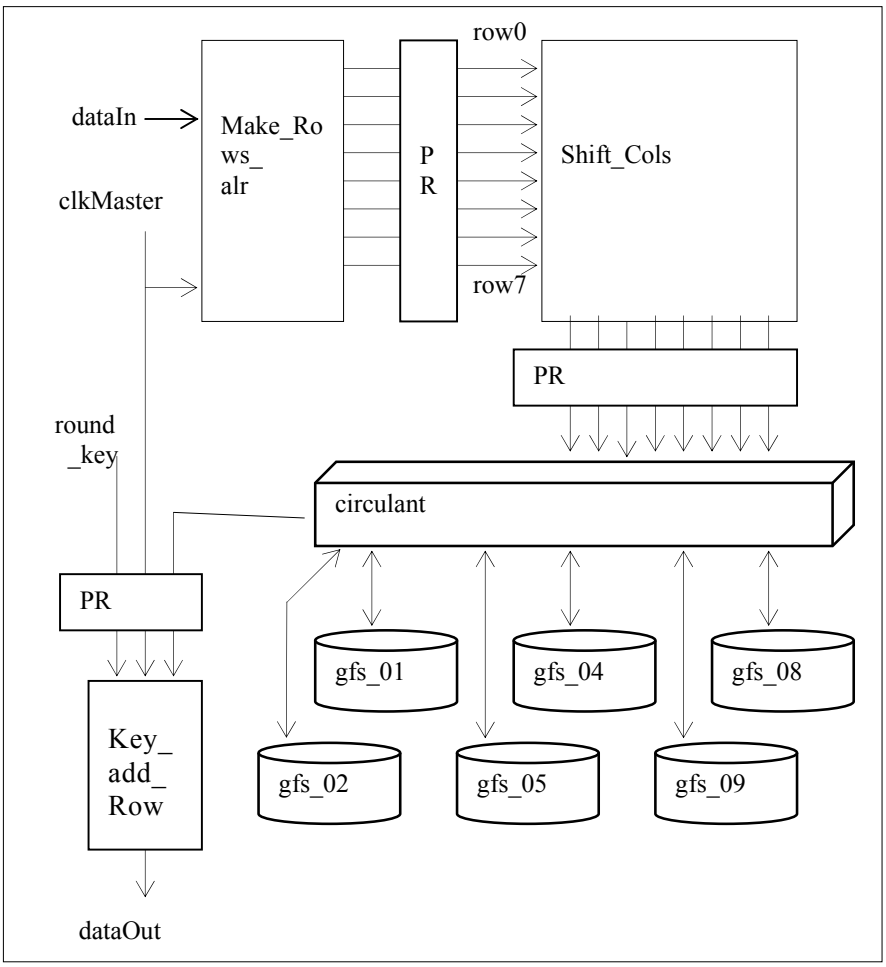

Figure 2. Internal block level architecture of the round function

```
stateIn[511:0] ←  get 512-bit round function input
                  from input register
for each j ←  0 to 7
   i = (7 – j) * 64;
   sRowj[63:0]  ←  stateIn[(i + 63):i];
for each j ←  0 to 7
  do rowj[63:0] ← SBOXLUT (srowj[63:0])
  for each k ← 0 to 7
  outputresultk[63:0]  ← rowj[63:0]
        // outputresultk is a registered output
end
```

Figure 3. Pseudo-code for makeRows_alr implementation

Using the repetitive operations in Fig. 4, multiplication of the 8 × 8 state with circulant can be implemented using 1408 XOR gates for each round. The resultant 8 × 8 matrix after circulant operation is divided into eight internally pipelined 64-bit rows using eight 64-bit registers.

```
▶ function mult is
  return  γ ← (state[07:00] << 1) ^
          (8'b00011101 & {8{state[07]}});
▶ function mult_02 is return mult_02 ← γ(stateByte);
▶ function mult_04 is
  return mult_04 ← γ (mult_02);
▶ function mult_05 is
  return mult_05 ← γ (mult_04) ^ (stateByte);
▶ function mult_08 is return mult_08 ← γ (mult_08);
▶ function mult_09 is
 return mult_09 ← γ (mult_08) ^ (statebyte);
```

Figure 4. Pseudo-code for implementing θ function.

The Key_add_Row block takes as input the output of circulant step plus the 512-bit round key generated by the key schedule hardware block. The round key is divided into eight 64-bit blocks. The Key_add_Row adds the roundkey to the 8 × 8 matrix output from the circulant step passed as input to key_add_Row.

Fig. 5 shows the internally pipelined round key block that is instantiated ten times to create all ten round keys. This is an implementation with two pipeline registers, one between the makeRows_alr block and the shiftRows block, and the other between the shiftRows block and the circulant block. The key schedule block uses ten instances of the round key block.

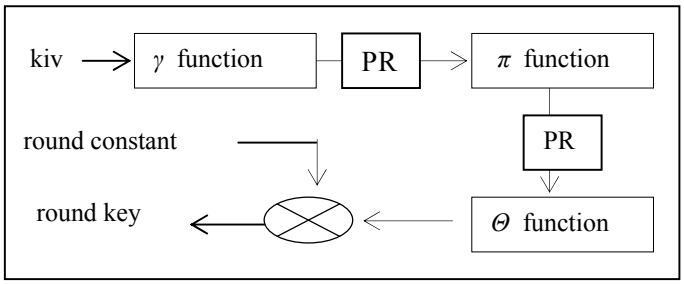

Figure 5. Pipelined round key block.

3. Performance comparison

In our implementation, XST (Xilinx Synthesis Tool) reported the post synthesis FF-to-FF worst path delay to be from *RND3_shift_Col7_30 (FF)* to *RND3_circ_Col7_10 (FF)* with a path delay of 0.912 ns. Xilinx ISE6.1 was used for synthesis, placement, routing and for producing a post layout database. The post layout database is simulated with a comprehensive vector driven simulation in Cadence's NC-Verilog/NC-Sim environment. The results of our simulation are verified against a Perl based software simulation.

Table 1 shows the results of our implementation compared against [5] and [6]. It is clear that out implementation produces a high throughput compared to the existing implementations.

Implement.	Proposed	[6]	[5]
Device	XC2VP125-6ff1704	X4VLX100	XCV1000EFG 1156-8
Hardware Utilization	IOBs 1025 BRAMs 320 Slices 22681	Slices 13210	Slices 5585
Throughput	56.89 Gbps	4.90 Gbps	4.48 Gbps
Frequency	111.1 Mhz	47.8 Mhz	87.5 Mhz

Table 1. Comparison of results from this implementation to others

4. Conclusion

In this paper, we proposed a high-speed parallel architecture of the Whirlpool hash function. Also, we showed how a fully synchronous parallel pipelined architecture with ten stages of pipelining between rounds and internal pipelining within each round stage can tremendously improve the performance of the Whirlpool hash function. Our throughput is considerably high compared to those of earlier reports.

References

[1] Annex A: Approved Security Functions for FIPS PUB 140-2, Security Requirements for Cryptographic Modules, National Institute of Standards and Technology, http://csrc.nist.gov/cryptval/, April 03, 2006.
[2] P. Barreto and V. Rijmen, “The Whirlpool Hashing Function”, May 24, 2003.
[3] Portfolio of Recommended Cryptographic Primitives, NESSIE Consortium, New European Schemes for Signatures, Integrity and Encryption, February 27, 2003.
[4] ISO/IEC 10118-3:2004, Information technology -- Security techniques -- Hash-functions -- Part 3: Dedicated hash-functions, International Organization for Standardization.
[5] P. Kitsos and O. Koufopavlou, “Efficient architecture and hardware implementation of the Whirlpool hash function”, IEEE Transactions on Consumer Electronics, Vol. 50, No. 1, February 2004, pp. 208-213.
[6] M. McLoone and C. McIvor, "High-speed & Low Area Hardware Architectures of the Whirlpool Hash Function," J. VLSI Signal Processing, vol. 47, no. 1, pp. 47-57, Apr. 2007.
[7] N. Pramstaller, C. Rechberger, and V. Rijmen. "A compact FPGA implementation of the hash function Whirlpool," Proc. ACM/SIGDA 14th Int. Sym. Field Programmable Gate Arrays (FPGA 2006), pp. 159-166, Feb. 2006.
[8] T. Alho, P. Hämäläinen, M. Hännikäinen, and T. D. Hämäläinen, "Compact hardware design of Whirlpool hashing core," Design, Automation and Test in Europe (DATE 07), pp.1247-1252, Apr. 2007.

Authors

Deen Kotturi received the M.S. degree in Electrical Engineering from the University of Alabama in Huntsville in 2004. He is working as a Lead Engineer at Cadence Design Systems, Inc since 2001 till to date. His primary interests are on Interconnect Modeling in Nanometer VLSI Designing, RC reduction algorithms for efficient Extraction for Timing and Signal Integrity Analysis, Transistor/Cell level simulation based design analysis for manufacturability and VLSI architectures for high speed digital circuits.

Seong-Moo Yoo received the M.S. and Ph.D. degree in computer science from the University of Texas at Arlington in 1989 and 1995, respectively. Since September 2001, he is an associate professor in Electrical and Computer Engineering Department of the University of Alabama in Huntsville, Huntsville, Alabama, U.S.A. From September 1996 to August 2001, he was an assistant professor in Computer Science Department of Columbus State University in Columbus, Georgia, U.S.A. Dr. Yoo is the conference chair of ACM Southeast Conference 2004, April, 2004, Huntsville, Alabama, U.S.A. He was the co-program chair of ISCA 16th International Conference on Parallel and Distributed Computing Systems (PDCS-2003), August 2003, Reno, Nevada, U.S.A. Dr. Yoo's research interests include wireless networks, parallel computer architecture, and computer network security. Dr. Yoo is a senior member of IEEE and a member of ACM.