IEICE TRANS. INF. & SYST., VOL.E101–D, NO.11 NOVEMBER 2018

PAPER Special Section on Information and Communication System Security

Design Exploration of SHA-3 ASIP for IoT on a 32-bit RISC-V Processor

Jinli RAO[†], *Nonmember*, **Tianyong AO**[††a)], *Member*, **Shu XU**[†††], **Kui DAI**[††††], *and* **Xuecheng ZOU**[†], *Nonmembers*

SUMMARY Data integrity is a key metric of security for Internet of Things (IoT) which refers to accuracy and reliability of data during transmission, storage and retrieval. Cryptographic hash functions are common means used for data integrity verification. Newly announced SHA-3 is the next generation hash function standard to replace existing SHA-1 and SHA-2 standards for better security. However, its underlying *Keccak* algorithm is computation intensive and thus limits its deployment on IoT systems which are normally equipped with 32-bit resource constrained embedded processors. This paper proposes two efficient SHA-3 ASIPs based on an open 32-bit RISC-V embedded processor named *Z-scale*. The first operation-oriented ASIP (OASIP) focuses on accelerating time-consuming operations with instruction set extensions to improve resource efficiency. And next datapath-oriented ASIP (DASIP) targets exploiting advance data and instruction level parallelism with extended auxiliary registers and customized datapath to achieve high performance. Implementation results show that both proposed ASIPs can effectively accelerate SHA-3 algorithm with 14.6% and 26.9% code size reductions, 30% and 87% resource efficiency improvements, 71% and 262% better maximum throughputs as well as 40% and 288% better power efficiencies than reference design. This work makes SHA-3 algorithm integration practical for both low-cost and high-performance IoT systems.

key words: *SHA-3, application specific instruction-set processor, RISC-V, IoT*

1. Introduction

With the widespread use of embedded IoT devices, information security attacks are also evolving [1]. Security becomes important as functionality, speed, resources and power in embedded system design [2], [3]. Data integrity is a key metric of security goals and used to confirm system data has not been tampered during transmission, storage and retrieval. It provides data accuracy and reliability for further processing [4]. Hash functions are common means used in integrity verification and authentication fields. In 2012, National Institute of Standards and Technology(NIST) announced the next-generation hash function standard named SHA-3 based on the *Keecak* [21] algorithm. It's proposed to replace the existing common hash functions like MD5, SHA-1 and SHA-2 for better security [5], [6].

Comparing to other SHA-3 candidate algorithms, *Keccak* uses a large internal state and operations on it are 64-bit friendly [22]. Its software implementation on 64-bit systems are efficient, but computational complexity increases significantly on 32-bit resource-constrained embedded systems. The achievable throughput is limited and extra heavy burden also slows down the whole system [28]. Hardware solutions like co-processors utilize specialized hardware to do parallel algorithm acceleration. They can effectively improve performance, but also bring additional resource overhead with limited flexibility and scalability which limits their deployments in IoT systems [7]–[10].

An alternative solution is to do fine-grain hardware based acceleration like Instruction Set Extensions(ISE) and datapath customization etc. on standard processors. It's a feasible way to achieve balance of performance, resource and flexibility. The processor designed on this basis is also known as ASIP (Application Specific Instruction-set Processor). An ASIP targets accelerating specification applications. It is normally based on a general-purpose processor with special instruction set extensions and datapath customizations to deliver high performance [11]–[14]. It introduces limited resource overhead while still provides promising software programmability which has been widely adopted in areas such as mobile computation, network communications and digital signal processing. *ASIP design* for SHA-3 has drawn attentions since the competition was launched. [15] explores acceleration of the SHA-3 candidate algorithms by doing ISE on a 16-bit PIC24 processor. The optimization for *Keecak* algorithm can achieve 29.8% performance improvement and 30.6% code reduction with negligible resource overhead. It shows the effectiveness of ASIP in accelerating SHA-3 algorithm. [16] extends a 32-bit LEON3 processor achieves about 87% performance improvement and 9.5% code size reduction with instruction extensions for 64-bit rotation. [17] implements 2 ASIPs. One shares the main processor datapath and the other customize a co-processor datapath. With tailored ISE, they can achieve 20% and 38% performance improvements with 9% and 26% hardware resource overhead, respectively. However, existing works mostly focus on basic operation acceleration in instruction level while has little exploration of advance data and instruction level parallelism for SHA-3 algorithm. This means there is still large space for more efficient SHA-3 ASIP design and implementation.

Manuscript received November 14, 2017.
Manuscript revised April 3, 2018.
Manuscript publicized August 22, 2018.

[†]The authors are with School of Optical and Electronic Information, Huazhong University of Science and Technology, Wuhan 430074, China.
[††]The author is with the School of Physics and Electronics, Henan University, Kaifeng, China.
[†††]The author is with Science and Technology on Information Assurance Laboratory, Beijing, China.
[††††]The author is with Institute of National Network Security and Information, Peking University, Beijing, China.
a) E-mail: tyaohust@gmail.com

DOI: 10.1587/transinf.2017ICP0019

This paper targets efficient SHA-3 ASIP design for IoT based on an open RISC-V embedded processor namely *Z-scale* [18] with help of *Synopsys ASIP Designer* [20]. On basis of SHA-3 algorithm performance profiling, we do ISE for intensive operations to remove most operation hotspots with minimal hardware resource overhead. This leads to the first operation-oriented SHA-3 ASIP. Further analysis shows memory bandwidth becomes performance domination. With new optimizations of auxiliary register extensions and customized datapath, it can successfully extends the life cycle of state data during processing and paralyzes computations with memory accesses. These exploitations result in another datapath-oriented SHA-3 ASIP which significantly improves the performance and reduces code size with acceptable resource overhead. Both solutions are resource and power efficient to make SHA-3 integration practical for low cost and high performance IoT systems. Technologies in this paper are implemented and verified on a 32-bit embedded processor which are also helpful for similar 32-bit SHA-3 ASIP designs.

The rest of this paper is organized as follows. Section 2 elaborates SHA-3 algorithm and the reference Z-scale processor architecture. Section 3 describes the iterative design flow based on the *ASIP Designer* and Sect. 4 proposes two ASIP architectures namely operation-oriented ASIP and datapath-oriented ASIP based on profiling data of SHA-3 algorithm. Hardware implementation and evaluation results are presented and discussed in Sect. 5. Section 6 draws the conclusion.

2. Background

2.1 SHA-3 Algorithm

NIST launched the Next Generation Hash Function Calling Initiative in 2007. The goal was to replace the existing SHA-1, SHA-2 standards which have been used for many years and address new security needs. After several rounds of open competitions, the algorithm based on *Keecak spong* function family [21] became the final SHA-3 standard in 2012 [23].

The *sponge* function construction consists of three main steps: padding, absorbing and squeezing as in Fig. 1. It uses multi-rate padding to pad input message to be absorbed. Padded 1600-bit message is stored in an internal state which is organized as a $5*5*64$ three-dimensional array shown in Fig. 2. The index addressing order is $x \rightarrow y \rightarrow z$. After transformations, scrambled internal state is sequentially squeezed out to compose the desired length of digest. With different digest lengths, SHA-3 standard can be referred as *SHA-3(224)*, *SHA-3(256)*, *SHA-3(384)* and *SHA-3(512)*.

During the *spong* function construction for SHA-3, Keccak-f[1600] function implements the key permutations on internal state for 24 rounds. Each round consists of five steps called θ (Theta), ρ (Rho), π (Pi), χ (Chi) and ι (Iota). θ step *XORs* each state bit with parities of adjacent columns.

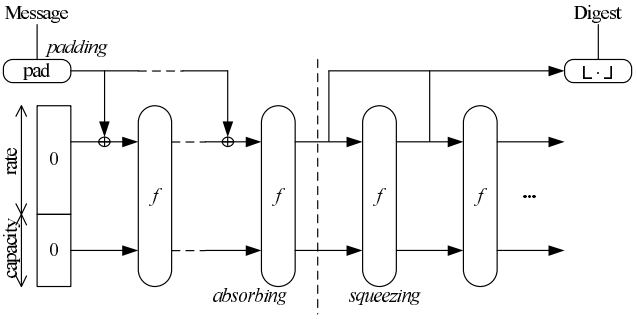


Fig. 1 SHA-3 algorithm structure

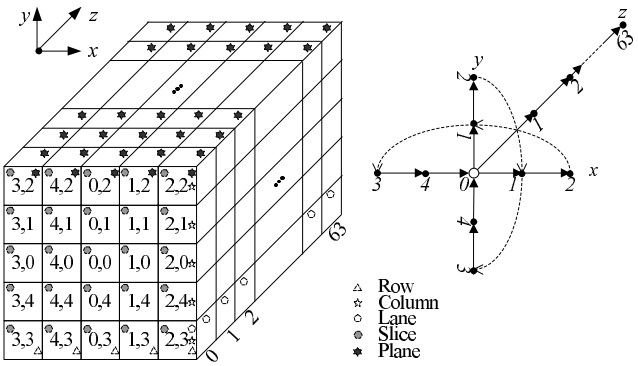


Fig. 2 SHA-3 state data layout

Algorithm 1 Keccak-f[1600] function

Require: state array **S**, round number n_r, rotation constant $ROTC$, round constant RC
Ensure: updated state array **S**
for $i_r = 0; i < n_r; i_r + +$ **do**
  $(\theta \cdot (\rho \cdot \pi) \cdot \chi \cdot \iota)$ (S, $RC[i_r]$)
end for
return S

θ:
  P[x] = S[x,0] $\oplus$ S[x,1] $\oplus$ S[x,2] $\oplus$ S[x,3] $\oplus$ S[x,4]
  Q[x] = P[$(x-1)$%5] $\oplus$ ROTL(P[$(x+1)$%5], 1)
  S[x,y] = S[x, y] $\oplus$ Q[x]
$\rho \cdot \pi$:
  T[y,$(2x+3y)$%5] = ROTL(S[x,y], $ROTC_{xy}$)
χ:
  S[x,y]=T[x,y] $\oplus$ (~(T[$(x+1)$%5,y]) & T[$(x+2y)$%5])
ι:
  S[0,0]=S[0,0] $\oplus$ $RC[i_r]$

Then ρ step rotates each lane based on its location. Next π step scrambles the state by rearranging positions of different lanes with variable lengths. Following χ step is the only non-linear step which improves algorithm security. It does a series of *AND*, *NOT* and *XOR* operations among different lanes. Finally, the ι step *XORs* a round constant with lane 0. The detailed definition of Keccak-f[1600] function is shown as algorithm 1. For all five steps, lane addressing indexes are modulo 5 to reside in valid range $[0, 4]$.

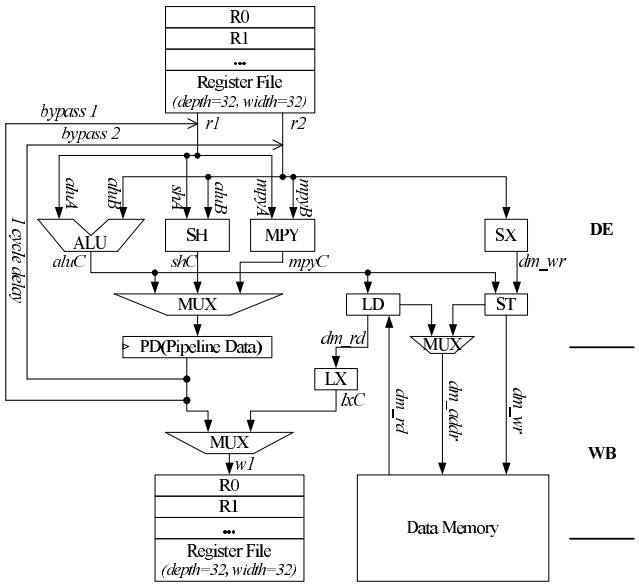


Fig. 3 Architecture of RISC-V model in ASIP Designer

2.2 Z-Scale Processor Architecture

RISC-V is an open Instruction Set Architecture(ISA) proposed to facilitate processor design for different domains from embedded devices to servers [24]. It supports both 32-bit and 64-bit data width with minimal base set of instructions and rich optional instruction extensions. Designers can choose to implement subsets of instructions with customized extensions to meet application requirements while reduce resource overhead and power consumption. It has drawn attentions in both academia and industry. *Z-scale* is a processor based on the *RV32IM* instruction set targeting embedded system domain [25].

Synopsys ASIP designer also ships an RISC-V model which is fully compatible with *Z-scale* processor specification. This model implements following architecture features:

(1) 32-bit wide data path with a single cycle ALU, shifter, multiplier and multi-cycle division/remainder unit.

(2) 32 entry central register file.

(3) Load/store architecture with supporting of 8, 16 and 32-bit memory transfers and an indexed addressing mode.

(4) Single-issue, in-order and 3 stage pipeline.

(5) 32-bit and 16-bit instructions.

The detailed micro-architecture is shown in Fig. 3. The model also provides a series of libraries such as *libc* and *libm* for rapid application porting and analysis. It can significantly save efforts for software mapping.

With efficient optimizations by *ASIP designer*, this model can outperform the standard *Z-scale* implementation with 20% *Dhrystone* performance improvement and 10.1% resource reduction [19].

All of above advantages make the model is a good start point for SHA-3 ASIP design exploration. And following analysis, design and implementation in this paper take it as the reference design.

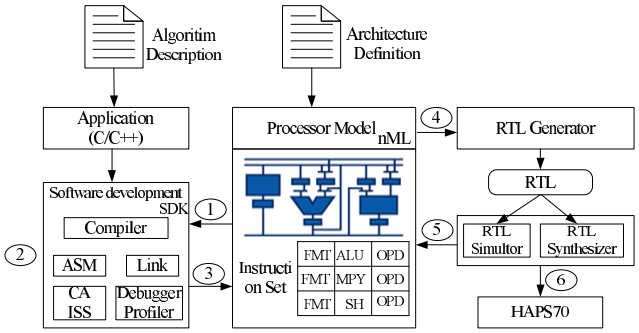


Fig. 4 ASIP design flow

3. Design Flow

ASIP Designer is a *Synopsys* tool for accelerating ASIP design and programmable accelerators. Based on a uniform *nML* [26] architecture description, it can generate an Software Development Kit (SDK) including an optimized compiler, cycle accurate simulator, assembler, linker, debugger and profiler for software development, produce efficient *RTL*, *UVM*-based environment and random test cases for hardware verification and implementation. It efficiently speeds up ASIP design. And the design flow in this paper is show as Fig. 4.

(1) Implement process model in *nML* based on architecture definition and export its *SDK* for later software development.

(2) Port application, verify functionality and do performance profiling to locate bottlenecks.

(3) Update architecture definition by extending instruction set and customizing datapath to tackle performance bottlenecks. Repeat steps (1) to (3) until there is no available space for software optimization.

(4) Generate RTL, perform functional verification, synthesize design and update architecture definition for power, performance and area (PPA) optimization. Repeat steps (1) to (4) until unit implementation meets hardware design targets.

(5) Map design to FPGA, run application and do a comprehensive comparison of performance, code space and hardware resource between implemented ASIPs and reference design.

4. Proposed ASIPs

This paper takes the algorithm implementation in *Keccak* Package as target application [27]. It is a standalone pure *C* implementation without any architecture optimization and look-up tables †. This exposes all computational complexity of SHA-3 algorithm and avoids interference caused by pre-optimization.

†KeccakCodePackage/Standalone/CompactFIPS202/Keccak-readable-and-compact.c

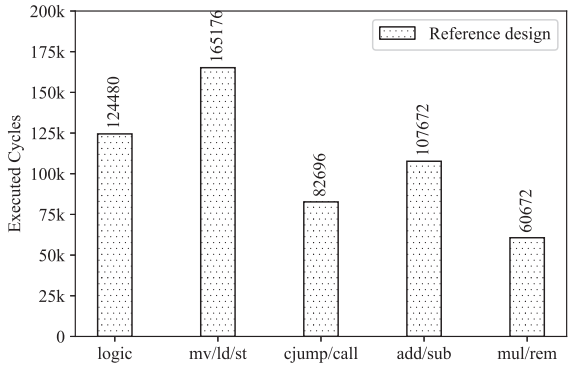


Fig. 5 Keccak-[1600] instruction cycle distribution on reference design

Table 1 Keccak-[1600] operation patterns on reference Z-scale

Operation patterns	Stpes	Description
$r = i_{i32}\%5$	All	Lane indexing
$r = (a_{i64} << b_{i32}) \mid (a_{i64} << (64 - b_{i32}))$	θ, ρ	Lane rotation
$r = (a_{i32} << 1) \oplus ((a_{i32}\&0x80)?0x71 : 0x00)$	*itoa*	Round constant
$loop(count_{i32})$ { *body*; }	All	Loop control

4.1 Operation-Oriented ASIP Architecture

4.1.1 Performance Profiling

With the out-of-the-box reference RISC-V model, it can quickly port the algorithm implementation and do initial profiling of the Keccak-f[1600] function. Instruction types and their executed cycle counts are the most interested targets which shows us the performance bottlenecks. Their distributions on reference design are show as Fig. 5.

The main part is data transfer instructions which is used for internal state management. And logical instructions including OR, XOR, NAND and 64-bit rotation are executed for transformation. It also requires nested loops to traverse all lanes of the internal state presented in comparison and jump instruction group. Multiply and modulo instructions are for lane indexing with a fixed step 5. Addition and subtraction instructions are basic operations for general programming also required for loop counter updating, lane index and indirect memory address calculation. Beyond basic instruction analysis, we have examined SHA-3 algorithm and summarized common time-consuming operation patterns for dedicated acceleration. They are shown in Table 1.

4.1.2 ISE Design

Operation-oriented ASIP targets accelerating operations on existing datapath with instruction extensions in Table 2.

remv takes advantage of predictable input rage and do modulo within single cycle instead of multi-cycle division instruction for lane index calculation. To accelerate 64-bit rotation which is inefficiently emulated by a series of shift, AND, OR operations on 32-bit processors, this paper extends a 64-bit rotation auxiliary register. *pack* concatenate

Table 2 Instruction extensions for OASIP

Instruction	Description
*remv rd, rs*0, #*imm*	tmp=rs0+#imm; rd=tmp-5 if *tmp*∈[5, 9]; rd=tmp+10 if *tmp*∈[−9, −6]; rd=tmp+10 if *tmp*∈[−5, −1]; rd=tmp+10 if *tmp*∈[0, 5); rd=tmp+10 if *tmp*∈[0, 5);
*hmask rd, rs*0	rd=(rs/2)%64;
*pack rs*0, *rs*1	aux[63:32]=rs0; aux64[31:0]=rs1;
*rotl rsd, rs*0	aux=left_rotate(aux,rs0); rd=aux[63:32];
extlw rd	rd=aux[31:0];

*lfsr rd, rs*0	tmp=rs<<1;
	mask=(rs&0x80)?0x70:0x00;
	d=tmp⊕mask;

*doloop rs*0, #*LE*	hardware loop *rs*0 times to #*LE*;

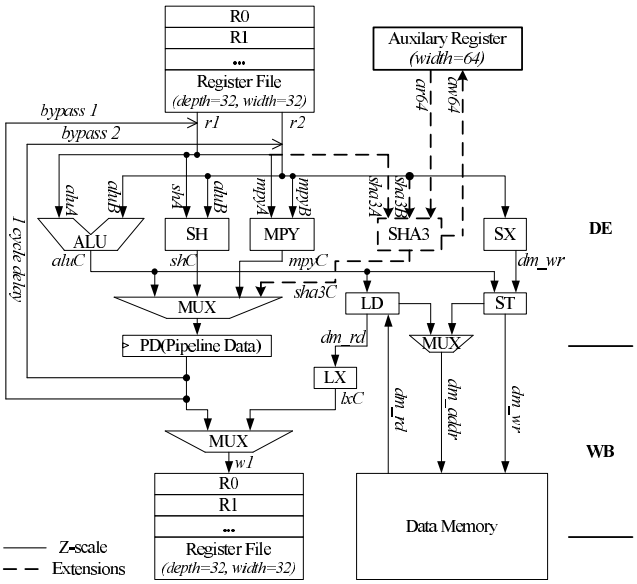


Fig. 6 Operation-oriented SHA-3 ASIP architeture

two 32-bit data into it. *rotl* does 64-bit left rotation and stores high 32-bit result back to a general register. And the low 32-bit result can be extracted to another general register with *extlw* later. *hmask* and *lfsr* merges common operations for index updating in π and round constant generation in ι.

All five steps take 2 to 3 levels of loops to traverse each lane during transformation. This is implemented with 2-cycle delay conditional jump instructions. Without enough instructions to fill these slots, it will significant increase the performance overhead. Hardware zero-overhead loop can efficiently eliminate delays of loops with fixed counts for high loop performance. This paper introduces two level hardware zero-overhead loops and software unrolling on the third level small loop to balance performance, hardware resource and code size.

By merging operations and providing new operations in hardware, it can both improve performance and reduce code size. Figure 6 illustrates architecture of the proposed OASIP.

4.2 Datapath-Oriented ASIP Architecture

4.2.1 Performance Profiling

Based on the optimized OASIP, we have updated the algorithm implementation and done re-profiling. The new distribution of different instruction executed cycles are show as Fig. 7.

It shows that OASIP can accelerate time-consuming operations such as modulus, 64-bit rotations etc. with dedicated ISE. But it has limited contributions to solve following problems:

(1) **Dominant *load/store* operations:** The state space of SHA-3 algorithm is $5 * 5 * 64$ bits while reference model only has a 32-bit register file with depth of 32. Compiler has to insert *load/store* instructions to exchange data between registers and memory for register spilling.

(2) **Heavy 32-bit logical operations:** Each 64-bit logical operation on 32-bit data path requires two 32-bit logical operations.

4.2.2 Data Level Parallelism Exploitation

To tackle bottleneck (1) described in Sect. 4.1.1, this paper extends a 64-bit auxiliary register file with depth of 6 to extend life cycle of state data in each step. The entry number mainly comes from step χ which requires 5 for state column data and 1 for operation result. As a 64-bit operand contains two 32-bit sub-words, this can ideally reduce instructions by a factor of 2 comparing to common instructions with 32-bit operands. It's also known as data level parallelism.

4.2.3 ISE Refinement

As there are 64-bit registers, it's natural to extend logical operation like AND, NOT, XOR etc. with 64-bit operands. To speed up exchanging data between auxiliary registers and memory, a special register relative addressing mode with step 5 is introduced. And it also requires a new data movement instruction among auxiliary registers for data reorganizing.

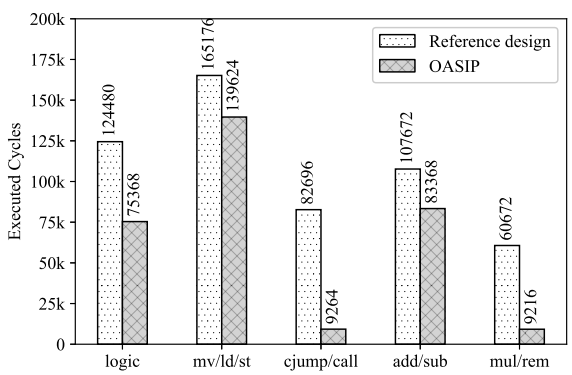


Fig. 7 Keccak-[1600] instruction cycle distribution on OASIP

4.2.4 Instruction Level Parallelism Exploitation

Further algorithm analysis shows that it's common to traverse all lanes for state transformation while logical operations and memory accesses with different lanes are independent. This provides us an opportunity to paralyze different type instructions. By grouping 64-bit logical operations and memory accesses on extended auxiliary register as operation bundles, this paper introduces *VLIW* instructions to exploit instruction level parallelism for better performance and code size.

Above data level and instruction level parallelism exploration leads to a datapath-oriented ASIP with ISE defined in Table 3. And Fig. 8 illustrates architecture of the proposed DASIP.

Table 3 Instruction extensions for DASIP

Instruction	Description
*hmask rd, rs*0	rd=rs[8:1]; //rd=(rs0>>1)%64
*xor ard, ars*0, *ars*1	ard= XOR(ars0, ars1);
*rotl ard, ars*0, *rs*1	ard= ROTL(ars0, rs1)
*nandxor ard, ars*0, *ars*1	ard= XOR(ard,AND(NOT(ars0),ars1))
*mvar ard, ars*0	ard= ars0;
*ldlw*32*s*5*i ard*, *rs*0, *rs*1, *#imm*	ard[31:0]=mem32[rs0+rs1*5+#imm];
*ldhw*32*s*5*i ard*, *rs*0, *rs*1, *#imm*	ard[63:32]=mem32[rs0+rs1*5+#imm];
*stlw*32*s*5*i ard*, *rs*0, *rs*1, *#imm*	mem32[rs0+rs1*5+#imm]=ard[31:0];
*sthw*32*s*5*i ard*, *rs*0, *rs*1, *#imm*	mem32[rs0+rs1*5+#imm]=ard[63:32];
*ldlw*32*m*5*i ard*, *rs*0, *rs*1, *#imm*	ard[31:0]=mem32[rs0+(rs1+#imm)%5];
*ldhw*32*m*5*i ard*, *rs*0, *rs*1, *#imm*	ard[63:32]=mem32[rs0+(rs1+#imm)%5];
*stlw*32*m*5*i ard*, *rs*0, *rs*1, *#imm*	mem32[rs0+(rs1+#imm)%5]=ard[31:0];
*sthw*32*m*5*i ard*, *rs*0, *rs*1, *#imm*	mem32[rs0+(rs1+#imm)%5]=ard[63:32];
*ldhw*32*s*5*i*; *xor* *sthw*32*s*5*i*; *xor*	ILP for θ step;
*ldhw*32*s*5*i*; *rotl* *sthw*32*s*5*i*; *rotl*	ILP for θ step;
*ldhw*32*s*5*i*; *mvar* *sthw*32*s*5*i*; *mvar*	ILP for ρ step;
*ldhw*32*s*5*i*; *nandxor* *sthw*32*s*5*i*; *nandxor*	ILP for χ step;

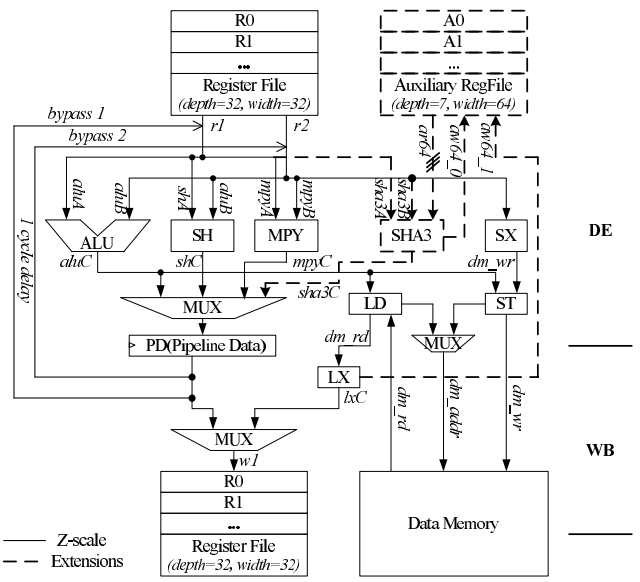


Fig. 8 Datapath-oriented SHA-3 ASIP architecture

Table 4 SHA-3 ASIP evaluation results on FPGA targeting 100MHz

Implementation	Performance (Bytes/cycle)*10^{-3}	Area Slices	Code Size Bytes	Res. Efficiency (Bytes/(cycles*slices))*10^{-6}	Res. Eff. Speedup
Z-scale	2.01	747.5	1248	2.69	-
OASIP	3.43(171%)	981(131%)	1066(85.4%)	3.50(130%)	3.68
DASIP	7.67(382%)	1522(204%)	912(73.1%)	5.04(187%)	5.30
MIPS	4.49	6074	3529	0.74	-
MIPS+Native ISE	5.61(125%)	6595(109%)	3320(92.4%)	0.85(115%)	-
MIPS+Co-processor ISE	7.25	7643(126%)	3292	0.95(128%)	1.00
LEON3	1.44	7902	20000	0.18	-
LEON3+ISE	2.71(188%)	8648(109%)	18100(91.5%)	0.31(123%)	-

Table 5 SHA-3 ASIP evaluation results on ASIC targeting maximum frequencies

Implementation	Fmax (MHz)	Power(Static+Dynamic) (mW)	Max Throughput (MBytes/second)	Max TP. Speedup	Power Efficiency (MBytes/mJ)	Power Eff. Speedup
Z-scale	590	3.92(0.43+3.49)	1.19	1.00	0.30	1.00
OASIP	590	4.77(0.51+4.26)	2.03	1.71	0.42	1.40
DASIP	560	3.66(0.72+2.94)	4.29	3.62	1.17	3.88

5. Evaluation

In this chapter, reference design, OASIP and DASIP are implemented in hardware and mapped to *Synopsys* HAPS-70 platform equipped with a Virtex-7 XC7V2000T FPGA chip at 100MHz. It also runs optimized SHA-3 algorithm implementations for functional verification and evaluation. This paper evaluates SHA-3 hashing performance, hardware area, software code size and resource efficiency. Performance refers to hash speed and defined as data bytes hashed per cycle. It uses a built-in hardware counter to collect cycles running on HAPS-70. As padding overhead is dependent on message length, the counter starts after it to remove the interface. Xilinx FPGA slice number in synthesis report is taken to quantify the area. And code size includes both text and static data sections. To quantify the efficiency between different ASIP schemes of existing works, this paper uses resource efficiency which is defined as bytes of message hashed per second and slice. Table 4 shows detail results of proposed ASIPs with reference design and relevant existing works on FGPA.

It shows that both ASIPs effectively improve the performance of SHA-3 algorithm over the reference design. The OASIP gains 71% performance improvement with hardware acceleration of time-consuming operations. The 31% area overhead is mainly comes from the 64-bit auxiliary register and shifter for 64-bit rotation support. It also gets 14.6% code size reduction which is benefit from operation merging, less instructions of 64-bit rotation than software emulation with 32-bit instructions only and zero-overhead loops for program flow control. While DASIP significantly improves performance by 282% with 104% area overhead including extended register file and customized datapath. Its code size reduction is 26.9% with advantages of data and instruction level parallelism exploitation. The resource overhead of both proposed ASIP looks relative large, this is because the reference design is an extremely lightweight RISC-V processor with only basic subset instructions. It's expected to have lower overall overhead in IoT systems with more RISC-V instruction subsets implemented. And comparing with relevant researches, the resource efficiency speedup varies from 3.68X to 5.30X on FPGA.

Meanwhile, this paper does further evaluation of maximum throughput and average power consumption based on *Synopsys ASIC Design Reference Methodology* [29] as these metrics are crucial for resource and power sensitive embedded processor design for IoT systems. Different version of Z-scale implementations are synthesized with TSMC28HPM technology for maximum frequency, and power evaluation is done with *Synopsys Power Compiler* based on SAIF(Switching Activity Interchange Format) generated in gate-level simulation of SHA-3 algorithm. Table 5 shows the detail results on ASIC [†].

OASIP adds simple *compare*, *add*, *shift* and *select* operations which are not on the critical timing path, while comparing with simple multiplier in reference Z-scale, MAC(Multiply-Add) operation in DASIP adds a chained adder after multiplier and slightly decreases achievable maximum frequency from 590MHz to 560MHz. Nevertheless, the overall throughputs of OASIP and DASIP still outperform reference Z-scale. Both OASIP and DASIP consume more static power as they increase the design logic. OASIP increases the dynamic power as it has to pack two 32-bit word to a 64-bit word for each rotation and convert the result back to two 32-bit words afterwards, while DASIP does 64-bit operations on extended 64-bit auxiliary register files as much as possible, and the dedicated ILPs can also avoid redundant operations which significantly improves its power efficiency. And comparing to reference Z-scale, the speedup of achievable maximum throughputs are 1.71X and 3.62XX with 1.40X and 3.88X better power efficiencies on ASIC.

[†]Research [16] and [17] only evaluate performance on FPGA prototypes without any evaluation of power.

All for these advantages make them more practical for integrations in modern IoT systems.

6. Conclusion

This paper explores SHA-3 ASIP design for IoT systems based on a lightweight embedded RISC-V processor model named *Z-scale*. The whole work is done with a fast ASIP iterative design flow in *Synopsys ASIP Designer*. It leads to two efficient SHA-3 ASIP designs. The first operation-oriented ASIP (OASIP) is proposed for accelerating time-consuming operations with minimal impact of existing datapath. And the other datapath-oriented ASIP (DASIP) targets more advance data and instruction level parallelism exploitation for high performance with extended registers and customized datapath. Both ASIPs are hardware implemented and mapped to HASP-70 for evaluation. Comparing to the reference design, there are 71% and 282% performance improvements, 14.6% and 26.9% code size reduction with 31% and 104% resource overhead. And their resource efficiencies also outperform known best work by a factor from 3.68X to 5.30X. Further evaluation on ASIC shows that OASIP and DASIP are 1.71X and 3.62X more maximum throughput efficient with 1.40X and 3.88X more power efficient than reference design for SHA-3 algorithm support. Technologies proposed in this paper are based on a typical 32-bit embedded processor which can facilitate either low-cost or high performance SHA-3 integrations for other 32-bit IoT systems.

References

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," Future Generation Computer Systems, vol.29, no.7, pp.1645–1660, 2013.

[2] P. Kocher, R. Lee, G. McGraw, A. Raghunathan, and S. Ravi, "Security as a new dimension in embedded system design," Proc. 41st annual Design Automation Conference, pp.753–760, 2004.

[3] S. Ravi, A. Raghunathan, P. Kocher, and S. Hattangady, "Security in embedded systems: Design challenges," ACM Trans. Embedded Computing Systems (TECS), vol.3, no.3, pp.461–491, 2004.

[4] M.U. Farooq, M. Waseem, A. Khairi, and S. Mazhar, "A critical analysis on the security concerns of internet of things (IoT)," Int. J. Computer Applications, vol.111, no.7, 2015.

[5] X. Wang, D. Feng, X. Lai, and H. Yu, "Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD," IACR Cryptology ePrint Archive, 2004.

[6] M. Szydlo, "SHA-1 collisions can be found in 2^63 operations," CryptoBytes Technical Newsletter, 2005.

[7] M. Sundal and R. Chaves, "Efficient FPGA Implementation of the SHA-3 Hash Function," 2017 IEEE Computer Society Annual Symposium on VLSI, pp.86–91, 2017.

[8] E. Homsirikamol, M. Rogawski, and K. Gaj, "Throughput vs. Area Trade-offs in High-Speed Architectures of Five Round 3 SHA-3 Candidates Implemented Using Xilinx and Altera FPGAs," CHES 2011, pp.491–506, 2011.

[9] M. Rao, T. Newe, and I. Grout, "Secure hash algorithm-3 (SHA-3) implementation on Xilinx FPGAs, suitable for IoT applications," 8th International Conference on Sensing Technology, 2014.

[10] D. Kundi and A. Zziz, "A low-power SHA-3 designs using embedded digital signal processing slice on FPGA," Computers & Electrical Engineering, Supplement C, pp.138–152, 2016.

[11] O. Schliebusch, H. Meyr, and R. Leupers, Optimized ASIP synthesis from architecture description language models, Springer Science & Business Media, 2007.

[12] D. Liu, Embedded DSP Processor Design, Application Specific Instruction-set Processors, MA: Elsevier Morgan Kaufmann, 2008.

[13] A. Hoffmann, O. Schliebusch, A. Nohl, G. Braun, O. Wahlen, and H. Meyr, "A methodology for the design of application specific instruction set processors (ASIP) using the machine description language LISA," IEEE/ACM International Conference on Computer Aided Design, 2011, pp.625–630, 2011.

[14] D. Lanneer, J. Van Praet, A. Kifli, K. Schoofs, W. Geurts, F. Thoen, and G. Goossens, "CHESS: Retargetable code generation for embedded DSP processors," Code generation for embedded processors, pp.85–102, Springer, 2002.

[15] J.H.F. Constantin, A.P. Burg, and F.K. Gürkaynak, "Instruction set extensions for cryptographic hash functions on a microcontroller architecture," ASAP, pp.117–124, 2012.

[16] Y. Wang, Y. Shi, C. Wang, and Y. Ha, "FPGA-based SHA-3 acceleration on a 32-bit processor via instruction set extension," EDSSC, pp.305–308, 2015.

[17] M.A. Elmohr, M.A. Saleh, A.S. Eissa, K.E. Ahmed, and M.M. Farag, "Hardware implementation of a SHA-3 application-specific instruction set processor," 28th International Conference on Microelectronics (ICM), IEEE, pp.109–112, 2016.

[18] Y. Lee, A. Ou, and A. Magyar, "Z-scale: Tiny 32-bit RISC-V Systems," 2nd RISC-V workshop, 2015.

[19] S. Cox and D. Taussig, "Extending RISC-V for Application-Specific Requirments," 5th RISC-V workshop, 2016.

[20] Synopsys, ASIP Designer, https://www.synopsys.com/dw/ipdir.php?ds=asip-designer.

[21] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, The Keccak reference, SHA-3 competition (round 3), 2011.

[22] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer, "Keccak implementation overview," SHA-3 competition (round 3), 2012.

[23] NIST, SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, 2015.

[24] A.S. Waterman, "Design of the RISC-V Instruction Set Architecture," Tech. Rep., University of California, Berkeley, 2016.

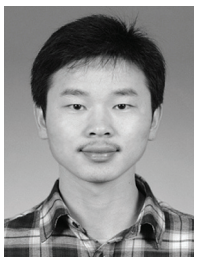
[25] A. Waterman, Y. Lee, D. Patterson, and K. Asanov, "The RISC-V Instruction Set Manual, Volume I: User-Level ISA," Tech. Rep., University of California, Berkeley, 2016. Version 2.1.

[26] A. Fauth, J. Van Praet, and M. Freericks, "Describing instruction set processors using nML," European Design and Test Conference, pp.503–507, 1995.

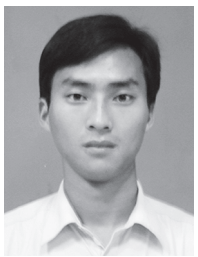
[27] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer, Keccak Code Package, https://github.com/gvanas/KeccakCodePackage.

[28] D.J. Bernstein and T. Lange (editors), "eBACS: ECRYPT Benchmarking of Cryptographic Systems," https://bench.cr.yp.to.

[29] Synopsys, Reference Methodology Retrieval System, https://solvnet.synopsys.com/rmgen.



Jinli Rao was born in 1987. He received the B.S. and M.S. degrees in Microelectronics and Solid State Electronics from Huazhong University of Science and Technology, China in 2009 and 2012 respectively. Currently He is a Ph.D. candidate in Huazhong University of Science and Technology, China. His research interests include information security, computer architecture and VLSI design.



Tianyong Ao was born in 1981. He received B.S. degree from Huazhong Normal University in 2004, M.S. degree from Henan University in 2009 and Ph.D. degree in Huazhong University of Science and Technology. Now he is a lecturer in Henan University. His research interests include information security and computer architecture.

Shu Xu received B.S. degree in mathematics from Peking university in 1984 and Ph.D. degree in cryptology from Zhengzhou Information Science and Technology Institute in 2008 respectively. Now he is a research fellow in Science and Technology on Information Assurance Laboratory. His research interests include computer architecture, cryptology and information security. He has been awarded 1 time the special grade prize, 1 time the first prize and 2 times the second prize of national award for Science and Technology Progress, 7 times the first prize of ministerial awards, 7 times the second prize of ministerial awards.

Kui Dai received B.S. degree in information science and technology from Harbin University of Technology and the Ph.D. degrees in Computer Science and Technology from National University of Defense Technology, China in 1989 and 1994 respectively. He joined the National University of Defense Technology in 1995. Since 2008 he was a professor and doctoral supervisor in Huazhong University of Science and Technology. Now he is a professor in the Institute of National Network Security and Information, Peking University. He contributes to the computer system architecture, high performance processor system, embedded system and information security for almost 25 years.

Xuecheng Zou received the B.S. M.S. and Ph.D. degrees, all in electrical engineering, from Huazhong Unversity of Science and Technology, China in 1985, 1988 and 1995 respectively. He is currently a professor and doctoral supervisor in School of Optical and Electronic Information, Huazhong University of Science and Technology. His research interests span microelectronics and solid state electronics, information security, VLSI design.