

Hashchip: A shared-resource multi-hash function processor architecture on FPGA

T.S. Ganesh^{a,*}, M.T. Frederick^{a,2}, T.S.B. Sudarshan^b, A.K. Somani^a

^a*Dependable Computing and Networking Laboratory, Iowa State University, Ames, IA - 50011, USA*

^b*Computer Science and Information Systems, Birla Institute of Technology and Science, Pilani - 333031, India*

Abstract

The ubiquitous presence of mobile devices and the demand for better performance and efficiency have motivated research into embedded implementations of cryptography algorithms. In this paper, we propose and explore multiple architectural options for the *HashChip*. The *HashChip* is a hardware architecture aimed at providing a unified solution to the task of message hashing with integrated message padding by aggressive exploitation of similarities in the structure of three commercially popular hash algorithms, namely, MD5, SHA1 and RIPEMD160. A generic approach to prototype digital systems on the Xilinx Virtex 2P embedded FPGA platform is presented and utilized for evaluating the *HashChip* architectures. The performance of the architectures is studied and evaluated for different design metrics. Throughputs in the range of 200–330 Mbps are obtained on the Xilinx Virtex2P FPGA depending on the input message size and algorithm choice.

© 2006 Elsevier B.V. All rights reserved.

Keywords: Hash algorithms; Hardware architecture; FPGAs; Prototyping

1. Introduction

The importance of information security cannot be underestimated in an age where the Internet is emerging as a tool of commerce. An essential aspect of secure communication over any type of network is cryptography. Cryptography algorithms fulfill specific information security requirements such as authentication, confidentiality, integrity and non-repudiation. Hash algorithms are a class of cryptographic primitives which ensure integrity and authentication. They are further classified as MDCs (manipulation detection codes) and MACs (message authentication codes). While MDCs are able to ensure integrity only, MACs can perform authentication also [1]. The most popular MDCs are MD5 [2], SHA1 [3] and RIPEMD160 [4].

Software implementations of cryptography algorithms are straightforward. However, they are usually slow, vulnerable and can be easily compromised. Hardware implementations are intrinsically more physically secure, offer higher throughput and consume lesser power. Implementations on reconfigurable hardware offer a host of other advantages including algorithm agility, algorithm uploadability, architecture efficiency and resource efficiency.

Hardware implementation of multiple hash algorithms on a single architecture has been an actively explored topic [5–8]. Preprocessing in the form of message padding is an important aspect of hash algorithms which has been largely ignored by most of the previous implementations. In this paper, we present unified architectures for three different MDCs, and their mapping on to a reconfigurable platform. Compared to previous implementations, our architectures offer many advantages including integration of message padding functionality in the main core, better resource usage efficiency despite integrating higher amounts of functionality [7] and higher throughput even when padding latency is considered [5].

*Corresponding author.

E-mail address: ganesh@iastate.edu (T.S. Ganesh).

¹The research work reported in this paper was done in part by the author at Birla Institute of Technology and Science, Pilani, India.

²The research work reported in this paper is partially supported by the Jerry R. Junkins endowment at Iowa State University, USA.

The remainder of this paper is structured as follows. A discussion of the structure of one way hash algorithms is presented in Section 2. Section 3 elucidates the datapath design approach for *HashChip*. Section 4 presents a generic approach to prototype digital systems on the Xilinx Virtex 2P embedded FPGA platform and also details how the prototyping of the designed architectures is performed. Section 5 presents the performance analysis and the paper concludes in Section 6.

2. Message digest generation using manipulation detection codes

Manipulation detection codes are usually non-keyed hash functions which are designed as iterative processes hashing arbitrary input lengths by processing successive fixed size blocks of the input. The size of each processed block in MD5, SHA1 and RIPEMD160 is 512 bits. The nature of preprocessing in these algorithms is detailed in Fig. 1. Minor differences amongst the considered algorithms are explained below.

The message of bit length K is padded to ensure that its length in bits is 64 short of an integral multiple of 512. Padding is always performed even if the length of the message is already congruent to 448 modulo 512. Padding consists of a single 1-bit followed by the necessary number of 0-bits. A 64-bit message size representation is padded at the end of the message in little-endian format for MD5 and RIPEMD160, and in big-endian for SHA1. A message block which is an integral multiple of 512 bits in length is obtained after the completion of the padding. Each 512-bit block is used once in the complete processing by the chain of compression functions. The digest size is 160 bits for SHA1 and RIPEMD160, and 128 bits for MD5 [1].

Fig. 2 brings out the structure of the algorithms' compression functions. Each 512-bit block of the message is subjected to four rounds of iterations in MD5 and SHA1, while there are two parallel instances of five rounds in RIPEMD160. Each round in MD5 and RIPEMD160

consists of 16 elementary operations, while there are 20 in SHA1. To obtain the chaining variable at the end of a single compression function iteration, the final result of the processing of the rounds is added to the chaining variable input at the beginning. This addition is performed on corresponding words in both MD5 and SHA1. For RIPEMD, it is done according to the ordering in Fig. 2.

The elementary operation of each hash algorithm is shown in Fig. 3. A, B, C, D and E are words of 32 bits each, but E is missing in MD5 since the message digest produced by it is only 128 bits long. The additions involved are modulo 2^{32} . In all three algorithms, a 32 bit part of the message ($X[k], W_t, X_i$) is added to a constant ($T[i], K_t, K_j$) during each iteration. Another similarity is that B, C and D are given as inputs to a primitive function block in each step. The functions implemented on these inputs vary from round to round and algorithm to algorithm. The elementary operations of MD5 and RIPEMD160 directly use a 32 bit word from the 16 message words available but SHA1 expands the initially available 16 words to 80 words on the basis of the following transformation and uses them once in each of the compression function iterations.

$$W_t = M_t \quad [0 \leq t \leq 15],$$

$$S^1(W_{t-16} \oplus W_{t-14} \oplus W_{t-8} \oplus W_{t-3}) \quad [16 \leq t \leq 79].$$

M_t represents the t th word of the sixteen 32-bit words available from the current 512 bit block, and S^1 represents circular left shift by one place of its argument.

3. The HashChip architecture

The three unified architectures presented here are loosely based upon the architecture proposed by the authors in [9,8]. In comparison to [7], the architecture in [9,8] is clearly better at exploiting similarities in the shared algorithms to achieve optimal resource usage, particularly in the compression block of the design. Using a similar core as the base point for the present set of architectures retains this advantage. We perform aggressive optimization by

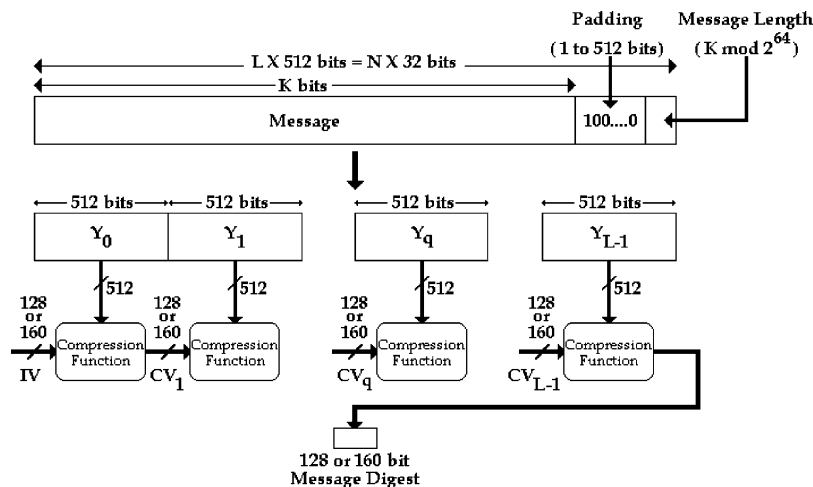


Fig. 1. Preprocessing steps in MD5, SHA1, and RIPEMD160.

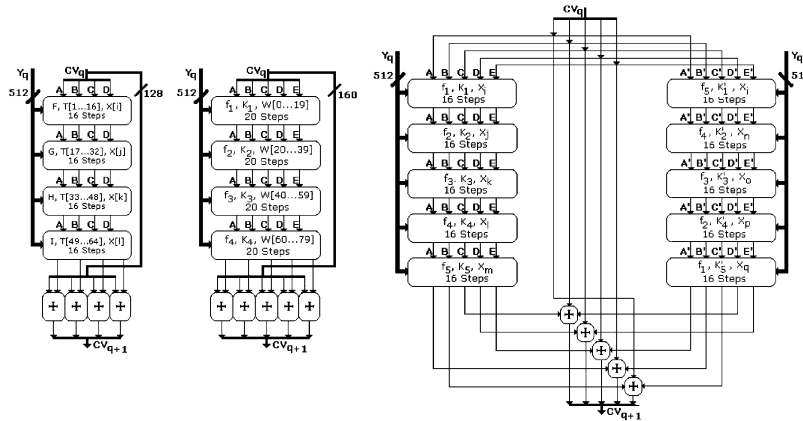


Fig. 2. Compression functions (left to right) MD5, SHA1, RIPEMD160.

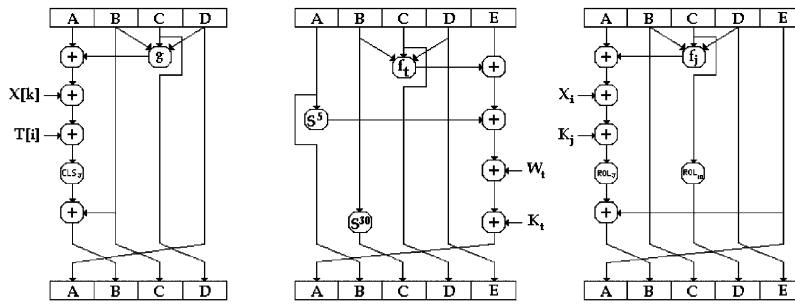


Fig. 3. Elementary operations (left to right) MD5, SHA1, and RIPEMD160.

removing redundant RAM blocks with negligible performance penalty. The critical paths are modified to accommodate lower cycle times, and enable operation at higher frequencies. These modifications result in a better performance in almost all respects in comparison to [9].

The architectures assume that the message to be hashed is already available in a byte organized memory bank, which supplies data in a synchronous manner depending on the validity of a data request signal given as input to it. The end of the message is signified by a sentinel byte, though the architecture can be easily reconfigured to respond to a special signal indicating this. Another input decides the algorithm to be applied to the data to hash. The outputs of the system include the 160 bit final digest, a signal indicating whether the data available as the final digest output is valid or not and a signal to indicate whether there is an internal error or an invalid algorithm request. A holistic view of the designed datapath is presented in the following subsection.

3.1. The HashChip Datapath

The six major components of the HashChip datapath are presented in Fig. 4. The Padder & Memory Block handles the system’s interface to the external memory bank. It

stores the message words and the various constants required during the different iterations and also ensures that the algorithm starts processing as soon as the minimum requirement of 64 bytes are input. It is necessary to duplicate some of the components from the main compressor block to the parallel compressor block because RIPEMD has two concurrently running rounds, even though MD5 and SHA1 have only one chain of rounds in their compression function. The chaining variable updation block handles the chaining variables update in a generic manner at the end of the processing of each compression function iteration. A digest generation block ensures that the chaining variable value at the end of all the iterations of the compression functions is transferred in the proper Endian format as the final digest. The validity status of the chaining variable is generated in this block. The microcode control unit issues different control signals needed in the datapath. The following subsections detail each of the datapath components.

3.2. Padding block design

The requirements for the padding block indicate that an FSM (finite state machine)-based design approach must be followed. The input is taken a byte at a time,

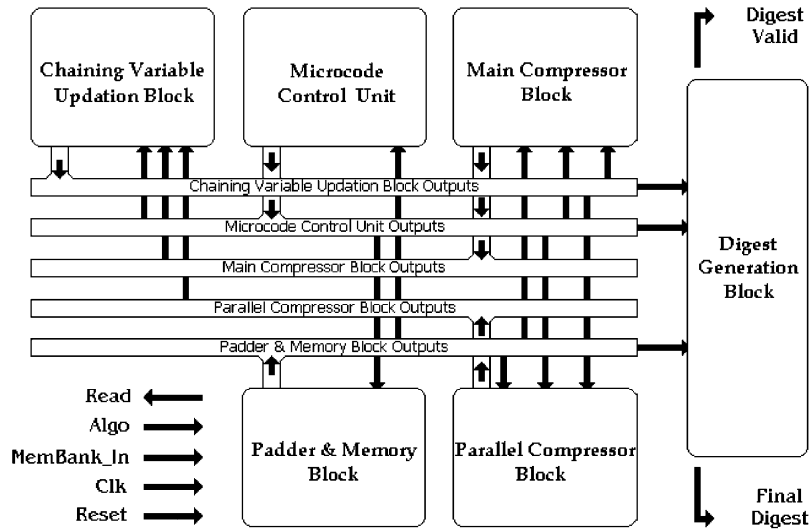


Fig. 4. HashChip datapath block diagram.

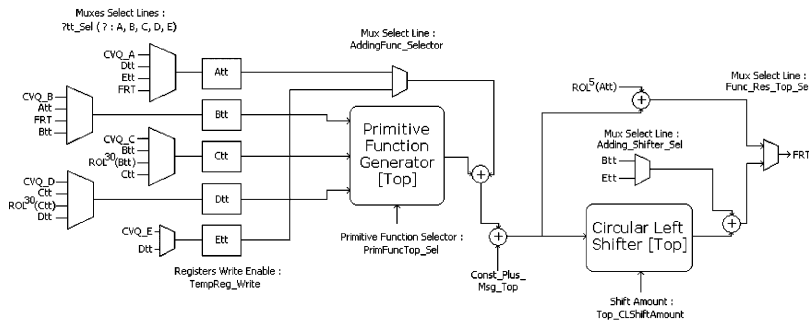


Fig. 5. Main compressor block.

but not immediately transferred for further processing. The sequencing controller must check for the sentinel byte and also start the padding process when required. These indicate that a single 512-bit block cannot be fed within 64 cycles. It would be possible to design a sequencer for feeding data blocks in lesser number of cycles than the present cost, but this would mean a loss in the generic nature of the padding block. The advantage of the designed padding block is that it can be used for other algorithms also, albeit with minor modifications.

3.3. Memory block design

The convoluted method of selecting the message word representation in SHA1 ensures that the register file to be used in the memory block is not a straightforward dual ported RAM, but should also double up as a type of linear feedback shift register, depending on the algorithm. The memory block also contains a dual ported ROM block for storing the various constants needed during the different iterations.

3.4. Compressor blocks design

The elementary operations outlined in Fig. 3 are implemented by the *main compressor block* (Fig. 5) and the *parallel compressor block* (Fig. 6). The multiplexers placed before the registers take care of the various possibilities with which the registers could be updated. The *parallel compressor block* is not as complex as the *main compressor block* since this segment of the architecture is devoted entirely to the processing of RIPEMD160 only. Depending on the algorithm, step and round, one out of the seven distinct combinational primitive functions implemented on the *B*, *C* and *D* registers is chosen. The output of this primitive function block is added to either *A* (for MD5 and RIPEMD160) or *E* (for SHA1). The result of this step is added to the output of the memory block in all the three algorithms. For generating *FRT*, the output from the previous step is subjected to a variable circular left shift and added to *B* in MD5 and *E* in RIPEMD160, while for SHA1, the output is simply added to the contents of *A* rotated to the left by five bits.

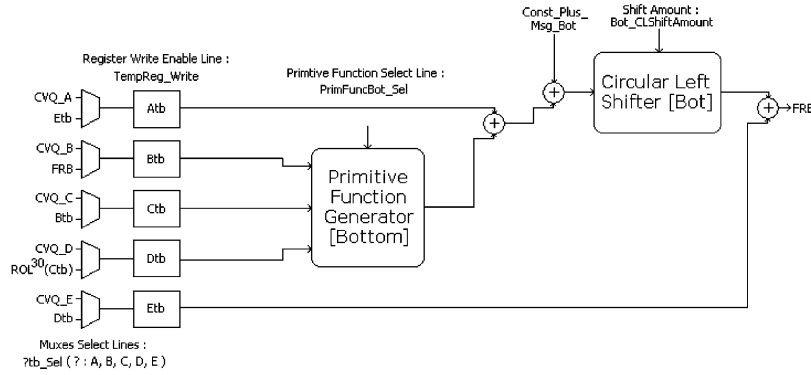


Fig. 6. Parallel compressor block.

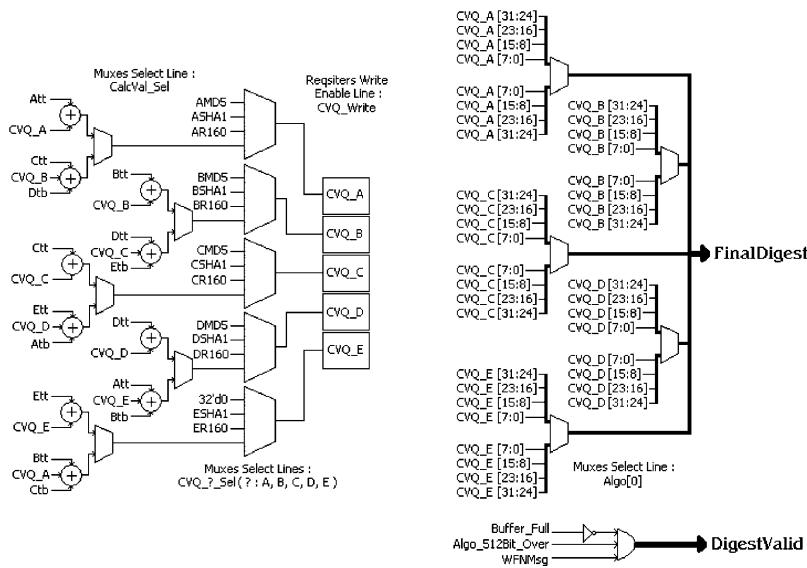


Fig. 7. Chaining variable update block (left) and digest generation block (right).

3.5. Chaining variable update block

Fig. 7 shows the structure of the *Chaining Variable (CV) updation block*. It implements the adders shown in the compression functions (Fig. 2). The registers need to be loaded with either the initializing vector of the requested algorithm or the chaining variable from the previous iteration, depending on whether it is the first iteration of the first 512 bit block or that of an intermediate 512 bit block (Fig. 1). The chaining variables need to be written only once every 64 cycles in MD5, while it is 80 cycles for SHA1 and RIPEMD160.

3.6. Digest generation block and microcode unit

The *digest generation block* is shown in Fig. 7. The multiplexer arrangement performs the conversion of the data in the chaining variable registers to the appropriate endian format. The *DigestValid* signal indicates the completion of all the compression function iterations for

all the message blocks. The micro-programmed control unit provides the control signals needed at various points in the datapath. It also generates the *Error* signal which identifies whether there is an invalid algorithm request or an internal error in which the system steps into an erroneous microcode ROM address or any similar invalid state, or the system tries to start operations without being properly reset.

3.7. Critical path identification and architecture modifications

Initial simulations revealed that the throughput of the system was severely constrained due to the nature of the padding block, which could accept only a byte per cycle. While MD5 requires 64 iterations, SHA1 and RIPEMD160 require 80 iterations for each 512 bit block. Since the hashing core remains idle for quite a number of cycles before the padding block is able to supply it with the new message block, an iteration can be made to span multiple cycles and the cycle time can be reduced. To identify the

register insertion points, an analysis of the synthesis results and the critical path is made. The critical path (through the compressor blocks) is split into multiple chunks and registers are inserted at strategic locations in order to reduce the cycle time and balance the amount of logic between them. This entails modifications in the control unit also. Two such architectures are developed and their performance is analyzed.

4. Prototyping the HashChip

The HashChip is implemented as part of an embedded system using the Virtex II Pro and its associated on-die PowerPC Microcontroller. This has been done to facilitate design, testing, and end-use for security applications. A simplified block diagram of the HashChip architecture and how it is integrated with the rest of the embedded system is shown in Fig. 8. The primary components of the system include the PowerPC 405, the block random access memories (BRAMs), the processor local bus (PLB), the on-chip peripheral bus (OPB), the HashChip input memory, and the OPB memory interface controller. The input memory is a shared resource between the processor and the HashChip peripheral to facilitate loading the input into memory from the processor side, and the consumption of the input from the HashChip side. The processor accesses the memory contents via the OPB bus and memory controller, and has full read/write capability. The HashChip accesses the memory directly through an 8-bit interface and has read-only capability, as it only consumes the data to generate a digest. The HashChip itself is a slave peripheral in the system, and the OPB bus provides access to its configuration and status registers to the processor. The register set includes the following:

- STATUS: Stores the counter overflow, valid, and error flags.

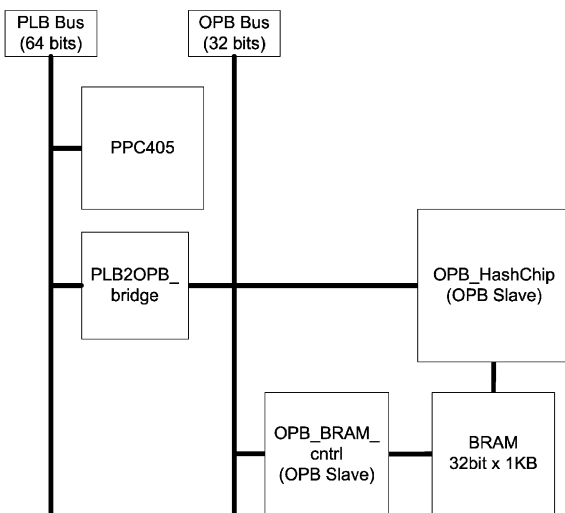


Fig. 8. Embedded system HashChip environment.

- ENABLE: Enables and disables the hash calculation.
- ALGO: Sets the active hash, either MD5, SHA1, or RIPEMD160.
- CNTR: Stores the value of the cycle counter that measures the time needed to compute the hash.
- DIGEST: A set of five registers that store the 128 or 160 bit hash values.

5. Performance analysis

The architectures described above are coded at the RTL level in Verilog HDL. A test bench module is developed to verify the functionality of the datapath. Performance analysis of the architectures in terms of the number of cycles taken for hashing messages of different sizes is done. The steps of the compression function finish much before the next set of 64 bytes are transferred into the padding block's buffer in the single cycle implementation. The difference in the number of clock cycles taken for MD5 and SHA1 or RIPEMD160 remains constant at 16 (the first compression function iteration of MD5 takes 64 cycles, while both SHA1 and RIPEMD160 take 80 cycles, after which each 512 bit block is processed in a constant amount of time which equals the time taken for 64 bytes to get transferred from the memory bank to the buffer and then to the internal register file). To prevent the hashing core from remaining idle, a single iteration is allowed to span multiple cycles. A brief summary of the results obtained for the three designed architectures are presented in Table 1. It can be observed from the table that the dual and triple cycle versions do not have the constant difference of 16 cycles in between the processing of MD5 and SHA1/RIPEMD160 irrespective of the message size. In fact, the difference in the number of cycles widens as the size of the message increases. The number of cycles taken to hash a message of a particular size has been determined from simulating the testbench, and there is no particular formula to predict the value for a message of a given size for the three architectures.

Table 1
Number of cycles to produce a valid message digest in the HashChip datapath

Algorithm	Datapath	Message size (number of 512-bit blocks)				
		16	32	64	128	256
MD5	Single	2524	4876	9580	18988	37804
	Dual	2683	5083	9883	19483	38683
	Triple	3788	7228	14108	27868	55388
SHA1	Single	2540	4892	9596	19004	37820
	Dual	3227	6139	11963	23611	46907
	Triple	4604	8812	17228	34060	67724
RIPEMD160	Single	2540	4892	9596	19004	37820
	Dual	3227	6139	11963	23611	46907
	Triple	4604	8812	17228	34060	67724

5.1. FPGA implementation of the HashChip

Each of the *HashChip* implementations have been mapped to a Xilinx V2P FPGA, part number XC2VP30FF896-7. The Xilinx ISE v7.1 Foundation tool set has been used to synthesize, translate, map, place and route, and download the design onto the FPGA. Characteristics of the implementation are shown in Table 2. The results indicate the synthesis frequency, number of V2P slices, where each V2P slice consists of configurable logic blocks (CLBs), and the flip-flops (FFs) resource usage. The FPGA synthesis and mapping tool does logic minimization to a very good extent for optimal implementation. Each Virtex LUT can be used as a 16-bit memory element also, and this significantly improves resource usage. Also, in the reported slice usage, each slice need not be completely utilized. Hence, it should not come as a surprise that the double and triple cycle implementations utilize lesser number of slices despite having additional pipelining registers.

The throughput results of the core are displayed in Fig. 9 and indicate that as the input file size reaches or exceeds 16 KB, the initial control overhead is overcome, and the throughput reaches a maximum bound. The effect of splitting each iteration into multiple cycles is that the

synthesis frequency of each design increases, but additional clock cycles are needed to perform each computation. In this case, the single cycle and triple cycle versions of the *HashChip* perform at about the same level with respect to overall throughput. However, the double cycle version outperforms the other two implementations. The clock frequency gain of the triple cycle version, only about 6 MHz over the double cycle, is not enough to balance the additional data latency through the system. The double cycle *HashChip* strikes a balance between clock cycle period and the total number of cycles necessary to complete each iteration.

5.2. Architectural comparison

Several other hash architectures in literature can be used to judge the performance of the *HashChip*. Only one of the published architectures unify the MD5, SHA1, and RIPEMD160 hashes, some unify two of them, and others present only performance relative to one hash algorithm. The throughput of these architectures, as well as those of the *HashChip*, are presented as numerical results in Table 3. The variety of host architectures chosen, such as ASICs, general purpose processors, and FPGAs make hard performance comparisons difficult, particularly in terms of area, but it is observed that the *HashChip* performs favorably in comparison to existing implementations in terms of throughput, keeping in mind the fact that it is on a reconfigurable platform and that it implements multiple hash primitives on a single architecture. The architectures from Dominikus [5] and Deepakumara [10], which are implemented on Xilinx Virtex FPGAs, utilize around 2000 and 4700 slices, respectively, for achieving the throughputs mentioned in Table 3.

Table 2
FPGA synthesis results for the *HashChip*

Implementation	Operating frequency (MHz)	Area (V2P slices)	Area (FFs)
Single cycle	58.137	2971	1386
Double cycle	97.426	2906	1515
Triple cycle	104.875	2837	1604

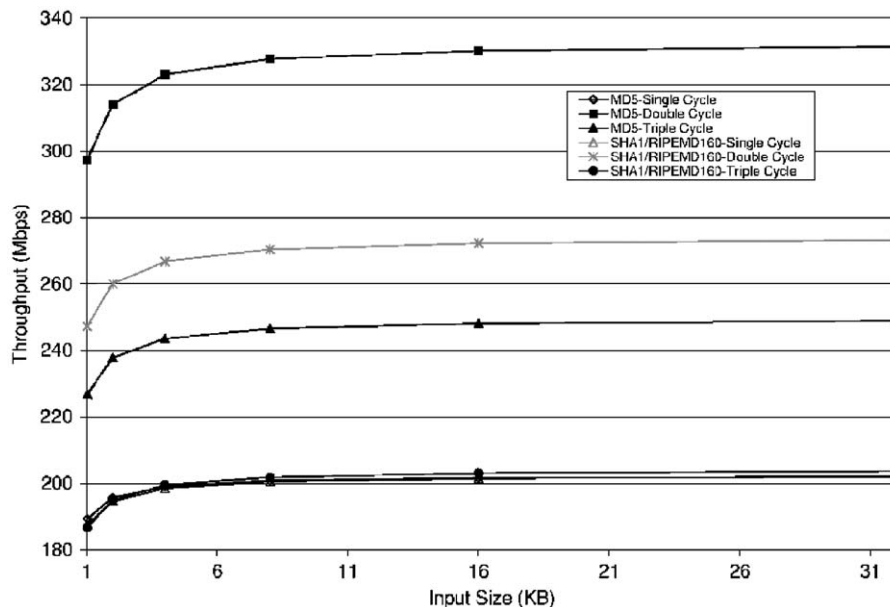


Fig. 9. Throughput comparison of datapath architectures.

Table 3
Throughput comparison of the *HashChip* and previously proposed architectures (Mbps)

Implementation	Device	MD5	SHA1	RIPEMD160
Single cycle	Virtex 2 Pro XC2VP30FF896-7	202	201	201
Double cycle	Virtex 2 Pro XC2VP30FF896-7	331	273	273
Triple cycle	Virtex 2 Pro XC2VP30FF896-7	248	203	203
Dobbertin [4]	Pentium	114	40	47
Domnikius [5]	Virtex 300E	107	119	65
Ng [6]	Altera EPF10K50SBC356-1	206	NA	84
Kang [7]	Altera EP20K1000EBC652-3	142	114	NA
Deepakumara [10]	Virtex V1000FG680-6	354	NA	NA
Wang [11]	Altera EP20K1000EBC652-1	178.6	143.3	NA
Touch [12]	ASIC	256	NA	NA

6. Conclusions

The application of cryptography algorithms in general, and MDC hash primitives in particular, for the purpose of secure communication has been investigated. The characteristics of the various platforms available for implementing cryptographic primitives have been compared and contrasted. The functioning of MD5, SHA1 and RIPEMD160 hash algorithms has been studied, and analysis of the similarities in their structure has been done. A unified architecture, *HashChip*, has been drawn up for implementing them on a hardware platform. The proposed architecture aggressively exploits the similarities in the structures to achieve maximal resource sharing. A generic approach to prototyping digital systems by taking advantage of the embedded PowerPC cores on the Xilinx Virtex 2P family of FPGAs has been proposed. The results of performance analysis of the *HashChip* architectures are presented above after using the proposed approach for implementation. The designed architecture compares favorably with many of the previously proposed architectures. Possible future extensions to the work include handling power dissipation concerns, mapping other cryptography algorithms into this architecture and usage of a standardized communication protocol to enable ease of deployment as a soft IP core.

References

- [1] W. Stallings, *Cryptography and Network Security—Principles and Practices*, third ed., New Delhi, India, Pearson Education, Singapore, 2003.
- [2] R. Rivest, RFC 1321 (The MD5 Message Digest Algorithm), MIT Laboratory for Computer Science and RSA Data Security Inc., 1992.
- [3] Secure Hash standard: National Institute of Standards and Technology FIPS PUB 180-2, August 2002.
- [4] H. Dobbertin, A. Bosselaers, B. Preneel, RIPEMD-160, in: D. Gollmann (Ed.), *A strengthened version of RIPEMD*, Fast Software Encryption, Lecture Notes in Computer Science, vol. 1039, Springer, Berlin, 1996, pp. 71–82.
- [5] S. Dominikus, A hardware implementation of MD4-family hash algorithms, in: *Proceedings of the Ninth International Conference on Electronics Circuits and Systems*, vol. 3, 2002, pp. 1143–1146.
- [6] C.-W. Ng, T.-S. Ng, K.-W. Yip, A unified architecture of MD5 and RIPEMD-160 Hash algorithms, *IEEE International Symposium on Circuits and Systems*, May 2004.
- [7] Y.K. Kang, D.W. Kim, T.W. Kwon, J.R. Choi, An efficient implementation of Hash function processor for IPSEC, in: *Proceedings of the 2002 IEEE Asia-Pacific Conference on ASIC*, 6–8 August 2002, pp. 93–96.
- [8] T.S. Ganesh, T.S.B. Sudarshan, ASIC implementation of a unified hardware architecture for non-key based cryptographic Hash primitives, in: *Proceedings of the 2005 IEEE International Conference on Information Technology Coding and Computing (ITCC)*, vol. I, 2005, pp. 580–585.
- [9] T.S. Ganesh, T.S.B. Sudarshan, N.K. Sreenivasan, K. Jayapal, Pre-silicon prototyping of a unified hardware architecture for cryptographic manipulation detection codes, in: *Proceedings of the Third IEEE International Conference on Field Programmable Technology*, 2004, pp. 323–326.
- [10] J. Deepakumara, H.M. Heys, R. Venkatesan, FPGA implementation of MD5 Hash algorithm, *2001 Canadian Conference on Electrical and Computer Engineering 2 (2001)* 919–924.
- [11] M.-Y. Wang, C.-P. Su, C.-T. Huang, C.-W. Wu, An HMAC processor with integrated SHA-1 and MD5 algorithms, in: *Proceeding of Asia and South Pacific Design Automation Conference*, 2004, pp. 456–458.
- [12] J.D. Touch, Performance analysis of MD5, *Proceedings of ACM SIGCOMM'95*, Cambridge, MA, 1995.



Ganesh T.S. is currently pursuing his Masters in Computer Engineering at Iowa State University under the guidance of Professor Arun K. Somani. He earned his Bachelor's degree in 2004 with a double major in Electrical Engineering and Computer Science from Birla Institute of Technology and Science, Pilani. His research interests include Fault Tolerant VLSI Architectures, Reconfigurable Computing and Design Automation using system-level description languages.

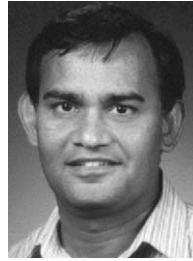


Michael Frederick is a Ph.D. candidate at Iowa State University. Michael earned his Bachelor's degree with a minor in Spanish in 2001 and his Master's degree in 2002, each in Computer Engineering from Iowa State University. He is a part of the Dependable Computing and Networking Laboratory under the direction of Professor Arun K. Somani. His research interests include network fault tolerance, image proces-

sing, dynamically reconfigurable computing, and the design and implementation of embedded systems.



Sudarshan T.S.B. is an Assistant Professor in Birla Institute of Technology and Science, Pilani, India. He did his Bachelor's in Electrical and Electronics Engineering from Bangalore University, Master's in Systems from Birla Institute of Technology, Mesra, Ranchi and Ph.D. from BITS, Pilani. His research interests are cryptohardware, high performance computer architecture and web performance.



Arun K. Somani is currently Jerry R. Junkins Endowed Chair Professor of Electrical and Computer Engineering at Iowa State University. He earned his MSEE and Ph.D. degrees in electrical engineering from the McGill University, Montreal, Canada, in 1983 and 1985, respectively. Professor Somani's research interests are in the area of fault tolerant computing, computer interconnection networks, WDM-based optical networking, and parallel computer system architecture. He has served on several program committees of various conferences in his research areas. He has been elected a Fellow of IEEE for his contributions to theory and applications of computer networks.