# General-Purpose FPGA Platform for Efficient Encryption and Hashing

Jakub Szefer, Yu-Yuan Chen and Ruby B. Lee
Department of Electrical Engineering
Princeton University
Princeton, NJ, USA
Email: {szefer, yctwo, rblee}@princeton.edu

***Abstract*—Many applications require protection of secret or sensitive information, from sensor nodes and embedded applications to large distributed systems. The confidentiality of data can be protected by encryption using symmetric-key ciphers, and the integrity of the data can be ensured by using a cryptographic hash function to calculate a "digital fingerprint." In this paper, we propose reconfigurable FPGA hardware components that enable rapid deployment of cryptographic and other algorithms. The novelty of our hardware components is in their general-purpose design which enables easy mappings of algorithms to allow customizations of data protection for different usage scenarios. Since we utilize only a small part of an FPGA chip, our design can be readily integrated with other processing needs of a mobile device, a sensor node or a System-on-Chip. Important block ciphers like the Advanced Encryption Standard (AES) as well as advanced cryptographic hash algorithms like Whirlpool map well onto our general-purpose components. Our solution facilitates easy hardware implementation of customizable encryption and hashing solutions, with area and speed performance comparable to custom FPGA implementations targeted at a specific cipher or hash algorithm. We achieve the best efficiency in Mbps/slice for Whirlpool. Furthermore, the components that we have proposed can be used for many other applications - not just for implementing block ciphers and cryptographic hash functions.**

## I. Introduction

Symmetric key cryptography can be used to protect the confidentiality of secret or sensitive information, and hash algorithms can be used to protect the integrity of the data. Many hardware ASIC (Application-Specific Integrated Circuit) implementations of cryptographic and hash algorithms exist. While very good at implementing these algorithms efficiently, the ASIC designs are fixed once manufactured and cannot be changed if a new algorithym is needed. FPGA (Field-Programmable Gate Array) hardware designs are more flexible since they may be upgraded in the field, but they are typically slower than the ASIC designs. Also, if the FPGA design is targeted at a specific algorithm, it requires a completely new design to be written from scratch in HDL (Hardware Description Language) to implement a different algorithm.

In this paper, we propose an FPGA platform that contains two general-purpose and reconfigurable components: *Ptab* implements versatile constant-time parallel table lookup operations and *Perm* implements versatile permutations of the subwords of its input, where a subword can be defined as any number of bits. We show that different cryptographic and hash algorithms can be decomposed into parts that correspond to the two components. Furthermore, efficient implementation of various algorithms based on these components is possible, e.g., AES [1], Whirlpool [2] and several of the newly proposed candidates for the Advanced Hash Algorithms (AHS) from the first two rounds of the NIST competition [3].

Section II describes the proposed components of the FPGA platform. Section III describes the design flow. Section IV describes mapping of different algorithms into the two components. Section V presents an evaluation of the design. Section VI has comparison to related work and section VII concludes our paper.

## II. Proposed Architecture

Our FPGA platform for fast encryption and hashing makes use of two components described below. A parallel table component, *Ptab*, is used for constant-time accesses to a set of configurable data tables. A permutation component, *Perm*, is used for performing various shifts and permutations on the subwords of its input. The two components can be used individually or in combination. The platform can also be expanded with other components in the future.

Our work is a "platform" because it defines a number of general-purpose components which can easily be parameterized. These components can be used as building blocks for various algorithms. The platform can be thought of as a set of pre-designed components which can be resynthesized with different options and put together to efficiently implement different designs. The actual inputs, outputs and interconnection logic depend on the algorithm and the parameters with which the components were synthesized.

### A. Ptab - A Parallel Tables Component

Various implementations of cryptographic algorithms are optimized by looking up pre-computed tables. These tables are read many times and usually not updated during the execution of the algorithm. Often these table lookups can be performed in parallel since each of them performs an independent operation.

The *Ptab* is a parameterized hardware FPGA implementation of table lookups which is an improvement over the more restricted `ptrd` processor instruction presented by [4]. We use block RAM memories (BRAMs) available in the Xilinx

TABLE I
PTAB RE-CONFIGURATION PARAMETERS: PTAB(B,T,E,W,I,O,IW,OW)

| Parameter | Description |
|---|---|
| B | Number of banks of tables |
| T | Number of tables per bank |
| E | Number of entries per table |
| W | Number of bits per table entry |
| I | Number of inputs |
| O | Number of outputs |
| IW | Bit width of inputs |
| OW | Bit width of outputs |

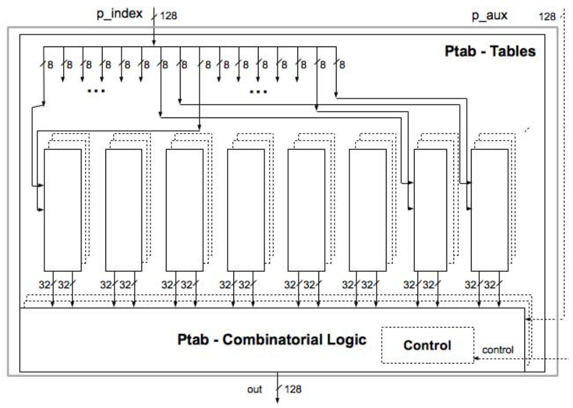

Fig. 1. Ptab component, configured with 128-bit inputs and outputs.

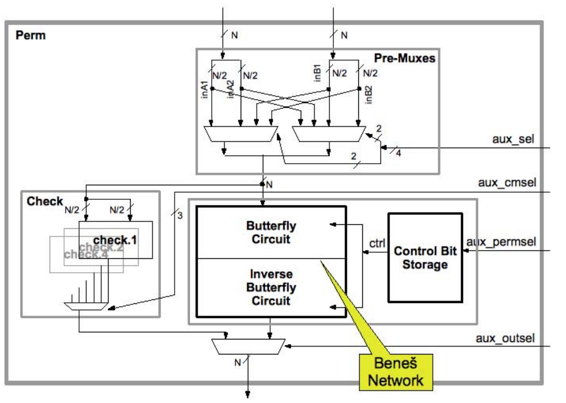

Fig. 2. Permutation, Perm, component.

TABLE II
PERM RE-CONFIGURATION PARAMETERS: PERM(N, SW, C, P)

| Parameter | Description |
|---|---|
| N | Number of items to permute, N=0 indicates that Beneš datapath will not be included in the unit |
| Sw | Subword size |
| C | Include check datapath in the unit? |
| P | Include input pre-muxes? |

FPGAs to store the tables and save FPGA slices used by the implementation. BRAMs allow the lookup of different tables to be performed in parallel, with only one cycle latency. *Ptab* can be configured with different parameters, summarized in Table I. We allow for multiple banks, or sets, of tables. By selecting a different bank, a different set of tables is activated and the table lookup facilitates a different algorithm or function (e.g. encryption versus decryption). Fig. 1 shows a sample block diagram of *Ptab* configured with eight 256-entry tables per bank, with 32-bit entries.

In addition to the tables themselves, the *Ptab* component contains a combinatorial logic portion which can be used to implement logic for combining the table outputs; it is often the case that the outputs of the tables need to be combined. For example, for AES, the combinatorial logic block needed is a simple XOR tree which combines all the table outputs and also performs the key addition (by XORing in the key input from the `p_aux` input, see Fig. 1). While the combinatorial block in Fig. 1 can be arbitrarily specified for the given application, we do define a few common implementations, expressed as a tree of multiplexors for selecting different ways of combining the table outputs.

## B. Perm – A Byte Permutation Component

The *Perm* component is used to perform permutations, which can arbitrarily reorder the bits, or disjoint subwords of bits, in its input. The *Perm* component can augment the functionality of the *Ptab* component: it can serve as a flexible means of reordering *Ptab*'s input. Fig. 2 shows the top level view of the *Perm* component, with some options shown in Table II.

The main datapath of the *Perm* component is based on a Beneš network. A Beneš network [5] is composed of a butterfly network followed by an inverse butterfly network. The network can be configured to permute different subword sizes. In general, a Beneš network with N inputs needs N$\log$ N control bits. These control bits are obtained from the control bit storage (Fig. 2).

The main Beneš datapath can optionally be augmented with a set of pre-muxes, shown in Fig. 2. The pre-muxes can improve *Perm*'s functionality: by having the option to select portions, e.g. halves, of the inputs that will be permuted, eliminating the need to combine two inputs in a separate step before using the *Perm* component.

In addition, *Perm* can be configured to include a dedicated check datapath, inspired by the `check` instruction [6]. Furthermore, synthesizing the *Perm* with only the `check` circuits, instead of the Beneš datapath, constrains the functionality of *Perm*, but this reduces circuit size considerably when fully general permutations are not needed.

## C. Other Components

The design of the proposed FPGA platform is not restricted to only using these two components. As a general purpose platform, it can be extended with other components – after suitable research and design work is performed to identify the more useful ones. What we have found is that these two general-purpose components, *Ptab* and *Perm*, are especially useful for implementing block ciphers and cryptographic hash functions.

## III. Design Flow

To make use of the components of the platform, the design that is to be mapped into our FPGA platform has to be systematically broken down into constituent parts and those parts can then be mapped to the *Perm* or *Ptab* components. While the mapping is done manually at present, in the future, it could be done with new EDA (Electronic Design Automation) tools. Because the components of the FPGA platform are predefined and are reconfigurable with a small number of parameters (see Tables I and II), the EDA tools could be extended to automatically map certain code constructs to our components.

## IV. Algorithm Mapping

### A. AES Encryption and Decryption

AES is the Advanced Encryption Standard announced in 2001 [1]. All AES versions have the same underlying structure, except for the number of rounds in the algorithm and the key size. AES consists of a number of rounds, each round performing four functions: `SubBytes`, `ShiftRows`, `MixColumns` and `AddRoundKey`. When implementing the algorithm, the designers of AES suggested a method to implement each round of AES using table lookups [7]; the same approach can be translated to an FPGA hardware design. Essentially, the different steps of the round transformation can be combined into a single set of table lookups. In particular, the `MixColumns` and `SubBytes` steps of AES can be combined as a look up of a single table of 256 32-bit entries. We employ this table transformation in our AES implementation by using the *Ptab* component. The table outputs are combined by an XOR tree, and the `AddRoundKey` step is also implemented in the combinatorial logic block of the *Ptab*.

Consequently, each round of AES can be broken up into two parts, which map to our two components *Perm* and *Ptab*. First, *Perm* takes a 128-bit AES round input and permutes the bytes according to the indices required to address the appropriate tables in the *Ptab* structure. Second, the output of the *Perm* unit is passed to the *Ptab* to perform the table lookups.

### B. Whirlpool Hash

Whirlpool [2] is a cryptographic hash function which was selected for inclusion in the NESSIE portfolio of cryptographic primitives. In addition it has been adopted as part of a joint ISO/IEC 10118-3 international standard. It is a hash algorithm with a structure similar to AES, except that it operates on an 8x8 matrix of bytes. The algorithm operates by iterating a compression function that has fixed-size input and fixed-size output. Its compression function is an AES-like round which operates on the 512-bit state. This hash function can be implemented using table lookup operations which implement the non-linear substitution and linear diffusion steps of each round, as suggested by the authors of the algorithm [2], and also in [8]. When implementing the hash function with table lookups, each round consists of two parts. First the columns of the Whirlpool state matrix are cyclically rotated at the beginning of each round. Second, the bytes of the rotated state matrix serve as indices into the lookup tables. When decomposing the algorithm, the column rotations map to the *Perm* component, and the table lookups map to the *Ptab* component.

## V. Evaluation

The area and performance of implementations of the algorithms described in this section are labelled "Our" in Tables III and IV.

### A. AES Cost and Performance

For the AES-128 encryption algorithm, *Ptab* has been configured with 1 bank of eight 256-entry tables with 32 bits per entry and 1 bank of eight 256-entry tables with 8 bits per entry (for the last round of AES encryption). A similar setup is done for decryption. The storage in *Ptab* needed for these four sets of tables is then 20KB = 8KB + 2KB + 8KB + 2KB. The *Perm* has been configured with 128-bit inputs and 8-bit subwords. The AES algorithm requires 2 sets of control bits for *Perm* (64-bits for each set) to perform the various permutations: one for encryption and one for decryption. When executing the algorithm, each round (permutation of the input and table lookup) requires one pass (one cycle) through the unit, taking a total of 11 cycles for one 128-bit input block. This slower (but smaller) unpipelined version of AES is what we report in Table II. By pipelining the *Ptab* and *Perm*, we can essentially double the MHz rate, the Mbps and the Mbps/slice.

### B. Whirlpool Cost and Performance

For Whirlpool [2], the *Ptab* has been configured with 1 bank of eight 256-entry tables with 64-bit entries. The combinatorial logic part consists of 2 sets of XORs used to combine each of the eight 8-byte table outputs to form a 16-byte final output. The *Perm* has been configured with 128-bit input and 8-bit subwords. In our first implementation, the *Perm* (with the Beneš datapath) has been duplicated 24 times, and there are 3 levels of 8 *Perms* which work together to rotate the columns of the input. All *Perm* units are used in one cycle to perform the column rotation. Due to interconnection between the units, only 3 different permutations: `check.1`, `check.2` and `check.4` are needed; this requires 3 sets of control bits (to configure the Beneš networks which implement the three check permutations). In our second implementation of Whirlpool, we use the dedicated check operations without the full Beneš datapath in a *Perm* unit. This achieves a much more efficient implementation, especially in the Mbps/slice efficiency metric.

## VI. Comparison to Related Works

Tables III and IV show our work and compare it to previous works, sorted by the FPGA chip used. The throughput is calculated as follows:

$$throughput = \frac{frequency}{latency} \times inputsize \tag{1}$$

The *freqency* for each implementation is in MHz, the *latency*, in number of cycles, depends on the implementation,

TABLE III
SUMMARY OF AES IMPLEMENTATIONS.

| Implementation | FPGA Board | Slices | BRAMs | MHz | Mbps | Mbps/slice |
|---|---|---|---|---|---|---|
| Our | Virtex-5 (xc5vls110t-1ff1136) | 374 | 25 | 220 | 2600 | 6.95 |
| Helion [9] | Virtex-5 | 349 | 0 | 350 | 4100 | 11.67 |
| Bulens, et al. (Encryption) [10] | Virtex-5 | 400 | 0 | 350 | 4100 | 10.20 |
| Bulens, et al. (Encryption) [10] | Virtex-4 | 700 | 8 | 250 | 2900 | 4.10 |
| Lemsitzer, et al. [11] | Virtex-4 (FX100) | 3800 | 114 | 140 | 17900 | 4.70 |
| Liberatori, et al. [12] | Spartan-3 | 1643 | 0 | 52 | 123 | 0.07 |

TABLE IV
SUMMARY OF WHIRLPOOL IMPLEMENTATIONS.

| Implementation | FPGA Board | Slices | BRAMs | cycles/byte | MHz | Mbps | Mbps/slice |
|---|---|---|---|---|---|---|---|
| Our (configured to use dedicated check) | Virtex-5 (xc5vls110t-1ff1136) | 2063 | 64 | 0.34 | 145 | 3411 | **1.65** |
| Our (configured to use Beneš to implement check) | Virtex-5 (xc5vls110t-1ff1136) | 6173 | 88 | 0.34 | 41 | 964 | 0.16 |
| McLoone, et al. [13] (Iterative) | Virtex-4 (xc4vlx100) | 4956 | 68 | 0.16 | 94 | 4790 | 0.97 |
| McLoone, et al. [13] (Unrolled) | Virtex-4 (xc4vlx100) | 13210 | 0 | 0.08 | 48 | 4896 | 0.37 |
| Pramstaller, et al. [14] | Virtex-II Pro (xc2vp40-7fg676) | 1456 | 0 | 2.74 | 131 | 382 | 0.26 |
| Alho, et al. [15] | Virtex-II Pro (xc2vp40) | 376 | 0 | 21.00 | 214 | 81.5 | 0.21 |
| Kitsos, et al. [16] (LB) | Virtex (v1000efg1156-8) | 5585 | 0 | 0.16 | 87.5 | 4480 | 0.80 |
| Kitsos, et al. [16] (BB) | Virtex (v1000efg1156-8) | 5713 | 0 | 0.16 | 72 | 3686 | 0.65 |

and the *input size* depends on the algorithm, e.g. 128 bits for AES.

The comparison between the different implementations is difficult because numbers for the same FPGA board are not available for all the implementations. Also, certain designs target specific FPGA boards and are usually customized for a specific algorithm. Finally, certain implementations include I/O and padding, while our does not. Our platform may not outperform a specific algorithm implementation optimized for a specific FPGA device. However, it achieves excellent efficiency in Mbps/slice, especially for Whirlpool. Its two units: *Ptab* and *Perm* can easily be re-synthesized with new parameters and new table contents for new algorithms – providing both good performance and flexibility.

## VII. CONCLUSION

This paper presented an FPGA platform for fast encryption and hashing which is based on a re-configurable parallel table component (*Ptab*) and a re-configurable permutation component (*Perm*). We propose that the platform can be used to efficiently implement symmetric key block ciphers and hash algorithms. We showed that different algorithms can be mapped to use these two components and that the mapping resulted in good performance, despite the general-purpose nature of the components. Our implementations were also efficient with respect to the Mbps/slice metric.

## VIII. ACKNOWLEDGMENTS

This work was supported in part by National Science Foundation grants: NSF EEC-0540832 and NSF CCF-0917134.

## REFERENCES

[1] NIST, "Advanced encryption standard (aes), fips 197," November 2001. [Online]. Available: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[2] P. Barreto and V. Rijmen, "Whirlpool hash function." [Online]. Available: http://www.larc.usp.br/ pbarreto/WhirlpoolPage.html

[3] NIST, "Cryptographic hash algoirthm competition." [Online]. Available: http://csrc.nist.gov/groups/ST/hash/sha-3/index.html

[4] A. M. Fiskiran and R. B. Lee, "On-chip lookup tables for fast symmetric-key encryption," *Application-Specific Systems, Architectures and Processors, IEEE International Conference on*, vol. 0, pp. 356–363, 2005.

[5] V. E. Beneš, *Mathematical theory of connecting networks and telephone traffic*. Academic Press, 1965.

[6] R. B. Lee, "Subword permutation instructions for two-dimensional multimedia processing in microsimd architectures," *Application-Specific Systems, Architectures and Processors, IEEE International Conference on*, vol. 0, p. 3, 2000.

[7] J. Daemen and V. Rijmen, *The design of Rijndael: AES–the Advanced Encryption Standard*. Springer, 2002.

[8] Y. Hilewitz, Y. L. Yin, and R. B. Lee, "Accelerating the whirlpool hash function using parallel table lookup and fast cyclical permutation," in *Lecture Notes in Computer Science*, vol. 5086/2008, 2008, pp. 173–188.

[9] H. T. Limited, "Helion Fast AES Encryptor," http:www.heliontech.com/downloads/aes_xilinx_helioncore.pdf.

[10] P. Bulens, F.-X. Standaert, J.-J. Quisquater, P. Pellegrin, and G. Rouvroy, "Implementation of the aes-128 on virtex-5 fpgas," in *AFRICACRYPT*, 2008, pp. 16–26.

[11] S. Lemsitzer, J. Wolkerstorfer, N. Felber, and M. Braendli, "Multi-gigabit gcm-aes architecture optimized for fpgas," in *CHES '07: Proceedings of the 9th international workshop on Cryptographic Hardware and Embedded Systems*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 227–238.

[12] M. Liberatori, F. Otero, J. Bonadero, and J. Castineira, "Aes-128 cipher. high speed, low cost fpga implementation," *Programmable Logic, 2007. SPL '07. 2007 3rd Southern Conference on*, pp. 195–198, Feb. 2007.

[13] M. McLoone, C. McIvor, and A. Savage, "High-speed hardware architectures of the whirlpool hash function," in *Field-Programmable Technology, 2005. Proceedings. 2005 IEEE International Conference on*, Dec. 2005, pp. 147–153.

[14] N. Pramstaller, C. Rechberger, and V. Rijmen, "A compact fpga implementation of the hash function whirlpool," in *FPGA '06: Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays*. New York, NY, USA: ACM, 2006, pp. 159–166.

[15] T. Alho, P. Hamalainen, M. Hannikainen, and T. Hamalainen, "Compact hardware design of whirlpool hashing core," in *Design, Automation and Test in Europe Conference and Exhibition, 2007. DATE '07*, April 2007, pp. 1–6.

[16] P. Kitsos and O. Koufopavlou, "Whirlpool hash function: architecture and vlsi implementation," *Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004 International Symposium on*, vol. 2, pp. II–893–6 Vol.2, May 2004.