An Embedded Memory-Centric Reconfigurable Hardware Accelerator for Security Applications

Christopher Babecki, *Student Member, IEEE*,
Wenchao Qian, *Student Member, IEEE*,
Somnath Paul, *Member, IEEE*,
Robert Karam, *Student Member, IEEE*, and
Swarup Bhunia, *Senior Member, IEEE*

Abstract—Security has emerged as a critical need in today's computer applications. Unfortunately, most security algorithms are computationally expensive and often do not map efficiently to general purpose processors. Fixed-function accelerators offer significant improvement in energy-efficiency, but they do not allow more than one application to reuse hardware resources. Mapping applications to generic reconfigurable fabrics can achieve the desired flexibility, but at the cost of area and energy efficiency. This paper presents a novel reconfigurable framework, referred to as hardware accelerator for security kernel (HASK), for accelerating a wide array of security applications. This framework incorporates a coarse-grained datapath, supports for lookup functions, and flexible interconnect optimizations, which enable on-demand pipelining and parallel computations in multiple ultralight-weight processing elements. These features are highly effective for energy-efficient operation in a diverse set of security applications. Through simulations, we have compared the performance of HASK to software and field programmable gate array (FPGA) platforms. Simulation results for a set of six common security applications show comparable latency between HASK and FPGA with 2.5X improvement in energy-delay product and 4X improvement in iso-area throughput. HASK also shows 5X improvement in iso-area throughput and 45X improvement in energy-delay product compared to optimized software implementations.

Index Terms—Security applications, domain-specific hardware accelerator, energy-efficiency, reconfigurable computing

1 Introduction

Security is becoming an important design metric, specifically in the domain of embedded systems [1]. Due to the ever-growing importance of data security in these systems, the inclusion of hardware cryptographic modules is rapidly becoming a requirement. In particular, hardware modules for diverse security tasks, ranging from encryption to hashing, are more effective than software realizations in terms of meeting energy-efficiency and/or real-time performance demands [2], [3], [4]. These security modules are implemented either inside embedded processors or outside them as co-processors or hardware accelerators [2]. Typically, these modules are realized in one of two ways: (a) as a specialized custom hardware module, or (b) as a reconfigurable accelerator using a field programmable gate array (FPGA) or similar platform. While the first option provides optimal performance and energy-efficiency, the second one offers the flexibility to map a variety of security tasks satisfying diverse compute and communication requirements.

The Advanced Encryption Standard (AES) is widely used in embedded applications to achieve data security. Unfortunately, most encryption algorithms like AES are slow on general purpose processors (GPPs), with a single block encryption typically taking hundreds of cycles on an embedded processor without hardware support [5]. This strains real-time applications where a software-only approach may not meet throughput requirements. The energy efficiency of security algorithms on GPPs is also poor, which can be unattractive in energy-constrained systems.

One approach to alleviate this energy and performance bottleneck is to develop dedicated hardware for specific application kernels like AES, such as the AES-NI instruction set extension for the x86 platform [6]. This yields excellent performance and energy results, but only functions for a specific algorithm and requires substantial development and verification effort to implement for each kernel. Therefore, this approach does not scale well to diverse security kernels [3]. Similarly, FPGAs can be used to improve flexibility, and while this may enhance energy efficiency over software-only implementations, the highly reconfigurable interconnect architecture can impose significant penalties to power, area, and latency [17]. Instead, domain-specific reconfigurable architectures have been introduced which demonstrate improved performance and power results over general purpose reconfigurable platforms, [2], [7], [8]. Notable architectures include asynchronous array of simple processors (AsAP) [7], [15] and Morphosys [2], while the Totem system [8] aims to automate the process of domain specific accelerator design. However, none of these platforms has been designed specifically for the domain of security acceleration.

To address this important need, in this paper, we present a hardware accelerator for security kernels (HASK), a novel and highly energy-efficient reconfigurable framework for the acceleration of cryptographic kernels, based on an analysis of security applications such as *AES*, *Blowfish*, *IDEA*, *SHA-1*, *MD5*, and *CAST-128*. HASK is an array of parallel, coarse-grained nano-processors with hardware support for common operations in security tasks. It aims to balance spatial and temporal computing by time-multiplexing hardware resources while having a large number of interconnected processing elements to distribute a task. The overall system serves as a loosely coupled accelerator on a shared bus accessible by both a host processor and by peripherals. Fig. 1 illustrates a conceptual diagram of a modern system on a chip (SoC) which incorporates an acceleration engine for security algorithms.

HASK's primary distinctions from existing reconfigurable architectures, such as AsAP and Morphosys, are as follows: (a) a novel, fully distributed memory architecture with multiple instruction, multiple data (MIMD) support; (b) a processing and interconnect architecture tailored to security applications; (c) fully routerless communication, which improves data transfer energy and latency; and (d) hardware support for lookup table operations with varying input and output width, and fused logical operations to implement complex functions as atomics. In particular, the paper makes the following key contributions:

1) It proposes HASK, a scalable and energy-efficient reconfigurable architecture suitable for the domain of security applications. HASK supports spatio-temporal computing with an array of light-weight processing elements. The datapath and interconnect structure for HASK are optimized to commonly occurring operations and communication patterns for target applications.
2) It explains the micro-architecture level optimizations for HASK and the corresponding design trade-offs. It also describes the design space exploration that trades off between spatial and temporal computing to optimize energy efficiency.
3) It evaluates the performance and energy efficiency of HASK for six common applications and then compares the results against a commercial FPGA device, a GPP, and alternative reconfigurable accelerator MorphoSys [2] and AsAP [15].

- *C. Babecki, W. Qian, R. Karam, and S. Bhunia are with the Department of Electrical Engineering and Computer Science, Case Western Reserve University, Cleveland, OH 44106.*
  E-mail: {christopher.babecki, wenchao.qian, robert.karam, swarup.bhunia}@case.edu.
- *S. Paul is with Intel Labs, Intel Corporation, Hillsboro, Hillsboro, OR 97124.*
  E-mail: somnath.paul@intel.com.

Manuscript received 14 Nov. 2014; revised 13 Nov. 2015; accepted 9 Dec. 2015. Date of publication 24 Dec. 2015; date of current version 14 Sept. 2016.
Recommended for acceptance by W.W. Ro.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2015.2512858

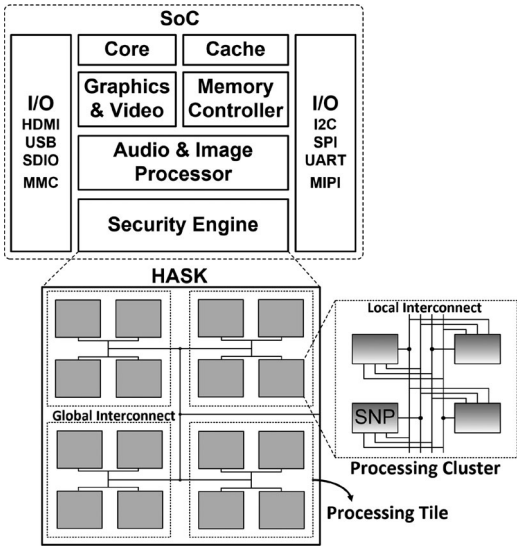

Fig. 1. Block diagram of a SoC (or a processor) using HASK as an external accelerator for security applications.

The rest of the paper is organized as follows: Section 2 explains the system architecture for HASK, with details on both datapath and interconnect structure. Section 3 presents the performance results for candidate security applications and compares with FPGA and other reconfigurable platforms. Section 4 concludes the paper and provides future directions.

2 System Architecture

A single HASK processing element, termed a security nano-processor (SNP) is shown in Fig. 2. Each SNP operates independent of the others as a MIMD machine and has its own local data and instruction memory. The interconnect fabric (both local and global as shown in Fig. 1) does not contain any memory elements. SNPs are organized into two levels of hierarchy: *Clusters*, which contain four SNPs, and *Tiles*, which contain four Clusters. Each Tile has a central controller responsible for writing to the instruction memory of each SNP and transferring data between the main system memory or a peripheral and local SNP memory. In this section, we analyze the application requirements and describe in detail the μ-architecture of each SNP and the interconnects between them.

2.1 Analysis of Security Applications

We have considered six security applications to map into the HASK framework, including *AES* [9], *Blowfish* [10], *CAST-128* [11], *IDEA* [12], *MD5* [13], and *SHA-1* [14]. Inputs to each kernel are assumed to be stored in the local memory, so data reads need to be performed before execution. These applications make use of addition, bitwise logical operations (AND, OR, XOR), circular and logical shifts, and non-linear functions (Substitution Boxes, or S-Boxes). The S-Box functions these applications use take 8 bits of input and have either 8 or 32 bits of output. The Bitwise logical operations are 8, 16, or 32 bits wide. Logic operations are bit-sliceable, for example, a 32-bit AND operation can be sliced into four 8-bit wide AND operations, which can be performed in parallel. Complex logical operations are also commonly used in these applications. For example, cryptographic hashing algorithms such as MD5 and SHA-1 contain complex logical operations such as $(A \wedge B) \vee (\bar{A} \wedge C)$. These types of operations typically use the same inputs but perform different combinations of logic functions over time. Increasing computing efficiency of these operations helps to improve performance and energy efficiency.

The deterministic nature of the security kernels is amenable to pipelining and parallelism. Due to their iterative nature, each application has a certain number of computing rounds and each computing round requires almost the same set of operations using different keys as inputs. Therefore, the latency for each round is almost the same, and pipelining can be easily applied to improve the throughput. Latency can also be improved by applying instruction-level and data-level parallelism within each computing round.

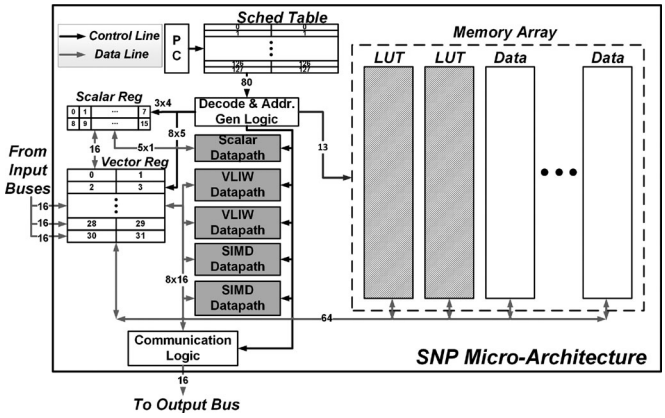

Fig. 2. Block diagram of security nano-processor.

2.2 SNP Hardware Architecture

The SNP is modeled after a standard RISC-style processing element, containing an 8 kB SRAM memory array, a lightweight custom datapath, a 32-byte register file, and a program counter. The register file is designed to have a large number of read ports (8) and write ports (4) to support the wide execution engine of the SNP. Unlike a processor which typically would fetch instructions from memory, each SNP has a dedicated schedule table that holds the 80-bit wide instructions which are preloaded when the SNP is configured. The proposed SNP design holds 128 instruction words. Support is provided for several standard operations including add, shift, simple logical operations, and load/store.[1]

Although each SNP behaves like a processor, it implements many features that attempt to mimic common hardware components necessary to not just accelerate security applications, but also maximize energy efficiency. These domain-specific optimizations are as follows:

- a two-way execution engine;
- hardware support for variable width vectorized lookup table (LUT) operations;
- a fused logical unit for arbitrary three-input functions;
- a byte-addressable register file;
- support for SIMD-style datapath operations.

Instruction-level and data-level parallelism can be realized through statically mapped VLIW and vector architectures. The SNP exploits this by allowing two operations to be statically scheduled to execute in a given cycle. Each SNP operation can be encoded in 40 bits; thus, the full 80 bit instruction can hold any two independent operations, even simultaneous memory accesses. Security applications also commonly exploit highly nonlinear functions to "mask" the data being processed (e.g. S-boxes in ciphering algorithms like AES, Blowfish, and CAST-128). Each SNP supports 8-bit input lookup operations with variable output sizes, including 8, 16, and 32 bits to map these functions efficiently. The first 4 kB of memory in each SNP is reserved for LUTs and uses an asymmetric memory design to achieve a 40 percent reduction in read energy

1. Due to space limitation, a detailed description of the instruction set architecture has been moved to the supplementary material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TC.2015.2512858

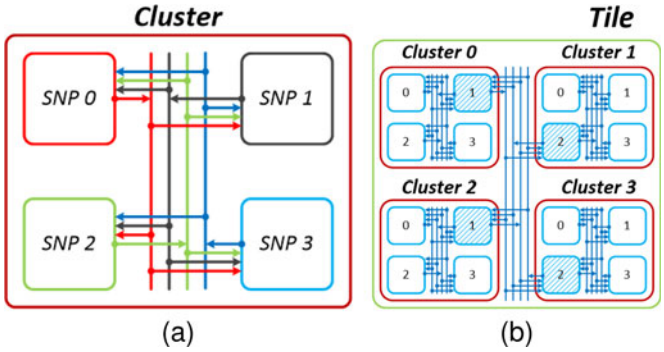

Fig. 3. Hierarchical bus-based interconnect structure of: (a) a single Cluster of 4 SNPs, (b) a single Tile of four Clusters. The tiles are connected in a mesh topology.

[4], as well as fine-grained wordline segmentation allowing efficient access to the variable width LUTs. The remaining memory is used as a byte addressable scratchpad-memory that stores inputs and resultant data.

For complex logical operations, such as $(A \wedge B) \vee (\bar{A} \wedge C)$, it could take up to 4 cycles by using atomic logic operations. Though HASK supports bit-sliced logic operations, mapping this complex operation requires substantially more energy than dedicated logic. In addition, a LUT-based approach would require a substantial amount of memory space to hold the responses. Instead, these complex logical operations are mapped to a novel reconfigurable logic datapath inside each SNP capable of implementing arbitrary logical functions of up to three inputs. This is realized and encoded using a Reed-Muller expansion, which results in substantially fewer required transistors than a canonical representation for an arbitrary function of three inputs [16]. For example, the fused datapath can be configured to perform $(A \wedge B) \vee (\bar{A} \oplus C)$ in a single instruction as $1 \oplus A \oplus C \oplus AB \oplus AC$.

Most security kernels are dominated by byte level operations (e.g. S-boxes in AES, Blowfish, and CAST-128 and Galois Field multiplications in AES). To mitigate this requirement, all data operations in HASK support variable input/output size down to a single byte for increased energy efficiency. The register file is also designed accordingly to be byte addressable, enabling more compact information storage.

Finally, many security tasks are amenable to additional parallelism beyond what a VLIW-2 architecture allows. For example, consider the *MixColumns* step of the AES algorithm, which requires 12 XOR operations comprising four independent expressions. The SNPs on-demand SIMD instruction exploits this data parallelism by

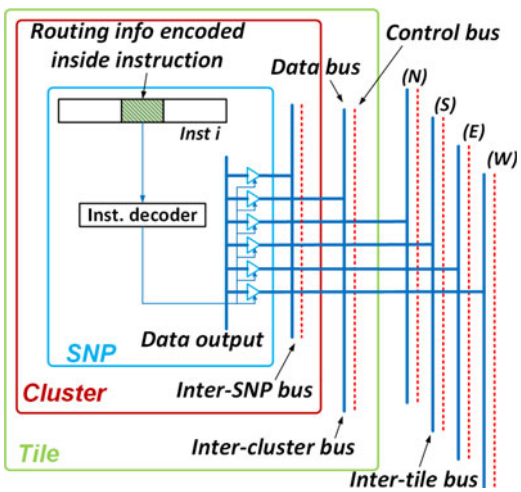

Fig. 4. Detailed view of the SNP bus interface.

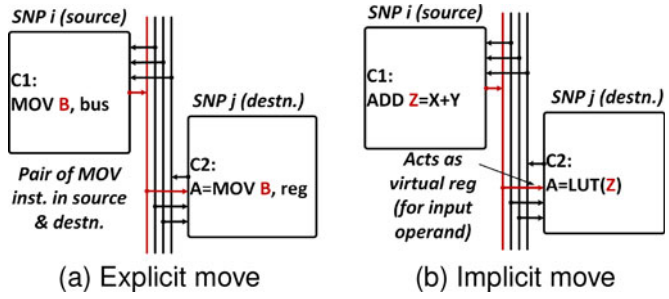

Fig. 5. Two modes of inter-SNP communications: (a) Explicit data transfer through a MOVE instruction and (b) implicit transfer using intra-cluster bus as virtual register.

simultaneously reading four sets of input operands from the register file. The inputs are then run through four separate datapaths and written back through separate write ports. This improves the overall execution time of the AES algorithm by about 10 percent. Addition operations can be accelerated in this manner as well.

2.3 Interconnect Structure

HASK employs a sparse hierarchical interconnect that exploits data locality. SNPs within the same Cluster have a fully connected shared bus structure (Fig. 3a). Each 16-bit connection allows for the transmission of a single 8 or 16-bit value per cycle; at 1.25 GHz (see Section 2.4), this leads to a rate of 20 Gbps. Similarly, a 16-bit shared bus is used for inter-cluster communication (Fig. 3b); however, unlike the fully-connected intra-cluster bus, the inter-cluster bus can only be reached from one SNP per cluster, termed Gateway SNP, or gSNP, through which all communications must be routed. Ideally, applications can be mapped such that only the gSNP needs to communicate with other clusters, maximizing the inter-cluster bus throughput. The lower bound of the bandwidth per SNP is 1/4 of the intra-cluster bus, or 5 Gbps. At higher levels, gSNPs can communicate with each of the 16 gSNPs in adjacent Tiles through a 16-bit wide 2D bi-directional mesh interconnect structure. It enables the architecture to easily add large number of tiles (and hence, SNPs). A gSNP can broadcast to multiple inter-Tile buses in the same cycle, allowing the architecture to scale to an arbitrary size while maintaining limited connectivity between any two nodes. Data transmission between tiles requires two cycles without any stall. Thus, the cycle time is independent of communication latency between distant nodes. In the worst case, this communication is limited to 625 Mbps, when all 16 gSNPs need to communicate as often as possible.

These communication buses form a time-multiplexed programmable interconnect. Because the communication requirements for the security applications are both constant and known *a priori*, they can be scheduled at compile time. Routing information is stored in the instructions as an immediate value and decoded at runtime to control communication buses. If a buffer is enabled on a given cycle, output data from the SNP's operation is written to the appropriate bus. Subsequently, other SNPs can read the data into their local register files. A detailed view of the bus output structure is provided in Fig. 4. As the communications are statically scheduled, routers are not required in the fabric which eliminates their associated power, latency, and area overhead.

Communication at any level of the hierarchy can be handled either implicitly as part of an instruction, or explicitly as a separate MOVE instruction. Fig. 5a shows an example instruction sequence that performs this type of data transfer. If data Z needs to be transferred from the register of SNP_i to the register of SNP_j, SNP_i needs a MOVE to send data onto the bus, and in the next cycle, SNP_j needs another MOVE to get the data from the bus and store it locally. Additionally, some instructions contain a single bit field that, when enabled, writes the result to the local bus as well as local memory. Every instruction is capable of an implicit read through

TABLE 1
SNP Energy Breakdown for Different Operations

Operation	Sched. Table & Decoder (pJ)	Reg. file (pJ)	Execution (pJ)	Total Energy (pJ)	Implicit MOVE[a] (pJ)	Total Energy with Implicit MOVE (pJ)
Load/Store (64b)			2.234	3.913		4.082
8x8 LUT Op			0.283	1.962		2.131
8x16 LUT Op			0.563	2.242		2.411
8x32 LUT Op			1.123	2.802		2.971
Add/Simple Logical Op	1.358	0.32	0.017	1.696	0.17	1.866
Shift			0.035	1.714		1.883
Fused Datapath Op			0.018	1.697		1.866
Intra-cluster Move			0.169	1.848		2.018
Inter-cluster Move			0.511	2.189		2.359
Inter-tile Move			0.678	2.356		2.526
Leakage Power (mW/SNP)				3.024		

[a] *Energy for an implicit move is the same as the execution energy for an intra-cluster move.*

the use of virtual register ports, which map directly into the register files of other SNPs within a cluster (Fig. 4). This method of communication is only available on the intra-cluster bus (Fig. 5b). The output Z of the operation in SNP_i can be directly sent onto the bus, and in the case that Z is used immediately in SNP_j as an input, it can be directly read from the bus because the bus is treated as a virtual register. Implicit moves are useful in many security applications, including AES, where four SNPs within 1 Cluster can operate on data cyclically and in parallel, taking only 83 cycles for one encrypt operation. This structure allows for very high data availability at the lowest level where it is needed, while reducing interconnect complexity, routing delays, and communication energy between the upper levels.

2.4 Modeling Approach

We developed a register-transfer level (RTL) model of the HASK framework to obtain performance characterization of the architecture. Key components (e.g. datapath, register file, etc.) were modeled in RTL then synthesized to a 32 nm technology library provided by Synopsys using DesignCompiler. The area, delay, and power consumption (considering a 12.5 percent activity factor) for each of these components were then used to model a full SNP. Based on the area estimates for a single SNP, the estimated communication delay and energy requirements were computed using an RC wire-loading model assuming a standard Fan-Out of 4 load on each bus line. The RTL model for a single SNP has been extensively validated to confirm that each operation functions correctly in simulations using Synopsys VCS. Table 2 provides a summary of the key model parameters.

Memory elements (schedule table, LUT and data memory) were modeled using the CACTI toolset to determine area, energy, and delay estimates for SRAM arrays of the appropriate sizes. For the schedule table and 2 kB of the 8 kB main memory, the SRAM design was assumed to be read-skewed and a 40 percent reduction in energy use for read operations was accordingly considered [4]. CACTI models a memory bank assuming that it is part of a larger cache, and therefore includes latency and energy components not relevant to this SRAM model. These components, including the H-tree delay and sub-array output driver energy, are therefore removed. Wordline segmentation using AND-gates is added as an overhead to the CACTI model, and LUT accesses for 8, 16, and 32 bits are similarly reduced to the appropriate proportion of the data access energy reported by CACTI.

TABLE 2
SNP Key Parameters

SNP Area	70,000 μm^2
Critical Path Delay	790 ns
Max Clock Frequency	1.25 GHz
Registers	32 x 8 b
Schedule Table	128 x 80 b
LUT Memory	4 kB
Data Memory	4 kB

The delays for each component were used to compute the critical path through a single SNP. The critical path lies on the datapath of a memory read operation, and the total path delay is calculated to be approximately 800 ps, yielding a maximum clock speed of 1.25 GHz. The estimated area for a single SNP is approximately 70,000 μm^2, about 60 percent of which is the 8 kB SRAM array, 18 percent the datapath elements, and the remaining 22 percent the schedule table and register file. The energy requirement for each major operation type is presented in Table 1 along with the estimated static leakage power consumption of a single SNP. The majority of the leakage energy comes from the SRAM arrays (schedule table, LUT memory, and data memory). Unfortunately, CACTI cannot effectively model advanced manufacturing processes such as High-K Metal Gates or strained channel devices [20]. Hence, the actual leakage power of a HASK system should be substantially lower than the reported result.

2.5 Design Space Exploration

The spatio-temporal computing model of HASK provides the opportunity to explore the right balance of spatial and temporal computing during application mapping to achieve optimal energy efficiency. We select SHA-1 as an example kernel to study this. We vary the number of SNPs from 8 to 1,024, to map this kernel, and observe varying performance and energy profiles. To map SHA-1, we first created its control and data flow graph and then translated the operations into a set of instructions for the SNP architecture. By increasing the number of SNPs, the total 80 rounds required for SHA-1 can be distributed into multiple pipelining stages. In general, higher SNP usage resulted in fewer cycles between hashed blocks; however, increasing pipelining stages resulted in diminishing returns, as each stage begins to be dominated by the transfer time rather than actual computation. If area is not a constraint, one can solve the presented optimization problem to find the point of minimal energy delay product (EDP) as shown in Fig. 6. When data transfer is not the dominant contributor to total delay (in the case of 8-128 SNPs), performance improves as the number of SNPs increases. We observe that using 128 SNPs is very close to optimal in terms of EDP, which combines the impact from both delay and energy, while using substantially less area than alternative implementations with 256 SNPs or more.

3 Performance Analysis

Our application suite consists of *AES*, *Blowfish*, *IDEA*, *SHA-1*, *MD5*, and *CAST-128*. It includes a combination of both symmetric key

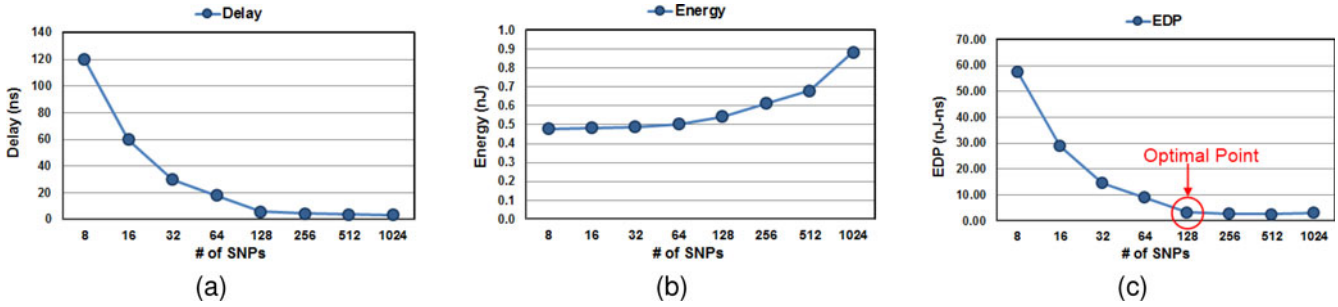

Fig. 6. Results for design space exploration in case of SHA-1 by trading off spatial versus temporal computing: (a) Delay, (b) Energy, and (c) EDP.

ciphers and cryptographically secure hashes that are representative of the security domain. Each benchmark was implemented in the HASK framework, on an Altera Stratix V FPGA (E series, 28-nm CMOS process, 0.9V supply), and in software on a quad-core Intel Q8200 processor (45 nm, 0.85-1.35 V). For each case, the power, latency, EDP, and area were collected for a single instance of a given kernel working on a single block of data. Additionally, maximum throughput values for a single instance of the kernel are presented. For HASK and FPGA, we also present the size of the configuration bitstream required to program the device. The results are shown in Table 3. The HASK, FPGA, and CPU platforms differ greatly, and to facilitate a fair comparison, all devices are scaled to the same process node (32 nm) and voltage (0.95 V), and the same 12.5 percent switching activity is used for power analysis.

Each application was hand-mapped to the HASK framework using the approach outlined in Section 2.5. We considered a soft area constraint of 4 SNPs with an exception for IDEA, where nine SNPs were used to allow for pre-loading the round keys, greatly reducing the energy cost from load operations. Latency values were computed assuming a 800 ps clock period (1.25 GHz), based on the critical path delay, and multiplying by the total number of cycles required. To obtain the dynamic energy values, the number of each operation type for a given application was counted and then multiplied by corresponding energy consumption, as presented in Table 1. Static power was computed by taking the total leakage power per SNP (Table 1), then multiplying by the number of required SNPs and the latency of the application. Energy values presented are the sum of the static and dynamic energy.

FPGA mapping was accomplished by describing an application in Verilog and then compiling it to the Stratix V using Quartus II. To match the HASK memory, access to an 8 kB memory array is assumed to load the initial values. Latency estimates were obtained using TimeQuest, and energy estimates were obtained by multiplying the number of effective cycles, cycle time, and power estimates reported in PowerPlay. To estimate the area, we multiply the ALM tile area, which includes estimated routing area [19], by the number of utilized ALMs reported by Quartus II.

The software implementations use the crypto++ C++ library, compiled and optimized for the target architecture, and timestamps are measured using the C++ chrono library. The programs run 3E+5 input vectors 200 times each and return the average energy and delay values per application to mitigate measurement noise. The target Q8200 processor does not support the AES-NI instruction set extension, but can exploit SSE (Streaming SIMD Extensions) instructions to enhance performance. This is chosen to illustrate performance on a modern superscalar processor without any form of hardware acceleration. The CPU power consumption is assumed to be half the rated TDP (95 W) [7] divided by 4, since only one of the four cores was active. We estimate that a single core occupies approximately one quarter of the die shown [18], or roughly 25 mm^2 at 45 nm, and use this approximation when calculating the throughput per unit area. Results for the Q8200 are presented in the "GPP" columns of Table 3.

The area results presented for both HASK and FPGA include the area of a single scaled GPP core. This is more representative of a real system where a GPP core would be present, and the security tasks would be offloaded to an accelerator. The table shows average ratio (GPP over HASK and FPGA over HASK) for each parameter, which is computed by taking average over the individual ratios for all benchmarks. This is done to accommodate for wide variations in latency/energy across the benchmarks. Compared to GPP, FPGA and HASK improve latency about 40 and 10 percent, respectively. As a result, both platforms improve iso-area throughput substantially. Similarly, both accelerators see an order of magnitude improvement in energy efficiency, specifically, average EDP improvements of 34x (FPGA) and 45x (HASK). These improvements for both FPGA and HASK relative to GPP are due to the following common factors:

- HASK and FPGA perform their computations in light-weight processing elements.
- The GPP's inclusion of hardware structures not beneficial to the target domain lead to poor resource utilization.
- The highly parallel nature of HASK and FPGA lend themselves better to pipelining of processing steps.

TABLE 3
Comparison of Latency, Throughput, Energy and EDP among HASK, FPGA and GPP

Kernel	Latency (ns)			Throughput (bps/μm^2)			Energy/bit (pJ/bit)			EDP (nJ-ns)		
	HASK	FPGA	GPP	HASK	FPGA	GPP	HASK	FPGA	GPP	HASK	FPGA	GPP
AES	66.4	33.0	128.20	152.48	306.82	78.98	14.22	38.81	470.60	1.21E+2	1.64E+2	7.72E+3
Blowfish	93.6	93.5	82.06	54.09	54.14	61.70	9.96	49.01	385.58	5.97E+1	2.93E+2	2.02E+3
CAST-128	167.2	128.0	88.97	90.83	39.55	56.90	35.30	52.07	453.35	3.78E+2	4.27E+2	2.58E+3
IDEA	70.4	126.0	155.62	647.19	40.18	32.53	37.09	14.43	1386.87	1.67E+2	1.16E+2	1.38E+4
MD5	566.4	396.8	323.91	35.75	12.76	15.63	17.41	51.47	751.05	5.05E+3	1.05E+4	1.25E+5
SHA-1	714.4	584.0	495.15	14.17	4.33	5.11	21.58	130.37	1755.04	7.89E+3	3.90E+4	4.45E+5
Avg. Ratio (GPP / HASK)	1.14X	1.39X	—	5.1X	1.5X	—	41X	25X	—	45X	34X	—
Avg. Ratio (FPGA / HASK)	0.93X	—	—	4.3X	—	—	3.1X	—	—	2.5X	—	—

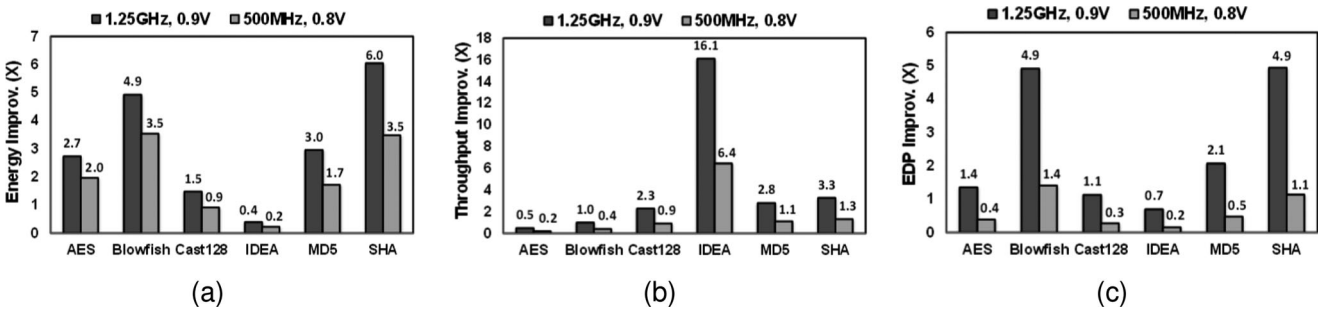

Fig. 7. The effect of scaling operating frequency (with associated scaling of voltage) on (a) energy, (b) throughput, and (c) EDP improvement (higher is better) in HASK compared to FPGA.

- Complex and fused functions are performed in dedicated hardware structures and/or LUTs.

For the majority of the benchmark applications, HASK latency is slightly worse than that of FPGA. However, since HASK implementations generally use less die area than FPGA, the iso-area throughput is 4.3X better than that of FPGA, and HASK uses 3.1x less energy than FPGA on average. Moreover, a 2.5X improvement in EDP is observed. Note that in practice, we anticipate the energy improvement of HASK over FPGA to be even higher. As stated in Section 2.4, the HASK leakage energy is overestimated, since CACTI cannot model leakage reduction techniques used for nanoscale processes. Conversely, the FPGA leakage is underestimated, since Quartus does not report contributions from programmable interconnects and embedded memory blocks. The primary reasons behind the improvement in energy efficiency for HASK over FPGA are as follows: (1) HASK supports LUT operations of different bitwidth and has dedicated hardware for fused logic operations, which reduces the total number of operations and hence, energy; (2) the spatio-temporal mapping greatly reduces the interconnect complexity and energy; and (3) the highly customizable memory structure of FPGAs results in energy inefficient memory accesses [19]. Additionally, the HASK configuration bitstream is an average 2.3 percent smaller than equivalent FPGA implementation. It is worth noting that HASK's energy performance on the IDEA cipher is poor compared to FPGA. The major reason behind this is that IDEA uses multiplications which must be implemented as LUT operations in the SNP. These memory accesses are substantially less energy efficient than using the embedded multiplier blocks of the Stratix V, so HASK's energy efficiency suffers.

We also consider a comparison to three other prominent alternative hardware acceleration platforms: (a) AsAP, (b) Morphosys, and (c) graphics processing units (GPUs). GPUs are suitable for floating point intensive kernels, but cannot provide a fine-grained spatio-temporal mapping like HASK, AsAP, and Morphosys, resulting in sub-optimal performance. In contrast to these frameworks, HASK has four key distinguishing factors: (1) HASK requires no shared memory on-die, (2) it provides hardware support for variable input and output LUT operations, (3) SNPs contain local fixed function optimization for security, and (4) HASK implements a hierarchical interconnect appropriate for security applications. We present Singh's [2] and Liu's [7] findings scaled to a 32 nm process node operating at 0.95 V and compare them with HASK results in Table 4. We derive iso-area throughput and energy per bit as points of comparison between these disparate computing fabrics. Since no energy results were provided for Morphosys, only throughput is compared.

TABLE 4
Comparison of HASK with CGRA/GPU with Respect to Throughput and Energy

Benchmark	Fabric	Scaled Throughput (Gbps/mm^2)	Energy/Bit (pJ/bit)
AES	HASK	7.44	13.20
	AsAP	1.29	742.9
	GPU	0.39	2147.2
IDEA	HASK	14.03	—
	Morphosys	10.34	—

A more conservative operating frequency of 500 MHz has also been considered for HASK. In this case, the operating voltage can be reduced to 0.8 V. Energy and latency values are scaled accordingly for all the components of the HASK model and improvement versus FPGA for all benchmark applications is compared in Fig. 7. Throughput improvement scales down linearly to 1.7X versus FPGA; however, total energy increases due to a greater leakage energy contribution. As a result, the EDP improvement compared to an FPGA at its maximum frequency goes down to 0.64X. However, energy required per bit for HASK still remains 2X better than FPGA.

4 Conclusions

We have presented HASK, a novel reconfigurable framework for accelerating security applications. The proposed architecture offers comparable latency and die area to FPGA with an average of 3X improvement in energy efficiency. This is achieved using high-density SRAM for lookup operations, a sparse interconnect, application-level pipelining, and a custom datapath tailored to the computing needs of the security application domain. We presented simulation results which show the improvement compared to implementations in FPGAs and GPPs. The interconnect fabric can accommodate a large number of SNPs and hence is scalable, enabling the mapping of larger kernels or even parallel instances of a kernel. The performance of HASK is also compared with alternative CGRAs such as AsAP and Morphosys, as well as GPUs. The memory-dominated architecure of HASK is very amenable to technology scaling. Hence, emerging high-density nanoscale memory technologies can significantly improve its performance and area. Future work will include the development of an application mapping tool and adding support for dynamic instruction scheduling.

Acknowledgments

This work was supported in part by Semiconductor Research Corporation (SRC) under Grant 2015-EP-2650.

References

[1] P. Kocher, R Lee, G. McGraw, A. Raghunathan, and S. Ravi, "Security as a new dimension in embedded system design," in *Proc. Des. Autom. Conf.*, 2004, pp. 753–760.

[2] H. Singh, M.-H. Lee, G. Lu, F. J. Kurdahi, N. Bagherzadeh, and E. M. Chaves Filho, "MorphoSys: An integrated reconfigurable system for data-parallel and computation-intensive applications," *IEEE Trans. Comput.*, vol. 49, no. 5, pp. 465–481, May 2000.

[3] K. Eguro, "RaPiD-AES: Developing an encryption-specific FPGA architecture," M.S. thesis, Univ. Washington, Seattle, WA, USA, 2002.

[4] S. Paul, S. Chatterjee, S. Mukhopadhyay, and S. Bhunia, "Energy-efficient reconfigurable computing using a circuit-architecture-software co-design approach," *IEEE J. Emerging Sel. Topics Circuits Syst.*, vol. 1, no. 3, pp. 369–380, Sep. 2011.
[5] D. A. Osvik, J. W. Bos, D. Stefan, and D. Canright, *Fast Software AES Encryption*. Berlin, Germany: Springer, 2010, pp. 75–93.
[6] S. Gueron, "Intel Advanced Encryption Standard (AES) new instructions set," Intel Corp., Santa Clara, CA, USA, Tech. Rep. 323641-001, 2012.
[7] B. Liu and B. Baas, "Parallel AES encryption engines for many-core processor arrays," *IEEE Trans. Comput.*, vol. 62, no. 3, pp. 536–547, Mar. 2013.
[8] K. Compton and S. Hauck, "Totem: Custom reconfigurable array generation," in *Proc. 9th Annu. IEEE Symp. Field-Programmable Custom Comput. Mach.*, 2001, pp. 111–119.
[9] J. Daemen and V. Rijmen, "AES proposal: Rijndael," in *Proc. 1st Adv. Encryption Standard Candidate Conf.*, Mar. 1999.
[10] B. Schneier, "Description of a new variable-length key, 64-bit block cipher (Blowfish)," in *Fast Software Encryption*. Berlin, Germany: Springer, 1994.
[11] C. Adams, "The CAST-128 encryption algorithm," Entrust Technol., Addison, TX, USA, RFC 2144, May 1997.
[12] X. Lai, *On the Design and Security of Block Ciphers* (ETH Series in Information Processing). Konstanz, Germany: Hartung-Gorre Verlag, 1992.
[13] R. L. Rivest, *The MD5 Message-Digest Algorithm*, MIT Lab. Comput. Sci. RSA Data Security, Cambridge, MA, USA, 1992.
[14] D. Eastlake and P. Jones, "US secure hash algorithm 1 (SHA1)," Motorola and Cisco Systems, San Jose, CA, USA, RFC 3174, Sep. 2001.
[15] D. N. Truong, W. H. Cheng, T. Mohsenin, Y. Zhiyi, A. T. Jacobson, G. Landge, M. J. Meeuwsen, C. Watnik, A. T. Tran, X. Zhibin, E. W. Work, J. W. Webb, P. V. Mejia, and B. M. Baas, "A 167-processor computational platform in 65 nm CMOS," *IEEE J. Solid-State Circuits*, vol. 44, no. 4, pp. 1130–1144, Apr. 2009.
[16] X. Wu, X. Chen, and S. L. Hurst, "Mapping of Reed-Muller coefficients and the minimization of exclusive OR-switching functions," *IEE Comput. Digital Tech.*, vol. 129, no. 1, pp. 15–20, 1982.
[17] A. Rahman, S. Das, A. P. Chandrakasan, and R. Reif, "Wiring requirement and three-dimensional integration technology for field programmable gate arrays," *IEEE Trans. Very Large Scale Integration (VLSI) Syst.*, vol. 11, no. 1, pp. 44–54, Feb. 2003.
[18] V. George, S. Jahagirdar, C. Tong, K. Smits, S. Damaraju, S. Siers, V. Naydenov, T. Khondker, S. Sarkar, and P. Singh, "Penryn: 45-nm next generation Intel core^{TM}2 processor," in *Proc. Asian Solid-States Circuits Conf.*, 2007, pp. 14–17.
[19] H. Wong, V. Betz, and J. Rose, "Comparing FPGA vs. custom CMOS and the impact on processor microarchitecture," in *Proc. Int. Symp. Field Program. Gate Arrays*, 2011, pp. 5–14.
[20] S. Li, K. Chen, J. H. Ahn, J. B. Brockman, and N. P. Jouppi,"CACTI-P: Architecture-level modeling for SRAM-based structures with advanced leakage reduction techniques," in *Proc. Int. Conf. Comput.-Aided Design*, 2011, pp. 684–701.