DRAB-LOCUS: An Area-Efficient AES Architecture for Hardware Accelerator Co-Location on FPGAs

Jacob T. Grycel
Email: jtgrycel@wpi.edu
Worcester Polytechnic Institute
Department of Computer Science
Worcester, MA

Robert J. Walls
Email: rjwalls@wpi.edu
Worcester Polytechnic Institute
Department of Computer Science
Worcester, MA

***Abstract*—Advanced Encryption Standard (AES) implementations on Field Programmable Gate Arrays (FPGA) commonly focus on maximizing throughput at the cost of utilizing high volumes of FPGA slice logic. High resource usage limits systems' abilities to implement other functions (such as video processing or machine learning) that may want to share the same FPGA resources. In this paper, we address the *shared resource challenge* by proposing and evaluating a *low-area*, but high-throughput, AES architecture. In contrast to existing work, our DSP/RAM-Based Low-CLB Usage (DRAB-LOCUS) architecture leverages block RAM tiles and Digital Signal Processing (DSP) slices to implement the AES Sub Bytes, Mix Columns, and Add Round Key sub-round transformations, reducing resource usage by a factor of 3 over traditional approaches. To achieve area-efficiency, we built an inner-pipelined architecture using the internal registers of block RAM tiles and DSP slices. Our DRAB-LOCUS architecture features a 12-stage pipeline capable of producing 7.055 Gbps of interleaved encrypted or decrypted data, and only uses 909 Look Up tables, 593 Flip Flops, 16 block RAMs, and 18 DSP slices in the target device.**

I. Introduction

As the Advanced Encryption Standard (AES) lies at the core of many important security operations—ranging from securing network connections to full disk encryption—improvements to performance and efficiency can benefit a large and diverse set of systems. One means to achieve this performance increase is through the use of *hardware acceleration*. For instance, a number of studies have leveraged field programmable gate arrays (FPGAs) for accelerating AES [1]. While these FPGA-based AES architectures achieve high throughput, they typically do so at the cost of high resource usage on the FPGA, i.e., monopolizing large quantities of components such as flip flops and look up tables. While higher throughput is valuable, we argue in this paper that *area-efficiency* is often an equally important design goal for AES architectures.

Area-efficiency, intuitively, is a measure of how effectively an architecture capitalizes on the FPGA's resources. The key challenge is not only making effective use of under-utilized components (such as digital signal processing slices), but understanding the device layout and incorporating that knowledge into the design. However, the potential payoff is great as area-efficient designs offer substantial benefits over those focusing purely on throughput. Most notably, area-efficiency opens up possibilities for using hardware-accelerated security in new domains. A variety of embedded systems simply cannot implement strong, efficient encryption due to processor limitations or other constraints (e.g., power)—FPGA-based approaches are a promising way to overcome these challenges. Indeed, manufacturers now release cheap system-on-a-chip (SoC) platforms that feature co-located FPGAs and CPUs. However, without area-efficient AES designs, the system developer may have to choose between security and other operations that benefit from hardware acceleration, such as deep learning [2], [3] or video processing [4]. An area-efficient AES architecture would allow other types of hardware acceleration to run concurrently on the same FPGA.

In this paper, we propose a novel AES architecture that considers resource-efficiency as a first-order design principle, balancing resource usage and throughput. Key to our design is the use of block RAM and digital signal processing slices—resources that are largely under-utilized by prior works [1]—to efficiently implement the AES sub-round transformations without the need for large numbers of logic slices. The DSP/RAM-Based Low-CLB Usage (DRAB-LOCUS) architecture offers several advantages over existing approaches, including: *(i)* high throughput (7.055 Gbps) on cheaper hardware; *(ii)* more efficient use of FPGA resources which reduces logic slice utilization in systems that require high volumes of flip flops and look up tables; *(iii)* more functionality, allowing for concurrent encryption and decryption on multiple blocks.

II. Area-Efficient Design

The DRAB-LOCUS AES architecture utilizes a mixture of slice logic, block RAM (BRAM), and digital signal processing slices (DSP slices) to perform encryption and decryption on concurrent blocks of data. This design consists of three main components: Datapath, Controller, and Key Schedule. A more detailed description of DRAB-LOCUS can be found in the extended version of this paper [5].

A. Datapath Architecture

The DRAB-LOCUS datapath implements an iterative, inner-pipelined realization of the AES round transformations. Where possible, transformations occur in block RAM (BRAM) and digital signal processing (DSP) slices to effectively use all available FPGA resources within the physical bounds of the circuit. For example, the datapath contains 4 and 8 BRAM lookup tables for the sub bytes and mix columns transformations, as opposed to utilizing lookup tables (LUTs) and flip flops, which may result in high fanout nets that struggle to meet timing constraints. Furthermore, the add round key and mix columns transformations use cascaded DSP slice inputs and outputs to perform large `XOR`s. The only transformation not mapped to BRAM or DSP slices is shift rows, which we implement as a large multiplexer in slice logic that selects between row mappings for encryption and decryption.

We add register stages to the datapath pipeline by enabling additional output registers on BRAM and a combination of input/output registers on DSP slices. While two sets of flip flops are used to acquire register stages for shift rows and mix columns, all other stages come at no extra cost in FPGA resources; in total, the datapath contains 12 register stages. DRAB-LOCUS connects the AES transformations in the classic order used for encryption: sub bytes, shift rows, mix columns, add round key. The architecture uses this datapath for both encryption and decryption by employing the AES *equivalent inverse cipher* [6], which uses the same order of operations for decryption but with a modified key schedule.

The datapath supports continuous operation of the transformations, allowing any block to be repeatedly processed for either encryption or decryption without any stall cycles required to insert new data or perform final round transformations. DRAB-LOCUS achieves this continuous operation by instantiating two extra copies of the add round key transformation to handle initial and final round key additions without disturbing blocks in the datapath.

To support decryption, the key schedule runs all derived round keys through the mix columns transformation in decryption mode to obtain inverse keys. To avoid high fanout from the controller to select between the key schedule and shift rows inputs to mix columns, the two inputs are `OR`'ed together before mix columns. Then during key initialization the controller holds shift rows in reset using dedicated reset paths, and during cipher operation the key schedule resets the registers used in initialization. While this arbitration still requires controller intervention, it is faster than using logic slice routing. A similar technique is applied to sub bytes to select between the key schedule and add round key instances.

B. Control Module and Key Schedule

Control Module: The DRAB-LOCUS controller tracks the progress of each block in the pipeline by counting the number of completed rounds and specifying the correct mode (encryption/decryption) for each datapath transformation. Since the datapath contains 12 register stages, and performs 10 rounds of transformations for 128-bit security, each full process takes 115 clock cycles (this is 5 less than 120 due to omission of mix columns in the final round). We achieve this functionality using a two-part controller: An array of shift registers, and a finite state machine (FSM).

The array consists of two types of shift registers. The first type tracks which process (encryption or decryption) should be used in each transformation, and is implemented using a cascade of flip flops. Registers of the second type count how long each data block has been processed, and indicates when processing is complete. These registers are 115 bits long to represent each clock cycle in a full encryption/decryption process. Since only the final bit is used to determine when processing is complete, they are implemented using LUTs in SRL32E configuration as opposed to a chain of 115 flip flops and LUTs. This style is also more efficient than using traditional counters, which would require 7 flip flop/LUT pairs to achieve the same function. By using SRL32E configuration, each register takes only 4 LUTs. To track all possible data blocks in the pipeline we use 12 of these shift registers.

The FSM-part of the controller handles flushing and interpretation of the shift register array. With 12 blocks of data in the pipeline at a given time, each going through encryption or decryption concurrently, the cipher state is too complicated to describe discretely with an FSM. The shift register array informs the FSM of whether each block is finished, and whether there is space available in the pipeline.

Key Schedule: The key schedule utilizes an FSM to control the computation of round keys using internal registers and connections to the sub bytes and mix columns instances. The FSM iteratively computes round keys and inverse round keys for the equivalent inverse cipher so that the key for any round is available on request. This is necessary since any block in the pipeline can be in any round. We use BRAM to store all of the computed keys, and use 12 counters to keep track of which round key is needed for each block in the datapath.

Due to having three instances of the add round key transformation, it is possible for the datapath to need the initial, final, and an arbitrary round key at the same time. We solve this challenge by using one BRAM port to produce an arbitrary round key, another port to always produce the final round key, and by adding a 128-bit register to hold the initial round key. This register, and two other 128-bit registers that hold intermediate keys during initialization, make the key schedule have the highest logic slice usage in the entire architecture.

III. Implementation

We synthesized and implemented the DRAB-LOCUS architecture using a Zynq 7000 SoC featuring co-located ARM processors and an Artix-7 grade FPGA (xc7z030sbg485-3). As a result of our focus on using BRAM and DSP slices for area-efficiency, the entire implementation fits in one half of a clock region. This uses less power because the FPGA routes clocks to only one section of the device.

In addition to fitting within a single clock region, all of the logic elements are contained within two columns of BRAM tiles. This compact layout allows DRAB-LOCUS to run at the maximum frequency supported by BRAM. This is because there is less physical distance between components and, therefore, less delay on the critical path.

In total, the DRAB-LOCUS architecture uses only 909 LUTs, 593 flip flops, 16 BRAM, and 18 DSP in the target FPGA. The AES sub-round transformations use only 266 LUTs and 256 flip flops, less than half of architecture's total slice utilization. It runs on a 528 MHz clock, producing 7.055 Gbps of interleaved encrypted and decrypted data when the pipeline is full, and has a latency of 217 ns (115 clock cycles) for a single block to finish encryption or decryption.

IV. Evaluation

Our evaluation focuses on answering the following key questions. *(i)* How does DRAB-LOCUS compare to other FPGA-based AES architectures? Specifically, what is the impact of different architectural decisions on throughput, resource usage, and functionality? *(ii)* How efficiently does DRAB-LOCUS use its allocated resources? How do we evaluate efficiency for AES designs in general? *(iii)* How does the use of non-logic-slice resources affect power consumption?

We evaluate DRAB-LOCUS by comparing to another AES architecture designed for area efficiency, which we call Drimer-AES, designed by Drimer et al. [7]. There are many AES designs, such as the design by Wang and Ha that achieves a high throughput of 78.22 Gbps at the expense of high resource usage [8], or the design by de la Piedra et al. which uses minimal slice logic but has a low throughput of 124 Mbps [9]. However, there are few that utilize all FPGA primitives and aim for area efficiency. The resource usage and performance of the DRAB-LOCUS and Drimer-AES architectures is shown in Table I. The numbers presented for Drimer-AES come from the original publication, as we did not build an implementation due to ambiguities in the controller and key schedule. In the extended version of this study we discuss an implementation of DRAB-LOCUS on a Virtex 5 device. Refer to the extended paper for details on this additional implementation [5].

A. Architectural Effects on Performance

DRAB-LOCUS and Drimer-AES use similar iterative inner-pipelined architectures, but with Drimer-AES using 72 more LUTs, 368 more flip flops, and 4 less BRAM. This difference in BRAM comes mainly from the design decisions that enable DRAB-LOCUS to perform both encryption and decryption concurrently on 12 blocks of data. Drimer-AES implements the round using T-Boxes, which combine the sub bytes and mix columns sub-round transformations into a single look up and XOR operation [10]. However, this technique utilizes an entire 36 kilobit BRAM to store lookup values for encryption. With a single copy of the round datapath, this approach prevents Drimer-AES from supporting concurrent encryption and decryption.

By keeping the sub bytes and mix columns sub-round transformations separate, DRAB-LOCUS is able to perform both encryption and decryption at the expense of using 4 more BRAM. This technique increases the latency of DRAB-LOCUS by two extra delay cycles for the sub bytes sub-round transformation, but also increases the capacity of the datapath to operate on two more blocks of data than if the design used T-boxes. Even with extra latency, DRAB-LOCUS maintains higher throughput than Drimer-AES, and supports both encryption and decryption.

The DRAB-LOCUS key schedule was also designed to be able to provide any round key at any time, in order to support encryption and decryption for 12 concurrent blocks in the datapath. In order to achieve this, the key schedule utilizes 616 LUTs, 303 flip flops, and 4 BRAM, which increases the total LUT usage beyond that of Drimer-AES. But, at the expense of these extra resources, DRAB-LOCUS achieves more functionality. This shows that including extra algorithm optimizations such as T-boxes can reduce the potential for design functionality, and that minimally increasing resource usage can allow for extra functionality, such as supporting encryption and decryption concurrently on multiple blocks.

B. Implementation Efficiency

Studies that build FPGA designs often evaluate the efficiency of their implementation by computing the amount of throughput produced per logic slice. For example, Drimer-AES achieves 22.6 Mbps/slice and DRAB-LOCUS achieves 22.75 Mbps/slice. However, this metric does not give any information on how much the BRAM and DSP resources contribute to the design throughput. This approach also neglects whether all of the slices in a design are part of the datapath, key schedule, or controller. As the controller and key schedule do not process input data directly, it may be inappropriate to include their slice usage in efficiency measurements. This is a philosophical question we leave to future research for more discussion.

To address these two issues, we propose that evaluation of design efficiency should incorporate the following metrics for resources in the datapath only: Mbps/LUT, Mbps/flip flop, Mbps/BRAM multiplied by the average BRAM memory usage, and Mbps/DSP.

The LUT and flip flop metrics would be useful in applications where a limited number of logic slices are available, as it shows how well the implementation would capitalize on the remaining available logic slice elements. On the other hand, the BRAM and DSP metrics would be informative in the case where these resources are limited, and a designer is concerned about whether a design uses them to their full potential. Additionally, we incorporate the percentage of memory utilized in BRAM into the metric to indicate how effectively the available memory is used.

As Drimer et al. address in their study, we suggest that future studies report as much information as possible about their implementations in order to increase transparency when evaluating AES designs, as we do in this study. For example, neither de la Piedra et al. nor Drimer et al. state how their control mechanisms contribute to resource usage, and neither Wang and Ha nor de la Piedra et al. state how their key

TABLE I
ARCHITECTURE PERFORMANCE COMPARISON

	Resource Usage (#)					Performance Metrics			
	Slices	LUTs	Flip Flops	BRAMs	DSPs	Frequency (MHz)	Latency (Cycles)	Throughput (Gbps)	Target Device
Drimer-AES	296	393	665	9	16	550	84	6.7	Virtex 5
Datapath	259	338	624	8	16				
DRAB-LOCUS	310	909	593	16	18	528	115	7.055	Zynq 7000
Datapath	167	266	256	12	18				

TABLE II
THROUGHPUT PER RESOURCE (MBPS/#)

	LUT	Flip Flop	BRAM	DSP
Drimer-AES	19.8	10.7	837.5	418.75
DRAB-LOCUS	26.5	27.56	220.47	391.94

TABLE III
DATAPATH POWER CONSUMPTION (MW)

	Logic Slices	BRAM	DSP	Signal +Clock	Total
Drimer-AES	56	285	111	39	491
Drimer-Expanded	165	2140	833	74	3212
DRAB-LOCUS	7	259	58	88	412

schedules contribute to resource usage. This makes it difficult to accurately evaluate their efficiency using these metrics. The adjusted efficiency measurements for Drimer-AES and DRAB-LOCUS is shown in Table II, as these two designs state the resource usage for the datapath alone. This table also reveals that it is effective to use block RAM and DSP slices in the AES datapath, as each singular unit is able to process large chunks of data (32-bit for BRAM, 48-bit for DSP), instead of splitting operations across multiple LUTs and flip flops.

C. *Design Effects on Power Consumption*

While there are fewer FPGA AES studies that focus on low-power implementations, power consumption is still an area of interest for many system designers. Drimer et al. do report the power consumption of Drimer-AES, and also present a fully-unrolled implementation, Drimer-Expanded, built on the base structure of Drimer-AES which we include here in our power analysis. The power consumption of these two implementations and DRAB-LOCUS is shown in Table III.

Drimer-AES uses more LUTs and flip flops than DRAB-LOCUS, which is the cause of the higher slice power consumption. However, Drimer-AES uses slightly less BRAM and DSP resources than DRAB-LOCUS, which is not reflected in the power comparison. This may be due to differences in the target device, as Drimer-AES is implemented on a high-performance Virtex 5 FPGA, which generally consumes more power than the Zynq 7000 SoCs. This highlights the importance of disclosing the target device as part of an AES implementation analysis.

On the other hand, Drimer-Expanded consumes nearly 10 times the power of DRAB-LOCUS and Drimer-AES. With a fully-pipelined structure, this design also uses significantly more resources. Table III shows that the large increase in power consumption from Drimer-AES to Drimer-Expanded comes from BRAM and DSP slices, which both have increase factors of 7.5, while the logic slice power only increases by a factor of 3. This shows that using more BRAM and DSP slices in a design will significantly increase the power usage. Therefore, while low-area designs certainly consume less power due to their lower resource usage, ones that use BRAM and DSP slices will have higher power consumption than equivalent designs that primarily use logic slices.

Finally, while the power consumption of a running implementation is important for power supply considerations, it is also desirable to know how much power over time is required to process data. We propose that an additional power metric of nanowatt-seconds required to process a single block is needed to better measure the tradeoffs between latency and pipeline length. Overall, this metric reflects how effectively the design uses its power to process data. The DRAB-LOCUS implementation uses 7.47 nanowatt-seconds, Drimer-AES uses 9.37 nanowatt-seconds, and Drimer-Expanded uses 622.17 nanowatt-seconds to process a single block of input data.

V. CONCLUSIONS

We proposed an area-efficient AES architecture, DRAB-LOCUS, that achieves a balance between resource usage and throughput by incorporating under-utilized FPGA components, such as BRAM tiles and DSP slices. We identified how architectural design decisions influence key trade-offs and discussed new metrics for evaluating the efficiency and power usage of cryptographic accelerators. These metrics provide a measure of how effectively a design capitalizes on the available resources to improve performance. DRAB-LOCUS achieves higher resource efficiency than throughput-focused AES architectures, higher throughput than low-area designs, and more functionality and lower power usage than other area-efficient designs.

References

[1] N. Shylashree, N. Bhat, and V. Shridhar, “FPGA implementations of Advanced Encryption Standard: A survey,” *International Journal of Advances in Engineering and Technology*, pp. 265–285, May 2012.

[2] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie, and X. Zhou, “DLAU: A scalable deep learning accelerator unit on FPGA,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 3, pp. 513–517, Mar 2017.

[3] C. Cong, X. Zhang, Y. Li, S. Huang, J. Xiong, K. Rupnow, W. Hwu, and D. Chen, “FPGA/DNN co-design: An efficient design methodology for IoT intelligence on the edge,” in *Proceedings of the 56th Annual ACM Design Automation Conference*, no. 206, Jun 2019.

[4] J. Hoozemans, J. van Straten, T. Viitanen, A. Tervo, J. Kadlec, and Z. Al-Ars, “ALMARVI execution platform: Heterogeneous video processing SoC platform on FPGA,” *Journal of Signal Processing Systems*, vol. 91, no. 1, pp. 61–73, Jan 2019.

[5] J. Grycel and R. Walls, “DRAB-LOCUS: An area-efficient AES architecture for hardware accelerator co-location on FPGAs,” arXiv, p. 1911.04378, 2019.

[6] *Advanced Encryption Standard (AES)*, National Institute of Standards and Technology Federal Information Processing Standards 197, Nov 2001.

[7] S. Drimer, T. Guneysu, and C. Paar, “DSPs, BRAMs, and a pinch of logic: Extended recipes for AES on FPGAs,” *ACM Transactions on Reconfigurable Technolgy Systems*, vol. 3, no. 1, Jan 2010.

[8] Y. Wang and Y. Ha, “High throughput and resource efficient AES encryption/decryption for SANs,” in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2016, pp. 1166–1169.

[9] A. de la Piedra, A. Touhafi, and A. Braeken, “Compact implementation of CCM and GCM modes of AES using DSP blocks,” in *2013 23rd IEEE International Conference on Field programmable Logic and Applications*, Oct 2013.

[10] D. Kundi, A. Aziz, and N. Ikram, “Resource efficient implementation of t-boxes in AES on virtex-5 FPGA,” *Information Processing Letters*, vol. 110, no. 10, pp. 373–377, Apr 2010.