A Fast Parameterized SHA3 Accelerator

Colin Schmidt
U.C. Berkeley
colins@eecs.berkeley.edu

Adam Izraelevitz
U.C. Berkeley
adamiz@eecs.berkeley.edu

ABSTRACT

Security is an extremely important feature in modern computer systems. The fundamental basis of many security problems, most notably authentication, is a secure hashing algorithm. These algorithms take a message, return a fixed size hash for the message, and require the following important properties: ease of hash computation, inability of generating the message from the hash, inability to change the message and not the hash, and inability to have the same hash for two different messages.

A family of these algorithms has been standardized by NIST and given the name Secure Hashing Algorithm (SHA). The newest of this family is SHA3, formerly known as Keccak, and thus contains the most opportunity for design space exploration. Since this hashing function will be frequently used, requires many operations per input and is generally inefficient on a general purpose CPU, it is an ideal function to be implemented with a hardware accelerator to improve performance (hashes/sec) and energy use (hashes/Watt). In order to facilitate choosing a hashing accelerator for a particular project, a parameterized implementation is developed and explored using automated tools.

General Terms

SHA3, Keccak, Chisel, Design Space Exploration, Accelerator, RISCV

1. INTRODUCTION

Security is an extremely important feature in modern computer systems. As more and more users push their information to the cloud, cloud providers need to protect this growing amount of data. The fundamental basis of many security problems, most notably authentication, is a secure hashing algorithm. These algorithms take a message, and return a fixed size hash for the message and require the following important properties: ease of hash computation, inability of generating the message from the hash, inability to change the message and not the hash, and inability to have the same hash for two different messages. Fortunately, a family of these algorithms has been standardized by NIST and given the name Secure Hashing Algorithm (SHA)[5]. The newest of this family, and thus containing the most opportunity for design space exploration, is SHA3, formerly known as Keccak. Since this hashing function will be frequently used and requires many operations per input, it is an ideal function to be implemented with a hardware accelerator to improve performance (hashes/sec) and energy use(hashes/watt).

The accelerator is implemented as a co-processor adhering to the ROCC interface for integration with the Rocket RISC-V processor. The functionality contract between the accelerator and the programmer is designed to be as simple as possible. The programmer provides pointers to the message and the location to store the resulting hash, as well as the length of the message, using two ROCC instructions. This follows the standard c implementation, and allows for a simple change to use the accelerator rather than a library. The role of the processor is to set up and drive the hashing computations, including any intermediary computations done to find the next message to hash or to validate the hashed results. To actually perform the hash calculation, the scalar processor will only need to ensure the message is in memory and gives its length to the accelerator.

Given this programming model the system designer will decide based on their projects criteria whether or not they should include an accelerator. To help establish this decision the accelerator has been implemented in a parameterized fashion that allows it to trade off performance, area, and energy. These parameters include the number and arrangement of execution resources, and the amount of execution contexts to support simultaneous hashing of multiple messages. All instantiations of the accelerator provide the same interface, but vary in their performance, energy efficiency, and size.

The design space for this project is large, and the best design will depend heavily on the typical use case in the environment for which it is deployed. Tools are used to explore this design space and given a specific weight for each metric, performance, energy, area, a decision can be made about which accelerator to include (if any). Several typical applications hashing requirements are also given to establish reasonable bounds.

In addition to the hardware implementations from the fully explored design space the SHA3 algorithm, written in C, will be run on the Rocket RISC-V core to compare as a software baseline. This will serve as a simple naive baseline that should be easy to beat in both energy and performance. Because this is such a common problem, there are many papers which discuss the results of SHA algorithms running on a variety of different platforms. All of these designs are surpassed in either performance, energy, or both, by some parameterization of the accelerator.

2. BACKGROUND

2.1 SHA3

Secure hashing algorithms represent a class of hashing functions that provide four attributes: ease of hash computation, inability to generate the message from the hash, inability to change the message and not the hash, and inability to have the same hash for two different messages. The National Institute of Standards and Technology (NIST) recently held a competition for a new algorithm to be added to its set of Secure Hashing Algorithms (SHA)[5]. In 2012 the winner was determined to be the Keccak hashing function and a rough specification for SHA3 was established[1]. The algorithm operates on variable length messages with a sponge function, and thus alternates between absorbing chunks of the message into a set of state bits and permuting the state. The absorbing is a simple bitwise XOR while the permutation is a more complex function composed of several operations, χ, θ, ρ, π, ι, that all perform various bitwise operations, including rotations, parity calculations, XORs, etc. The Keccak hashing function is parameterized for different sizes of state and message chunks but this design only supports the Keccak-256 variant with 1600 bits of state and 1088 message chunks.

2.2 Chisel

The accelerator was designed and implemented in a parameterized fashion using the new hardware construction language (HCL) Chisel, being developed at UC Berkeley. Chisel has several features that were used extensively during the design, verification, and evaluation of this project. The major feature used during the design of this accelerator was Chisel's ability to use the full power of its Scala hosting language during the creation of the circuits. This allows for much more expressive composition of modules and control logic.

Chisel also provides multiple backend targets for generation, including C++ and Verilog, allowing the accelerator to be verified functionally in a fast C++ simulator before evaluation using Verilog and synthesis tools.

3. RELATED WORK

There have been many hashing hardware accelerators in the past including those for previous standardizations of SHA[2][3] that have focused on optimizing the algorithm for implementation in silicon. These projects have been able to build upon a large foundation of previous research into the algorithm, uncovering small optimizations throughout the process. This allows previous papers to synthesize many of these ideas together to build an entire system that is better than any previous versions. Our SHA3 accelerator, by contrast, uses the same architecture and algorithm as previous versions, but provides parameterization at the microarchitecture level ito understand its inherent tradeoffs.

The simple SHA3 hardware implementations that exist [4] are based on an exploration of all SHA3 candidates and often ignore control and interface issues to actually instantiate and use the hardware. These implementations were able to provide one optimization of combining two of the permutation functions, into one block. This also provides the possibility of future work to explore if further combination would be beneficial in addition to other various algorithmic transformations similar to those found eventually in SHA2.

Other work focuses on efficient accelerators designs such as [3], despite being focused on FPGA implementation, still provide interesting approaches for saving space and energy. The modular approach taken in [3] allows for different portions of the algorithm to be reimplemented with different blocks that may have different performance characteristics. In addition to strong support for the module approach, a strong focus on evaluation including comparisons to other previous implementations was a useful approach to take in the development of an accelerator. Pushing the modular design forward our design uses parameters to codify and expand the design space, as well as using automated tools to help validate and evaluate this space.

4. PROJECT OBJECTIVES

One goal of this project is to design a highly parameterized SHA3 hardware accelerator where different parameterizations result in different levels of performance, energy efficiency and size. Ideally this creates a design space that is very large and is incapable of being explored exhaustively. Such a large design space leads to the second and third goals of the project: to develop an automated system to intelligently explore the design space to first validate the points and then evaluate them using a VLSI toolflow. The automated tool will facilitate the creation of pareto optimal graphs showing which design points are most valuable. Finally, from a systems design perspective this project should enlighten a designer as to which accelerator parameters they should choose, or whether to include an accelerator.

5. TECHNICAL APPROACH

The accelerator is designed around three sub-systems, an interface with the processor, an interface with memory, and the actual hashing computation system (Figure 1). The interface with the processor is designed using the ROCC interface for coprocessors integrating with the RISC-V Rocket processor. It includes the ability to transfer two 64 bit words to the co-processor, the request for a return value, and a small field for the function requested. The accelerator receives these requests using a ready/valid interface. The ROCC instruction is parsed and the needed information is stored into one of the T execution contexts, only if there is one available. These execution contexts contain the memory address of the message being hashed, the memory address to store the resulting hash in, the length of the message, and several other control fields.

Once the execution context is valid the memory subsystem

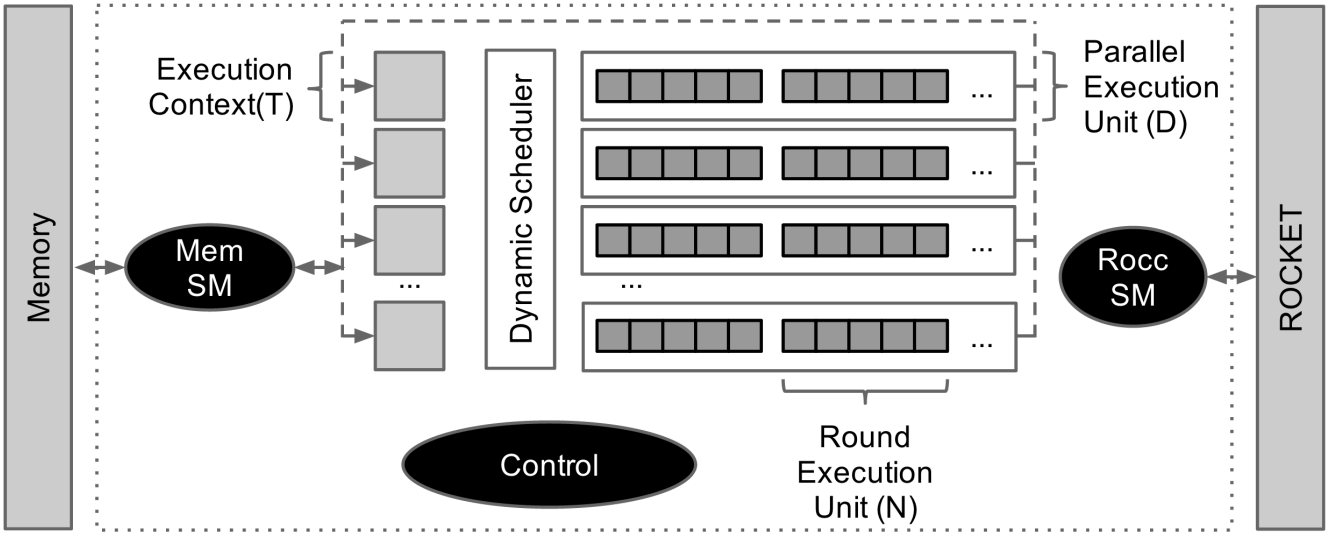


Figure 1: Microarchitecture of the parameterizable SHA-3 design

then begins to fetch chunks of the message. The memory subsystem is fully decoupled from the other subsystems and maintains either T or 4 memory buffers, whichever is smaller. The accelerators memory interface can provide a maximum of one 64 bit word per cycle which corresponds to 17 requests needed to fill a buffer (the size is dictated by the SHA3 algorithm). Memory requests to fill these buffers are sent out as rapidly as the memory interface can handle, with a tag field set to allow the different memory buffers requests to be distinguished, as they may be returned out of order. Once the memory subsystem has filled a buffer the control unit absorbs the buffer into the appropriate execution context, at which point the execution context is free to begin permutation, and the memory buffer is free to send more memory requests.

After the buffer is absorbed, the hashing computation subsystem begins the permutation operations. Because the hashing subsystem has a parameterized number of execution units in parallel, D, as well as a parameterized number of round execution units, N, it requires a dynamic scheduler to determine which execution context are ready to run and on which available execution unit they should be run. Once the message is fully hashed, the hash is written to memory with a simple state machine.

The complexity of this design, including its many parameters, necessitates a system to validate the correctness of each design point and then evaluate them to further drive the design space exploration. In order to do this efficiently for this project, a system was designed to integrate well with Chisel, called Jackhammer. This tool allows for the definition of a set of parameters in the Chisel source which are expanded into a design space during the runtime construction of the design. This space can then be pruned and validated or evaluated. The designer is additionally given the option of which portions of the VLSI design flow to run (e.g. synthesis, place-and-route, or post place-and-route power simulations), and therefore can use a simple RTL simulation to validate their designs are functionally correct or a sequences of VLSI tool runs to generate accurate power and area evaluations. All of this is conveniently automated and frees the designer from painful manual evaluation and testing.

6. RESULTS

Before evaluating the performance of the accelerator it is useful to understand what the overhead of using a more expressive HCL, compared to a language like Verilog, has on the quality of results generated by the VLSI synthesis tools. The previous highest performing implementation [4] has made their Verilog datapath source code available online providing an interesting point of comparison (labeled HPI). Figure 2 represents the results of comparing the accelerator's datapath with parameters that make it most similar to the HPI implementation and using the same synthesis and place and route commands. The area of the accelerator is slightly smaller than the HPI version and consumes significantly less power. This comparison suggests that there is no overhead to adding parameters to the design in Chisel, and that using Chisel over pure Verilog actually provides additional performance benefits.

Each of the design points were evaluated using a simple microbenchmark in which 8 identical hashes, each 8KBs long, were fed to the accelerator in sequence. This benchmark allows accelerators with more resources to use the memory-level parallelism (MLP) inherent in hashing different messages and additional execution resources to improve performance. Each accelerator's performance (product of cycles to complete and the clock period), area and power spent were measured during this benchmark.

In addition, Figure 3 was created by running this benchmark on the C++ simulator and counting the total number of memory requests. This figure shows that the accelerator is able to take advantage of the increased MLP from an increased number of execution contexts, or threads. In addition, increasing the number of execution resources also increases memory utilization because the accelerator finishes the permutations faster and empties more memory buffers.

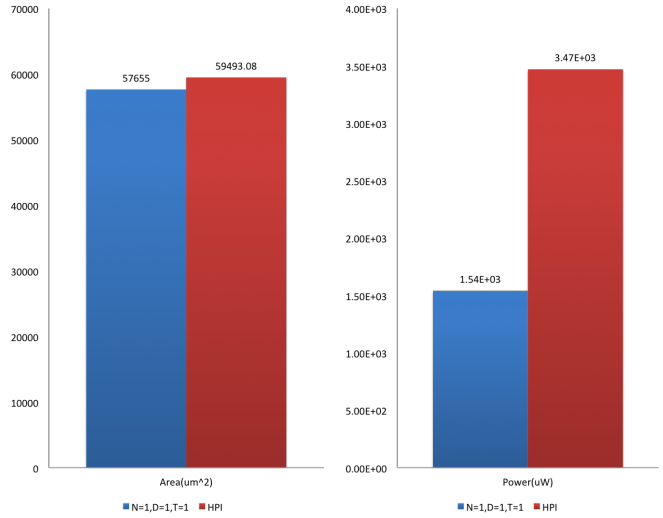


Figure 2: We compare our 1-1-1 parameterized design written in Chisel with the SHA3 design released by [4]. After pushing both designs through place-and-route with equivalent tool constraints, our design has comparable area yet significantly less power is consumed.

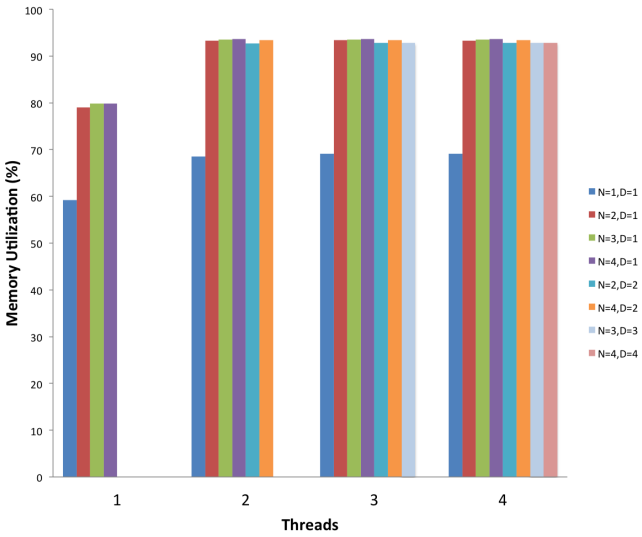


Figure 3: Memory Utilization. As more resources are added, the memory bandwidth is saturated at approximately 93%

The maximum utilization peaks at approximately 93% because the data must reside completely in the memory buffer before it can be absorbed. This detail of our system causes a one cycle gap in memory requests every 17 cycles.

With such a large set of design points and three different, yet all valuable, metrics, it is difficult to determine which designs are the most important to consider when architecting a system. Figure 4 shows different pareto optimal curves for performance, energy, and area. In each graph the origin is the most optimal point, implying that any pareto optimal point is one where no other point is better in both axes. In the performance vs power and area vs performance graphs there are two pareto optimal points, colored green and red. The green circle is the accelerator parameterized at D=1, N=1, T=1 and the red square is parameterized at D=2, N=2, T=2. The measured throughput for these two points is 41.9 Gbits/sec and 85.6Gbits/sec respectively, making it the fastest SHA3 hardware accelerator published to date. These design points represent an optimal allocation of resources given the memory bandwidth, but neither is strictly better than the other unless the only consideration is area and power in which case the smaller, simpler D=1, N=1, T=1 parameterization is the only pareto optimal point.

Figure 5 and Figure 6 show the post place and route layout of the two pareto optimal designs. The total area breakdown for both designs is approximately 20% control logic and 80% datapath. The VLSI tools spread the permutation blocks ThetaModule, RhoPiModule, etc. among the datapath to move the computation nearer to the state elements storing the result. The control logic rests on the edges of the chip determining when to allow the state to be permuted, as well as interfacing with the exterior interfaces of the accelerator: the memory system and the ROCC interface. The increase in area by doubling all parameters is more than a factor of two due to the significant increase in control logic needed to manage multiple parallel execution units in the datapath.

Determining exactly when to use either of the pareto optimal accelerators is difficult to do without the exact specifications of a particular project, but an approximate method is shown in Table 1. In this characterization each column represents a different set of weights placed on each of the three metrics: performance, area, and power. Each row shows the result of the calculation for a set percentage of computation time spent hashing. Several common uses of hashing can be placed along this spectrum; for example, SSL is approximately 3-5% hashing, while a more intensive application such as a Merkle tree computation would be upwards for 80-90% hashing. This type of table allows a systems designer to know for a Merkle tree, if performance or power are the most important metrics, that including an accelerator parameterized at D=1, N=1, T=1 is the optimal system.

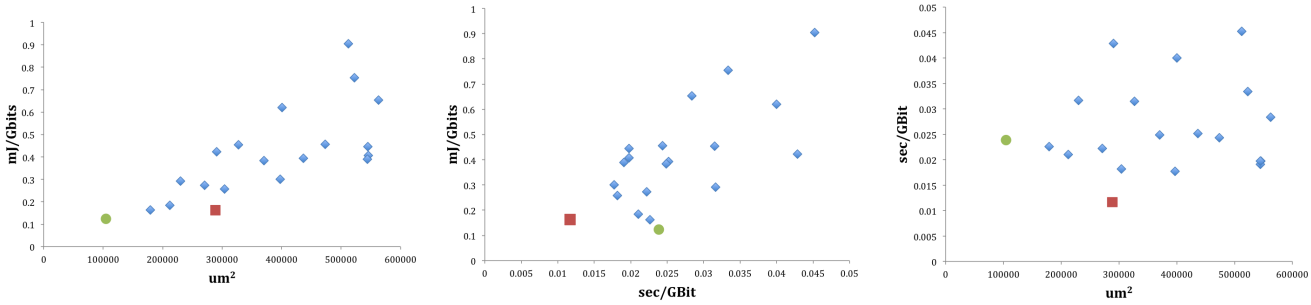
7. FUTURE WORK

Using Chisel and Jackhammer, this project developed a well parameterized SHA3 accelerator and conducted a sizable evaluation. However, many ideas still need exploring. Jackhammer is much more tightly coupled with this accelerator than necessary; a more standalone version should be developed. In addition, the process of using Jackhammer needs to be streamlined into the general Chisel experience to allow users to take full advantage of its autonomous features.

With a more robust Jackhammer there are several more parameters that would be interesting to explore for the accelerator. The biggest detractor from larger and more power-

[2]The best design is defined as the smallest sum of the weighted zscores of each metric, where the titled column had a weight of 20

[2]1-1-1 is the design point where D=1, N=1, and T=1 (see Section 5), while Rocket is the workload running on a general purpose CPU with no accelerator



(a) Power vs Area

(b) Power vs Performance

(c) Performance vs Area

Figure 4: Pareto optimal graphs (closer to origin is better). The green circle is 1-1-1, red square is 2-2-2

Table 1: Optimal design per workload[12]

Example Workload	% Hashing	Performance	Power	Area
	0	Rocket	Rocket	Rocket
SSL/TLS[8]	10	Rocket	Rocket	Rocket
	20	Rocket	Rocket	Rocket
	30	Rocket	Rocket	Rocket
	40	Rocket	Rocket	Rocket
	50	Rocket	1-1-1	Rocket
	60	Rocket	1-1-1	Rocket
	70	Rocket	1-1-1	Rocket
	80	1-1-1	1-1-1	Rocket
Merkle Tree[7]	90	1-1-1	1-1-1	Rocket
SHA3[6]	100	1-1-1	1-1-1	Rocket

ful accelerators was the lack of increased performance due to the memory bandwidth bottleneck. Adding a parameter to specify the number of words returned within a memory request, or the number of independent memory channels, could show that increasing the amount of resources in addition to memory bandwidth would still be pareto optimal. Another set of parameters that were left unexplored included a variety of area and energy parameters. The larger sets of execution contexts would benefit from being able to use SRAM for their state blocks rather than flip-flops. Some of the control logic could be parameterized to be simpler as well, which would decrease energy use at the cost of power. For example, the memory subsystem could issue requests in order and wait for all requests to return before continuing, removing the need to track which requests belong to which thread. Additionally, if the hashing is also suspended during this time, a memory buffer is unnecessary. Other optimizations include classic tradeoffs in the datapath such as time multiplexing components to reduce area at the cost of performance. In general, an increase in the number of parameters could potentially lead to a more interesting design space containing more than two pareto optimal points.

Given a larger set of pareto optimal points, a more thorough systems level evaluation would be more meaningful. To facilitate such an evaluation, the accelerator would need to be integrated completely with the Rocket processor. This integration would allow a full program, such as SSL, to run on the combination of a Rocket processor and the accelerator to enable very accurate performance, area, and energy numbers. With this type of experimental setup, it would also be interesting to analyze a parameterized processor. This additional baseline comparison would allow the designer to choose to increase the performance of the processor or increase the performance of the accelerator to see at what point the truly optimal design lies. Finally, a different accelerator programmer model, where only chunks of the message are provided and a streaming interface is supported, could prove beneficial for certain applications.

8. CONCLUSIONS

Modern computer systems are increasingly concerned with security and are composed more and more of heterogeneous components. One computationally significant portion of security is a secure hashing algorithm. The highly parameterized SHA3 accelerator developed in this paper is designed to solve both of these issues, by providing a high-performance secure hash calculation, and being highly configurable allowing a system on a chip designer to choose the level of performance, energy efficiency and size of the accelerator.

The accelerator has proven to scale its performance with the resources allotted and provides comparable if not better performance than previous implementations. The numerous designs that can be generated by the parameters have been validated with a large suite of test cases using the Jackhammer automated design space exploration tool in conjunction with the Chisel HCL. Within its current memory and processor interfaces, the accelerator's parameters provide two pareto optimal design points that should be con-

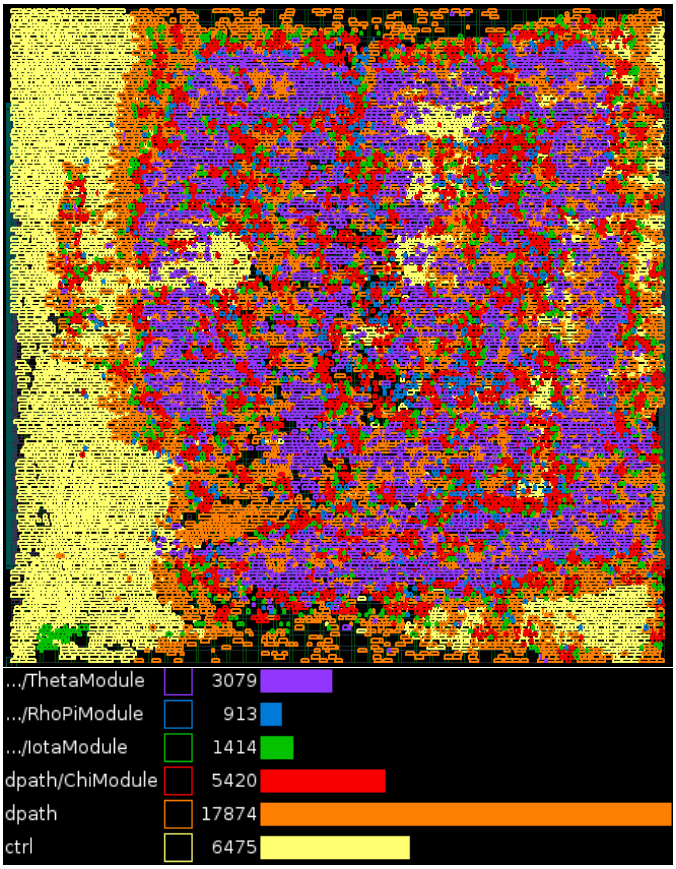


Figure 5: Amoeba plot for 1-1-1. Size is 334x334 um^2

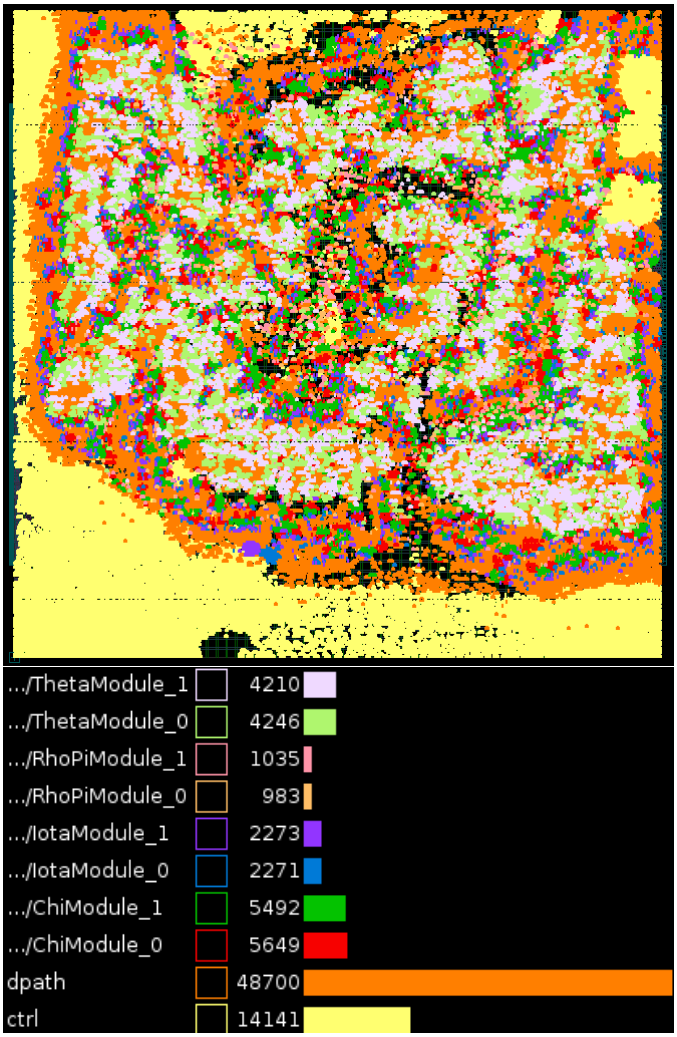


Figure 6: Amoeba plot for 2-2-2. Size is 534x534 um^2

sidered when building a system with a significant hashing element. These two parameterization were found through a large scale automated design space evaluation and represent the fastest, most energy efficient SHA3 accelerators published to date. Determining when to use such an accelerator in a conventional processor co-processor system is also explored, showing that for certain workloads, including a small efficient accelerator is advantageous. Extending the current parameterization to include more energy-efficient trade-offs and exploring more holistic evaluation strategies could lead to even stronger results suggesting the viability of hashing accelerators in computer systems.

9. ACKNOWLEDGMENTS

The authors would like to thank their instructors; Professor Jonathan Bachrach, Professor John Lazzaro, Professor Anthony Joseph, and Professor John Kubiatowicz for a myriad of helpful guidance; their TA Ben Keller, for helpful, and experienced advice and extremely beneficial systems administration; as well as their advisors Professor Krste Asanovic and Professor Jonathan Bachrach, for their support.

10. REFERENCES

[1] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. The keccak reference'. http://keccak.noekeon.org/, January 2011.

[2] R. Chaves, G. Kuzmanov, L. Sousa, and S. Vassiliadis. Cost-efficient sha hardware accelerators. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 16(8):999–1008, 2008.

[3] L. Claesen, J. Daemen, M. Genoe, and G. Peeters. Subterranean: A 600 mbit/sec cryptographic vlsi chip. In *Computer Design: VLSI in Computers and Processors, 1993. ICCD '93. Proceedings., 1993 IEEE International Conference on*, pages 610–613, 1993.

[4] K. Kobayashi, J. Ikegami, S. Matsuo, K. Sakiyama, and K. Ohta. Evaluation of hardware performance for the sha-3 candidates using sasebo-gii. Cryptology ePrint Archive, Report 2010/010, 2010.

[5] N. I. of Standards and Technology. Cryptographic hash and sha-3 standard development. http://csrc.nist.gov/groups/ST/hash/index.html, 2013.

[6] M.-J. O. Saarinen and J. Kornblum. hashdeep. https://github.com/jessek/hashdeep/blob/master/src/sha3.c/, 2012.

[7] weichaoguo. merkle-tree. https://github.com/weichaoguo/merkle-tree/, 2011.

[8] L. Zhao, R. Iyer, S. Makineni, and L. Bhuyan. Anatomy and performance of ssl processing. In *Performance Analysis of Systems and Software, 2005. ISPASS 2005. IEEE International Symposium on*, pages 197–206, 2005.