



Deposited via The University of Sheffield.

White Rose Research Online URL for this paper:

<https://eprints.whiterose.ac.uk/id/eprint/158510/>

Version: Accepted Version

Proceedings Paper:

Yalcin, T. and Kavun, E.B. (2020) Almost-zero logic implementation of Troika hash function on reconfigurable devices. In: 2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig). 2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig), 09-11 Dec 2019, Cancun, Mexico. IEEE. ISBN: 9781728119588. ISSN: 2325-6532. EISSN: 2640-0472.

<https://doi.org/10.1109/reconfig48160.2019.8994780>

© 2019 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other users, including reprinting/ republishing this material for advertising or promotional purposes, creating new collective works for resale or redistribution to servers or lists, or reuse of any copyrighted components of this work in other works. Reproduced in accordance with the publisher's self-archiving policy.

Reuse

Items deposited in White Rose Research Online are protected by copyright, with all rights reserved unless indicated otherwise. They may be downloaded and/or printed for private study, or other acts as permitted by national copyright laws. The publisher or other rights holders may allow further reproduction and re-use of the full text version. This is indicated by the licence information on the White Rose Research Online record for the item.

Takedown

If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

Almost-Zero Logic Implementation of Troika Hash Function on Reconfigurable Devices

Tolga Yalçın

School of Informatics, Computing, and Cyber Systems
Northern Arizona University, Flagstaff, AZ, US
tolga.yalcin@nau.edu

Elif Bilge Kavun

Department of Computer Science
The University of Sheffield, Sheffield, UK
e.kavun@sheffield.ac.uk

Abstract—Blockchain technology has gained immense popularity in the recent years due to its decentralized computing architecture. While it originally emerged as a technology for (crypto)currencies, it has since found many different application areas including (but not limited to) payments, money transfers, smart contracts, supply-chain management, networking, IoT, etc. Initially, it was only Bitcoin, the de facto standard for cryptocurrencies, but then it was followed by several (in fact hundreds of) others. Each new cryptocurrency had or claimed to have certain advantages over Bitcoin, such as transaction speed and cost. However, they all relied on the original idea of distributed ledger where each block has maintained a complete history of each transaction in the network. Blockchain technology has more recently been challenged by two new technologies called Tangle and Hashgraph, which are “directed acyclic graphs”, i.e. in layman’s terms blockchains without blocks and chains. IOTA network is the original Tangle technology, which relies on ternary arithmetic architecture and uses ternary hash function “Troika”. It works on $GF(3)$ and its design follows the sponge construction. Two of the main claims of IOTA are scalability and micro-transactions, both of which are likely to utilize compact hardware platforms in practical implementations. In this paper, an almost-zero logic compact and yet adequately fast hardware architectures of Troika hash function targeting reconfigurable devices are presented. The proposed architectures mainly depend on the utilization of BRAMs on FPGAs. Three different RAM-based hardware implementations have been realized on Xilinx Artix-7 xc7a12tcbg238-3 device; all using only a single BRAM tile with minimal number of LUTs and FFs. The proposed architectures can easily be implemented on different reconfigurable devices with similar efficiency. To the best of our knowledge, this is the first reported hardware implementation of Troika hash function on reconfigurable devices which is also compact and fast.

I. INTRODUCTION

In very simple terms, a blockchain is a growing list of records (blocks) which are linked to each other using cryptographic techniques. Each block contains a cryptographic hash of the previous block, a timestamp, and the transaction data [1]. In other terms, the list of all transactions since the creation of the blockchain is kept by all the blocks in the chain, with new transactions being added on to existing ones. The whole network is fully distributed and decentralized, there is no central authority who has control over the ecosystem [2]. Such an architecture makes it resistant to modification of the data by design; the recorded data in any given block cannot be modified retroactively without alteration of all subsequent blocks and the consensus of the network majority.

As a result of these properties, blockchain has been the underlying technology for cryptocurrencies, especially as the public transaction ledger of the cryptocurrency bitcoin [3]. The introduction of Bitcoin Cash (BCH) [3] design has inspired many other cryptocurrencies. Among these, Ethereum (ETH) [4], Litecoin (LTC) [5], Ripple (XRP) [6] and IOTA [7] are just a few popular examples.

Decentralized architecture of blockchain technology, which makes it so trustworthy and desirable for secure transactions, also makes it so cumbersome. Each transaction within the network must be distributed throughout the whole network, making it extremely slow. The transaction speed for Bitcoin can be as high as tens of minutes [8]. As a result, several attempts have been made in order to speed up blockchain, such as the lightning network [9]. In addition to modifications on the original blockchains, new technologies, namely “Tangle” and “Hashgraph” have also been introduced. Both have certain advantages and also disadvantages over blockchain.

In our study, we focus on Tangle [10], which differentiates itself from other decentralized networks. It is actually not a blockchain. While preserving the decentralization and security features of blockchain, the Tangle forms the distributed ledger like a web of information instead of forming *blocks* to create a *chain* of information. Tangle aims to solve problems like scaling and slow transaction times that other blockchain projects are currently struggling with.

The name of the token in Tangle network, which also is the name of the foundation that has developed the Tangle technology, is IOTA. The security of the core components in IOTA system still relies on the security of a cryptographic hash function and it is crucial that the hash function fulfills the security requirements in order to ensure the validity of the transactions on the Tangle. In previous works, Keccak/SHA-3 [11] has been used as the underlying hash function in IOTA system [12]. In 2018, Troika [13] is announced by CYBERCRYPT as a new ternary hash function for IOTAs ternary architecture and platform. Troika is based on sponge-based construction [14], which uses a ternary permutation function with state and output lengths of 729 and 243 trits, respectively.

One of the main target applications of Tangle is the Internet-of-Things. However, it is very unlikely that today’s IoT devices will be able to cope with proof-of-work (PoW) computational

requirements of Tangle. In fact, it has been demonstrated that both PoW and transaction signing are not practical without hardware-accelerated cryptography on battery-powered devices [15]. Low-cost Field Programmable Gate Array (FPGA) devices, with their capability to implement both cryptographic and logic functions are viable choices as hardware acceleration and implementation platforms for Troika and consequently IOTA. Furthermore, it is also possible to implement Troika hash function utilizing mainly the block RAMs (BRAMs) on the FPGA with minimal usage of look-up tables (LUTs) and flip-flops (FFs), thereby freeing more logic for other functions.

In this work, an almost-zero logic and fast hardware architecture of Troika hash function on configurable devices (FPGAs) is presented. Three different variants of the proposed architecture have been implemented on a state-of-the-art Xilinx Artix-7 FPGA device. Each of these implementations take the same amount of cycles to process a new data block and use 1 BRAM tile. The LUT and register utilization for these three different implementations slightly differ due to different implementation targets. To the best of our knowledge, this is the first academic work on hardware implementation of Troika hash function.

The remainder of this work is organized as follows: The Troika ternary hash function is presented in Section II. Sections III and IV describe the architecture and implementation of Troika on FPGA platform in detail and, finally, Section V discusses the results of the proposed implementations.

II. TROIKA – A TERNARY HASH FUNCTION

Troika [13] is announced by CYBERCRYPT in 2018 as a new ternary hash function designed especially for IOTAs ternary architecture and platform. It is a sponge-based construction (Figure 1) with a permutation designed for ternary platforms. It works on a state of 729 trits with a rate r of 243 trits and capacity c of 486 trits. Its output length is 243 trits. The claimed security level is 243 trits for first and second preimages and 243/2 trits for collisions.

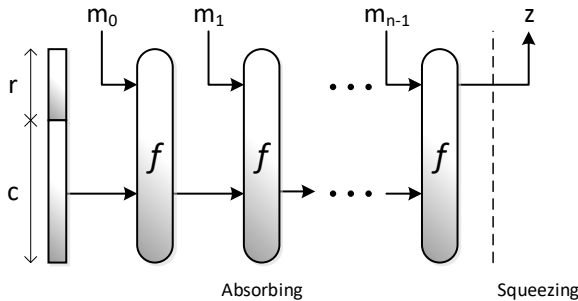


Fig. 1. Overview of sponge construction

The 729-trit state is organized as a $9 \times 3 \times 27$ cuboid of trits. The designers of Troika use the same naming convention as in Keccak/SHA-3 [11] in order to address different parts of the

state. Figure 2 shows this naming convention for a $9 \times 3 \times 27$ cuboid.

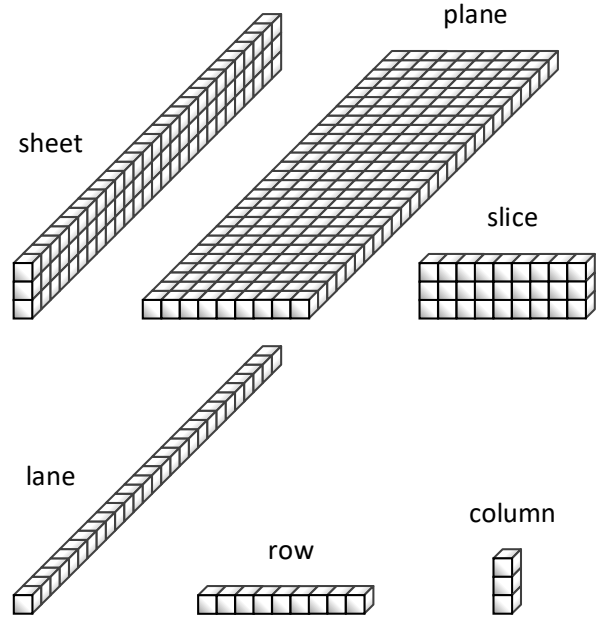


Fig. 2. Naming convention for Troika state ($9 \times 3 \times 27$ cuboid)

For the hash computation, the state is initialized with all zeros. A message is padded by appending a 1 and a number of zeros (to make it a multiple of 243 trits) at its end. The padded message is then formatted into n blocks of $r = 243$ trits each. Each 243-trit message block is then assigned to the rate part of the state followed by a chained call to the f function, which is a 729-trit permutation. Specifically, the 243-trit rate part of the state corresponds to the first nine slices of the state. A 729-trit Troika permutation f is used to update the 729-trit state for 24 rounds.

One round of Troika permutation updates the state using the following operations:

- **SubTrytes:** A 3-trit (1 tryte) Sbox in $GF(3)$ is applied on each tryte of the state. Define

$$F(x_0, x_1, x_2) = (x_0, x_1, x_0 \cdot x_1 + x_2)$$

and the trit permutations

$$\pi(x_0, x_1, x_2) = (x_1, x_2, x_0)$$

$$\rho(x_0, x_1, x_2) = (x_2, x_1, x_0).$$

Then the Sbox output is defined via

$$(x_0, x_1, x_2) \rightarrow \rho(F(\pi(F(\pi(F(x_0 - 1, x_1, x_2)))))).$$

This corresponds to a 3-round source-heavy unbalanced 3-trit Feistel network with the Feistel function F , the cyclic shift π , the additional affine mapping $((x_0, x_1, x_2) \rightarrow (x_0 - 1, x_1, x_2))$ at the input, and the

trit permutation ρ at the output.

The mapping *SubTrytes* on the entire state X is then defined as $X_{i,j,k} \rightarrow s(X_{i,j,k})$.

- **ShiftRows:** Each row of the state is rotated by a constant amount.

This is defined as

$$\begin{pmatrix} X_{.,0,i} \\ X_{.,1,i} \\ X_{.,2,i} \end{pmatrix} \rightarrow \begin{pmatrix} X_{.,0,i} \\ X_{.,1,i} \ggg 1 \\ X_{.,2,i} \ggg 1 \end{pmatrix}$$

for all slices.

- **ShiftLanes:** Each lane of the state is rotated by an amount read from a look-up table.

This is defined as

$$\begin{pmatrix} x_{i,0,.} \\ x_{i,1,.} \\ x_{i,2,.} \end{pmatrix} \rightarrow \begin{pmatrix} x_{i,0,.} \ggg R_{i,0} \\ x_{i,1,.} \ggg R_{i,1} \\ x_{i,2,.} \ggg R_{i,2} \end{pmatrix}$$

where $R_{i,j}$ are read from the look-up table.

- **AddColumnParity:** Trits of each column are updated by adding sum of parities of two adjacent columns. This mapping provides diffusion along columns by adding to each column $x_{x,..,z}$ the parities of the two adjacent columns $x_{x-1,..,z}$ and $x_{x+1,..,z+1}$, where indices are taken modulo their respective dimensions.

- **AddRoundConstant:** A round-dependent constant is added to the (rows of) state.

AddRoundConstant is the only mapping in the round transformation that differs from round to round. It is defined as

$$x_{.,0,.} \rightarrow x_{.,0,.} + RC^r.$$

III. TROIKA ARCHITECTURE ON FPGA

Troika is similar to Keccak with its permutation-heavy structure. However, unlike Keccak where addition and multiplication are bitwise and can be implemented with single XOR and AND gate, respectively, Troika operations are ternary operations, i.e. trit-wise, as shown in Figure 3. The Troika Sbox uses a 3-stage Feistel structure, where one ternary multiplication and one ternary addition takes place at each stage. The gate count (in our case LUT count) for a single Sbox depends on the gate counts for trit-wise addition and multiplication circuits. Therefore, we have decided to start with optimizing implementation of these two gates.

+	0	1	2	×	0	1	2
0	0	1	2	0	0	0	0
1	1	2	0	1	0	1	2
2	2	0	1	2	0	2	1

Fig. 3. Ternary trit-wise addition (left) and multiplication (right) operations

Due to the binary nature of logic implemented within the FPGAs, ternary operations need to be implemented using binary gates. This requires selection of a binary representation for ternary symbols prior to circuit implementation. The most

basic and also one of the most area effective representation is using ‘00’, ‘01’ and ‘10’ for ternary symbols 0, 1 and 2, respectively. Previous studies in the literature suggest other more efficient representations, such as ‘01’, ‘11’ and ‘10’ as given in [16].

Although initially we tried all 24 (4!) possible representations, we decided to go with a different approach, where we represent the ternary symbols with 3 bits, using a scheme similar to one-hot encoding. In our representation, 0, 1 and 2 are represented using ‘001’, ‘010’ and ‘100’, respectively. The truth tables using this representation are given in Figure 4.

+	001	010	100	×	001	010	100
001	001	010	100	001	001	001	001
010	010	100	001	010	001	010	100
100	100	001	010	100	001	100	010

Fig. 4. Truth tables for ternary addition (left) and multiplication (right) using the selected representation

This representation simplified the binary definitions for ternary operations considerably:

- Ternary addition formulas:

$$\begin{aligned} y_2 &= (a_0 \wedge b_2) \vee (a_1 \wedge b_1) \vee (a_2 \wedge b_0) \\ y_1 &= (a_0 \wedge b_1) \vee (a_1 \wedge b_0) \vee (a_2 \wedge b_2) \\ y_0 &= \neg(y_1 \vee y_2) \end{aligned}$$

- Ternary multiplication formulas:

$$\begin{aligned} y_2 &= \neg(y_1 \vee y_0) \\ y_1 &= (a_1 \wedge b_1) \vee (a_2 \wedge b_2) \\ y_0 &= a_0 \vee b_0 \end{aligned}$$

Conversion between default 2-bit ternary representation to our ternary representation is also quite simple, making it possible to store ternary symbols (trits) in 2-bit registers:

- 2-bit ternary (d_1d_0) to 3-bit ternary ($b_2b_1b_0$) conversion:

$$\begin{aligned} b_2 &= d_1 \\ b_1 &= d_0 \\ b_0 &= \neg(b_2 \vee b_1) \end{aligned}$$

- 3-bit ternary ($b_2b_1b_0$) to 2-bit ternary (d_1d_0) conversion:

$$\begin{aligned} d_1 &= b_2 \\ d_0 &= b_1 \end{aligned}$$

Using the 3-bit representation, ternary adder, ternary multiplier and ternary S-box require 3, 3 and 16 LUTs, respectively.

While this coding solves the area problem for a single S-box and it is possible to come up with a serial implementation

using a single S-box, the state size is still a big problem. Troika state requires storage of 729 trits, which is equivalent to 1458 bits. A register-based implementation would occupy just too many registers, which is a precious commodity for a low-cost FPGA. As going with high-end FPGAs would directly contradict with our initial target of a low-cost implementation targeting IoT applications, we opted for a RAM-based architecture.

We efficiently use BRAM tiles of Xilinx 7 Series FPGAs [17] (an Artix-7 device is selected) to store Troika state, where the Troika hashing algorithm is implemented using two dual-port RAMs: 1458-trit (1458x3-bit) RAM-1 and 729-trit (729x3-bit) RAM-2. In the placement these are interpreted as a single BRAM tile consisting of two 18K BRAMs (see Section V). The required state contents are called trit-by-trit in every clock cycle with an algorithm-specific addressing scheme in order to apply the corresponding round's f function operations.

In addition to the RAMs, a single Sbox, a single accumulator to implement parity generator functionality for columns and a round constant generator are used to implement the full Troika functionality. These blocks are implemented on LUTs using combinational logic.

The functional operation of the architecture takes place in a data loading phase (*WriteInput*), two operation phases for each round and result reading phase (*ReadOutput*). It can be summarized as follows:

- **Data loading phase:** Initially, data is divided into 243-trit blocks and padded using the specified Troika padding scheme. This phase is executed by the external processor. The first 243 trits are loaded to the first 243 locations inside RAM-1. The following 486 locations are then loaded with all-zero trits (using our representation with '001' bits). This completes the state initialization step of the algorithm. The initialized state is then applied 24 rounds of the Troika permutation, f .
- **Phase-1:** In this phase, three operations of Troika permutation are executed. These are *SubTrytes*, *ShiftRows* and *ShiftLanes*, respectively. In implementation of the *SubTrytes*, each trit is read from the first 729 locations (first half) of RAM-1, one by one in order. Every read trit is shifted into a 2-trit shift register. At every 3 trits (i.e. 1 tryte), combined output of the shift register is sent into the Sbox, which generates the corresponding 1-tryte parallel output combinationaly. This output is then shifted into a second 2-trit shift register. Output of the shift register is written into both the second 729 locations (second half) in RAM-1 and the RAM-2. However, in writing the result, instead of a linearly increasing addressing scheme, a much more complex addressing scheme is used. This scheme is fact corresponds to the combined *ShiftRows* and *ShiftLanes* permutations. Once all trits in the first half of RAM-1 are processed, *Phase-1* is completed.
- **Phase-2:** In this phase, three read operations are done in parallel: One from trits of the columns in increasing order from second half of RAM-1, and the other two

from the trits of the two adjacent columns from both ports of RAM-2. While all 3 trits of adjacent columns are added to form adjacent column parities, the main column trits are stored in a 3-trit shift register. After every 3rd read trit, required column parities are accumulated and are added to the stored target column trits. This completes the *AddColumnParity* operation. In parallel, the round constant generator circuit, which is an 11-tap ternary LFSR is run. However, its output is updated once in every third trit and added to the *AddColumnParity* output. The resultant output is the round function f output, and it is written back to the first half of RAM-1 in increasing address order. Once, all the columns are processed, *Phase-2* and therefore the corresponding round is completed, and the output is back in the first half of RAM-1.

- **Output reading phase:** The two operation phases are repeated for a total of 24 times, which completes the Troika permutation function on the target data block. Once, it is completed, if there are further data blocks to be processed, the next 243-trit data block is loaded to the first 243 locations of RAM-1. However, this time the following 486 locations are NOT cleared. They are left untouched, and the whole 2-phase 24-round permutation is repeated. Upon completion of processing of all data blocks, the result is read from the first 243 locations of RAM-1 by the host processor.

The operation and data flow of this scheme and the corresponding block diagram are shown in Figure 5 and Figure 6. Note that the 2-to-3 and 3-to-2 bit transform blocks (marked in RED) between memories (registers and RAMs) and combinational logic are only used in implementations 2 and 3.

In this scheme, each phase takes 734 cycles to complete, resulting in a total of 35232 cycles for 24 rounds of Troika. In addition, 243 clock cycles are needed to write the new input block to RAM-1 in every 24 rounds. As stated earlier, it is assumed that data is written to RAM-1 by a host processor and therefore it is not implemented as part of our architecture. Hashing of one message block takes 35475 clock cycles (approximately 35K clock cycles).

In our architecture, registers are only used for temporary data storage and implementation of state machines of the complex addressing scheme and the round constant generator.

IV. IMPLEMENTATIONS

We have implemented three variants of this architecture, which differ from each other in minor details:

- **Implementation 1:** This version implements exactly the functionality explained in the previous section. It requires use of a 1458x3 and a 729x3 RAM. In practice, two block RAMs of a single BRAM slice are used, both in 2048x8 configuration. While not used in this version, this gives us flexibility for future implementations (such as masked implementations with multiple shares).
- **Implementation 2:** In this version, each trit is stored as 2 bits using the default ternary representation. These

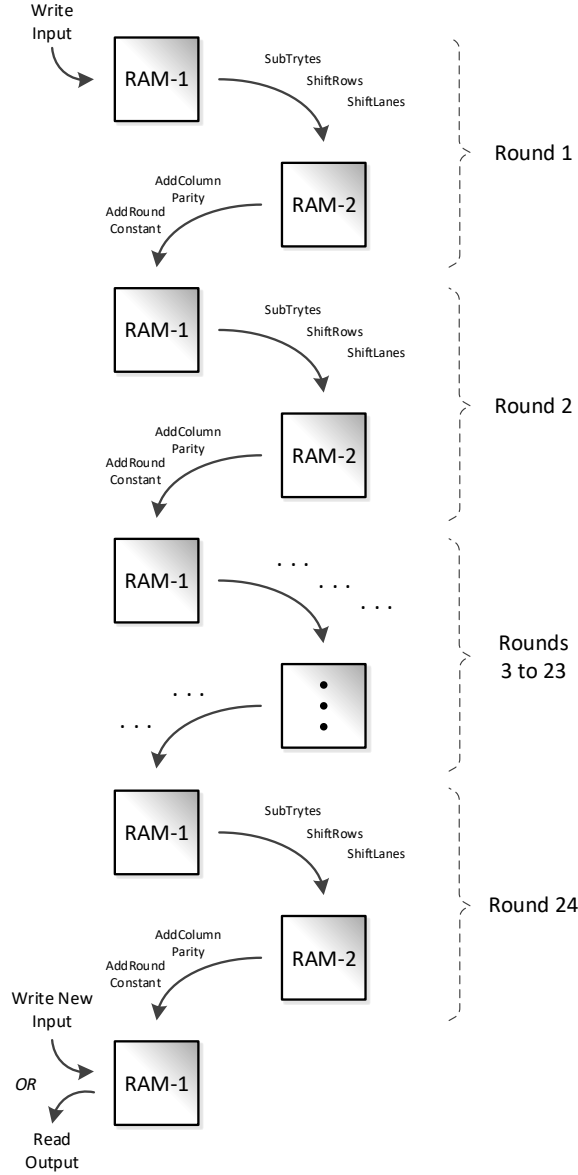


Fig. 5. Data flow of Troika FPGA implementation

2-bit trits are then converted to 3-bit trit representations at the output of the RAMs for Sbox and column parity logic operations, and then re-converted back to 2-bit representations at the input of RAMs. This architecture requires use of a 1458x2 and a 729x2 RAM, which in practice has no effect on the overall area. However, it also uses less LUTs and flip-flops, and leaves more space inside RAMs for possible future extensions.

- **Implementation 3:** This version is derived from the second implementation, i.e. it too stores each trit using 2 bits. However, this time the look-up table where *ShiftLanes* parameters are stored are not implemented as combinational logic. Instead, they too are stored inside

RAM, to be specific, in the 27 locations following the first 729 locations of RAM-2. Moreover, they require 5-bits for storage. In other words, this time RAM-2 size is 756x5 bits. Again, it does not change anything in terms of RAM usage. However, total number of used LUTs is reduced considerably.

V. RESULTS

All designs are simulated and tested in ModelSim using the available test vectors in the Troika specification document. All three implementations are synthesized and placed using Xilinx's Vivado Design Suite v2019.1 on a small FPGA. In order to demonstrate the area effectiveness of our architectures, we opted for the smallest version of Xilinx Artix-7 devices, i.e. xc7a12tcbg238-3. The area results are given in Table I for all implementation flavors. In our implementations, we targeted minimal area. However, the designs also proved to be rather high speed, i.e. about 150 MHz for all three implementations, respectively, corresponding to a hashing speed of 4200 blocks/sec. If encoded, a transaction in IOTA consists of 2673 trytes [18] and the average transaction (confirmation) time is reported to be around 1-2 minutes [19]. So, our hashing speed seems to be in line with IOTA transaction speed according to these values.

TABLE I
TROIKA HASH FUNCTION IMPLEMENTATION RESULTS ON XILINX
ARTIX-7 (XC7A12TCBG238-3) FPGA

	Impl 1	Impl 2	Impl 3	Available
LUT slices	227	206	190	8000
– as Logic	214	193	183	8000
– as Shift Reg	13	13	7	5000
Register slices (as FF)	180	160	191	16000
Block RAM tiles	1	1	1	20
– as RAMB18E1 only	2	2	2	40

As expected, the second version of Troika implementation saves both LUT and flip-flops thanks to the savings coming from the 2-bit operations (except for the switch to the ternary operations). The third implementation saves LUT slices as the shift amount look-up table is stored in the RAM, while using more flip-flops. However, the total number of slices used is lower, resulting in the most compact circuit. All three versions require only two block RAMs, or in other words only a single BRAM tile.

To the best of our knowledge, our work presents the only hardware implementation of Troika hash function to date. In addition, the only available software implementation is the reference implementation provided by CYBERCRYPT. Due to lack of implementations in the literature, it is unfortunately not possible to compare our results with other works.

VI. CONCLUSION

In this paper, three different hardware implementations of Troika hash function, which is designed as a hash function for the cryptocurrency IOTA, are proposed. All implementations follow a RAM-based design and use only a single BRAM tile

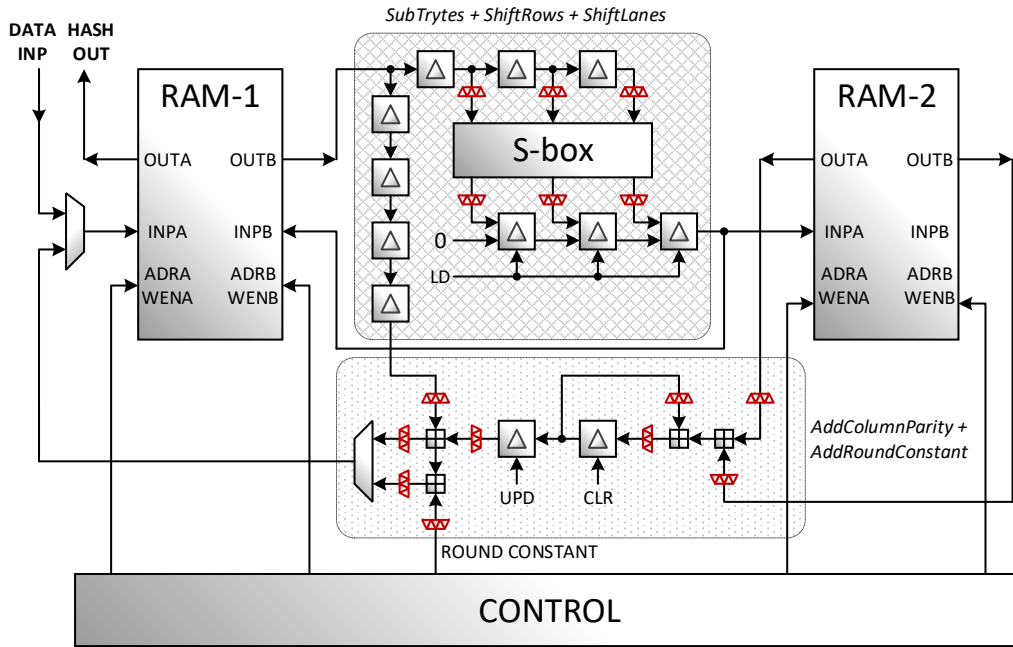


Fig. 6. Block Diagram Troika FPGA implementation

from Xilinx 7 Series FPGAs. The utilization of LUT and flip-flop slice are based on the datapath implementation and the efficient utilization of BRAM spaces. Using this approach, we have achieved an almost-zero logic implementation, however with an acceptable hashing speed. Our speed target is derived from the reported highest speeds in [15]. It is of course possible to lower the total cycle count and hence increase throughput, but at the expense of additional Sboxes and BRAMs, which would contradict with our almost zero logic target in this study.

We can conclude that the proposed implementations in this paper would be mainly interesting for existing FPGA applications that already occupy a lot of LUTs and FFs, but have many available DSP slices and BRAM tiles. As a future work, we plan to implement fully parallel versions of Troika for even faster implementations. We also plan to explore protected versions against physical attacks.

REFERENCES

- [1] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder, *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton, NJ, USA: Princeton University Press, 2016.
- [2] C. Catalini and J. S. Gans, "Some Simple Economics of the Blockchain," NBER Working Papers 22952, National Bureau of Economic Research, Inc, 2016.
- [3] S. Nakamoto, "Bitcoin: A Peer-to-peer Electronic Cash System," 2009.
- [4] G. Wood, "Ethereum: A Secure Decentralised Generalised Transaction Ledger," *Ethereum Project Yellow Paper*, vol. 151, pp. 1–32, 2014.
- [5] J. Reed, *Litecoin: An Introduction to Litecoin Cryptocurrency and Litecoin Mining*. USA: CreateSpace Independent Publishing Platform, 2017.
- [6] I. Takashima, *Ripple: The Ultimate Guide to the World of Ripple XRP, Ripple Investing, Ripple Coin, Ripple Cryptocurrency, Cryptocurrency*. USA: CreateSpace Independent Publishing Platform, 2018.
- [7] R. Alexander, *IOTA - Introduction to the Tangle Technology: Everything You Need to Know About the Revolutionary Blockchain Alternative*. Independently published, 2018.
- [8] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. Gün Sirer, D. Song, e. J. Wattenhofer, Roger", S. Meiklejohn, P. Y. Ryan, D. Wallach, M. Brenner, and K. Rohloff, "On Scaling Decentralized Blockchains," in *Financial Cryptography and Data Security*, (Berlin, Heidelberg), pp. 106–125, Springer Berlin Heidelberg, 2016.
- [9] J. Poon and T. Dryja, "The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments," 2016.
- [10] S. Popov, *The Tangle*. White Paper, 2018.
- [11] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, "Keccak Specifications," 2009.
- [12] T. Pototschnig, "https://medium.com/@pumpck/iota-crypto-core-fpga-3rd-progress-report-3611b030d80d," 2019.
- [13] CYBERCRYPT - Troika Reference Document, "https://www.cybercrypt.com/wp-content/uploads/2019/07/20181221.iota_troika-reference.v1.0.1.pdf," 2018.
- [14] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, *Sponge Functions*. Ecrypt Hash Workshop, 2007.
- [15] A. Elsts, E. Mitskas, and G. Oikonomou, "Distributed Ledger Technology and the Internet of Things: A Feasibility Study," in *Proceedings of the 1st Workshop on Blockchain-enabled Networked Sensor Systems, BlockSys'18*, (New York, NY, USA), pp. 7–12, ACM, 2018.
- [16] W. Geiselmann and R. Steinwandt, "A Redundant Representation of $GF(Q^N)$ for Designing Arithmetic Circuits," *IEEE Trans. Comput.*, vol. 52, pp. 848–853, July 2003.
- [17] Xilinx, "7 Series FPGAs Memory Resources User Guide UG473 (v1.14)," 2019.
- [18] D. Schiener, "The Anatomy of a Transaction https://domschiener.gitbooks.io/iota-guide/content/chapter1/transactions-and-bundles.html," 2018.
- [19] Web resource, "What is the average transaction time in IOTA? https://iota.stackexchange.com/questions/88/what-is-the-average-transaction-time-in-iota," 2017.