2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM) | 978-1-6654-8332-2/22/$31.00 ©2022 IEEE | DOI: 10.1109/FCCM53951.2022.9786081

Leveraging FPGA Runtime Reconfigurability to Implement Multi-Hash-Chain Proof-of-Work

Tong Wu
School of Computer Science and Engineering
University of New South Wales
tong.wu@unsw.edu.au

Oliver Diessel
School of Computer Science and Engineering
University of New South Wales
o.diessel@unsw.edu.au

***Abstract*—In the cryptocurrency mining field, algorithms have been developed to frustrate the development of ASICs that greatly out-compete general purpose hardware in both performance and power efficiency. A class of algorithms that claims to be ASIC-resistant is randomized multi-hash-chain proof-of-work algorithms, such as X16R. For these algorithms, the result of one iteration depends on the chained application of several randomly selected hash functions, which disadvantages fixed-function ASICs due to their inflexibility. FPGAs lie between GPUs and ASICs in terms of raw performance and flexibility. We investigate the usage of FPGAs for this type of proof-of-work, in particular, by leveraging the ability of modern FPGAs to quickly reconfigure at runtime. We implemented a proof-of-concept design that runs the X16R algorithm by partially reconfiguring the FPGA for every hash function in the chain and processing the data in batches. We show that our system achieves better performance when compared to GPUs from the same semiconductor process technology node, while being nine times more power efficient.**

I. Introduction

Initially, on blockchains such as Bitcoin, miners were able to use commodity hardware such as CPUs and GPUs to perform proof-of-work. However, over time ASICs were developed that offered orders of magnitude better performance and energy efficiency over CPUs and GPUs. Today, many blockchains try to introduce ASIC resistance to their proof-of-work algorithms in order to maintain the viability of CPU or GPU mining. One ASIC-resistant method is a *multi-hash* algorithm, which is an algorithm that chains together several different hash functions. An example of such an algorithm is X16R [1], which has the added feature that the order of the chain of hash functions changes randomly every 60 seconds.

In this paper we propose to leverage the dynamic reconfiguration capabilities of modern FPGAs in order to achieve better performance and especially better power efficiency than the GPUs that are typically used to mine X16R.

II. Background

A. Proof-of-Work and Cryptocurrency Mining

Proof-of-work (PoW) schemes are used in blockchains to provide spam protection and guard against bad actors who aim to disrupt the consensus (the agreement across blockchain operators). Evidence of work must be submitted before a participant can modify the blockchain. This work must be computationally expensive, but the validity of the end result must be easy to check. The effort required to find a solution to a PoW is ideally no less than that of random guessing. As such, hash functions are commonly used as PoW, taking in a publicly known value, i.e., the *block header*, and outputting a pseudo-random value. If the hash output is numerically smaller than the target that is set by the blockchain network, then it is a valid PoW.

Typically a block header contains some set fields such as a timestamp, the hash of the previous block and, a nonce field. The nonce is an arbitrary value that the individual prover (commonly called a *miner*) can modify in order to influence the resulting hash. Block headers with nonce values that hash to a valid PoW are called *golden nonces*.

B. X16R Proof-of-Work Algorithm

One PoW algorithm is X16R, which is a multi-hash-chain algorithm. It is comprised of 16 chained hash functions, where the output of each hash function feeds into the next. The order in which the functions are applied is determined by the last 16 nibbles of the hash of the previous block, where each nibble corresponds to one of the 16 possible hash functions. The pool of hash functions are: (0) Blake, (1) Bmw, (2) Groestl, (3) Jh, (4) Keccak, (5) Skein, (6) Luffa, (7) Cubehash, (8) Shavite, (9) Simd, (A) Echo, (B) Hamsi, (C) Fugue, (D) Shabal, (E) Whirlpool, and (F) Sha512.

Since the order in which the hash functions are applied depends on a cryptographic hash, the order is uniformly random and may contain repeated functions (a function that occurs in the chain more than once). For example, if the last 16 nibbles of the previous block are "025FF981EEF8EE2A", then the hash ordering is as follows: Blake $\rightarrow$ Groestl $\rightarrow$ Keccak $\rightarrow \cdots \rightarrow$ Groestl $\rightarrow$ Echo. Also note the repeats of hash functions 2, 8, E and F. The possibility of repeated functions is essential to understanding the difficulty of building a hardware solution for this problem.

We can immediately see that the chance of having **no** repetitions in a chain is extremely small: $16!/16^{16} \approx 1.34 \times 10^{-6}$. Therefore, an ASIC-like (fixed function) hardware implementation of X16R that only implements one hash core per function will almost always be bottle-necked at less than half of its full theoretical throughput. Table I shows the probability of having N or more repetitions in an X16R chain, obtained

978-1-6654-8332-2/22/$31.00 ©2022 IEEE

TABLE I: Probability of $> N$ repetitions in an X16R chain

	>1	>2	>3	>4	>5
Repetitions	99.9%	80.5%	23.1%	3.7%	0.4%

by categorizing a large number of generated X16R-style hexadecimal strings. We calculate the efficiency (utilization) of a fixed-function hardware design, where there is only one core for every hash function, to be 34.4%. That is, on average such a design will be bottle-necked at 34.4% of its peak throughput. This is obtained by summing the probabilities of having exactly N repetitions multiplied by the bottle-necked throughput $(1/N)$.

C. Dynamic Reconfiguration

Modern SRAM-based FPGAs are capable of being reconfigured at runtime, without needing to be power cycled. Additionally, parts of an FPGA can be reconfigured while neighboring logic remains undisturbed, this is called *Dynamic Partial Reconfiguration*.

FPGA configuration binaries are called bitstreams, and bitstreams which configure only a portion of available FPGA resources are called partial bitstreams. These bitstreams can be loaded by writing them to configuration interfaces, for example, to the Internal Configuration Access Port (ICAP) for Xilinx devices or the Secure Device Manager for Intel devices.

D. Related Work

In a paper analyzing the ASIC-resistance of multi-hash-chain algorithms [2], Cho implemented a similar algorithm to X16R on a Xilinx ZU9EG based board. The algorithm consisted of fifteen of the sixteen hash functions used by X16R. As their goal was to study estimated ASIC performance, their design was static. In order to fit fifteen hash functions onto the ZU9EG, they were not unrolled. In absolute terms, Cho's FPGA-based design achieves 1.5 to 2 MH/s, which is worse than GPUs in performance. Instead, Cho's goal was to assess the performance relative to a SHA256D (Bitcoin's PoW) baseline. Cho showed that in general, multi-hash-chains are not ASIC-resistant, with the exception of randomized chains like X16R, which offer some level of ASIC-resistance.

Much previous work has implemented the SHA-3 hash function candidates in hardware [3], [4], [5]. However, their implementations are not suitable for mining applications. The fixed input size of typical PoW algorithms allow the hash functions to be fully unrolled. Many variables in these hash functions thus become constants and their associated operations can be optimized away, such as variable amount shifts becoming simple rewirings at compile time.

There have been commercial implementations of X16R on both GPUs and FPGAs. A popular X16R miner for NVIDIA GPUs is Trex [6], which we used for some of our GPU comparisons. An FPGA based X16R miner is available from Altered Silicon, which teams two VU9P based FPGA boards (2x VCU1525) to achieve an average aggregate hash rate of 240MH/s while consuming 440W. There also exists an ASIC miner, OW1 which uses 72 ASIC chips to achieve 250MH/s while consuming 1500W[1].

III. Design and Implementation

When implementing proof-of-work algorithms for crypto-mining, throughput (hashes per second) and power efficiency (hashes per joule) are the most important metrics. Therefore, the main data-paths of PoW algorithms are typically unrolled and deeply pipelined. As discussed in Section II-B, a static design would be heavily underutilized. A design whereby the hash chain is physically reorganized every block interval, such that data may flow through it without stall, would be ideal. However, it would be difficult to fit sixteen unrolled hash functions in even large high-end FPGAs (e.g. VU13P with 1.7M LUTs). Instead, we propose time-slicing the X16R algorithm into sixteen pieces, and processing the data in batches. The hash functions are sequentially configured onto the FPGA. For the first hash function in the chain, the initial nonces are fed in and the output hashes are buffered in DRAM as so-called *mid-states*. For each subsequent hash function, the mid-states are read from DRAM, processed by the next configuration, and then written back again. In the final iteration (having configured the sixteenth hash function), the output hashes are filtered for golden nonces.

A. Target Hardware

For our FPGA-based X16R implementation we target the Xilinx Virtex Ultrascale+ family of devices. These are FPGAs manufactured on the TSMC 16nm node and were readily available to miners in 2018, when the X16R algorithm was popular. These FPGAs are constructed from one or more *Super Logic Regions* (SLRs), which are separate logic dies that are stacked on top of an interposer. The fastest method to reconfigure these FPGAs is by using the *internal configuration access port* (ICAP). The ICAP on Ultrascale+ devices is capable of 762.9MiB/s throughput (operating at 200MHz) when configuring a local SLR and 476.8MiB/s (limited to 125MHz) when configuring a distant SLR. The ICAP throughput determines how quickly the FPGA can be dynamically reconfigured, which affects the reconfiguration overhead.

We implemented our design using a SQRL FK33 mining board, which is based around the VU33P (medium speed grade) FPGA. This device has a single SLR and contains 440K LUTs and 8GB of *High Bandwidth Memory* (HBM).

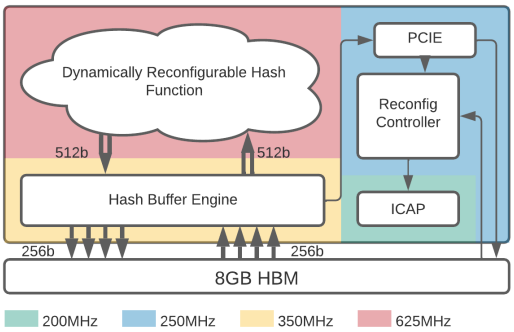
B. System Overview

The system consists of the host computer, on which the FPGA card is connected by a PCIE Gen3 x1 interface, and on the FPGA the available logic is split between the *static region* and the *dynamic region* (Fig. 1). The static region contains the *shell*, which is responsible for communicating with the host, interfacing with the HBM, and controlling the reconfiguration of the dynamic region.

The dynamic region takes up most of the FPGA logic, and is where our hash functions are swapped in and out.

[1]References to the Altered Silicon FPGA miner and OW1 ASIC miner have recently been removed from the web, however one may still find reseller listings that advertise performance.

Fig. 1: System Architecture



C. Static Shell

The static shell consists of the PCIE subsystem, reconfiguration controller and the hash buffer engine. The shell takes up 80K LUTs, which is 15% of a VU33P. While the actual utilization of the static region is only 43K LUTs (52%), the constraints on the positions of the hard blocks (i.e. ICAP, PCIE and HBM interface) required that the bounding box of the static region be expanded to encompass them.

To communicate with the host, we chose to use a PCIE Gen3 x1 link for maximum compatibility with existing mining machines, which typically use one lane PCIE risers in order to attach as many devices to a single host as possible.

On initialization, the host transfers all sixteen partial bitstreams (one for each possible hash function) to a reserved region in the HBM (512MB out of 8GB is reserved for partial bitstreams). We use compressed bitstreams, as generated by Vivado, and the exact size of each partial bitstream varies between different compilation runs. The total size of the sixteen partial bitstreams is approximately 352MB.

The *reconfiguration controller* is a simple finite state machine with sixteen registers. Each register contains a pointer to one of the partial bitstreams associated with one of the sixteen hash functions that are needed in X16R. The registers are written to by the host at the beginning of each block interval. Once the reconfiguration controller has started, it fetches the partial bitstreams in order and writes them to the ICAP, thus programming the corresponding hash function into the dynamic region. In between each reconfiguration, the controller waits for a done signal from the hash buffer engine, indicating that the current batch of mid-states has been pushed through the current hash function and the next configuration can be loaded. While the PCIE subsystem and reconfiguration controller operate at 250MHz, the ICAP is limited to 200MHz (for a maximum bandwidth of 762.94MB/s); an asynchronous FIFO is used to cross clock domains.

The *hash buffer engine* is responsible for reading the mid-states from the HBM, feeding them to the hash function that resides within the dynamic region, and writing the resulting mid-states back to the HBM. The engine is connected to the HBM by eight AXI3 ports, four for writing and four for reading. On the first pass, it feeds the sub-chain with the block header, while incrementing the nonce field. On the last pass, instead of writing the mid-states back to the HBM, it filters the returning stream of hashes for hashes that meet the target (golden nonces). Golden nonces are inserted into a golden nonce fifo, which is periodically polled by the host.

Internally, the hash buffer engine consists of four FIFOs (one for each AXI3 write port), a 4-to-1 multiplexer and a 1-to-4 de-multiplexer. Due to the architecture of the hardened AXI HBM switch, the mid-states are read from the HBM over four 256-bit AXI3 channels. For each AXI3 channel, every two consecutive 256-bit words that are read are recombined to form a 512-bit mid-state every two clock cycles, which is then merged through the multiplexer to form one 512-bit stream that is fed to the hash function. The output of the hash function is de-multiplexed to the appropriate write channel whereby the nonces represent the memory addresses where the mid-state is stored.

Due to the nature of the HBM used on the VU33P, which is DRAM based, bank refreshes can stall the read and write channels for up to 260ns every 3.9μs (the actual refresh interval depends on temperature), therefore the FIFOs on the write ports are needed to prevent mid-states from being dropped (recall that the hash cores are unable to stall for performance reasons). The reason we use four HBM ports instead of two is because the hash cores run at a higher frequency (625MHz) than the hash buffer engine (350MHz).

D. Dynamic Hash Functions

We were unable to find open-source hardware implementations of hash functions that were suitable for crypto-mining applications. Typically in crypto-mining, the input sizes are limited to known sizes such as 80 or 64 bytes. This allows us to unroll and pipeline the entire hash function to optimize throughput/area. We implemented a library of the 16 hash cores needed for X16R in Vitis HLS. Each hash core comes in 80-byte and 64-byte input versions. All cores have a 64-byte output width, have an initiation interval of 1 (i.e., can output one hash every clock cycle), and can be clocked at more than 625MHz. Table II shows the resource utilization for each core. Note that they can all fit within the 360K LUT budget for the dynamic region, but no currently available device could accommodate all of them at once.

The 80-byte versions are only deployed as the initial hash function in a chain. It may be noted that some 80-byte circuits are smaller than their 64-byte counterparts. This is due to the fact that certain hash functions permit further optimization that involves pre-computing the initial mid-state.

IV. Results

When using a batch size of 62.9M nonces (taking up 3.75GB to store mid-states), we found that it takes 2.28s to process a batch of X16R hashes. With an average block interval of 60s, there is some overhead associated with batch processing. If the block interval ends while a batch is in the middle of processing, the partial results are discarded. In the worst case, 2.28s of processing time is lost for each 60 seconds. Performance for

TABLE II: Hash core resource utilization

Core	LUTs	FFs	Core	LUTs	FFs
blake_80	90,189	133,269	shavite_80	199,605	64,504
blake_64	88,826	130,809	shavite_64	146,975	54,813
bmw_80	55,178	81,891	simd_80	324,806	428,267
bmw_64	56,632	81,524	simd_64	321,783	427,834
groestl_80	299,630	84,539	echo_80	201,244	107,892
groestl_64	297,938	84,196	echo_64	199,781	89,305
jh_80	97,794	173,738	hamsi_80	47,904	52,613
jh_64	97,832	171,900	hamsi_64	139,335	153,609
keccak_80	51,054	78,445	fugue_80	175,441	84,290
keccak_64	49,293	74,112	fugue_64	154,279	72,674
skein_80	93,959	113,926	shabal_80	84,471	74,398
skein_64	87,075	103,894	shabal_64	76,664	68,531
luffa_80	66,946	63,541	whirlpool_80	50,045	76,336
luffa_64	107,251	99,022	whirlpool_64	74,140	115,393
cubehash_80	270,925	369,668	sha512_80	79,371	128,062
cubehash_64	327,106	437,639	sha512_64	79,807	127,182

crypto-mining is typically given in hashes per second. Given that we process a batch of $62.9M$ hashes every 2.28 seconds, we get 27.59MH/s. However, we may need to throw away up to $62.9M$ hashes due to being interrupted by the start of a new block interval, which happens on average every 60 seconds. This gives us an additional overhead of 3.8%. Thus, our performance is $(1 - 0.038) \times 27.59 = 26.54$MH/s.

The power consumption of the FPGA board was measured at the 12V inputs of both the PCIE edge and AUX connectors. We measured an average power consumption of 26.1W over eight hours, while running random hash chain configurations.

TABLE III: X16R Performance Comparison

	MH/s	Watts	MH/Joule	Node
GTX 1070	17.8	125	0.14	TSMC 16nm
GTX 1080	17.71	150	0.12	TSMC 16nm
GTX 1080 Ti	19.33	170	0.11	TSMC 16nm
RTX 2060	19.88	90	0.22	TSMC 12nm
RTX 2070	25.5	110	0.23	TSMC 12nm
RTX 2070 Super *	34	110	0.31	TSMC 12nm
RTX 2080	34	108	0.31	TSMC 12nm
RTX 2080 Ti	43	145	0.30	TSMC 12nm
RTX 3090 *	63.99	330	0.19	Samsung 8nm
OW1 ASIC	250	1500	0.17	Unknown
AS VCU1525	120	220	0.55	TSMC 16nm
ZU9EG [2]	2	Unknown	Unknown	TSMC 16nm
VU33P (ours)	**26.54**	**26**	**1.02**	**TSMC 16nm**

In Table III, we list the performance of various GPUs, along with existing FPGA-based solutions, the OW1 ASIC, and our own. The performance metrics for GPUs were taken from NiceHash [7], a popular mining pool, which maintains a database of GPU performance for various mining algorithms. The entries which are marked with an asterisk were benchmarked by us.

V. Conclusion

We showed that our solution has 37% higher performance while being nine times more power efficient in terms of mega-hashes per joule than a high-end NVIDIA GTX 1080 Ti GPU that was manufactured on the same node. While more recent GPUs that were manufactured on newer nodes beat our solution in terms of raw performance, they are still more than three times less power efficient. We also see that our solution is twice as power efficient as the competing FPGA based miner from Altered Silicon.

Our design shows that a time-sliced approach using runtime reconfiguration can be used to achieve good performance where a static design would not be possible. We also note that a hypothetical *unbalanced* hash chain, as described in [2], will not be effective in decreasing FPGA performance. We hope that the cryptocurrency community will consider FPGAs as a separate class of computing device from ASICs. Runtime reconfiguration gives FPGAs the flexibility to optimize an algorithm in both the space and time dimensions. PoW algorithm designers should therefore also consider whether they want properties of FPGA-resistance, separate from ASIC-resistance.

VI. Future Work

In our current design the dynamic region is not fully utilized. We can see from Table II that many hash functions only use a fraction of the dynamically reconfigurable region. By combining hash functions into *sub-chains*, we can save on the number of reconfigurations by processing multiple hash functions in one stage. If we consider sub-chains of two hash functions (of the 64-byte variant), there are 136 combinations (we include a multiplexer so that the order in which the hash functions are combined does not matter), of which 92 fit within the dynamic region (assuming a max utilization of 90%). This would save on average 6 reconfigurations per batch, which leads to an estimated total hashrate of 42MH/s on the VU33P. In order to realise the double hash function scheme, without having to compile all 92 combinations, we plan to employ nested dynamic reconfiguration, where the dynamic region is configured in two stages. The first stage configuration subdivides the dynamic region into two smaller regions of varying sizes, along with a network that connects them. The second stage configures each of the smaller regions as hash functions.

The VU33P is one of the smallest devices, in terms of logic resources, in the Virtex Ultrascale+ family, with only one SLR. A two SLR device such as the VU35P, would effectively double the hashrate, except that ICAPs are local to one SLR and can only reconfigure different SLRs at a reduced bandwidth. The static region in our design should therefore be expanded to encompass the ICAP on the second SLR, and the dynamic region should be split into two, with one in each SLR. This way, we could reconfigure both SLRs independently and in parallel at the maximum ICAP bandwidth. The estimated hashrate for the VU35P is 57MH/s for the the single hash function per reconfiguration scheme, and estimated at 84MH/s for the double hash function per reconfiguration scheme.

Xilinx Versal devices are capable of eight times faster dynamic reconfiguration than the VU33P. Coupled with a more regular fabric, hardened ARM processor, and network-on-chip, this opens up more flexibility in implementing time-sliced algorithms based on runtime reconfiguration.

REFERENCES

[1] T. Black and J. Weight, "X16R: ASIC resistant by design." [Online]. Available: https://ravencoin.org/assets/documents/X16R-Whitepaper.pdf

[2] H. Cho, "ASIC-resistance of multi-hash proof-of-work mechanisms for blockchain consensus protocols," *IEEE Access*, vol. 6, pp. 66 210–66 222, 2018.

[3] R. Shahid, M. U. Sharif, M. Rogawski, and K. Gaj, "Use of embedded FPGA resources in implementations of 14 round 2 SHA-3 candidates," in *2011 International Conference on Field-Programmable Technology*, 2011, pp. 1–9.

[4] B. Baldwin, A. Byrne, L. Lu, M. Hamilton, N. Hanley, M. O'Neill, and W. P. Marnane, "FPGA implementations of the round two SHA-3 candidates," in *2010 International Conference on Field Programmable Logic and Applications*, 2010, pp. 400–407.

[5] H. E. Michail, L. Ioannou, and A. G. Voyiatzis, "Pipelined SHA-3 implementations on FPGA: Architecture and performance analysis," in *Proceedings of the Second Workshop on Cryptography and Security in Computing Systems*, ser. CS2 '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 13–18. [Online]. Available: https://doi.org/10.1145/2694805.2694808

[6] Trex, "Trex miner," https://github.com/trexminer/T-Rex, 2021.

[7] "Leading cryptocurrency platform for mining and trading." [Online]. Available: https://www.nicehash.com/profitability-calculator