Approximate Bitcoin Mining

Matthew Vilim
University of Illinois at Urbana-Champaign
mvilim2@illinois.edu

Henry Duwe
University of Illinois at Urbana-Champaign
duweiii2@illinois.edu

Rakesh Kumar
University of Illinois at Urbana-Champaign
rakeshk@illinois.edu

ABSTRACT

Bitcoin is the most popular cryptocurrency today. A bedrock of the Bitcoin framework is *mining*, a computation intensive process that is used to verify Bitcoin transactions for profit. We observe that mining is inherently error tolerant due to its embarrassingly parallel and probabilistic nature. We exploit this inherent tolerance to inaccuracy by proposing approximate mining circuits that trade off reliability with area and delay. These circuits can then be operated at Better Than Worst-Case (BTWC) to enable further gains. Our results show that *approximation* has the potential to increase mining profits by 30%.

CCS Concepts

•**Hardware** → *Fault tolerance;*

Keywords

Bitcoin; SHA-256; Approximate Computing; Error-Tolerance

1. INTRODUCTION

The Bitcoin cryptocurrency provides a decentralized and distributed method of verifying monetary transactions between trustless parties[1]. Although cryptocurrencies had been proposed previously, Bitcoin was the first to provide a truly trustless solution. Unlike a traditional monetary system which is issued and backed by a single entity, Bitcoin requires no central administrator nor trust between participants.

Traditionally, the difficulty in creating a distributed currency is the need for a scheme to prevent double spending. One party might simultaneously broadcast two transactions, sending the same coins to two separate parties on the network; but without a central server to arbitrate both transactions and decide which is valid, disagreement arises over the true history and ownership of a given coin. Created in 2008, Bitcoin resolves this problem and guarantees consensus of ownership by maintaining a public ledger (Section 2.1) of all transactions, called the *blockchain* [10]. New transactions are grouped together and are checked against the existing history to ensure all new transactions are valid.

Bitcoin's authenticity is assured by those who contribute computation power to its network (known as miners) to verify and append transactions to a public ledger. Miners' willingness to lend their computation power to the network, typically in the form of ASICs dedicated to mining, in exchange for reward (*profit*) is critical to the security and survival of Bitcoin.

In this paper, we observe that Bitcoin mining is a suitable candidate for approximate computing. As we demonstrate, Bitcoin mining is intrinsically resilient to errors; its parallel nature minimizes the propagation of errors, and Bitcoin's distributed verification system detects and invalidates any potentially erroneous solutions. As such, a Bitcoin mining ASIC can be built out of approximate circuits that trade off circuits' reliability for reduced delay and area; an appropriate approximate circuit will maximize profit even when producing results that are not guaranteed to be correct.

We propose two forms of approximation. *Functional approximation* is performed by replacing circuits with approximate versions to reduce area or delay. The reclaimed timing slack may then be used to raise frequency and increase throughput. *Operational approximation* is performed by reducing guard bands and running the circuit with negative timing slack (i.e. at an even higher frequency), allowing occasional timing failures and Better Than Worst-Case (BTWC) operation. Our results show a 30% increase in mining profit from these approximation techniques.

2. BITCOIN MINING

2.1 Overview

To maintain the validity of transactions in the Bitcoin network, there must be an incentive to contribute to verifying transactions within the blockchain. Bitcoin provides this incentive by rewarding miners who contribute with new bitcoins for every block created. Without miners, new transactions cannot be added to the public ledger, and Bitcoin will not function. The mining process is summarized in Figure 1. Mining consists of searching for a cryptographic *nonce* value within a block such that the hash of the block falls within a certain range. The network scales the range to maintain an average rate of one new block every ten minutes.

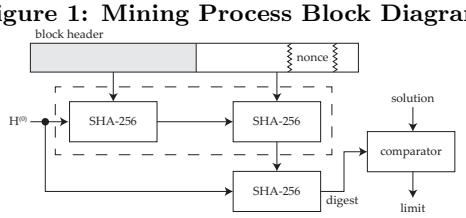
As a result, miners naturally compete against each other

[1]At the time of this writing, Bitcoin's market capitalization is $5.5 billion USD.

DAC 2016 Austin, Texas USA

DOI: 10.475/123_4

Figure 1: Mining Process Block Diagram



to gain a higher fraction of the network's hash rate in order to maximize reward. In a race to capture the network's rewards, miners have developed increasingly sophisticated solutions, culminating in the development of Bitcoin ASIC accelerators [12]. A miner's revenue is determined by the accelerator's hash rate (GHash/s); operating costs are determined by its energy efficiency (GHash/J).

The mining algorithm is shown in Algorithm 1. In short, mining is a search for the nonce value that results in a double SHA-256 hash (Appendix A) value less than a given *threshold*. The nonce is a 32-bit field within a 1024-bit block *header*. In order to verify transactions at a steady rate, this threshold varies over time as a function of difficulty $D(t)$. Difficulty is adjusted by the network regularly such that a solution is expected to be found approximately every 10 minutes, regardless of the network's collective hash rate.

By their very nature, hash functions are designed to be non-invertible, so mining is performed by brute force, guessing nonce values and comparing the hash output. This task is perfectly parallel as multiple hashes may be computed at once. It follows that one's probability of finding a solution is proportional to one's hash rate. The first miner to find a valid nonce broadcasts the value on the network for verification and is rewarded with newly minted digital (bit)coins.

Algorithm 1 Mining Process

```
1: nonce ← 0
2: while nonce < 2^32 do
3:     limit ← ((2^16 − 1) ≪ 208)/D(t)
4:     digest ← SHA-256(SHA-256(header))
5:     if digest < limit then
6:         return nonce
7:     else
8:         nonce ← nonce + 1
9:     end if
10: end while
```

2.2 Related Work

Although some work has been done to improve the performance of SHA-256 ASICs in context of other applications [6] [9], no published research, to the best of our knowledge, attempts to optimize ASICs for Bitcoin mining. The most closely related work is by Courtois et al. [5] who explore mining optimizations from an algorithmic perspective. Their central observation is that the first half of the block header (shown in gray in Figure 1) does not change across nonce iterations, so its hash may be precomputed. Since this precomputation cost is amortized across 2^{32} nonce iterations, it halves the cost of the first SHA-256 round (shown within the dashed line in Figure 1). Our paper is the first work to explore hardware optimizations, specifically approximation-based optimizations, unique to Bitcoin mining.

2.3 Baseline Hardware

2.3.1 Implementation

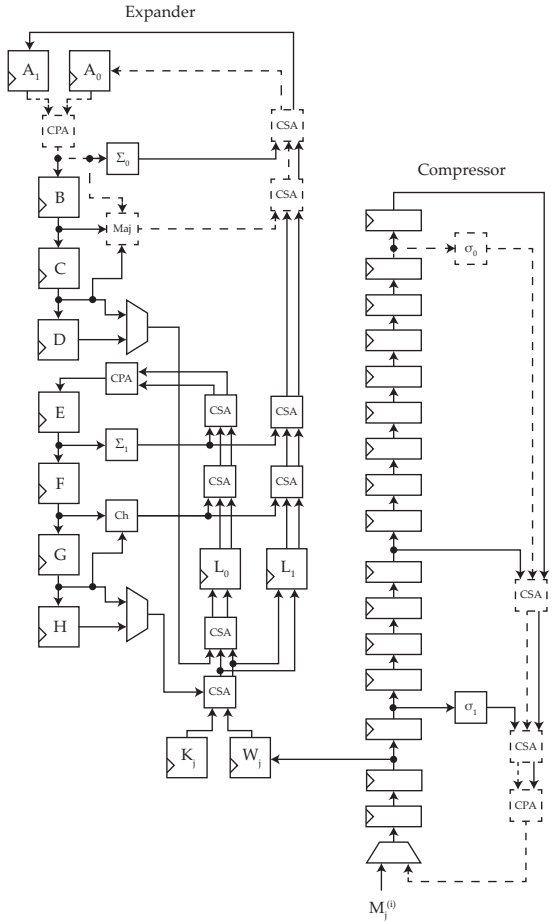
For our studies, we selected as baseline the SHA-256 ASIC design outlined by Dadda et al. [6]. A summary of SHA-256 is provided in Appendix A. The hashing core in this design is implemented as two parallel pipelines, the Compressor (Line 9 of Algorithm 2) and the Expander (Line 3 of Algorithm 2) shown in Figure 2. The logic functions Ch, Maj, Σ_0, Σ_1, σ_0, and σ_1 in the figure are defined in Appendix A.

A single iteration of the algorithm's compression and expansion loops are performed each clock cycle. The expander circuit receives a new 32-bit message chunk $M_j^{(i)}$ every j^{th} cycle and feeds the compressor the expanded message through register W_j. Conversely, the compressor receives a 32-bit chunk of the expanded message and 32-bit constant K_j every j^{th} cycle and compresses these sequences. After 64 cycles, the final 256-bit hash is given by $A_0 + A_1$, B, C, D, E, F, G, H.

In order to reduce delay, most additions are performed by carry-save adder (CSA) trees to avoid unnecessary carry propagation. The ultimate carry propagation is performed only once by some form of carry-propagate adder (CPA) (e.g. ripple-carry adder (RCA) or carry-lookahead adder (CLA)).

2.3.2 Tradeoffs

Figure 2: SHA-256 Pipeline Datapath



In general, ASIC designers seek the implementation that maximizes profit. A miner's instantaneous profit $p(t, f)$ at time t and frequency f is a function of the mining yield $Y(t)$ (USD/GHash), hash rate $H(f)$ (GHash/s), power consumption $P(f)$ (kW), and cost of electricity $C_e(t)$ (USD/kWh).

$$p(t, f) = H(f) \cdot Y(t) - P(f) \cdot \frac{C_e(t)}{60 \cdot 60} \quad (1)$$

As such, Bitcoin mining ASIC design presents a trade-off between a design's area A and delay $1/f$. For fixed die area, any reduction in area allows more hashing cores to be allocated per die, and any reduction in delay implies a corresponding increase in frequency and throughput. Hashing is perfectly parallel so we expect $H(f) \propto f/A$. Thus, designs that minimize the delay-area product in order raise $H(f)$ should be expected to maximize profits.

3. MINING APPROXIMATION

3.1 Motivation

Given the delay-area trade-offs presented above, we propose approximation as a technique to reduce delay and area of Bitcoin mining circuits, thereby increasing profits. Hashing on a Bitcoin mining ASIC is embarrassingly parallel and does not require any communication between cores; this limits the propagation of hardware approximation errors. Furthermore, the hash of all new blocks generated by the ASIC are verified by the rest of the Bitcoin network; any invalid solutions (outside the difficulty range) broadcast on the network by the ASIC would be immediately rejected.

An approximate Bitcoin miner with high false positive rate (invalid solutions[2] that appear valid) could incur significant overheads (either broadcasting or verifying invalid solutions[3]). Fortunately, Bitcoin miners have an inherently low false positive rate. The Bitcoin mining algorithm ensures that the valid solution space (difficulty range) is a minute subset of the 256-bit hash space. For a uniform error distribution, the probability an approximate solution appears valid depends only on the difficulty range, regardless of accuracy. Thus, the probability an invalid solution appears valid (a false positive) can at most be the probability a valid solution is found by an accurate miner. By design, the solution rate for the entire network is roughly once every ten minutes (Section 2), so a single approximate miner will find a valid solution at much larger intervals, on average. Therefore, false positives must also occur at intervals much larger than ten minutes. An ASIC miner performs on the order of 10^9 Hashes/s, so the cost of a single hash verification at intervals larger than ten minutes is negligible.

The larger cost of approximation is false negatives (valid solutions that appear invalid). These hashes represent missed opportunities because a potentially sound solution[4] may be overlooked. These errors occur undetected and uncorrected; thus, a miner's effective hash rate is lowered.

3.2 Effect of Approximation on Profits

In the presence of approximation, the effective hash rate changes. A fraction $E(f)$ (error rate) of the computed hashes will be incorrect, and a normalized reduction in area $\hat{A}$ may occur. We assume any reclaimed area is allocated towards additional hashing cores. A miner's effective hash rate $\tilde{H}(f)$ due to approximation is then given by[5]:

$$\tilde{H}(f) = \frac{1 - E(f)}{\hat{A}} \cdot H(f) \quad (2)$$

Combining with Equation 1, these results suggest an estimate of profit in the face of approximation:

$$\tilde{p}(t, f) = \tilde{H}(f) \cdot Y(t) - \tilde{P}(f) \frac{C_e(t)}{60 \cdot 60} \quad (3)$$

3.3 Functional Approximation

To identify what component(s) in the hashing pipelines should be approximated, we analyzed the critical paths in the two hashing pipelines (Section 2.3.1). The critical path of both pipelines are equal and drawn as dashed lines in Figure 2. The Expander's delay is $delay(\sigma_0) + 2 \cdot delay(CSA) + delay(CPA)$, and the Compressor's delay is $delay(CPA) + delay(Maj) + 2 \cdot delay(CSA)$. Furthermore, we observe the critical path through the carry-propagation logic of CPA dominates the other terms. Thus, the carry-propagate adders (CPA) are good candidates for approximation.

In general, adder designs provide a trade-off between area and delay. For example, an n-bit ripple-carry adder (RCA) propagates signals in $O(n)$ time with $O(n)$ area, but certain carry-lookahead adders (CLA) propagate in $O(\log_2(n))$ time with $O(n \log_2(n))$ area, reducing delay but increasing area [8]. The parallel prefix form Kogge-Stone adder (KSA) minimizes delay at the expense of area [7]. A 16-bit Kogge-Stone parallel prefix graph is pictured in Figure 3. The graph's breadth is proportional to the adder's width n, and its depth (propagation delay) is $O(\log_2 n)$; as a result, its area grows as $O(n \log_2(n))$. Since hash rate is inversely proportional to delay as well as area (Section 2.3.2), we considered all three adders — RCA, CLA, and KSA — as the baseline adder implementations.

Approximate variants of these base adders retain their trade-offs but reduce the delay and area by an additional factor at the expense of inaccuracy. The basic principle of approximate addition is that carry propagation chains longer than a certain length are a rare event [13]. By allowing certain carry propagation patterns to generate erroneous sums, the logic may be simplified, reducing area and delay.

However, only a small number of approximate adder variants will be interesting. Bitcoin mining is particularly sensitive to errors in addition. The sensitivity derives from three CPA modulo 32-bit additions each iteration, so there will be $64 \cdot 3 = 192$ additions in a single round of SHA-256, each with error rate E_{CPA}. The error rate of a single round in the hashing core, therefore, is:

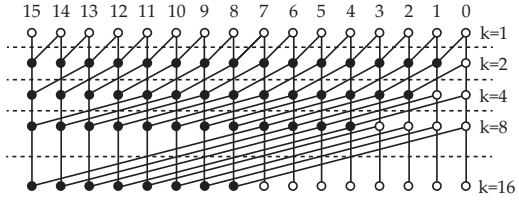
$$E_f = 1 - (1 - E_{CPA})^{192} \quad (4)$$

If the E_f target is 2%, E_{CPA} cannot be higher than 10^{-4}. This result limits the choices of approximate adders suitable for mining.

We consider two approximate adder designs in this work. In [14] rearranging carry-lookahead logic of a CLA adder is

[2]A solution is valid if its hash falls within the difficulty range.
[3]A false positive may be verified locally or by the network.
[4]A solution is sound if it is valid and its hash is accurate.
[5]This expression is conservative; it is possible for a valid but unsound solution to be a sound solution to another nonce.

Figure 3: KSA Parallel Prefix Graph (*n=16*)



proposed to construct a reconfigurable adder. This gracefully-decaying adder (GDA) may be configured for a certain area-delay tradeoff. We select the $GDA_{(1,4)}$ configuration with a 16-bit carry chain as it lowers the error rate to an acceptable threshold.

We also consider an approximate KSA design [7]. Inspecting the graph structure in Figure 3, the maximum length of carry propagation k of an n-bit KSA ($k = n$ for an accurate adder) doubles at each level of the graph. Thus, pruning the lower levels reduces the length of carry propagation, decreasing area and delay. Inputs that generate more than k consecutive carries[6] will produce erroneous outputs. We consider KSA_{16} and KSA_8 implementations.

4. METHODOLOGY

4.1 Simulation

Various approximate and non-approximate versions of the hashing core—one expander and one compressor pipeline per core—were implemented using System Verilog and synthesized using Synopsys Design Compiler [4] and a 65nm TSMC GP cell library. Place and route was performed using Cadence SoC Encounter [1]. The hashing cores differ in their choice of the adder replacement for the CPAs in their datapath.

4.1.1 Functional

An analytical derivation of the approximate adders' error rate E_{CPA} is not straightforward [13]. Instead, Monte Carlo simulations were performed with uniform random inputs ($\approx$ 1 million samples), a confidence interval of 95%, and 5% relative error. The results are listed in Table 1.

4.1.2 Operational

In addition to simplifying logic through functional approximation, approximation can also be performed by tolerating occasional timing violations. Instead, a Better Than Worst-Case (BTWC) operation can be allowed. We estimate the operational error rate $E_o(f)$ for each adder configuration through simulation at variable frequency. At each discrete frequency step, SDF files generated by place and route were used to perform gate-level timing simulations in ModelSim-Altera [3]. Monte Carlo simulations were performed as before. During simulation, the resulting hash vectors were compared against the correct values to determine the error rate at each frequency.

4.2 Profit Model

While it is possible to derive profits directly from our designs in Table 2, the calculated profit values may not be credible since the designs neglect the optimizations that commercial mining ASICs may perform. Instead, we select an existing commercial ASIC for profit calculations. We choose a Bitmain BM1385 [2] with hash rate of $H_0 = 38.8$ GHash/s and power consumption of $P_0 = 10.2$ W at nominal frequency and voltage. We assume that the ASIC is implemented using a KSA_{32} design since KSA_{32} minimizes the hashing pipeline's delay-area product (Table 2). To determine how profits change between adder designs, we calculate normalized changes to area, delay, and power with respect to KSA_{32}, using data from Table 2. We assume that the same relative changes would occur to the Bitmain ASIC in terms of area, delay, and power when its adders are changed.

For example, to predict the change in profits from adopting a $GDA_{(1,4)}$ design, we first calculate the normalized changes to area, delay, and power between $GDA_{(1,4)}$ and KSA_{32} based hashing core (Table 2). Next, we scale the Bitmain ASIC's area, delay, and power by these normalized values to predict the modified Bitmain ASIC's area, delay, and power. Finally, profit is derived from these predicted values, the Bitcoin mining difficulty, exchange rate, and price of electricity.

The error rate at each operating point is found through simulation (Section 4.1). Each simulated SHA-256 round has error rate $E_i(f)$, the sum of functional and operational error rates. Bitcoin requires two rounds for each nonce iteration; hence, we can extrapolate to calculate cumulative error rate $E(f)$, assuming the hash inputs and outputs to be uniform random variables.

$$E_i(f) = E_f + E_o(f) \qquad E(f) = 1 - [1 - E_i(f)]^2 \tag{5}$$

The frequency of each design is swept above its nominal value f_0 while keeping voltage fixed. Hashing is completely parallel, so the hash rate $H(f) \propto f$. The design's normalized operating frequency is $\hat{F}(f) = f/f_0$. Combining with Equation 2, we expect the effective hash rate to be:

$$\tilde{H}(f) = \left(\frac{1 - E(f)}{\hat{A}}\right) \cdot H_0 \cdot \hat{F}(f) \tag{6}$$

At 65nm with high duty cycle, dynamic power dominates leakage power in the designs, so $P(f) \propto f$, implying:

$$\tilde{P}(f) = P_0 \cdot \hat{F}(f) \tag{7}$$

Substituting these expressions into Equation 3, we determine $\tilde{p}_0(t_0, f)$, the predicted profit at time t_0 of the approximate Bitmain ASIC.

5. RESULTS

We perform the synthesis and simulations discussed above for each adder configuration. Table 1 lists the adders' delay and area. Each adder was inserted into the hashing core pipelines in the CPA slots indicated in Figure 2. The resulting hashing core area and delay are provided in Table 2. Approximate variants are highlighted in gray. Figure 4 shows the error rate-frequency characteristic $E_i(f)$ of each hashing core for various adders after simulating a full round of SHA-256. The resulting frequency-profit relation is shown in Figure 5.

[6]We denote these adders as KSA_k.

Table 1: Adder Comparison

Adder	delay (ns)	area (μm^2)	delay · area ($ns \cdot \mu m^2$)	P (mW)	E_{CPA}
RCA	4.13	1723	7116	0.170	0
CLA	1.40	3453	4834	0.889	0
$GDA_{(1,4)}$	1.18	3016	3558	0.950	1.90×10^{-5}
KSA_{32}	0.94	3863	3631	0.867	0
KSA_{16}	0.82	3491	2862	0.814	4.60×10^{-5}
KSA_8	0.72	2920	2102	0.715	2.26×10^{-2}

Table 2: Hashing Core Comparison (Expander & Compressor Pipelines) for Different Adder Choices

Adder	delay (ns)	area (μm^2)	delay · area ($ns \cdot \mu m^2$)	P (mW)	E_f
RCA	4.78	44,058	210,0597	7.19	0
CLA	2.63	47,097	123,865	12.1	0
$GDA_{(1,4)}$	2.32	46,641	108,207	13.8	$7.27 \times 10^{-}3$
KSA_{32}	1.86	48,801	90,769	17.33	0
KSA_{16}	1.73	47,829	82,744	19.0	8.79×10^{-2}
KSA_8	1.58	46,299	73,152	20.4	1.00

Figure 4: Frequency/Error Rate Trade-off for Hashing Cores

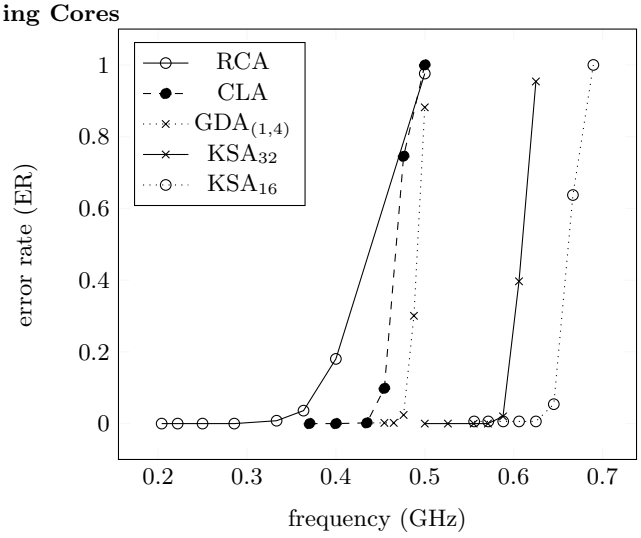
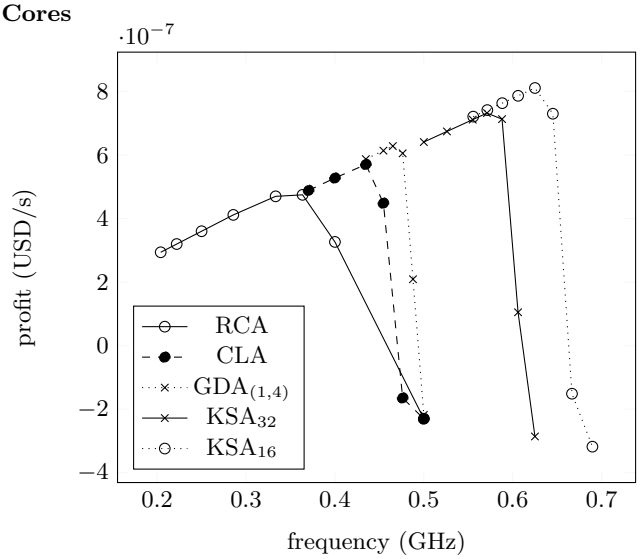


Figure 5: Frequency-Profit Trade-off for Hashing Cores



There are several conclusions to be drawn from the results. First, the results show that approximation is feasible in the context of Bitcoin mining since some approximate adder choices raise profits with respect to their exact implementation. For example, observing the frequency-error characteristics of Figure 4, the hashing cores corresponding to both approximate adders, $GDA_{(1,4)}$ and KSA_{16}, have negligible error rates at nominal frequency. Also, their nominal operating frequencies are higher than their non-approximate counterparts, CLA and KSA_{32} respectively. Consequently, Figure 5 shows that profits of both approximate adders at nominal frequency are greater than that of the corresponding accurate adders.

Second, the results show that approximation can increase mining profits significantly. For example, KSA_{16} performs significantly better than its non-approximate counterpart, producing 15% greater profit at its nominal frequency. A further increase in profit can be gained by operating the design past its nominal frequency. As shown in Figure 5, both KSA designs produce approximately 15% greater profit compared to nominal at their peaks. This indicates KSA_{16} can raise profits by 30%, 15% from functional approximation and 15% from operational approximation.

Third, while mining profit depends on both delay and area of the hashing core, the results show that in a choice between adders with low delay and low area, adders with low delay should be chosen to maximize mining profits. For example, KSA_{32} generates more profits than both RCA and CLA at all frequency operating points in spite of the fact that the error rate of both RCA and CLA rises more slowly when pushed past nominal frequency. This is not surprising considering that while adders are on the critical path of the hashing core, their contribution to the overall area of the hashing core is small (Tables 1 and 2). This result indicates designers should always choose parallel prefix form adders to maximize profits. In particular, approximate adder designs should mimic parallel prefix adder trade-offs.

Finally, many approximate designs are unsuitable for Bitcoin mining. KSA_8, in fact, leads to a hashing core error rate of approximately 100% (Table 2). At such high error rates, mining profits are negative (i.e. revenue does not even offset electricity costs).

6. CONCLUSION

We have proposed approximation as a technique to improve the profits of Bitcoin mining. Bitcoin mining is a particularly good candidate for approximation because its parallelism mitigates error propagation and a built-in verification system detects any false positives. Furthermore, we have identified adders as beneficial choices for approximation in hashing cores in a mining ASIC. However, not all approximate adders yield increases in profit. Profits are maximized by adders that minimize delay at the expense of area, and approximate adders should be chosen accordingly. Moreover, profits may be improved by operating the hashing cores at Better Than Worst-Case (BTWC) operating points, past their nominal frequencies. We have showed that a Kogge-Stone adder using functional and operational approximation has the ability to raise profits by 30%.

7. REFERENCES

[1] Cadence SoC Encounter User's Manual. http://cadence.com.

[2] List of Bitcoin mining ASICs. https://en.bitcoin.it/wiki/List_of_Bitcoin_mining_ASICs. Accessed: November 24, 2015.

[3] ModelSim-Altera User's Manual. https://www.altera.com.

[4] Synopsys Design Compiler User's Manual. http://synopsys.com.

[5] N. T. Courtois, M. Grajek, and R. Naik. The unreasonable fundamental incertitudes behind bitcoin mining. *CoRR*, abs/1310.7935, October 2013.

[6] L. Dadda, M. Macchetti, and J. Owen. The design of a high speed ASIC unit for the hash function SHA-256 (384, 512). In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition Designers' Forum (DATE)*, February 2004.

[7] D. Esposito, D. De Caro, E. Napoli, N. Petra, and A. Strollo. Variable latency speculative Han-Carlson adder. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 62(5):1353–1361, May 2015.

[8] S.-L. Lu. Speeding up processing with approximation circuits. *Computer*, 37(3):67–73, Mar 2004.

[9] H. Michail, G. Athanasiou, A. Kritikakou, C. Goutis, A. Gregoriades, and V. Papadopoulou. Ultra high speed SHA-256 hashing cryptographic module for ipsec hardware/software codesign. In *Proceedings of the 2010 International Conference on Security and Cryptography (SECRYPT)*, pages 1–5, July 2010.

[10] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf. Accessed: November 7, 2015.

[11] National Institute of Standards and Technology (NIST). FIPS PUB 180-4 secure hash standard (SHS). August 2015.

[12] M. B. Taylor. Bitcoin and the age of bespoke silicon. In *Proceedings of the 2013 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, CASES 2013, 2013.

[13] A. Verma, P. Brisk, and P. Ienne. Variable latency speculative addition: A new paradigm for arithmetic circuit design. In *Design, Automation and Test in Europe, 2008. DATE '08*, pages 1250–1255, March 2008.

[14] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu. On reconfiguration-oriented approximate adder design and its application. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 48–54, Nov 2013.

APPENDIX

A. SHA-256 ALGORITHM

Algorithm 2 presents a basic description of SHA-256. For details on message padding, initial hash values $H^{(0)}$, and constants K_j, see [11].

- The message M is divided into N 512-bit blocks $M^{(0)}, M^{(1)}, \ldots, M^{(N-1)}$. Each of these blocks is further subdivided into 16 32-bit words $M_0^{(i)}, M_1^{(i)}, \ldots, M_{15}^{(i)}$.
- The intermediate hash value $H^{(i)}$ is composed of 8 32-bit words $H_0^{(i)}, H_1^{(i)}, \ldots, H_7^{(i)}$.
- $Ch(x, y, z) \equiv (x \wedge y) \oplus (\neg x \wedge z)$
- $Maj(x, y, z) \equiv (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$
- $\Sigma_0(x) \equiv x \ggg 2 \oplus x \ggg 13 \oplus x \ggg 22$
- $\Sigma_1(x) \equiv x \ggg 6 \oplus x \ggg 11 \oplus x \ggg 25$
- $\sigma_0(x) \equiv x \ggg 7 \oplus x \ggg 18 \oplus x \gg 3$
- $\sigma_1(x) \equiv x \ggg 17 \oplus x \ggg 19 \oplus x \gg 10$

Algorithm 2 SHA-256

function SHA-256(M)
  for i from 0 to $N-1$ **do**
    for j from 0 to 15 **do**
      $W_j = M_j^{(i)}$
    end for
    for j from 16 to 63 **do**
      $W_j = \sigma_1(W_{j-2}) + W_{j-7} + \sigma_0(W_{j-15}) + W_{j-16}$
    end for

    for j from 0 to 63 **do**
      $t_0 \leftarrow h + \Sigma_1(e) + Ch(e, f, g) + K_j + W_j$
      $t_1 \leftarrow \Sigma_0(a) + Maj(a, b, c)$
      $h \leftarrow g$; $g \leftarrow f$
      $f \leftarrow e$; $e \leftarrow d + t_1$
      $d \leftarrow c$; $c \leftarrow b$
      $b \leftarrow a$; $a \leftarrow t_1 + t_2$
    end for

    $H_0^{(i)} \leftarrow H_0^{(i-1)} + a$; $H_1^{(i)} \leftarrow H_1^{(i-1)} + b$
    $H_2^{(i)} \leftarrow H_2^{(i-1)} + c$; $H_3^{(i)} \leftarrow H_3^{(i-1)} + d$
    $H_4^{(i)} \leftarrow H_4^{(i-1)} + e$; $H_5^{(i)} \leftarrow H_5^{(i-1)} + f$
    $H_6^{(i)} \leftarrow H_6^{(i-1)} + g$; $H_7^{(i)} \leftarrow H_7^{(i-1)} + h$
  end for
  return $H^{(N-1)}$
end function