ASIC Design for Bitcoin Mining

Yiqiu Sun
University of Michigan
Ann Arbor, Michigan
sunsusan@umich.edu

Haichao Yang
University of Michigan
Ann Arbor, Michigan
hcyang@umich.edu

Wentao Zhang
University of Michigan
Ann Arbor, Michigan
zwtao@umich.edu

Yufeng Gu
University of Michigan
Ann Arbor, Michigan
yufenggu@umich.edu

ABSTRACT

This paper first gives a brief introduction to the bitcoin and how the SHA-256 hash function is related with the bitcoin mining. We examine how the hash function is implemented on the CPU and GPU. Three ASIC designs (Naïve, Novel Counter-Based, Pipeline) are then given, beating the CPU and GPU in terms of power efficiency and latency. Finally, the costs of all hardware designs are compared. The code can be found at the github repository[1].

1 INTRODUCTION

Cryptocurreny has become increasing relevance in the financial world[8]. Although many cryptocurrencies had been proposed, bitcoin was the first among them to provide a truly trustless solution. As a form of currency with no central administrator, Bitcoin's authenticity is represented by the fact that the more computation power is contributed to its network, the larger reward (profit) one can get. Therefore, a bitcoin mining accelerator with higher energy-efficiency and cost-efficiency is in growing demands.

A number of hardware designs have been proposed so far, including approximate computing[6], quasi-pipelined hash functions [9] and Goldstrike 1 architecture[8]. The purpose of this paper is to present and analyze some hardware implementations (Naïve version,) of the bitcoin mining in terms of energy-efficiency and cost-efficiency.

2 BACKGROUND

The bitcoin miner source code can be found on github[2], and is surprisingly simple. The basic computation is the SHA-256 Hash, a cryptographic hash function widely used in security applications and protocols.

2.1 SHA-256 Algorithm

SHA-256 Algorithm takes a message M as input and returns a 256 bits number as the result. The message M is first padded and then divided in N 512-bit blocks $M^{(1)}, M^{(2)}, ...M^{(N)}$. Each block 1 is made up by 16 32-bits words $w_0^{(i)}, w_1^{(i)}, ...w_{15}^{(i)}$. The first 16 words will be extended into the remaining 48 words $w[16..63]$ of the message schedule array. The 512-bit blocks and message schedule array will be passed to the compression function to generate the final 256 bits result. Each round is dependent on the last round, creating a stable chain of dependencies between operations. The compression function algorithm is shown in Algorithm 1.

Algorithm 1: SHA-256 Algorithm Compression function Loop

Result: Take a 512-bit block (a,b,c,d,e,f,g,h) and message schedule array W as input, return a 256 bit hash number

```
// Initialize working variables to current hash value
a := h0, b := h1, c := h2, d := h3, e := h4, f := h5, g := h6, h := h7;
// Compression function main loop
for i from 0 to 63 do
    S1 := (e rightrotate 6) xor (e rightrotate 11) xor (e rightrotate 25);
    ch := (e and f) xor ((not e) and g);
    temp1 := h + S1 + ch + k[i] + w[i];
    S0 := (a rightrotate 2) xor (a rightrotate 13) xor (a rightrotate 22);
    maj := (a and b) xor (a and c) xor (b and c);
    temp2 := S0 + maj;
    h := g;
    g := f;
    f := e;
    e := d + temp1;
    d := c;
    c := b;
    b := a;
    a := temp1 + temp2;
end
// Add the compressed chunk to the current hash value:
h0 := h0 + a, h1 := h1 + b, h2 := h2 + c, h3 := h3 + d, h4 := h4 + e, h5 := h5 + f, h6 := h6 + g, h7 := h7 + h;
// Produce the final hash value (big-endian)
hash := h0 append h1 append h2 append h3 append h4 append h5 append h6 append h7
```

[1] https://github.com/susansun1999/eecs570_final_project
[2] https://github.com/bitcoin/bitcoin/blob/master/src/miner.cpp

3 IMPLEMENTATION ON GENERAL PROCESSORS

In the evolution of compute systems intended for Bitcoin mining, there are some notable challenges and developments starting from general purpose machine like CPU and GPU.

3.1 CPU

The basic program running on CPU can leverage existing high-performance libraries for implementing SHA256. Although separate rounds of a SHA256 computation cannot be parallelized, CPU can leverage multi-thread cores to achieve parallelism to some degree. Meanwhile, some of the operations inside a round are parallelizable. However, typical multicore CPU have extra hardware and OS overheads for memory coherence, resulting in wasted performance and energy efficiency[10].

3.2 GPU

An open-source OpenCL miner was released on the web in October 2010, and it was rapidly optimized and adapted by several open-source efforts. Compared with CPU, GPU can achieve much higher average throughput of threads due to its speciality in fast context switching.

Since the SHA256 hash computation does not exercise the memory system or floating point units heavily, many of the critical paths and bottlenecks in the GPU are not exercised. This implies that the system can be pushed beyond the normal bounds of reliability[10].

4 ASIC DESIGN

An application-specific integrated circuit (ASIC) is an integrated circuit (IC) chip customized for a particular use, rather than intended for general-purpose use. To the opposite of CPU and GPU, ASIC intends to achieve lower latency and power consumption at the cost of generality. In this section, three ASIC designs (Naïve, Counter-Based, Pipeline) for the SHA-256 algorithm are presented.

4.1 Naïve ASIC Design

The baseline naïve design contains an array of eight registers to store the values. The value a and e will be updated every cycle, while other values will be shifted one index up. This design suffers from long critical path latency (marked in red). And the power in shifting unchanged values are wasted.

4.2 Novel Counter-Based Design

Both the pipeline and naïve version need to shift data between registers in each clock to feed the computation module. However, only two value will be updated in each clock cycle, which means most work to shift the data to the new place is useless. Based on this observation, we proposed a novel design to use a counter to choose where to collect and store the data .

4.2.1 Mathematical Background. As mentioned in the section 2.1, each chunk is divided into 8 32 bits blocks (a, b, c, d, e, f, g, h). In each iteration, the SHA-256 updates blocks a and e with new_a and new_e, while shifts the block values to the next blocks and keep the index of each block the same, e.g.
value: $new\_a \rightarrow a, a \rightarrow b, b \rightarrow c, c \rightarrow d, new\_e \rightarrow e, e \rightarrow f, f \rightarrow g, g \rightarrow h$
index: $a = 0, b = 1, c = 2, d = 3, e = 4, f = 5, g = 6, h = 7$.

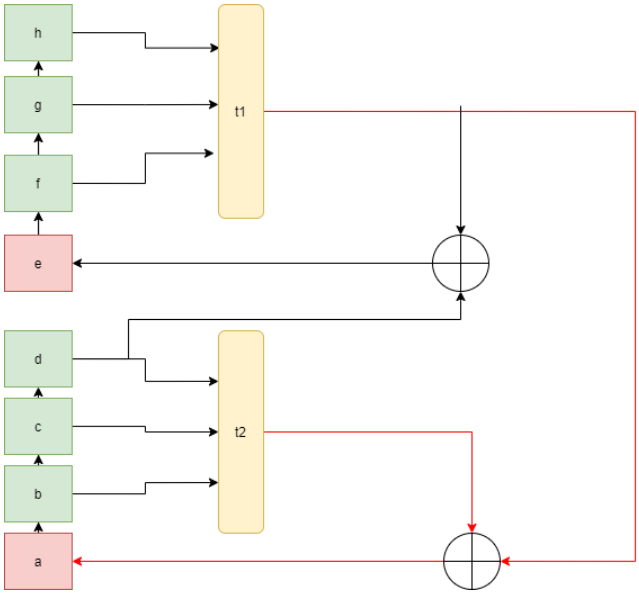

Figure 1: Baseline Naïve Design

We can transform the flow of data into flow of index. That is keeping the data in their original places while treating each block as the next block by changing the block indexes according to the iteration.

Define X_i as the index for the element X in the iteration i. In the i_{th} iteration, the new_a and new_e should be written to place a_{i+1} and e_{i+1} correspondingly, which is the same as h_i and d_i. The index for each block in each iteration can be represented as below. We can use i mod counter_value to look up the correct index for iteration i. For example, in the 3rd iteration, the indexes for a to h is $[3, 4, 5, 6, 7, 0, 1, 2]$.

$$\begin{aligned} a_i &= [0, 1, 2, 3, 4, 5, 6, 7] \\ b_i &= [1, 2, 3, 4, 5, 6, 7, 0] \\ c_i &= [2, 3, 4, 5, 6, 7, 0, 1] \\ d_i &= [3, 4, 5, 6, 7, 0, 1, 2] \\ e_i &= [4, 5, 6, 7, 0, 1, 2, 3] \\ f_i &= [5, 6, 7, 0, 1, 2, 3, 4] \\ g_i &= [6, 7, 0, 1, 2, 3, 4, 5] \\ h_i &= [7, 0, 1, 2, 3, 4, 5, 6] \end{aligned} \tag{1}$$

4.2.2 Architecture Design. Figure 2 shows the top-view of the Novel Counter-Based Design. Same as the naive design, block values are stored in an array of registers (result array). However, instead of shifting every block value among registers every cycle, only two register values (new_a and new_e) will be updated, while keeping other values in their original registers. A counter, allocation muxes and set of computation muxes are introduced to achieve this goal.

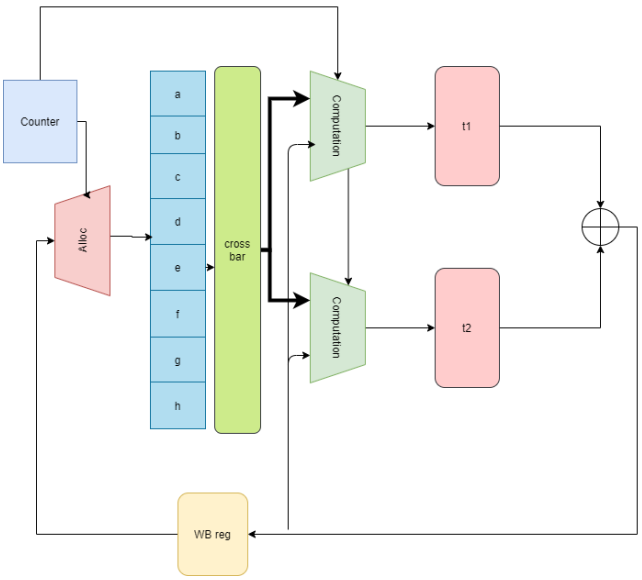

Figure 2: Novel Counter-Based Design

The counter is used to represent which iteration the accelerator is currently processing. Based on this counter value, the allocation mux determines which two registers will be updated with new values, and the set of computation muxes choose the values for computation unit by referencing the index look up table mentioned above.

The computation unit t1 and t2 will calculate the temp1 and temp2, as mentioned in the section 2.1, add them with d to get new_a and new_e, and feed them to the write back register.

The Write Back register will record the result in the current iteration and write back the result to the result array in the next cycle. At the same time, the result is forwarded for the next iteration. This is intended to reduced the critical path between the set of computation muxes and the allocation mux.

4.3 Pipeline Design

Pipeline breaks circuits with long critical paths by introducing clock memories, thus increasing the throughput of the circuits. The circuits are divided into several sections, with regarding to stages in the pipeline. The clock frequency can be increased with reduced critical path in the pipeline.

The pipeline implementation of SHA-256 consists of three stages. Register M and L are inserted to save the temporary values. Two multiplexers are introduced to select G or H in stage 1, and select C or D in stage 2.

Stage 0: Register W and K are loaded. Sum of W, K and $mux2$ are calculated and ready to feed into M.

Stage 1: Register M is updated. Hash values $E-H$ are updated. Sum of Ch, $S1$ and M are calculated and ready to feed into L. The new value of E is also calculated at the mean time.

Stage 2: Register L is updated. Hash values $A-D$ are updated. The new value of A is calculated by sum of Maj, $S1$ and L.

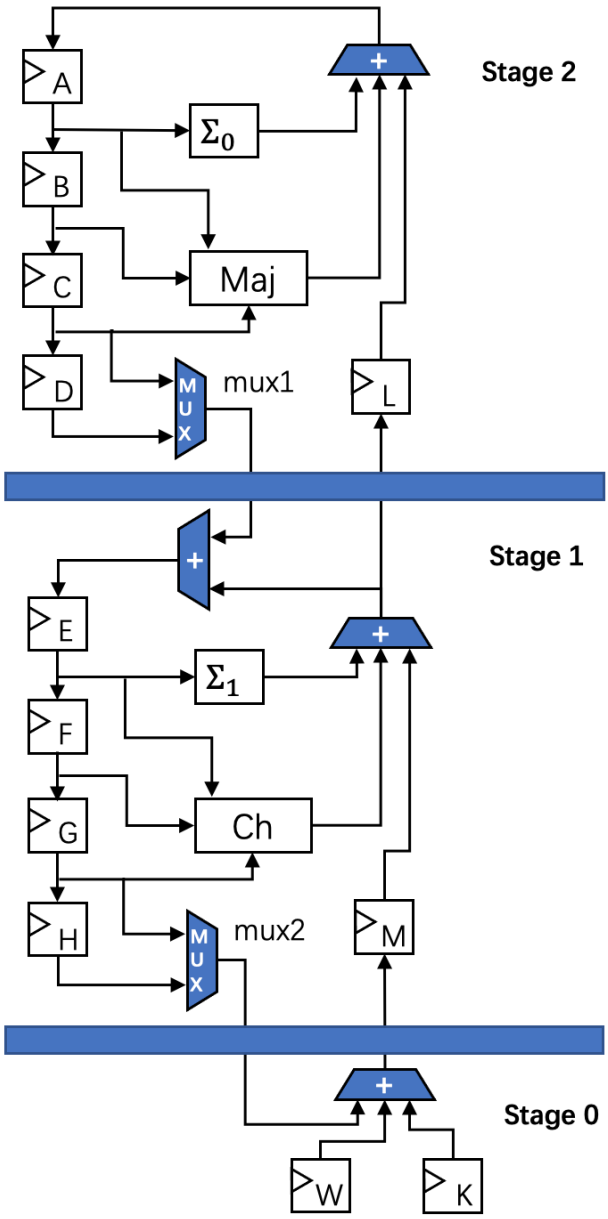

Figure 3: Pipeline Design

In the first clock cycle, only stage 0 is executed. Meanwhile, $A-H$ are loaded with the initial hash values: $A(0) - H(0)$. Register M and L are cleared. $mux1$ outputs $D(0)$ and $mux2$ outputs $H(0)$. In the second clock cycle, both stage 0 and 1 are executed. Register M is updated. $mux1$ outputs $D(0)$ and $mux2$ outputs $F(0)$. In the third clock cycle, all stages are executed. Both register M, L are updated. $E-H$ are updated with new hash value: $E(1) - H(1)$. $mux1$ outputs $C(0)$ and $mux2$ outputs $F(0)$. The fourth clock cycle repeats the computation in the third one, except that $A-D$ are updated with new hash value: $A(1) - D(1)$. From now on, the pipeline has been initialized and the computation in the fourth clock cycle will be

repeated for 60 cycles until the ending stages in the pipeline. In clock cycle 64, The W and K are not loaded anymore and it is the last time for execution of stage 1. In clock cycle 65, $E-H$ are not updated and register M is cleared. In clock cycle 66, circuit outputs $A-H$ the pipeline implementation of SHA-256 is all done.

5 EVALUATION

5.1 Synthesis Results

The ASIC design is described in System Verilog and synthesized using Synopsys Design Compiler. The target technology library is the lec25dscc25_TT library, featuring 130 nm process. The area and timing results are given in Table 1. Area figures are expressed in square micron units; the latency column expresses the number of clock cycles that are needed to complete the hashing of one 512 bit message block times the clock cycle. The latency is expressed in nano seconds.

It is clear from the table that the area is traded for speed. Both Novel counter design and Pipeline design achieve great speed up compared with naive design. However, the pipeline utilizes area better than novel counter design. The pipeline design has area overhead of 1.81x and 3.05x speedup, while novel counter has 2.09x and 1.6x speedup.

	Latency(ns)	Area(um^2)
Naïve Design	$64 \times 9.76 = 624.6$	229453
Novel Counter	$64 \times 6.04 = 386.56$	480556
Pipeline	$66 \times 3.10 = 204.6$	416344

Table 1: ASIC synthesis result

5.2 CPU and GPU Performance

We also tested out the respective performance of CPU and GPU. We ran the C++ version of SHA256 code on Intel Xeon Gold 6240 CPU and its CUDA version on one Nvdia Tesla V100 GPU [11]. For both design, we make sure we run large enough iterations for the machine to reach *saturation*.

The average hash latency for one thread is less trivial for GPU because GPU adopts a specialized scheme to hide individual thread latency. We experimented the average latency per hash at different number of threads on GPU. From the results shown in Fig. 4, our GPU reaches a saturation at a capacity of about 2^{16} threads. After this threshold, the performance of GPU degrades exponentially. We got the final average hash latency by running different iterations with 2^{16} threads on GPU and applying curve-fitting towards the results (shown in Fig. 5). The average hash latency for both CPU and GPU is then given in Table 2.

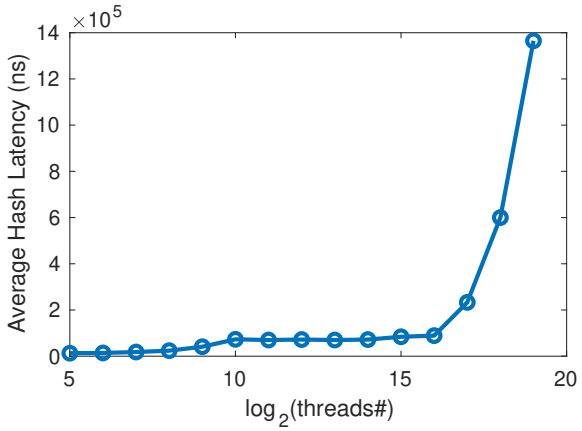

Figure 4: Average hash latency at different threads on GPU

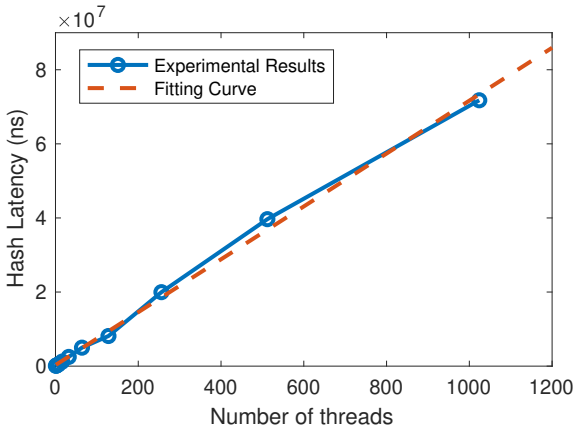

Figure 5: Average hash latency at different iterations on GPU

	Latency(ns)
CPU	347.3
GPU	70977.5

Table 2: CPU and GPU simulation result

5.3 Energy Efficiency Analysis

The power of CPU is measured by running 48-thread (24 cores * 2 SMT width) hash function 100 million times on Intel Xeon Gold 6240 CPU, using powertop to record the power and time and taking the average to reach the final result.

The power of GPU is measured by referencing the max power in NVIDIA TESLA V100 data sheet[3]. TESLA V100 has a max power consumption of 250 W with 5120 cores.

The power of the ASIC design is measured by report_power command in the Power Compiler in the Synopsys Design Compiler

[3]https://images.nvidia.com/content/technologies/volta/pdf/437317-Volta-V100-DS-NV-US-WEB.pdf

set. Due to lack of access to the ASIC library resources, we won't analysis the power in the dc_shell -topo mode. In dc_shell, the report_power command reports the power after the logic synthesis but before the place and route. The power may not the same as the final power due to the following factors: Logic synthesis uses wire load models, High fanout nets are not synthesized and Clock trees do not exist in the design at the time of synthesis. However, it is still a good indicator to roughly estimate the power cost of the ASIC designs.

Table 3 shows the power and the energy efficiency, which is defined as MHash / J, for every method mentioned in section 3 and section 4.

	GHash/s	Power(W)	MHash/J
Naïve (512 HE)	0.82	12.8	64.0
Novel Counter (512 HE)	1.33	42.0	31.5
Pipeline (512 HE)	2.50	44.9	55.7
Xeon Gold 6240	0.14	165	0.83
Tesla V100	1.33	250	3.69

Table 3: The power and energy efficiency of different designs at 512 Hash Elements (HE)

The table 3 shows the ASIC designs outperform the CPU and GPU 67x and 15x respectively. And for the comparison between different ASIC designs, although the Naive design has a small margin in terms of energy efficiency, it has a about 2x-worse area efficiency (measured by throughput per mm^2) than the Pipeline design. Therefore, we consider the Pipeline design as the best, based on which a cost is estimated in the next section.

5.4 Cost Analysis

5.4.1 Cost per Wafer. Wafer costs and defect rates are hard to obtain. Moreover, the technology node we are designing with (IBM 130) is obsolete since IBM is no longer in the fab business. Alternatively, we found the price for TSMC 90 in 2020, which is $1650 per 12-inch wafer[1]. Here we are using the price for a more advanced technology node (90 nm vs 130 nm) to offset the overhead of defects.

5.4.2 Dies per Wafer. The number of dies per wafer is given by the following equation[2]

$$N = \left\lfloor \pi \left(\frac{r^2}{A} - \frac{2r}{\sqrt{2A}} \right) \right\rfloor$$

Where N is the number of dies per wafer, r is the radius of the wafer, and A is the area of the die. The area of our 512 Hash Element Pipeline design is $215mm^2$ under IBM 130 technology. Also plugging in the wafer diameter, which is 12 inches, we have 336 dies per wafer for our design.

5.4.3 Per Die Cost. The mask cost of 130 nm technology is about $200,000[3]. Here we optimistically assume that our volume will be 1 million, the overall cost per die is $4.2. This number does not take packaging or testing into account. However, there is a large enough margin compared against off-the-shelf CPUs and GPUs.

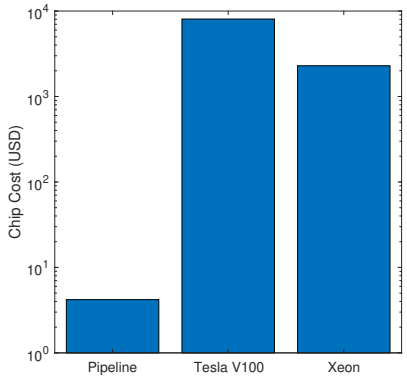

Figure 6: Per Chip Cost Comparison with CPU and GPU

5.5 ASIC Benchmarks

Figures 7 compares the frequency and area of Pipeline design and design by Dadda et al.[4]. Unfortunately, our design, despite using a more recent technology, is beaten by 3x on area and 2x on frequency. This is probably due to the extensive customization of adder modules in [4]. Our design, on the other hand, uses the "+" operator and leaves the work to the Design Compiler. We have tried integrating different adders, including Brent–Kung, Kogge-Stone, and CLA, but none of these out-performed the default "+" operator. The adders are responsible for a large fraction of the area and latency. For instance, a Brent-Kung adder occupies $0.025mm^2$ of area, while introducing about $1ns$ latency. There are 7 adders in our design, and the longest path contains 2 adders. This means that adders alone can take up nearly 42% of area, while being on the critical path. Therefore, we must look further into the optimization of adders in order to improve our overall performance.

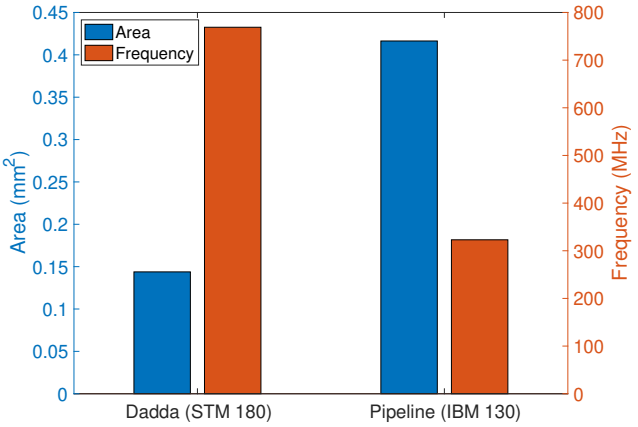

Figure 7: Area and frequency of Pipeline and Dadda

6 FUTURE WORK

6.1 Full-Custom Adder Design

Since the adders have become a bottleneck in our design, it can be beneficial to implement a full-custom adder to optimize for speed and area. Most additions may be performed by carry-save adder (CSA) trees to avoid unnecessary carry propagation, thus reducing circuit delay. Meanwhile, some form of carry-propagate adder (CPA) (e.g. ripple-carry adder (RCA) or carry-lookahead adder (CLA))[6] can be leveraged to perform the ultimate carry propagation for only once.

6.2 Peripherals

We only implemented the core functionality of Bitcoin mining. In order to make our design practical, some peripheral devices are also essential. We need an in-package DRAM to store the Bitcoin data to be hashed. Also, there should be a PCIe interface that transfers data to and from the CPU with proper bandwidth and speed.

7 RELATED WORK

There are some other works exploring the hardware design for bitcoin mining.

Taylor et al. [5] gave an overview of the Bitcoin mining hardware in the past decade, which introduced the Bitcoin price curve and various hardware design covering CPUs, GPUs, FPGAs and ASIC-Miner.

Vilim et al. [6] explored the potential for approximation to improve the profit of Bitcoin mining. The implementation of Kogge-Stone adder using functional and operational approximation has the ability to raise the Bitcoin mining profits by 30%.

Martino et al. [7] designed an SHA-256 accelerator meeting the computational demands of a block-chain application, but at the same time satisfying the constraints that are typically found in IoT environments. Gold-Strike 1 [8], Coin-Terra's first generation Bitcoin mining processor provided solutions for the high power density and cooling problems in Bitcoin ASICs, achieving the energy efficiency of 4 GHash/J on a 16nm process node.

8 CONCLUSION

In this paper we presented three ASIC implementations (Baseline, Naive Counter Based, Pipeline) of the SHA-256 algorithm compression function part. These three implementations are compared with latest CPU and GPU algorithm implementations. We show that the best ASIC design achieves 2.50 Ghash/s with energy efficiency 55.7 Mhash/J, which outperforms CPU and GPU by 17.85x and 1.87x in terms of performance; 67.01x and 15.09x in terms of energy efficiency.

9 WORK CONTRIBUTION

Member	Work
Yiqiu Sun	GPU, Baseline implementation
Yufeng Gu	Pipeline version implementation
Haichao Yang	CPU, Verification and validation
Wentao Zhang	Novel Counter version design and implementation

REFERENCES

[1] Anton Shilov, "TSMC's Estimated Wafer Prices Revealed," Web., Sept. 2020, Available: https://www.tomshardware.com/news/tsmcs-wafer-prices-revealed-300mm-wafer-at-5nm-is-nearly-dollar17000, Accessed: Apr. 19th, 2021.

[2] Neil Weste and David Harris, *CMOS VLSI Design: A Circuits and Systems Perspective. 4e.* Addison-Wesley, Boston, USA, 2010. p. 650.

[3] AnySilicon, "Semiconductor Wafer Mask Costs," Web. Available: https://anysilicon.com/semiconductor-wafer-mask-costs/, Accessed: Apr. 19th, 2021.

[4] L. Dadda, M. Macchetti and J. Owen, "The design of a high speed ASIC unit for the hash function SHA-256 (384, 512)," Proceedings Design, Automation and Test in Europe Conference and Exhibition, 2004, pp. 70-75 Vol.3, doi: 10.1109/DATE.2004.1269207.

[5] Taylor, Michael Bedford. "The evolution of bitcoin hardware." Computer 50.9 (2017): 58-66.

[6] Vilim, Matthew, Henry Duwe, and Rakesh Kumar. "Approximate bitcoin mining." 2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC). IEEE, 2016.

[7] Martino, Raffaele, and Alessandro Cilardo. "Designing a SHA-256 processor for blockchain-based IoT applications." Internet of Things 11 (2020): 100254.

[8] Barkatullah, Javed, and Timo Hanke. "Goldstrike 1: Cointerra's first-generation cryptocurrency mining processor for bitcoin." IEEE micro 35.2 (2015): 68-76.

[9] M. Macchetti and L. Dadda. -pipelined hash circuits". In: 17th IEEE Symposium on Computer Arithmetic (ARITH'05). 2005, pp. 222-229. doi: 10.1109/ARITH.2005.36

[10] M. Bedford Taylor, "Bitcoin and the age of Bespoke Silicon," 2013 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES), 2013, pp. 1-10, doi: 10.1109/CASES.2013.6662520.

[11] David B. Kirk and Wen-mei W. Hwu. 2016. Programming Massively Parallel Processors, Third Edition: A Hands-on Approach (3rd. ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.