Linzhi Semiconductors

Linzhi Working Papers
No 10

Pseudo-Random Logic ASIC Design.

by Sonia Chen

March 2019

Keywords: LWP10, ProgPoW, ASIC Design, Estimates

Linzhi ASICs
Mar 29 · 4 min read

EIP 1057 (ProgPoW): Open Chip Design for 1% cost/power increase
仅带来1%能耗比提升的开源芯片设计

Adding a pseudo-random program to the PoW algorithm is a key technical idea described in EIP 1057 (ProgPoW). The theory goes that the compute area of a GPU is underutilized compared to memory bandwidth. ProgPoW is then designed such that it “saturates both compute and memory bandwidth at once”.

https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1057.md

> *Since the GPU is almost fully utilized, there’s little opportunity for specialized ASICs to gain efficiency.*
> *There should be little opportunity for efficiency gains compared to a commodity GPU.*
> *While a custom ASIC is still possible, the efficiency gains available are minimal.*
> *These would result in minimal, roughly 1.1x-1.2x, efficiency gains.*

Bitmain and Innosilicon both currently have Ethash systems, a good starting point for ProgPoW. What would they need to add for the new EIP 1057 compute logic?

. . .

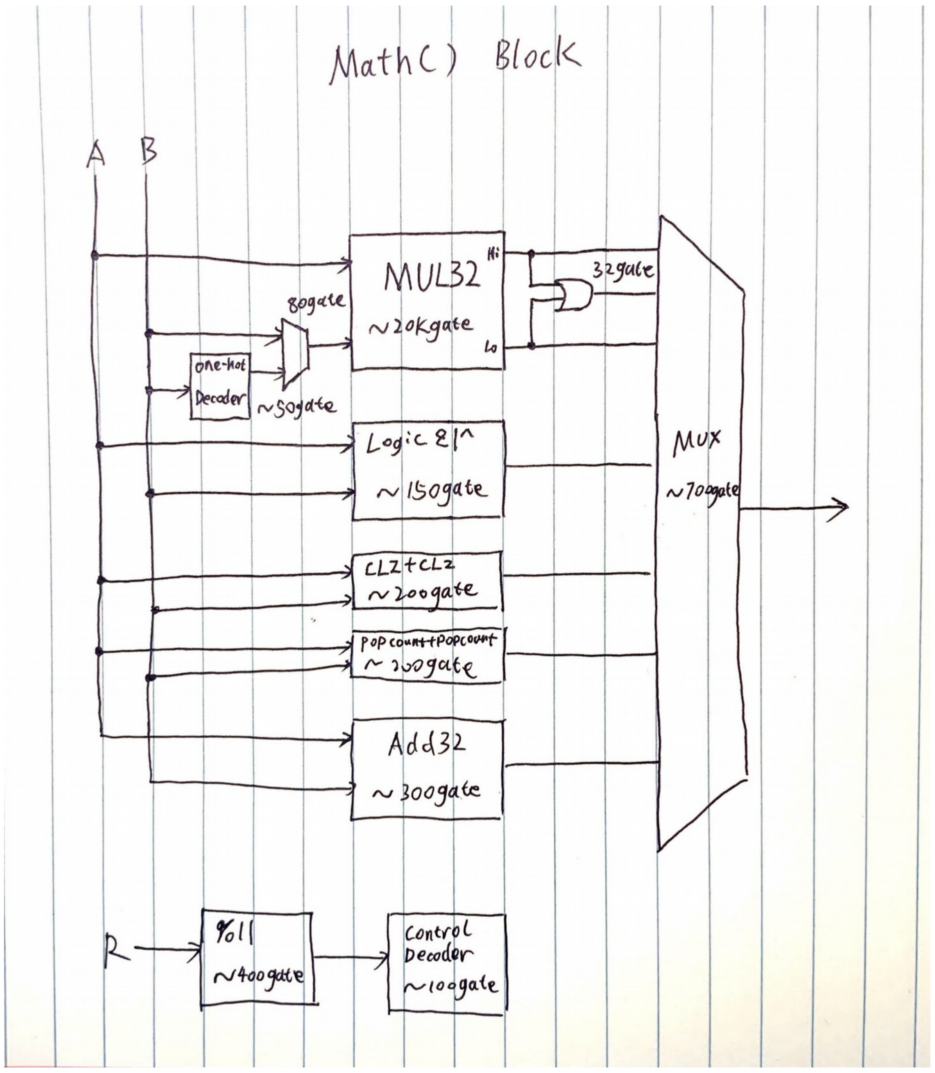
Math Block

First let’s look at Math() which includes 11 different instructions

```
// Random math between two input values
uint32_t Math(uint32_t a, uint32_t b, uint32_t r)
{
  switch (r % 11)
  {
    case 0: return a + b;
    case 1: return a * b;
    case 2: return mul_hi(a, b);
```

```
        case 3: return min(a, b);
        case 4: return ROTL32(a, b);
        case 5: return ROTR32(a, b);
        case 6: return a & b;
        case 7: return a | b;
        case 8: return a ^ b;
        case 9: return clz(a) + clz(b);
        case 10: return popcount(a) + popcount(b);
    }
}
```

- modulo 11 operation, this is quite small logic, ~400gate, 1ck latency, can be a parallel process during mix read so we can hide this latency.
- 32-bit Add, simple logic, ~300gate for a fast one.
- 32-bit Multiplier, mature IP, ~20Kgate for a fast one, since multiplier only have ~4/11 activity rate, we can use a two cycle multiplier to half the area, small possibility to increase delay.
- Rotation operation can easily map to a multiplier, for example I want to calculate ROTL(0x12345678, 8), I can do 0x12345678 * 0x00000100 = 0x0000001234567800, then we just need to OR higher word and lower word together to get 0x34567812. so just cost ~160gate extra logic
- logic operation, A&B only cost 32 gate, A|B 32 gate, A^B 96 gate, it looks like three different instructions but actually extremely small on silicon (<30um^2)
- clz and popcount are also very small
- We only need a multiplexer to select output.
- Total size of Math() is about 0.0015mm^2 on a TSMC16ULP process.
- Merge() is similar but even smaller, only shifter, adder, and tiny logic (no multipliers because constant multiply can be mapped into adder).
- Size of Merge() is roughly ~0.0005mm^2.

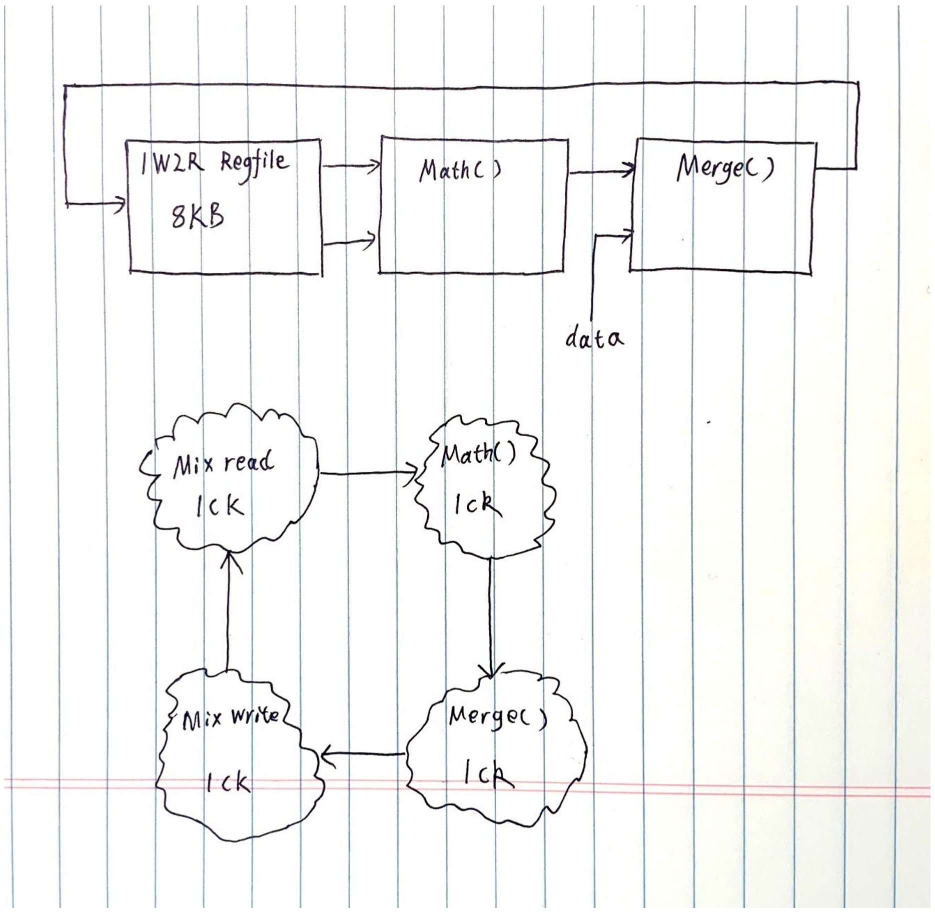


. . .

Pipeline

Now we can build our pipeline, 4-stage:

CK1: Mix regfile read, two operators in parallel (calculate %11 at same time)
CK2: Math()
CK3: Merge()
CK4: Mix write back to regfile



Task latency is 4 cycles, but we can have 4 independent threads so we can make this pipeline fully loaded.

Mix Regfile we use a 1W2R (1 write port, 2 read port) Regfile, mature IP, 8KB (for 4 threads), single cycle read/write, 1GHz operation.

If you want load/unload mix data on the fly without disturbing the pipeline, we can upgrade to a 12KB 2W3R Regfile. We then have extra read/write ports for load/unload, and 12KB is enough for 6 tasks (4 running, 1 loading, 1 unloading).

An ASIC can implement 10K sets of that block:

- running at 1GHz frequency
- 0.55V voltage (typical voltage for TSMC16ULP)
- generating ~10T Math() + Merge() throughput per second
- Power estimate roughly 3mW each pipeline, 30W in total.
- No customized circuit/layout

- All standard cells, auto placement route
- Use mature IP only
- No aggressive overclocking
- No aggressive under voltage
- ~8.4K Merge() and ~4.8K Math() per hash
- pipeline includes 1 Merge() and 1 Math()
- pipeline is only the logic part, not the memory part
- the xGB memory are not included
- we have 10T throughput on single chip asic, divided by 8.4K Merge() per hash, means 1.2GHash

. . .

Performance

The design (logic part only) can provide 1.2 GHash ProgPoW performance at 30W power.
Nvidia GTX1070Ti can provide 15.7 MHash at 115W, but including memory.

. . .

Summary

The random instruction of EIP 1057 increases die cost/power by about 1%, and causes a die increase of <1mm². The proposed open design is demonstrating a logic-only performance of 1.2 GHash at 30W and could be deployed by Bitmain or Innosilicon, resulting in a machine with about half the hashrate of their predecessors, similar to the best GPUs.

If you are an independent chip designer and want to take the ProgPoW design idea further, please get in touch. We hereby release the design into the public domain. Hopefully the open design process can also inspire some developers to learn more about the world of chip design.

Linzhi Team, Shenzhen
Telegram: https://t.me/LinzhiCorp
email: sonia@linzhi.io

. . .

Thanks

Peter Salanki provided the initial idea for this writeup and encouraged us to do it. Alexey Akhunov always encourages everyone to think for themselves.

. . .

References

https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1057.md
https://medium.com/@ifdefelse/understanding-progpow-performance-and-tuning-d72713898db3
https://medium.com/@infantry1337/comprehensive-progpow-benchmark-715126798476
https://etcsummit.com/2018-etc-summit/
Launch of Linzhi Ethash chip at ETC Summit:
https://www.youtube.com/watch?v=LMofyroBfio
Linzhi Website: https://linzhi.io/