KECCAK-BASED BITCOIN MINER DESIGN PROJECT B

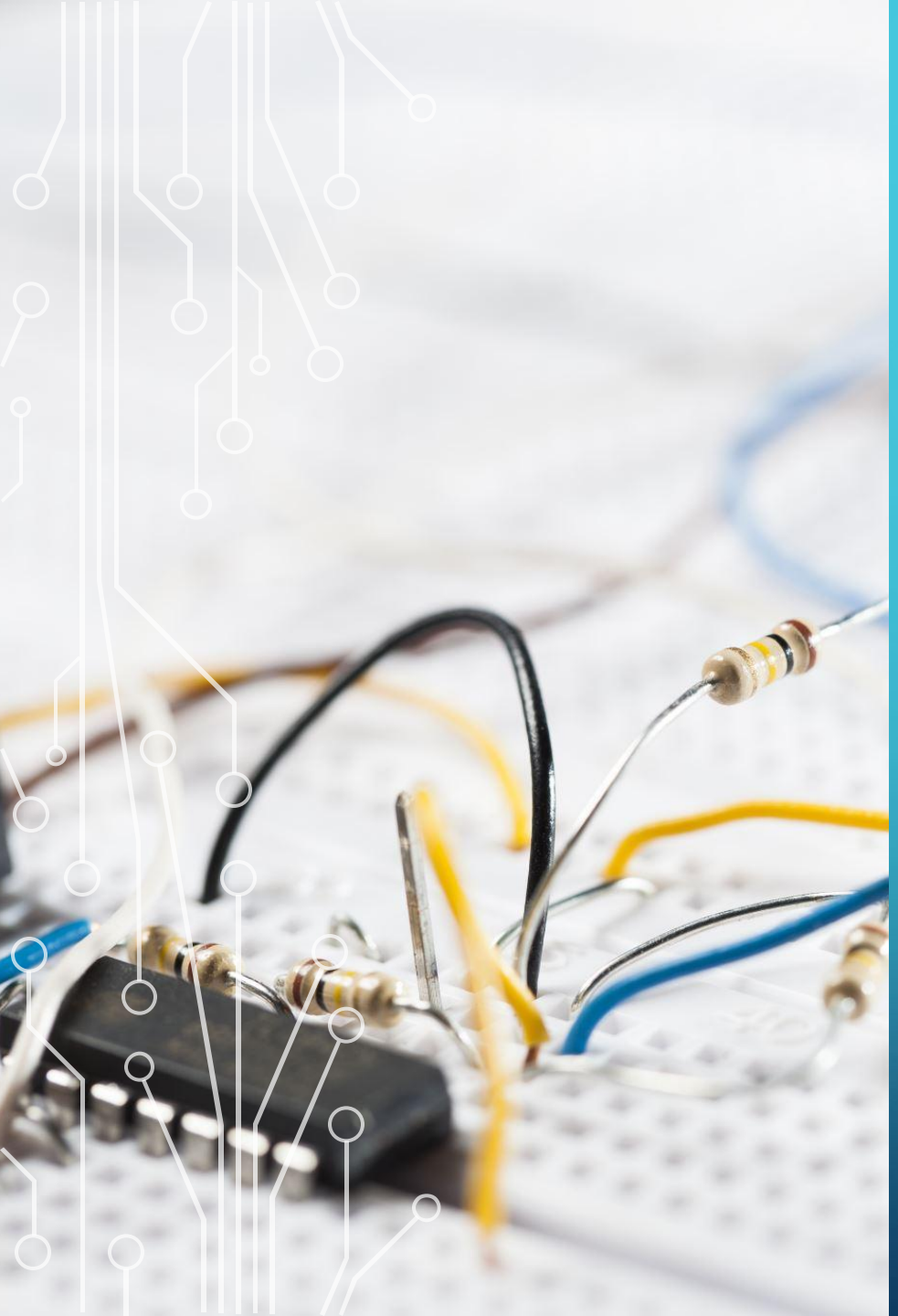
BY Desmond, Jack, Felix and Thenuja

PROBLEM SPECIFICATION

- What is Bitcoin Mining?
- Challenges of SHA3-256 in Hardware
- Why use Keccak (SHA3-256)

WHAT IS BITCOIN MINING?

- The process of finding a numeric *nonce* that, when combined with a block's header data and hashed, produces a hash value below a target threshold
- Miners iterate over nonces, hashing the 80-byte block header
  - checking if the resulting hash is sufficiently small
- This ensures that finding a valid block is *computationally hard*



CHALLENGES OF SHA3-256 IN HARDWARE

- The SHA3-256 algorithm involves 64 rounds per hash
  - Bitcoin uses a double hash, thus 128 rounds total
- Fully unrolling and pipelining all 64 rounds can yield one hash result per clock cycle
  - Doing this for double-hash doubles resource usage
- Hardware Bitcoin miners often split work:
  - Pre-compute the first SHA-256 of the static part of the block header
  - Repeatedly compute the second SHA-256 for each nonce in hardware

WHY USE KECCAK (SHA3-256)?

- Keccak (the core of SHA-3) is explored here as an alternative hash function for mining.
- Keccak was designed for efficiency in hardware:
  - *"Thanks to its symmetry and chosen operations, [Keccak's] design is well-suited for ultra-fast hardware implementations and the exploitation of pipelining".*
- Can achieve at least **4×** the performance of comparable SHA-2 implementations at roughly similar power consumption
- This Project is designed to investigate FPGA acceleration of the Bitcoin mining algorithm

$$Keccak[r,c,d](M)\{$$

Initialization and padding:

$$S[x,y] = 0, \qquad \forall (x,y)\ in\ (0 \dots 4, 0 \dots 4)$$

$$P = M \parallel 0x01 \parallel byte(d) \parallel byte\left(\frac{r}{8}\right) \parallel 0x01 \parallel 0x00 \parallel \cdots \parallel 0x00$$

Absorbing phase:

$\forall$ *block* P_i *in* P

$$S[x,y] = S[x,y] \oplus P[x+5y], \qquad \forall (x,y)\ such\ that\ (x+5y) < \left(\frac{r}{w}\right)$$

$$S = Keccak - f[r+c](S)$$

Squeezing phase:

$$Z = empty\ string$$

while output is requested

$$Z = Z \parallel S[x,y], \qquad \forall (x,y)\ such\ that\ (x+5y) < \left(\frac{r}{w}\right)$$

$$S = Keccak - f[r+c](S)$$

return Z

$\}$

```
Input: message M
1. Pad M with multi-rate padding (delimit M with 0x06 and 0x80)
2. Break padded M into blocks of r bits
3. Initialize 1600-bit state to zero
4. For each block:
     XOR block into the first r bits of state
     Apply Keccak-f[1600] permutation
5. Output first 256 bits of state (truncate or continue permutation for longer output)
```

ALGORITHM TO BE INVESTIGATED (BITCOIN MINING WITH KECCAK)

Designated as SHA3 by NIST in 2015, Keccak is an secure hash algorithm known for its radically different algorithmic approach to previous hash algorithms (SHA-1, SHA-2).

It is invulnerable to many attacks which could compromise previous algorithms and allows arbitrary-length input.

It is used by Ethereum as its new core algorithm.

PROJECT AIMS

- Functional Hardware Miner
- Accelerated Hash Computation
- FPGA Capability Showcase?
- Performance Comparison & Analysis

THE FLOW OF MINING

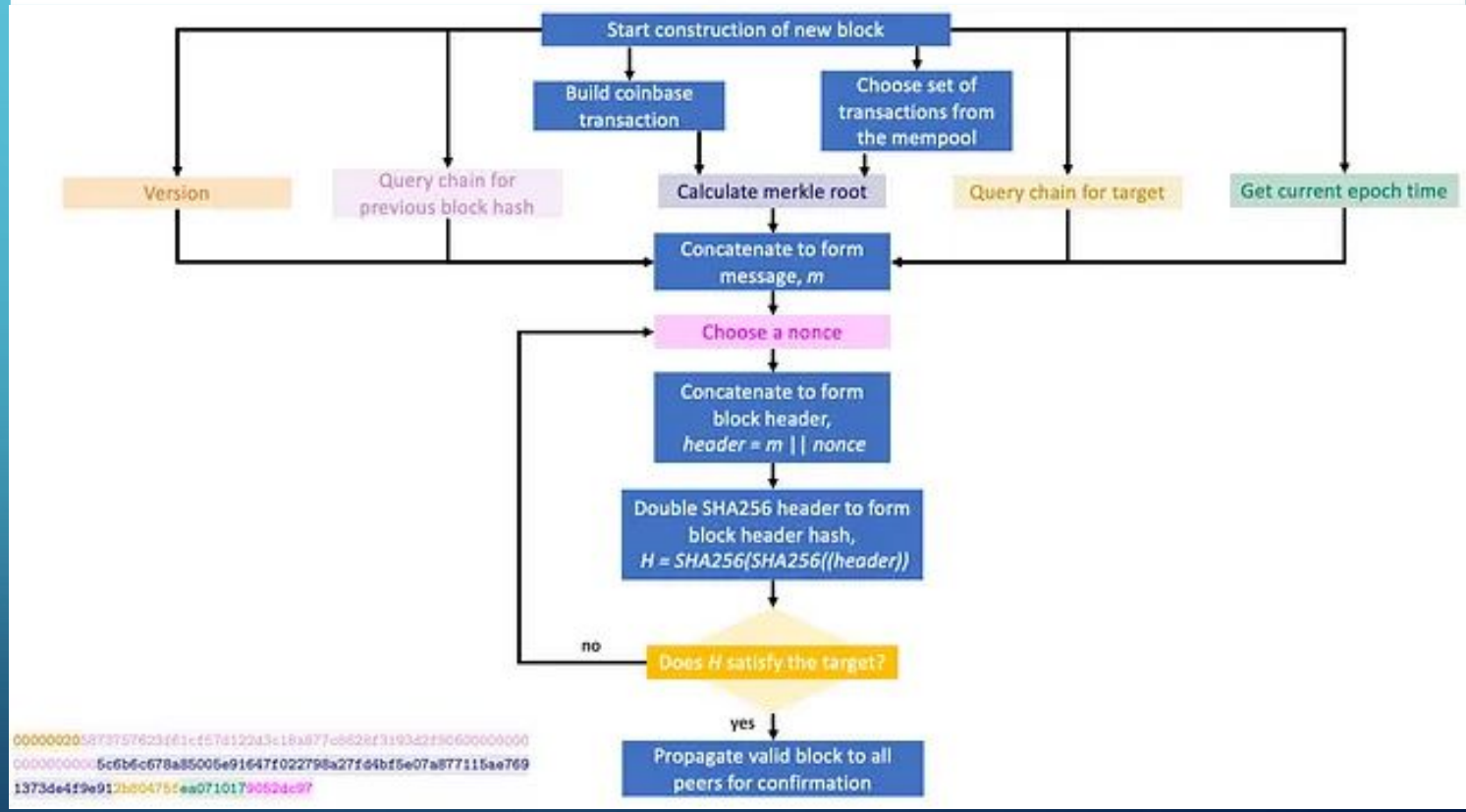
The process of mining in a blockchain network is essentially a competition of hashing.

For the sake of demonstration, we will ignore the computations of

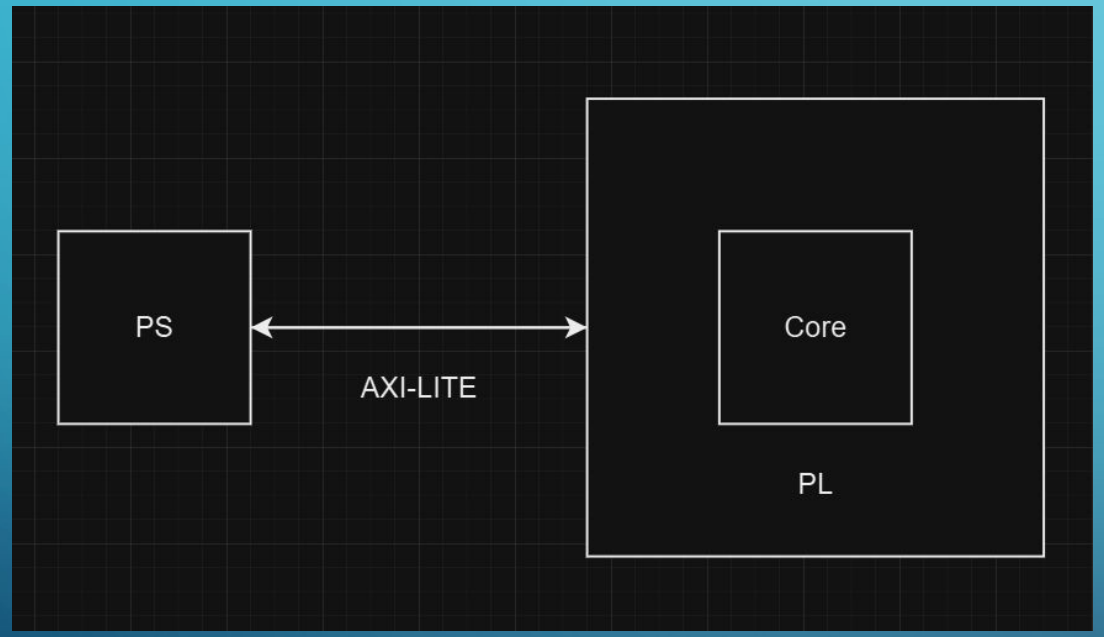
- Merkel roots
- Queries of previous block hash
- Version sync

We are interested in accelerating the Keccak algorithm as the core hash over a fixed header.

Name	Type	Bytes	Description
version	int32_t	4	Version number
previous hash	char[32]	32	Hash of the previous block header in internal byte order
merkle root	char[32]	32	Merkle root of the transactions included in the block formatted in internal byte order
time	uint32_t	4	Epoch timestamp of the block
bits	uint32_t	4	Encodes the network target difficulty
nonce	uint32_t	4	Dedicated number to be updated to generate unique hashes



FUNCTIONAL HARDWARE MINER

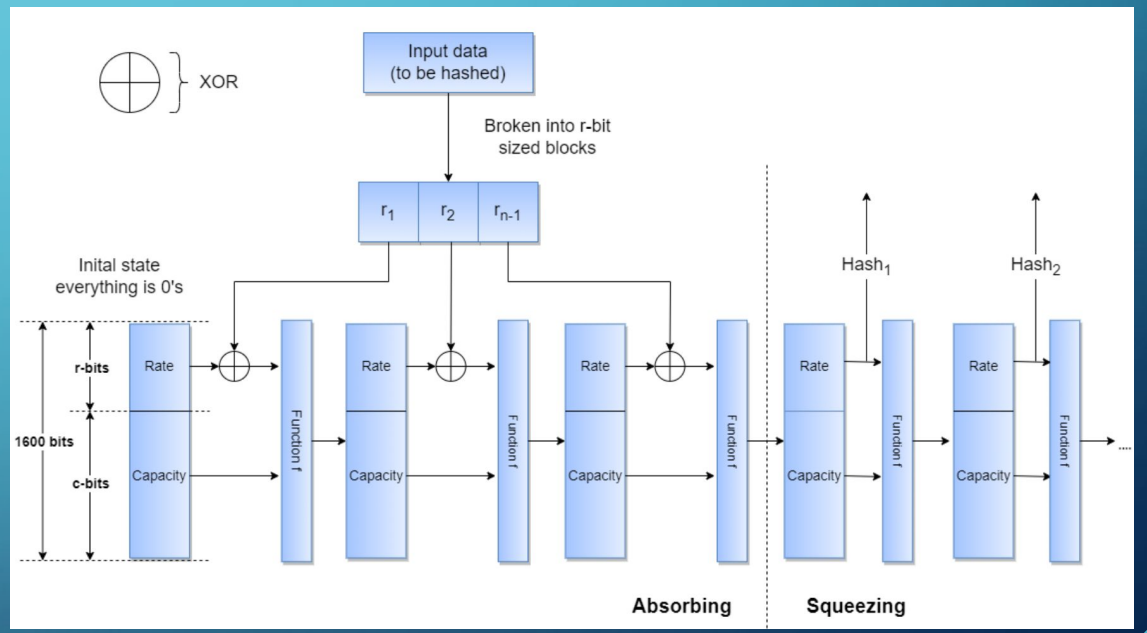


- This project uses a simple AXI-Lite interface for communication between PS and PL
- The PS contains a predefined target value to evaluate the mining result
- A nonce from the block is sent to PL via AXI-Lite
- The core processes the nonce, and the result is sent back to the PS
- The PS then compares the result with the target:
  - If the result >= target, then PS sends it back for the next iteration
  - If the result < target, then valid nonce is found

ACCELERATED HASH COMPUTATIONS

Keccak Function Overview:

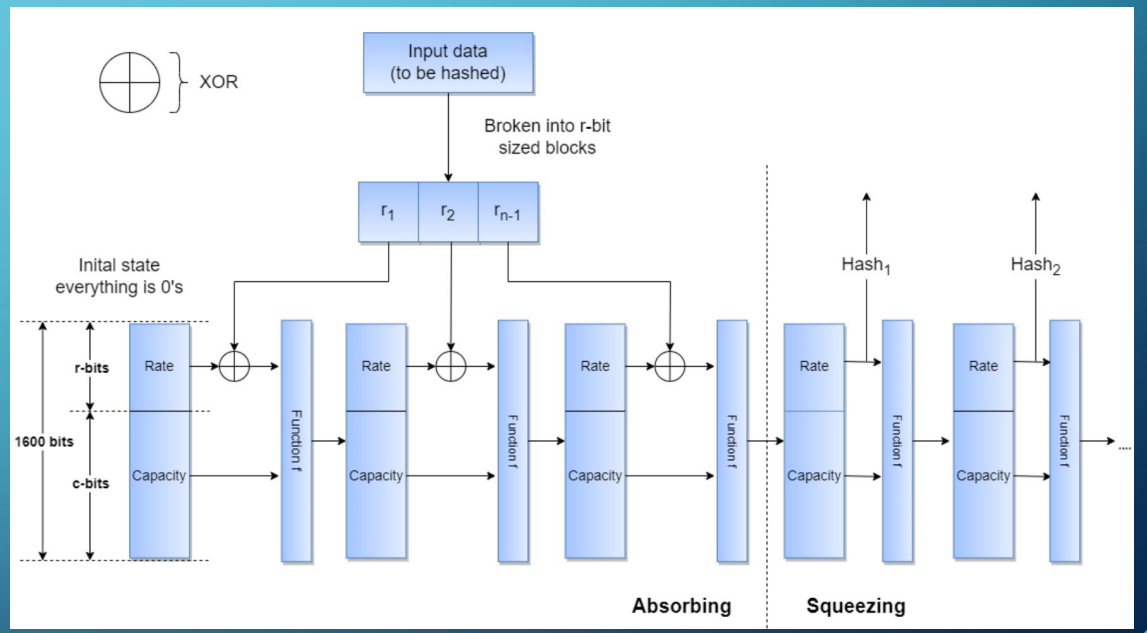
- Initial State:
  - A 1600-bit internal state initialized to all zeros, split into
    - r bits (Rate): interfaces with input and output
    - c bits (Capacity): Provides cryptographic strength



ACCELERATED HASH COMPUTATIONS

Absorbing Stage

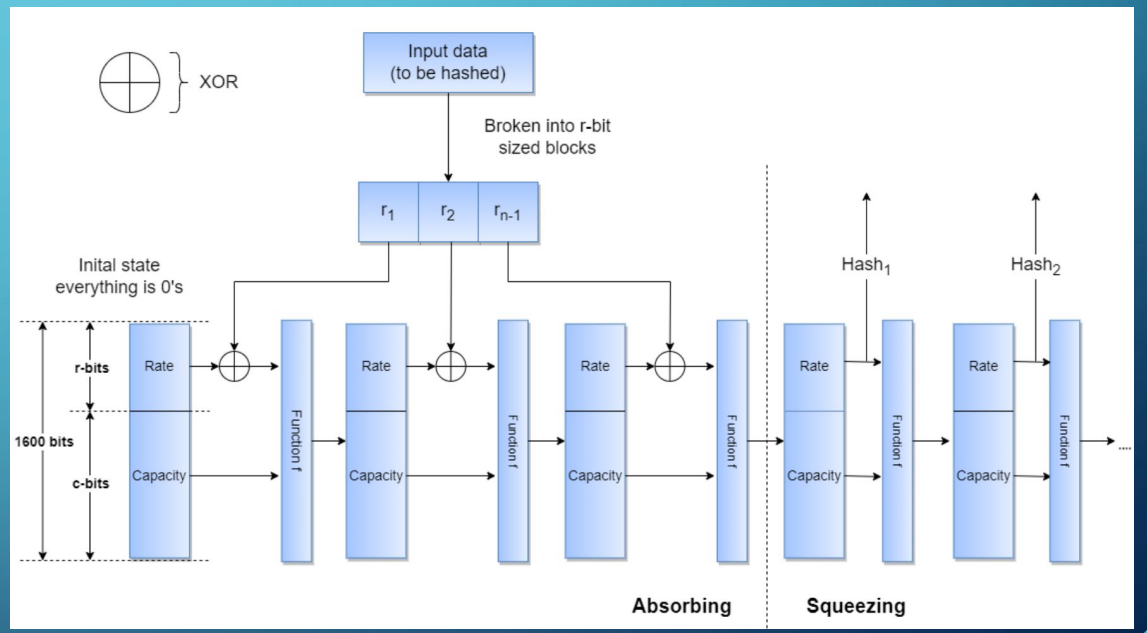
- Input data is divided into r-bit blocks
- Each block is processed in XOR operation into the rate portion of the state
- The entire state is updated via the Keccak permutation function f
- Repeated until all input blocks are absorbed



ACCELERATED HASH COMPUTATIONS

Squeezing Stage

- Output is extracted from the rate portion as Hash1, Hash2…
- If more output is needed, apply f again and extract more bits
- Continues until desired hash length is reached



Keccak-f

24 rounds of 5 steps:

- Theta: mix bits within columns
- Rho: rotate bits within lanes
- Pi: rearrange lanes spatially
- Chi: non-linear substitution
- Iota: XOR round constant

```
# Theta
```py
C[x] = A[x][0] ⊕ A[x][1] ⊕ A[x][2] ⊕ A[x][3] ⊕ A[x][4]
D[x] = C[x-1] ⊕ ROT(C[x+1], 1)
A[x][y] ← A[x][y] ⊕ D[x]
```

Rho
```py
A[x][y] ← ROT(A[x][y], r[x][y])
```

Pi
```py
A'[x][y] = A[(x + 3y) mod 5][x]
```

Chi
```py
A[x][y] ← A[x][y] ⊕ ((¬A[x+1][y]) ∧ A[x+2][y])
```

Iota
```py
A[0][0] ← A[0][0] ⊕ RC[round_index]
```
```

```
-- pi
i3001: for y in 0 to 4 generate
    i3002: for x in 0 to 4 generate
        i3003: for i in 0 to 63 generate
            --pi_out(y)(x)(i) <= pi_in((y +2*x) mod 5)(((4*y)+x) mod 5)(i);
            pi_out((2*x+3*y) mod 5)(0*x+1*y)(i) <= pi_in(y) (x)(i);
        end generate;
    end generate;
end generate;
```

PERFORMANCE COMPARISON & ANALYSIS

- **Bandwidth of the write:** sustained input with randomized file.
- **Latency of the algorithm:**
  - Theoretical performance comparison with synthesis / implementation reports (against other hardware-based implementation);
  - Realtime performance comparison with burst writes
- **Efficiency:** task-time power monitoring on both PS and PL
  - Combined with bandwidth, obtain H/s performance comparables

Hardware Optimization Strategies

Pipelining & Unrolling

- Implement a **two-stage pipelined architecture:** Theta (θ) step — high delay path, Rho, Pi, Chi, Iota — lower delay logic
- This division reduces the **critical path,**improving max clock frequency and trying to achieve an II = 1.
- Additional **inter-round pipeline registers** allow higher throughput, allowing to produce multiple hashes at once.
- Partial **loop unrolling** (e.g. factor of 2) using VHDL `generate` enables parallel round execution while balancing area usage
- These techniques follow the optimization approach from **Sideris et al. (2023)**

Bitwise Logic Parallelism

- Keccak's XOR, AND, and rotation operations map efficiently to FPGA logic
- Minimal interdependencies between rounds allow concurrent evaluation
- Optimized datapaths reduce synthesis and timing pressure

Static Input & RC Optimization

- Fixed 80-byte header simplifies control and eliminates runtime padding
- **Simplified 7-bit Round Constant (RC)** replaces full 64-bit table
  → Reduces iota step logic from 64 XORs to just 7 XORs

AXI-Lite Control Integration

- Lightweight **memory-mapped interface** connects PL (hash core) to PS
- Enables runtime control: trigger hashing, retrieve results
- The nonce will be handle inside the VHDL core to improve hashing efficiency.

PROJECT PLAN & TIMELINE

Week 7 - Research	Week 8 - VHDL Core	Week 9 - System Integration	Week 10 - Testing & Presentation
Review Bitcoin mining & Keccak-256 algorithm. Define scope, VHDL flow, assign roles.	Refine Keccak-256 miner core in VHDL for PL. Set up simulation and testbench.	Wrap core with AXI Interface (Lite), add PS logic, test hashing rounds.	Simulate full flow and present final results.

RISKS & CONTINGENCIES

Timing Closure Issues

- *Risk:* Pipelined Keccak core may not meet timing at target frequency
- *Contingency:* Reduce pipeline depth or unroll factor; adjust placement constraints

Excessive Resource Usage

- *Risk:* Full unrolling or deep pipelining may exceed available slices/LUTs
- *Contingency:* Scale back unrolling; optimize datapath and control logic

AXI-Lite Integration Bugs

- *Risk:* Communication failures between PS and PL (e.g., unresponsive core)
- *Contingency:* Use Vivado ILA to debug; validate AXI signals before full integration

Incorrect Hash Output

- *Risk:* Hardware hash doesn’t match Keccak reference output
- *Contingency:* Compare against NIST vectors; test round-by-round in simulation

ROLES OF TEAM MEMBERS

Felix : Working on the Keccak core design, optimizing and documentation.

Desmond : Focusing on debugging the VHDL core for hashing and simulation testing.

Jack : Working on system integration, PS-PL coordination and hardware testing.

Thenuja : Contributing to optimization strategy, PS–PL coordination and documentation.

THANK YOU!

Q & A