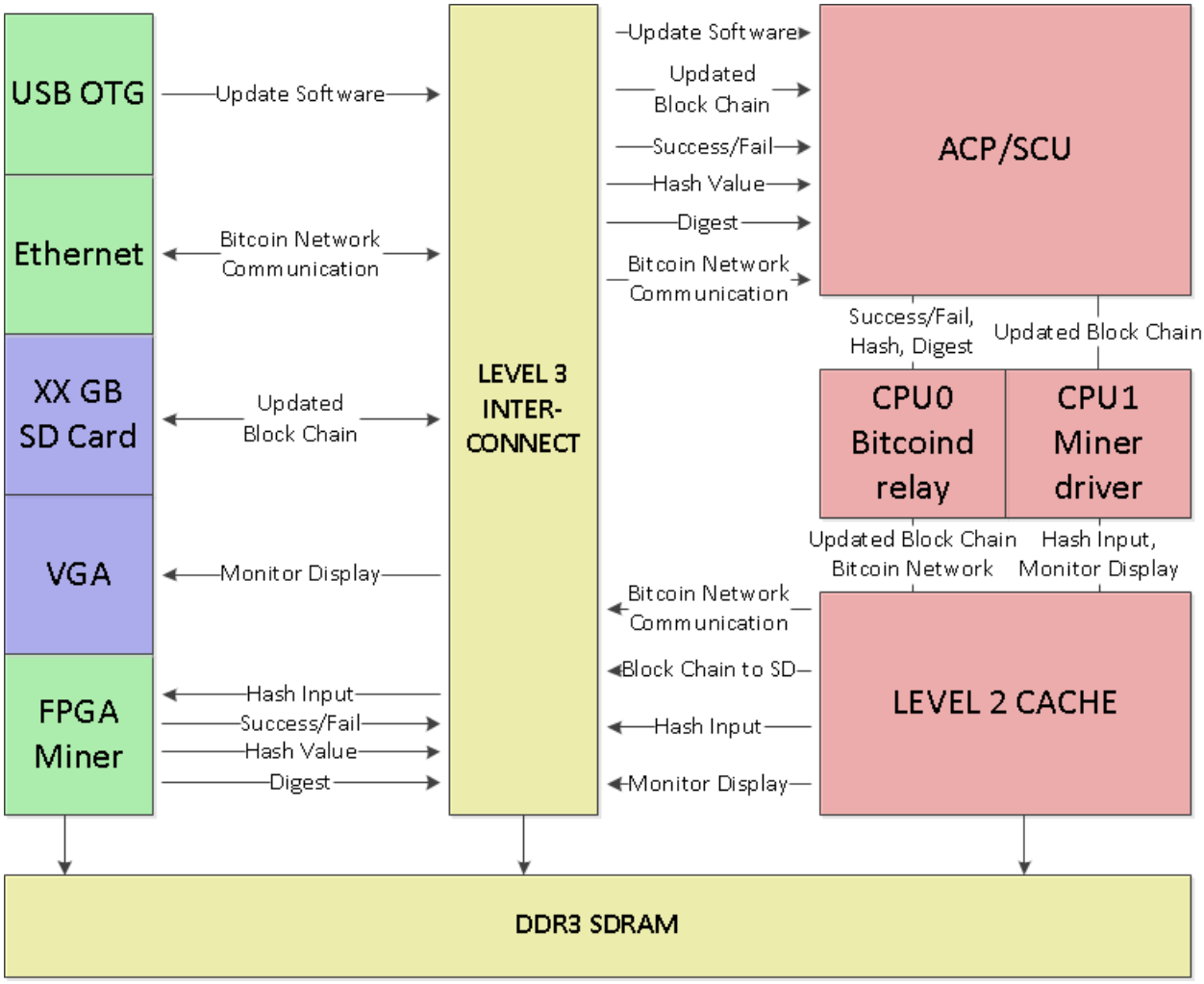
Half Fast Bitcoin Miner Design

Peter Xu (px2117)
Patrick Taylor (pat2138)
Ben Nappier (ben2113)
Matheus Candido (mcc2224)

Project Overview:

Our “Half Fast” Bitcoin miner design utilizes the full potential of the SoCKit board’s FPGA fabric and ARM-based high processor system (HPS) to mine Bitcoins at a rate exceeding those of CPU/GPU-based systems and approaching that of ASIC systems. Bitcoin mining relies heavily on the cryptographic principles of hash algorithms; we first use the non-graphical Linux program bitcoind to retrieve an 80-byte block header, consisting of a 4-byte version number, a 32-byte previous block header, a 32-byte Merkle Root of previous transactions, a 4-byte timestamp, a 4-byte difficulty, and a 4-byte nonce initialized to 0. Once this header is parsed from the Bitcoin network block retrieval process, the nonce-incremented header is then through a double SHA-256 hash, and the hashed result is compared to a 32-byte difficulty specified in the polled block to “solve” the block. Given our board’s ability to run the CentOS distribution of Linux, we are able to poll the Bitcoin network for blocks and mine the given blocks on the same piece of hardware using the FPGA fabric, as well as run multiple iterations of the miner in parallel to increase the rate at which successful blocks are determined. A block contains a history of unverified bitcoin transactions. When a block is solved the transactions are verified. The first transaction of every block, the coinbase, gives a set number of bitcoins to the miner who has solved the block. With our flexible design and streamlined hardware-software implementation, we are able to adapt to changes in Bitcoin software and operate at a speed comparable to existing miners.

High-Level Block Diagram



Communication protocol between network software to SHA hardware connections

The interface between software and hardware will primarily send work retrieved from the network to the SHA peripherals. If a SHA peripheral solves its work it will announce this to the software and the software will to retrieve additional work for the all the peripherals from the network. If the SHA peripherals do not solve their work within a given time, they will be updated with additional work. The software will parse and prepare the data to be hashed so that the only operations the hardware is concerned about is hashing and verifying the difficulty of the hashed data.

Since the hardware only needs to run the hashing algorithm on the half of the block header including the nonce value, the primary registers for communication from software to hardware will be 64 bytes. Each SHA Peripheral will have its own register for the initial data, and will also be sent an interrupt signal when its register is fed a new data block to work on. Once the peripheral receives work it will work until it solves its data or it will be interrupted by additional work. From the hardware to the software, there will be a signal representing the address of a SHA peripheral register's sent to the software if the peripheral solved its work, and the peripheral

will update a single 64 byte register reserved for solved work. The software will be listening for a peripheral to solve its work, and will then communicate with the network the solved work if this occurs.

Miner Peripheral

The main function of the miner is to run the SHA256 algorithm on the 80 byte value of the Block header once and then hash that result again. Each time it will increment a section of the header called the nonce. Each iteration of the nonce, it will then compare the new resulting 256-bit hash (or digest) with the difficulty specified in the Block header. The difficulty is a minimum value the resulting hash must be set by the bitcoin network. This difficulty can be the thought of as the number of trailing zeros. If it is within the range of the difficulty, it will signal the driver which will then alert the Bitcoin network or mining pool to which it is connected that it has solved the Block. It will pad the value that is to be hashed such that it is a multiple of 512 bits. Because the nonce is the only value altered of the header upon each iteration, the first 64 bytes of the header will only be hashed once, while the remaining portion containing the nonce will need to be padded with zeros, because it only contains 16 bytes, and hashed each iteration. The first 64, non-nonce related, bytes of the header is known as the midstate.

Required Functions

- Right Shift by n bits
- Right Rotate by n bits
- Mod 2^{32} adders
- Right/Left Padders/Appenders
- Concatenator of 8 32-bit words to obtain final 256-bit digest

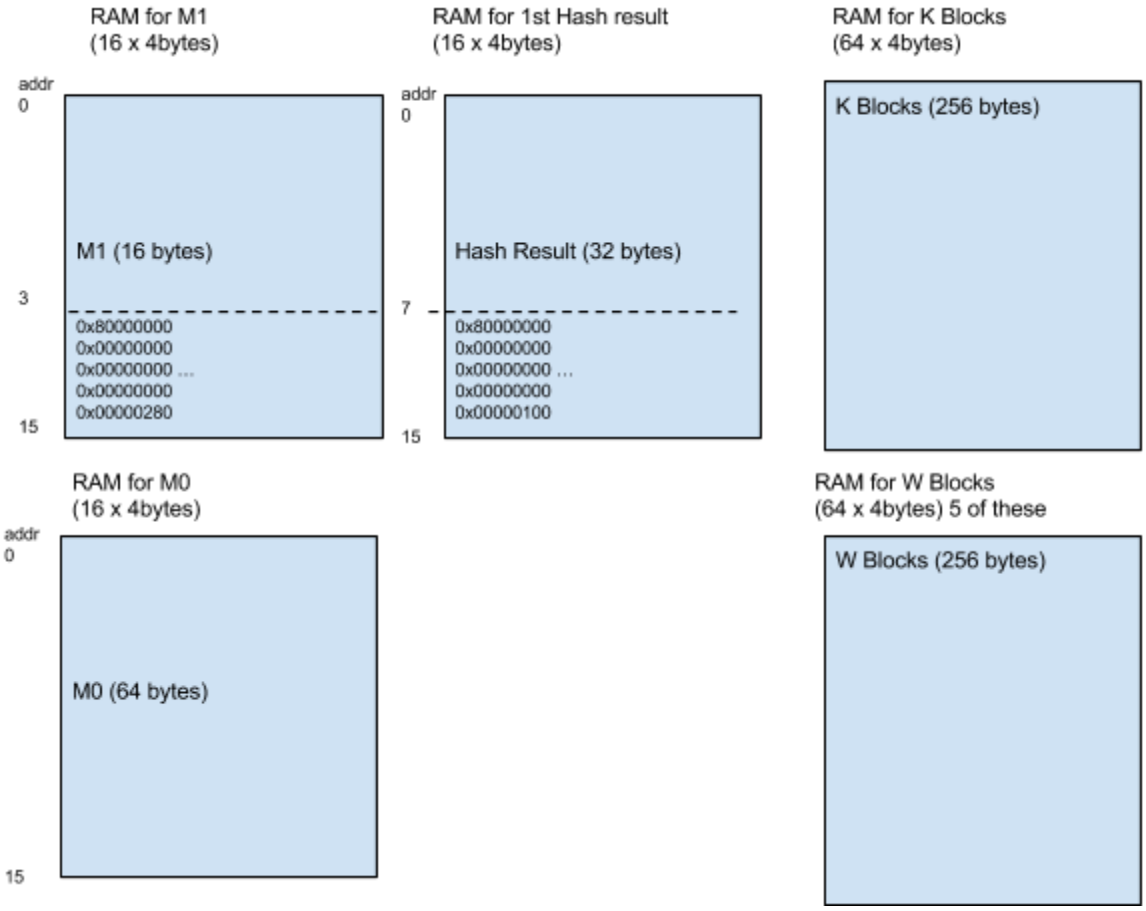
Memory Requirements

- (32 bytes, 4 bytes each) h0 - h7. These are registers that hold the intermediate hash values
- (64 bytes, 4 bytes each) a - h, tmp1, tmp2, S0, S1, s0, s1, maj, ch. These registers are temporary registers that hold the intermediate hash values when the compression function is executed in the algorithm
- (256 bytes, 4 bytes each) K[0..63]. A sequence of constant words used in SHA256
- (256 bytes, 4 bytes each) W[0..63]. Expanded message blocks that are computed.
- (80 bytes) Block header info. Data about the block to be processed that is retrieved from the bitcoin network or from the mining pool. This will be split into two register sets, M0 (64 bytes) and M1 (16 bytes). The software will write the block header information directly to these registers.
- (32 bytes) Hash/Digest. The result of the SHA256 algorithm must be stored so that the driver can access it and send it back to the bitcoin network for checking

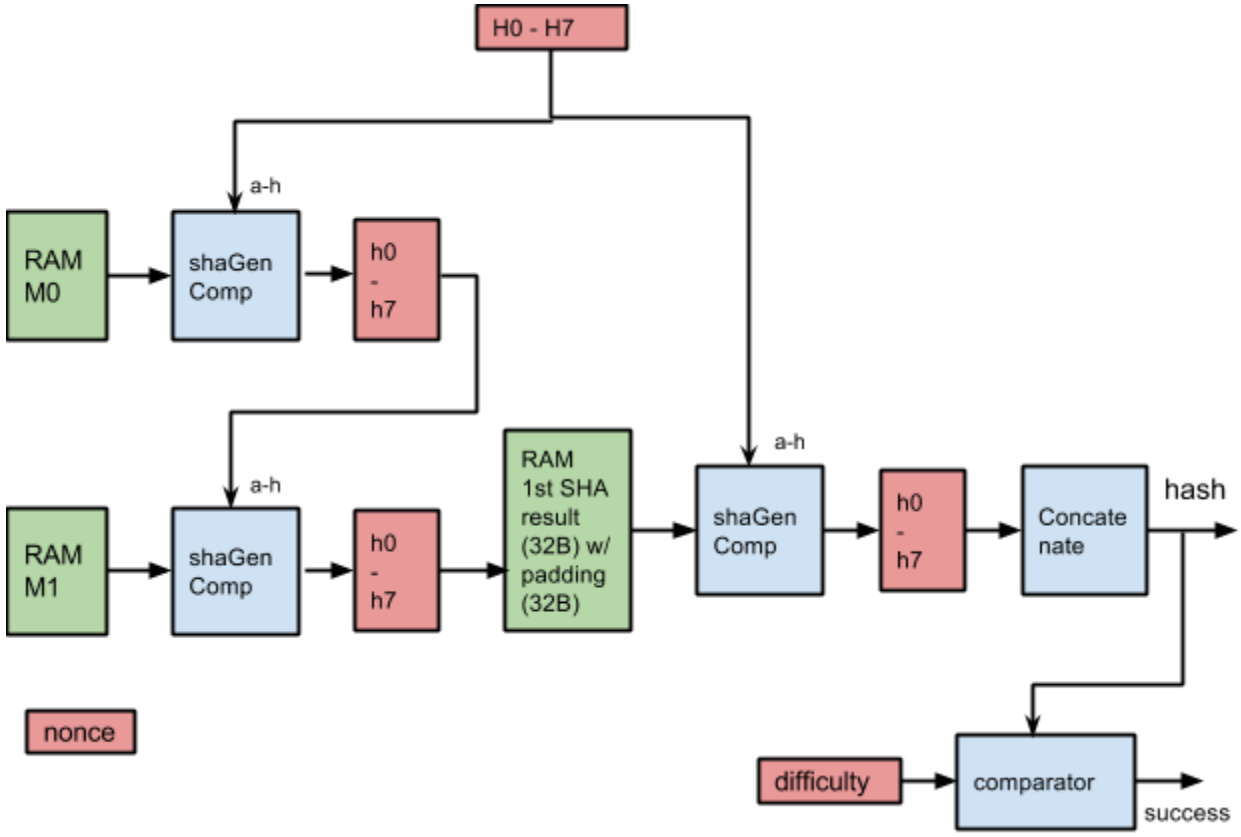
We will implement the peripheral in 3 modules: WBlocks, Compressor, shaGenComp. Their interfaces are defined below.

```
module WBlocks (
      input logic     clk, reset,
      input logic[31:0] Mi,
      output logic [31:0] Wi,
      output logic [7:0] address);

module Compressor (
      input logic clk, reset,
      input logic [31:0] a,b,c,d,e,f,g,h,
      input logic [31:0] W,
      input logic [31:0] K,
      output logic [31:0] W_out,
      output logic [31:0] a_out, b_out, c_out, … , h_out,
      output logic done);

module shaGenComp (
      input logic     clk, reset,
      input logic [31:0] hash_in,
      input logic [31:0] a,b,c,d,e,f,g,h,
      output logic [31:0] digest_data);
```

Memory Map


RAM for M1
(16 x 4bytes)
addr
0
M1 (16 bytes)
3
0x80000000
0x00000000
0x00000000 ...
0x00000000
0x00000280
15
RAM for 1st Hash result
(16 x 4bytes)
addr
0
Hash Result (32 bytes)
7
0x80000000
0x00000000
0x00000000 ...
0x00000000
0x00000100
15
RAM for K Blocks
(64 x 4bytes)
K Blocks (256 bytes)
RAM for M0
(16 x 4bytes)
addr
0
M0 (64 bytes)
15
RAM for W Blocks
(64 x 4bytes) 5 of these
W Blocks (256 bytes)

Basic Block Diagram of Miner Peripheral



Red - registers
Green - RAM
Blue - Computation blocks

Milestones

Milestone 1:

- Connecting to the bitcoin network via software and obtain/parse block header information
- RTL design of Miner/SHA256 controller

Milestone 2:

- Integrate software and hardware. Pass the block header information to the miner and send the results to the bitcoin network/mining pool

Milestone 3:

- Implement multiple miners in parallel.