BITCOIN MINING

Alan Belyea	adbelyea@bu.edu
Ding Zhong	dzhong3@bu.edu
Marcia Sahaya Louis	marcia93@bu.edu
Priya Baskaran	priyab@bu.edu
Tony Ye	tye1@bu.edu

Abstract:

For our project, we designed a bitcoin miner on two platforms, the Nexys 3 and Nexys 4 FPGAs. The bitcoin miner receives an 80 byte input from the bitcoin mining server and uses the SHA-256 hashing algorithm to generate a 256 bit output.

Inside the 80 byte message, there is a field called “target”, which is a 256 bit number that we want to compare the output with. If our hash output is less than target, then we have have generated a bitcoin, so we send our input message back to the server, which will add a bitcoin to our bitcoin wallet. If our output is not less than the target, then we increment a field in the 80 byte message called “nonce” and we try again. By changing the nonce, we have changed the input message, which leads to an entirely different output value due to the pseudo-random nature of the SHA-256 algorithm.

We have parallelized our computation by having multiple hashing units, which work on different inputs at the same time.

HASH Modules:

super.v: This is the top module of the project where all of the modules are instantiated. Four main control signals are block, select, nonce and nonce signal. Block and select signals are used to assign the correct message (block header) from the UART receiver module to different instantiations of hash modules. Each hash module receives a different value of nonce. The output of the hash modules are sent here, and is compared to the target to see if we have generated a coin. The super module also sends signals to the VGA modules, which displays the input and outputs of the hash module on the display.

hash.v: This is the main hashing module, which takes in a 1024 bit input and generates two hashes (called a double hash) every 130 cycles. It takes clk, select, block, nonce signal and message (block header) as inputs. The select signal is used to determine value of the regControl bit which is used to latch the H1- H8 register values in the a-h registers. Based on the current value of block and select, it assigns the input message to the H1-H8 registers . It instantiates H1.v to H8.v and iteration.v for hash computation and outputs 256 bit hash value.

iteration.v: This module does all of the hashing computation, using h1-h8, block, select, message (block header), constant K value and message hash as inputs. The input message is stored in the 17th register of 17x32 bit memory block and shifted every clock cycle. T1 and T2 are computed using a-h, K, and message values, and are latched back into a-h registers. This is repeated 64 times to get correct intermediate hash.

H1.v - H8.v: Each modules has block and nonce signal as input. Based on the block number, H1-H8 registers are assigned either intermediate values or initial values. The outputs are used by the iteration module.

UART Modules:

async.v: This file has three modules in it: a receiver, a transmitter, and a Baud generator. The Baud generator takes in parameters that contain your FPGA clock frequency, the Baud rate you want, the oversampling you want, and the FPGA clock as an input pin. It generates a clock as close to the desired rate as it can, and outputs it to the transmitter and receiver modules.

Async_transmitter: This is the transmitter module that takes in the baud clock generated from the baud generator, 8 bits of data to be transmitted, and a transmit bit. If the transmit bit is high for 1 clock cycle the input data is latched into a register. an internal state register which governs what bits are output to the TxD pin is shifted through every time the baud clock is high. When the transmitter has completely transmitted 8 bits a flag goes high letting other modules know more data can be latched to the transmitter.

Async_receiver: The receiver is the inverse of the transmitter. It has a different clock, one that is fofur times faster than the baud rate. It samples the receiver pin on the UART every time that clock goes high. It starts adding bits to a shift register after seeing a start bit, logic low, and stops adding data after seeing 8 more bits and a stop bit, logic high. It then raises a flag which tells other modules the data is good for the next clock cycle.

R3.v: This module runs on top of the receiver it outputs a message_flag, and an 608 bit register. Every time the receivers' flag goes high, denoting a valid byte can be read, that byte is added to the output array, and an internal counter which keeps track of how many bits have been processed increments. When a full 608 bits has been added to the output register, an output message_flag is set high denoting that the data is valid. This flag will go low when more bytes are received from the receiver module.

Transmitter.v: This module runs on top of async_transmitter, it takes in the transmit signal that controls async_transmitter, and a 32 bit data array that needs to be transmitted through the UART. It outputs a single transmission bit, and a done flag. When the transmit input goes high async_transmitter is turned on. When the transmitters ready flag goes high a counter increments and the next 8 bits in the 32 bit input array are passed to the transmitter. and the done flag, which is output to show weather new data can be sent to transmitter.v goes low. When the counter gets to three, the done_flag goes high again which alerts other modules they can turn the transmitter on again and latch new data in.

VGA display Modules:

terminal.v: This is the top level of VGA modules, it takes three inputs: global clock, hash message, output hash result. The output of this module is the rgb value and hsync/vsync signal for VGA on FPGA. In this project, this top module was merged into super.v module, but normally serves as the VGA shell.

vga_sync.v: This is the VGA controller, taking in a global clock and generating the varying hsync/vsync signal for VGA. Meanwhile, it outputs real coordinates to the graph.v module to generate corresponding rgb signals.

clock_pixel.v: This file transforms the global clock of 100MHz frequency into 25MHz, which is the working frequency of the vga.

graph.v: This is the top level of rgb signal generating modules. It defines the background color and digit color and sends out the rgb signals. In addition, coordinates are processed to split the screen into small grids. The processed coordinates are then used as the index in screen_mem module.

screen_mem.v: This module takes the input message and output hash value and stores them in two arrays. The screen is mapped as 128 x 64 8bit memory array. Based on the index given by graph.v, the inputs are stored in the appropriate memory array location and given as an input to char_mem.v.

char_mem.v: This module is the code book, which converts the input into binary rgb information. The rgb information represents the shape of each char, from 0 to f. “1” is drawn as white and “0” is drawn as black. graph.v assigns the colors to the screen.

Driver program:
windows_serial_port_comm.c: This is a c program that communicates with server and gets the message(block header) and transmitters through the UART module. It also receives the hash output and nonce value from the FPGA and send it backs to the server.