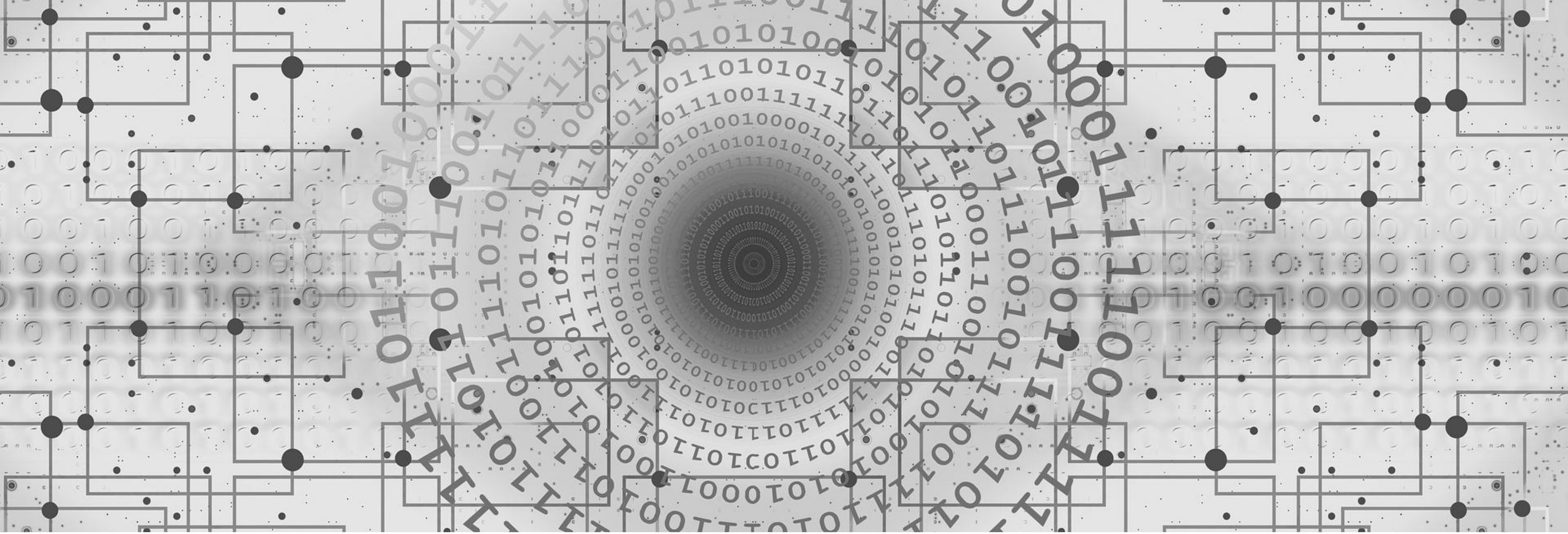


Blockchain, Bitcoin Hashing and Final Project Part-2

ECE-111

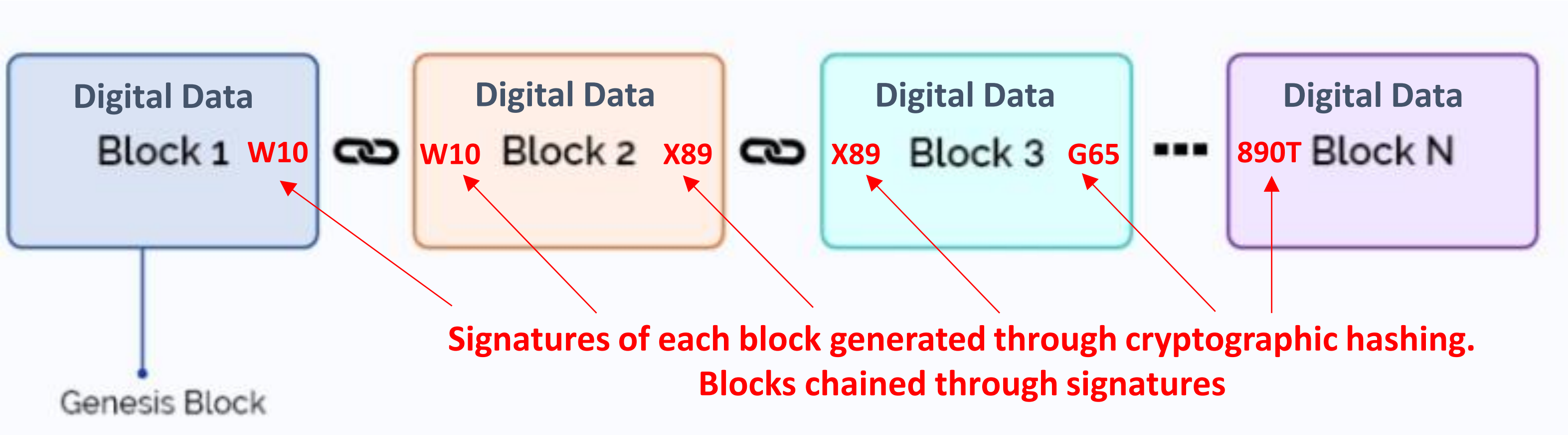
Vishal Karna

Summer 2025

UC San Diego
JACOBS SCHOOL OF ENGINEERING
Electrical and Computer Engineering

Blockchain and Bitcoin Hashing Concepts

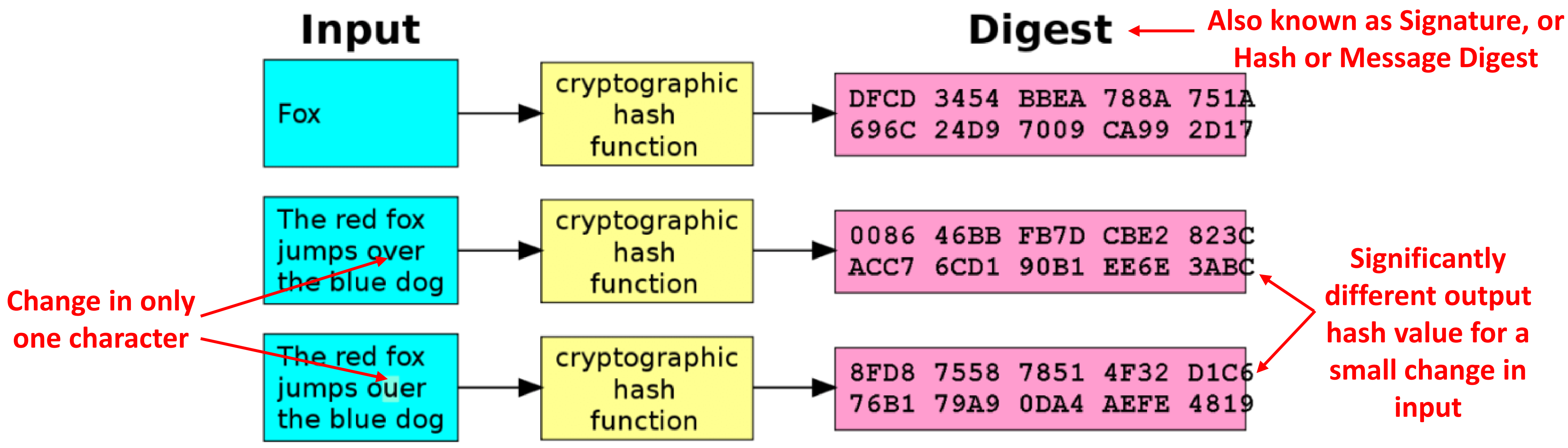
What is a Blockchain ?



❑ A blockchain is a chain of digital data blocks

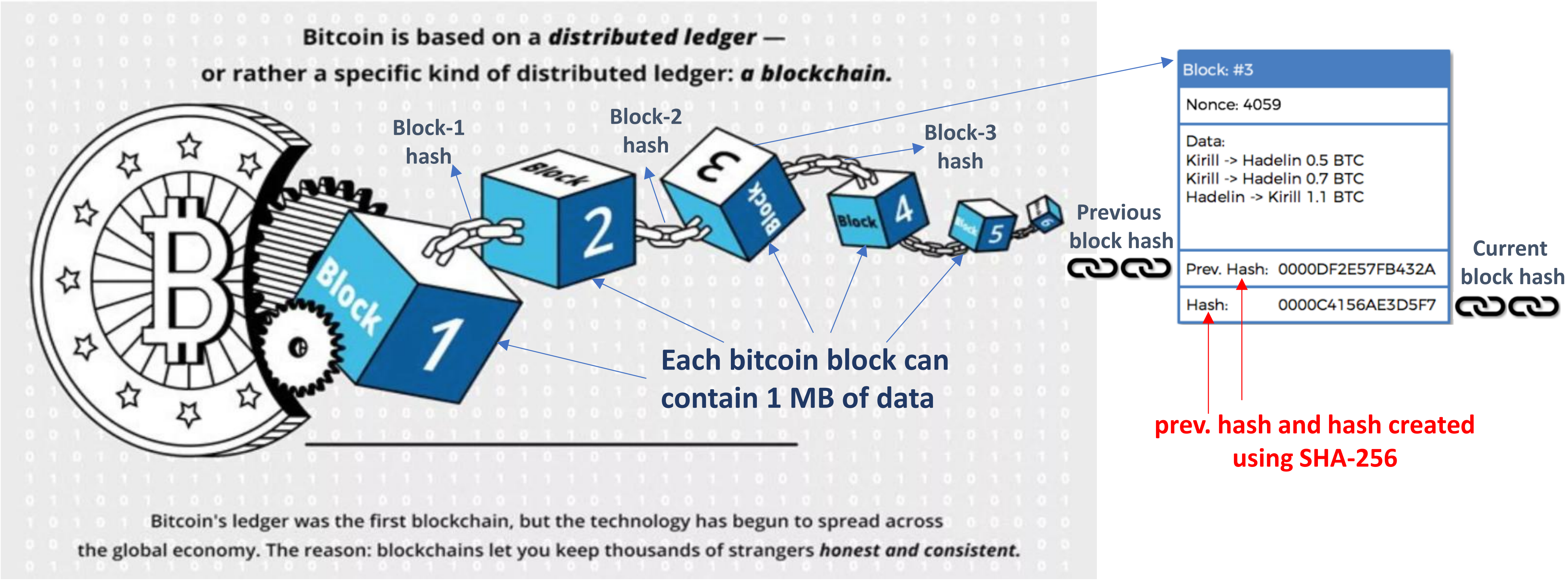
- Each blocks can store digital information about financial transactions such as date, time, dollar, sender, receiver or it can be medical records or property purchase deeds and much more.
- Chaining of blocks is done through cryptographic hashing algorithms, such as **SHA-256**, **Scrypt**, etc
- Blocks which are chained together, its data can never be changed again (**Immutable !**)
- Entire block chain is publicly available to anyone who wants to see it, in exactly the way it was once added to the blockchain.
- Blockchain is a distributed and decentralized public ledger.

Blockchain is dependent on Hashing



- **Hashing is a cryptographic method of converting input data of any kind and size, into a string of fixed number of characters.**
- **Characteristics of hashing cryptography algorithms : (Example : SHA-256, Scrypt, etc)**
  - The same input must always generate the same output. **(Determinstic !)**
  - The hash should be of a fixed number of characters, regardless the size or type of input data **(Compression!)**
  - There should be no way to reverse the hashing process to see the original data set. **(Pre-Image Resistant !)**
  - Any change in the input must produce an entirely different output **(Avalanche effect !)**
  - Practically impossible to find two different inputs that produce the same output **(Collision Resistance !)**
  - Creating the hash should be a fast process that doesn’t make heavy use of computing power **(Efficient !)**

Bitcoin Blockchain

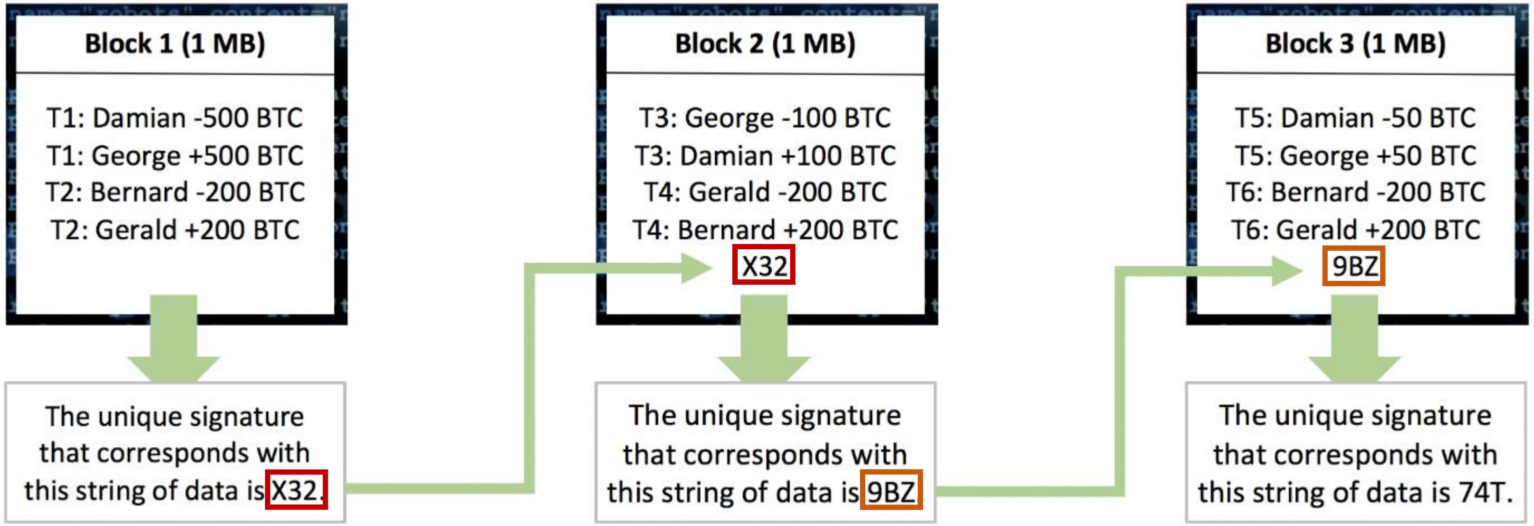


❑ The Bitcoin blockchain is the oldest blockchain in existence.

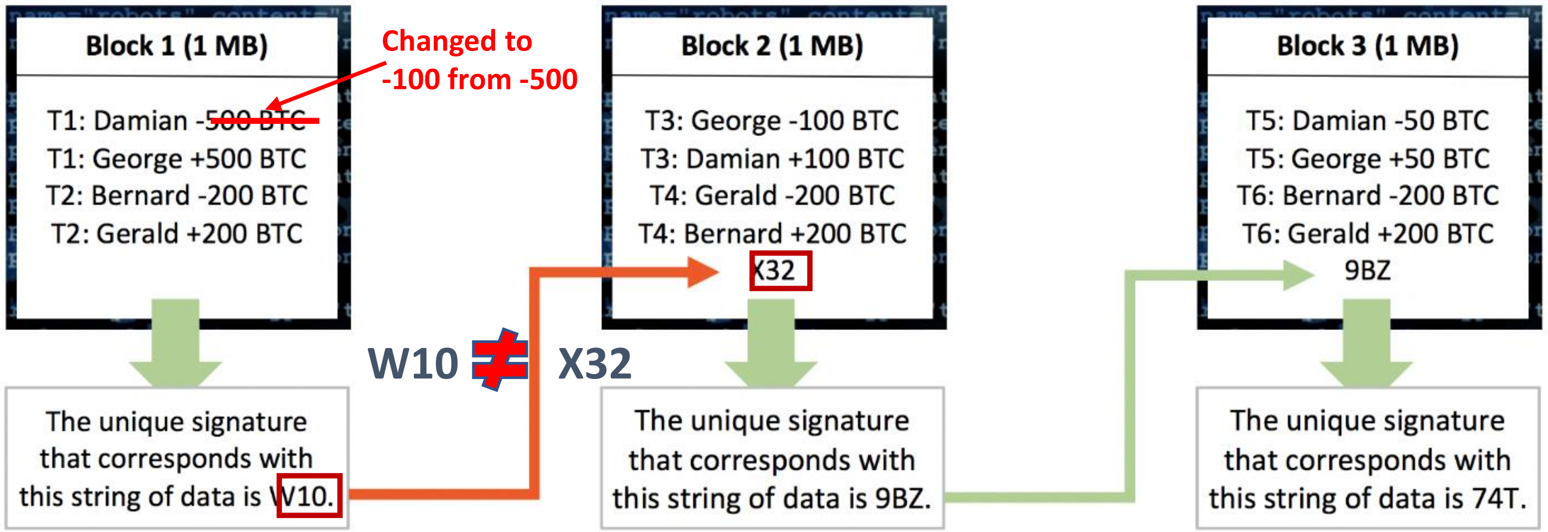
- The blocks on the Bitcoin blockchain consist of approximately 1 MB of data each.
- The data on the Bitcoin blockchain exclusively exists out of *transaction data* in regard to *Bitcoin transactions*.
- It is a giant track record of all the Bitcoin transactions that have ever occurred, all the way back to the very first Bitcoin transaction.
- Bitcoin blockchain uses **SHA-256** cryptographic hashing algorithm to chain blocks

Bitcoin Blockchain Example

- Block-1 registers two bitcoin transactions **T1** and **T2** between Damian and George.
- Signature is generated for Block-1 say, **X32** using hashing algorithm
- Block-2 registers two new bitcoin transactions **T3** and **T4**.
- The data in block 1 is now linked to block 2 by adding the signature of block 1 **(X32)** to the *data* of block 2.
- The signature of block 2 (**9BZ**) is now partially based on the signature of block 1, because it is included in the string of data in block 2
- **The signature links the blocks to each other, making them a chain of blocks !!**



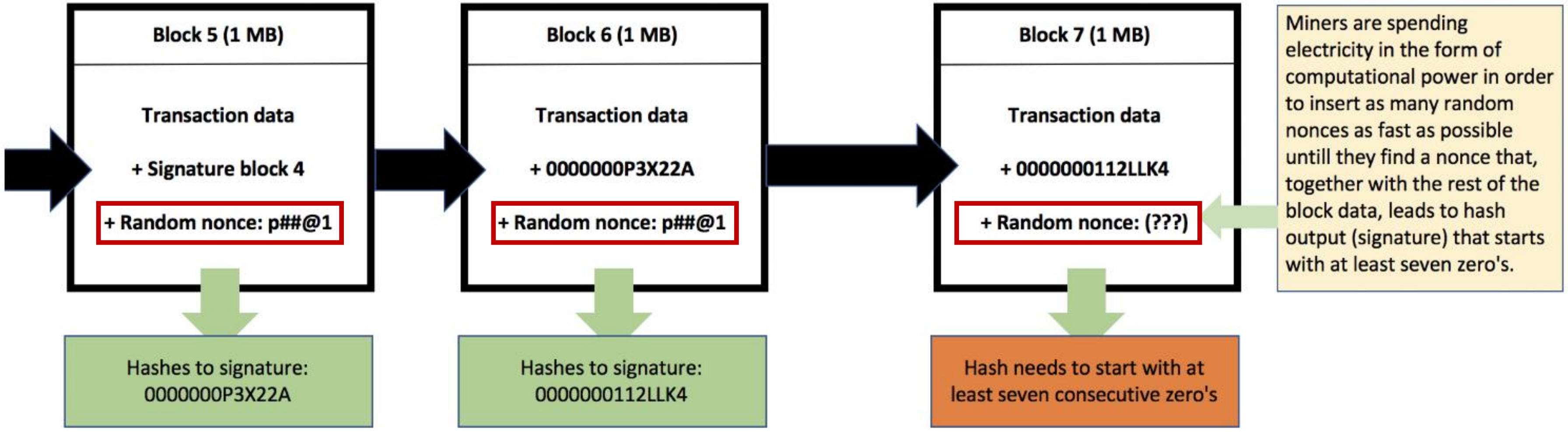
What if block data is altered by malicious user ?



- Let's say transaction between Damian and George is altered and Damian now supposedly sent 100 Bitcoin to George instead of 500 Bitcoin.
- Changing any single bit of data will generate a new signature for Block-1 (say **W10**)
- The new signature W10 does not match the signature that was previously added to block 2 anymore.
  - **Hence Block 1 and 2 are now considered no longer chained to each other !!**
- This indicates to other users of this blockchain that some data in block 1 was altered, and because the blockchain should be immutable :
  - **All users reject this change by shifting back to their previous record of the blockchain where all the blocks are still chained together (the record where Damian sent 100 BTC to George)**

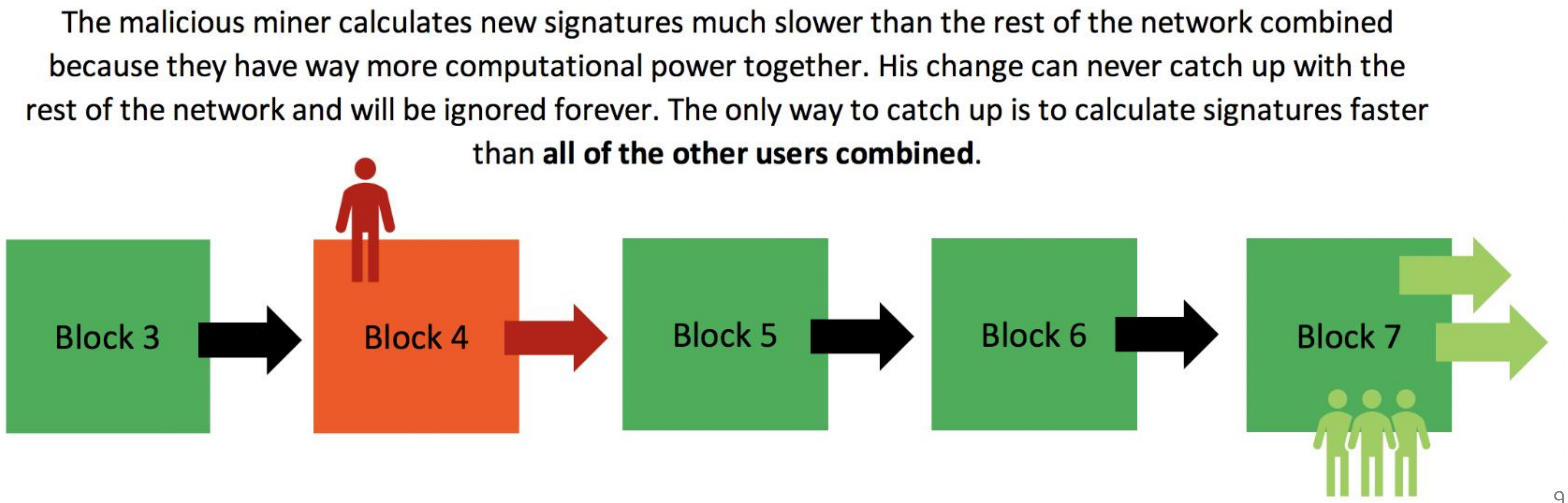
How is the data block accepted in blockchain ?

- A signature doesn't always qualify for block to be accepted in block chain
- A block will *only* be accepted on the blockchain if its digital signature starts with — for example — 7 consecutive number of *zeroes*.
- What if the signature (hash) of a block doesn't start with 7 zeroes?
- Well, in order to find the block a signature that meets the requirements, the string of data of a block needs to be changed *repeatedly* until that specific string of data leads to a signature starting with 7 zeroes.
- Because the transaction data and metadata (block number, timestamp, et cetera) need to stay the way they are, a small specific piece of data is added to every block that has no purpose except for being changed repeatedly in order to find an eligible signature. This piece of data is called the ***nonce*** of a block.
- The **nonce** is a completely random string of **numbers**

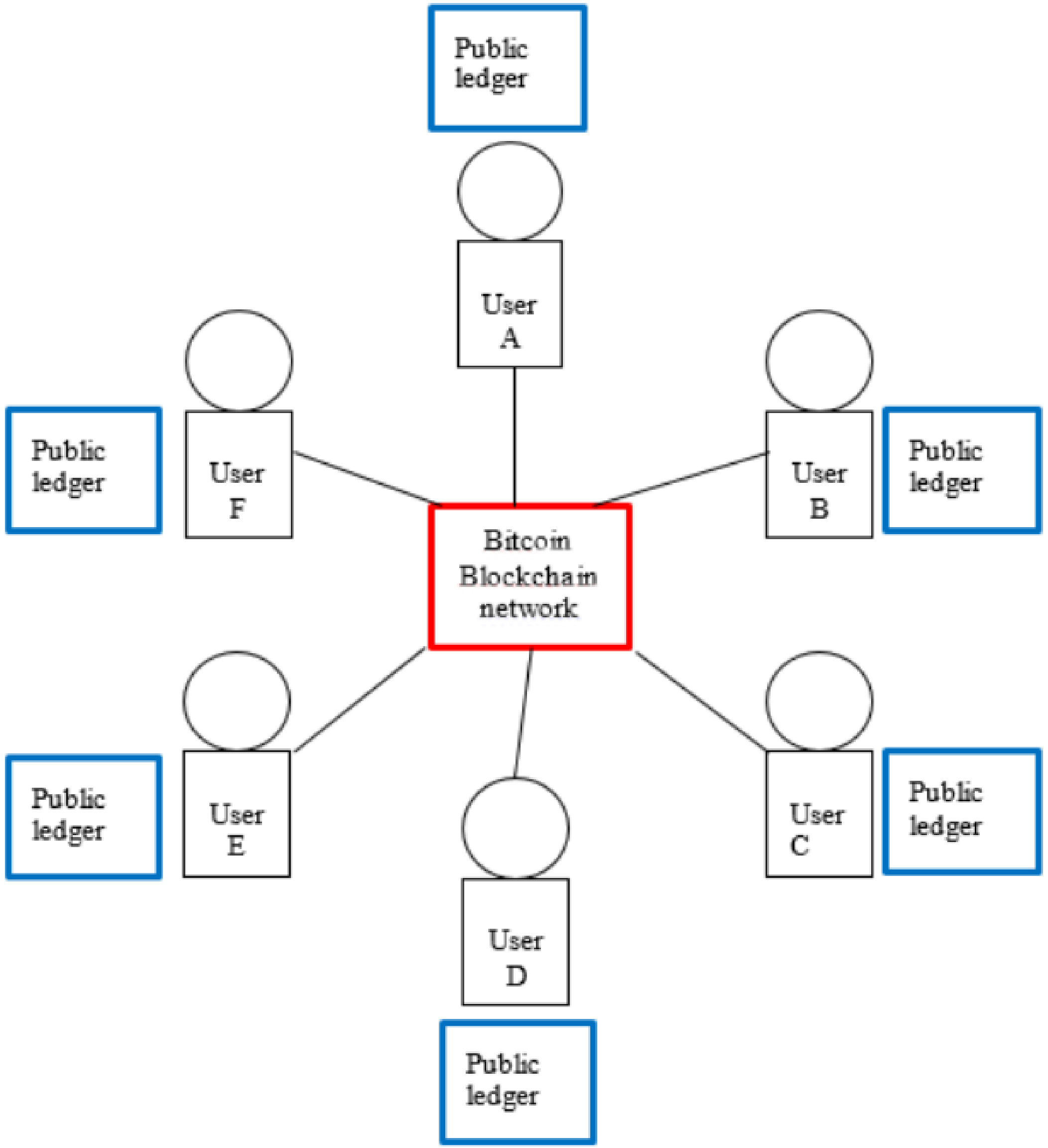


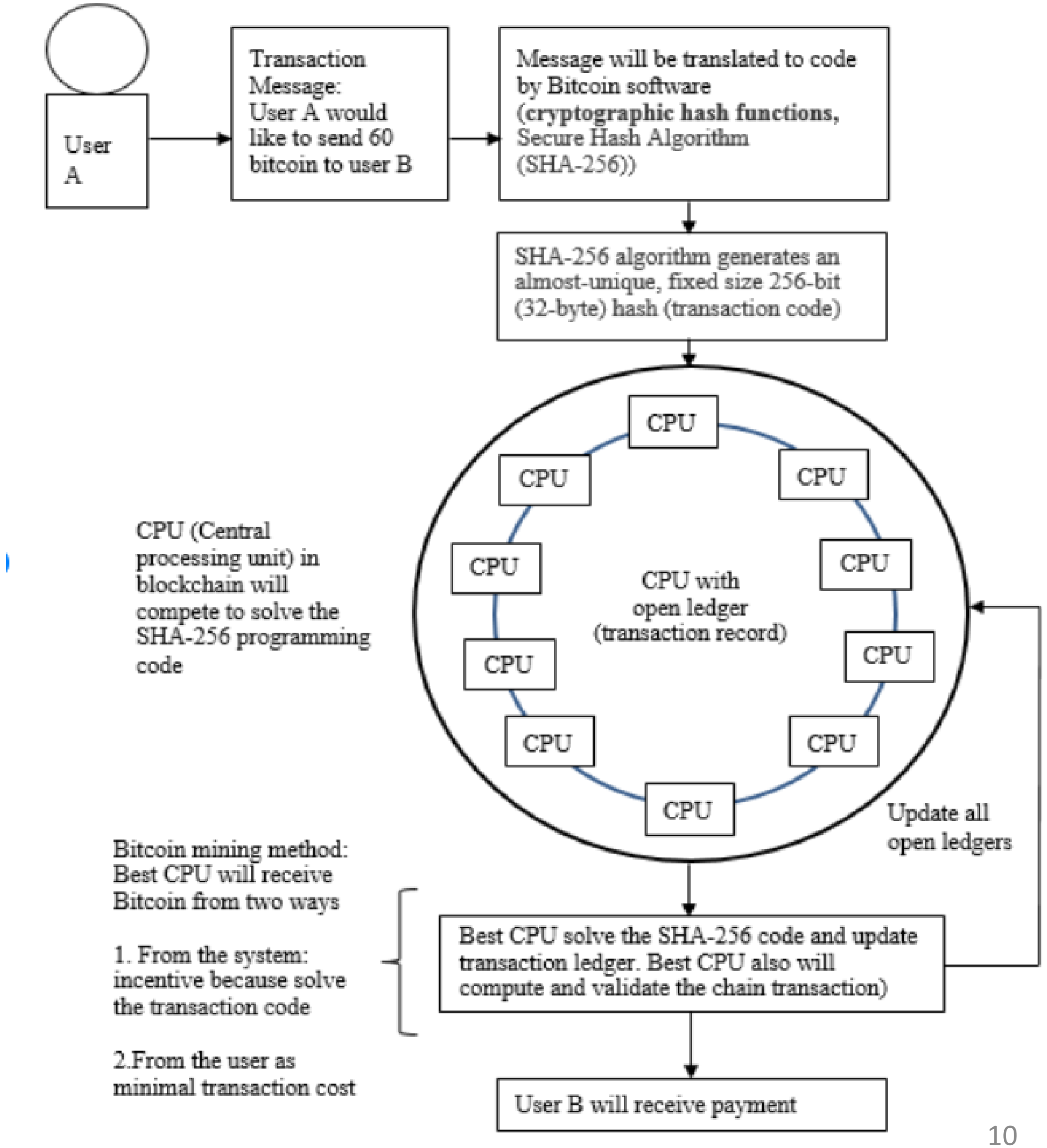
What makes blockchain immutable ?

- Let’s say a *corrupt* miner has altered a block of transactions and is now trying to calculate new signatures for the subsequent blocks in order to have the rest of the network accept his change.
  - The problem for him is, the rest of the network is also calculating new signatures for new blocks.
  - The corrupt miner will have to calculate new signatures for these blocks too as they are being added to the end of the chain. After all, he needs to keep **all of the blocks** linked, including the new ones constantly being added.
  - Unless the miner has more computational power than the rest of the network combined, he will never catch up with the rest of the network finding signatures.



Bitcoin Blockchain Mining Example





Final Project Part-2 : Bitcoin Hashing Project Details and Requirements and Report

Bitcoin Data Block Header

- **Bitcoin's header format :**

Main Point
These 19 words are given and fixed. These 19 words are stored in memory instantiated in testbench code

Field	Purpose	Updated when ...	Size (Words)
Version	Block version number	You upgrade the software and it specifies a new version	1
hashPrevBlock	256-bit hash of the previous block header	A new block comes in	8
hashMerkleRoot	256-bit hash based on all of the transactions in the block	A transaction is accepted	8
Time	Current timestamp as seconds since 1970-01-01T00:00 UTC	Every few seconds	1
Bits	Current target in compact format	The difficulty is adjusted	1
Nonce	32-bit number (starts at 0)	A hash is tried (increments)	1

Try different nonces (0 to 15 nonce value instead of random nonce values)

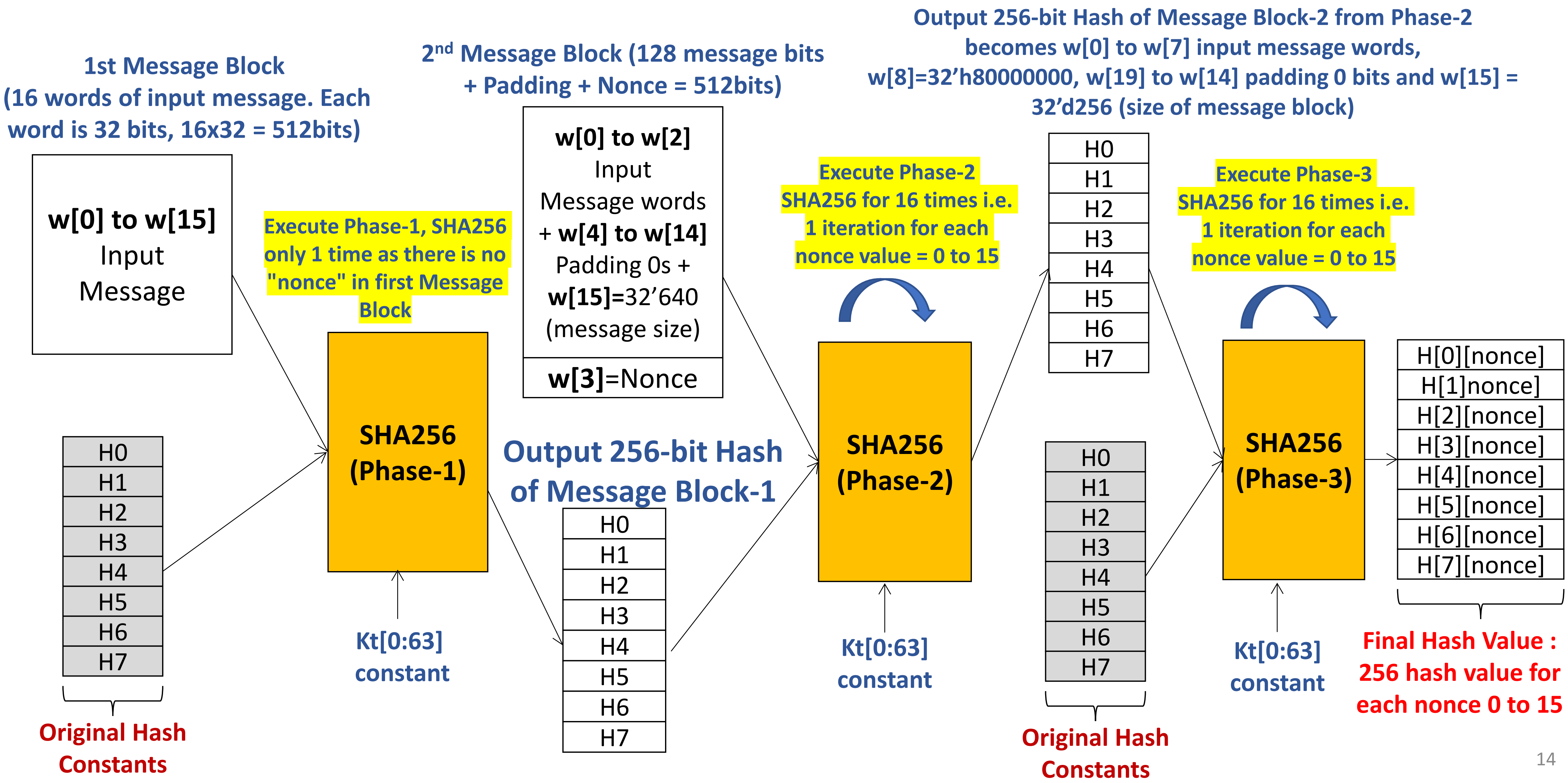
Bitcoin Hashing Project Assumptions

- **Input Message :**
  - Input message (Wt) is of size 20 words (where 1 word = 32 bits)
  - Hence total input message size : 20 x 32 = 640 bits
  - Size of each data block in bitcoin block chain is 512 bits
  - Since 640 bits cannot be stored in 1 digital block, there are total 2 digital blocks created for input message
  - 640 bits of message is stored in memory which is instantiated in testbench code. Actual content of input message is not important for code development and simulation.
  - Out of 20 words, 19 words are data/metadata and final 1 word is a nonce.

- **SHA-256 should be used to create output hash value for each digital block**
  - Since there two digital blocks provided in memory, there will be 2 hash values created (one hash value for each block)

- **For sake of simplicity and convergence, project should assume only 16 nonce values (0 to 15) and hence 16 attempts for each block to create its hash value.**
  - For first block hash value computed is same for each nonce value hence hash value is created only once
  - For second block, hash value is created 16 times. And during each iteration of nonce, nonce value is incremented by 1.

Bitcoin Hashing (Serial Implementation)



Bitcoin Hashing

- **There are 3 phases in bitcoin hashing :**
  - **Phase 1:** Processing 1st block of the 1st SHA 256 hash function
    - H0…H7 correspond to constants, 32'h6a09e667, etc.
    - Wt's correspond to first 16 words in memory
    - Kt[0:63] constant
  - **Phase 2:** Processing 2nd block of the 1st SHA 256 hash function
    - H0…H7 come from the Phase 1
    - Wt's correspond the last 3 words in memory, the nonce, 32'h80000000 padding, ten 32'h00000000 padding words, and 32'd640 message size padding
    - Kt[0:63] constant
  - **Phase 3:** Processing the 2nd SHA 256 hash function
    - H0…H7 correspond to constants, 32'h6a09e667, etc.
    - Wt's correspond the H0…H7 output has value from Phase 2, 32'h80000000 padding, six 32'h00000000 padding words, and 32'd256 message size padding
      - Note : In phase-3 message size is 256 bits as input message is 256-bit output has from phase-2
    - Kt[0:63] constant
  - **Phase-2 and 3 are performed 16 times. This will produce 16 finals hashes.**
    - Note : Phase-2 input message includes 1 word reserved for nonce whereas in Phase-3 in input message there is no nonce value to be added.

Bitcoin Data Block Header

- Compute final hash for SHA256(SHA256(message)) for **16 nonces** = 0, 1, … 15, each message = {block header, nonce}

- Will produce **16** final hashes

  H0[0], H1[0], H2[0], H3[0], H4[0], H5[0], H6[0], H7[0]
  H0[1], H1[1], H2[1], H3[1], H4[1], H5[1], H6[1], H7[1]
  :
  :
  H0[15], H1[15], H2[15], H3[15], H4[15], H5[15], H6[15], H7[15]

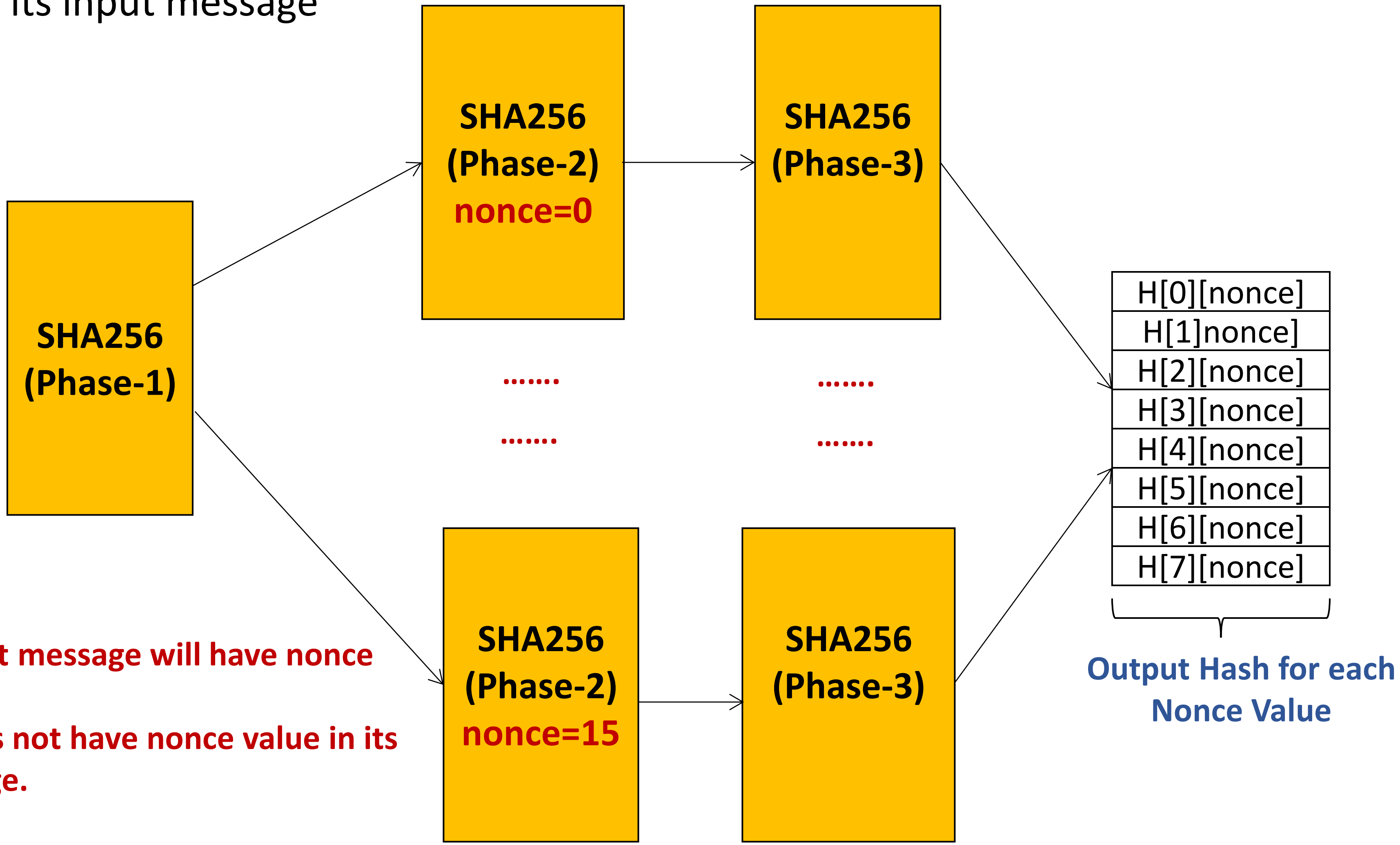
- We will just write to memory H0[0], H0[1] …, H0[15], a total of **16** words

Bitcoin Hashing

- Change input message by changing the “nonce” (32-bits = 1 word), starting with nonce = 0 …
- Keep trying new nonces 1, 2, … until finish hash < target goal
  - **Note :** There is no actual acceptance criteria and difficulty hash value target for this project. Instead after fixed 16 iterations of nonce (from 0 to 15), hash value obtained is considered as meeting target goal (also known as acceptance criteria for hash value)
- For the final project, we will simply compute final hashes for **16 nonces**, nonce = 0, 1, 2, … 15 without checking if any < target
- **Key observation**: The hash computation for the 1st block of the 1st hash is the **<u>same</u>** for all nonce values; therefore, can be computed just once.

Bitcoin Hashing (Parallel Implementation)

- For Each Nonce value, Execute Phase-2 in parallel as inputs for Phase-2 are available at the same time for all nonce values
- Phase-2 and Phase-3 for same nonce value has to be executed serially as Phase-3 needs output hash from Phase-2 as its input message



Note :
- **Phase-2 Input message will have nonce value**
- **Phase-3 Does not have nonce value in its input message.**

Implementing Parallelism

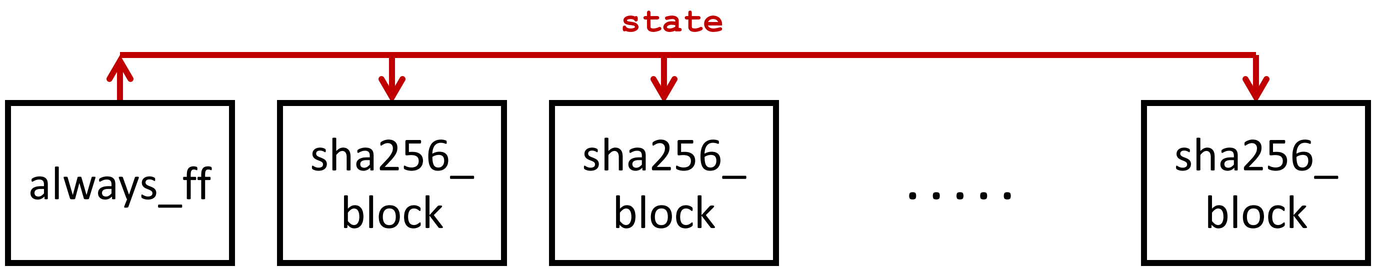
- Can implement “vectorization” like this (effectively doing SIMD execution like a GPU).

```
parameter NUM_NONCES = 16

logic [31:0] A[NUM_NONCES], B[NUM_NONCES], ..., H[NUM_NONCES];

always_ff @(posedge clk, negedge reset_n)
begin
    if (!reset_n) begin
      ...
    end else case (state)
    IDLE:
      ...
    COMPUTE: begin
      ...
      for (int n = 0; n < NUM_NONCES; n++) begin
        {A[n], B[n], ..., H[n]} <= sha256_op(A[n], B[n], ..., H[n], ...);
      end
      ...
    end
    ...
    endcase
end
```

- This will create 16 sets of A, B, … H registers and 16 sets of logic for sha256_op, but under the same state machine control.

Implementing Parallelism

- Can also use module instantiation to create multiple instances of the SHA256 unit.




```
parameter NUM_NONCES = 16

// INSTANTIATE SHA256 MODULES
genvar q;
generate
    for (q = 0; q < NUM_NONCES; q++) begin : generate_sha256_blocks
        sha256_block block (
            .clk(clk),
            .reset_n(reset_n),
            .state(state),
            .mem_read_data(mem_read_data),
            ...);
    end
endgenerate

always_ff @(posedge clk, negedge reset_n)
begin
    ...
end
```

Bitcoin Hashing (Parallel Implementation)

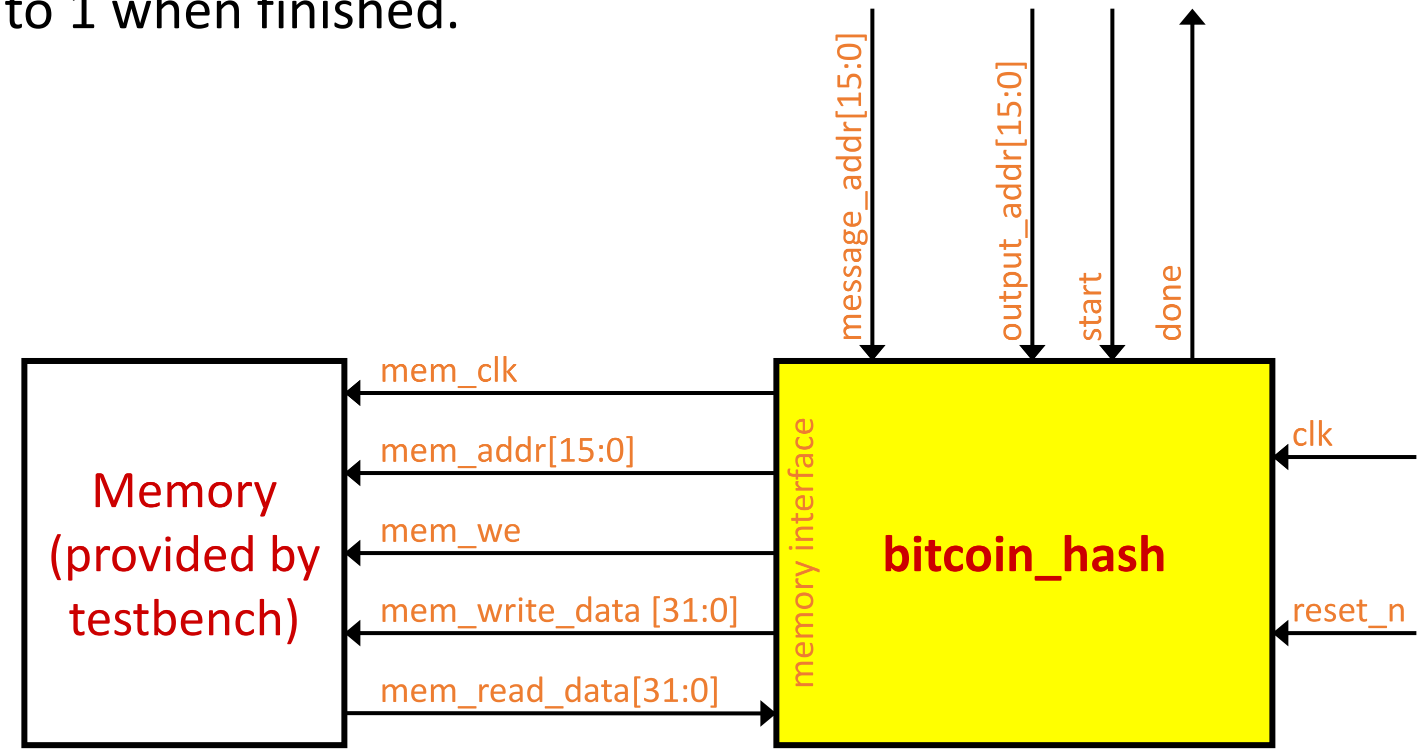
- **To Perform 16 SHA256 operations in parallel, 16 copies of SHA256 logic is required and this will consume more logic within FPGA.**

- **Arria-II FPGA will not be able to fit 16 instances of SHA256. To address this :**
  - First perform in parallel implementation of SHA256 for nonce 0 to 7 and then re-use same logic and one more time perform SHA256 operation in parallel for nonce 8 to 16. This will require 8 instances of SHA256
  - Also, in Part-1 project, simplified SHA256 should be optimized to have w[16] message word array implementation instead of w[63].
    - Not having w[16] implementation, and then still re-using simplified sha256 in part-1 even 8 instances of sha256 will not fit in message array. This is word expansion array, can we perform word expansion using only w[16] instead of w[63] ? – **Answer is yes.**

- **Note to Students :**
  - First Goal should be to make serial implementation of Bitcoin Hashing Model work and make sure functionality of the model is correct using bitcoin_hashing testbench which has self checker in test
  - And then optimize for performance at cost of extra hardware logic and implement parallel implementation

Final Project Module Interface

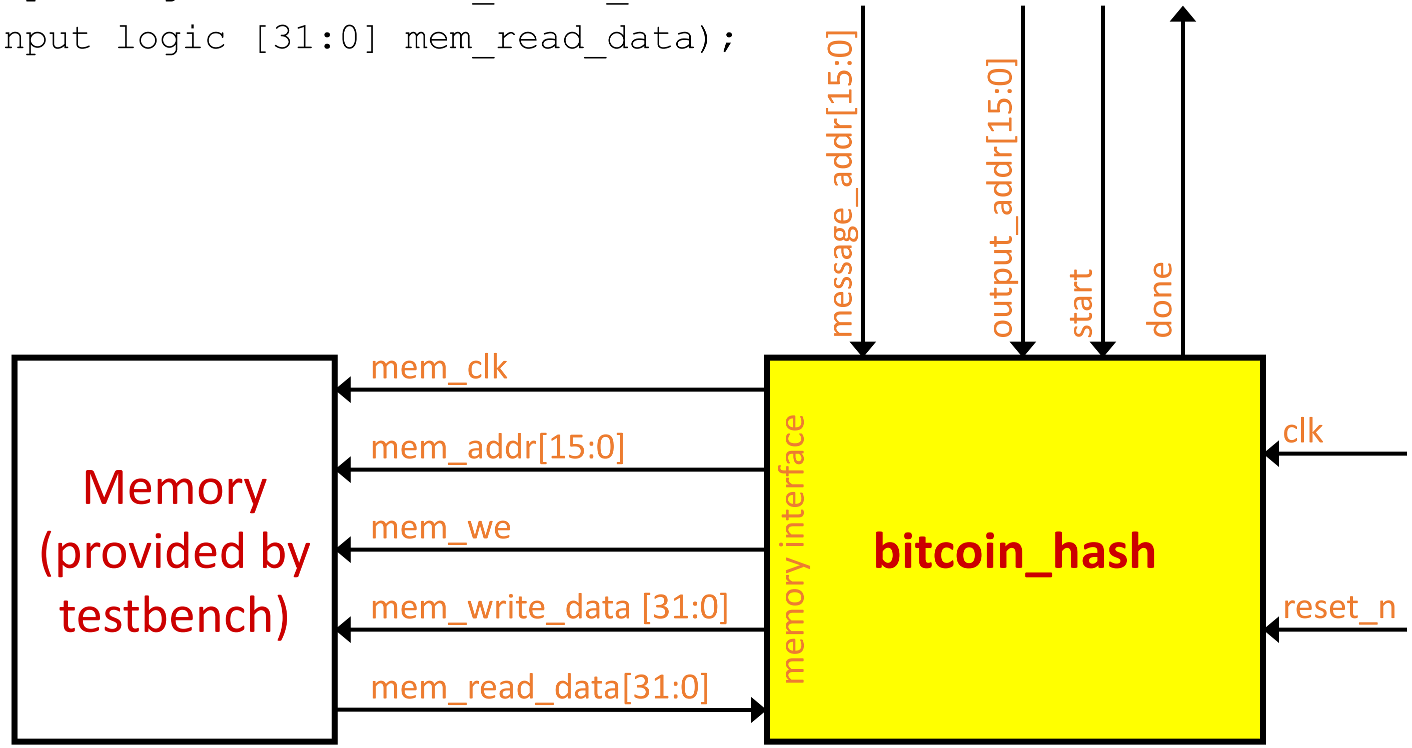
- Wait in idle state for **start**
- Read **19 word** block header starting at **block_addr**
- Compute final hash for SHA256(SHA256(message)) for **16 nonces**, each message = {block header, nonce}
- Just write final **H0** for each of the **16 nonces** into memory starting at **output_addr**.
- Set **done** to 1 when finished.



Final Project Module Interface

- Your assignment is to design the yellow box assuming below mentioned primary ports and module name:

```
module bitcoin_hash (input logic clk, reset_n, start,
                      input logic [15:0] message_addr, output_addr,
                     output logic done, mem_clk, mem_we,
                     output logic [15:0] mem_addr,
                     output logic [31:0] mem_write_data,
                      input logic [31:0] mem_read_data);

    ...
endmodule
```



Writing Final Hash Values to Memory

- Write the final hash values for **H0[0], H0[1] …, H0[15]** in **16 words** to memory starting at **output_addr** as follows:
  - Note : There are 2 blocks for input message, and there is 256-bit (8 words) hash value created for each block hence total (8 words of hash x 2=) 16 words for hash to be stored in memory.

output_addr	H0[0]
output_addr + 1	H0[1]
:	:
:	:
output_addr + 15	H0[15]

Rough Estimation of Cycles

- **Basic implementation: at least 2147 cycles**

Cycle Count	Step	Comments
19	Read 19 words	
64	Process 1st block in 1st SHA256 hash	Same for all 16 nonces
16*64 = 1024	For each nonce, process 2nd block of 1st SHA256 hash	
16*64 = 1024	For each nonce, compute 2nd SHA256 hash	
16	For each nonce, write out H0 (hash value)	

Rough Estimation of Cycles

- Hide reading: at least 2128 cycles

Cycle Count	Step	Comments
64	Process 1^{st} block in 1^{st} SHA256 hash	19 words read "on-the-fly". Same for all 16 nonces
16*64 = 1024	For each nonce, process 2^{nd} block of 1^{st} SHA256 hash	
16*64 = 1024	For each nonce, compute 2^{nd} SHA256 hash	
16	For each nonce, write out H0	

Tips

- Debug your design first with a smaller NUM_NONCES. e.g., by changing the NUM_NONCES parameter in testbench and your design to NUM_NONCES = 1 or NUM_NONCES = 2.

Testbench

```
module tb_bitcoin_hash();

parameter NUM_NONCES = 16
      :
Initial
begin
      :
    $stop;
end
      :
endmodule
```

Your Design

```
module bitcoin_hash(input logic clk, reset_n ...);

parameter NUM_NONCES = 16
           :
always_ff @(posedge clk, negedge reset_n)
begin
    if (!reset_n) begin
           :
    end else case (state)
           :
       endcase
end
endmodule
```

Can change this parameter to try smaller design

Tips

- **Many possible implementations, so no single “right way”.**

- **In FSM code, combined always block for sequential and combination logic using non-blocking assignments should be used**

- **Good rule of thumb is to make your code easy to read.**
  - If there are too many nested if-then-else such that the code is hard to read, try to simplify the code as it tends to lead to better implementations.
  - Minimizing the number of states is not necessarily good if it means that you have to add many if-then-else to effectively recreate the same next-state logic.

- **Add comments before each block of code explaining what it is trying to achieve**

No Inferred Megafunctions or Latches

- **In your Quartus compilation message**
  - **No inferred megafunctions**: Most likely caused by block memories or shift-register replacement. Can turn OFF “Automatic RAM Replacement” and “Automatic Shift Register Replacement” in “Advanced Settings (Synthesis)”. If you still see “inferred megafunctions”, contact Professor. Your design will not pass if it has inferred megafunctions.

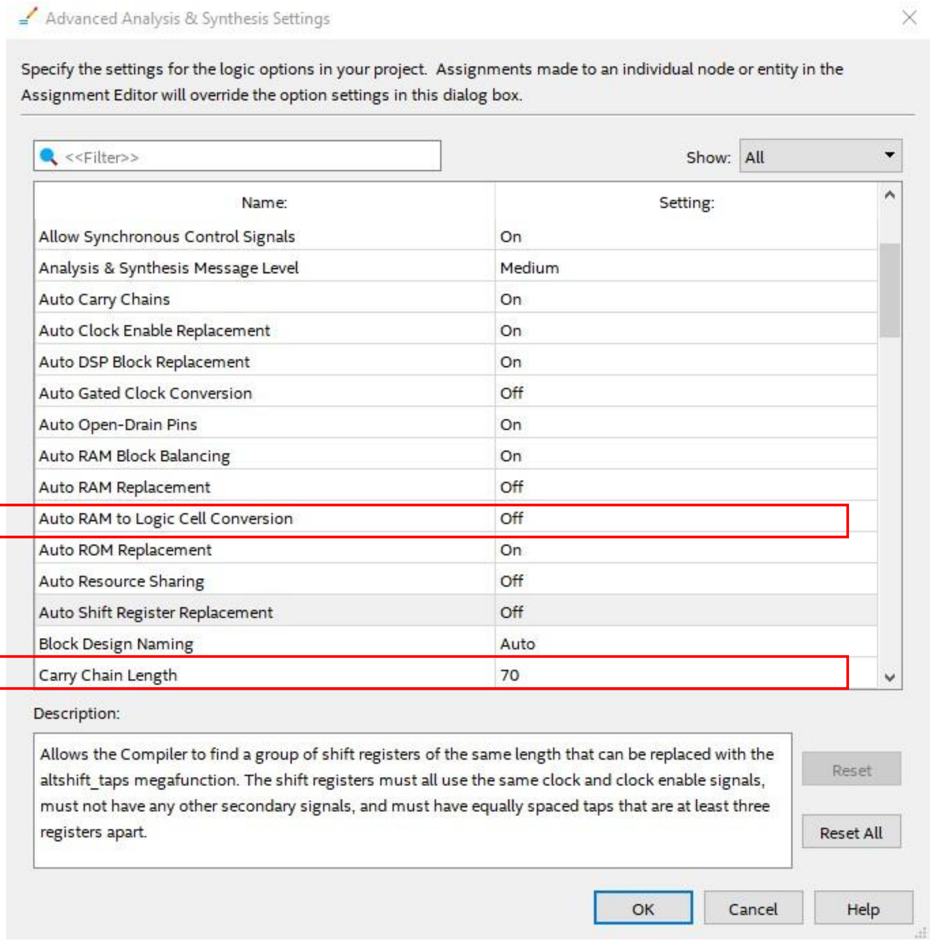
  - **No inferred latches**: Your design will not pass if it has inferred latches.

No Block Memory Bits

- In your bitcoin_hash.fit it **must** say **Total block memory bits is 0** (otherwise will not pass).

```
+--------------------------------------------------------------------------------+
; Fitter Summary                                                                 ;
+------------------------------------+-------------------------------------------+
; Fitter Status                      ; Successful - Wed May 09 15:37:04 2018     ;
; Quartus Prime Version              ; 17.1.0 Build 590 10/25/2017 SJ Lite Edition ;
; Revision Name                      ; bitcoin_hash                              ;
; Top-level Entity Name              ; bitcoin_hash                              ;
; Family                             ; Arria II GX                               ;
; Device                             ; EP2AGX45DF29I5                            ;
; Timing Models                      ; Final                                     ;
; Logic utilization                  ; 8 %                                       ;
;     Combinational ALUTs            ; 2,009 / 36,100 ( 6 % )                    ;
;     Memory ALUTs                   ; 0 / 18,050 ( 0 % )                        ;
;     Dedicated logic registers      ; 1,257 / 36,100 ( 3 % )                    ;
; Total registers                    ; 1257                                      ;
; Total pins                         ; 118 / 404 ( 29 % )                        ;
; Total virtual pins                 ; 0                                         ;
; Total block memory bits            ; 0 / 2,939,904 ( 0 % )                     ;
; DSP block 18-bit elements          ; 0 / 232 ( 0 % )                           ;
```

- If not, go to "Assignments→Settings" in Quartus, go to "Compiler Settings", click "Advanced Settings (Synthesis)"
- Turn OFF "Auto RAM Replacement" and "Auto Shift Register Replacement"



Final Project Submission

- **Put following files into (LastName, FirstName)_(LastName, FirstName)_finalproject.zip folder which should include :**
  - SystemVerilog code design files for both SHA256 and Bitcoin hashing project
  - Modelsim transcript files (i.e. msim_transcript) for both SHA256 and Bitcoin hashing project
  - For both SHA256 and bitcoin hashing provide, fitter and sta files (files with extension .fit, .sta)
  - Report for both SHA256 and Bitcoin hashing project.
    - Merge report in one file or separate either way is fine
    - Report file should be submitted as PDF file
  - Finalsummary.xls file with fmax, number of cycles, aluts, registers detail filled. Template of this file is provided as part of Final_Project.zip folder.
    - **finalsummary.xls should be submitted for both SHA256 and bitcoin hash!**

- **Note :**
  - Each project group to submit only one Final project zip folder on gradescope
  - Ensure final project zip folder and files within it can be opened on windows PC machine

Final Project Submission

- **Final report should be saved in PDF file format and it should include following mentioned :**
  - Explain briefly what SHA-256 is and bitcoin hashing (may use lecture slide contents)
  - Describe algorithm for both SHA-256 and Bitcoin hashing implemented in your code
  - Simulation waveform snapshot for both SHA-256 and Bitcoin hashing
  - Provide modelsim transcript window output indicating passing test results generated from self-checker in testbench for both SHA-256 and Bitcoin hashing
  - Provide synthesis resource usage and timing report for bitcoin_hash only.
    - Should include ALUTs, Registers, Area, Fmax snapshots
    - Provide fitter report snapshot. And provide Timing Fmax report snapshots
    - Make sure to use **Arria II GX EP2AGX45DF29I5** device and use Fmax for **Slow 900mV 100C Mod**

Note : Please make sure not to submit final report, SystemVerilog file and required other files as a .rar file

Final Project Submission

- **Put following files into (LastName, FirstName)_(LastName, FirstName)_finalproject.zip**
  - Both design files and also testbench code for both SHA256 and Bitcoin hashing project
  - Modelsim transcript files msim_transcript for both SHA256 and Bitcoin hashing project
  - For both SHA256 and bitcoin hashing provide, fitter and sta files (files with extension .fit, .sta)
  - Report for both SHA256 and Bitcoin hashing project
  - Finalsummary.xls file with fmax, number of cycles, aluts, registers detail filled. Template of this file is provided as part of Final_Project.zip folder. This should be submitted for both SHA256 and bitcoin hash

- **Final report should including following mentioned :**
  - Explain briefly what SHA-256 is and bitcoin hashing (may use lecture slide contents)
  - Describe algorithm for both SHA-256 and Bitcoin hashing implemented in your code
  - Simulation waveform snapshot for both SHA-256 and Bitcoin hashing
  - Provide modelsim transcript window output indicating passing test results generated from self-checker in testbench for both SHA-256 and Bitcoin hashing
  - Provide synthesis resource usage and timing report for bitcoin_hash only.
    - Should include ALUTs, Registers, Area, Fmax snapshots
    - Provide fitter report snapshot
    - Provide Timing Fmax report snapshots
    - Make sure to use **Arria II GX EP2AGX45DF29I5** device and use Fmax for **Slow 900mV 100C Mod**

Self-Reading on Project Submission Details

Fill up finalsummary.xlsx

- Fill up finalsummary.xlsx posted on Piazza as part of Final_Project.zip (to be filled for both simplified_sha256 bitcoin_hash project in separate fillsummary.xlsx)

Last Name	First Name	Student ID	SectionId	Email	Compiler Settings	#ALUTs	#Registers	Area	Fmax (MHz)	#Cycles	Delay (microsec)	Area*Delay (millisec*area)
SMITH	ROBERT BENJAMIN	A12345678	925042	r.smith@ucsd.edu	balanced	31607	20932	52539	134.01	242	1.806	94.877
JONES	ALICE MARIE	A23456789	925044	a.jones@ucsd.edu	balanced	31607	20932	52539	134.01	242	1.806	94.877

- If you worked alone, just fill out one row
- Spreadsheet already contains calculation fields: e.g. Area = #ALUTs + #Registers. Please use them.
- Students to fill ALUTs, Registers, Fmax and Cycles column in excel sheet.
- #cycles will be generated for your design from testbench code.
- Make sure to use **Arria II GX EP2AGX45DF29I5** device
- Make sure to use Fmax for **Slow 900mV 100C Model**
- Make sure to use **Total number of cycles**

How to Fill up finalsummary.xlsx

- See annotated snapshot below on how to get ALUT's, Registers, Fmax, Cycles information to provide in finalsummary.xlsx

Fmax Delay Area*Delay

Ensure in Quartus compiler settings you have set "Balanced Effort". See piazz post @223 on this. https://piazza.com/class/kqfrsqh3xmy5ui?cid=223

These three columns values are auto-generated by formula embedded in fillsummary.xls. So students should not fill these three columns. Once Student provides information in ALUTS, Registers, Fmax, #Cycles column then formula will auto calculate values in Area, Delay and Area*Delay Column

Last Name	First Name	Student ID	SectionId	Email	Compiler Settings	#ALUTs	#Registers	Area	Fmax (MHz)	#Cycles	Delay (microsec)	Area*Delay (millisec*area)
SMITH	ROBERT BENJAMIN	A12345678	925042	r.smith@ucsd.edu	balanced	31607	20932	52539	134.01	242	1.806	94.877
JONES	ALICE MARIE	A23456789	925044	a.jones@ucsd.edu	balanced	31607	20932	52539	134.01	242	1.806	94.877

This can be found in Synthesis report in Quartus

This can be found in synthesis report. Look for below mentioned in report to get this value.

Total registers
-- Dedicated logic registers

Quartus Timing Analysis report. See piazza post @222 for more details. You need to fill Fmax for Slow 900mV 100C corner. https://piazza.com/class/kqfrsqh3xmy5ui?cid=222

After simulation is performed in Modelsim and when tests completes. Then in modelsim transcript window you should a message generated from testbench with total number of cycles for your design implementation. See example below. This is also available in msim_transcript file.

Total number of cycles: 242

bitcoin_hash.fit Fitter Report

- Copy of the **fitter reports** (not the flow report) with area numbers.
- Make sure to use **Arria II GX EP2AGX45DF29I5** device
- **IMPORTANT**: Make sure **Total block memory bits is 0**.

```
+----------------------------------------------------------------------------------+
; Fitter Summary                                                                   ;
+------------------------------------+---------------------------------------------+
; Fitter Status                      ; Successful - Wed May 09 15:37:04 2018       ;
; Quartus Prime Version              ; 17.1.0 Build 590 10/25/2017 SJ Lite Edition ;
; Revision Name                      ; bitcoin_hash                                ;
; Top-level Entity Name              ; bitcoin_hash                                ;
; Family                             ; Arria II GX                                 ;
; Device                             ; EP2AGX45DF29I5                              ;
; Timing Models                      ; Final                                       ;
; Logic utilization                  ; 8 %                                         ;
;     Combinational ALUTs            ; 2,009 / 36,100 ( 6 % )                      ;
;     Memory ALUTs                   ; 0 / 18,050 ( 0 % )                          ;
;     Dedicated logic registers      ; 1,257 / 36,100 ( 3 % )                      ;
; Total registers                    ; 1257                                        ;
; Total pins                         ; 118 / 404 ( 29 % )                          ;
; Total virtual pins                 ; 0                                           ;
; Total block memory bits            ; 0 / 2,939,904 ( 0 % )                       ;
; DSP block 18-bit elements          ; 0 / 232 ( 0 % )                             ;
; Total GXB Receiver Channel PCS     ; 0 / 8 ( 0 % )                               ;
; Total GXB Receiver Channel PMA     ; 0 / 8 ( 0 % )                               ;
; Total GXB Transmitter Channel PCS  ; 0 / 8 ( 0 % )                               ;
; Total GXB Transmitter Channel PMA  ; 0 / 8 ( 0 % )                               ;
; Total PLLs                         ; 0 / 4 ( 0 % )                               ;
; Total DLLs                         ; 0 / 2 ( 0 % )                               ;
+------------------------------------+---------------------------------------------+
```

bitcoin_hash.sta

- Copy of the sta (static timing analysis) reports.
- Make sure to use Fmax for **Slow 900mV 100C Model**
- **IMPORTANT**: Make sure "clk" is the ONLY clock.
- You must,
  assign mem_clk = clk;
- Your bitcoin_hash.sta.rpt must show "clk" is the <u>**only**</u> clock.

```
+---------------------------------------------------+
; Slow 900mV 100C Model Fmax Summary                ;
+------------+-----------------+------------+------+
; Fmax       ; Restricted Fmax ; Clock Name ; Note ;
+------------+-----------------+------------+------+
; 151.95 MHz ; 151.95 MHz      ; clk        ;      ;
+------------+-----------------+------------+------+
```

Optimization in Quartus

- In practice, these modes don't always do what you want, so wait until the end to try out different optimization modes.

Optimization mode	Description
Balanced	Optimizes synthesis for balanced implementation that respects timing constraints.
Performance (High effort - increases runtime)	Makes high effort to optimize synthesis for speed performance. High effort increases synthesis run time.
Performance (Aggressive - increases runtime and area)	Makes aggressive effort to optimize synthesis for speed performance. Aggressive effort increases synthesis run time and device resource use.
Power (High effort - increases runtime)	Makes high effort to optimize synthesis for low power. High effort increases synthesis run time.
Power (Aggressive - increases runtime, reduces performance)	Makes aggressive effort to optimize synthesis for low power. Aggressive effort increases synthesis time and reduces speed performance.
Area (Aggressive - reduces performance)	Makes aggressive effort to reduce the device area required to implement the design.

Some Possible & Median Results

- ❑ Targeting Delay Only: effectively create 16 SHA256 units to work in parallel
- ❑ Targeting Area*Delay: effectively use one SHA256 unit to enumerate 16 nonces

	Possible Delay Only	Median Delay Only	Possible Area*Delay	Median Area*Delay
#ALUTs	25,201	31,607	1,627	1,525
#Registers	19,432	20,932	1,230	2,076
Area	44,633	52,539	2,857	3,601
Fmax (Mhz)	182.55	134.01	179.21	151.92
#Cycles	225	242	2,201	2,252
Delay (microsecs)	1.233	1.806	12.282	14.821
Area*Delay (millisec*area)	55.012	94.877	35.089	53.369