Lecture 10: Bitcoin Hashing

UC San Diego
JACOBS SCHOOL OF ENGINEERING
Electrical and Computer Engineering

Bitcoin Hashing

- Bitcoin's header:

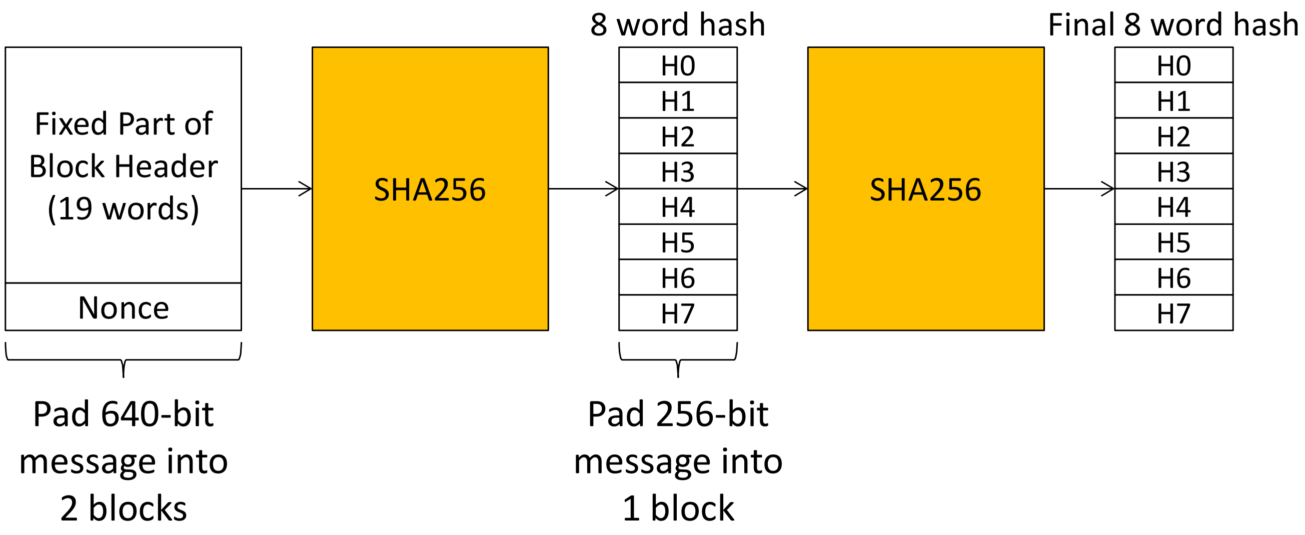
<u>Main Point</u>
These 19 words are given and fixed

Try different nonces

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 <u>target</u> in compact format	The <u>difficulty</u> is adjusted	1
Nonce	32-bit number (starts at 0)	A hash is tried (increments)	1

Bitcoin Hashing

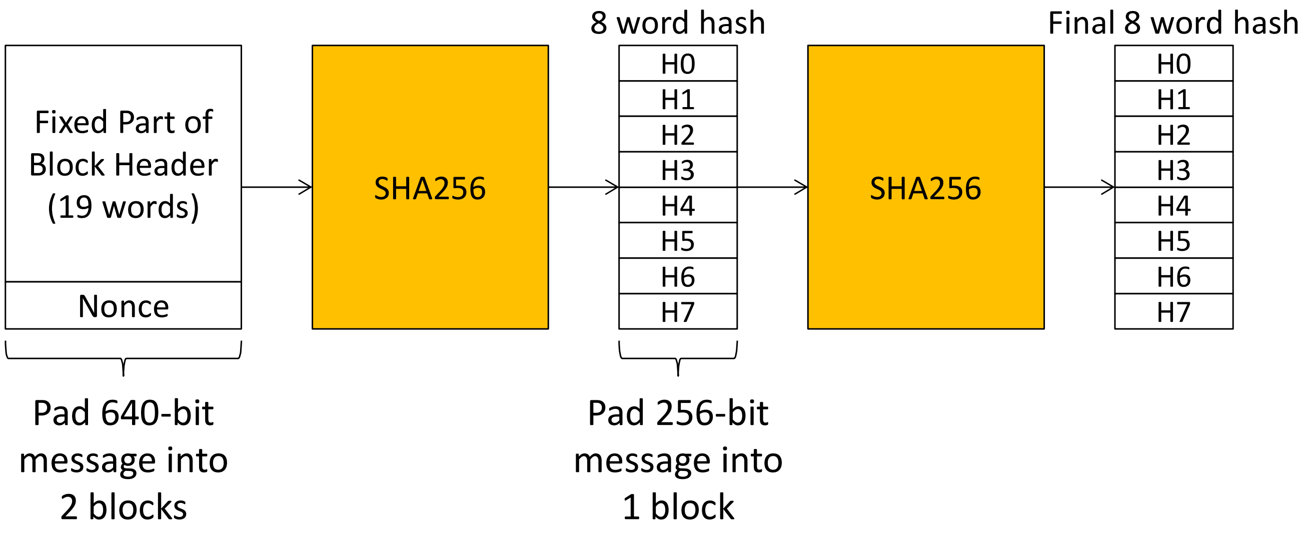
- Bitcoin hashing


8 word hash
Final 8 word hash
Fixed Part of Block Header (19 words)
Nonce
SHA256
SHA256
H0
H1
H2
H3
H4
H5
H6
H7
Pad 640-bit message into 2 blocks
Pad 256-bit message into 1 block

- 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

Bitcoin Hashing

- For the final project, we will simply compute final hashes for **16 nonces**, nonce = 0, 1, 2, … 15 without checking if any < target


8 word hash
Final 8 word hash
Fixed Part of Block Header (19 words)
Nonce
SHA256
SHA256
H0
H1
H2
H3
H4
H5
H6
H7
Pad 640-bit message into 2 blocks
Pad 256-bit message into 1 block

- **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

- There are 3 phases:
  - 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
  - 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, ten 32’h00000000 padding, and 32’d640
  - Phase 3: Processing the 2nd SHA 256 hash function
    - H0…H7 correspond to constants, 32’h6a09e667, etc.
    - Wt’s correspond the H0…H7 from Phase 2, 32’h80000000, six 32’h00000000 padding, and 32’d256

Bitcoin Hashing

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

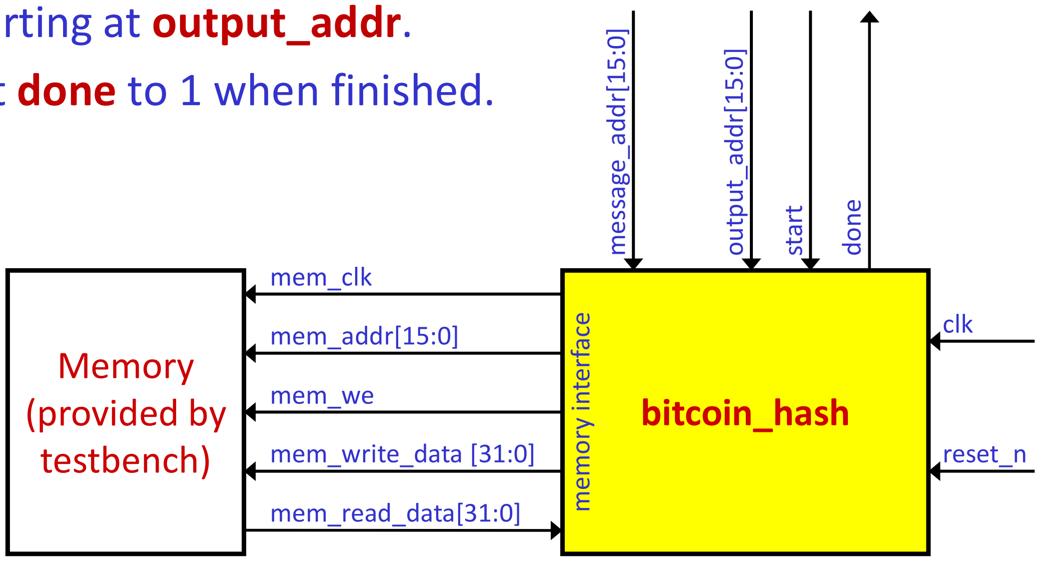
- Will produce **16** final hashes

  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

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

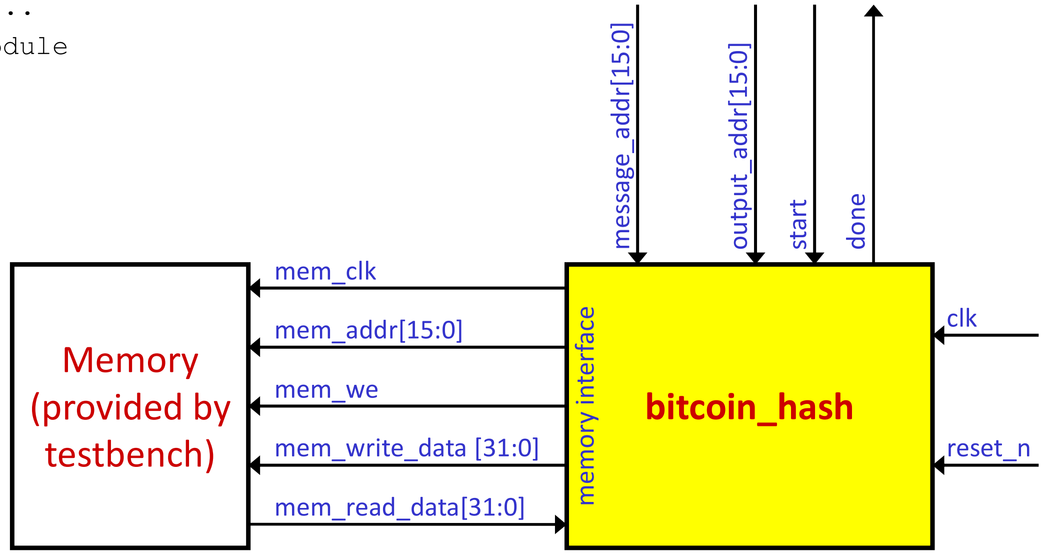
- Write the final hash values for **H0[0], H0[1] …, H0[15]** in **16 words** to memory starting at **output_addr** as follows:

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

Final Project Module Interface

- Your assignment is to design the yellow box:

```
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
```



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	

Rough Estimation of Cycles

- Hide reading: at least 2128 cycles

Cycle Count	Step	Comments
64	Process 1st block in 1st SHA256 hash	19 words read "on-the-fly". 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	

Rough Estimation of Cycles

- Parallel execution: at least 208 cycles

Cycle Count	Step	Comments
64	Process 1st block in 1st SHA256 hash	19 words read "on-the-fly". Same for all 16 nonces
64	**For all 16 nonces**, compute in **parallel** the 2nd block of 1st SHA256 hash	Requires more hardware
64	**For all 16 nonces**, compute in **parallel** the 2nd SHA256 hash	Requires more hardware
16	For each nonce, write out H0	

Implementing Parallelism

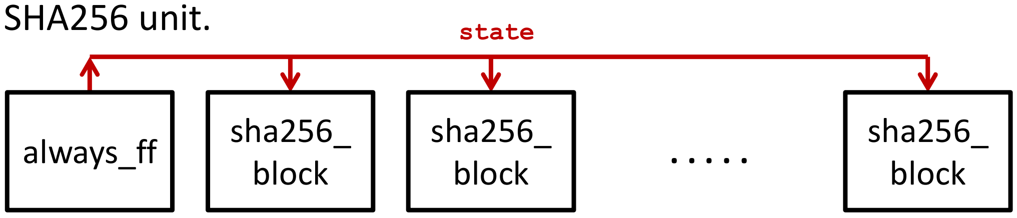
- Can implement “vectorization” like this (effectively doing SIMD execution like a GPU).

```
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

- Project 4
  - Design a “sequential” version to minimize Area*Delay

- Project 5 (Final Project)

  Submit 2 designs
  - An “sequential” version to minimize Area*Delay (can be same as Project 4 or improved version)
  - A “parallel” version to minimize Delay only

- Final Project Grading
  - Area*Delay score normalized by mean and stdev: s1 = $(X - \mu)/\sigma$
  - Delay score normalized by mean and stdev: s2 = $(X - \mu)/\sigma$
  - Score = s1 + s2, so both metrics equally important

Bitcoin Hashing

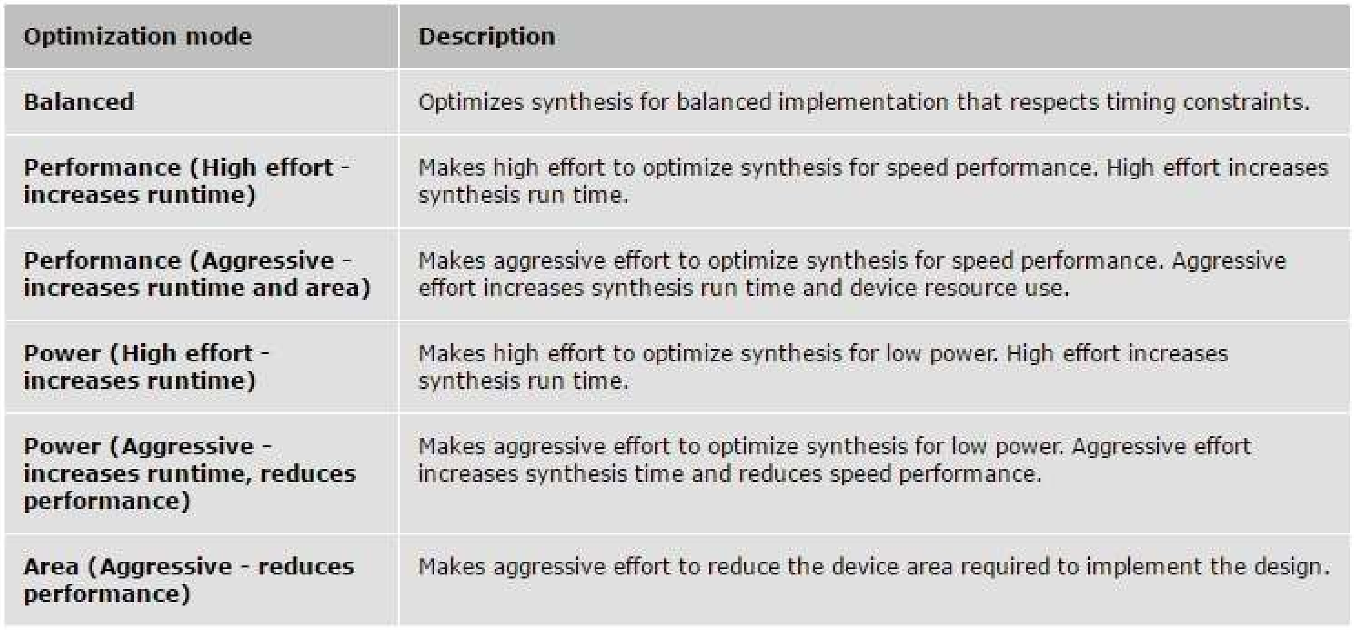
- Testbench
  - tb_bitcoin_hash.sv

- Intermediate values
  - bitcoin_hash.xlsx

- Intermediate W values
  - bitcoin_hash_w_values.xlsx

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

Tips

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

- 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.

- Complexity: Should be possible to implement complete design in 300-400 lines of code.

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

Design Review Meetings

- Wed 2/13, Wed 2/20, Mon 2/25, and Wed 2/27 class times will be used for design review meetings
- Will post Piazza and class webpage announcement later this evening.
- If you are doing fine with your design, please allow other groups to sign up first (meaning wait until tomorrow evening to sign up for remaining open slots).

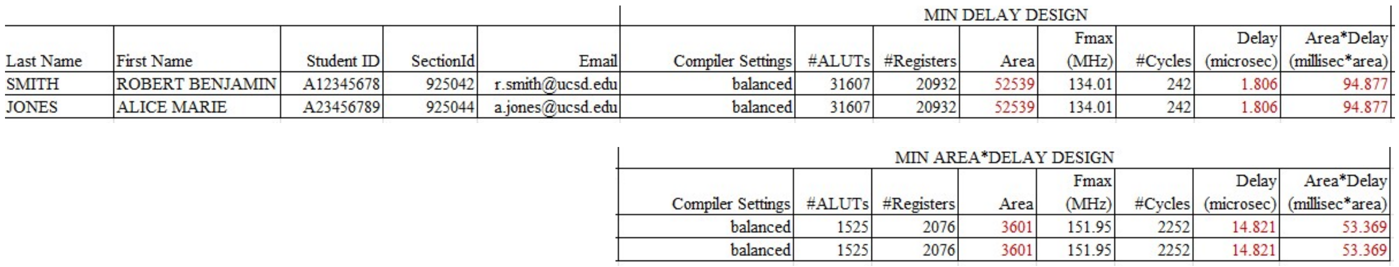
Final Project Submission

Put following files into
(LastName, FirstName)_(LastName, FirstName)_finalproject.zip

- finalsummary.xlsx (see link in Project5 page or Class schedule page)
- bitcoin_hash1.sv (min delay) and bitcoin_hash2.sv (min area*delay). Add other sv files if you split your designs into different sv files.
- transcript1.txt (min delay) and transcript2.sv (min area*delay)
- message1.txt (min delay) and message2.sv (min area*delay)
- bitcoin_hash1.fit.rpt (min delay) and bitcoin_hash2.fit.rpt (min area*delay)
- bitcoin_hash1.sta.rpt (min delay) and bitcoin_hash2.sta.rpt (min area*delay)

finalsummary.xlsx

- See finalsummary.xlsx template provided



Last Name	First Name	Student ID	SectionId	Email	MIN DELAY DESIGN 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

MIN AREA*DELAY DESIGN Compiler Settings	#ALUTs	#Registers	Area	Fmax (MHz)	#Cycles	Delay (microsec)	Area*Delay (millisec*area)
balanced	1525	2076	3601	151.95	2252	14.821	53.369
balanced	1525	2076	3601	151.95	2252	14.821	53.369

- See link to this spreadsheet in Final Project (Project 5) page
- If you worked alone, just fill out one row
- Spreadsheet already contains calculation fields: e.g. Area = #ALUTs + #Registers. Please use them.
- 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**

bitcoin_hash1.sv and bitcoin_hash2.sv

- Name your “min delay” design “bitcoin_hash1.sv” and your “min area*delay” design “bitcoin_hash2.sv”
- Include other sv files if you have more and rename them as needed.

transcript1.txt and transcript2.txt

- Copy of the ModelSim simulation results.
- Just need simulation results for **tb_bitcoin_hash.sv**.
- After you run the “run –all” command, you can save your transcript by going to the “File” menu and clicking on “save transcript as”.
- Transcript file will contain the history of all commands used in the current modelsim session. You can clear the current transcript by going to the “Transcript” menu on the GUI and clicking “Clear”.
- Use **Total number of cycles** for your cycle count.

message1.txt and message2.txt

- Copy of the Quartus compilation messages.
- You can save the messages by “right-clicking” the message window and choosing “save message”
- **IMPORTANT**: Make sure that are no warnings about “latches” or “inferred latches”.

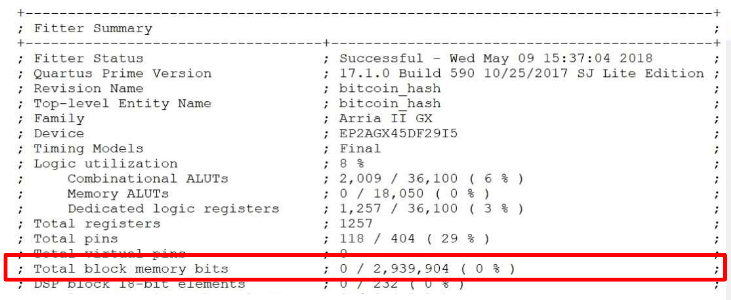
bitcoin_hash1.fit.rpt and bitcoin_hash2.fit.rpt

- Copy of the **<u>fitter reports</u>** (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**.

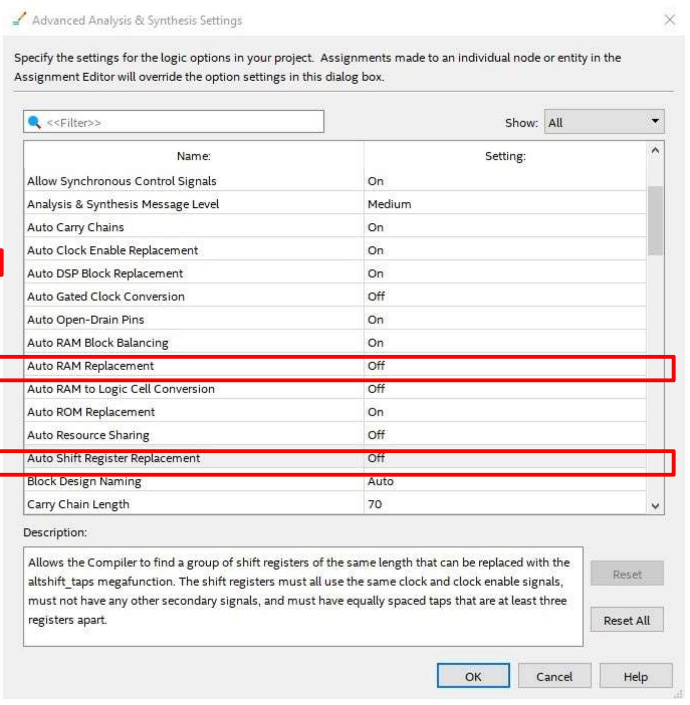
```
+--------------------------------------------------------------------------------+
; Fitter Summary                                                                 ;
+------------------------------------+-------------------------------------------+
; Fitter Status                      ; Successful - 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 % )                             ;
+------------------------------------+-------------------------------------------+
```

No Block Memory Bits

- In your bitcoin_hash1.fit.rpt and bitcoin_hash2.fit.rpt files, they **<u>must</u>** 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 % )                           ;
```



Advanced Analysis & Synthesis Settings

Specify the settings for the logic options in your project. Assignments made to an individual node or entity in the Assignment Editor will override the option settings in this dialog box.

<<Filter>> Show: All

Name:	Setting:
Allow Synchronous Control Signals	On
Analysis & Synthesis Message Level	Medium
Auto Carry Chains	On
Auto Clock Enable Replacement	On
Auto DSP Block Replacement	On
Auto Gated Clock Conversion	Off
Auto Open-Drain Pins	On
Auto RAM Block Balancing	On
Auto RAM Replacement	Off
Auto RAM to Logic Cell Conversion	Off
Auto ROM Replacement	On
Auto Resource Sharing	Off
Auto Shift Register Replacement	Off
Block Design Naming	Auto
Carry Chain Length	70

Description:

Allows the Compiler to find a group of shift registers of the same length that can be replaced with the altshift_taps megafunction. The shift registers must all use the same clock and clock enable signals, must not have any other secondary signals, and must have equally spaced taps that are at least three registers apart.

- 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”

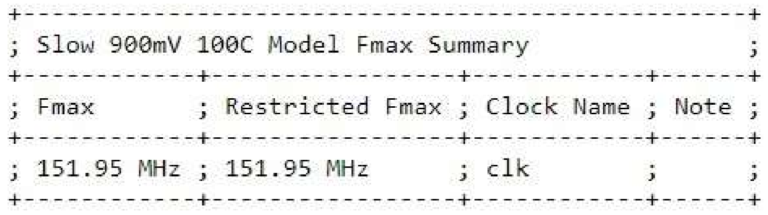
No Inferred Megafunctions/Latches

- 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.

bitcoin_hash1.sta.rpt and bitcoin_hash2.sta.rpt

- 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_hash1.sta.rpt and bitcoin_hash2.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        ;      ;
+------------+-----------------+------------+------+
```