# Lecture 11: Final Project Submission

UC San Diego
JACOBS SCHOOL OF ENGINEERING
Electrical and Computer Engineering

## Plan for Rest of the Quarter

- **Remaining Lectures**
  - Today 3/4 (A00@ 2-3:20 pm, B00 @3:30-4:50 pm in EBU1-2315):
    Go over final project submission and optimization tips
  - Wed 3/13 (A00@ 2-3:20 pm, B00 @3:30-4:50 pm in EBU1-2315):
    Last lecture on VHDL and Testbenches

- **Additional Design Review Meetings** (Note: Times changed to 3:30-6:30 pm instead of 2-3:20 pm/3:30-4:50 pm, and location changed to EBU1-5101 instead of EBU1-2315)
  - Wed 3/6: 3:30-6:30 pm in EBU1-5101
  - Mon 3/11: 3:30-6:30 pm in EBU1-5101
  - Thu 3/14: 3:30-6:30 pm in EBU1-5101

- Sign up sheet for addition design review meetings to be posted on Piazza tonight.

- No office hours Mon 3/11 @1-1:50 pm for Prof. Lin. Instead replaced by 3 additional hours for Design Reviews on Thu 3/14 @3:30-6:30 pm.

## Updates on Submission

- In addition to your source codes, Quartus and ModelSim related files, and the finalsummary.xlsx spreadsheet, you will need to submit a “Final Report”
- A template will be provided to you as “Final-Report-Template.docx” (to be discussed shortly)
- 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
- The “Final Report” will count for 5% of your final project on a P/NP basis as follows:
  - If you don’t submit a final report, 5% would be deducted from your normalized Delay-only and Area*Delay scores.

## Final Report Template

- “Final-Report-Template.docx” provides an outline of what you need in the report.

- You can use figures, bullet points, etc. from slides/class materials, but use your own words in the writing.

## 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 |

## Important Optimizations

- Use only 16 w arrays. e.g.,
  - Avoid logic [31:0] w[64];
- Avoid multiplexing/decoding delays. e.g.,
  - Avoid {a, b, …} <= sha256_op(a, b, …, w[t]);
  - Avoid w[t] <= …
  - Avoid w[t-15], w[t-2], …
- Set up pipeline to compute "sha256_op" and pre-compute "next w" in same cycle.
  - COMPUTE: begin

```
    {a, b, …} <= sha256_op(a, b, …, w[15]);
    w[15] <= mem_read_data;  // memory word for t + 1
    mem_addr <= …; // set memory address for t + 3
end
```

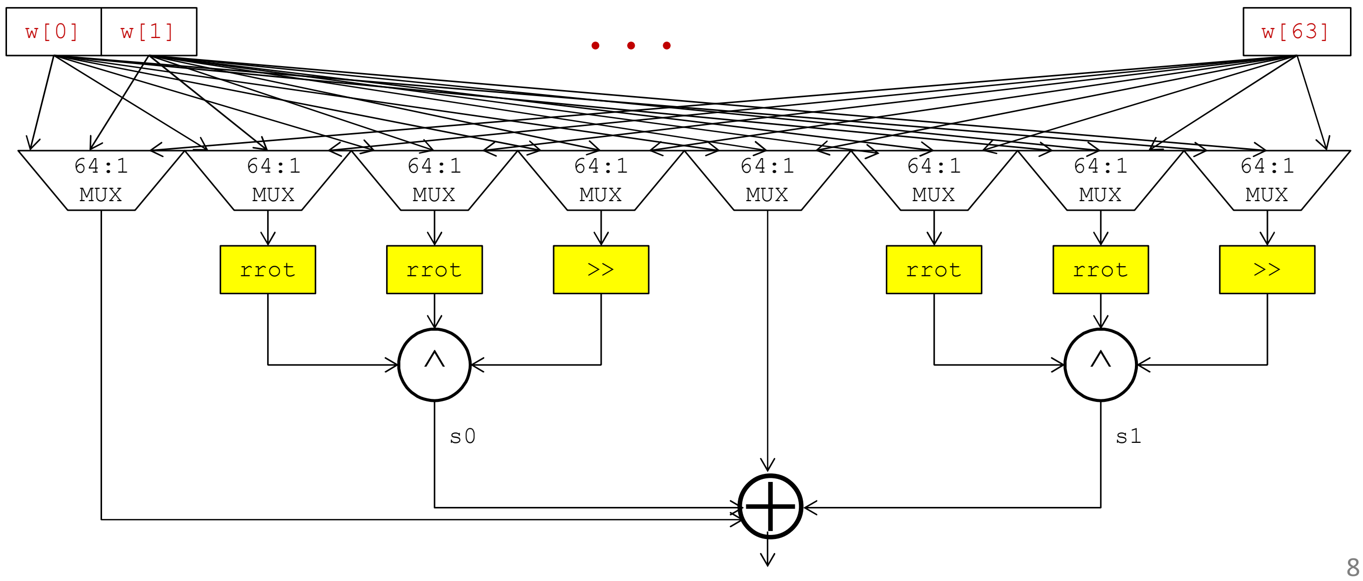
- Make sure "w-expansion logic" not in the critical path of "sha256_op" logic
  - {a, b, …} <= sha256_op(a, b, …, w[15]); // w[15] reg breaks path from wtnew
    w[15] <= wtnew();

## Important Optimizations

- Most important optimization for "Delay" is doing all 16 nonces in parallel.
  - However, your design has to be simple enough to fit 16 parallel computations on to our FPGA device.
  - Quartus will fail during "fitter" if design is too large. e.g.,
    - "Error (170012): Fitter requires 2019 LABs to implement the design, but the device contains only 1805 LABs"
    - "Error (171000): Can't fit design in device"
  - Try temporarily changing "NUM_NONCES" to 8 or something and find ways to simplify the design before switching back to "NUM_NONCES = 16"

## Hint for W[n] Array

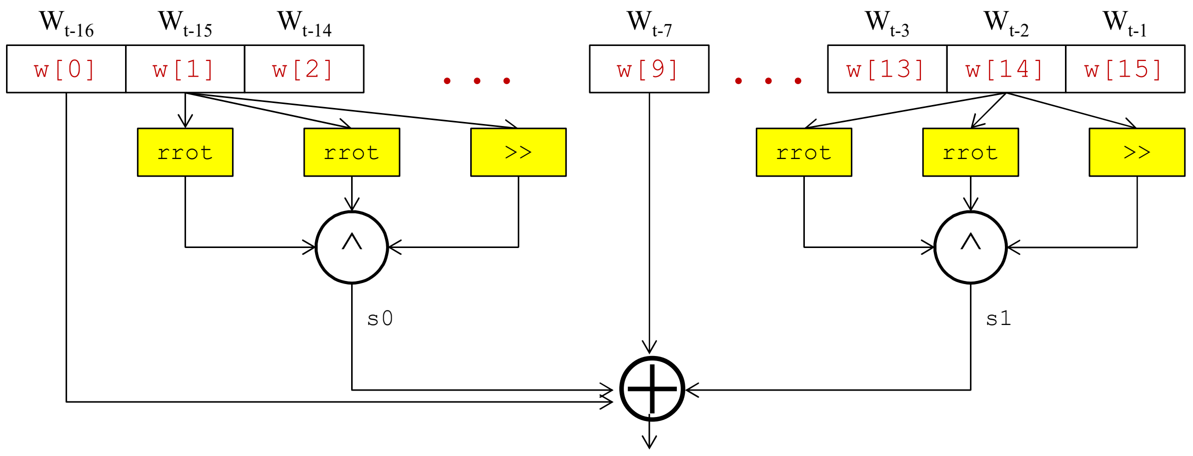
```
• function logic [31:0] wtnew; // function with no inputs
     logic [31:0] s0, s1;

     s0 = rrot(w[t-15],7)^rrot(w[t-15],18)^(w[t-15]>>3);
     s1 = rrot(w[t-2],17)^rrot(w[t-2],19)^(w[t-2]>>10);
     wtnew = w[t-16] + s0 + w[t-7] + s1;
  endfunction
```



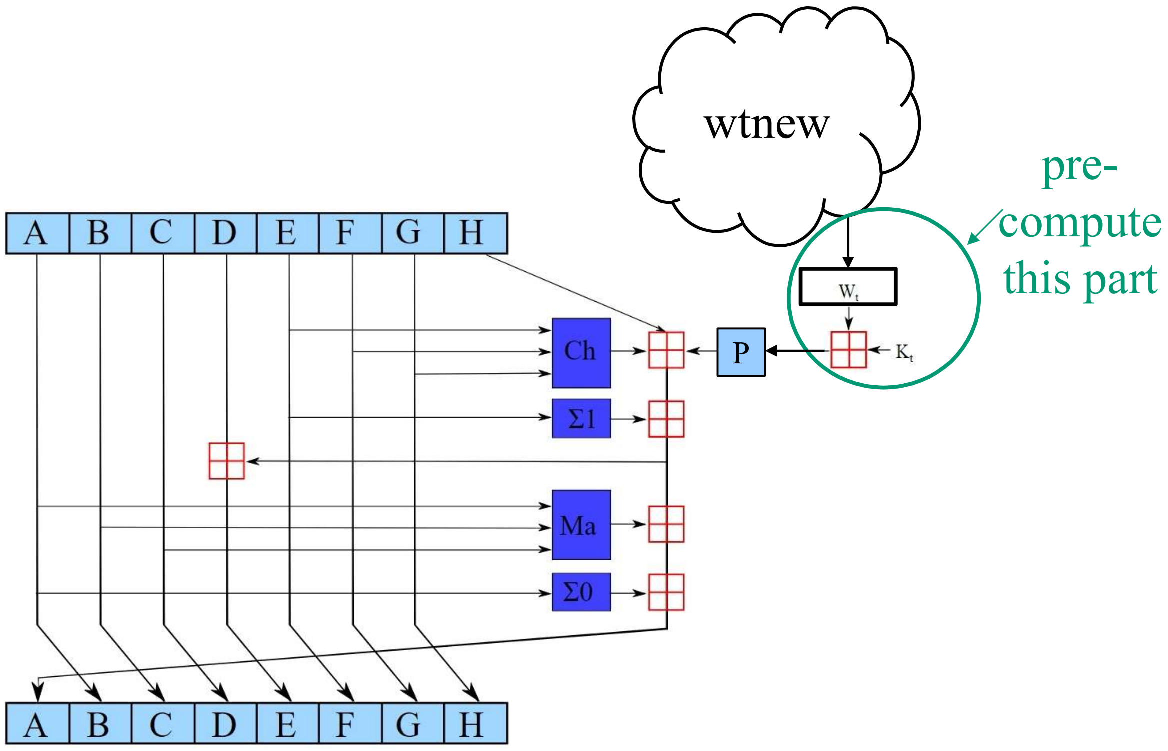
## Hint for W[n] Array

- We can do the following (i.e, “t-15” is “i = MAX – 15 = 1” for MAX = 16, so therefore $W_{t-15}$ would be `w[1]`). Then

```
function logic [31:0] wtnew; // function with no inputs
    logic [31:0] s0, s1;

    s0 = rrot(w[1],7)^rrot(w[1],18)^(w[1]>>3);
    s1 = rrot(w[14],17)^rrot(w[14],19)^(w[14]>>10);
    wtnew = w[0] + s0 + w[9] + s1;
endfunction
```

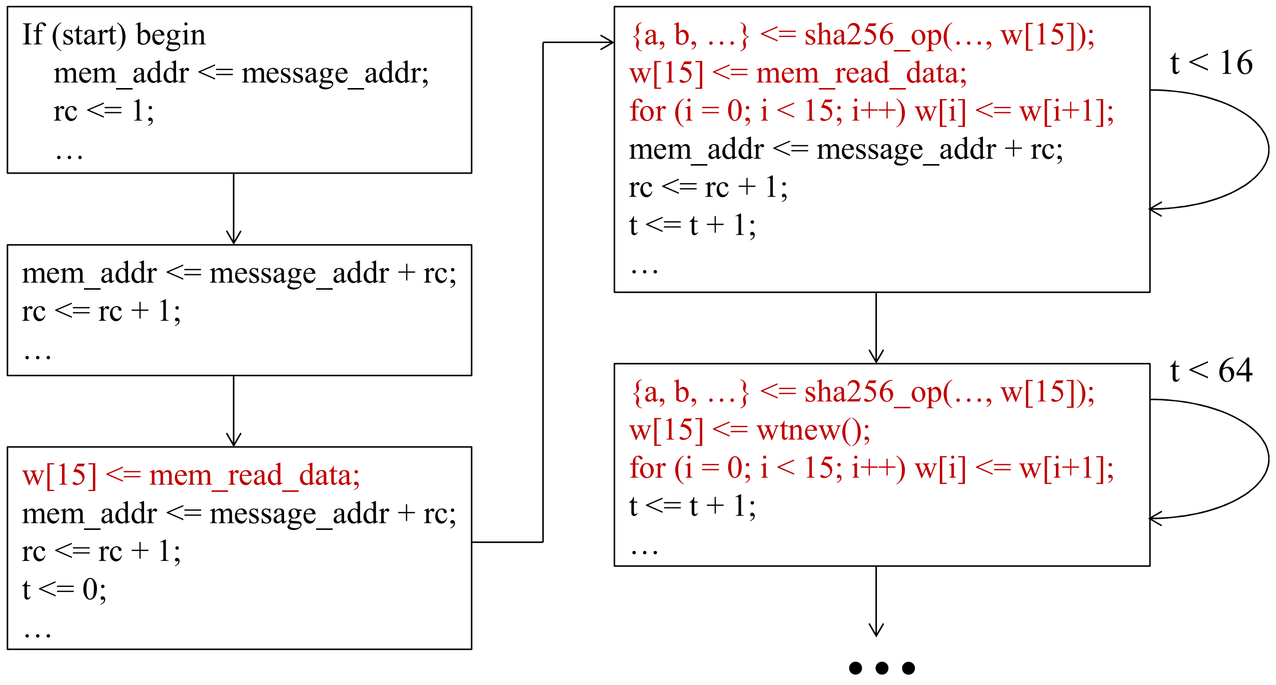


## Critical Path



## Setting Up the Pipeline

- Just an example flow-chart



```
If (start) begin
   mem_addr <= message_addr;
   rc <= 1;
   …
```

```
mem_addr <= message_addr + rc;
rc <= rc + 1;
…
```

```
w[15] <= mem_read_data;
mem_addr <= message_addr + rc;
rc <= rc + 1;
t <= 0;
…
```

```
{a, b, …} <= sha256_op(…, w[15]);
w[15] <= mem_read_data;
for (i = 0; i < 15; i++) w[i] <= w[i+1];
mem_addr <= message_addr + rc;
rc <= rc + 1;
t <= t + 1;
…
```

```
{a, b, …} <= sha256_op(…, w[15]);
w[15] <= wtnew();
for (i = 0; i < 15; i++) w[i] <= w[i+1];
t <= t + 1;
…
```

## Implementing Parallelism

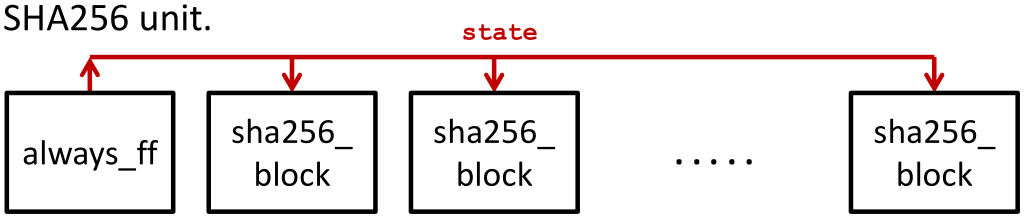
- Can implement "vectorization" like this (effectively doing SIMD execution like a GPU).

```
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
```

## More Optimizations

- Critical path should go through "sha256_op". Can make reduce critical path by pre-computing some parts of the sha256 logic.
  - Pre-compute h + k. We know h[t+1] is g, and we know k[t+1] is a constant.
  - Pre-compute h + k + w. Harder b/c we would need to pre-compute "w" an additional cycle earlier.  Need to start the memory read pipeline an additional cycle earlier.

- Can use multiple "always_ff" blocks.

- Should only try these optimizations once you have done all the other ones as they provide greater improvements.

## More Aggressive Pipelining

```
function logic [255:0] sha256_op(input logic [31:0] a, b, c, d, e, f, g, h, w, k);
    logic [31:0] S1, S0, ch, maj, t1, t2; // internal signals
begin
    S1 = rightrotate(e, 6) ^ rightrotate(e, 11) ^ rightrotate(e, 25);
    ch = (e & f) ^ ((~e) & g);
    t1 = ch + S1 + h + k + w;
    S0 = rightrotate(a, 2) ^ rightrotate(a, 13) ^ rightrotate(a, 22);
    maj = (a & b) ^ (a & c) ^ (b & c);
    t2 = maj + S0;
    sha256_op = {t1 + t2, a, b, c, d + t1, e, f, g};
end
endfunction
```

next "h"

Next "h" and "k" are are already known.
Requires little change in code.

## More Aggressive Pipelining

```
function logic [255:0] sha256_op(input logic [31:0] a, b, c, d, e, f, g, h, w, k);
    logic [31:0] S1, S0, ch, maj, t1, t2; // internal signals
begin
    S1 = rightrotate(e, 6) ^ rightrotate(e, 11) ^ rightrotate(e, 25);
    ch = (e & f) ^ ((~e) & g);
    t1 = ch + S1 + h + k + w;
    S0 = rightrotate(a, 2) ^ rightrotate(a, 13) ^ rightrotate(a, 22);
    maj = (a & b) ^ (a & c) ^ (b & c);
    t2 = maj + S0;
    sha256_op = {t1 + t2, a, b, c, d + t1, e, f, g};
end
endfunction
```

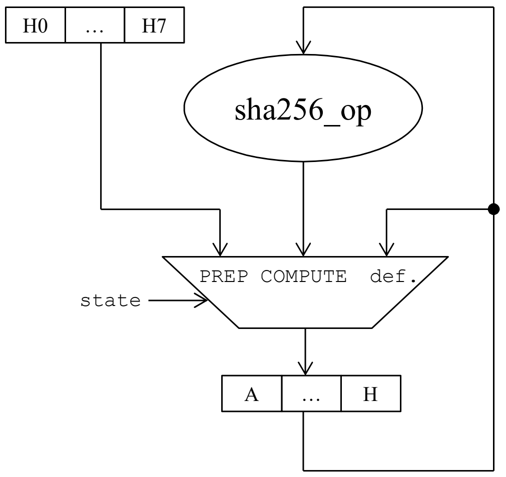
Next "h" and "k" are are already known.
Requires little change in code.

But need to pre-compute "w" one more cycle ahead then previous w-pipeline. Requires more substantial changes to the code.

## Use Separate always_ff blocks

- Can use separate "always_ff" blocks. e.g.

```
// always_ff block for updating only A ... H
always_ff @(posedge clk)
begin // no need for reset
    case (state)
    PREP:
      {A, B, ..., H} <= {H0, H1, ..., H7};
    COMPUTE:
      {A, B, ..., H} <= sha256_op(A, B, ..., w[15]);
    endcase // default A ... H remain the same
end
```



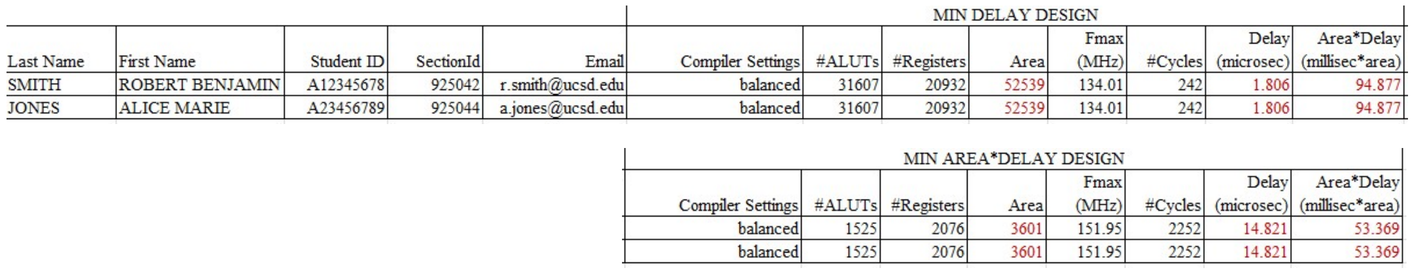
## Final Project Submission

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

- **NEW: Final-Report.docx** (already discussed. "Final-Report-Template.docx" as a template.)
- finalsummary.xlsx
- 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



| | | | | | MIN DELAY DESIGN | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 |

| 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

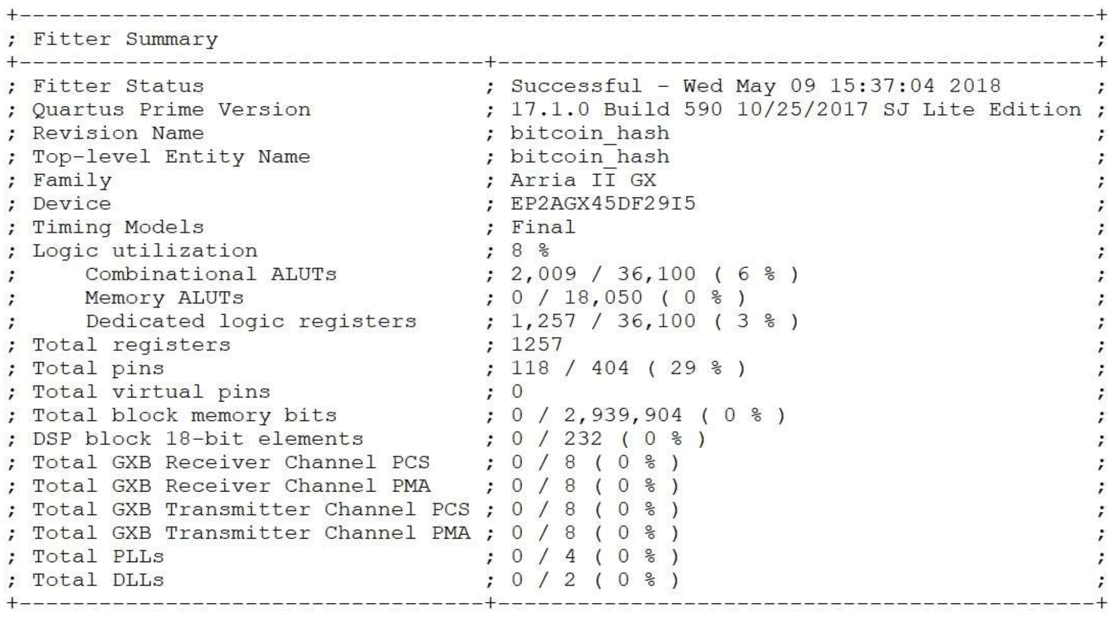
- Copy of the ModelSim simulation results.
- Just need simulation results for **tb_bitcoin_hash.sv**.
- After you run the “run –all” command, you can save your transcript by going to the “File” menu and clicking on “save transcript as”.
- Transcript file will contain the history of all commands used 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

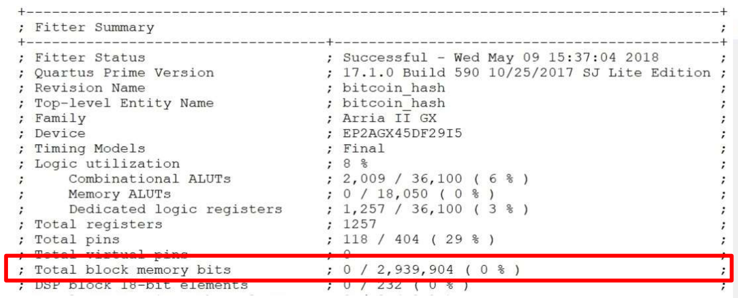
- Copy of the <u>**fitter reports**</u> (not the flow report) with area numbers.
- Make sure to use **Arria II GX EP2AGX45DF29I5** device
- **IMPORTANT**: Make sure **Total block memory bits is 0**.



```
+-------------------------------------------------------------------------------------+
; 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 % )                                  ;
+------------------------------------+------------------------------------------------+
```

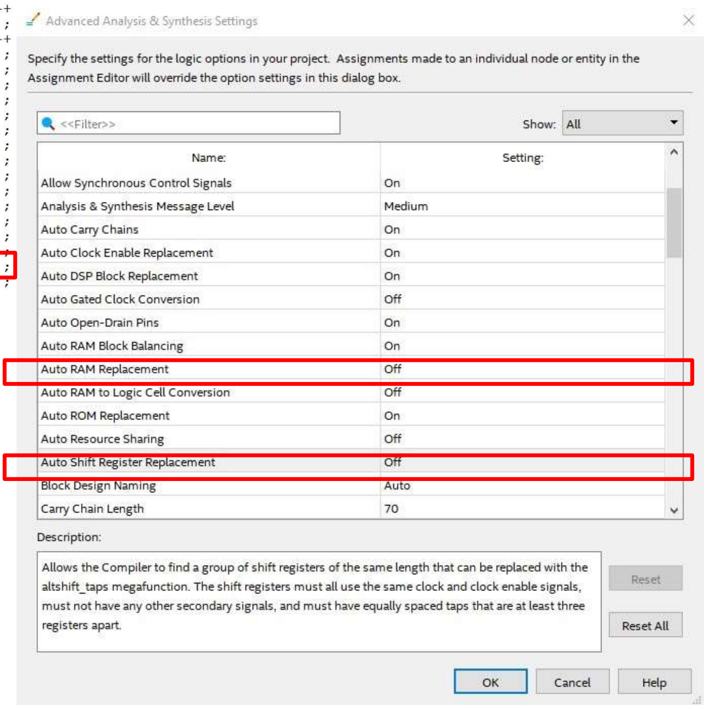
## No Block Memory Bits

- 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 % )                                ;
```

- 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”



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.

| 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.

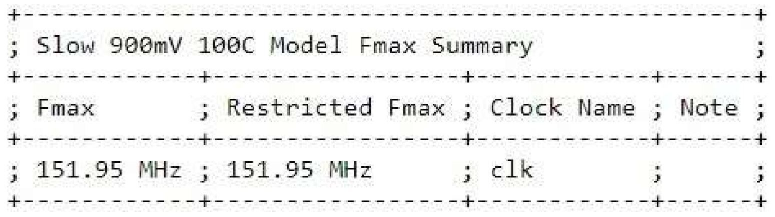
## No Inferred Megafunctions/Latches

- 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 ***only*** clock.



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

## Positive/Negative Impact Questions

In "Final-Report-Template.docx," there is a section on "Positive/Negative Impact" of Bitcoin, cryptocurrencies, and blockchain technologies.

In 3-4 sentences for each question, address the following:

1. Explain how Bitcoin or cryptocurrency technologies in general could have a positive global impact (economical, environmental, societal, public policy)?

2. Explain how Bitcoin or cryptocurrency technologies in general could have a negative global impact (economical, environmental, societal, public policy)?

3. Blockchain as a secure record-keeping mechanism is emerging as a a separate technology from Bitcoin or cryptocurrencies. Explain how blockchains as a separate technology could have a positive global impact (economical, environmental, societal, public policy)?

4. Explain how blockchains as a separate technology could have a negative global impact (economical, environmental, societal, public policy)?

## Due Date Changed

- To allow time for the final project report, final project due date has been extended to

  Wednesday 3/20 before 11:59 pm

- Absolutely no extension!