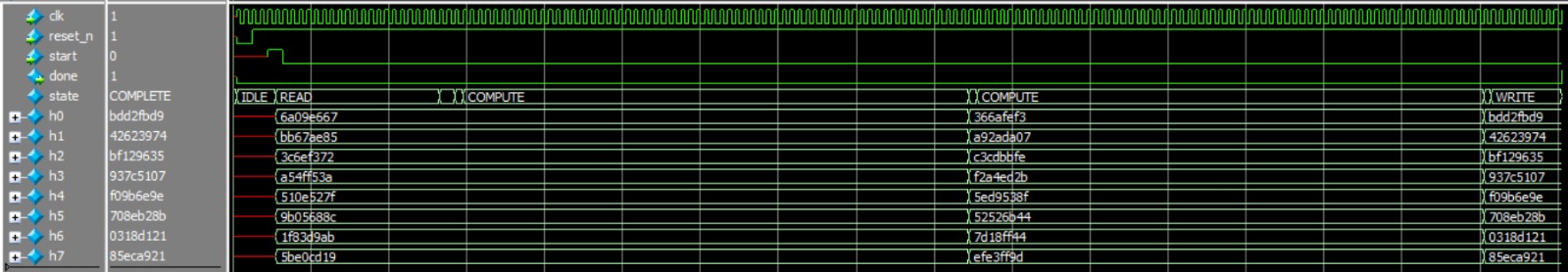
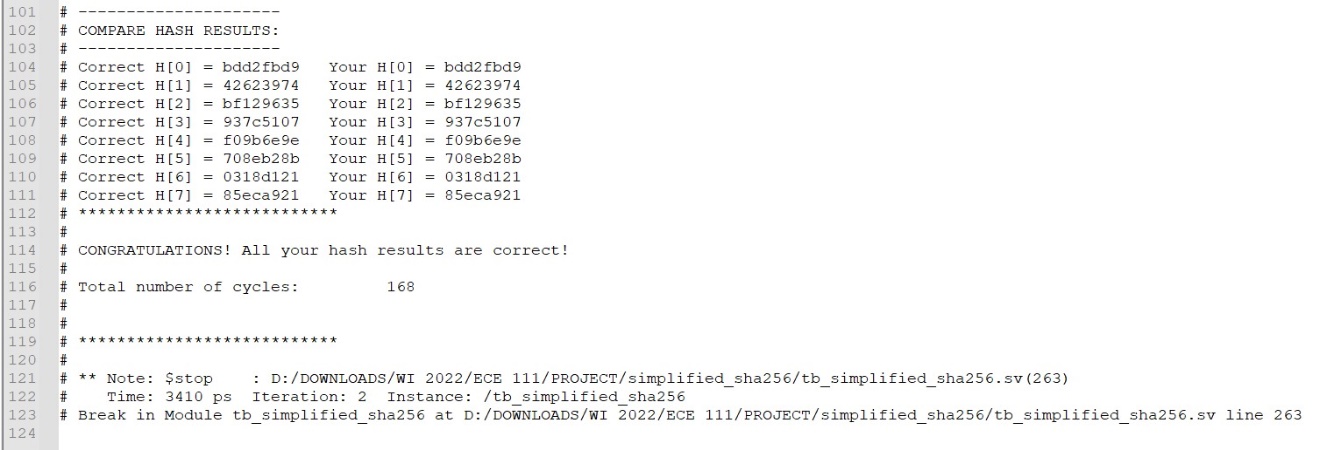
PART-1 SHA-256

WAVEFORM

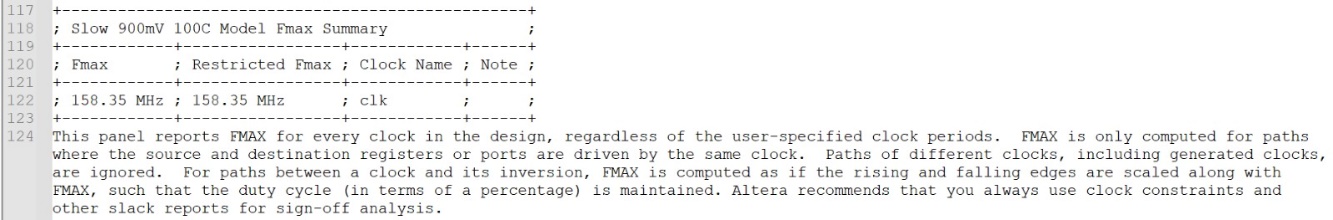


The initial hash values are updated and the new hash values from the SHA-256 module can be seen in the simulation window. At the end of the simulation, done signal is high (after the WRITE state).

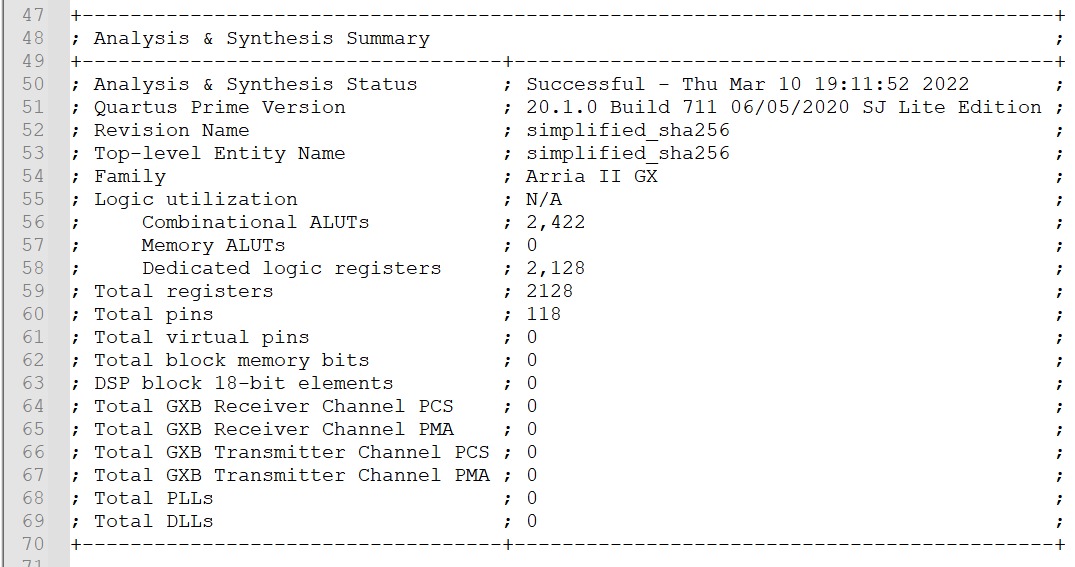
Total Cycles required and Comparison of Hash Values



Maximum Frequency of operation.



Area utilized



**Part- 1** of the project is generating new hash value using K constant and original 32bit word in the messages. We have generalized the number of blocks calculation for different length of messages, although given inputs has fixed word lengths of 640. Every input message is multiple of 512 bits and hence they are divided into blocks depending on padding.

**Summary:**

Firstly, the RTL includes four functions; one is *determine\_num\_blocks, sha256\_op, wt and rightrotate*

The *determine\_num\_blocks* calculate the number of Mn. The rightrotate function is used to calculate S0 and S1 in the sha256\_op and wt, which gets the required Hash value. The sha256\_result signal holds the final value and are assigned combinationally.

Now the simplified\_SHA RTL is modelled as 6 state FSM design i.e.

1. IDLE 🡪 When start =1, all initial hash values [H0…H7] and [a…h] are equal. current address value is set to message address (read location). Offset increments each cycle util all 20 message words are read (write enable is set to 0). Now, FSM will move to READ state. (Note: at the end done is high if the state is IDLE).

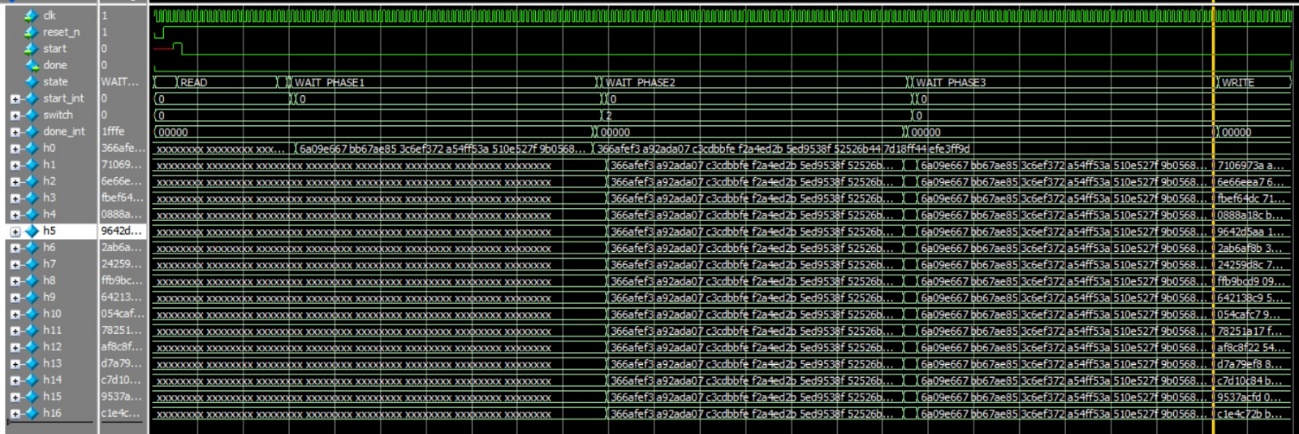
1. READ 🡪 In Read state, message words are fetched form the memory until all words are read. Offset value is incremented when a new word is read from the memory. FSM moves to READ\_DELAY state when all message words have been read from the memory.
2. READ\_DELAY 🡪 In the READ\_DELAY state, 21ST word in the message is assigned 1 (at MSB location) as a partition in actual message and 0s are padded. message length is also padded (31st and 32nd words) and the state is changed to BLOCK state. Now, we have both M**0** and M**1** ready for SHA256 calculation.
3. BLOCK🡪 We have optimized the number of **w[m]** used to 16 registers. Each 512-bit message block is loaded into **w[m]**. FSM state now changes to COMPUTE as we have the data to be hashed ready. FSM reaches BLOCK state after 64 iterations in COMPUTE state are completed. We check if all the 512-bit message blocks have been processed or any remain. If all blocks are processed, then the FSM moved to WRITE state. Otherwise, **w[m]** registers are loaded with the next 512-bit message block; FSM moves again to COMPUTE state.
4. COMPUTE 🡪 In COMPUTE state, **sha256\_op** function is run for 64 iterations. The **w[15]** block is loaded with the result from the **wt** function. Data in each **w[m]** register is left shifted to **w[m-1]** location. This allows us to use fewer registers to save area. Tstep variable tracks the number of iterations. After 64 iterations are complete: new message digest is calculated by adding the initial hash values [**h0-h7**] used in current iteration, and the **a-h** values calculated after 64 iterations of **sha256\_op** function. FSM now moves back to BLOCK state.
5. WRITE🡪 Here the sha256 result calculated after processing all 512-bit message blocks is loaded back into the memory. Offset variable is incremented after each memory write. Once offset reaches value of 8 (i.e., when all computed hash values are loaded into memory), FSM moves back to IDLE state. The module is now ready to process the next message block from memory.

**FSM TRANSITION DIAGRAM FROM QUARTUS.**



PART - 2 BITCOIN HASH

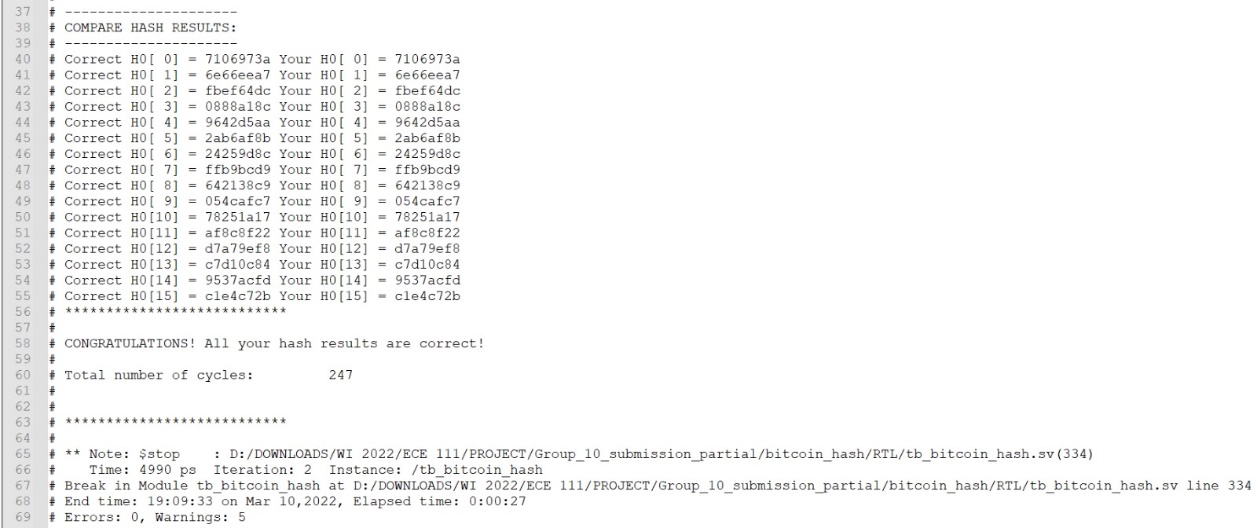
Waveform:



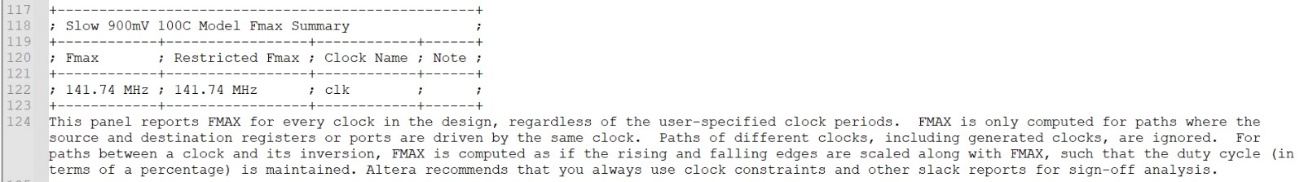
The hexadecimal output values of Bitcoin module and the correct hash values are written into the memory when the

done signal is 1 at end of the Write state.

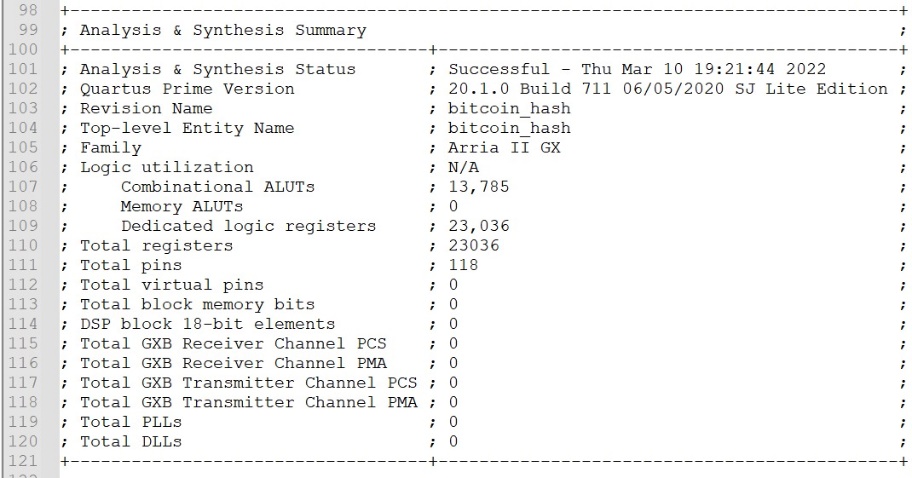
Total Cycles required and Comparison of Hash Values



Maximum Frequency of operation.



Area utilized



**Part-2** – The simplified\_sha256 module designed in Part-1 is re-used after doing the following modifications:

1. READ, READ\_DELAY, WRITE states from FSM are removed
2. COMPLETE state has been added – FSM reaches this state after 64 iterations of **sha256\_op**
3. New ports **in, switch** have been added to support the below feature:  
   When **switch=0**, initial hash values loaded are same as that in Part-1   
   When **switch=1**, hash values present on the **in** port are loaded into the design

Now the bitcoin\_hash RTL is modelled as 10 state FSM design i.e.

1. IDLE 🡪 When start =1, current address value is set to message address (read location). Offset, Start\_int, switch is set to 0 AND each cycle until all 20 message words are read (write enable is set to 0). Now, FSM will move to READ state. (Note: at the end done is high if the state is IDLE).
2. READ 🡪 In read state, the read location memory is written to message until offset 20 (), and offset is Incremented and state is transitioned to READ\_DELAY.
3. READ\_DELAY 🡪 In Read Delay state, we are loading from 21st to 32nd word of message similar to part 1.

FSM moves to Phase 1 State.

1. PHASE\_1 🡪 Here the SHA\_256 operation is done on 1ST 512 BITS of message block. Start signal is given to 1 SHA instance. FSM State moves to WAIT\_PHASE1
2. WAIT\_PHASE1 🡪FSM waits for SHA instance to completes its operations. Once its complete, **w0 -w15** are loaded with remaining message words, nonces, Padding and message size. FSM moves to PHASE\_2.
3. PHASE\_2 🡪 Here 16 SHA instances are triggered parallelly, and Hash value generated from PHASE\_1 is provided as seed for PHASE\_2 COMPUTATION. FSM moves to WAIT\_PHASE\_2 state.
4. WAIT\_PHASE\_2 🡪 FSM waits for 16 SHA instances to completes their operations, once its complete **w0 – w15** are loaded with the Hash Values generated from phase\_2 computation. FSM moves to Phase\_3 State.
5. PHASE\_3 🡪 Here 16 SHA instances triggered parallelly, and initial Hash values are same as Phase\_1. FSM moves to WAIT\_PHASE\_3 state.
6. WAIT\_PHASE\_3 🡪 FSM waits for 16 SHA instances to completes their operations, once its done FSM moves to WRITE state.
7. WRITE 🡪 The HASH values computed from Phase\_3 are loaded to Memory. Offset variable tracks number of memory writes. Once done FSM moves to Complete State.
8. COMPLETE 🡪 done is set to High to Indicate completion of all Steps. FSM moves to IDLE state and is ready to process Next Message.

**FSM TRANSITION DIAGRAM FROM QUARTUS.**

