**CMPE 315: Principles of VLSI Design**

**Project Cover Page**

**Project Title : Cache Design [ Submission Part 1]**

**Name : Marinna Ricketts-Uy & Ben Thompson**

**Section : 01**

**Date Submitted : April 22, 2018**

**CONTENTS Page#**

**----------------------------------------------------------------------------------------------------------**

**No table of contents entries found.**

**Introduction**

The purpose of this project is to design, implement, and simulate in VHDL, and a layout for a cache block.

**Design Process**

**Chip Design**

**Block Diagram**

Figure x shows the block diagram for the overall design of the chip.

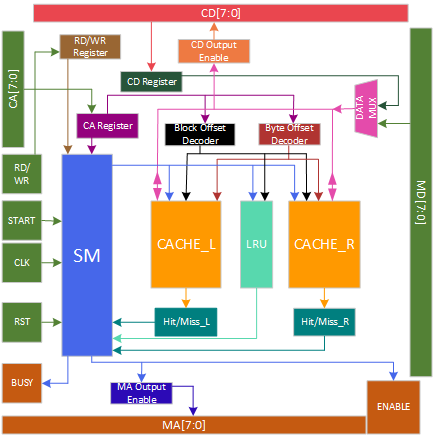


Table x details the inputs and outputs for the chip entity.

|  |  |
| --- | --- |
| Inputs | Outputs |
| CPU Address | Busy |
| CPU Data | Memory Enable |
| CPU RD/WR Signal | Memory Address |
| Start |  |
| CLK |  |
| Reset |  |
| Memory Data |  |
| Vdd |  |
| Gnd |  |

**Cache Design**

The cache is designed using the two-way associative cache design. The cache was designed using a hierarchical design. There is a 1-bit cache, 8-bit cache, cache block, and a full cache.

Figure x shows the schematic design for a 1-bit cache cell.

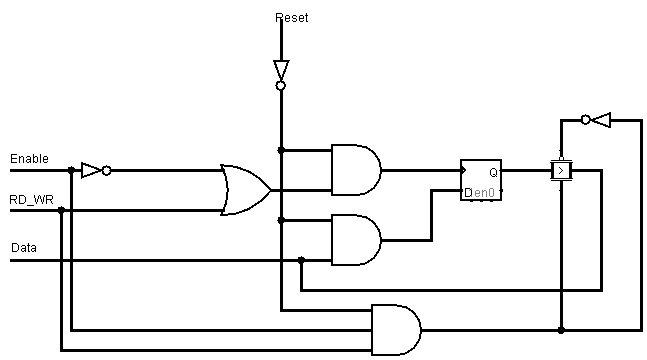


Table x details the input and output signals for each cache.

|  |  |
| --- | --- |
| Inputs | Outputs |
| Reset | Valid Bit Data (vb\_data) |
| Cache Enable Signal (Cache\_en) | Tag Data (tag\_data) |
| Hit/Miss Enable Signal (HM\_en) |  |
| Byte Select (CS) |  |
| Block Select (RS) |  |
| ValidBit/Tag RD/WR Signal (vb\_tag\_rd\_wr) |  |
| RD/WR signal (rd\_wr) |  |
| Valid Bit Data (vb\_data) |  |
| Tag Data (tag\_data) |  |

On a reset, the contents of each cache are zeroed out. The cache enable signal is received from the state machine. The left cache is enabled when this signal is low and the right cache is enabled when the signal goes high. The hit/miss enable signal is used to signify whether the state machine is currently determining whether there is a hit or a miss. When this signal is high, this signifies that both caches must be enabled to check for a hit or miss in both caches. The byte select value is received from the byte offset decoder and is used to select which byte to read or write from. This signal is active low. The block select value is received from the block offset decoder and is used to select which block to read or write from. This signal is active low. The valid and tag rd/wr signal is used to signify whether to read or write to the valid and tag. This signal is being received from the state machine. The rd/wr signal is used to signify whether to read or write to the appropriate byte. The valid bit data and tag data are inputs from the state machine and outputs to the hit/miss entity. The state machine will write this data to the cache when it is performing a read miss operation.

**State Machine Design**

Figure x shows the diagram for the state machine.

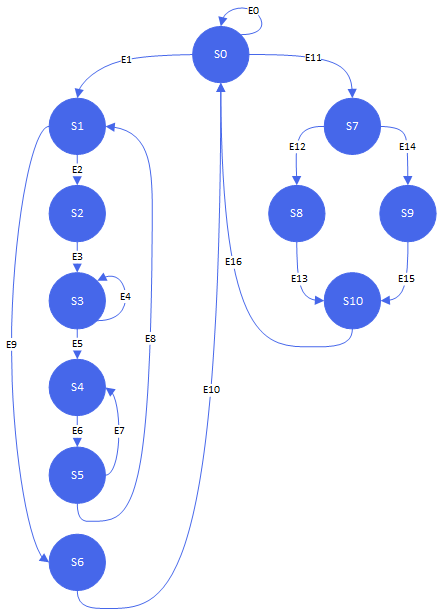


Table x details the inputs and outputs for the state machine

|  |  |
| --- | --- |
| Inputs | Outputs |
| CLK | Busy |
| Reset | CPU Data Output Enable |
| Start | Memory Address Output Enable |
| Hit/Miss Left | Data Mux Select |
| Hit/Miss Right | Valid Bit / Tag RD/WR |
| RD/WR | Cache RD/WR |
| CPU Address | Hit/Miss Enable Signal |
| LRU | Cache Enable |
|  | LRU RD/WR |
|  | Address |
|  | VB Data Left |
|  | VB Data Right |
|  | Tag Data Left |
|  | Tag Data Right |

When the reset input signal goes high, the state machine starts in the initial state 0. For any of the states, if the reset input signal goes high, the state machine will move to state 0. On edge 0, the state machine will remain in state 0 if start input signal remains low. From state 0, if the start signal goes high and the RD/WR input signal signifies a read operation, the state machine will move to state 1, otherwise the state machine will move to state 7 for a write operation. For edges 1 and 11, the busy output signal will be set high and the Hit/Miss output enable signal will be set high indicating that the state machine is currently doing a hit/miss operation.

**Read Operation**

For a read operation, the state machine will wait to receive the outcome of the hit/miss operation. If the hit/miss outcome for the left or right result in read hit, the state machine will move to state 6 on edge 9. During a read hit in state 6, the CPU data output enable signal will be set high. The LRU\_RD\_WR signal is also set low to indicate a write to the LRU. During edge 9, the busy output signal will be set low.

If the hit/miss operation during state 1 for both the left and right result in a read miss, the state machine will move to state 2. During state 2, the memory address is sent along with the memory output enable. The memory address that is sent is based on an address selector. For state 2, the last two bits in the memory address will be the first byte in the block (00).

Figure x shows the schematic for the address selector.

**Write Operation**

For a write operation, the state machine will wait to receive the outcome of the hit/miss operation. If the hit/miss operation for the left or right cache results in a hit, the state machine will move to state 8. During edge 12, the Cache RD/WR output signal will be set to 0 to indicate a write operation. In state 8, the LRU RD/WR output signal will be set to 0 to indicate a write to the LRU.

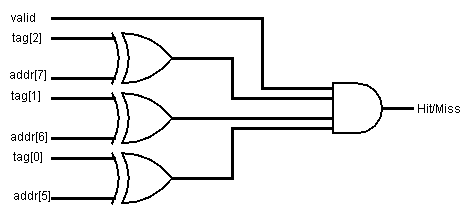
If the write operation results in a miss, the state machine will move to state 9. During a write miss, there is no operations to be made so the state machine will move from state 9 to state 10. During edge 15, the busy output signal will be set low.

**Hit/Miss Design**

Table x details the inputs and outputs for the hit/miss entity.

|  |  |
| --- | --- |
| Inputs | Outputs |
| Valid Data (valid) | Hit\_Miss (h\_m) |
| Tag Data [2:0] (tag) |  |
| CPU Address [7:5] (addr) |  |

Figure x shows the schematic for the hit/miss entity

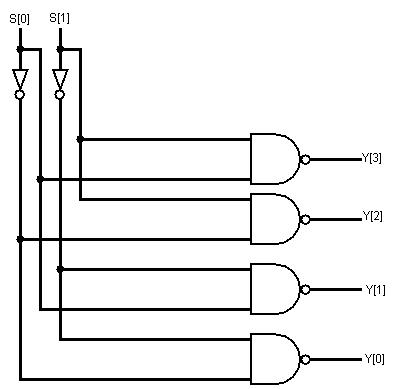


The valid and tag data input for the hit/miss entity is an output from the cache. The CPU address bits and the tag data bits are compared to see whether they match. The output of each xnor gate is then an input for an and gate with the valid bit. The hit/miss entity results in a hit when the valid bit is set to 1 and each CPU address bit matches the tag bits. There is a hit/miss entity for the left and the right cache.

**Decoder Design**

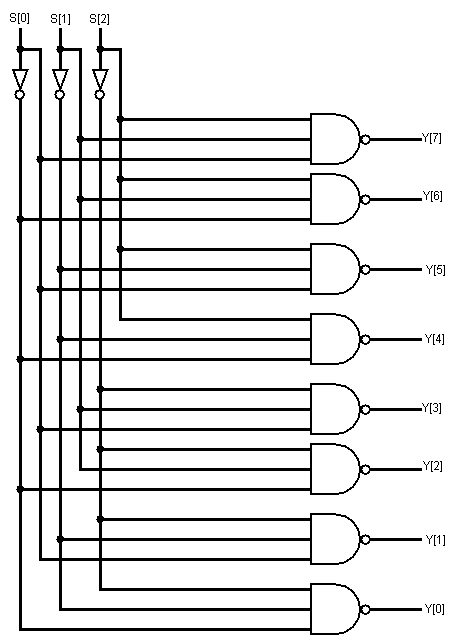
**Byte Offset Decoder**

Figure x shows the schematic design for the 2 to 4 decoder. This decoder is active low and is used for choosing the byte offset.

****

**Block Offset Decoder**

Figure x shows the schematic design for the 3 to 8 decoder. This decoder is active low and is used for choosing the block offset.



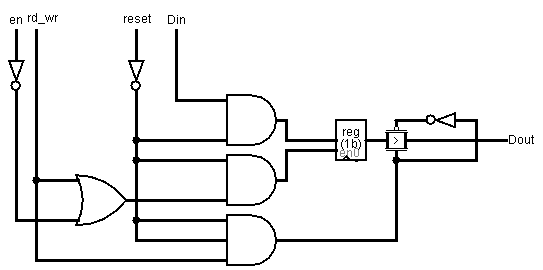
**LRU Design**

For the purpose of block replacement, there is a LRU entity that is used to determine the least recently used block in either of the caches. The LRU value for a block is written to when the state machine is at the end of a read hit/miss operation or a write hit operation. The state machine will read from the LRU entity during a read miss operation to see which cache block to write the new data to.

Table x details the inputs and outputs for the LRU entity.

|  |  |
| --- | --- |
| Inputs | Outputs |
| Enable (en) | Data Out (dout) |
| RD/WR (rd\_wr) |  |
| Reset (reset) |  |
| Data In (din) |  |

Figure x shows the schematic design for the 1 bit LRU entity.



The rd\_wr input signal is a signal from the state machine. When the state machine reads from the LRU, the transmission gate is turned on and the data from the latch will be the output.

Figure x shows the schematic design for the 8 bit LRU entity.

**Data Mux Design**

The data mux is used to select whether to be using memory data or cpu data. The data mux writes out data when

**Entity Architecture Pairs**

|  |  |
| --- | --- |
| **Entity** | **Using the Entity** |
| D\_latch |  |
| reg8 |  |
| oe8 |  |
| mux2to1\_8 |  |
| addr\_sel |  |
|  |  |
|  |  |

**Test Bench & Simulations**

Once the code for the state machine was written, a simple test bench was created to ensure that the state machine could move through the states for each of the four operations. The code for the state machine test bench and the relevant input and output files are included in Appendix X.

The design of the cache was verified with two test benched provided by Dr. Patel. The first test bench was used to confirm that the design performed each of the desired operations with the correct timing. The second test bench performed a more thorough verification by operating on every block of the cache. Figure X shows the timing of the read miss operation. This waveform has been checked against the sample output and found to be correct. 

Figure X shows the timing of the write hit, read hit, and write miss operations. This waveform was also checked against the provided sample and found to be correct. 

After the timing of the design was verified the second, more thorough test bench was applied to the design. The output ncsim.log was compared to the ncsim.log file proved by Dr. Patel. The comparison was made using the diff command. After the diff command was run the files were found to contain no differences.

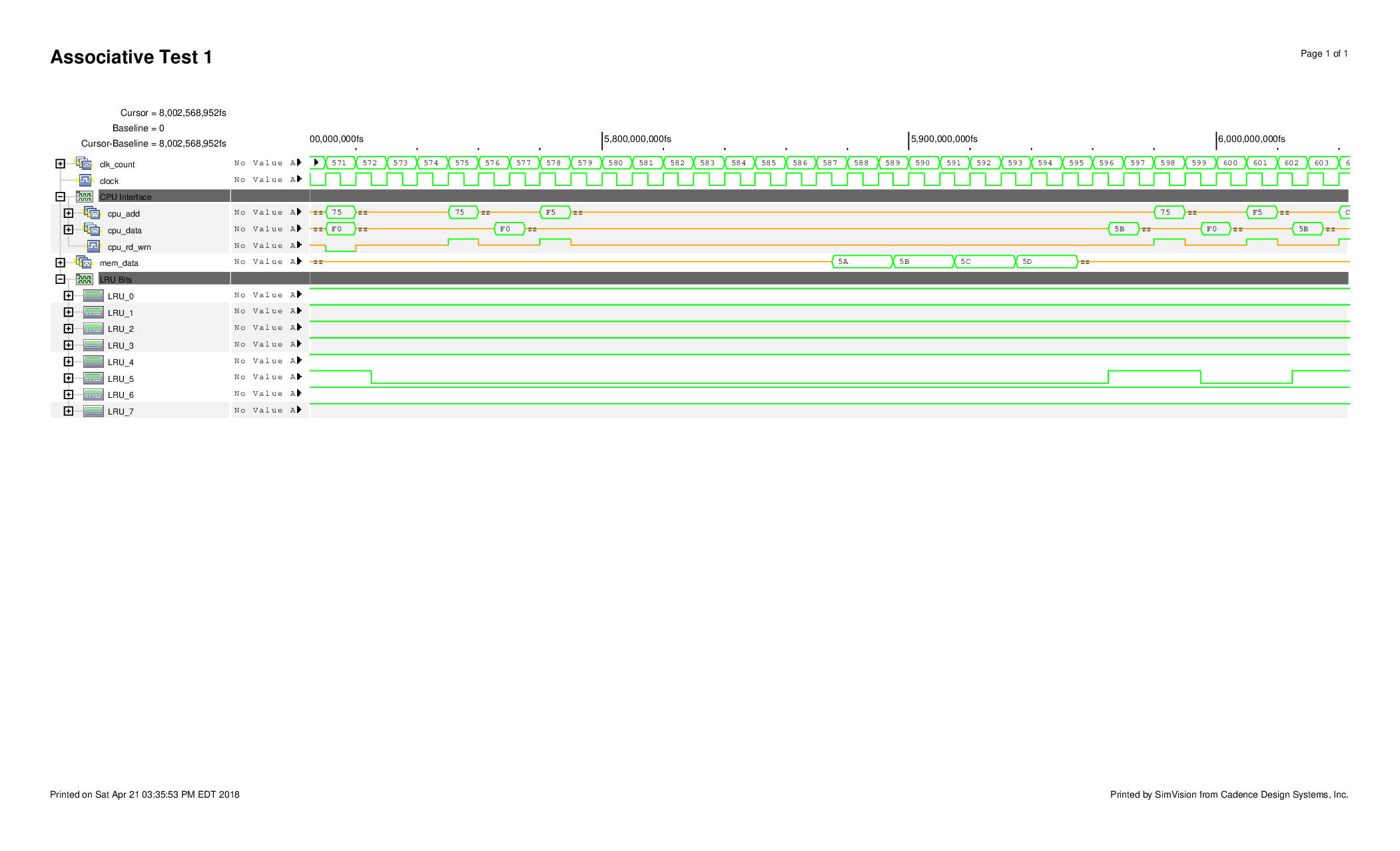
The test benches provided by Dr. Patel have no verification cases for the two-way associative cache. In order to confirm that the associative parts of the design are functional the second test bench input file was modified to contain more verification cases. The operations already in the input file remain unchanged. A summary of the operations and addresses used are shown in Appendix X. The waveforms for this verification step are shown in Appendix X.

The first modification was to repeat the same operations given in the original file for additional addresses. This allowed both sides of the cache to be filled and fully test the four operations on each side of the cache. A summary of the operations and addresses added are shown in Appendix X. The waveforms for this verification step are shown in Appendix X.

The next addition was to read the original address back to confirm both sides of the cache were used. After reading back from both sides of the cache the LRU bit for every block would be set to indicate that the right side of the cache is the least recently used. The addresses used for the read are shown in Table X. The waveforms for this verification step are shown in Appendix X.

The next addition to the input file was to switch the LRU for block five by performing a write hit followed by a read hit for the address corresponding to the right side of the cache. This would switch the LRU bit for block five to the left side of the cache. Next a read miss would be performed to verify if the result of the operation was to replace the entry in the left side of the cache. To confirm this, the addresses for the left and right sides were read from to confirm that the entries were correct. A summary of the additions for this verification step are shown in Table X.

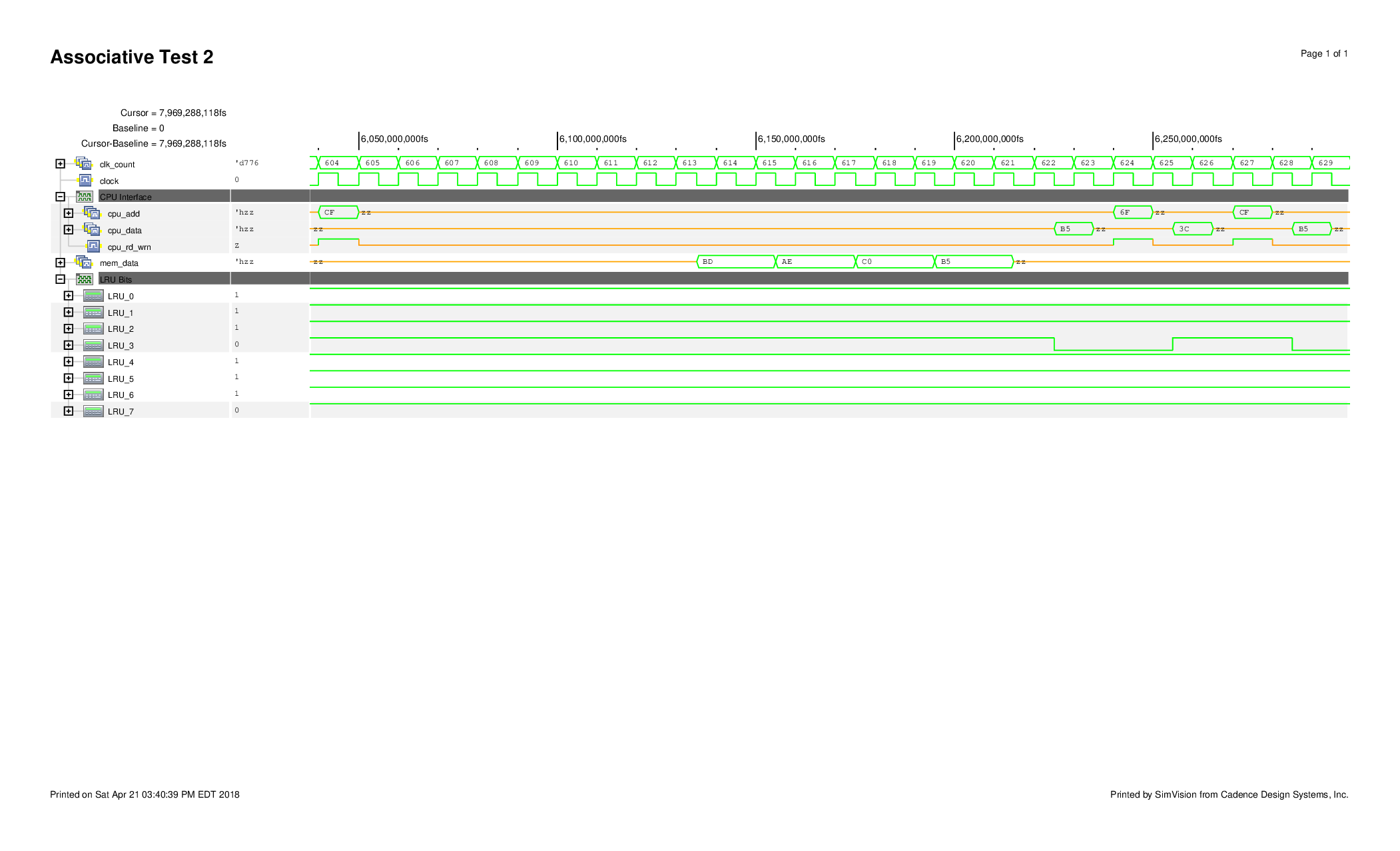
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Operation | Address | Side | Block Number | Byte | Data |
| Write Hit | 75 | R | 5 |  |  |
|  |  |  |  | Byte 01 | F0 |
| Read Hit | 75 | R | 5 |  |  |
|  |  |  |  | Byte 01 | F0 |
| Read Miss | F5 | L | 5 |  |  |
|  |  |  |  | Byte 00 | 5A |
|  |  |  |  | Byte 01 | 5B |
|  |  |  |  | Byte 10 | 5C |
|  |  |  |  | Byte 11 | 5D |
| Read Hit | 75 | R | 5 |  |  |
|  |  |  |  | Byte 01 | F0 |
| Read Hit | F5 | L | 5 |  |  |
|  |  |  |  | Byte 01 | 5B |

The output waveform for this verification step is shown in Figure X.

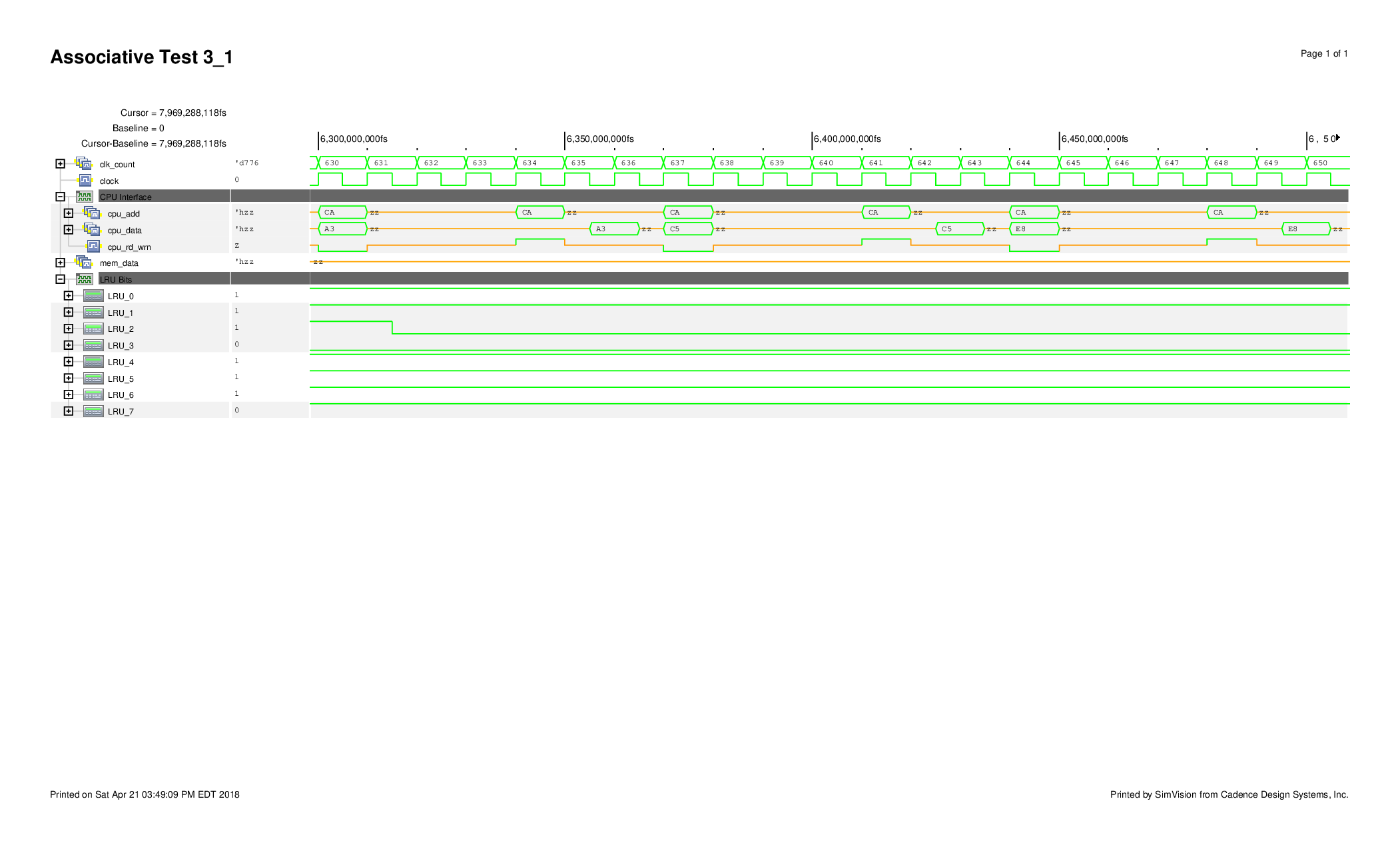
The next verification step was to confirm that a read miss without switching the LRU bit would cause the right side to be replaced. The additions for this step are summarized in Table X.

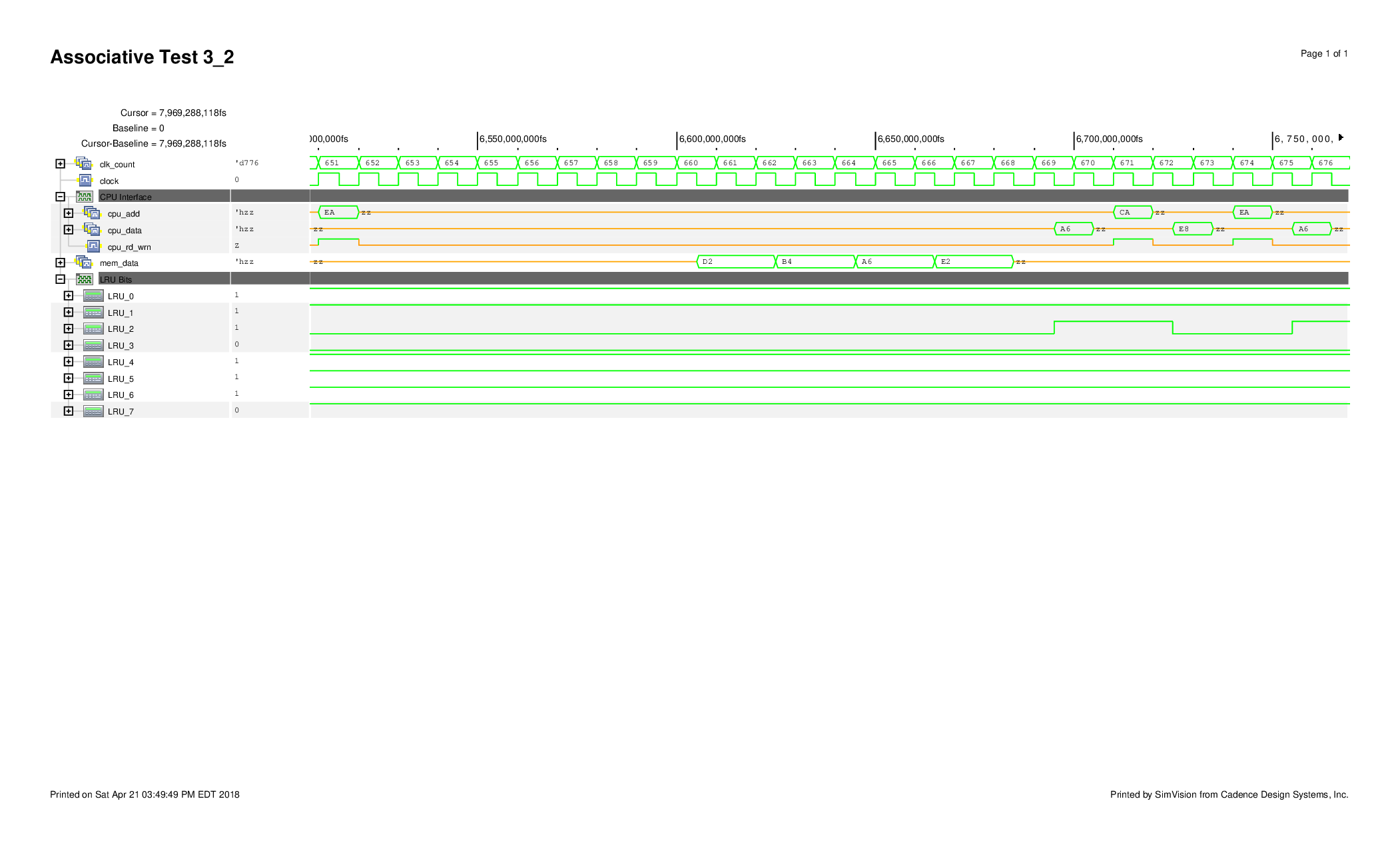
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Operation | Address | Side | Block Number | Byte | Data |
| Read Miss | CF | R | 3 |  |  |
|  |  |  |  | Byte 00 | BD |
|  |  |  |  | Byte 01 | AE |
|  |  |  |  | Byte 10 | C0 |
|  |  |  |  | Byte 11 | B5 |
| Read Hit | 6F | L | 3 |  |  |
|  |  |  |  | Byte 11 | 3D |
| Read Hit | CF | R | 3 |  |  |
|  |  |  |  | Byte 11 | B5 |

The output waveform for this verification step is shown in Figure X.

Another verification step was used to check to see if everything remained functional with a series of repetitive operations on the same side of the cache followed by a read miss. The operations were performed on the right side of the cache so the read miss replaced the data in the left side of the cache. A summary of these additions to the input file are shown in Table X.

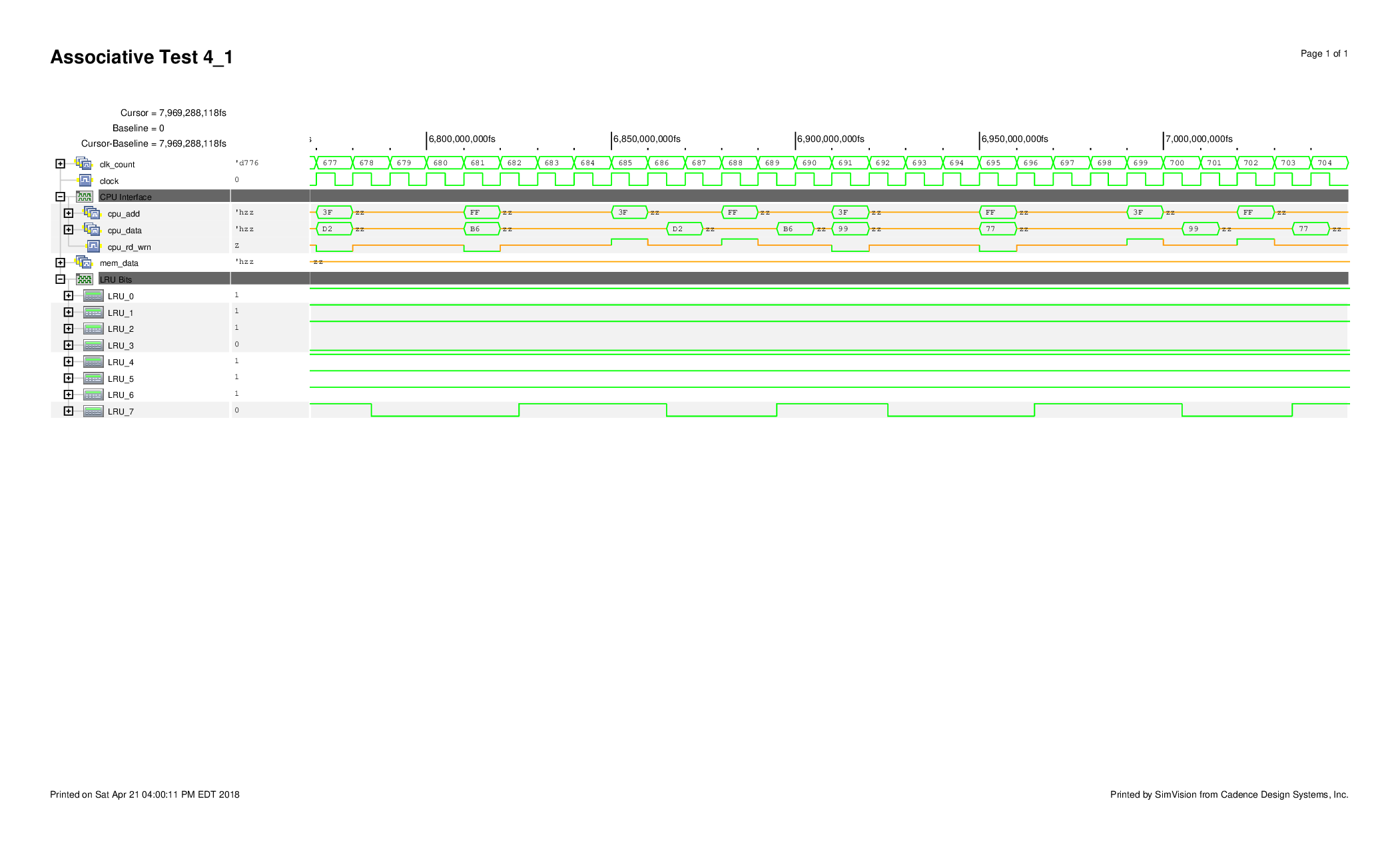
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Operation | Address | Side | Block Number | Byte | Data |
| Write Hit | CA | R | 2 |  |  |
|  |  |  |  | Byte 10 | A3 |
| Read Hit | CA | R | 2 |  |  |
|  |  |  |  | Byte 10 | A3 |
| Write Hit | CA | R | 2 |  |  |
|  |  |  |  | Byte 10 | C5 |
| Read Hit | CA | R | 2 |  |  |
|  |  |  |  | Byte 10 | C5 |
| Write Hit | CA | R | 2 |  |  |
|  |  |  |  | Byte 10 | E8 |
| Read Hit | CA | R | 2 |  |  |
|  |  |  |  | Byte 10 | E8 |
| Read Miss | EA | L | 2 |  |  |
|  |  |  |  | Byte 00 | D2 |
|  |  |  |  | Byte 01 | B4 |
|  |  |  |  | Byte 10 | A6 |
|  |  |  |  | Byte 11 | E2 |
| Read Hit | CA | R | 2 |  |  |
|  |  |  |  | Byte 10 | E8 |
| Read Hit | EA | L | 2 |  |  |
|  |  |  |  | Byte 10 | A6 |

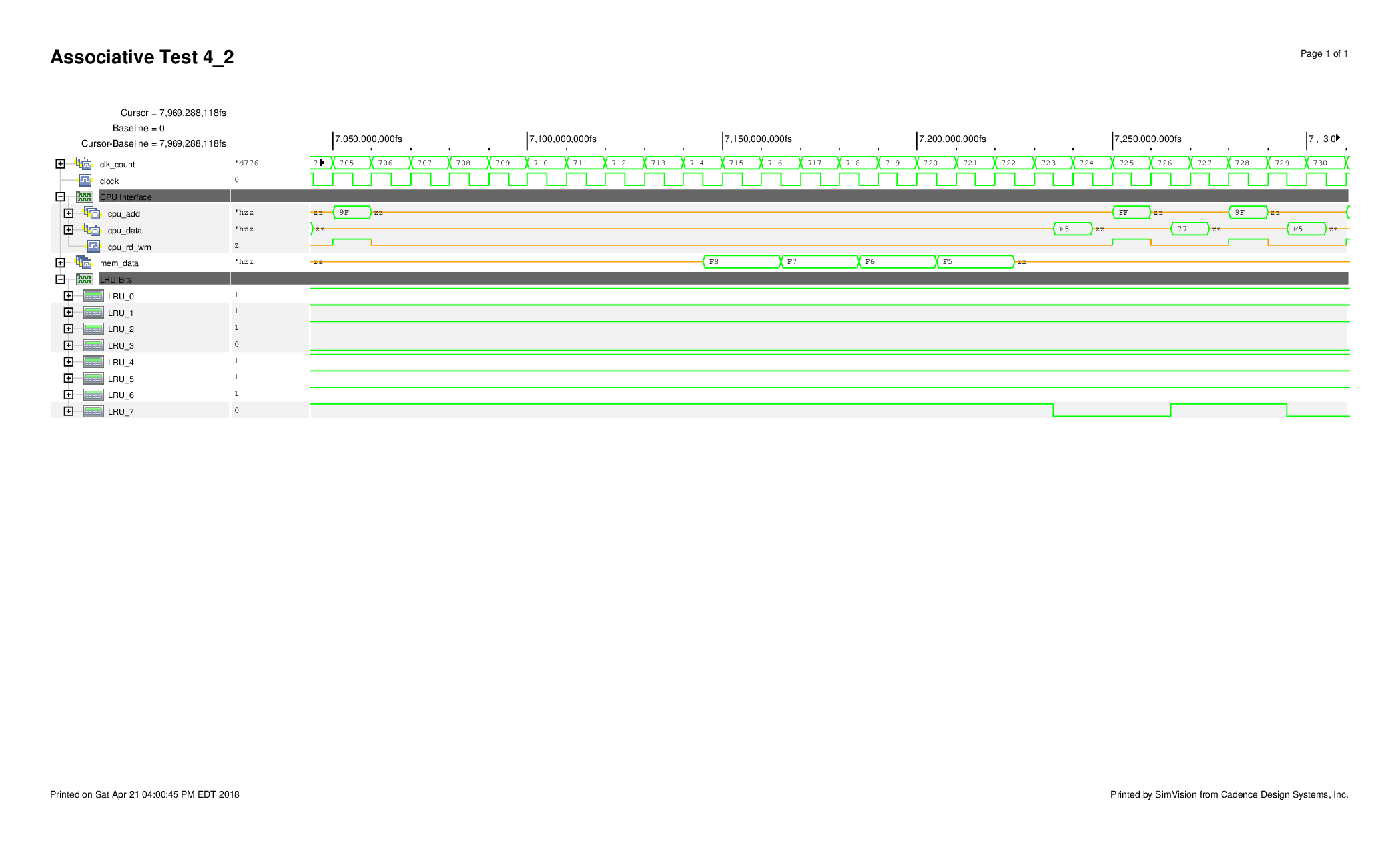
The output waveform for the write hit, read hit sequence is shown in Figure X and the output waveform for the read miss, read hit, read hit sequence is shown in Figure X.

Similarly a verification step was added to confirm the functionality when the LRU bit is toggled for a specific block then followed by a read miss. These additions are summarized in Table X.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Operation | Address | Side | Block Number | Byte | Data |
| Write Hit | 3F | R | 7 |  |  |
|  |  |  |  | Byte 11 | D2 |
| Write Hit | FF | L | 7 |  |  |
|  |  |  |  | Byte 11 | B6 |
| Read Hit | 3F | R | 7 |  |  |
|  |  |  |  | Byte 11 | D2 |
| Read Hit | FF | L | 7 |  |  |
|  |  |  |  | Byte 11 | B6 |
| Write Hit | 3F | R | 7 |  |  |
|  |  |  |  | Byte 11 | 99 |
| Write Hit | FF | L | 7 |  |  |
|  |  |  |  | Byte 11 | 77 |
| Read Hit | 3F | R | 7 |  |  |
|  |  |  |  | Byte 11 | 99 |
| Read Hit | FF | L | 7 |  |  |
|  |  |  |  | Byte 11 | 77 |
| Read Miss | 9F | R | 7 |  |  |
|  |  |  |  | Byte 00 | F8 |
|  |  |  |  | Byte 01 | F7 |
|  |  |  |  | Byte 10 | F6 |
|  |  |  |  | Byte 11 | F5 |
| Read Hit | FF | L | 7 |  |  |
|  |  |  |  | Byte 11 | 77 |
| Read Hit | 9F | R | 7 |  |  |
|  |  |  |  | Byte 11 | F5 |

The output waveform for the LRU flipping is shown in Figure X and the read miss, and read hit confirmation is shown in Figure X.





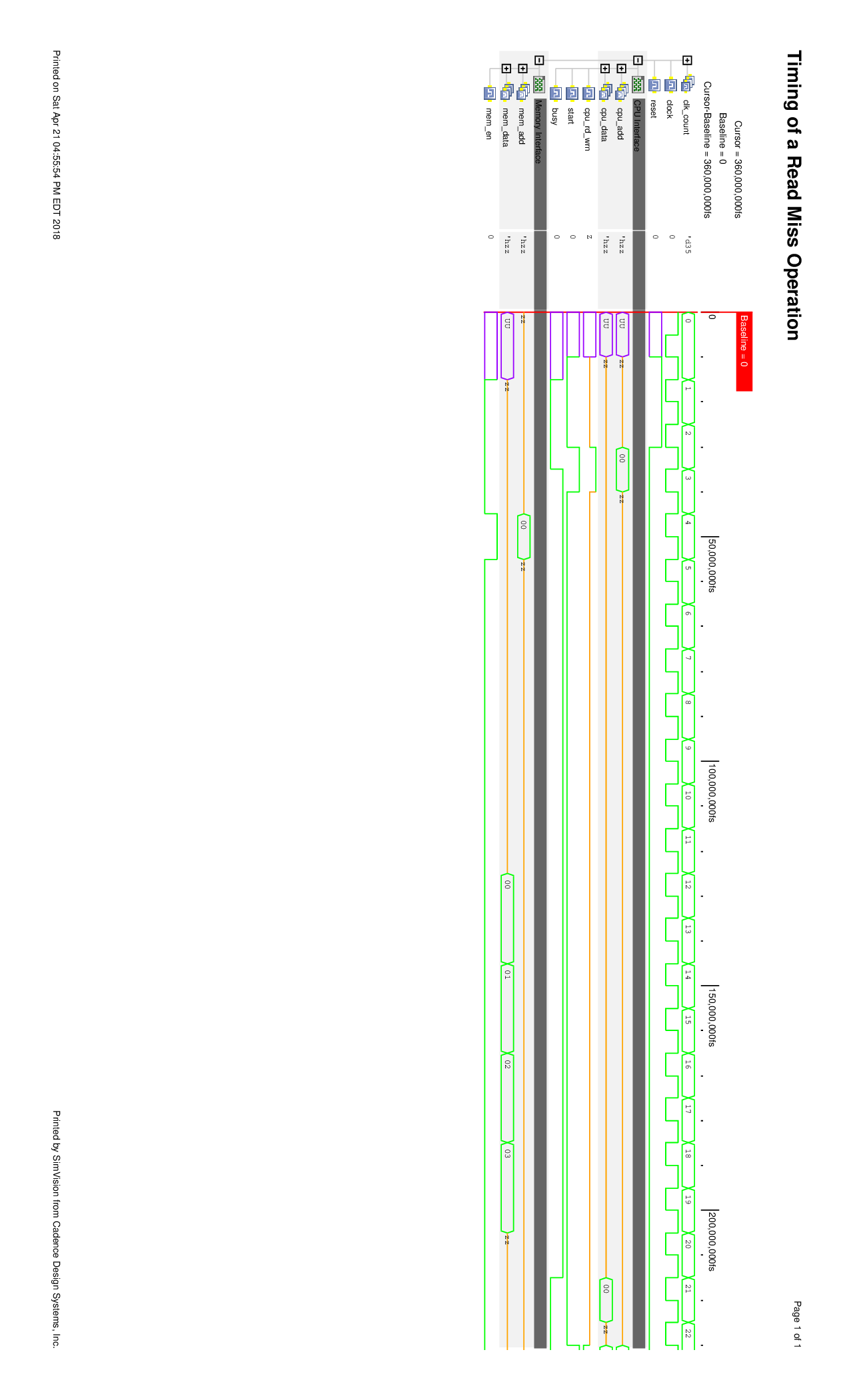
The last verification step was to confirm that back to back read misses replaced both sides of the cache. A summary of these additions are shown in Table X.

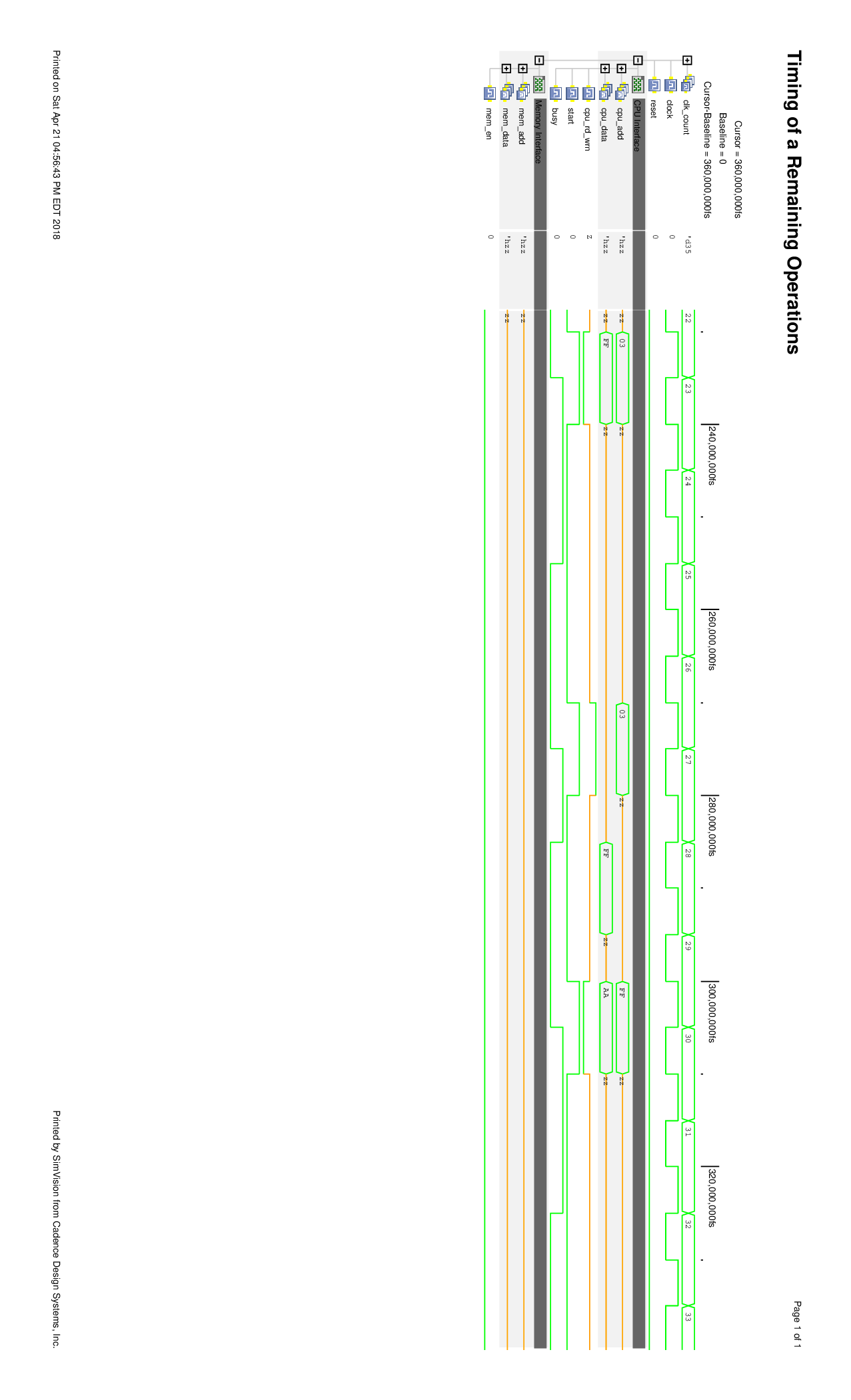
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Operation | Address | Side | Block Number | Byte | Data |
| Read Miss | 80 | R | 0 |  |  |
|  |  |  |  | Byte 00 | FF |
|  |  |  |  | Byte 01 | FE |
|  |  |  |  | Byte 10 | FD |
|  |  |  |  | Byte 11 | FC |
| Read Miss | C0 | L | 0 |  |  |
|  |  |  |  | Byte 00 | 0A |
|  |  |  |  | Byte 01 | 0B |
|  |  |  |  | Byte 10 | 0C |
|  |  |  |  | Byte 11 | 0D |
| Read Hit | 80 | R | 0 |  |  |
|  |  |  |  | Byte 00 | FF |
| Read Hit | C0 | L | 0 |  |  |
|  |  |  |  | Byte 00 | 0A |

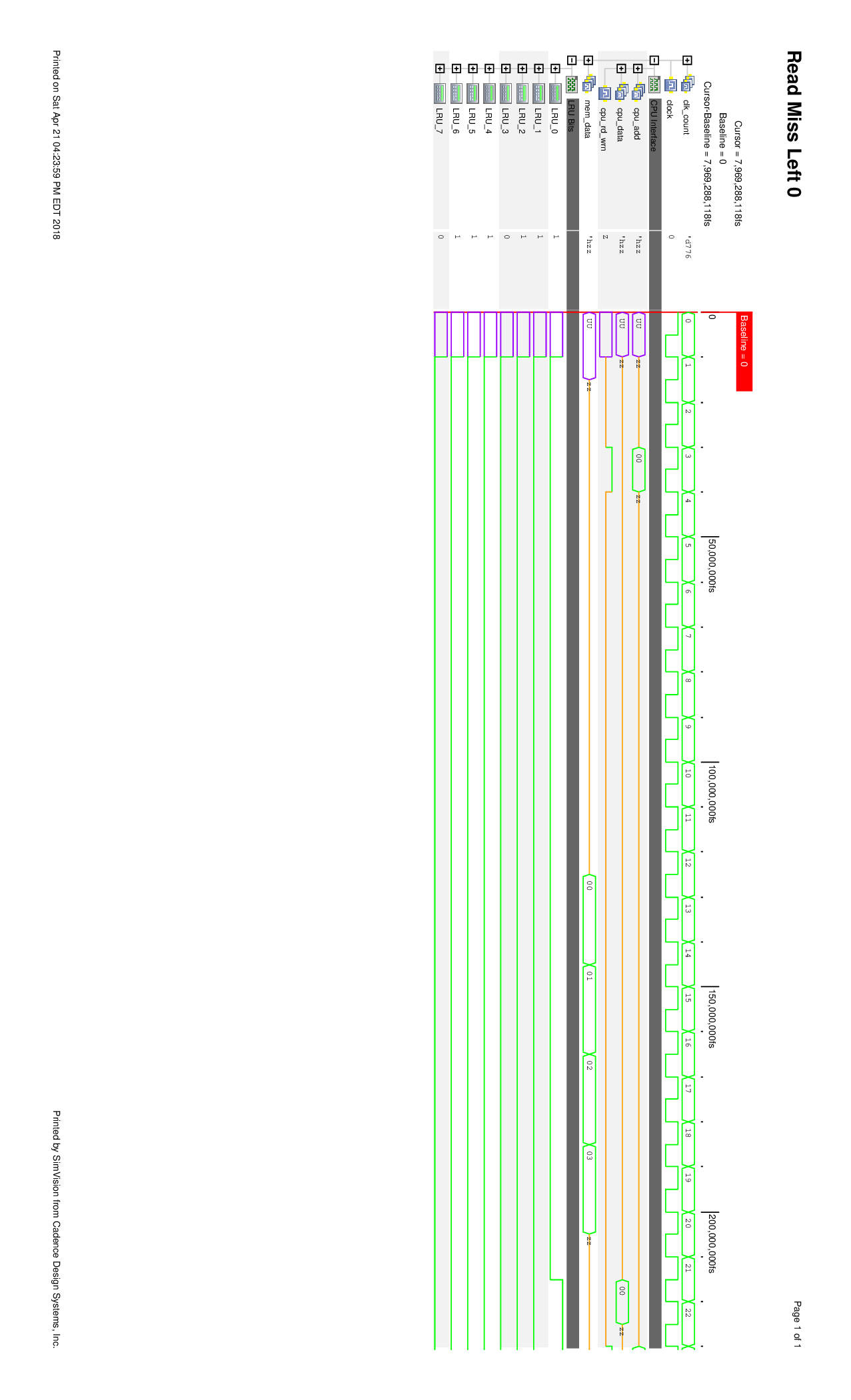
Additional waveforms are shown in Appendix X.

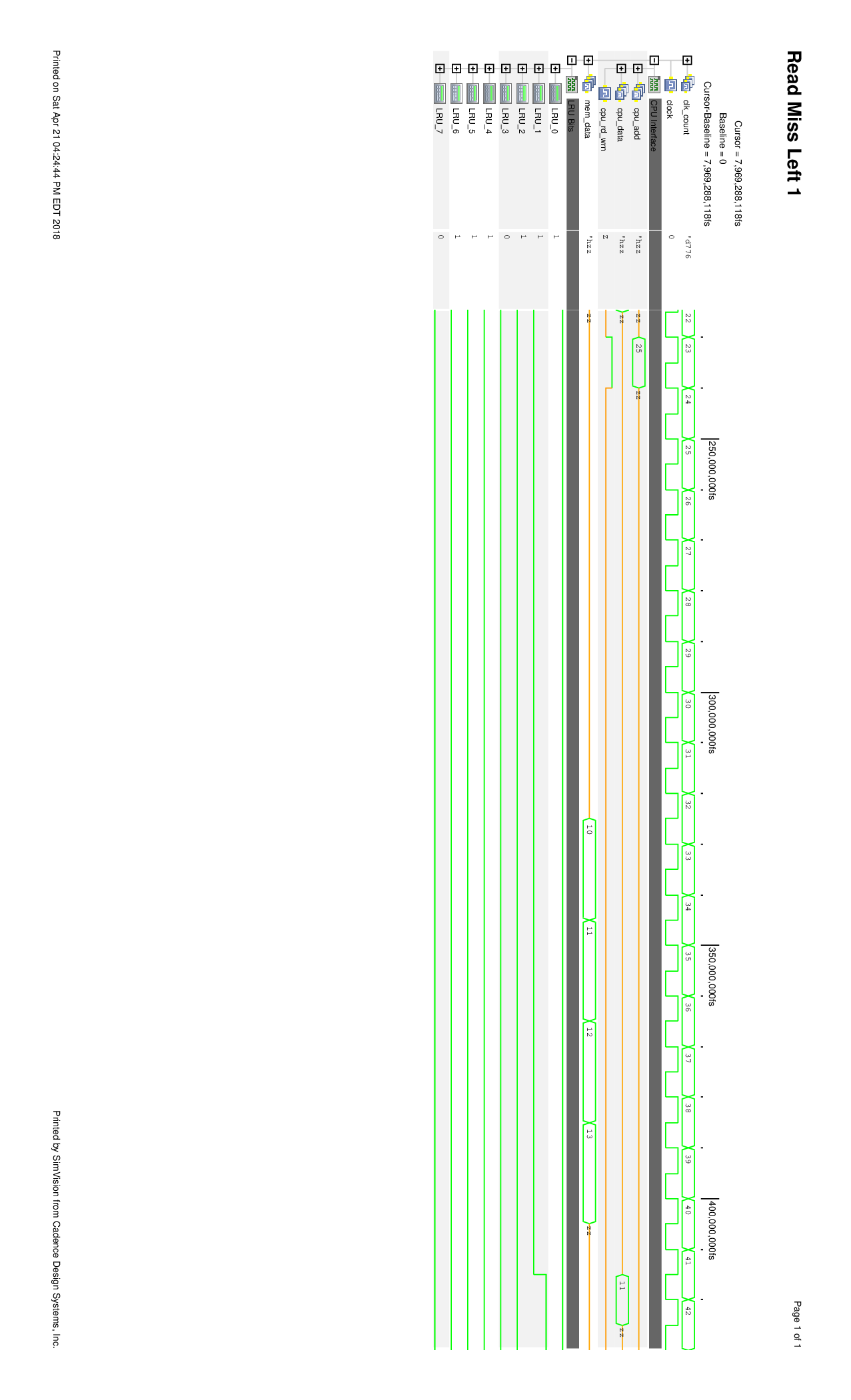
**Work Breakdown**

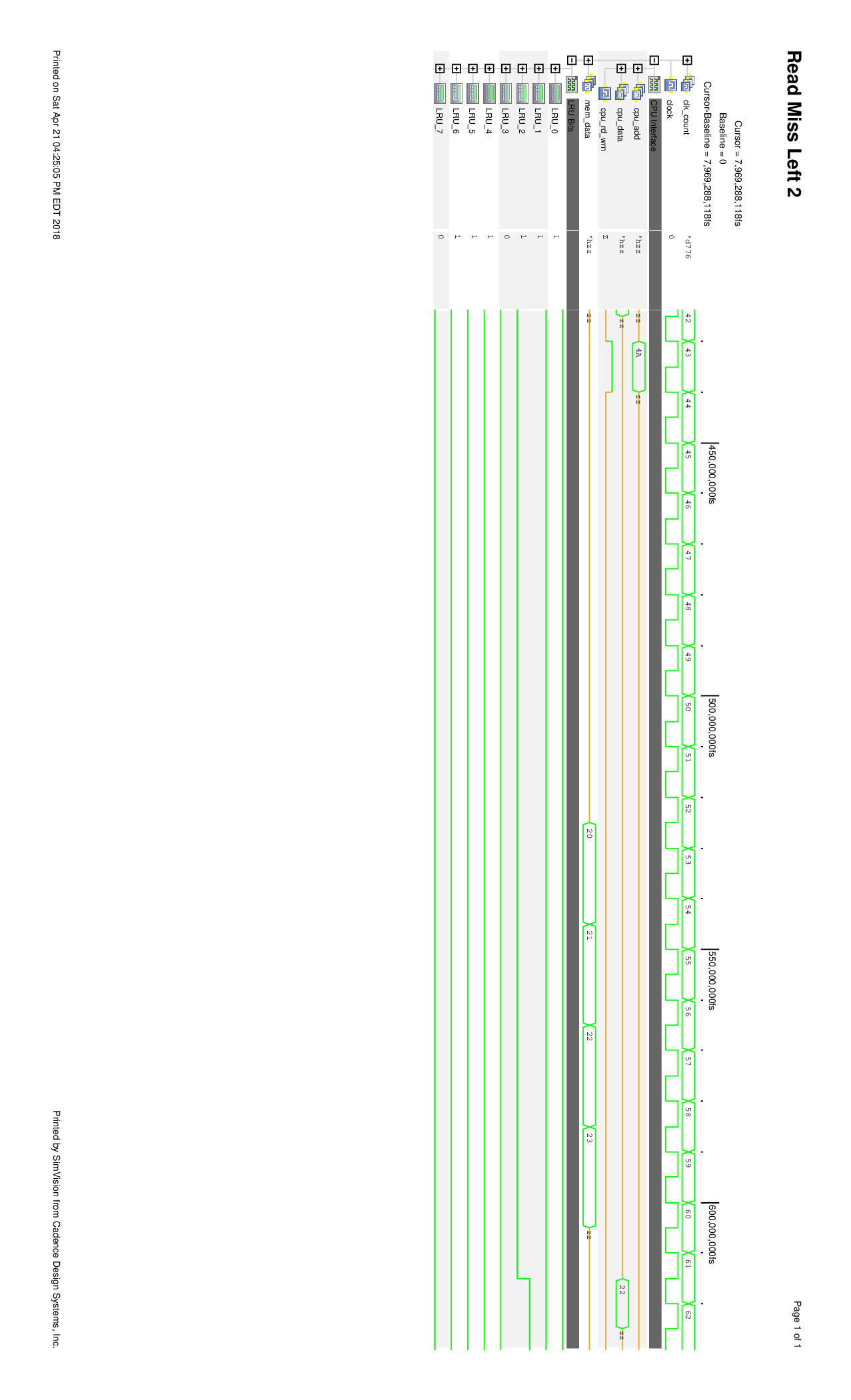
|  |  |
| --- | --- |
| **Ben** | **Marinna** |
| Cache Design (code) | Simple Logic Gate Code (and, xnor, tx, nand) |
| State Machine Output Logic | Decoder Design (code) |
|  | State Machine Input Logic |
| Test Benches for Two-way Associative | LRU Design |

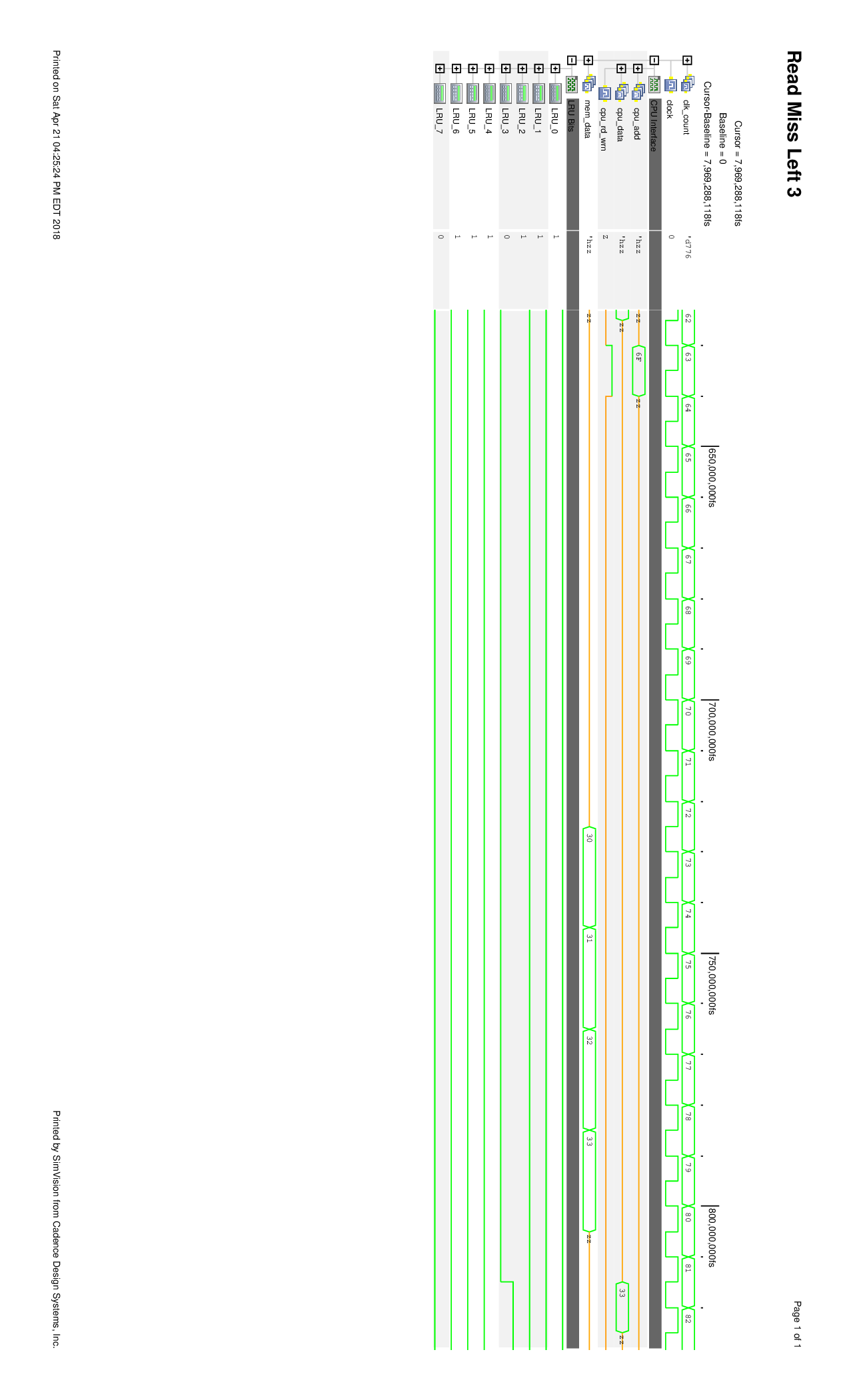
Appendix X. Timing Diagrams

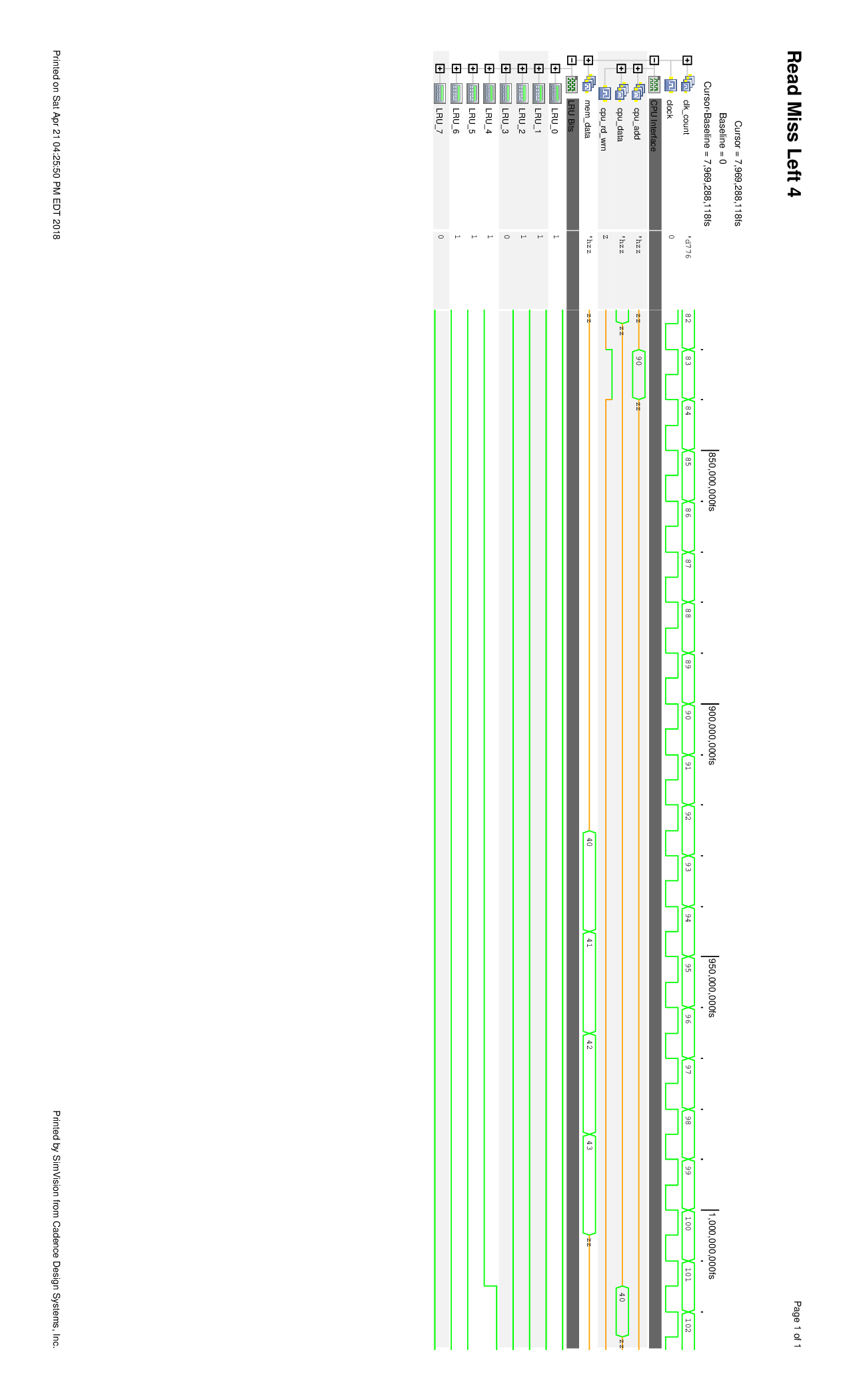


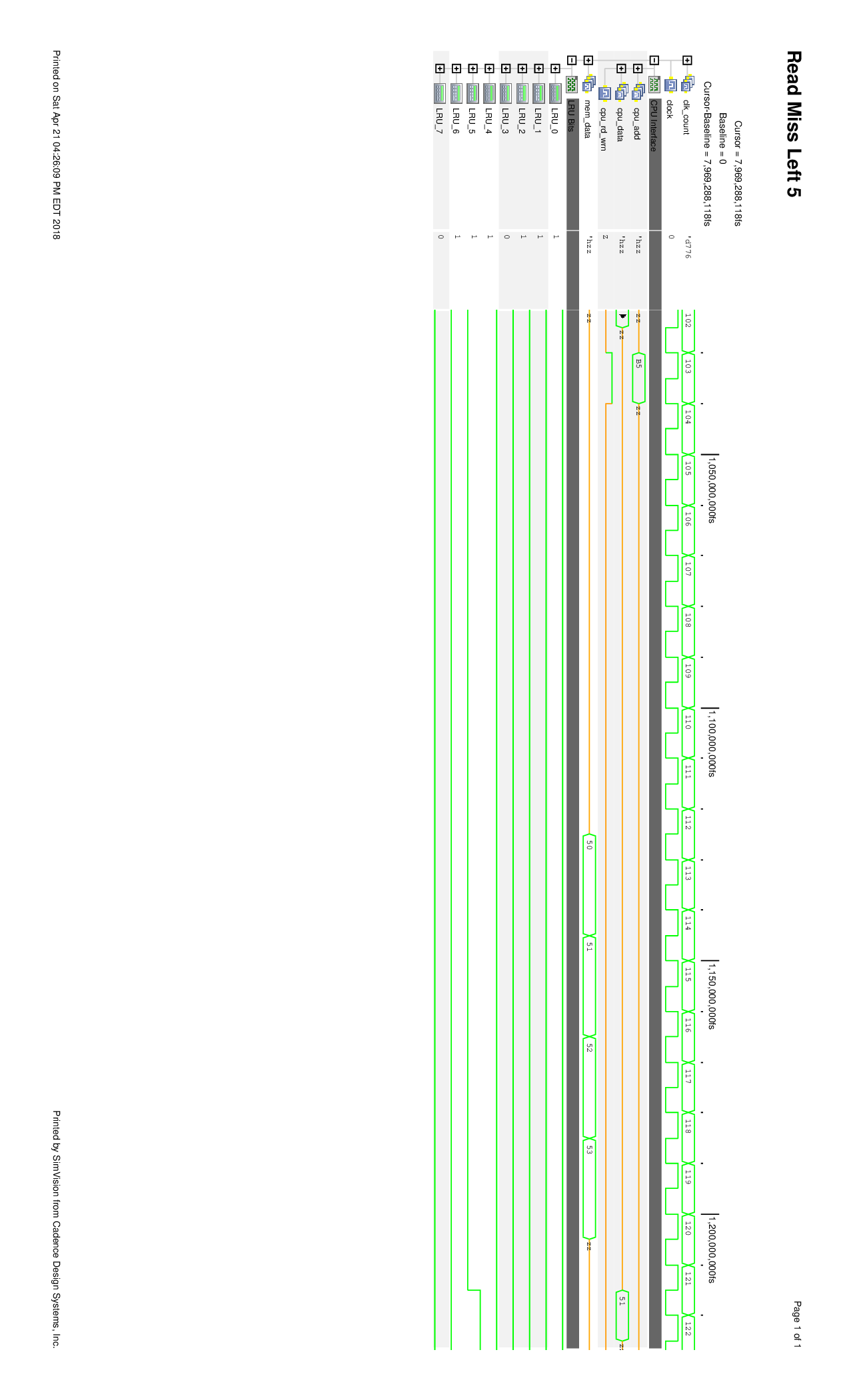


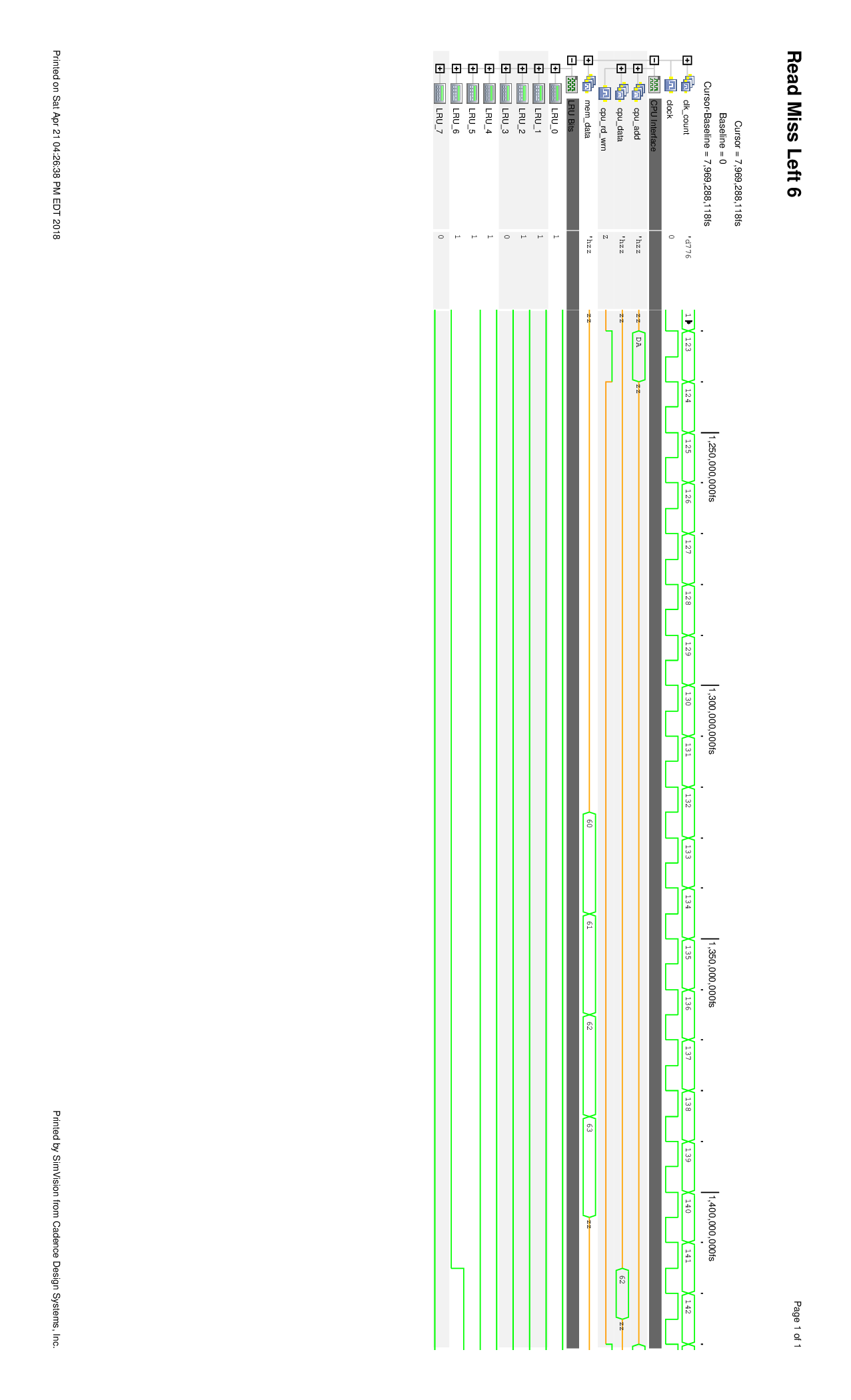


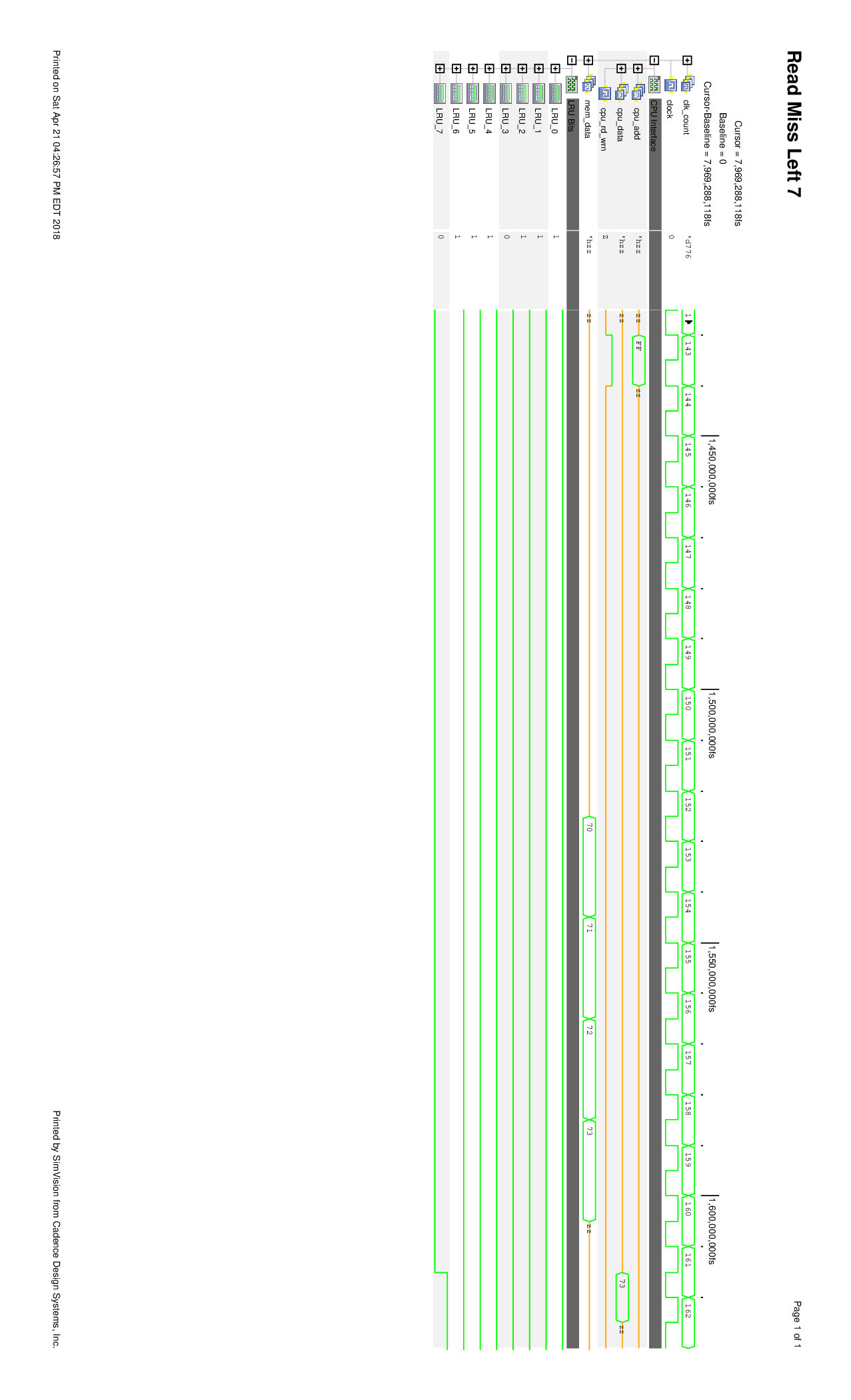


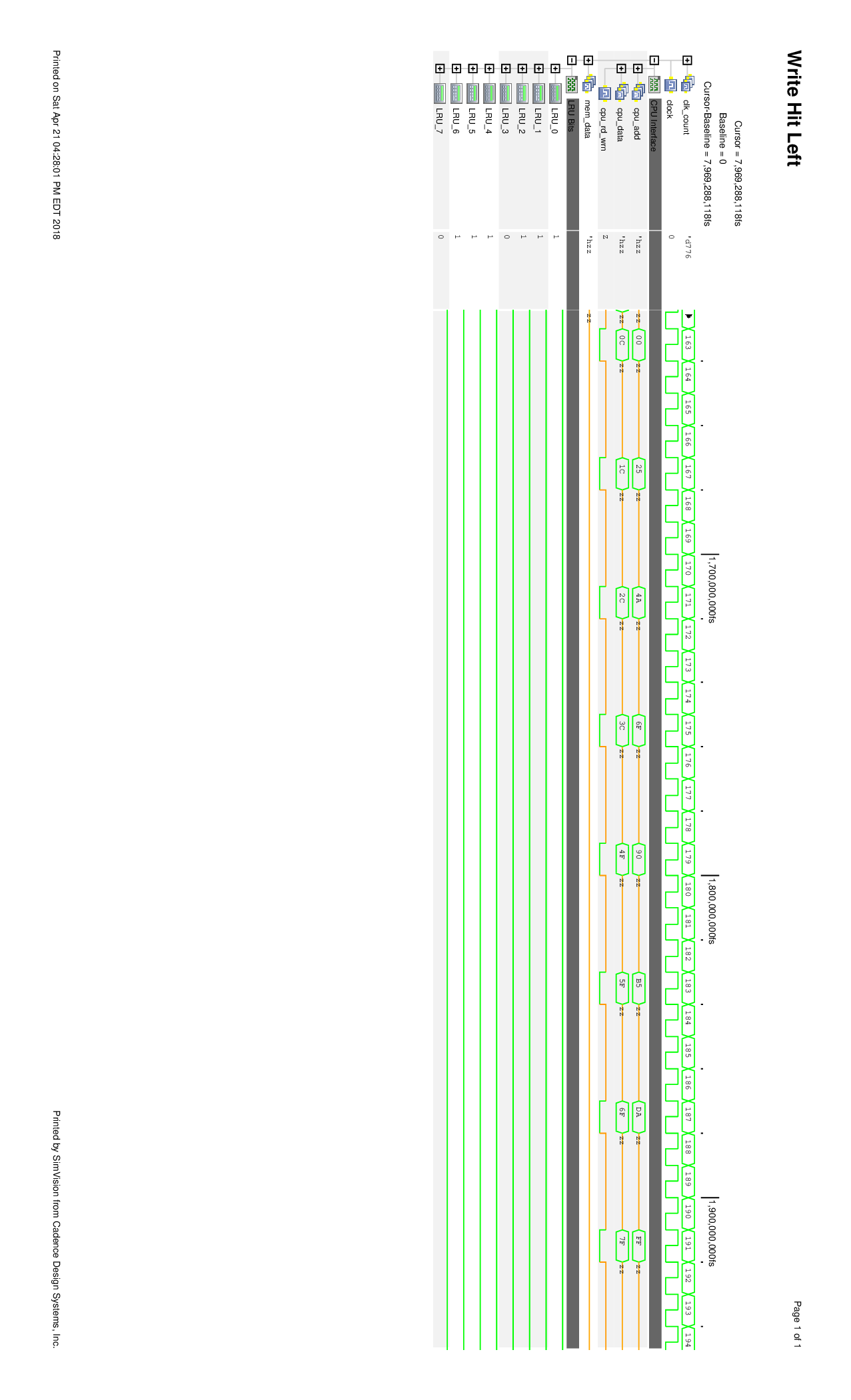


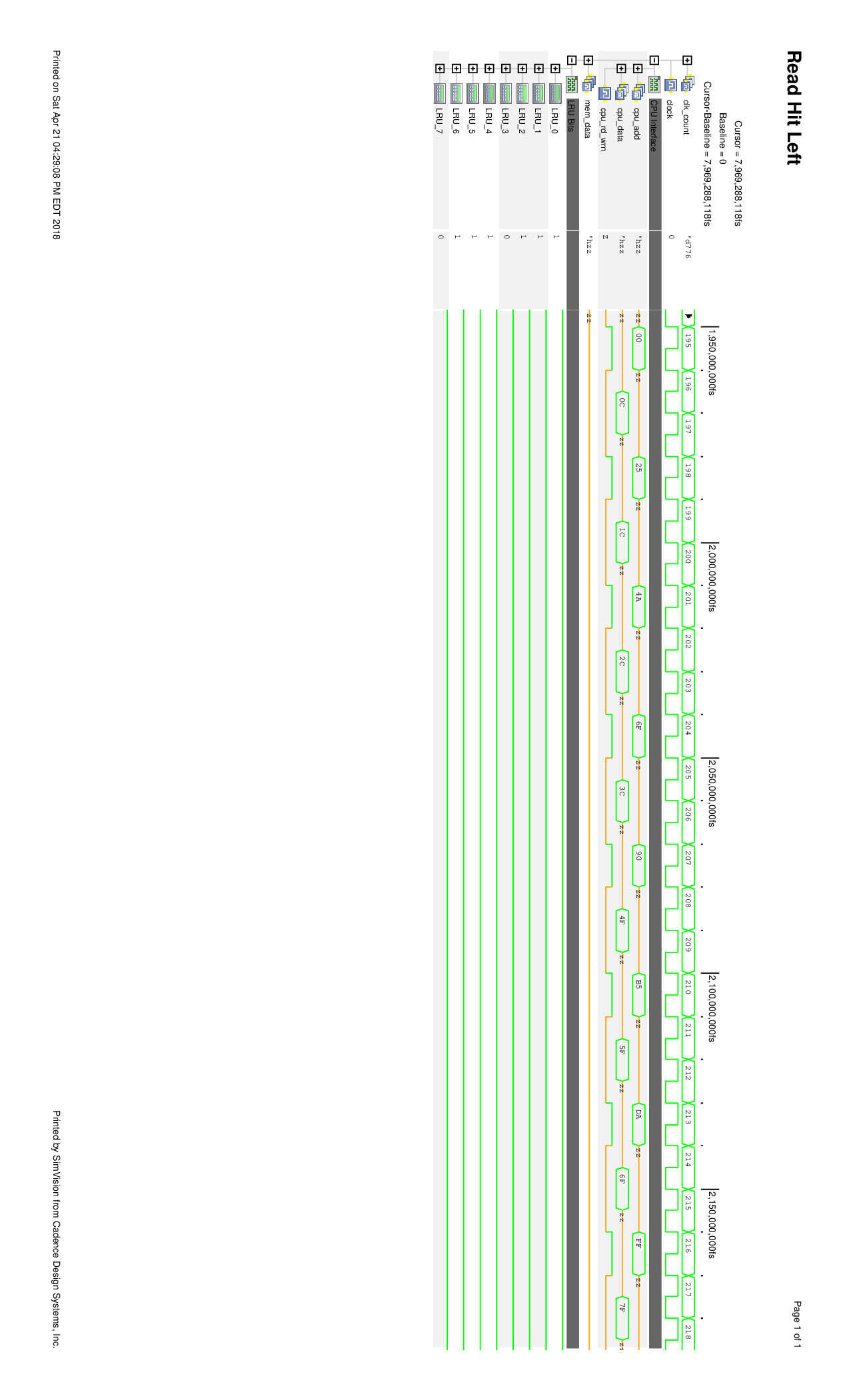


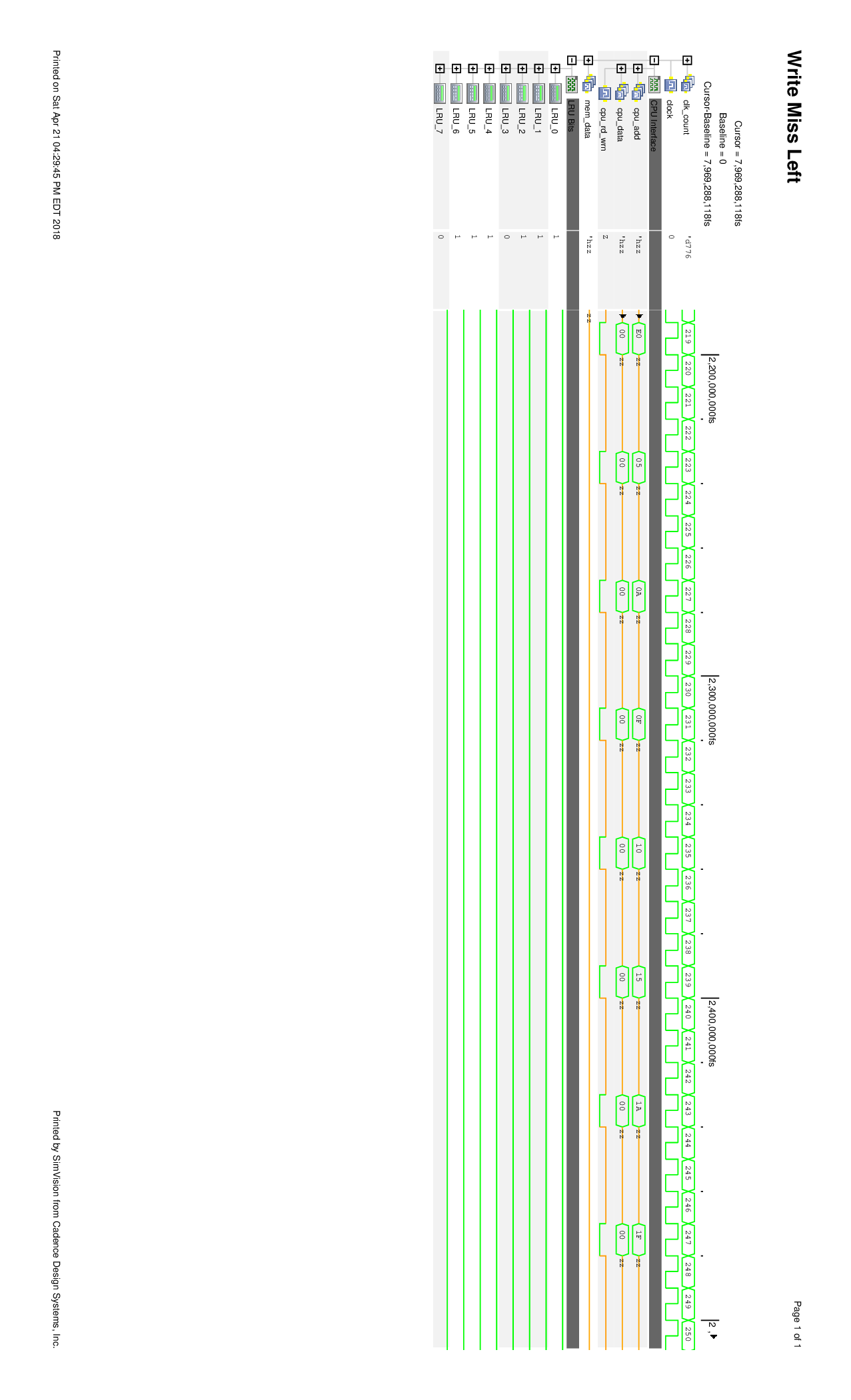


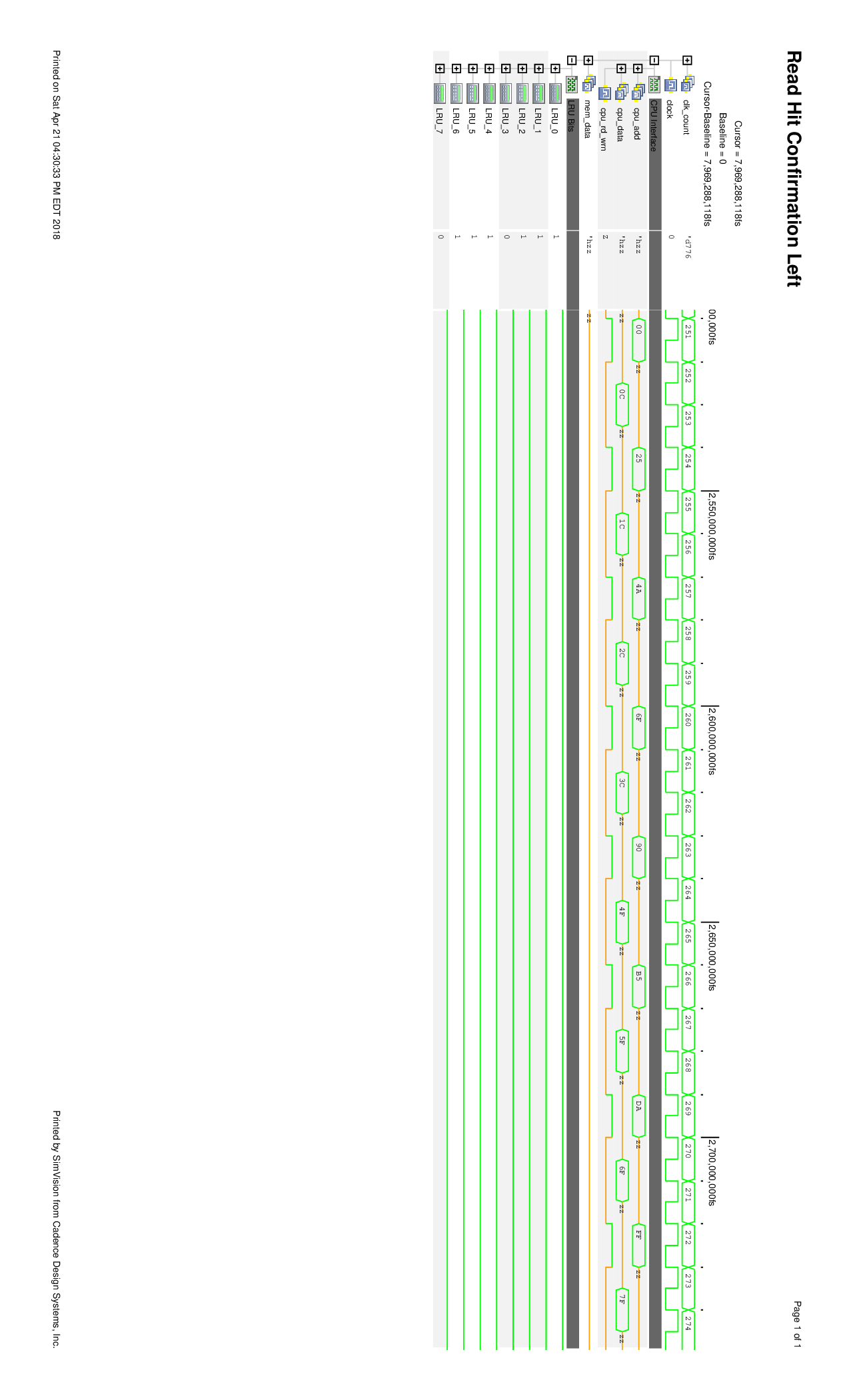


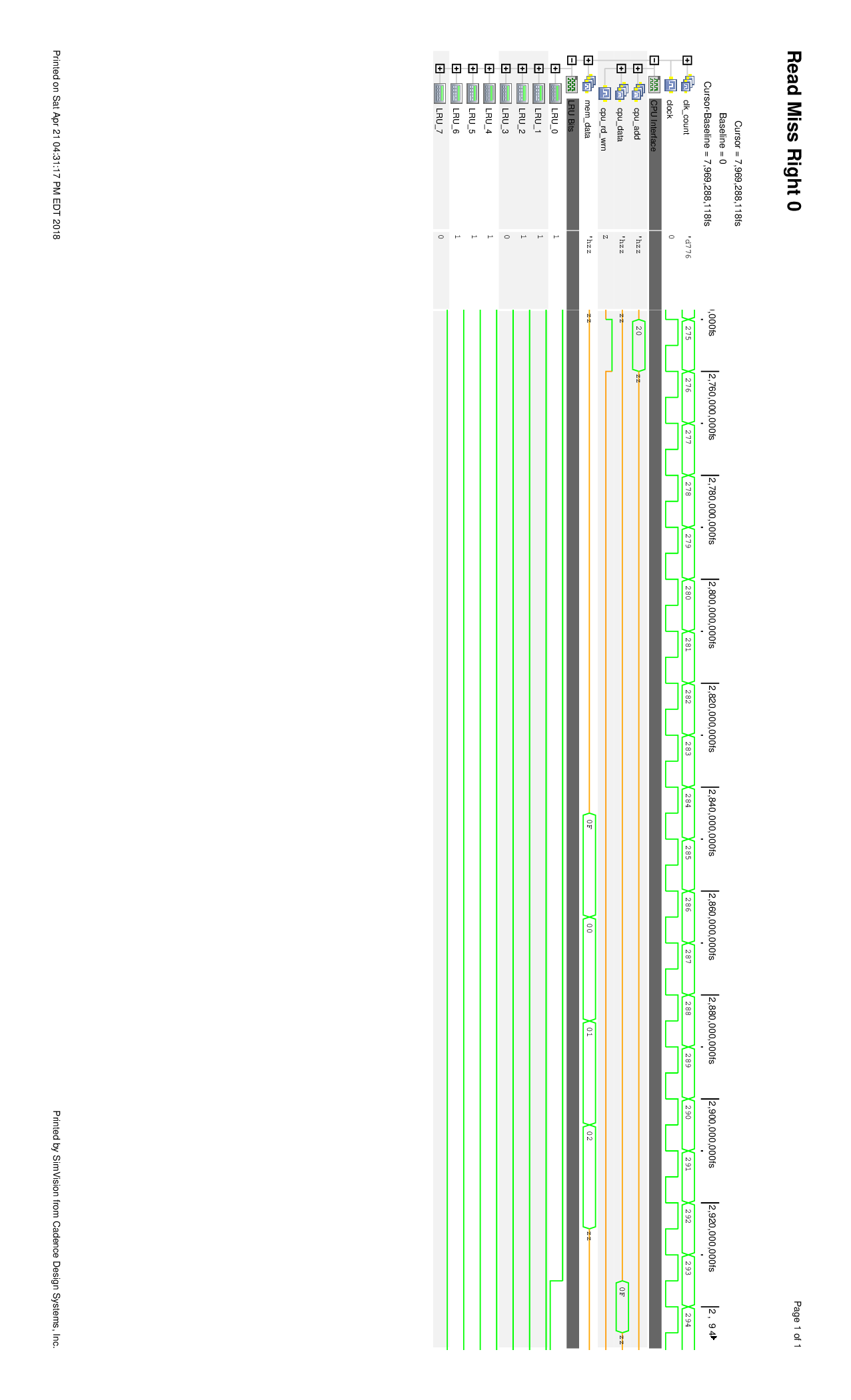


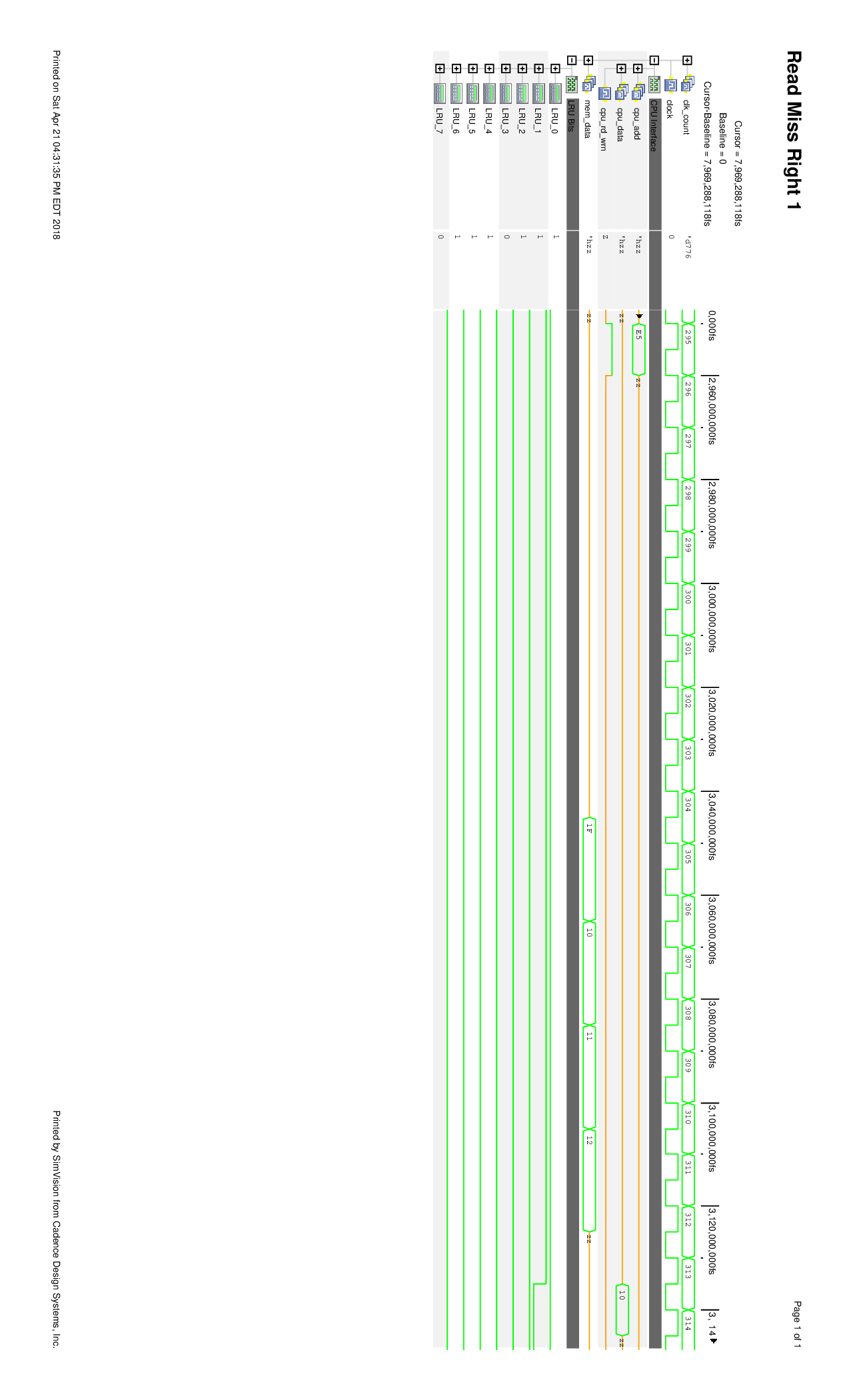


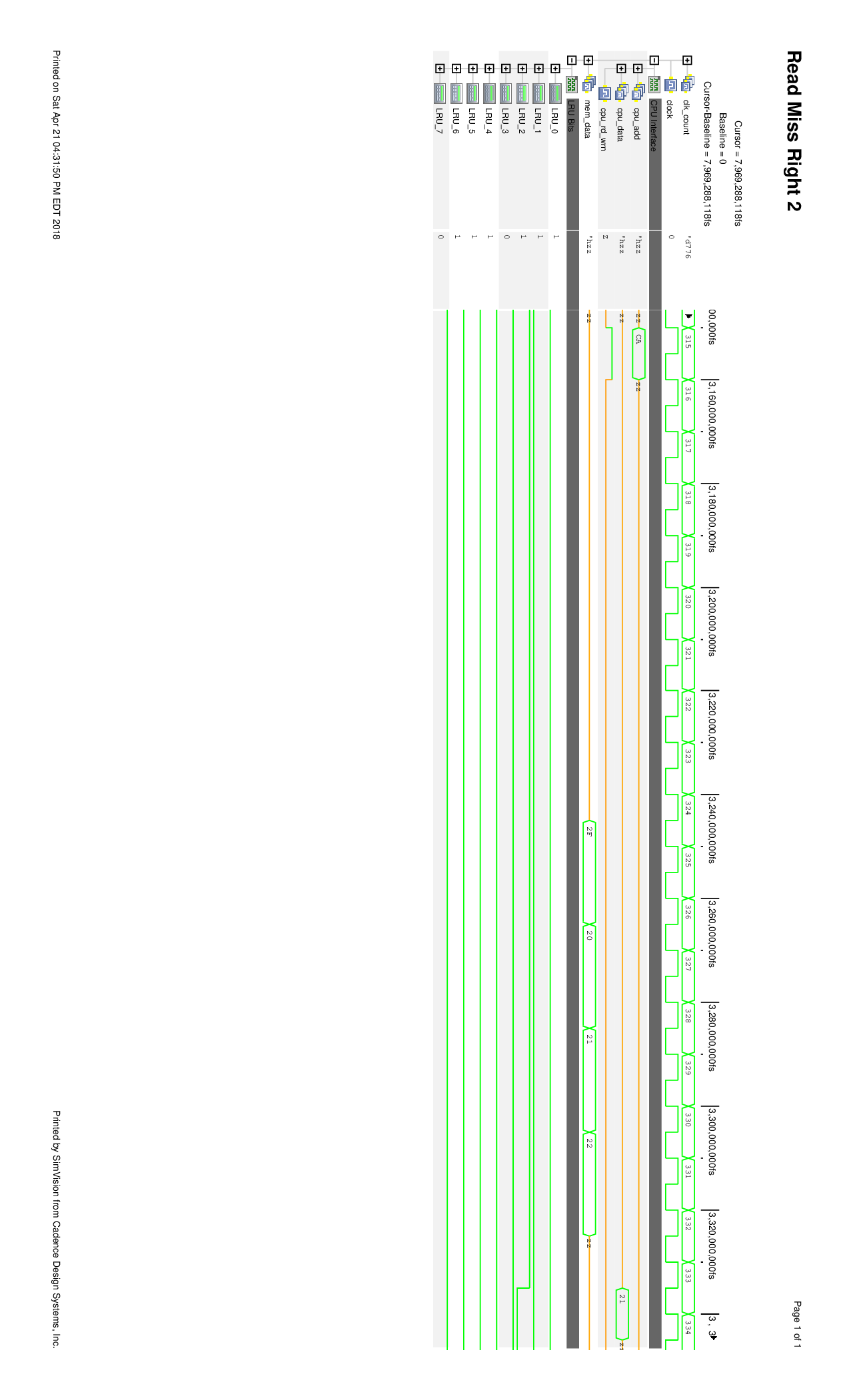


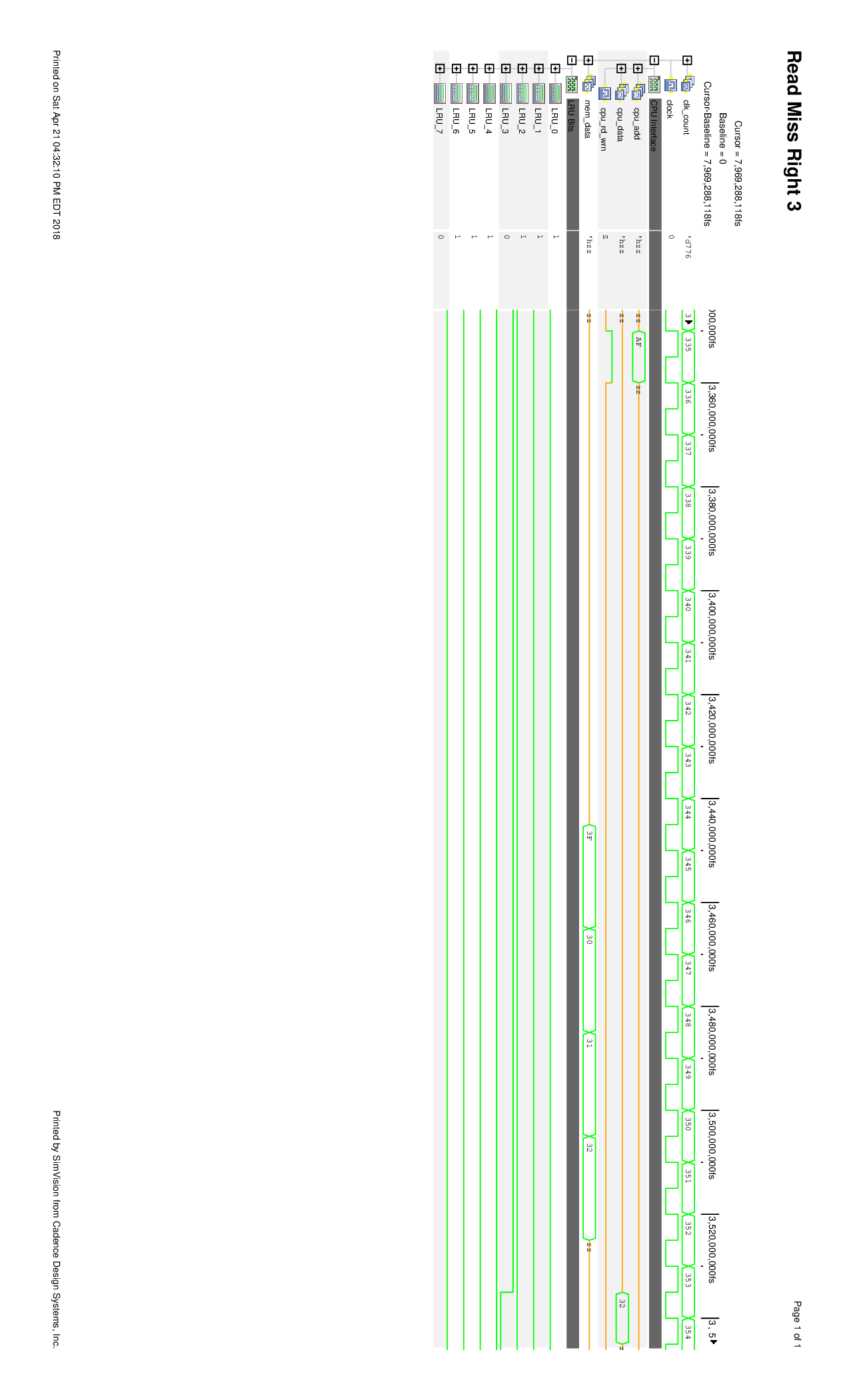


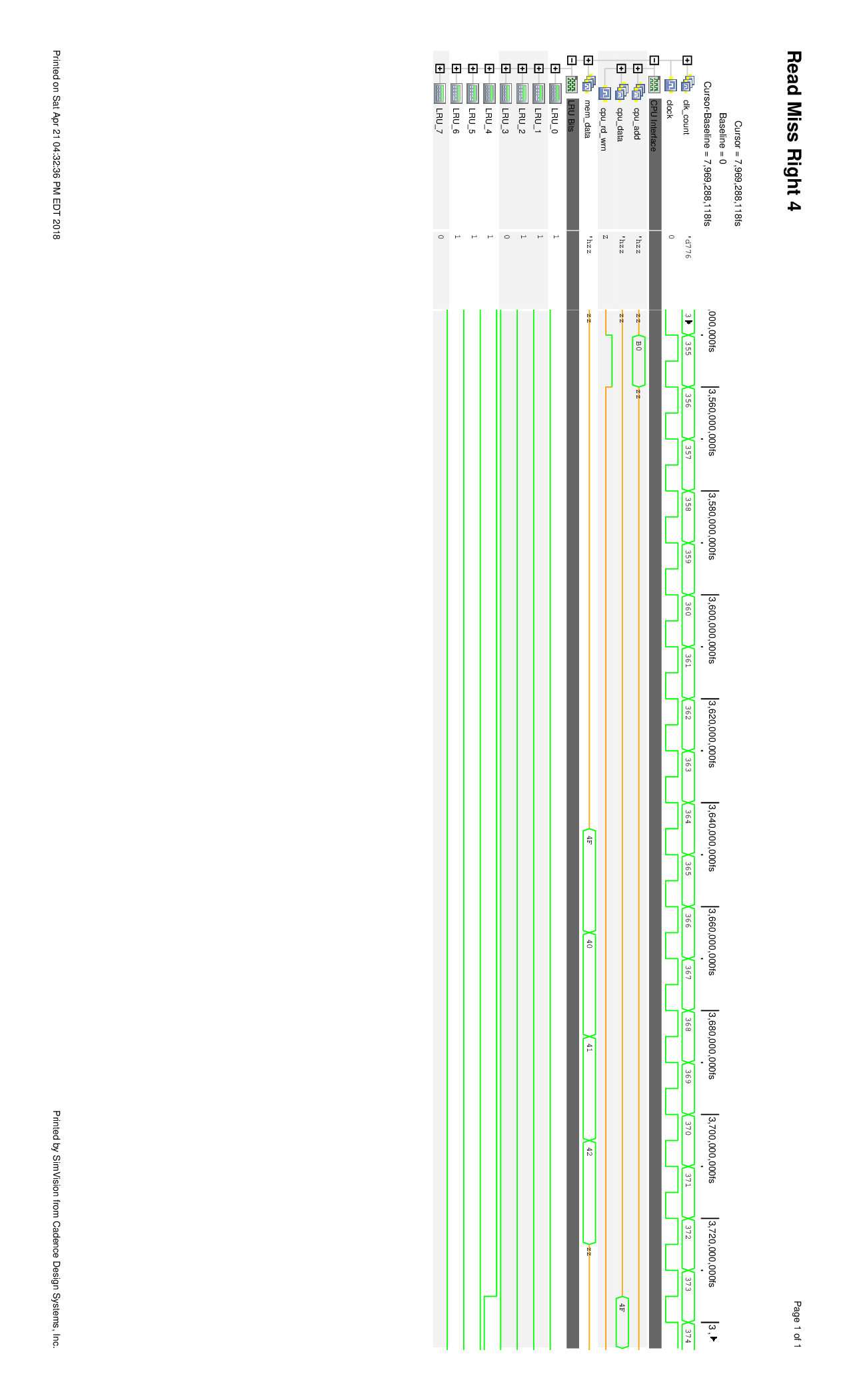


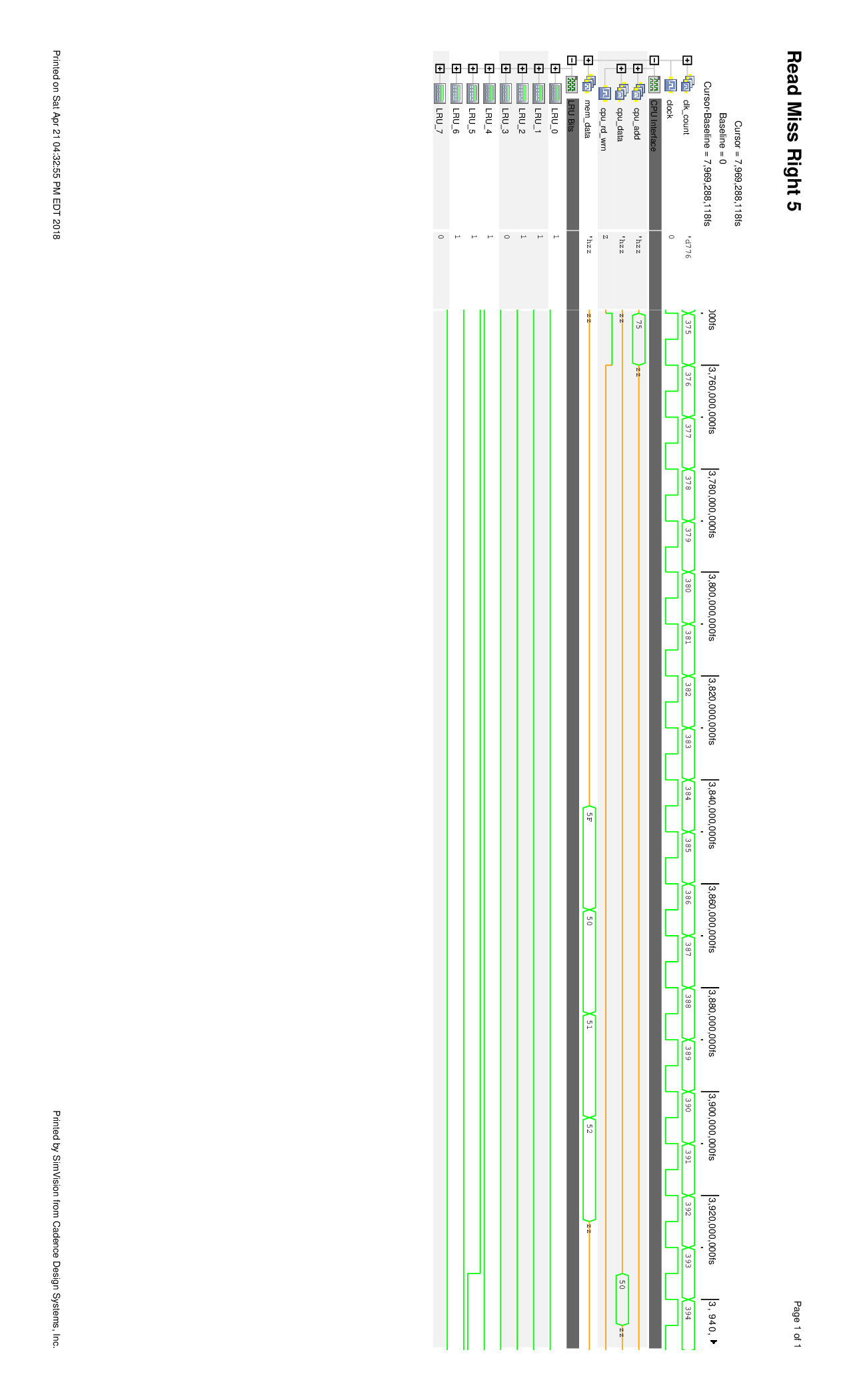


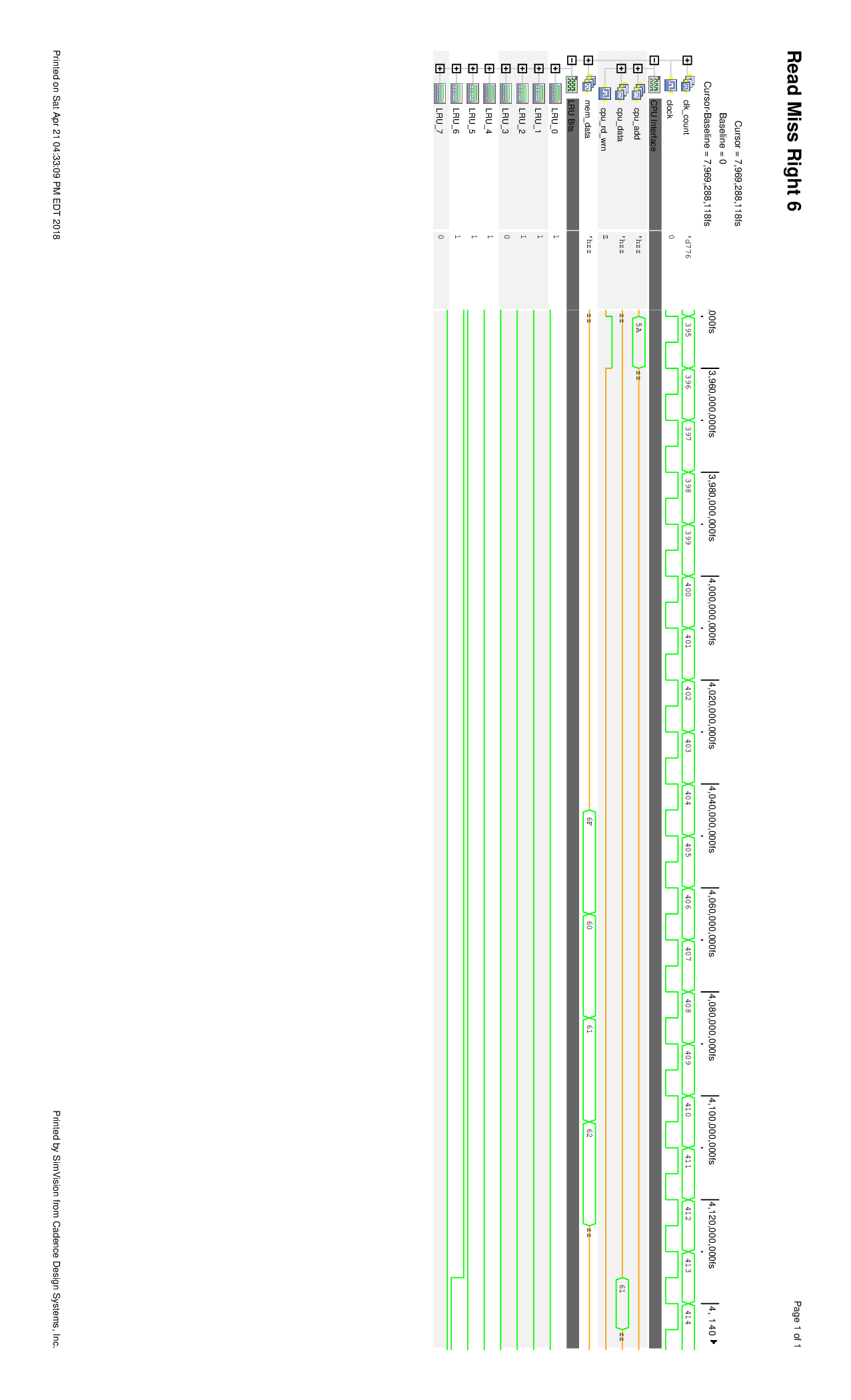


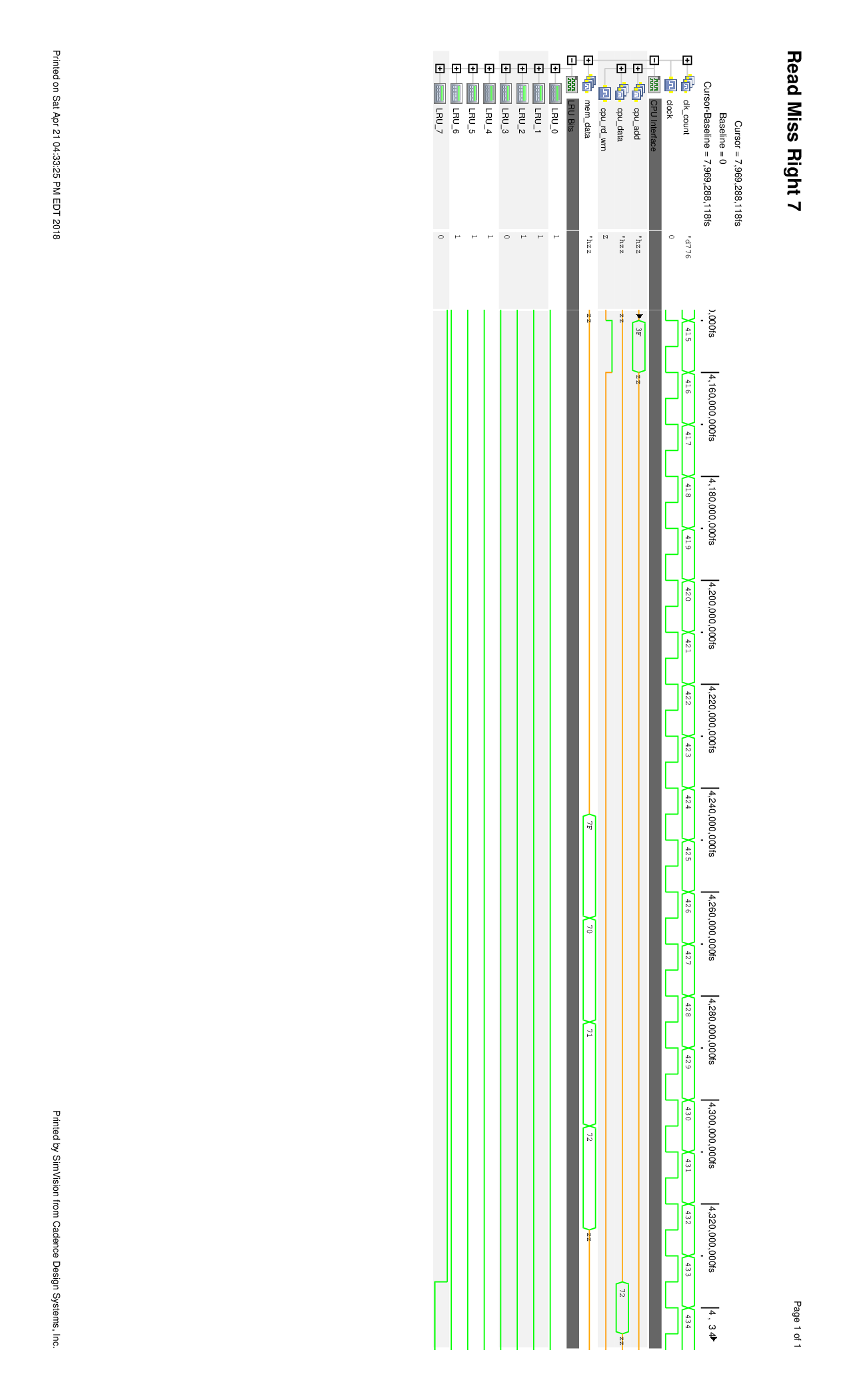


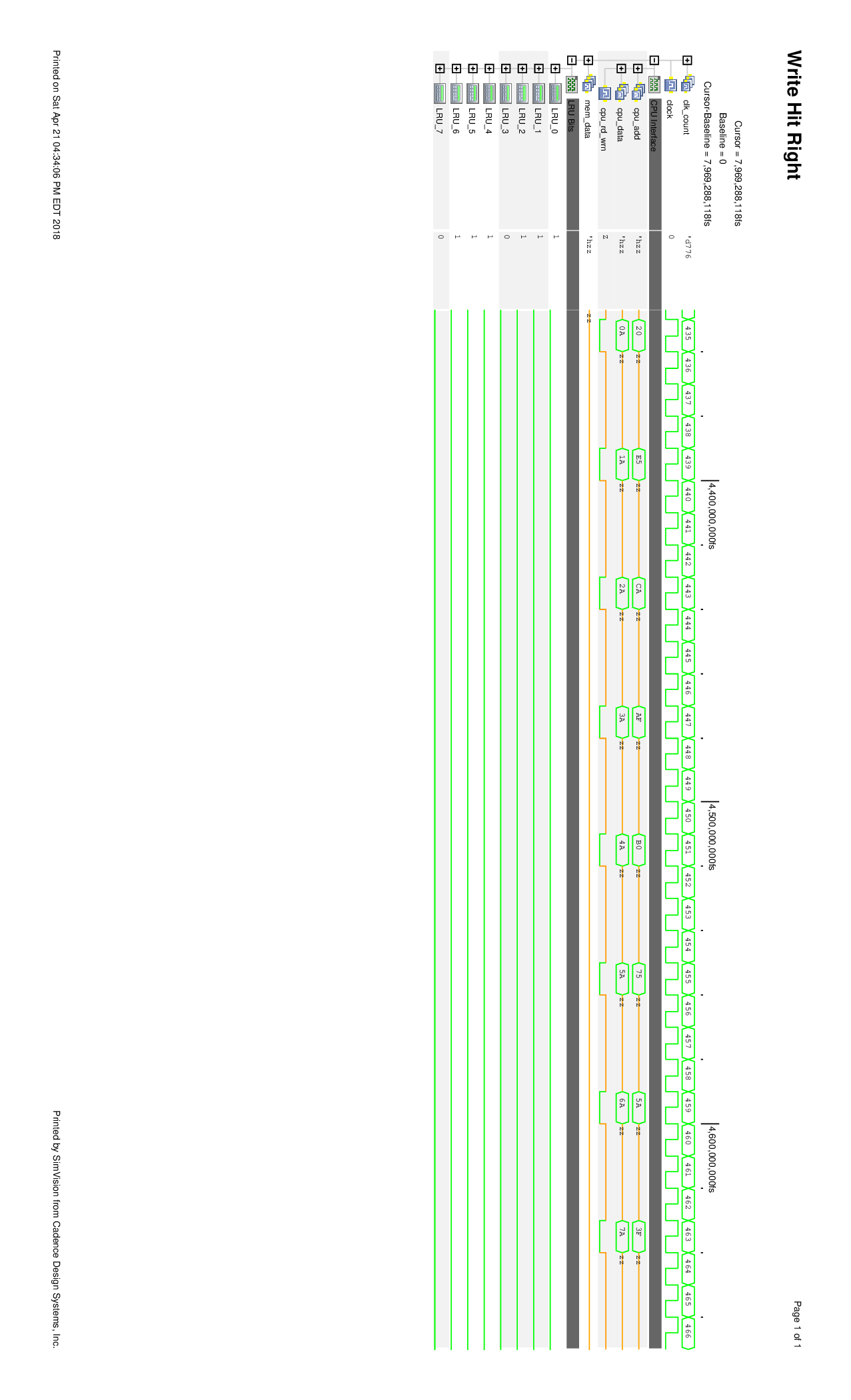


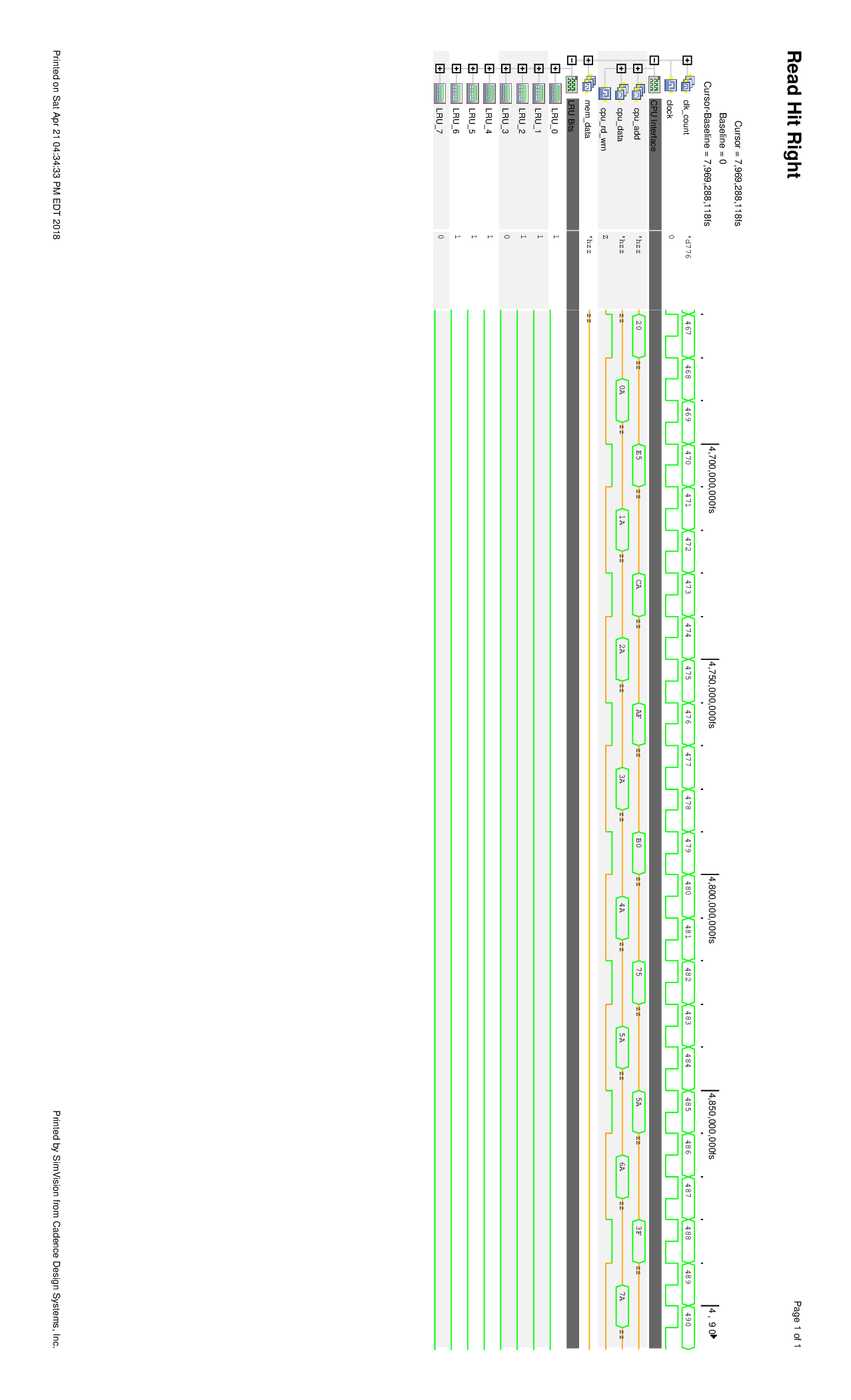


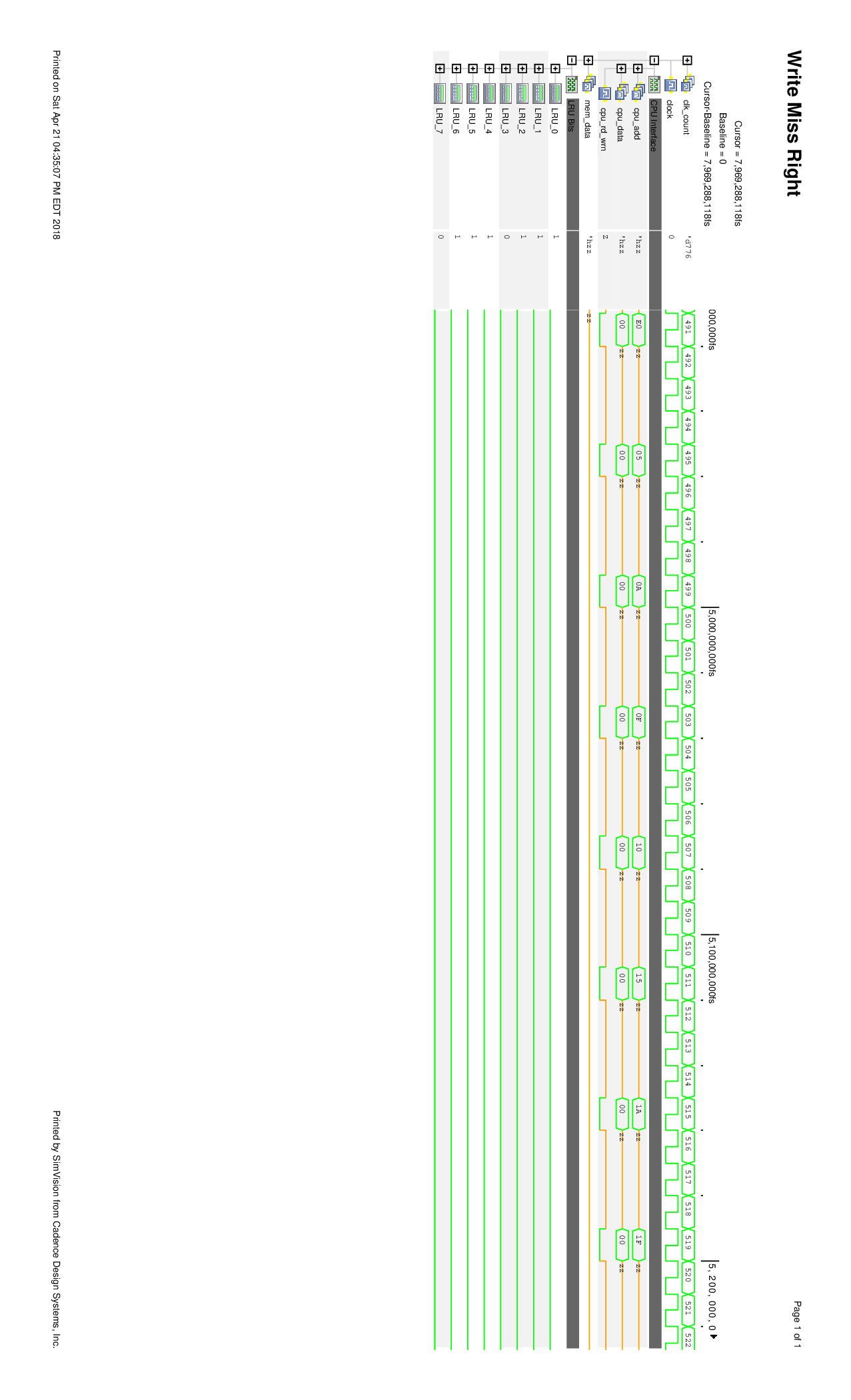


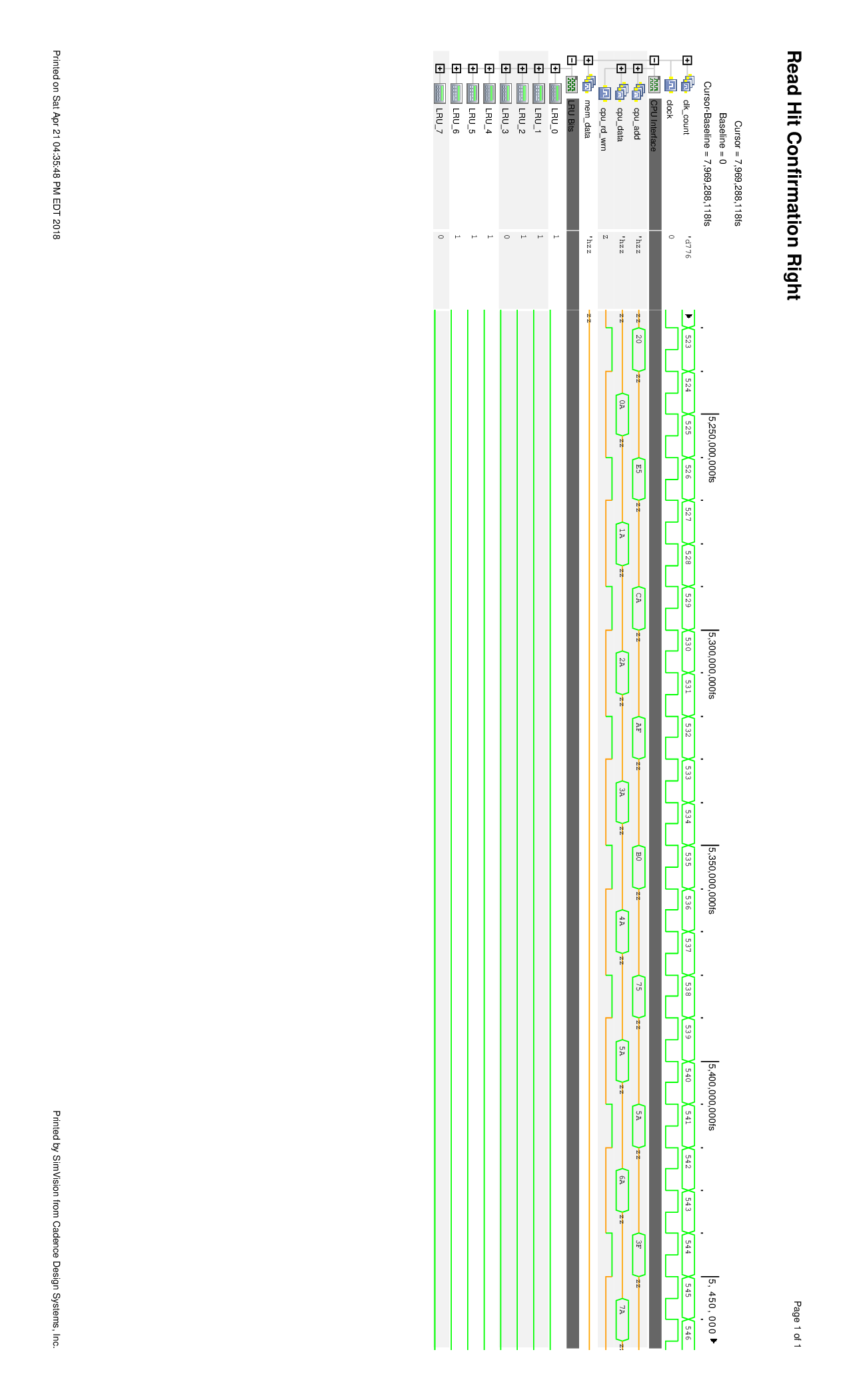


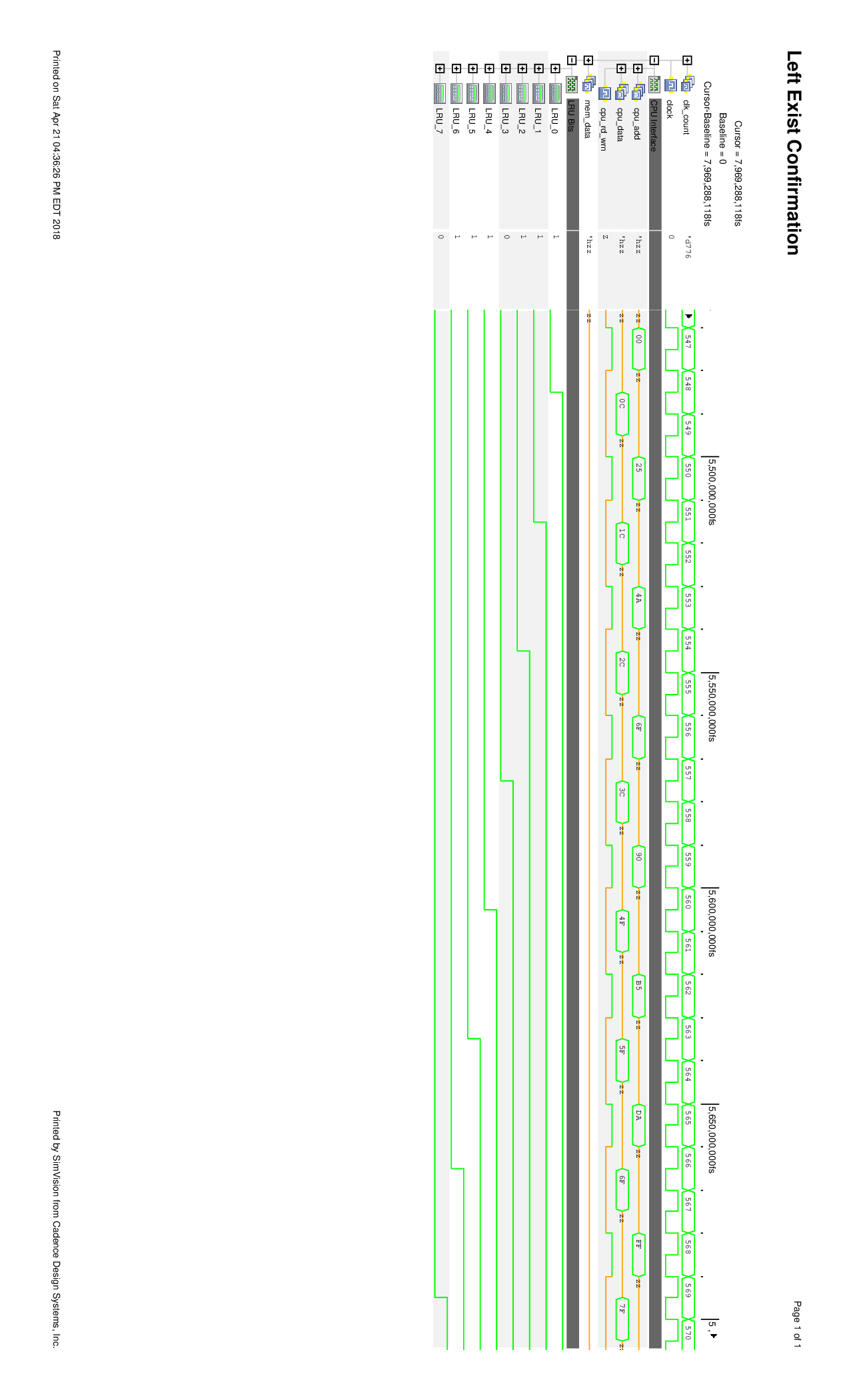


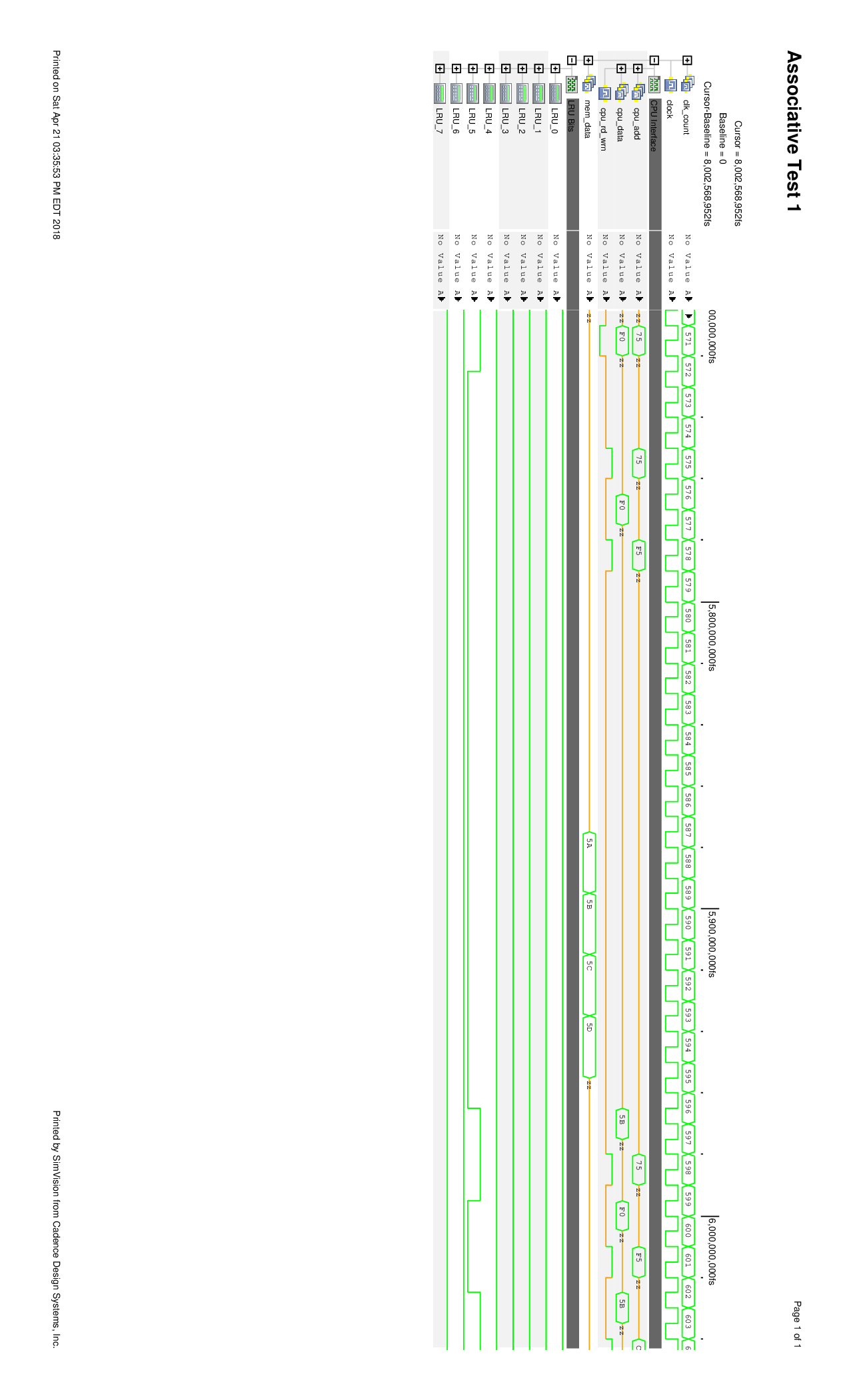


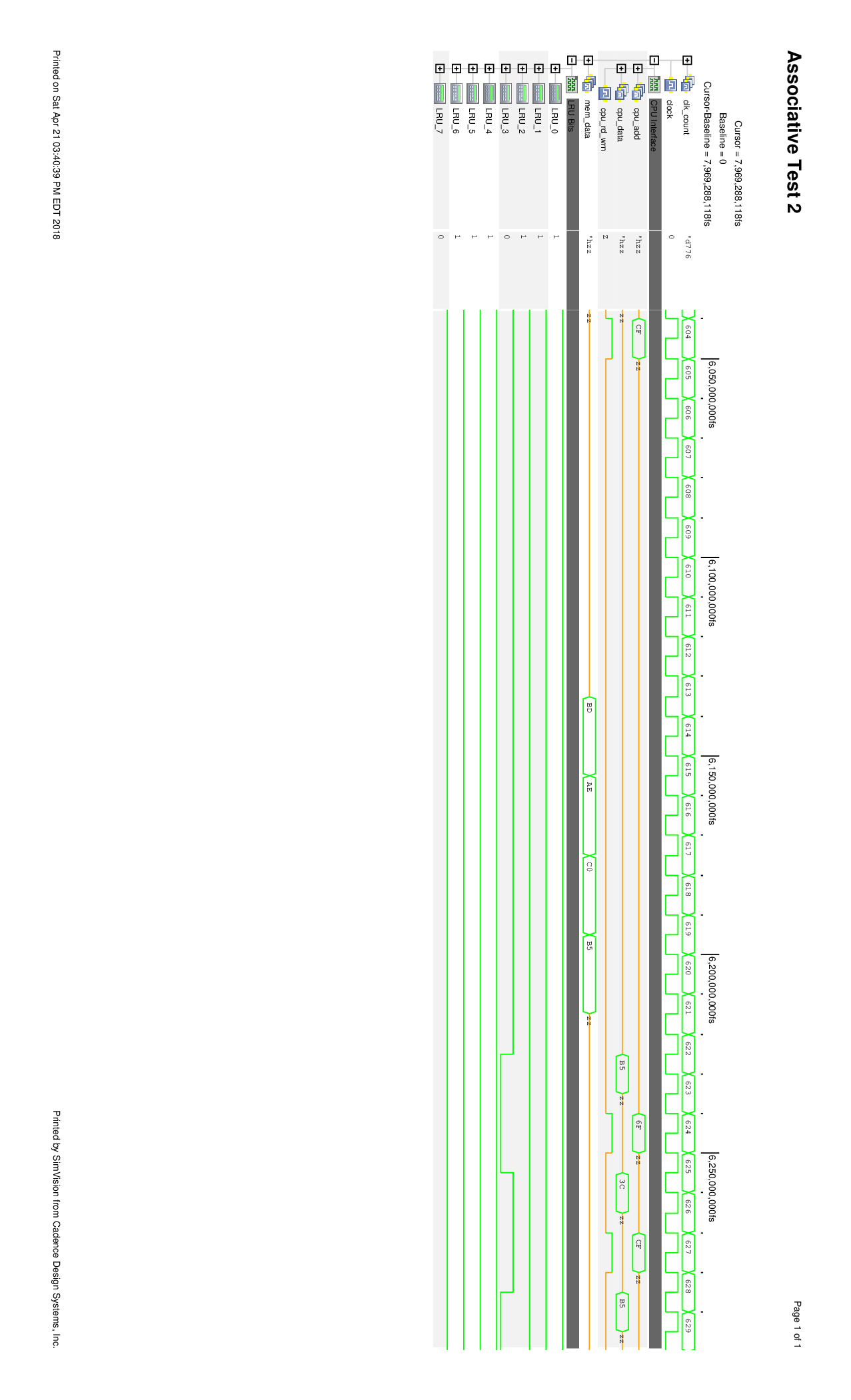


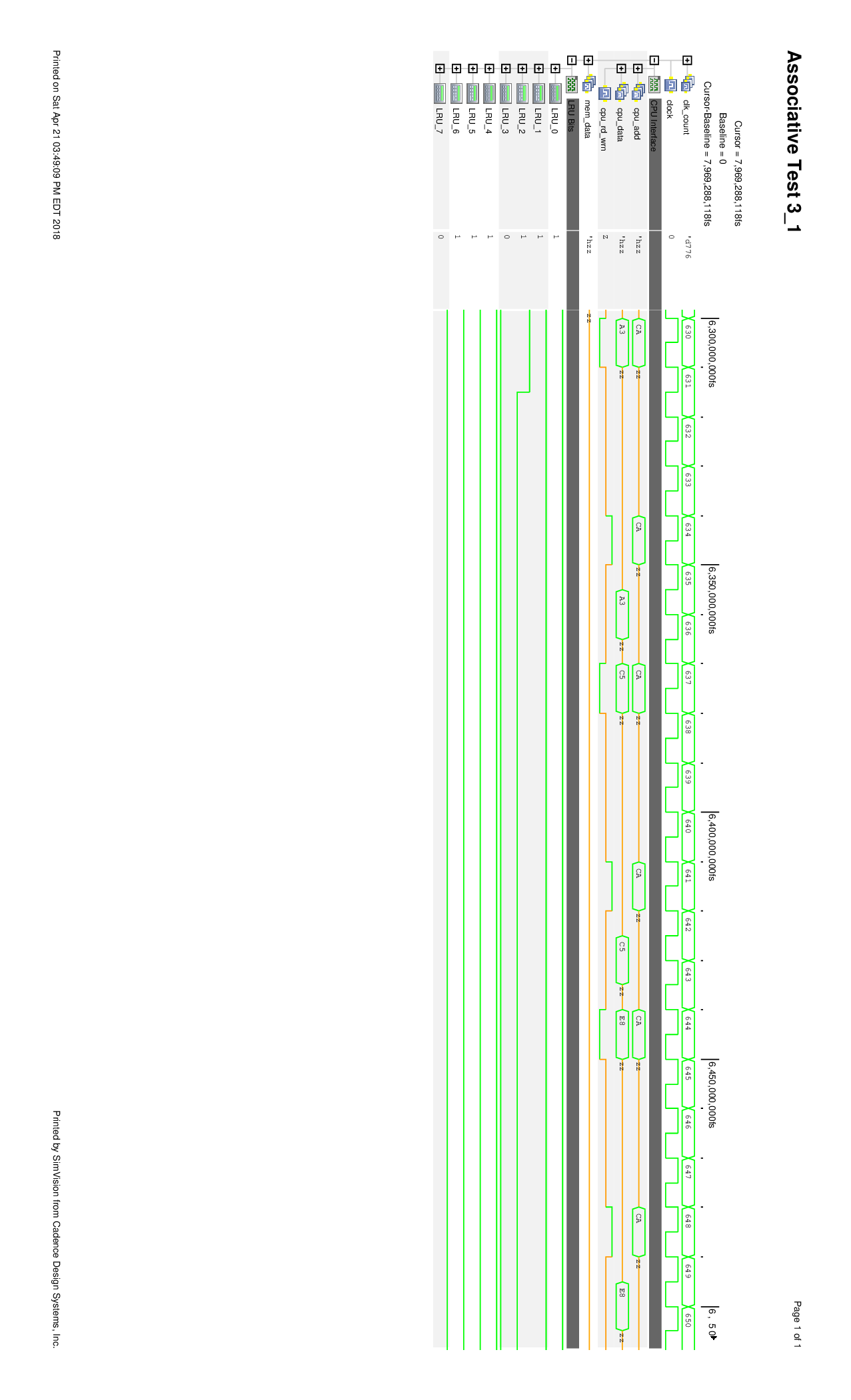


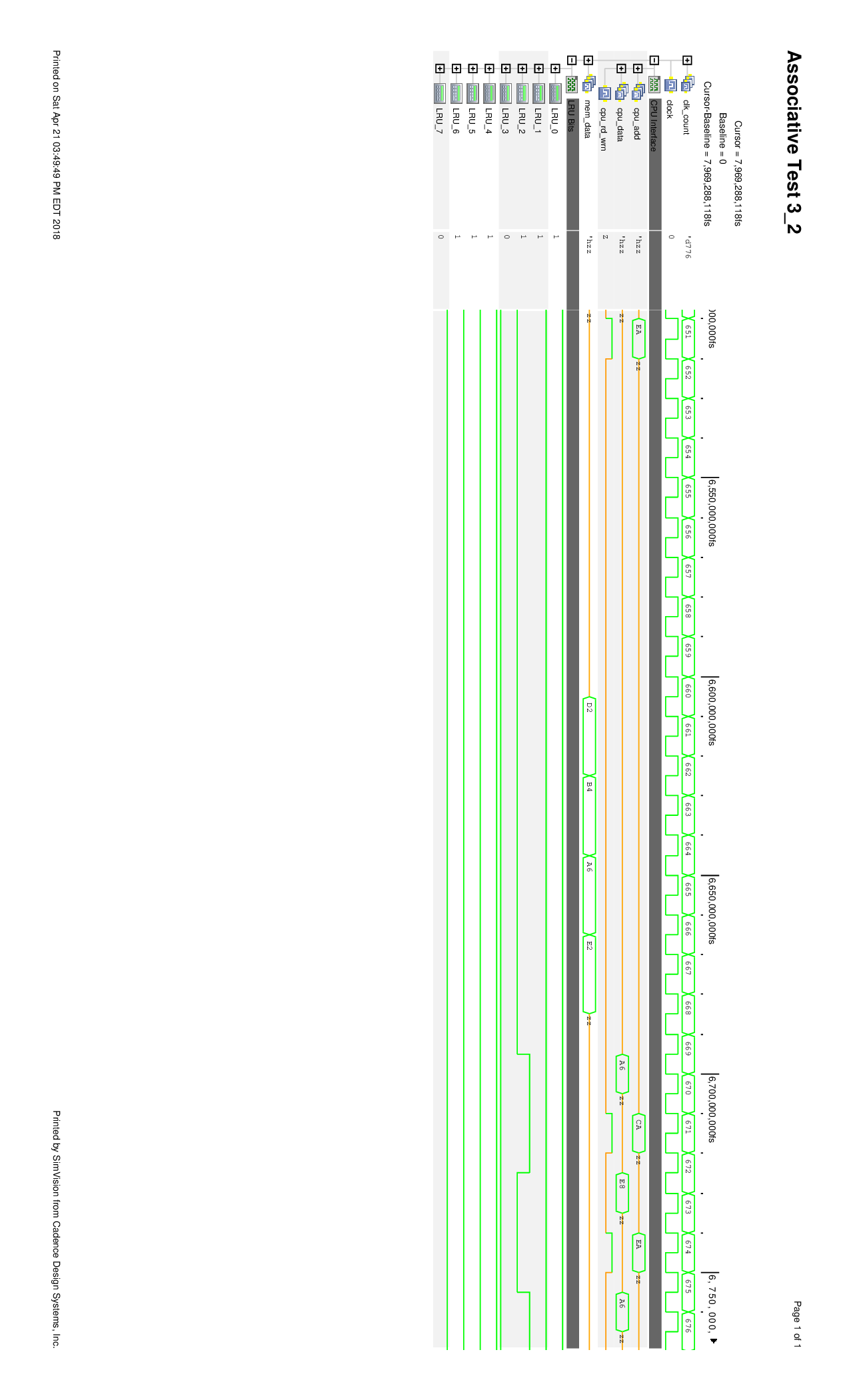


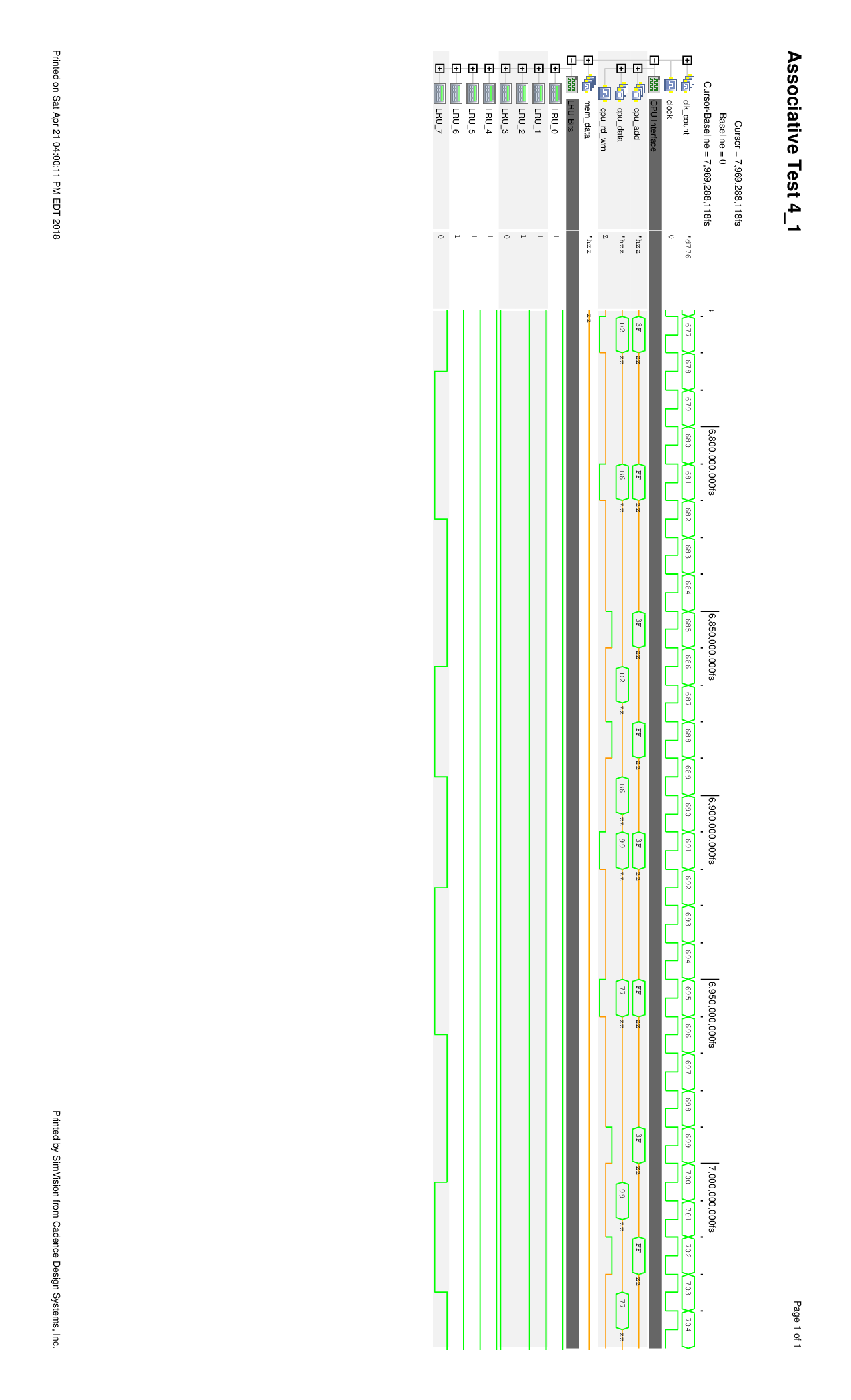


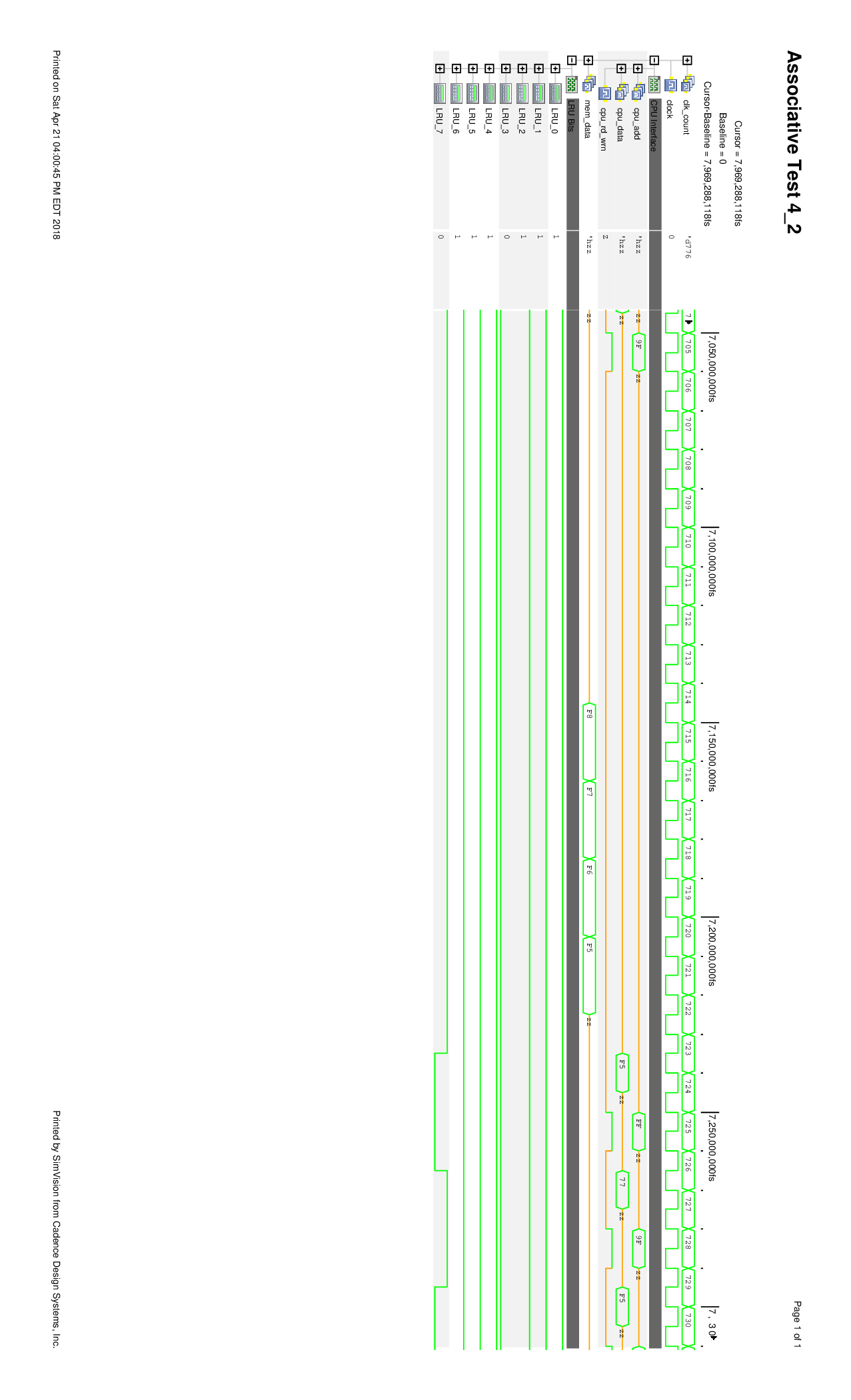


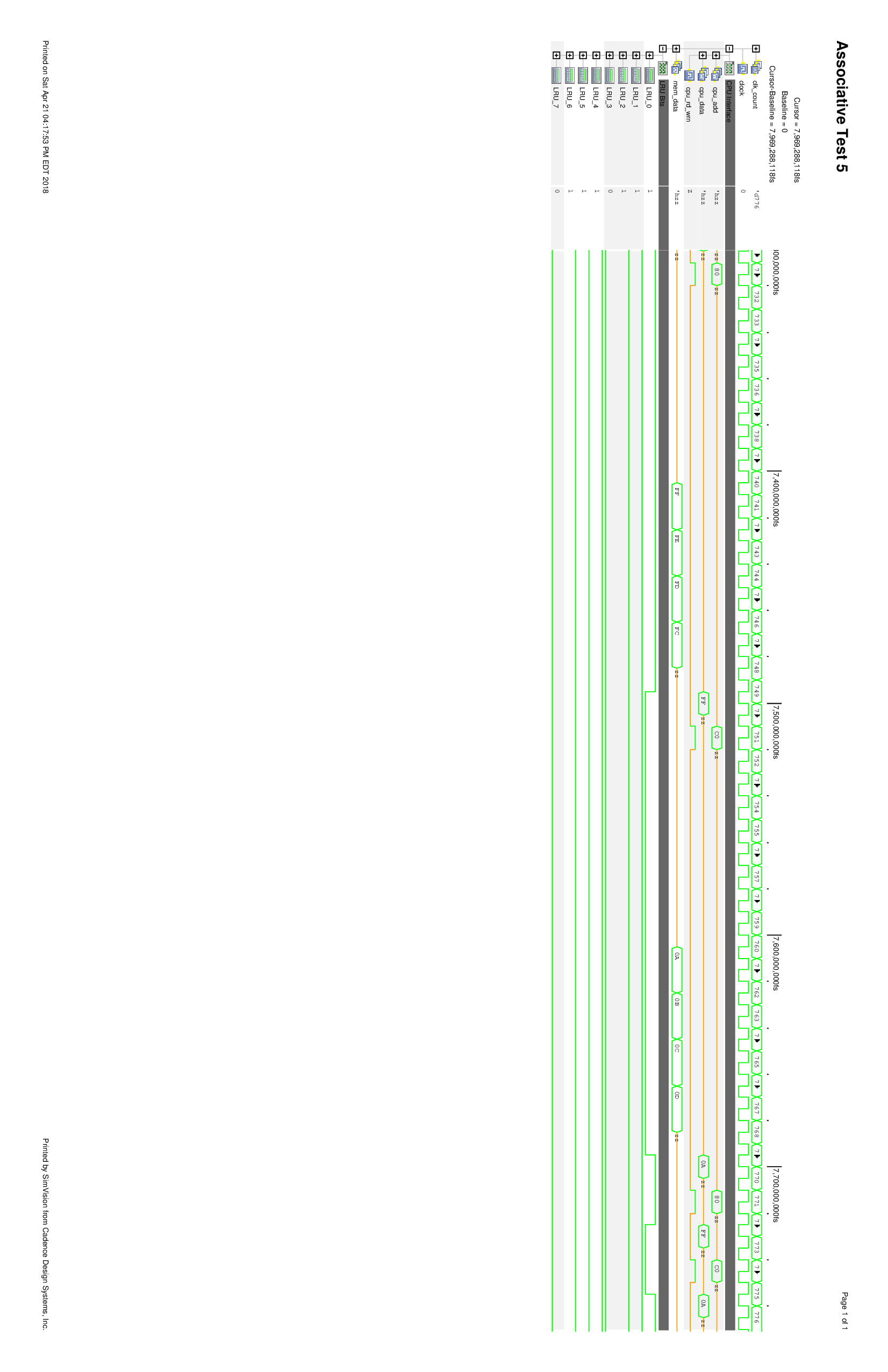












Appendix X: Source Code