ECE 559

Shiyu Li, Zhiyao Xie

Contents

Function Overview

- Simplifications:
  - Maximum Transfer Block Size is 1530Bytes = 12240Bits
  - CRC is attached only if the segmentation is performed, i.e., more than one code block is required.
  - Two possible block size: *(Might be different with project spec. but we followed the 3GPP Document and are proved by prof.)*
    - <= 1056bits
      - 1 small block, no CRC
    - > 1056bits and <=6144bits
      - 1 big block, no CRC
    - > 6144bits and <= (6144 + 1056 – 24*2) = 7152bits
      - [small block w/filling, big block] all with CRC
    - > 7152bits and <= 12,240 bits
      - [big block w/filling, big block] all with CRC

Function Overview

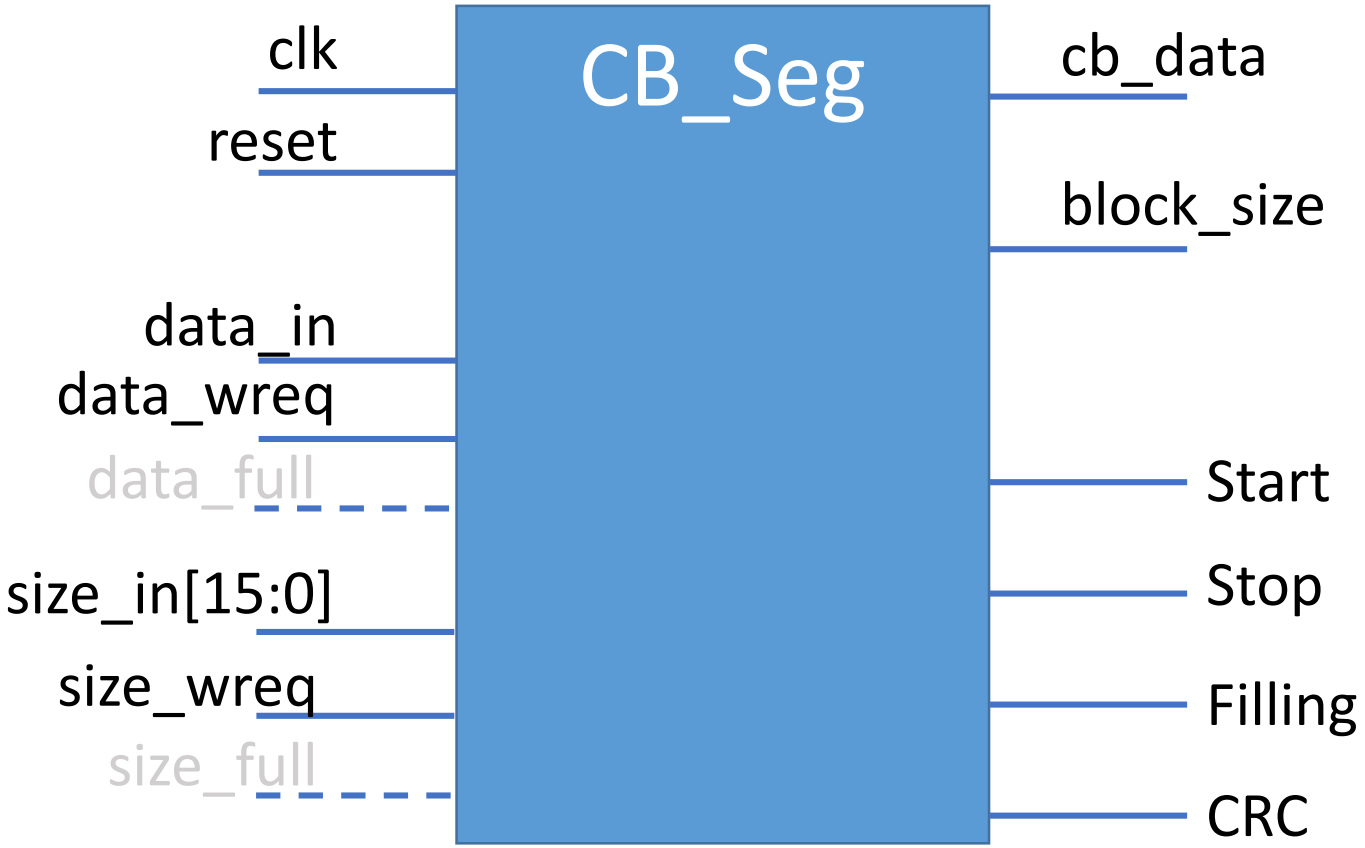
- Input interface assumption:
  - The size of the transfer block is asserted a cycle ahead of the data
  - The transmission of the data is performed with the same clock, same speed.
  - There’s no lag, no corruption.
  - Input data FIFO will not be full.
  - We don’t need to wait for the transfer block data during the process.

Function Overview

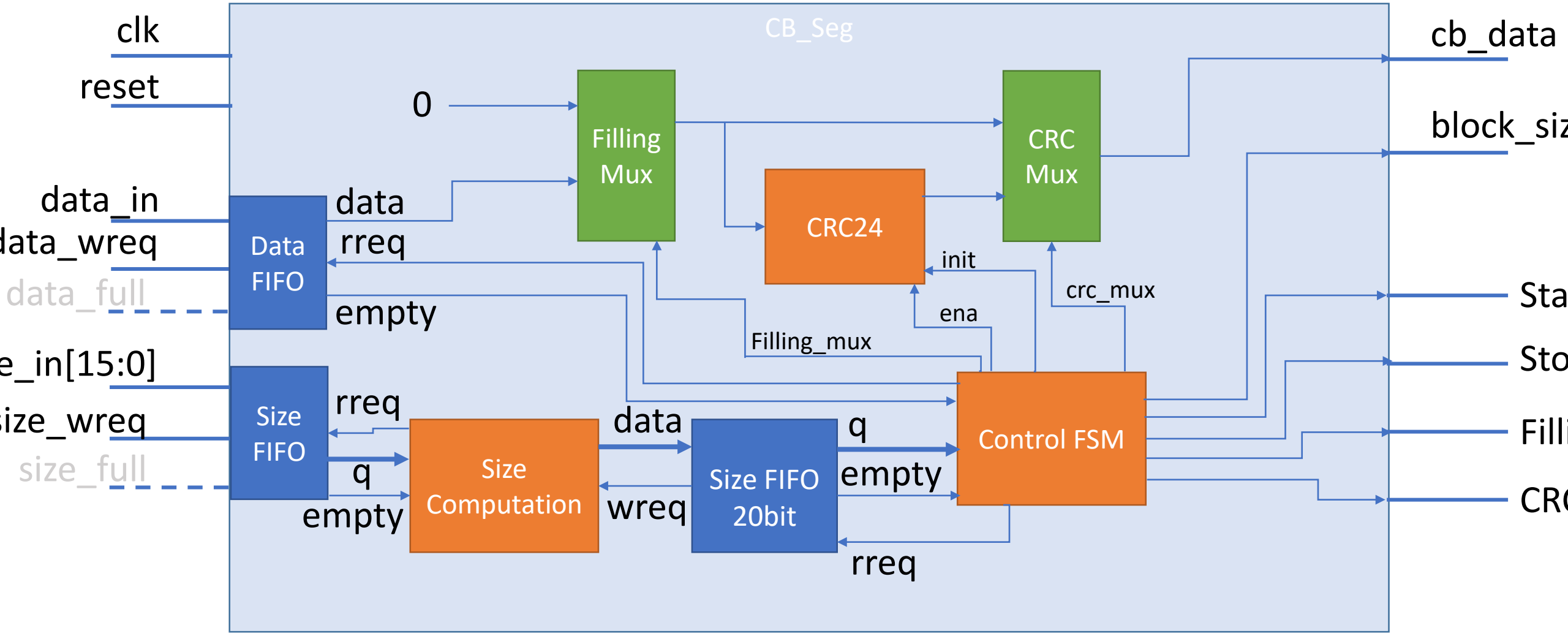
- Desired output timing:
  - **Start** Signal Asserted
  - Next Cycle, **block_size** signal asserted, 0 for small block, 1 for large block
  - The same cycle, the first bit of the data begin to transfer.
  - If current bit is the filling bit, the **filling** signal will be asserted **at the same cycle.**
  - If current bit is the CRC bit, the **crc** signal will be asserted **at the same cycle. Note: Filling bits also participate the computation of CRC as 0.**
  - **Stop** signal will be asserted the next cycle of the last bit of the data block(including CRC).

*Orange Part can be easily changed per request.

Top-Level Schematic



Top-Level Schematic



Sub-Module Design & Functional Simulation

- CRC24
- Control Path
- Block Size Computation

CRC24 Implementation

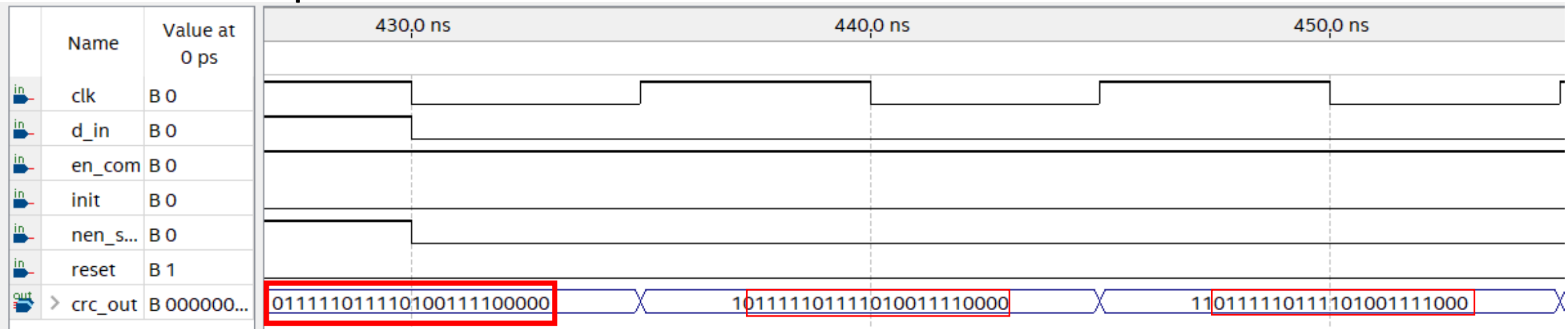
- $g_{CRC24A}(D) = [D^{24} + D^{23} + D^{18} + D^{17} + D^{14} + D^{11} + D^{10} + D^{7} + D^{6} + D^{5} + D^{4} + D^{3} + D + 1]$ and;
- $g_{CRC24B}(D) = [D^{24} + D^{23} + D^{6} + D^{5} + D + 1]$ for a CRC length $L = 24$ and;
- $g_{CRC16}(D) = [D^{16} + D^{12} + D^{5} + 1]$ for a CRC length $L = 16$.
- $g_{CRC8}(D) = [D^{8} + D^{7} + D^{4} + D^{3} + D + 1]$ for a CRC length of $L = 8$.

- Various GF for different kinds of block.
- For LTE Data Blocks, we use CRC24B.
- Modified based on the LFSR implementation of Homework2

CRC24 Sanity Check

Input Sequence: `0101010101010100000111110101010101010101`

Desired Ouput: `011111011111010011110000`



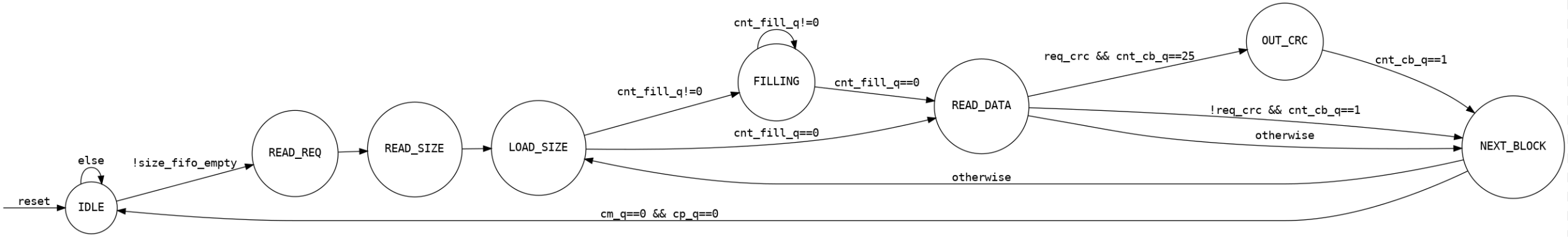
After de-assert shift function, we'll get the shifted output.

Compute Desire Result with: https://leventozturk.com/engineering/crc/

Control Path

- Port:
  - With Size FIFO:
    - Empty, Read Request, [19:0]Data
  - With Mux
    - Filling sel, Crc sel
  - With Data FIFO:
    - Empty, Read Request, Data
  - With CRC24:
    - Init, Compute_ena, nshift
  - Control Signals Output
    - Start, Stop, Filling, CRC

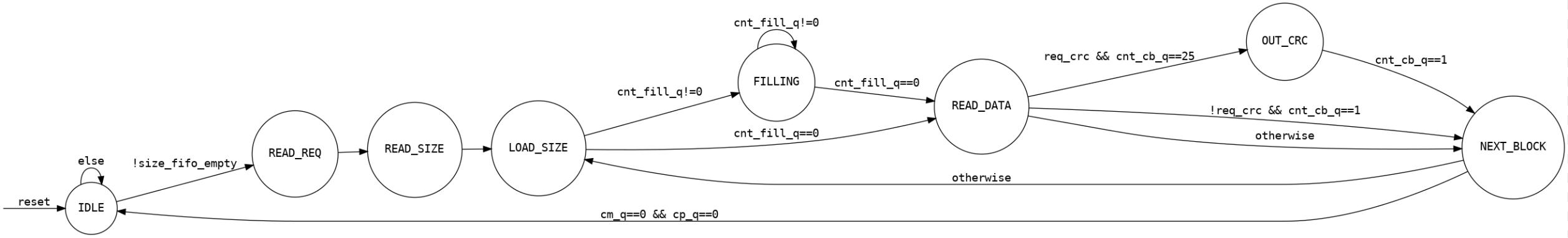
FSM Plot



- Follow the Encoding Scheme in Quartus' doc
- Fault-tolerant One-hot coding

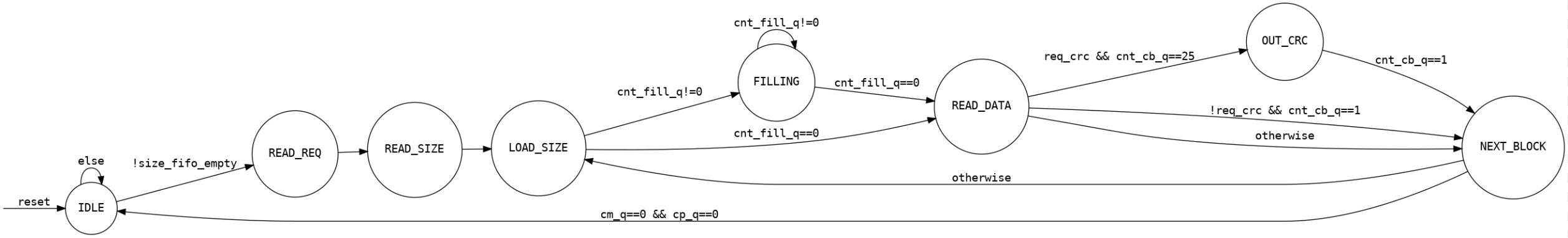
```
//FSM State Encoding
//Following the Quartus Encoding Scheme
parameter IDLE       =  8'b00000000,
          READ_REQ   =  8'b00000011,
          READ_SIZE  =  8'b00000101,
          LOAD_SIZE  =  8'b00001001,
          FILLING    =  8'b00010001,
          READ_DATA  =  8'b00100001,
          OUT_CRC    =  8'b01000001,
          NEXT_BLOCK =  8'b10000001;
```

FSM Behavior



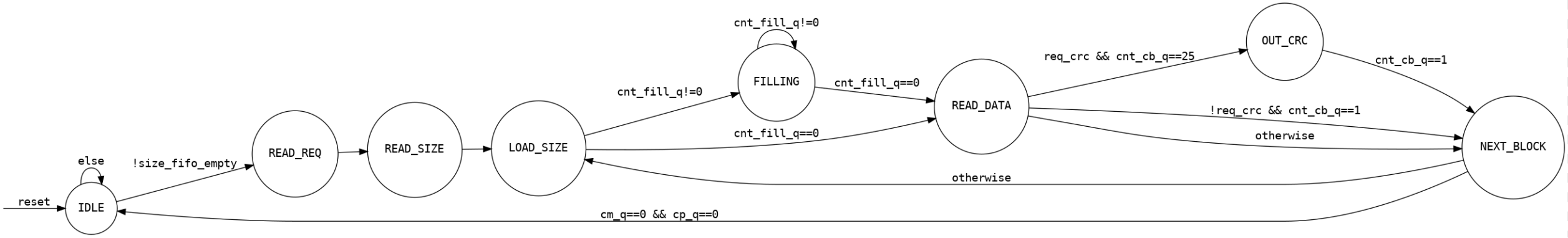
- IDLE
  - Initialize CRC register & crc_req register **[Moore]**
- READ_REQ
  - Assert rreq signal of the size FIFO **[Moore]**
- READ_SIZE
  - Load C+, C- into 2-bits register, Filling Bits to the counter. **[Moore]**
  - Based on C+ and C-, write crc_req register. **[Mealy]**

FSM Behavior



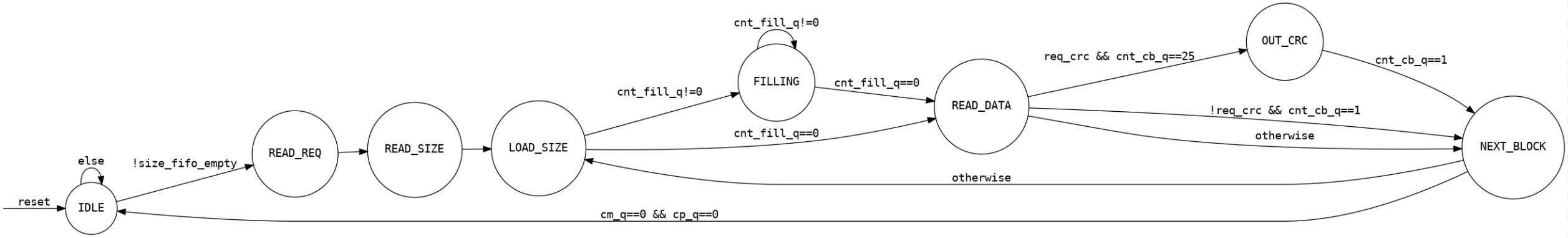
- LOAD_SIZE
  - Read C+ and C- register, decide the next code block to send. Small block first.
  - Load the next block size into the cb counter. **[Mealy]**
  - De-assert crc compute enable**[Moore]**
  - Update C+ and C- register. **[Mealy]**
  - Based on filling or not, pre-assert the rreq of data fifo **[Mealy]** and enable the cb counter. **[Moore]**
  - Assert Start signal. **[Moore]**

FSM Behavior



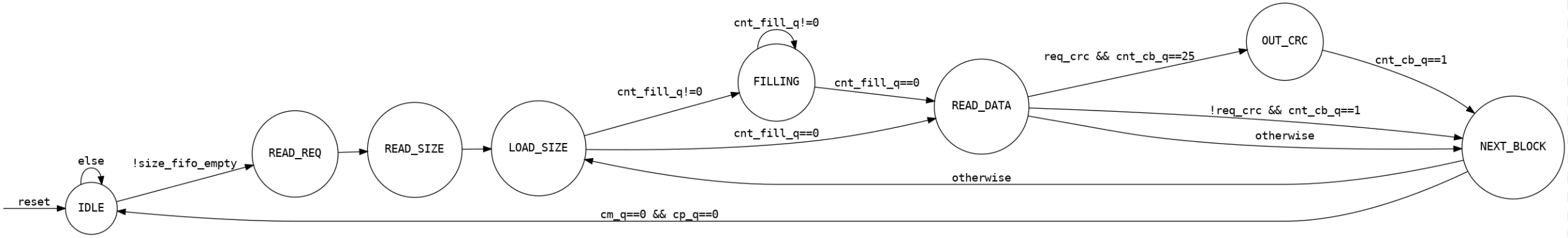
- FILLING
  - Assert filling_mux signal. **[Moore]**
  - Assert CRC_enable. **[Moore]**
  - Assert enable of the filling counter if it's not the last bit of filling. **[Mealy]***(To avoid underflow of the counter since we use the value of the counter later)*
  - Pre-assert rreq if it's the last bit of filling. **[Mealy]**

FSM Behavior



- READ_DATA
  - Assert filling_mux, CRC_enable, counter enable **[Moore]**
  - Assert rreq if it's not the last bit of the data. **[Mealy]**
- OUT_CRC
  - Let CRC register running in shift register mode.
  - Assert counter enable, crc_mux **[Moore]**
  - De-assert rreq, nshift_crc **[Moore]**

FSM Behavior

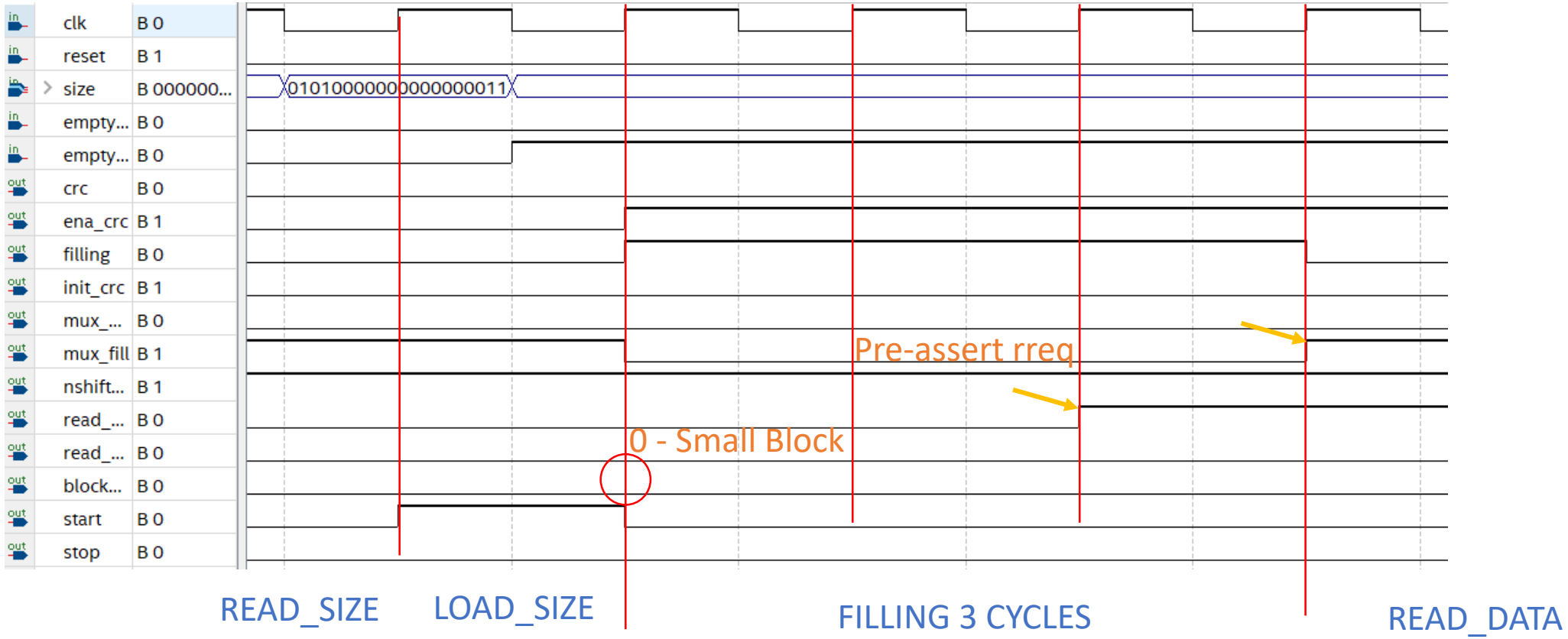


- NEXT_BLOCK
  - Initialize CRC register. **[Moore]**
  - Assert Stop Signal. **[Moore]**
  - Keep Assert CRC if required. **[Mealy]**

Functional Simulation

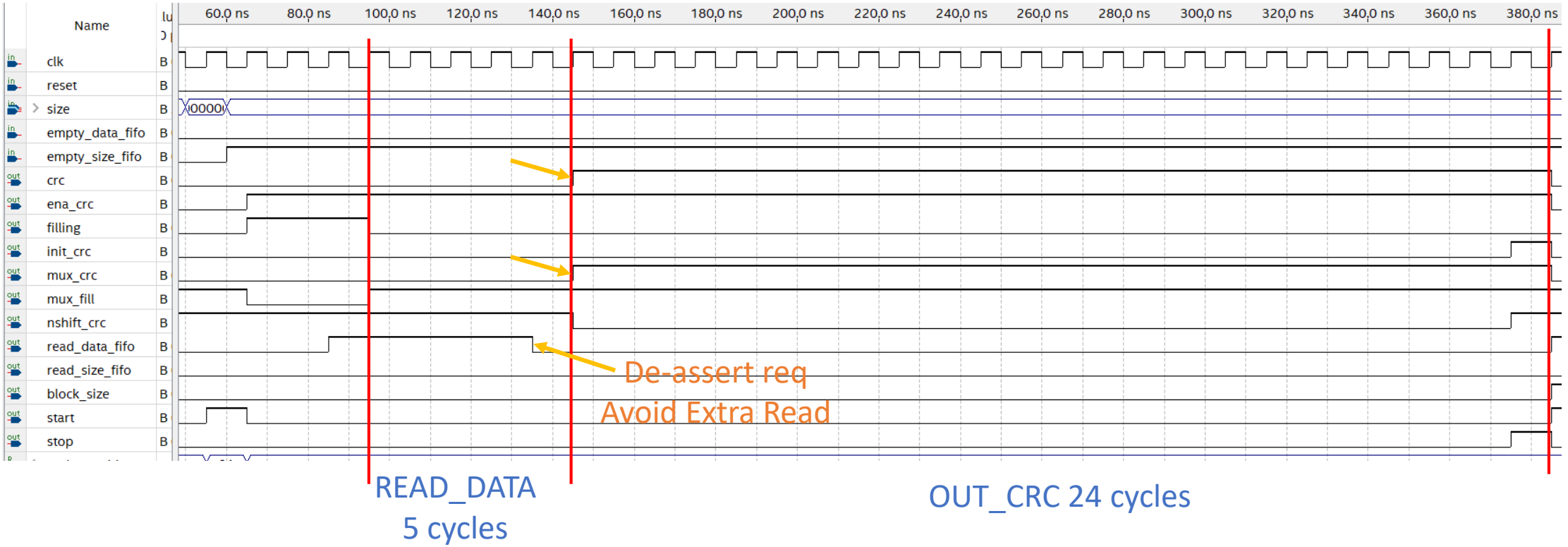
- Reduce Block size to 32 (small block) and 48(big block) for sanity check.
- Input Covers all the four situations mentioned above.

Functional Simulation



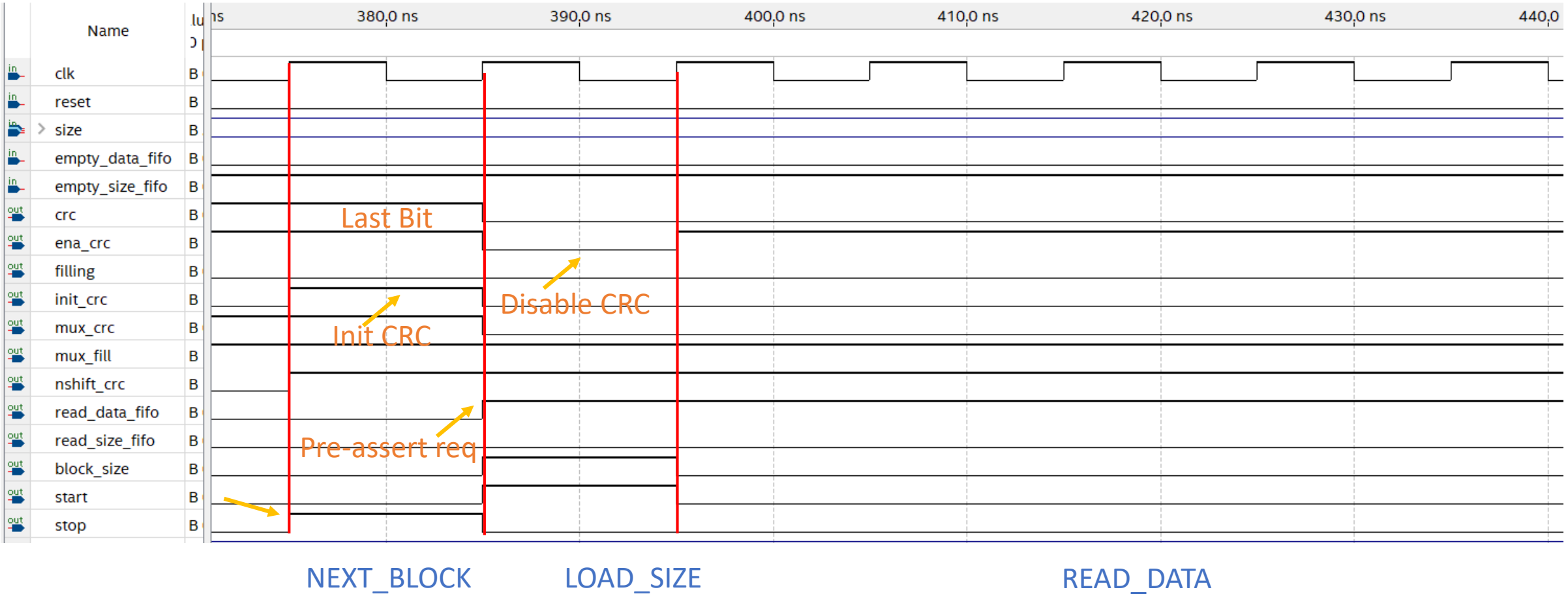
- 1 big block & 1 small block
- 3 filler bits

Functional Simulation

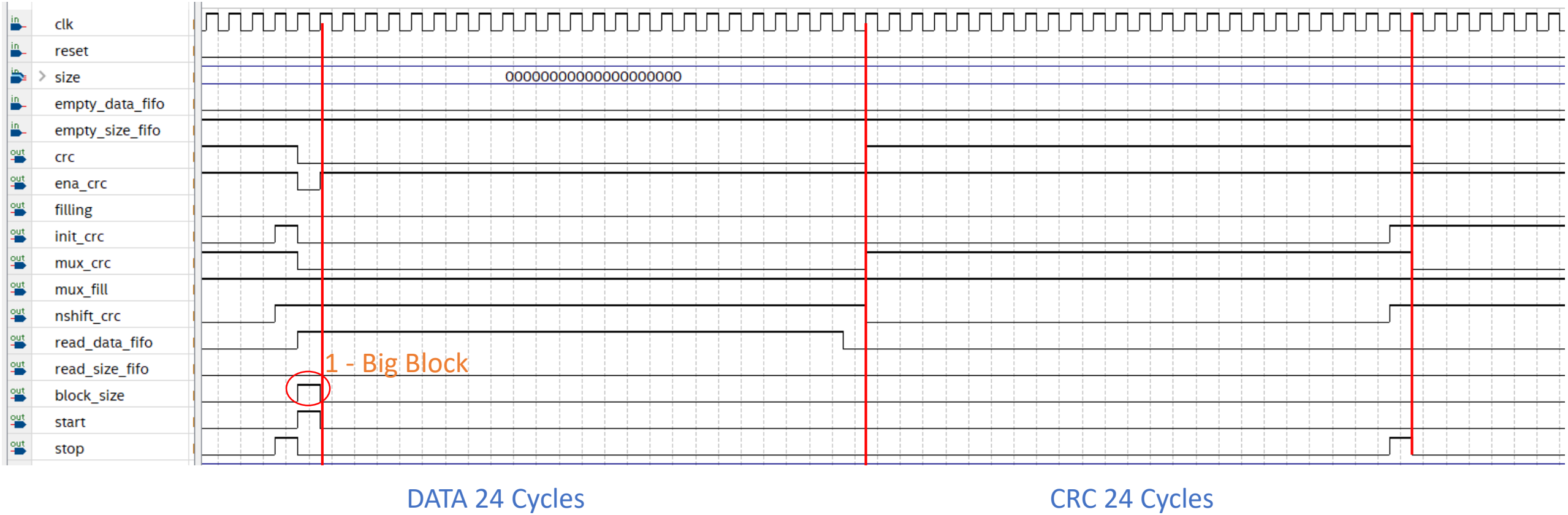


- 1 big block & 1 small block
- 3 filler bits

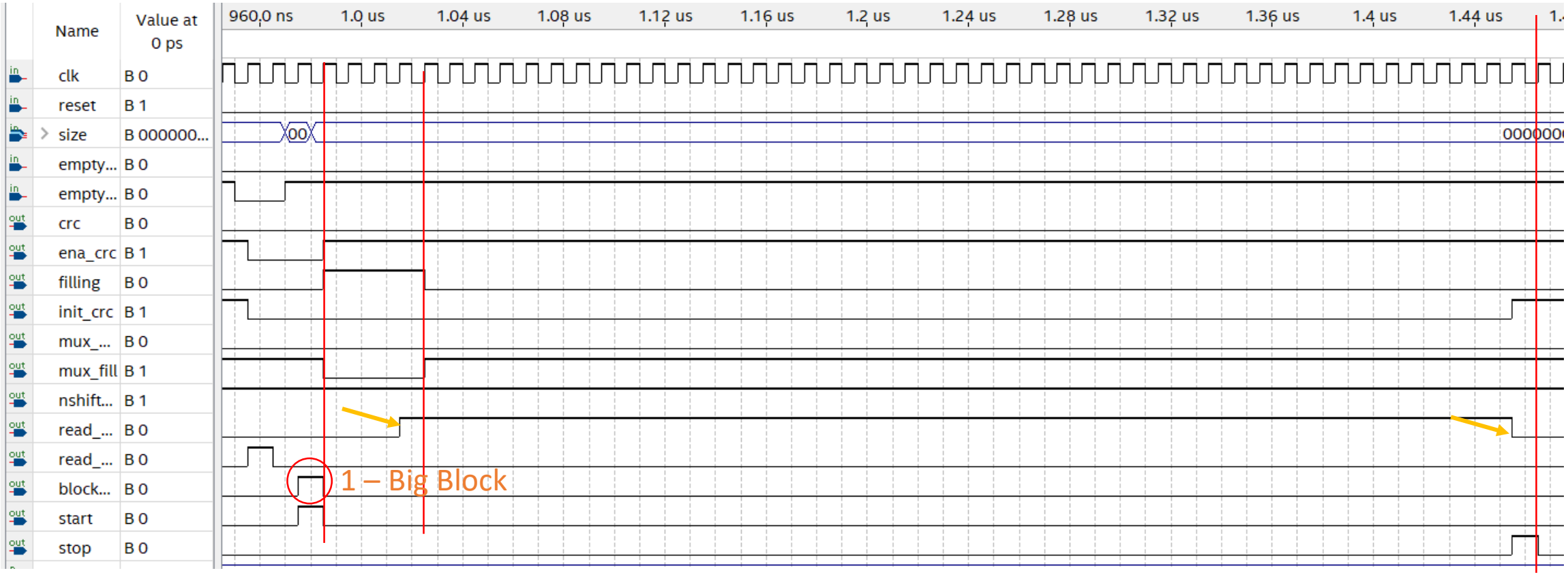
Functional Simulation



Functional Simulation



Functional Simulation



- 1 big block
- 4 filler bits

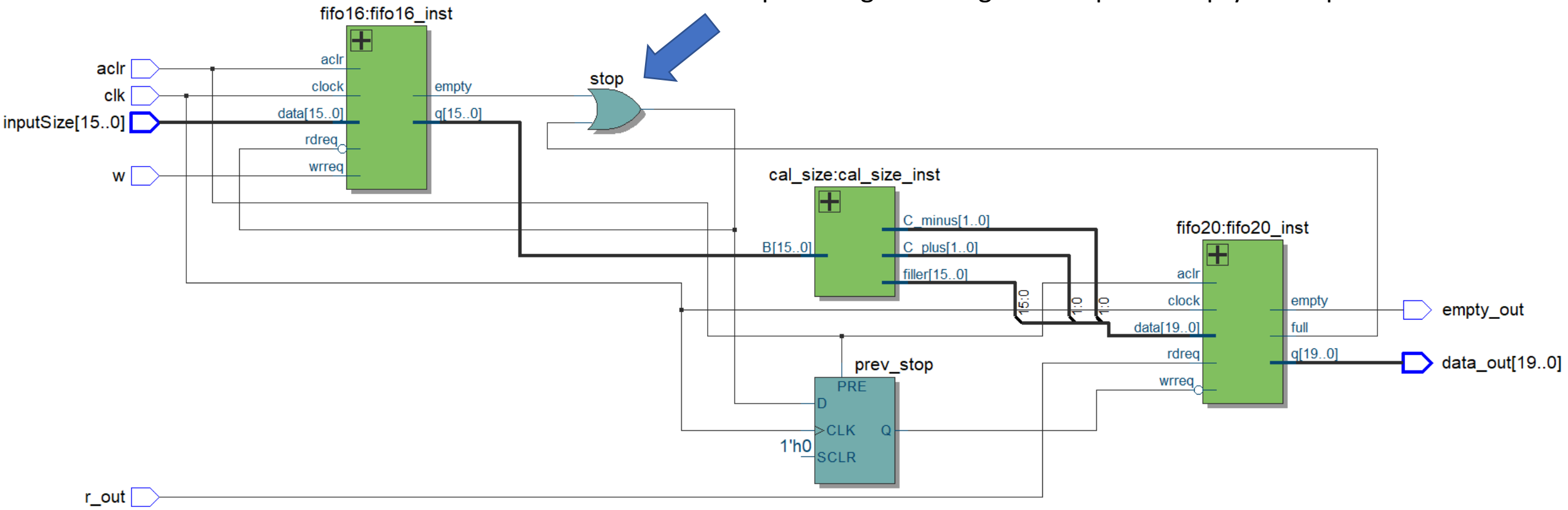
Block Size Computation

- Port:
  - Input size FIFO:
    - [15:0] size_in, write request,
  - (intra-subsystem) output block size FIFO:
    - [19:0] block size & numbers , read request, empty

- The 20-bit block size data include:
  - [1:0] number of large block (6144)
  - [1:0] number of small block (1056)
  - [15:0] filler bits
  - K+ & K- removed

Block Size Computation: RTL view

Stop reading & writing when input is empty **or** output is full.

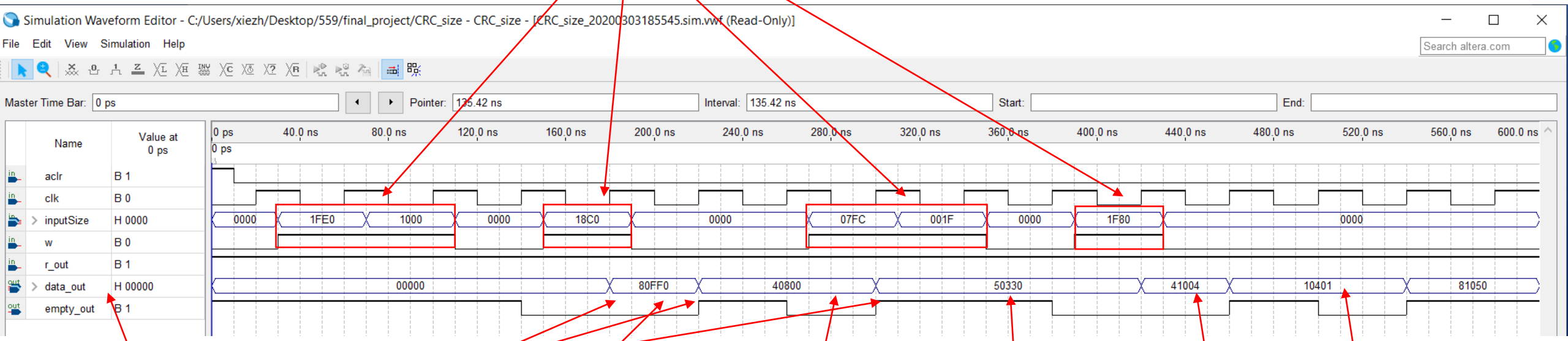


Functional Simulation

Four levels (of block size):

- 12240 (0x2fd0)
- 7152 (0x1bf0)
- 6144 (0x1800)
- 1056 (0x0420)

Writing input size



Output block sizes

Read output FIFO

Block size 1
Two large blocks
Filler = 0FF0
= 2FD0 - **1FE0**

Block size 2
One large block
Filler = 0800
= 1800 - **1000**

Block size 3
One large block
One small block
Filler = 0330
= 1BF0 – **18C0**

Block size 4
One large block
Filler = 1004
= 1800 - **07FC**

Block size 4
One small block
Filler = 0401
= 0420 – **001F**

Timing & Resource Analysis

- Max Frequency
  - 111.78 MHz
- Critical Path
  - Size FIFO in size computation

Logic utilization (in ALMs)	174 / 56,480 (< 1 %)
Total registers	151
Total pins	27 / 268 (10 %)
Total virtual pins	0
Total block memory bits	5,248 / 7,024,640 (< 1 %)
Total RAM Blocks	3 / 686 (< 1 %)
Total DSP Blocks	0 / 156 (0 %)

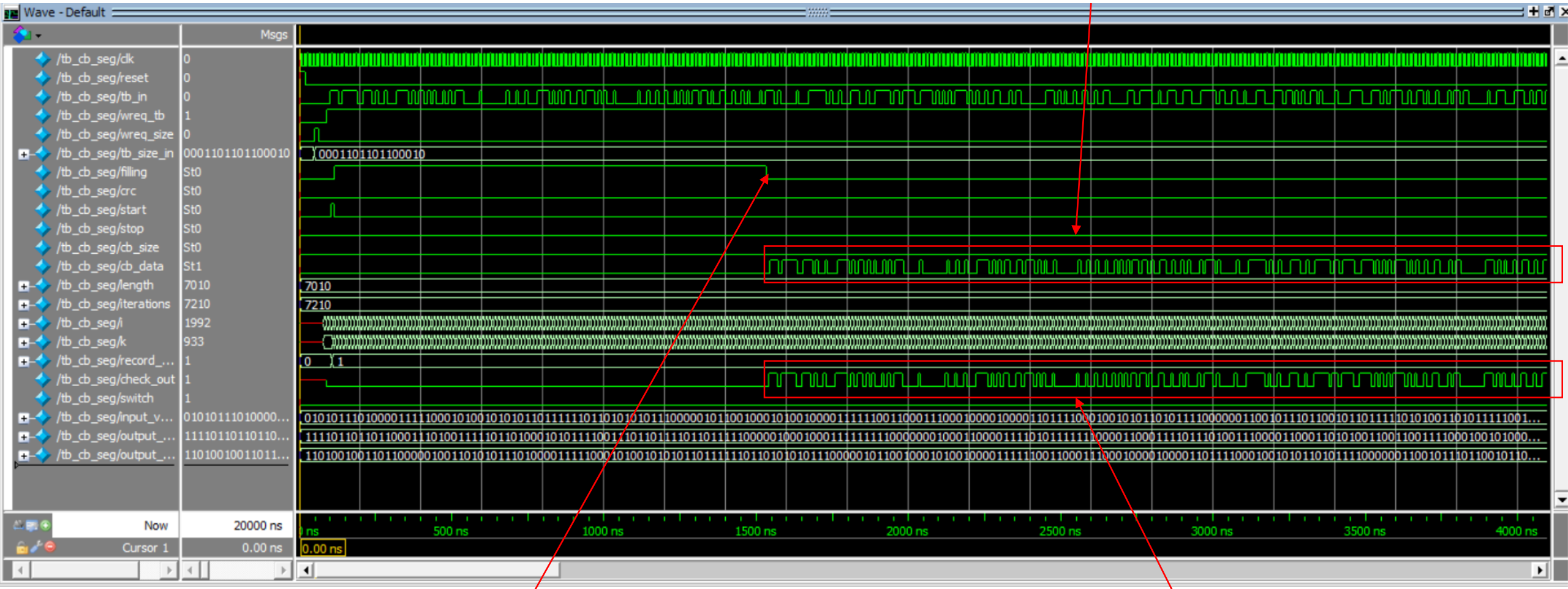
Simulation with Modelsim

- Python script generating simulation patterns
  - Generate random input vector with given size
  - Calculate block size & filler size
  - Generate multiple segmented blocks according to those sizes
  - Calculate CRC for each block
  - Generate correct outputs (MSB -> CRC + segmented block + *filler -> LSB)

- Run modelSim through testbenches
  - Feed the input vector (one bit/cycle)
  - Print the waveform of correct outputs according to control signals
  - Compare the waveform of outputs with correct ones

Simulation with Modelsim

Output data



Filler flag switch to 0

Correct output vector generated from python script