Frame Aligner Verification Plan

YAKIR AQUA

NOV 2024

Introduction

- The Frame Aligner is a component in serial communication systems. Its role is to detect and synchronize incoming data frames by identifying specific header patterns. It ensures that data is received correctly for further processing.

Behavior Description of the Aligner

- Receives an 8-bit serial data stream and searches for specific header patterns.
- After detecting a valid header, collects the frame's payload.
- Provides a synchronization signal through the frame detection signal.
- Enters synchronization mode after correctly identifying 3 frames.
- Exits synchronization mode after counting 48 bytes without detecting a valid header.
- Valid Frame: 16-bit header and 80-bit payload (10 bytes).
- POSSIBLE HEADER PATTERNS:

  - Header Type 1 (HEAD_1) LSB = 0xAA, MSB = 0xAF

  - Header Type 2 (HEAD_2) LSB = 0x55, MSB = 0xBA

Assumptions

- Data Rate: The data is received at a rate of one byte per clock.
- Known Header Patterns: The only possible headers are AFAA or BA55.
- Valid Frame Length: Consists of 12 bytes (2 bytes for the header and 10 bytes for the payload).
- Data Continuity: No interruptions or data loss in the incoming stream
- Invalid Frame: Comprises a random number of bytes (2 bytes for the header + a random number of payload bytes).

Verification Goals

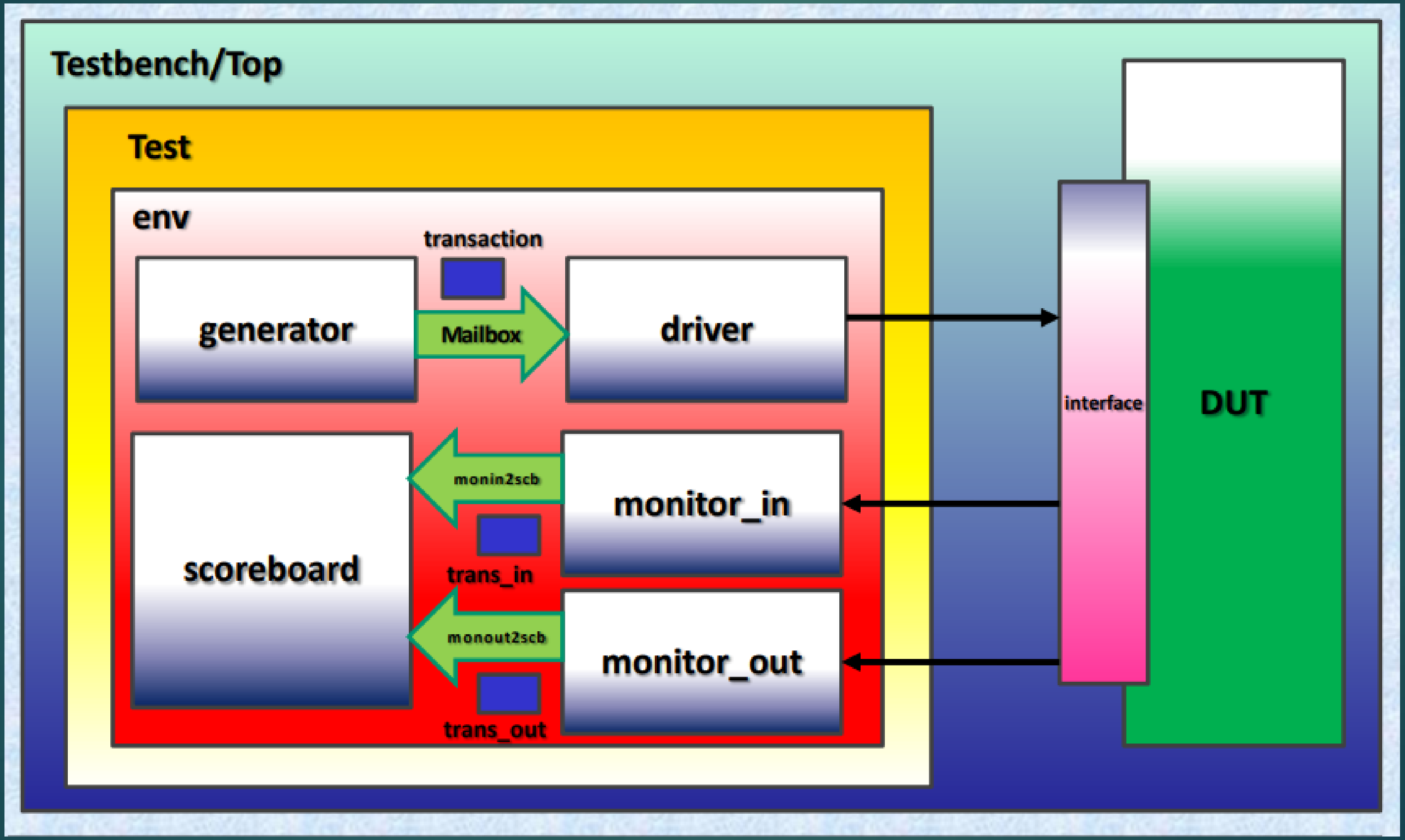
- Complete functional verification of the Frame Aligner according to system requirements.
- Ensure the Frame Aligner correctly identifies and synchronizes frames.
- Test the stability and error response of the Frame Aligner under various input conditions.
- Validate the Frame Aligner's function with different sequences of frames.
- Verify the module's behavior under error scenarios and extreme conditions.

Verification Environment

The verification environment is based on SystemVerilog

- ***Interface*** - Defines the interface between the simulator and the Frame Aligner and includes the necessary signals.
- ***Transaction*** - Represents the data transaction passing through the Frame Aligner.
- ***Generator*** - Generates transactions either randomly or deterministically according to the test plan.
- ***Driver*** - Pushes the input to the Frame Aligner.
- ***Monitors*** - Track the signals/data entering and exiting the Frame Aligner.
- ***Checker*** - Uses assertions to verify the Frame Aligner's behavior and ensures the outcome matches expectations.
- ***Scoreboard*** - Calculation of the expected result based on the input signals of the Device Under Test (DUT) and comparing them to the Device Under Test (DUT)'s output signals.
- ***Environment*** - Defines all the components of the environment and ensures proper interconnections.
- ***Random Test*** - Test file that defines test scenarios either randomly or based on required test types.
- ***Testbench*** - Organizes the verification environment, including connecting the Frame Aligner and simulating the environment.
- ***Device Under Test (DUT)*** - Represents the Device Under Test (DUT) (the Frame Aligner itself) used for testing.

Block diagram



Key Functional Tests

- Detection of valid headers.
- Rejection of invalid headers.
- Synchronization and loss of synchronization modes.
- Tracking byte positions and reset response.

Coverage Plan

Coverage Goals

- Cover all header types (HEAD_1, HEAD_2, ILLEGAL).
- Cover all possible payload lengths (HEAD_1 and HEAD_2 always 10 bytes, ILLEGAL between 0 and 47 bytes).
- Test all data input combinations to the rx_data[0:255] range.
- Cover all frame byte positions [0:10].
- Ensure frame detection signal reaches both low and high states across four coverage goals above.
- Achieve full state machine coverage (Code Coverage).
- Evaluate system response to extreme and unexpected scenarios.

Coverage Metrics

- 100% coverage of code lines, expressions, and branches.
- Coverage of all possible header types.
- Coverage of consecutive valid and invalid Frames when frame detection signal is high and when frame detection signal is low.
- All test cases execute correctly without critical bugs.
- All assertions pass successfully.

Test Plan

Test Description	Test purpose	Expected Outcome
Send 5 frames with HEAD_1	To check if the DUT enters synchronization mode with header type HEAD1. .	Frame detect = 1
Send 4 frames with an ILLEGAL header	To check if the DUT exits synchronization mode.	Frame detect = 0.
Send 5 frames with HEAD_2	To check if the DUT enters synchronization mode with header type HEAD2.	Frame detect = 1
Mix of 3 HEAD_1 and HEAD_2 headers	To check that there is no impact on the sequence of valid header types for entering synchronization mode.	Frame detect = 1
Send correct MSB as LSB and correct LSB as MSB from both header types	To check if there is an impact on the order of data	No valid header recognition.
Send headers with reversed bit order	To check if there is an impact on the order of data	No valid header recognition.
Send only correct MSB in the middle of a valid/invalid frame	To check if there is an impact on the component when part of the header information is sent in incorrect positions.	No impact on the system.

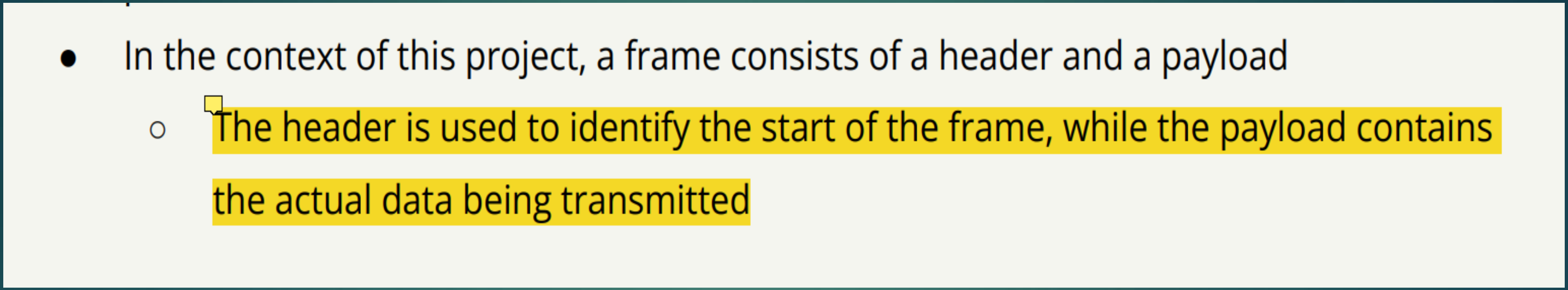
Test Description	Test purpose	Expected Outcome
Send correct LSB from both header types and incorrect MSB	To ensure that only a correct header is accepted.	No valid header recognition.
Send incorrect LSB and correct MSB from both header types	To ensure that only a correct header is accepted.	No valid header recognition.
When frame_detect is high, check that after 44 bytes without a valid header, and a valid header in bytes 45 and 46	To check an edge case.	Frame detect = 1
When frame_detect is high and valid header in bytes 46 and 47	To check an edge case.	Frame detect = 0
Send correct LSB in the middle of an invalid frame	To check the system's response when the LSB is sent.	fr_byte_position rises to 1 on the next clock cycle and resets after an additional clock cycle.
Send correct LSB in the middle of a valid frame	To check the system's response when the LSB is sent.	fr_byte_position continues counting.
Payload contains valid HEAD_1 or HEAD_2 headers within an invalid frame when frame_detect is high within a 45-byte interval without a valid header	To check if there is an impact on the system when a header is within the payload of an illegal frame, and how this affects in a specific scenario.	Frame detect = 1

Test Description	Test purpose	Expected Outcome
Payload contains valid HEAD_1 or HEAD_2 headers within an invalid frame when frame_detect is low, followed by 3 valid frames	To check if there is an impact on the system when a header is within the payload of an illegal frame, and how this affects in a specific scenario.	Frame detect = 0
Payload contains valid HEAD_1 or HEAD_2 headers within an invalid frame when frame_detect is low, followed by 2 valid frames starting immediately after 10 clock cycles from the valid header in the invalid frame's payload	To check if there is an impact on the system when a header is within the payload of an illegal frame, and how this affects in a specific scenario.	Frame detect = 1
Send an invalid frame with 3 valid frames in the payload	To check if there is an impact on the system when a header is within the payload of an illegal frame, and how this affects in a specific scenario.	frame_detect rises to 1 or remains unchanged if already high.
Send a valid header within the payload of a valid frame	To check if there is an impact on the system when a header is within the payload of an illegal frame, and how this affects in a specific scenario.	No impact on the system.
Send 2 consecutive valid headers followed by an invalid frame with only valid lsb.	To check an edge case.	frame_detect does not rise to 1, and if it is already high, it remains unchanged.
valid LSB is sent twice, followed by a valid MSB	Testing the component's detection mechanism.	the DUT will not recognize a valid header because it transitions to an MSB detection state, thereby missing the sequence of a valid header
A 49-byte frame filled with valid LSBs and MSBs of both header types in random locations.	To check the impact on the system when headers or parts of headers are sent in incorrect positions.	It depends according to the combinations received.

Gaps between the spec and the design, and conclusions

My assumption is that the design is bug-free, but there are gaps in the spec. Therefore, whenever there was a mismatch between the scoreboard results and the DUT, I corrected the scoreboard.

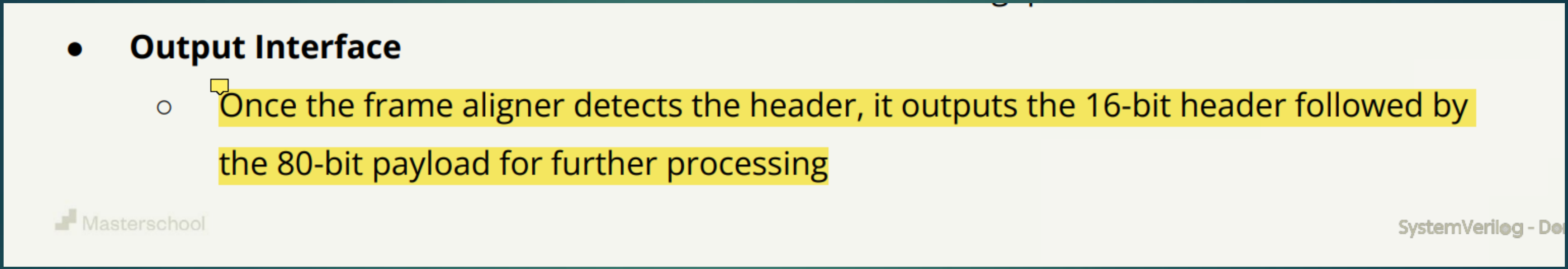
Gaps between the spec and the design



- In the context of this project, a frame consists of a header and a payload
  - The header is used to identify the start of the frame, while the payload contains the actual data being transmitted

- **The Device Under Test (DUT) identify only valid frame!**
- **Only valid frame contains a 2-byte header and a 10-byte payload!**
- **An invalid frame was not defined for us. For convenience, I defined it as follows: a 2-byte header and a dynamic array with a payload between 0 and 47 bytes.**

Gaps between the spec and the design



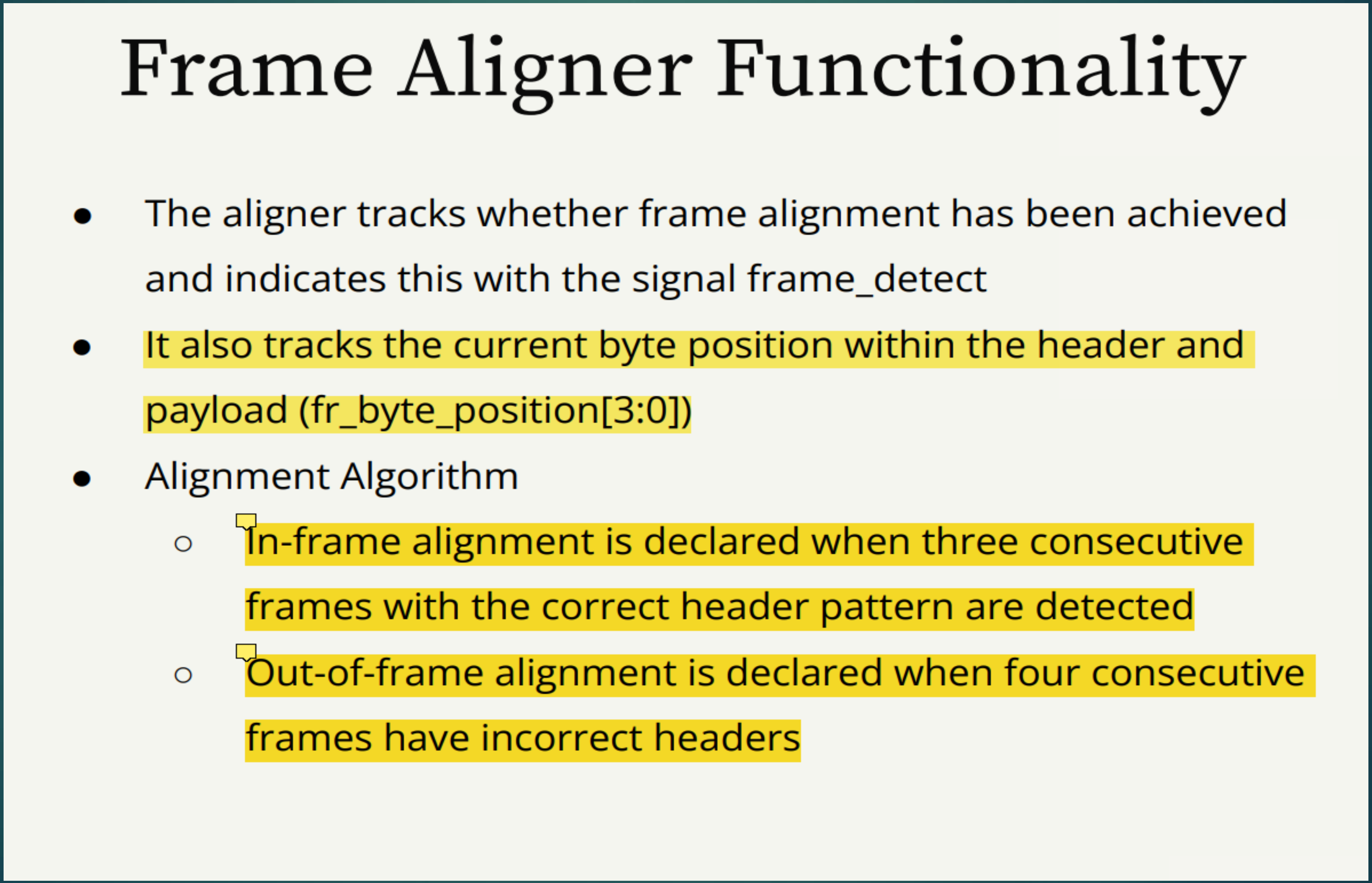
- **Output Interface**
  - Once the frame aligner detects the header, it outputs the 16-bit header followed by the 80-bit payload for further processing

Masterschool

SystemVerilog - Do

- An output interface is defined in the spec, but it does not exist in the design!

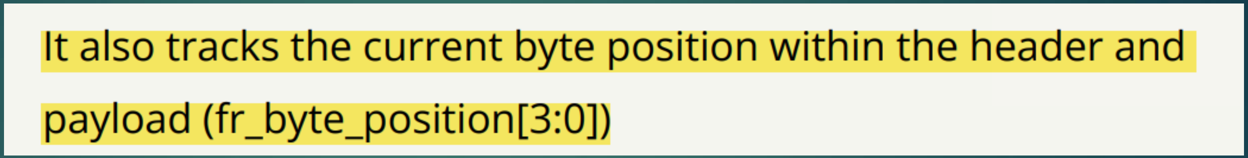
Gaps between the spec and the design

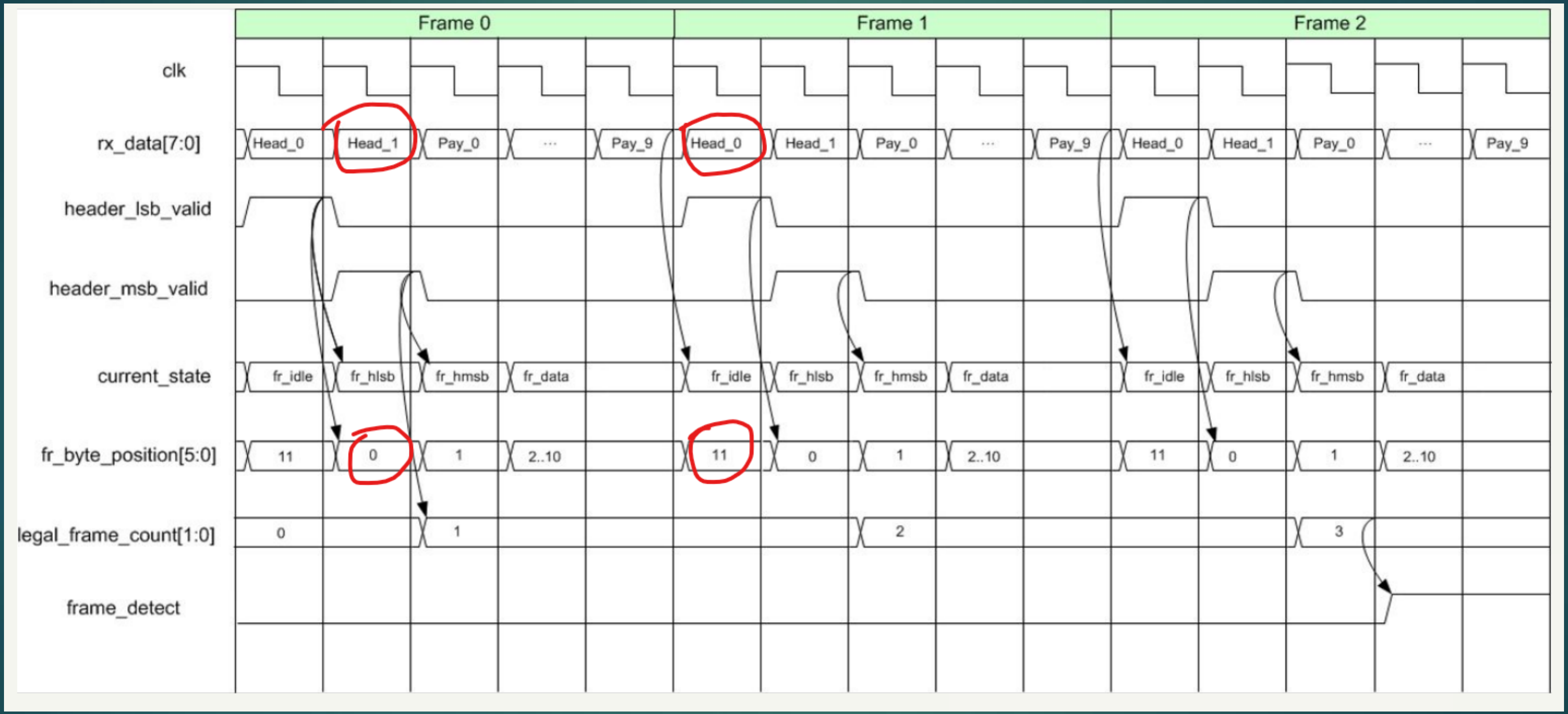


Frame Aligner Functionality

- The aligner tracks whether frame alignment has been achieved and indicates this with the signal frame_detect
- It also tracks the current byte position within the header and payload (fr_byte_position[3:0])
- Alignment Algorithm
  - In-frame alignment is declared when three consecutive frames with the correct header pattern are detected
  - Out-of-frame alignment is declared when four consecutive frames have incorrect headers

Gaps between the spec and the design

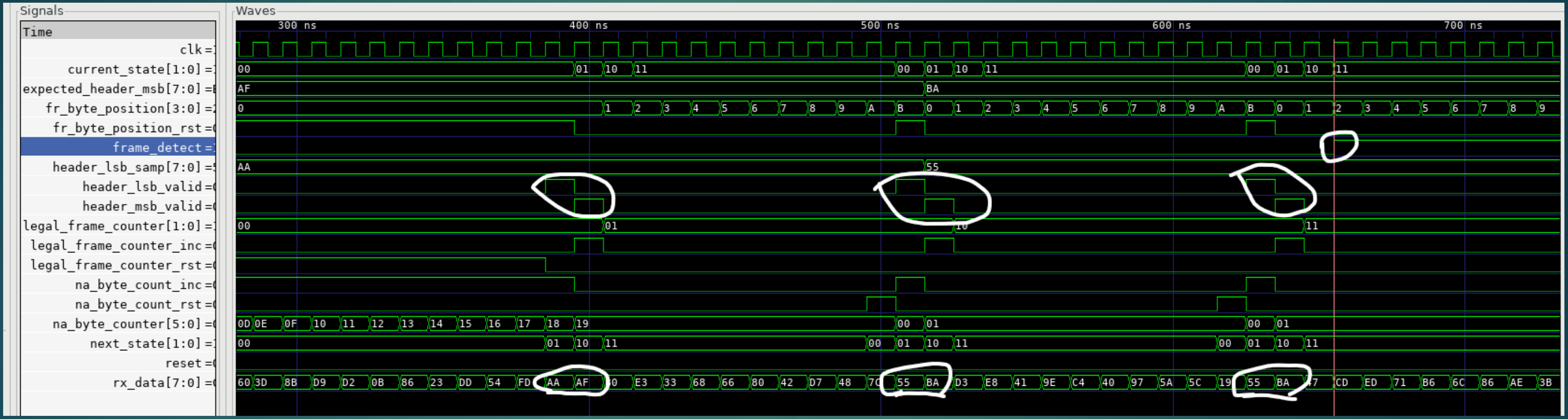

It also tracks the current byte position within the header and payload (fr_byte_position[3:0])



- **The byte count for a valid frame starts from the MSB of the valid header and ends at the LSB of the next header!**

Gaps between the spec and the design

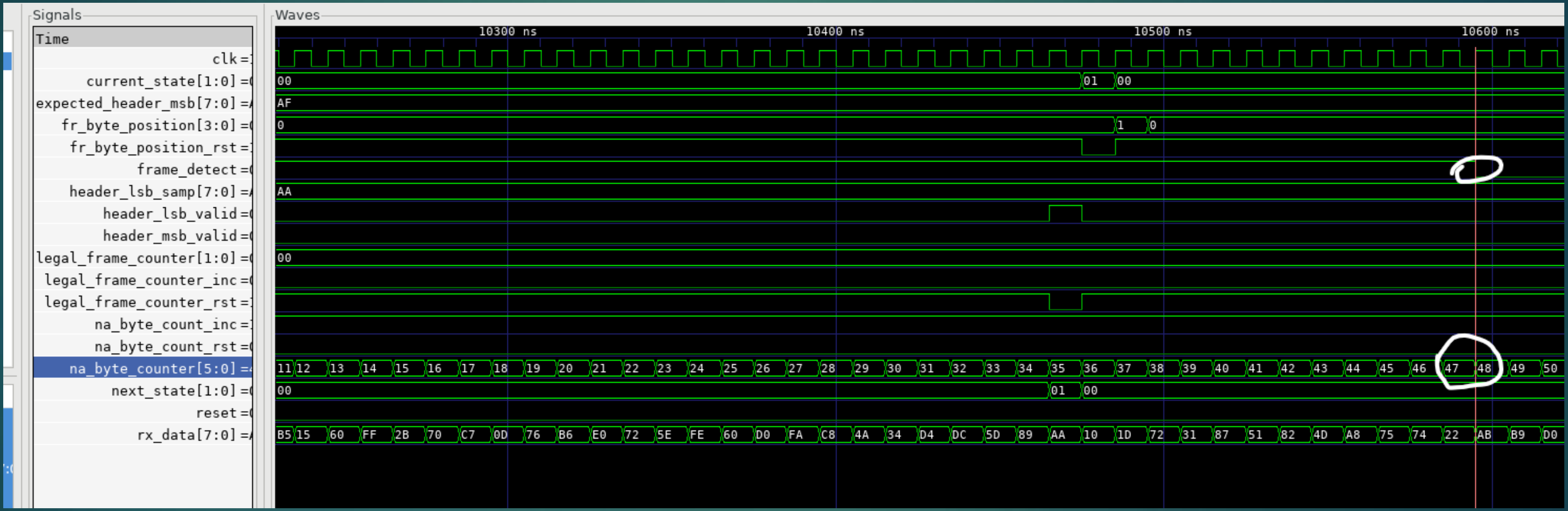
- In-frame alignment is declared when three consecutive frames with the correct header pattern are detected



- **When the DUT detects 3 consecutive valid frames after one clock cycle, it enters synchronization mode!**

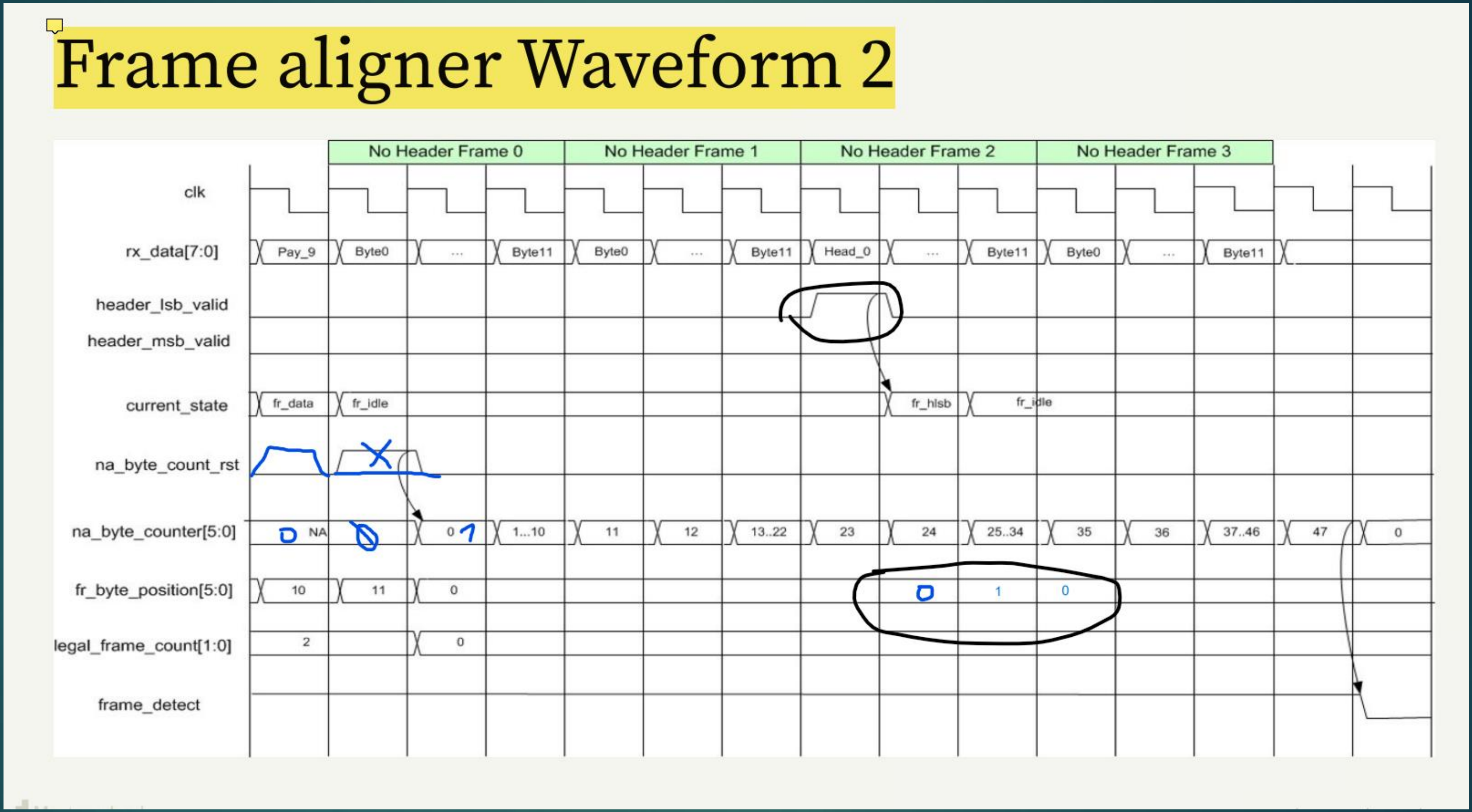
Gaps between the spec and the design

- Out-of-frame alignment is declared when four consecutive frames have incorrect headers



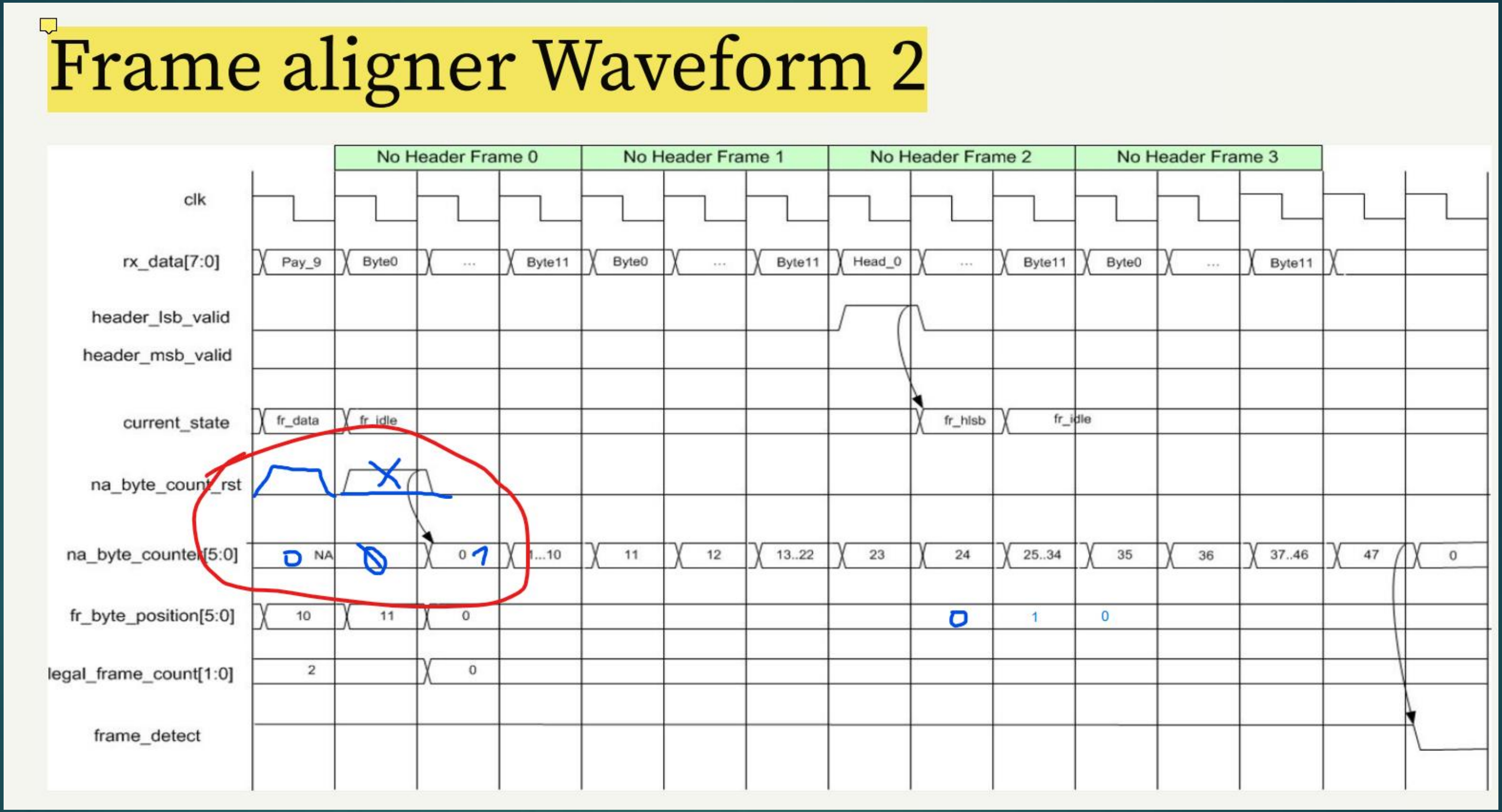
- **When 48 bytes are counted without a valid header, the DUT exits synchronization mode!**

Gaps between the spec and the design



- If a valid LSB is detected, the byte counter for a valid frame resets in the next clock cycle. If a valid MSB is also detected, it continues to increment as usual. If not, it increments to one in the next clock cycle and then resets afterward.

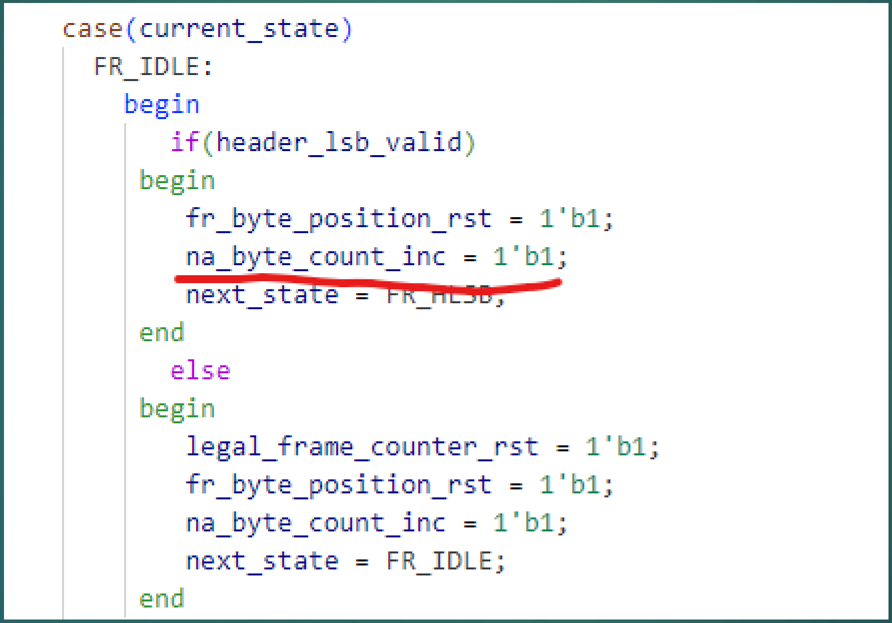
Gaps between the spec and the design



- After a valid frame, the reset signal for the counter of bytes without a header is set to 1 at the last byte of the payload. The reset occurs at the LSB of the next header, and the counter increments starting from the MSB.

Gaps between the spec and the design

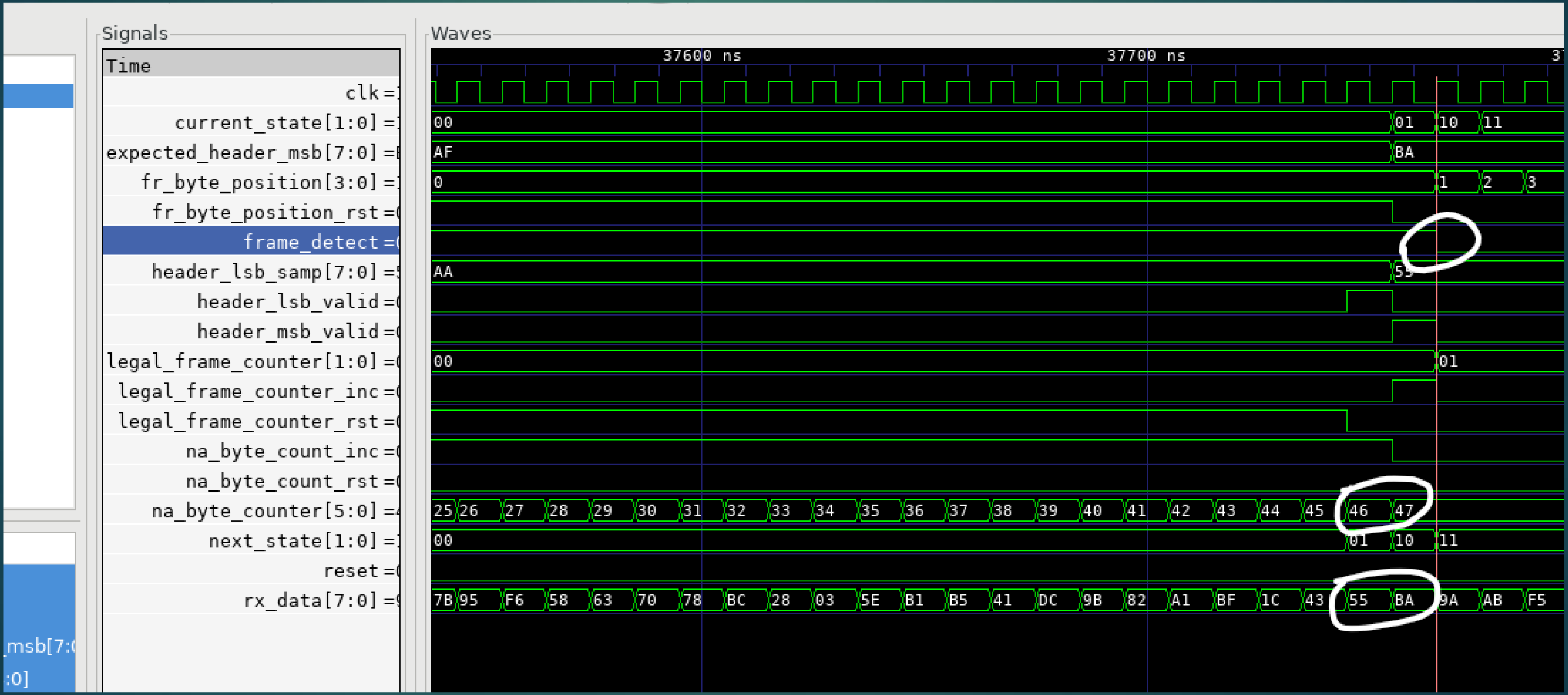
- **The last opportunity to remain in synchronization mode is if the DUT detects the start of a valid header at byte number 45. This is because if a valid LSB is detected, the DUT increments the counter for bytes without a header.**



```
case(current_state)
  FR_IDLE:
    begin
       if(header_lsb_valid)
     begin
        fr_byte_position_rst = 1'b1;
        na_byte_count_inc = 1'b1;
        next_state = FR_HLSB;
     end
       else
     begin
        legal_frame_counter_rst = 1'b1;
        fr_byte_position_rst = 1'b1;
        na_byte_count_inc = 1'b1;
        next_state = FR_IDLE;
     end
```

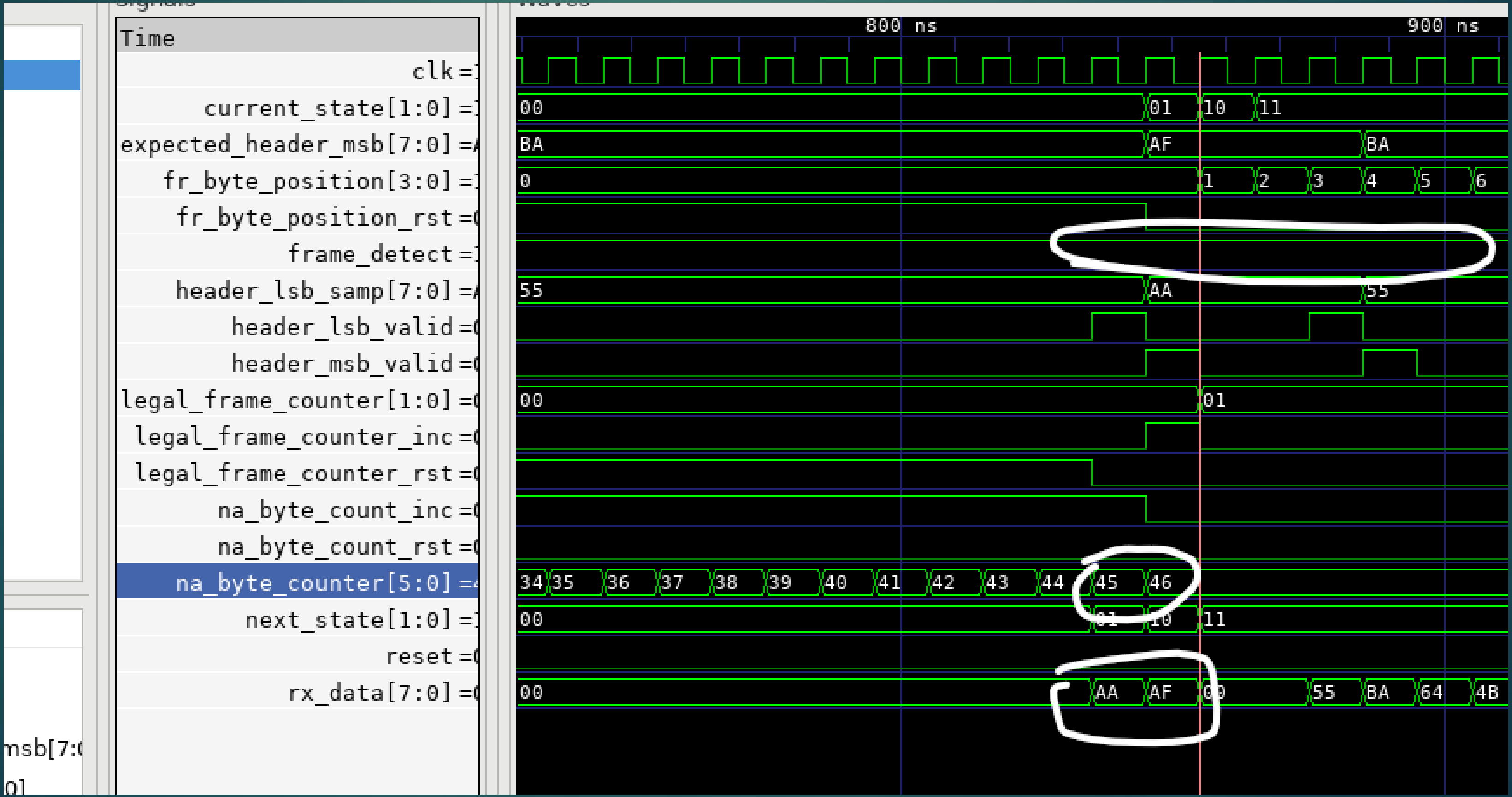
Gaps between the spec and the design

- The last opportunity to remain in synchronization mode is if the DUT detects the start of a valid header at byte number 45. This is because if a valid LSB is detected, the DUT increments the counter for bytes without a header.



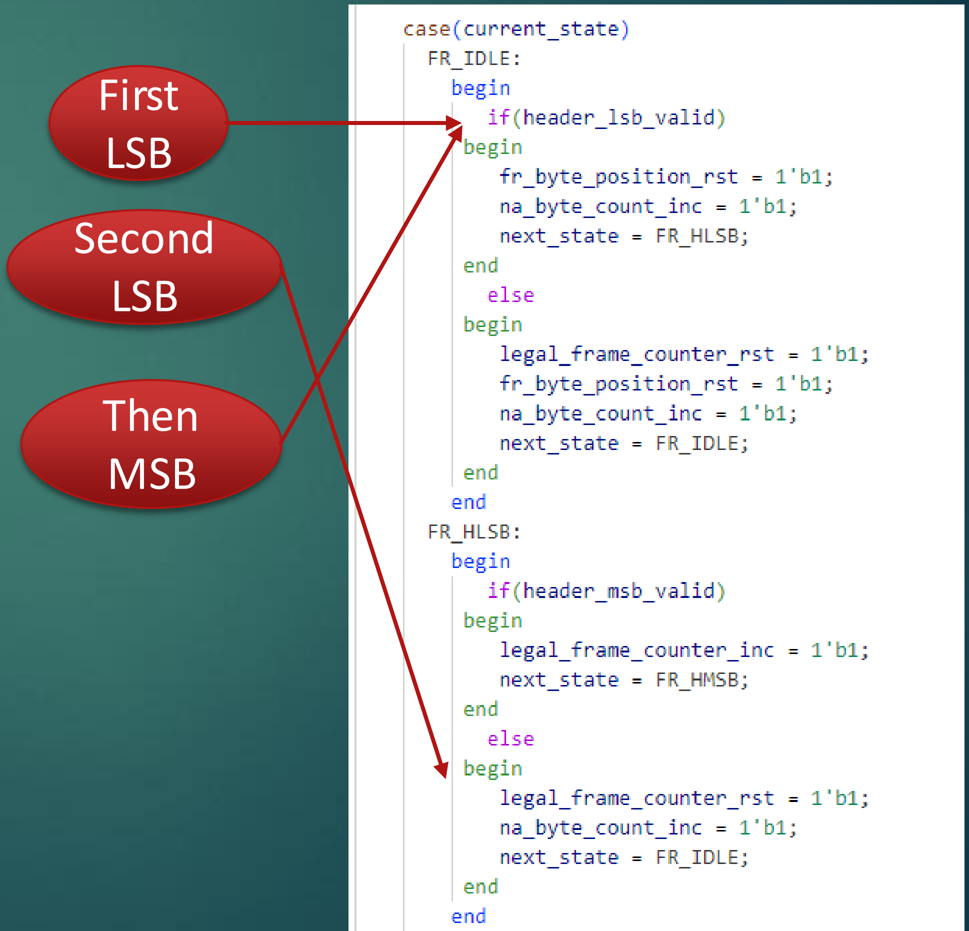
Gaps between the spec and the design

- The last opportunity to remain in synchronization mode is if the DUT detects the start of a valid header at byte number 45. This is because if a valid LSB is detected, the DUT increments the counter for bytes without a header.



Gaps between the spec and the design

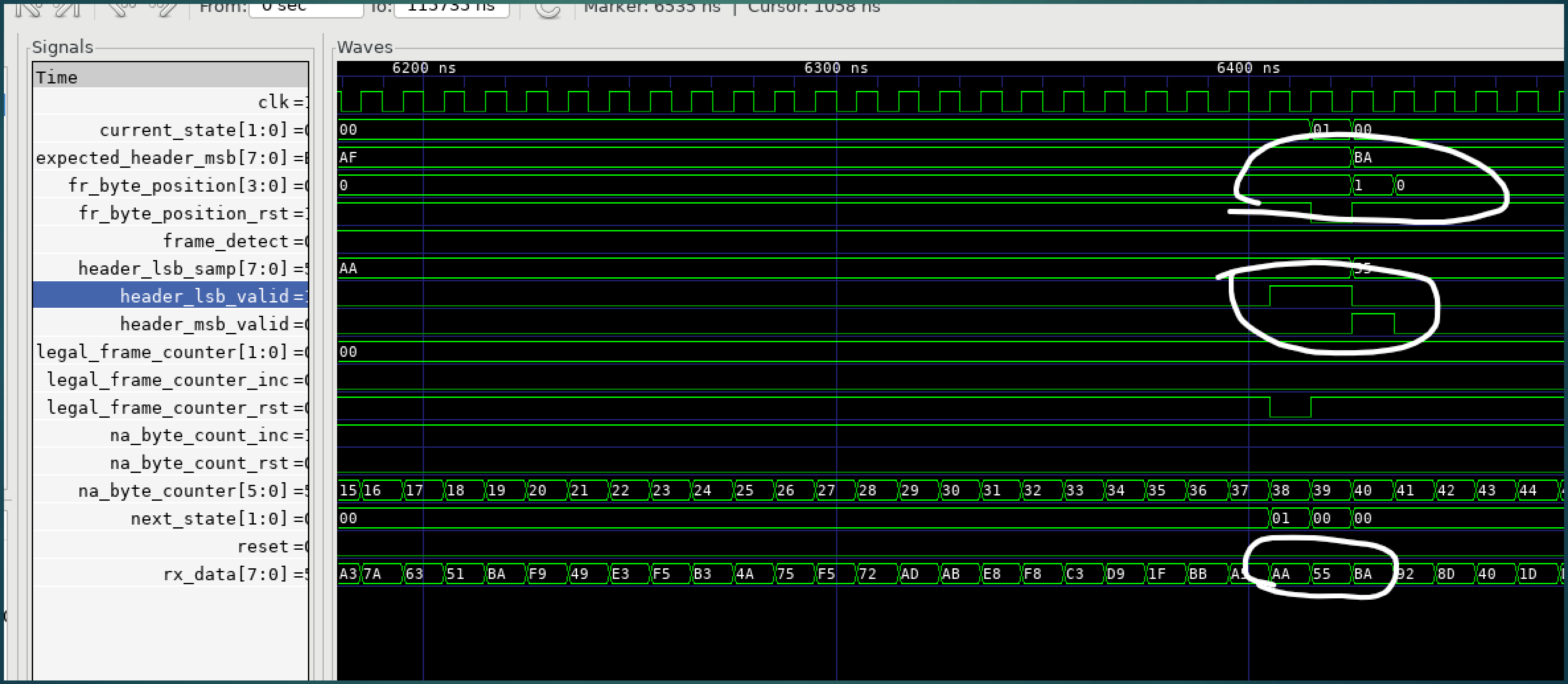
- If a valid LSB is sent twice, followed by a valid MSB, the DUT will not recognize a valid header because it transitions to an MSB detection state, thereby missing the sequence of a valid header.



```
case(current_state)
  FR_IDLE:
    begin
       if(header_lsb_valid)
     begin
        fr_byte_position_rst = 1'b1;
        na_byte_count_inc = 1'b1;
        next_state = FR_HLSB;
     end
       else
     begin
        legal_frame_counter_rst = 1'b1;
        fr_byte_position_rst = 1'b1;
        na_byte_count_inc = 1'b1;
        next_state = FR_IDLE;
     end
    end
  FR_HLSB:
    begin
       if(header_msb_valid)
     begin
        legal_frame_counter_inc = 1'b1;
        next_state = FR_HMSB;
     end
       else
     begin
        legal_frame_counter_rst = 1'b1;
        na_byte_count_inc = 1'b1;
        next_state = FR_IDLE;
     end
    end
```

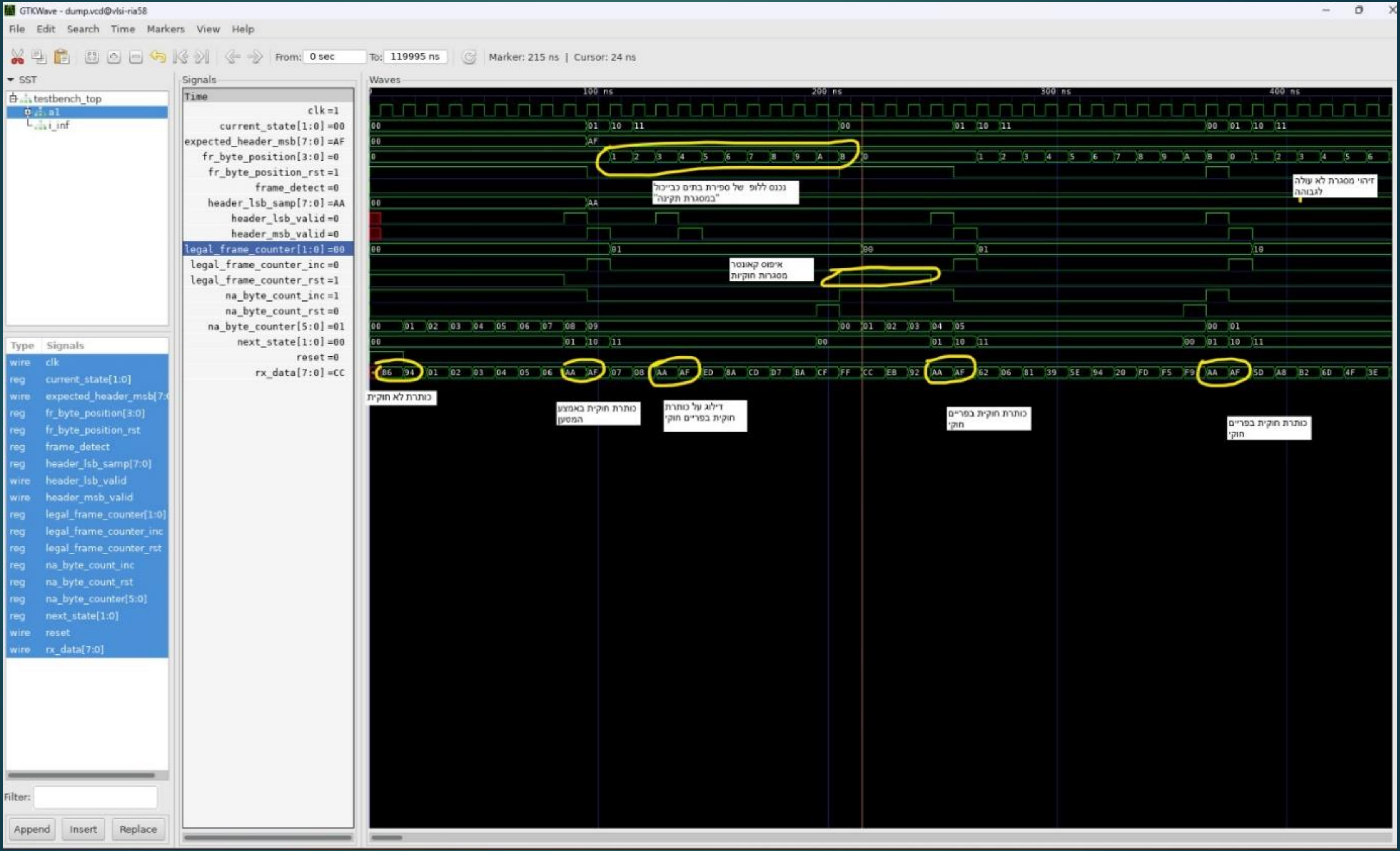
Gaps between the spec and the design

- If a valid LSB is sent twice, followed by a valid MSB, the DUT will not recognize a valid header because it transitions to an MSB detection state, thereby missing the sequence of a valid header.



Gaps between the spec and the design

- **If there is a valid header within the payload of an invalid frame, the DUT will recognize it as a valid header and start counting a valid frame. If immediately afterward, three valid frames are sent, the DUT will not enter synchronization mode because it will skip one valid frame header!**



```
Time
clk=1
current_state[1:0]=00
expected_header_msb[7:0]=AF
fr_byte_position[3:0]=0
fr_byte_position_rst=1
frame_detect=0
header_lsb_samp[7:0]=AA
header_lsb_valid=0
header_msb_valid=0
legal_frame_counter[1:0]=00
legal_frame_counter_inc=0
legal_frame_counter_rst=1
na_byte_count_inc=1
na_byte_count_rst=0
na_byte_counter[5:0]=01
next_state[1:0]=00
reset=0
rx_data[7:0]=CC
```

```
Type  Signals
wire  clk
reg   current_state[1:0]
wire  expected_header_msb[7:0]
reg   fr_byte_position[3:0]
reg   fr_byte_position_rst
reg   frame_detect
reg   header_lsb_samp[7:0]
wire  header_lsb_valid
wire  header_msb_valid
reg   legal_frame_counter[1:0]
reg   legal_frame_counter_inc
reg   legal_frame_counter_rst
reg   na_byte_count_inc
reg   na_byte_count_rst
reg   na_byte_counter[5:0]
reg   next_state[1:0]
wire  reset
wire  rx_data[7:0]
```

נכנס ללופ של ספירת בתים כבייכול "במסגרת תקינה"

זיהוי מסגרת לא עולה לגבוהה

איפוס קאונטר מסגרות חוקיות

כותרת לא חוקית

כותרת חוקית באמצע המטען

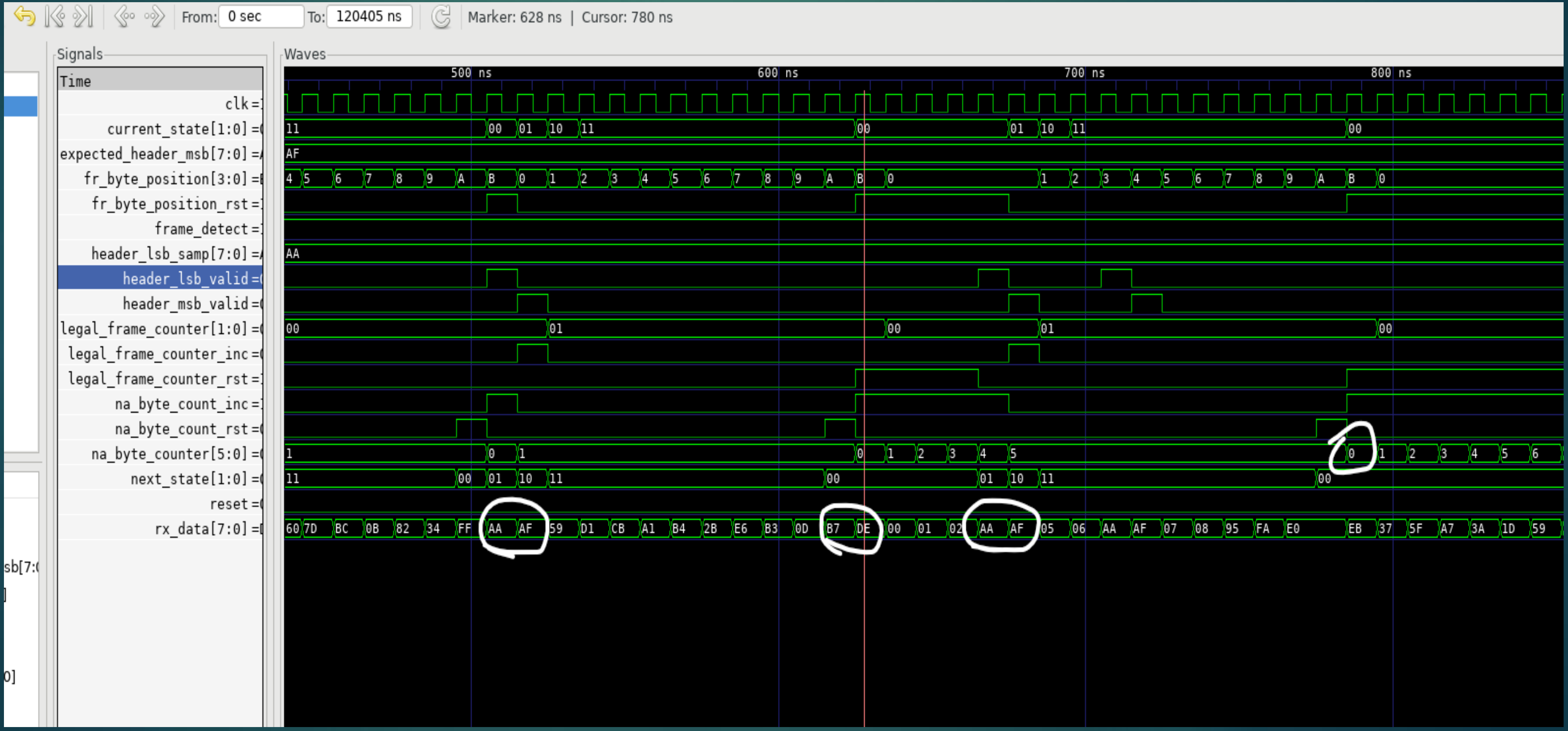
דילוג על כותרת חוקית בפריים חוקי

כותרת חוקית בפריים חוקי

כותרת חוקית בפריים חוקי

Gaps between the spec and the design

- When the DUT is in synchronization mode, if a frame is sent with a payload that contains a valid header, it will recognize it as a valid header, start counting a valid frame, and then reset the counter for bytes without a valid header. This means that if frames of this type are sent at intervals of 45 bytes without valid frames, the DUT will never exit synchronization mode.



From: 0 sec To: 120405 ns | Marker: 628 ns | Cursor: 780 ns

Signals

```
Time
clk
current_state[1:0]
expected_header_msb[7:0]
fr_byte_position[3:0]
fr_byte_position_rst
frame_detect
header_lsb_samp[7:0]
header_lsb_valid
header_msb_valid
legal_frame_counter[1:0]
legal_frame_counter_inc
legal_frame_counter_rst
na_byte_count_inc
na_byte_count_rst
na_byte_counter[5:0]
next_state[1:0]
reset
rx_data[7:0]
```

Gaps between the spec and the design

- **My main conclusion is The issue with this component is detecting or not detecting headers or partial headers in incorrect positions.**

GitHub link:
https://github.com/yakir1991/Frame-Aligner-verification