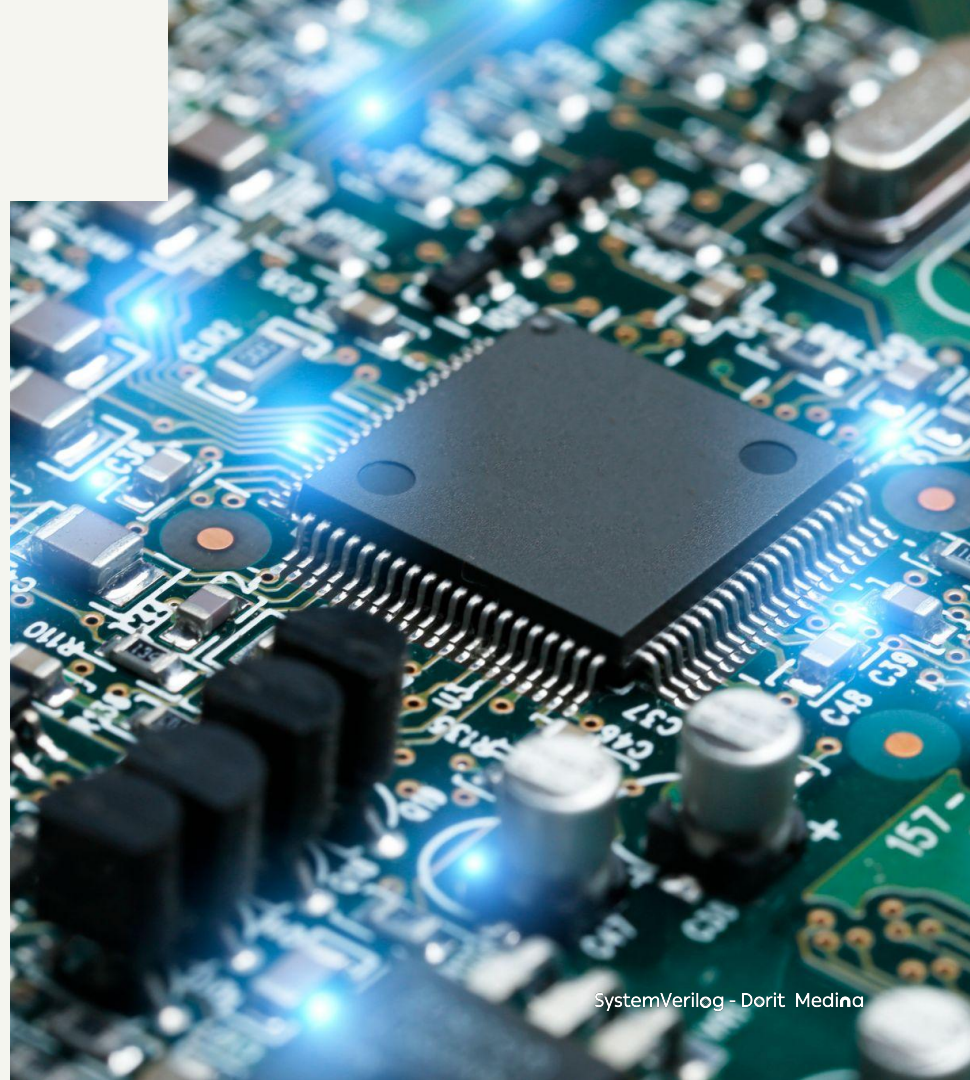
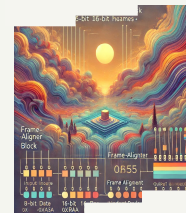


# Frame Aligner





# Frame Aligner Background



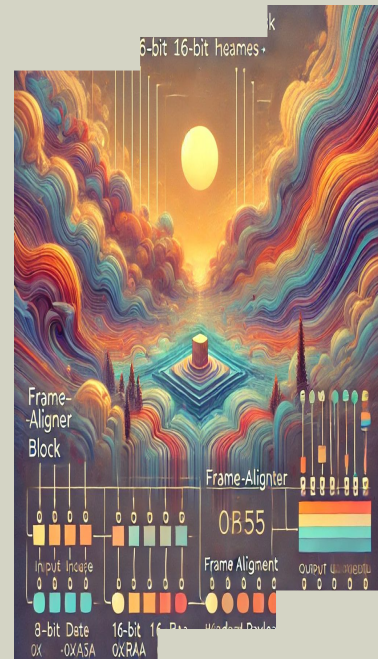
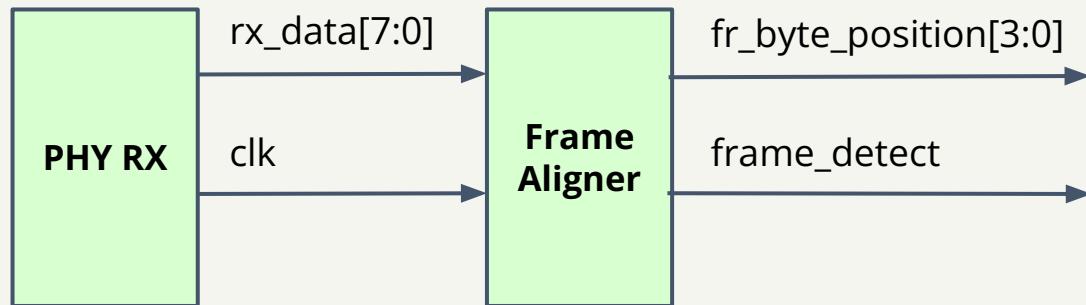
- The frame aligner works by constantly monitoring the incoming data stream for a specific header pattern (0xAFAA or 0xBA55 in this case).
  - Once it detects this pattern in three consecutive frames, it declares that alignment has been achieved
  - If four consecutive frames do not match the expected header, it declares the stream out of alignment
- In modern digital communication, such frame aligners are crucial to ensuring that data is correctly interpreted, especially in high-speed systems where frames are transmitted continuously without gaps.
- This project will give you hands-on experience in developing a verification environment for such a system using SystemVerilog (SV)

# Project Description

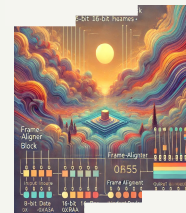
- You are tasked with designing and verifying a SystemVerilog (SV) environment to verify a frame aligner design
- Your goal is to ensure the frame aligner meets the specifications outlined in this specification



# Frame Aligner Block Diagram



# Frame Aligner Block Description



- **Input Interface**
  - The frame aligner will receive an 8-bit data stream (`rx_data[7:0]`) from the PHY (Physical Layer) along with a clock signal
- **Frame Alignment Unit**
  - Each frame consists of:
    - A header of 16 bits
    - A payload of 80 bits
    - The header contains a fixed pattern, either 0xAFAA or 0xBA55
    - The payload is random data
    - Frames are transmitted back-to-back with no gaps between them
- **Output Interface**
  - Once the frame aligner detects the header, it outputs the 16-bit header followed by the 80-bit payload for further processing

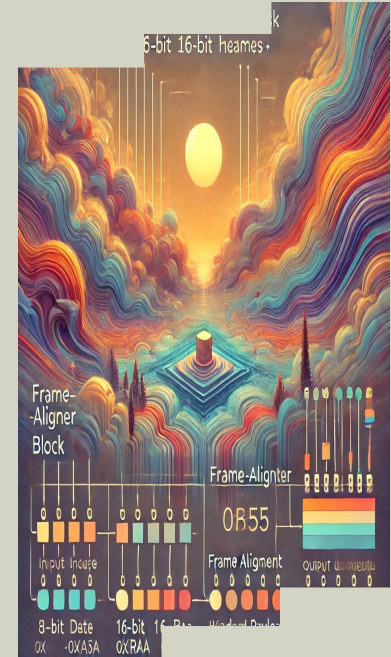
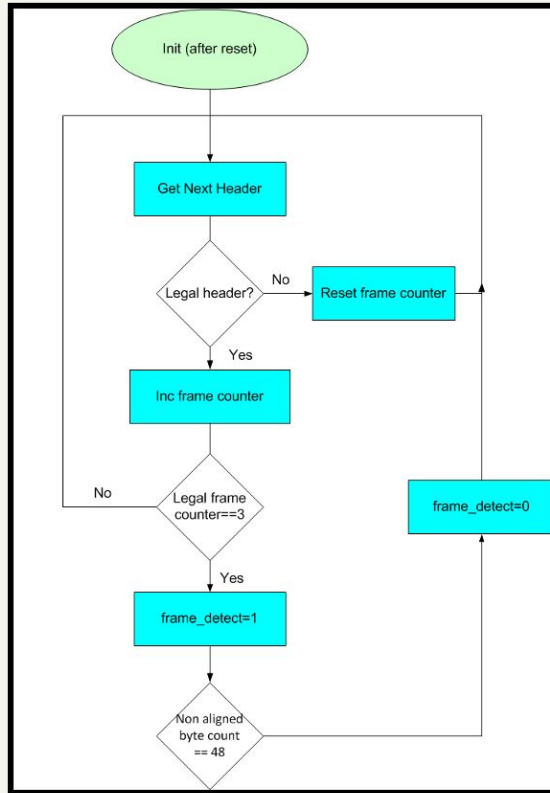
# Frame Aligner Functionality

- 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





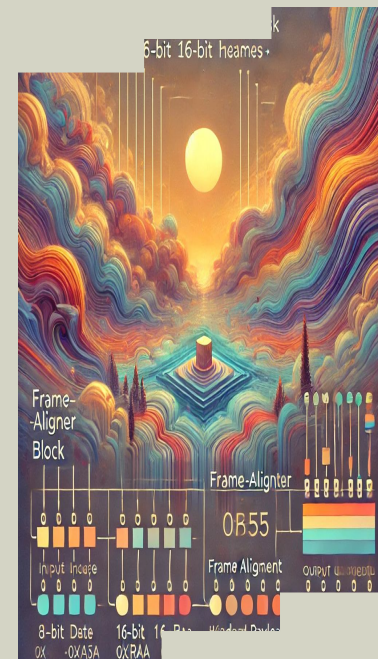
# Alignment Flow





# Project Deliverables

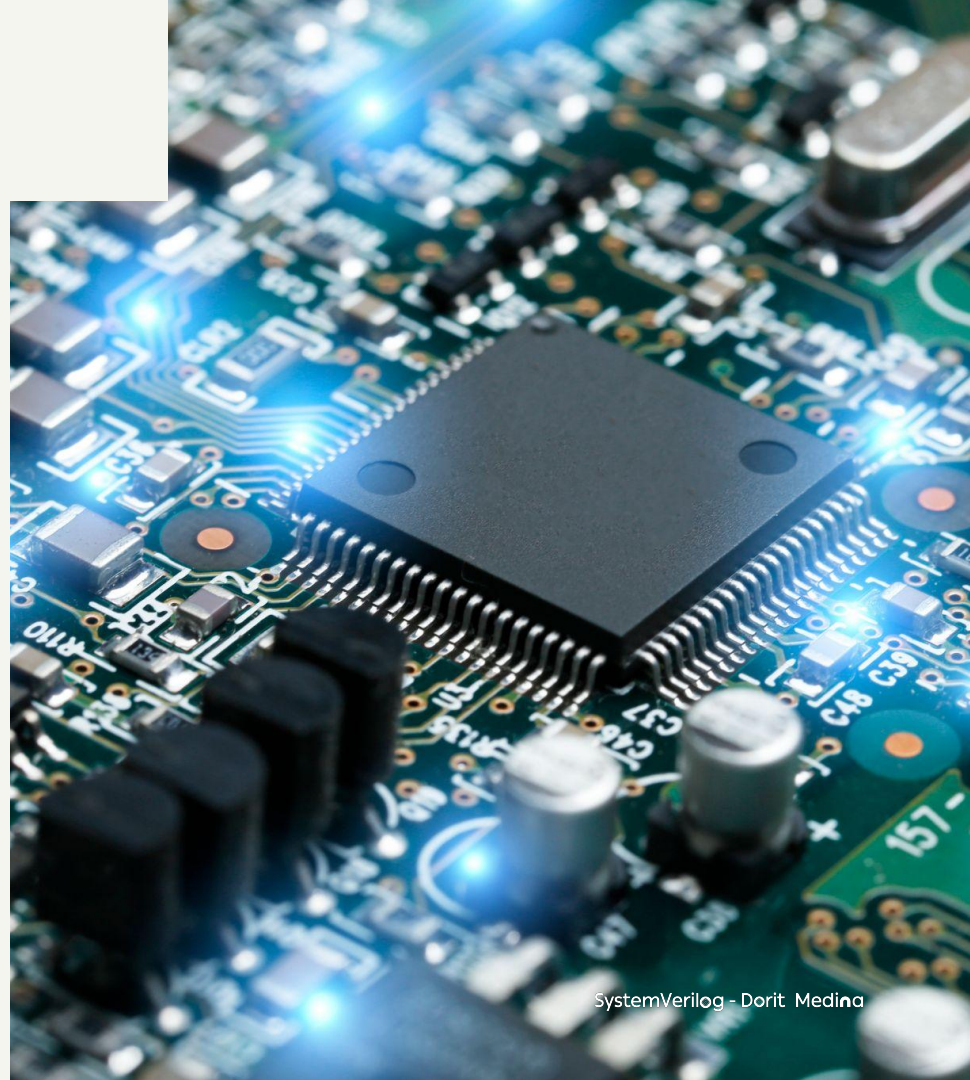
- You are required to build a complete SystemVerilog verification environment for this frame aligner block



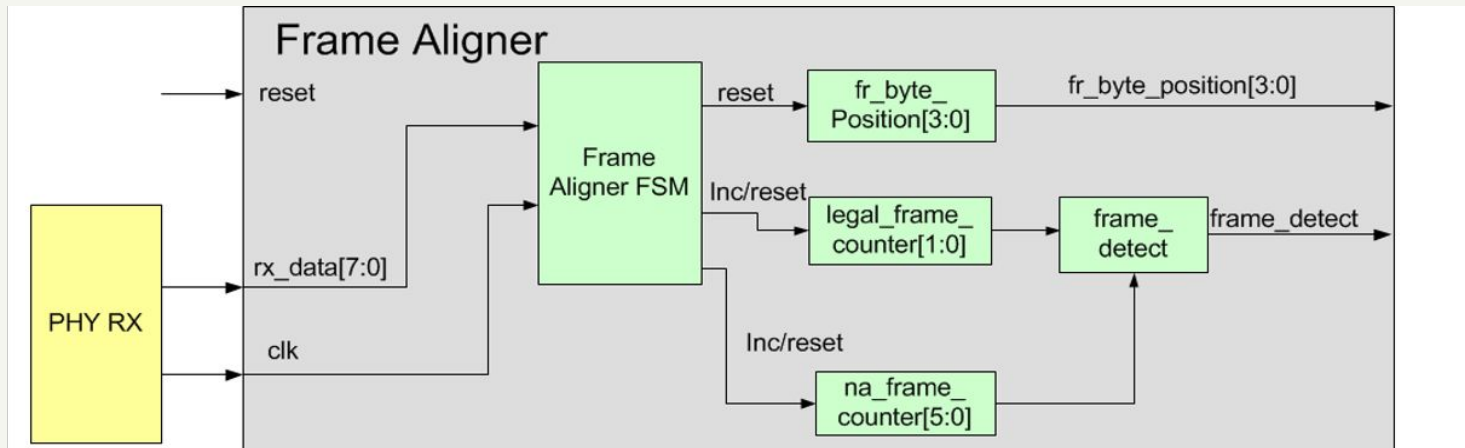
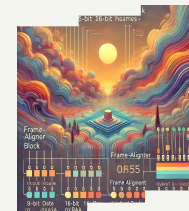
# Project Deliverables

- Your tasks include:
  - Specification, Assumptions, and Limitations
    - Document the behavior of the frame aligner block, including any assumptions and limitations
  - Functional Verification Plan
    - Outline a plan to test all functional aspects of the frame aligner, including its ability to detect aligned and misaligned frames
  - Coverage Plan
    - Define how you will achieve coverage for all possible scenarios (correct/incorrect headers, payload length, etc.)
  - Test Plan
    - Specify the different test cases that will be executed to verify the frame aligner

# Frame Aligner Design



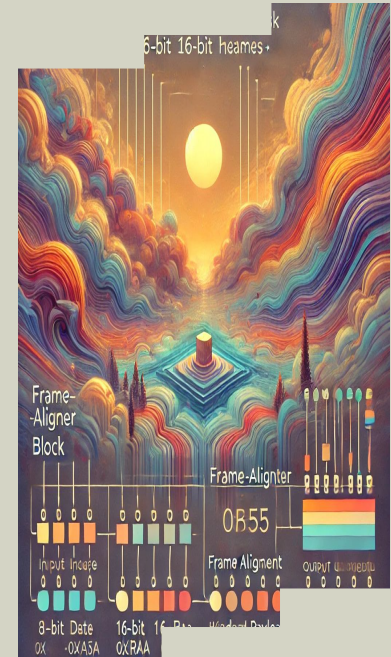
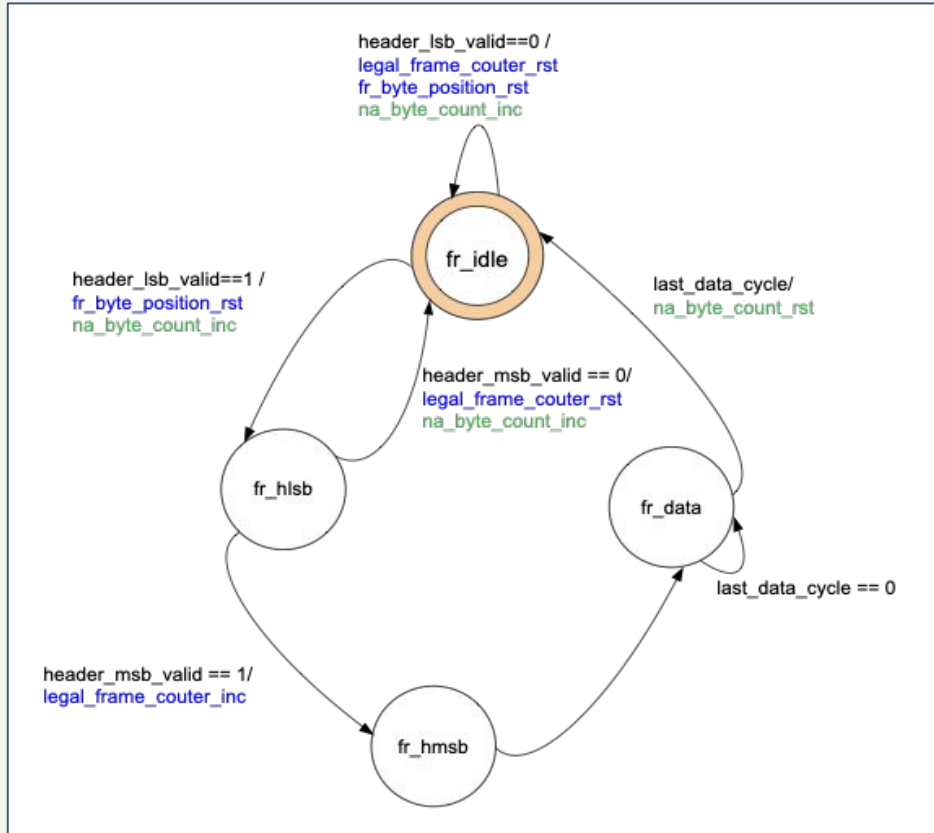
# Frame Aligner Block Diagram



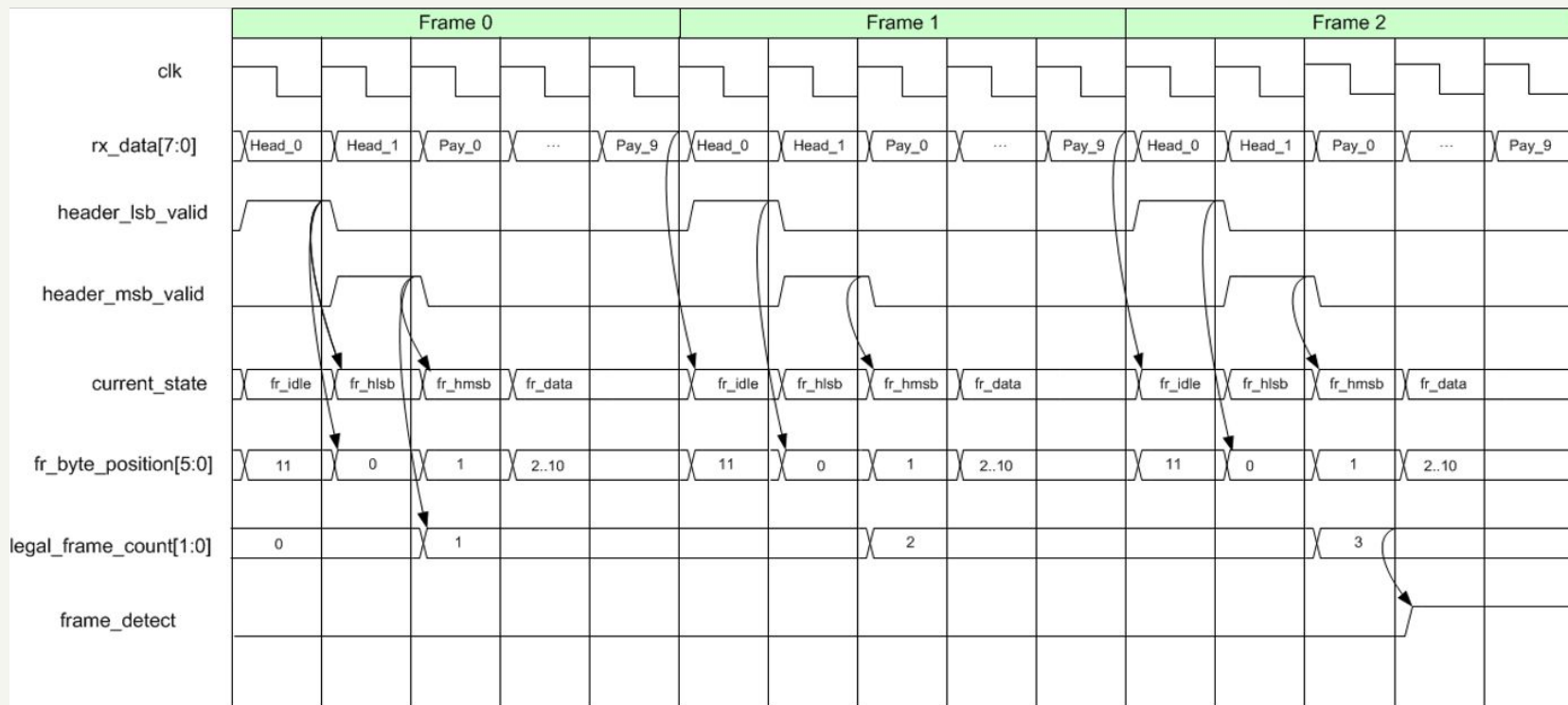
Port in	Description
clk	clock in
rx_data[7:0]	data in
Reset	Async reset active high
Port out	Description
fr_is_aligned	alignment algorithm indication
fr_byte_position[3:0]	Position of the current byte in the header or the payload range 0-11

Reg name	Description
legal_frame_counter[1:0]	when this counter reaches 3, frame_detect is set .
na_frame_counter[5:0]	when this counter reaches 48, frame_detect is reset .
Valid_h2	Sample h1 and sets only in case h2 is the correct one

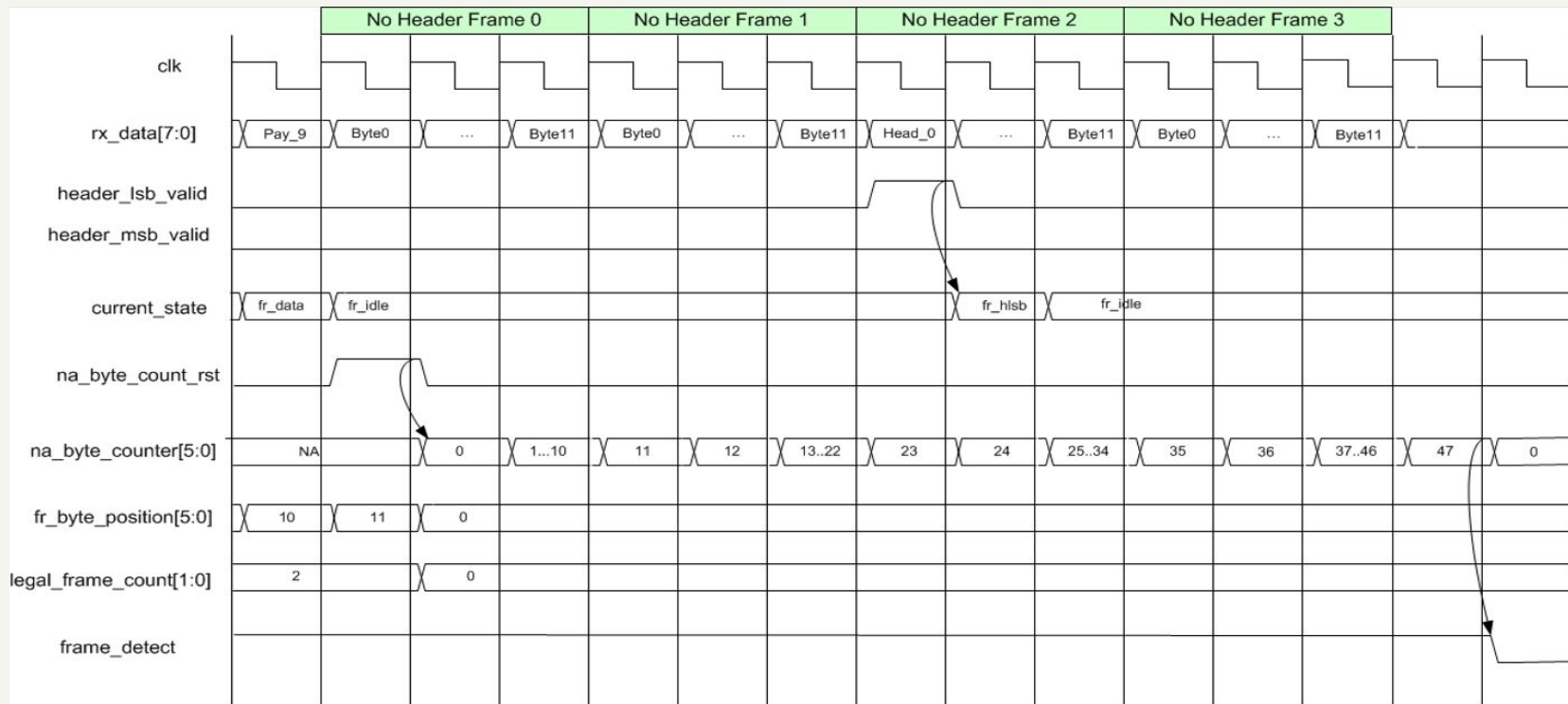
# Frame Aligner FSM



# Frame aligner Waveform 1

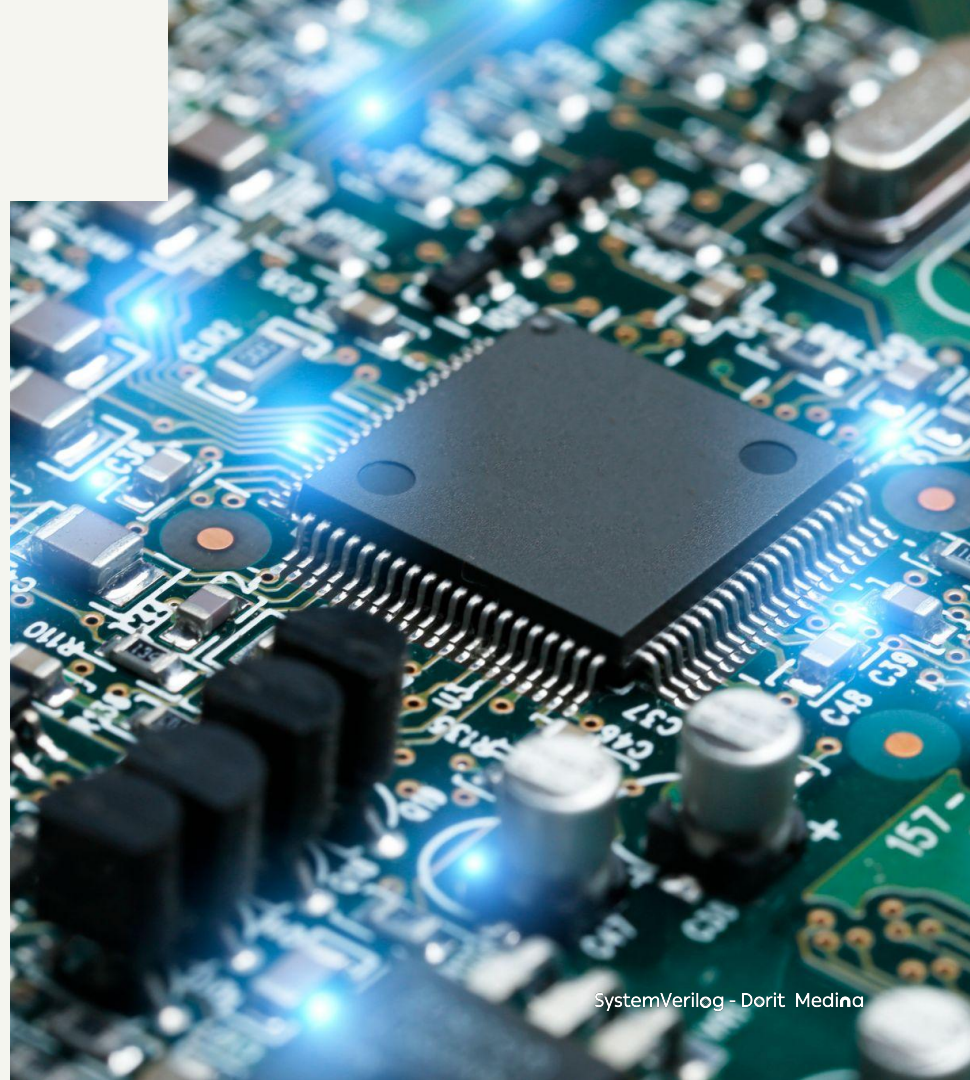


# Frame aligner Waveform 2





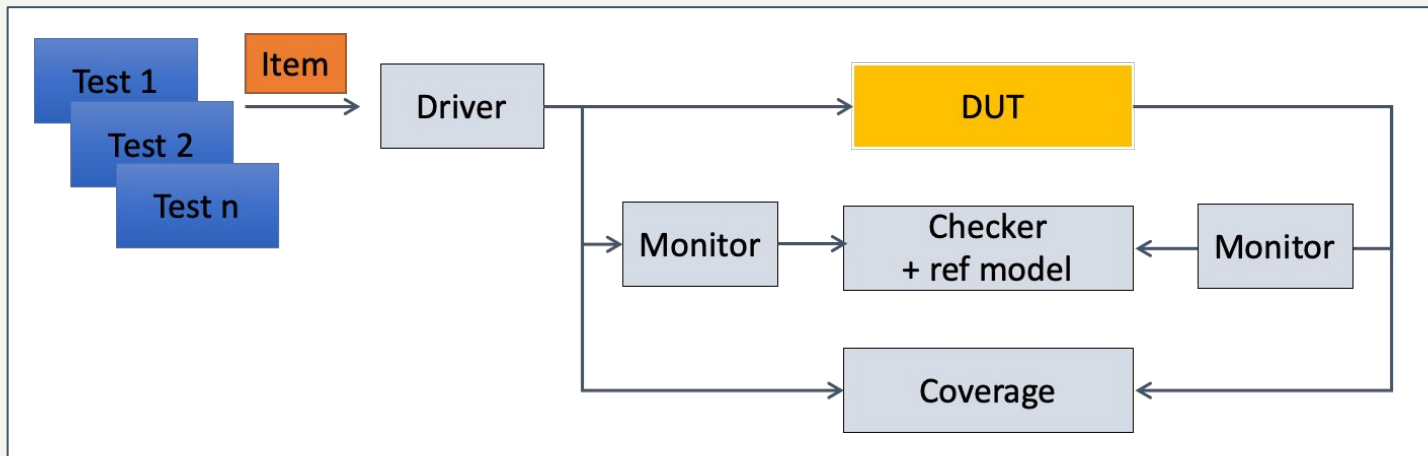
# Frame Aligner Verification



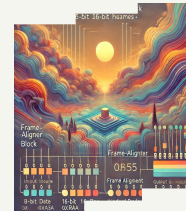
# Verification Environment Requirements



- Your task is to generate a SV block level environment for the frame aligner



# The Data Item



- `frame_item` is a class that holds the frame content and constraint needed to control it
- Definitions
  - `typedef enum bit [1:0] {HEAD_1 , HEAD_2 , ILLEGAL} header_type_t;`
- Main Members
  - `Payload[]` – dynamic array
  - `header` - the user can constraint `header_type = {HEAD_1 / HEAD_2 / ILLEGAL}`
    - According to this value header value will be set
- Constraints
  - Payload lengthCurrently size should between [10:40]
  - Header type – distribution for HEAD\_1 / HEAD\_2 / ILLEGAL

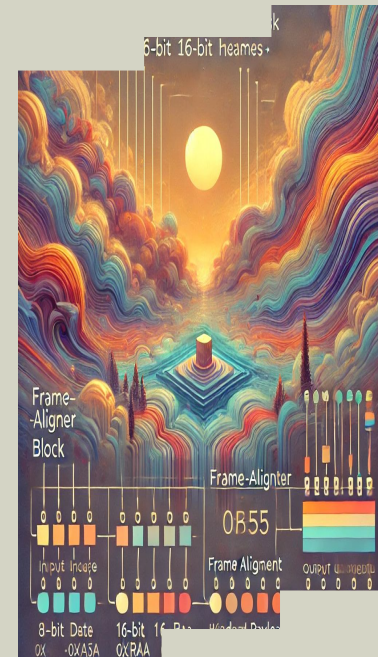
# Data item – Cont

- Add post\_randomize() function to
  - Initialize random values to the payload after the length was determined
  - Set the correct value to the header after the header type was determined



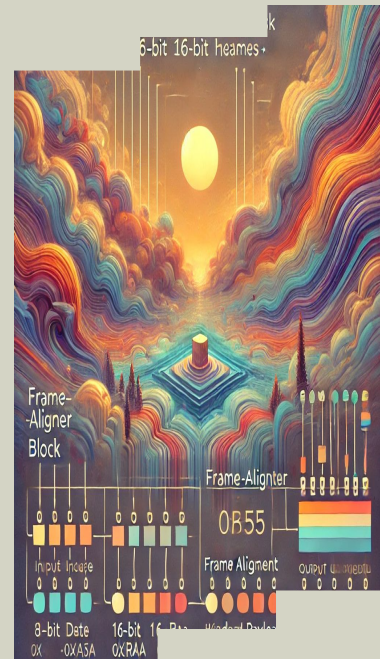
# The Driver

- When frame\_item is randomized, the driver role is to drive it to the DUT
- Contains send\_to\_dut(frame\_item frame\_data) task to implement the protocol RX frame\_aligner



# Test Suggestion

- The test should have the following sequence
  - Send 5 packets with HEAD\_1
  - Send 4 illegal packets
  - Send 5 packets with HEAD\_2



# Submission

- Submit your final project with the following
  - A PDF document containing your specification, functional verification plan, coverage plan, and test plan
  - A zipped folder containing your SystemVerilog source files for the verification environment, including all necessary components such as the frame item, driver, and testbench

**Good luck with your project!**

