The eth\_axis\_tx module is a implementation of an Ethernet frame transmitter over an AXI4-Stream interface. It takes Ethernet header fields (destination MAC, source MAC, Ethertype) and AXI-streamed payload data as input, combines them, and outputs a full Ethernet frame via the AXI stream interface. Here's a **detailed explanation** of its functionality:

**🔧 Module Parameters**

* DATA\_WIDTH: Width of the AXI stream data interface.
* KEEP\_ENABLE: If 1, the tkeep signal is used to indicate valid bytes.
* KEEP\_WIDTH: Width of tkeep = DATA\_WIDTH / 8.

**📥 Inputs**

* s\_eth\_hdr\_valid: Indicates header is valid.
* s\_eth\_hdr\_ready: Handshake signal; high when ready to accept a new header.
* s\_eth\_dest\_mac, s\_eth\_src\_mac, s\_eth\_type: Ethernet header fields (MACs + Ethertype).
* AXI-stream signals:
  + s\_eth\_payload\_axis\_tdata: Payload data.
  + s\_eth\_payload\_axis\_tkeep: Byte qualifiers.
  + s\_eth\_payload\_axis\_tvalid, tready, tlast, tuser: Standard AXI stream signaling.
* clk, rst: Clock and reset.

**📤 Outputs**

* AXI-stream output: m\_axis\_tdata, tkeep, tvalid, tlast, tuser.
* busy: Indicates the module is actively processing data.

**🧠 Internal Functionality Overview**

**📦 1. Header Handling**

* When s\_eth\_hdr\_valid and s\_eth\_hdr\_ready are both high:
  + The header fields are stored in internal registers.
  + A state machine begins transmitting the 14-byte header (6B dest + 6B src + 2B type).
  + Header transmission takes CYCLE\_COUNT cycles based on DATA\_WIDTH.

**🚚 2. Payload Handling**

* Payload data is forwarded after header transmission is complete.
* Handles data alignment if HDR\_SIZE isn't a multiple of data bus width (OFFSET logic).
* Manages a “save” and “shift” mechanism to align and merge leftover payload bytes with new ones across cycles.

**🔁 3. State Machine**

Two main states:

* send\_eth\_header\_reg: Asserts during header transmission.
* send\_eth\_payload\_reg: Asserts during payload transmission.  
  Transitions are managed by internal pointers (ptr\_reg) and offset conditions.

**📊 4. AXI Stream Output Interface**

* Internally buffers output data using a two-level staging:
  + Registers for direct output (m\_axis\_tdata\_reg, etc.).
  + Temporary registers (temp\_m\_axis\_tdata\_reg, etc.) to hold intermediate data when downstream is not ready.
* Implements a logic block to manage ready/valid handshake between internal logic and AXI output.

**🧾 Key Logic Highlights**

**✅ AXI Header Construction:**

Header bytes are assembled and streamed based on current ptr\_reg:

verilog

CopyEdit

if (ptr\_reg == offset/BYTE\_LANES) begin

m\_axis\_tdata\_int[...] = field;

m\_axis\_tkeep\_int[...] = 1'b1;

end

**📉 tkeep and Offset Logic:**

When the header ends mid-beat (e.g., on 3rd byte of 8-bit lane), the logic:

* Saves the remaining payload.
* Merges new payload with saved payload and aligns it.
* Adjusts tkeep accordingly to indicate valid byte lanes.

**🏁 End-of-Frame (EOP) Handling:**

* Sets tlast when the final beat of the payload is detected.
* Uses s\_eth\_payload\_axis\_tlast and tkeep bits to determine whether the current beat is final or needs an extra cycle.

**🧮 Example Use Case (DATA\_WIDTH = 64):**

* KEEP\_WIDTH = 8 (1 byte per bit in tkeep)
* Ethernet header (14B) fits into 2 full 8-byte cycles.
* If payload is streamed in next cycles, AXI stream will first contain:
  1. 8B header
  2. 6B header + 2B payload
  3. Next payload bytes

**🔄 Reset Behavior**

On reset (rst high), the module:

* Clears internal state and output registers.
* Resets header pointer and control FSMs.

A diagram of a computer program

AI-generated content may be incorrect.

**Bock diagram for eth\_axi\_tx**