AXI Adapter Specification

Introduction

The `axi_adapter` module is designed to manage communication with an AXI bus, facilitating data transfers between a processor core and memory or peripheral devices. It adheres to the AXI protocol, supporting both read and write operations, including atomic operations. The module is parameterized to allow flexibility in data width, address width, and other AXI-specific configurations.

Architecture

The `axi_adapter` module is structured around a finite state machine (FSM) that manages the various states of AXI transactions. Key operations include handling read and write requests, managing burst transactions, and supporting atomic operations. The module operates synchronously with a clock signal and uses an asynchronous active-low reset.

Key Components

- **State Machine**: Manages the transaction states such as IDLE, WAIT_B_VALID, WAIT_AW_READY, etc.
- **AXI Request and Response Handling**: Interfaces with AXI request and response channels to manage data flow.
- **Atomic Operations**: Supports atomic operations like swap, add, and logical operations.

Implementation Details

- **Clock Domain**: The module operates in a single clock domain, driven by `clk_i`.
- **Parameterization**: The module uses parameters to define data width, address width, and other AXI-specific configurations, allowing it to be tailored to specific system requirements.

Interface

Parameters

Parameter	Type	Default Value	Description
`cva6_cfg`	`ariane_pkg::cva6_cfg_t`	`ariane_pkg::cva6_cfg_empty`	Configuration for CVA6 core.
`DATA_WIDTH`	`int unsigned`	`256`	Width of the data bus.

Parameter	Type	Default Value	Description
CRITICAL_WORD_FIRST	logic	0	Determines if critical word is read first.
CACHELINE_BYTE_OFFSET	int unsigned	8	Byte offset for cache line.
AXI_ADDR_WIDTH	int unsigned	0	Width of the AXI address bus.
AXI_DATA_WIDTH	int unsigned	0	Width of the AXI data bus.
AXI_ID_WIDTH	int unsigned	0	Width of the AXI ID bus.
axi_req_t	type	ariane_axi::req_t	AXI request type.
axi_rsp_t	type	ariane_axi::resp_t	AXI response type.

Ports

Signal	Width	In/Out	Description
clk_i	1	In	Clock input.
rst_ni	1	In	Asynchronous reset, active low.
req_i	1	In	Request input signal.
type_i	-	In	Type of AXI request.
amo_i	-	In	Atomic operation type.
addr_i	riscv::XLEN	In	Address input.
we_i	1	In	Write enable signal.
wdata_i	(DATA_WIDTH/AXI_DATA_WIDTH)-1	In	Write data input.
be_i	(DATA_WIDTH/AXI_DATA_WIDTH)-1	In	Byte enable input.
size_i	2	In	Size of the data transfer.
id_i	AXI_ID_WIDTH	In	AXI transaction ID.
gnt_o	1	Out	Grant output signal.
valid_o	1	Out	Valid output signal.
rdata_o	(DATA_WIDTH/AXI_DATA_WIDTH)-1	Out	Read data output.
id_o	AXI_ID_WIDTH	Out	Output transaction ID.
critical_word_o	AXI_DATA_WIDTH	Out	Critical word output.
critical_word_valid_o	1	Out	Critical word valid signal.
axi_req_o	axi_req_t	Out	AXI request output.

Internal Signals

Signal	Type	Description
state_q	logic [3:0]	Current state of the FSM.
state_d	logic [3:0]	Next state of the FSM.
cnt_d	logic [ADDR_INDEX-1:0]	Next value of the counter.
cnt_q	logic [ADDR_INDEX-1:0]	Current value of the counter.
cache_line_d	logic [(DATA_WIDTH/AXI_DATA_WIDTH)-1:0][AXI_DATA_WIDTH-1:0]	Next cache line data.
cache_line_q	logic [(DATA_WIDTH/AXI_DATA_WIDTH)-1:0][AXI_DATA_WIDTH-1:0]	Current cache line data.
addr_offset_d	logic [(DATA_WIDTH/AXI_DATA_WIDTH)-1:0]	Next address offset.
addr_offset_q	logic [(DATA_WIDTH/AXI_DATA_WIDTH)-1:0]	Current address offset.
id_d	logic [AXI_ID_WIDTH-1:0]	Next transaction ID.
id_q	logic [AXI_ID_WIDTH-1:0]	Current transaction ID.
index	logic [ADDR_INDEX-1:0]	Index for data access.
amo_d	ariane_pkg::amo_t	Next atomic operation type.
amo_q	ariane_pkg::amo_t	Current atomic operation type.
size_d	logic [1:0]	Next size of the data transfer.
size_q	logic [1:0]	Current size of the data transfer.

Timing

The `axi_adapter` module operates synchronously with the `clk_i` signal. The latency for output validity depends on the state transitions within the FSM. The module samples inputs and updates outputs on the rising edge of the clock. The FSM ensures that outputs are valid once the necessary conditions are met, such as receiving valid responses from the AXI bus.

Usage

To use the `axi_adapter` module: 1. Provide the necessary input signals, including `req_i`, `type_i`, `amo_i`, `addr_i`, `we_i`, `wdata_i`, `be_i`, `size_i`, and `id_i`. 2. Monitor the `gnt_o` signal to determine when the module has granted the request. 3. For read operations, check `valid_o` and `rdata_o` for valid data. 4. For write operations, ensure `gnt_o` is asserted before proceeding. 5. Configure the module parameters to match the system requirements, such as data width and address width.

The module supports both single and burst transactions, with specific handling

for atomic operations. Ensure that the AXI subsystem supports the required features, such as wrapping reads, if `CRITICAL_WORD_FIRST` is enabled.

Functional Description (Generated by funcgen)

Module: axi_adapter

Source File: axi_adapter.sv

Purpose

The `axi_adapter` module acts as an interface manager that handles communication with an AXI (Advanced eXtensible Interface) Bus. It serves to bridge a high-performance core with the AXI bus, managing read and write transactions with the support for atomic operations.

Parameters

- **cva6_cfg**: Default is `ariane_pkg::cva6_cfg_empty`. This is a configuration parameter specific to the CVA6 core.
- **DATA_WIDTH**: Default is `256`. Specifies the data width of the system cache line.
- **CRITICAL_WORD_FIRST**: Default is `0`. Specifies if the AXI subsystem supports wrapping reads.
- **CACHELINE_BYTE_OFFSET**: Default is `8`. Used for addressing within a data cache line.
- **AXI_ADDR_WIDTH**: Default is `0`. Defines the width of addresses on the AXI bus.
- **AXI_DATA_WIDTH**: Default is `0`. Specifies the data width on the AXI bus.
- **AXI_ID_WIDTH**: Default is `0`. Represents the width of AXI transaction IDs.
- **axi_req_t**: Type `ariane_axi::req_t`. Defines the structure for AXI request transactions.
- **axi_rsp_t**: Type `ariane_axi::resp_t`. Defines the structure for AXI response transactions.

Ports

- **clk_i**: (Input, 1-bit) - System clock input.
- **rst_ni**: (Input, 1-bit) - Asynchronous reset input, active low.
- **req_i**: (Input) - Request input signal indicating that a transaction is needed.
- **type_i**: (Input) - Indicates the type of request (single or burst).
- **amo_i**: (Input) - Specifies the atomic operation type.

- **gnt_o**: (Output) - Grant output signal for the request.
- **addr_i**: (Input, riscv::XLEN) - The address for the current transaction.
- **we_i**: (Input) - Write enable signal; high for write transactions.
- **wdata_i**: (Input, Array of DATA_WIDTH/AXI_DATA_WIDTH by AXI_DATA_WIDTH) - Write data input.
- **be_i**: (Input, Array of DATA_WIDTH/AXI_DATA_WIDTH by AXI_DATA_WIDTH/8) - Byte enable signal for write data.
- **size_i**: (Input, 2-bit) - Size of the memory access.
- **id_i**: (Input, AXI_ID_WIDTH) - Transaction ID input.
- **valid_o**: (Output) - Indicates valid read data.
- **rdata_o**: (Output, Array of DATA_WIDTH/AXI_DATA_WIDTH by AXI_DATA_WIDTH) - Read data output.
- **id_o**: (Output, AXI_ID_WIDTH) - Transaction ID for response.
- **critical_word_o**: (Output, AXI_DATA_WIDTH) - The critical word of a read transaction.
- **critical_word_valid_o**: (Output) - Valid signal for critical word.
- **axi_req_o**: (Output, axi_req_t) - Structured AXI request output.
- **axi_resp_i**: (Input, axi_rsp_t) - Received AXI response input.

Internal Signals

- **state_q, state_d**: Internal state tracking signals for the FSM (Finite State Machine).
- **cnt_d, cnt_q**: Counters for tracking the number of AXI transfers.
- **cache_line_d, cache_line_q**: Buffers for storing the cache line data.
- **addr_offset_d, addr_offset_q**: Registers for storing address offset information.
- **id_d, id_q**: Registers for tracked transaction IDs.
- **amo_d, amo_q**: Registers for atomic operation tracking.
- **size_d, size_q**: Registers for the size of data being transferred.
- **index**: Calculated index used within combinatorial logic to reference data within a cache line.

Functionality

State Machine and Transaction Flow

- **IDLE**: Waits for a `req_i` signal. Determines if the incoming request is for a read or write operation.
- **WAIT_AW_READY**, **WAIT_LAST_W_READY**, **WAIT_LAST_W_READY_AW_RE** Handles the write control logic, waiting for address/write readiness signals.
- **WAIT_B_VALID**: Waits for write acknowledgment and processes responses. Handles atomic operation results if needed.
- **WAIT_R_VALID**, **WAIT_R_VALID_MULTIPLE**: Manages read operations, checks for valid data reception, handles burst reads.
- **COMPLETE_READ**: Finalizes read operations, asserting validity.

- **WAIT_AMO_R_VALID**: Specifically handles atomics that must wait for read data to complete operations.

Combinational Logic

- Calculates the appropriate index within cache lines for read/write operations.
- Composes AXI requests using data inputs, transaction IDs, byte enable signals, and AMO (atomic operations) flags.

Instantiations

- Internal functions such as `atop_from_amo` and `amo_returns_data` map atomic operations to AXI transaction properties.

Inter-Module Connections

This module functions as a bridge layer, interfacing between the host system and an external AXI bus. The inter-module connectivity is largely oriented around managing transactions through AXI protocol signals (requests/responses). Direct instantiations or submodule invokes are not detailed as code strictly interfaces with AXI.

The provided description gives a designer or engineer a clear understanding of the operation and structure of the `axi_adapter` module, equipped to handle reading, writing, and atomic operations via the AXI bus.

This description provides a comprehensive view of the `axi_adapter` module based on the provided parsed details and is structured according to Verilog/SystemVerilog coding standards. Each section addresses specific aspects that are critical for subsequent design development or verification.