GPIO Module Specification

Introduction

The General Purpose Input/Output (GPIO) module is designed to provide a flexible interface for digital input and output operations. It is part of the OpenTitan project and adheres to the Apache License, Version 2.0. The module supports interrupt generation, alert mechanisms, and configurable input filtering, making it suitable for a wide range of applications in digital systems.

Architecture

The GPIO module is structured into two main components: the top-level `gpio` module and the `gpio_reg_top` submodule. The top-level module handles the main GPIO operations, including input filtering, output control, and interrupt generation. The `gpio_reg_top` submodule manages the register interface, providing read and write access to various control and status registers.

Top-Level Module Structure

- **GPIO Input Filtering**: Each GPIO input can be filtered using a 2-stage synchronizer, controlled by the `GpioAsyncOn` parameter.
- **Output Control**: The module supports both direct and masked output modes, allowing for flexible control of GPIO outputs.
- **Interrupt Generation**: Interrupts can be generated based on rising, falling, high-level, and low-level events on the GPIO inputs.
- **Alert Mechanism**: Alerts are generated for specific error conditions, with support for asynchronous alert signaling.

Key Operations

- **Data Flow**: Input data is filtered and then stored in internal registers. Output data is controlled through direct or masked registers.
- **Interrupt Handling**: Interrupts are generated based on configurable conditions and are output as a 32-bit signal.
- **Alert Handling**: Alerts are managed through a dedicated alert sender primitive, supporting asynchronous operation.

Implementation Details

- **Clock Domain**: The module operates synchronously with the input clock `clk_i`.
- **Reset**: The module uses an active-low reset `rst_ni` to initialize internal states.
- **Optimization**: The design uses parameterized filtering and alert mechanisms to optimize for different use cases.

Interface

Parameters

Parameter	Type	Default Value	Description
AlertAsyncOn	logic	{NumAlerts{1'b1}}	Configures asynchronous alert signaling
GpioAsyncOn	bit	1	Enables 2-stage synchronizers on GPIO inputs

Ports

Signal	Width	In/Out	Description
clk_i	1	In	Clock input
rst_ni	1	In	Active-low reset input
tl_i	-	In	Bus interface input
tl_o	-	Out	Bus interface output
intr_gpio_o	32	Out	GPIO interrupt output
alert_rx_i	NumAlerts	In	Alert receive input
alert_tx_o	NumAlerts	Out	Alert transmit output
cio_gpio_i	32	In	GPIO input
cio_gpio_o	32	Out	GPIO output
cio_gpio_en_o	32	Out	GPIO output enable

Internal Signals

Signal	Type	Description
reg2hw	gpio_reg2hw_t	Register to hardware interface
hw2reg	gpio_hw2reg_t	Hardware to register interface
cio_gpio_q	logic [31:0]	Latched GPIO output
cio_gpio_en_q	logic [31:0]	Latched GPIO output enable
data_in_d	logic [31:0]	Filtered GPIO input data
data_in_q	logic [31:0]	Latched GPIO input data
event_intr_rise	logic [31:0]	Rising edge interrupt events
event_intr_fall	logic [31:0]	Falling edge interrupt events
event_intr_actlow	logic [31:0]	Active low level interrupt events
event_intr_acthigh	logic [31:0]	Active high level interrupt events
event_intr_combined	logic [31:0]	Combined interrupt events
alert_test	logic [NumAlerts-1:0]	Alert test signal

Signal	Type	Description
alerts	logic [NumAlerts-1:0]	Alert signals

Timing

The GPIO module operates synchronously with the input clock `clk_i`. The input signals are sampled on the rising edge of the clock, and outputs are updated accordingly. The module introduces a latency of one clock cycle for input filtering and output operations.

Usage

To use the GPIO module, follow these steps: 1. **Initialization**: Ensure the module is reset using `rst_ni` before operation. 2. **Configuration**: Set the desired parameters for input filtering and alert signaling. 3. **Input Handling**: Provide input signals through `cio_gpio_i` and configure interrupts as needed. 4. **Output Control**: Use `cio_gpio_o` and `cio_gpio_en_o` to control GPIO outputs. 5. **Interrupt and Alert Management**: Monitor `intr_gpio_o` for interrupt events and `alert_tx_o` for alert signals.

The module supports various configurations through its register interface, allowing for flexible adaptation to different application requirements.

Functional Description (Generated by funcgen)

Comprehensive Functional Description

Module: `gpio_reg_top`

Purpose

The `gpio_reg_top` module, described in the `gpio_reg_top.sv` file, serves as the top-level register interface for GPIO peripherals. It handles transactions from the TL-UL (TileLink-Unbuffered Link) interface, performs necessary integrity checks, and manages register access and configuration for GPIO hardware. It is auto-generated by `reggen`.

Parameters

- **AW**: Default value `6`. This represents the address width, determining the size of the addressable space.
- **DW**: Default value `32`. This parameter specifies the data width for registers.

- **DBW**: Derived value `DW/8`. Represents byte width, used in byte enable operations for register writes.

Ports

- **clk_i** (input, 1-bit): Clock input for synchronous operations.
- **rst_ni** (input, 1-bit): Active-low reset input to initialize the module states.
- **tl_i** (input, `tlul_pkg::tl_h2d_t`): TileLink interface input for host-to-device communication.
- **tl_o** (output, `tlul_pkg::tl_d2h_t`): TileLink interface output for device-to-host communication.
- **reg2hw** (output, `gpio_reg_pkg::gpio_reg2hw_t`): Interface for writing configuration to GPIO hardware.
- **hw2reg** (input, `gpio_reg_pkg::gpio_hw2reg_t`): Interface for reading status from GPIO hardware.
- **intg_err_o** (output, 1-bit): Signal to indicate if an integrity error has occurred.

Internal Signals

- **reg_we**: Write enable signal for register writes.
- **reg_re**: Read enable signal for register reads.
- **reg_addr**: Holds the address for the current register access.
- **reg_wdata**: Data to be written into a register.
- **reg_be**: Byte enable signals for sub-word write operations.
- **reg_rdata**: Data read from a register.
- **reg_error**: indicates if a register access operation encountered an error.
- **intg_err**: Signal set when an incoming payload integrity check fails.
- **reg_we_err**: Indicates error when multiple write enables are detected.
- **err_q**: Latched error state across clock cycles.
- **addr_hit**: Holds information about the register address being accessed.
- **wr_err**: Indicates an error due to write operation not aligning with permitted byte enable settings.
- **reg_we_check**: Vector for checking write enables across the registers.

Functionality

- **Sequential Logic**:
  - Synchronization and error latching are managed using positive edge-triggered flip-flops. `err_q` preserves error state across cycles.
- **Combinational Logic**:
  - Address decoding (`addr_hit`) determines if the accessed address corresponds to any of the defined registers.
  - Integrity checks and write-enable error detection are implemented to ensure data consistency and validity.

  - Register data (`reg_rdata_next`) is updated based on the decoded address and read enable signals.
  - Write logic ensures data is only written if enabled, the correct address is accessed, and no errors are present.
- **State Machines or Control Logic**:
  - The module handles control operations like decoding addresses (for register access), managing write and read sequences, and triggering error states when anomalies are detected.

Instantiations

- **tlul__cmd__intg__chk (u__chk)**:
  - Checks the integrity of the incoming TL-UL payload. Connects `tl_i` to `tl_i` and outputs to `intg_err`.
- **prim__reg__we__check (u__prim__reg__we__check)**:
  - Used to verify erroneous concurrent write enables sourced from a vector across different registers.
- **tlul__rsp__intg__gen**:
  - Generates outgoing integrity responses for the TileLink interface. Interacts with pre-altered output `tl_o_pre` and final output `tl_o`.
- **prim__subreg & prim__subreg__ext** (Multiple Instantiations):
  - Represent multiple instances managing specific field storage and access for registers such as interrupt state, enable, and control signals.

Inter-Module Connections

- The `gpio_reg_top` module interconnects with lower-level primitive sub-registers (e.g., `prim_subreg`, `prim_subreg_ext`) that handle specific register operations.
- It interfaces with the TL-UL protocol using two instantiated modules: one for integrity checking (`tlul_cmd_intg_chk`) and another for response generation (`tlul_rsp_intg_gen`).
- All these sub-modules rely on the core clock (`clk_i`) and reset (`rst_ni`) signals for synchronization while handling various peripherals' register states and configuration through the `reg2hw` and `hw2reg` interfaces.

This structured description provides a detailed view into `gpio_reg_top`, highlighting its significant role in the hardware design and interaction with subcomponents for handling GPIO register transactions.