HMAC-SHA256 Module Specification

Introduction

The HMAC-SHA256 module is designed to perform cryptographic hash operations using the HMAC (Hashed Message Authentication Code) construct with the SHA-256 (Secure Hash Algorithm 256-bit) hashing function. It adheres to the standards outlined in RFC 2104 for HMAC and FIPS PUB 180-4 for SHA-256, providing an efficient mechanism for message integrity and authentication in digital communication and storage systems.

Architecture

Top-level Structure

The HMAC module consists of multiple submodules that implement its functionality:

1. **HMAC Top Module**: This serves as the primary interface and orchestrates the flow of data and control signals through the entire system.
2. **HMAC Core**: Implements the HMAC logic by handling the inner and outer hash rounds as specified by HMAC standards.
3. **SHA256 Submodule**: Executes the SHA-256 hashing algorithm, performing the necessary padding, W array computations, and compression functions.
4. **Register Interface**: Manages configuration registers and interacts with external memory-mapped interfaces for configuration and result retrieval.
5. **FIFO Management**: Handles buffering of incoming data and provides a synchronous interface to the hashing core.

Key Operations

- **State Machines**: Control the flow of operation in HMAC Core and SHA256 for processing different stages of hashing.
- **Padding and Message Processing**: SHA2 includes padding logic to ensure messages conform to SHA-256 processing block sizes.
- **Error Handling**: Implements error codes and interrupt mechanisms to signal anomalies during operation.

Notable Implementation Details

- **Clock Domain**: The entire design operates in a single clock domain.
- **Configurable Parameters**: The design includes configurable alert signaling and endian swapping options for versatile deployment.
- **Pipeline Stages**: Uses pipelining in SHA256 to maintain efficient data throughput rates.

Interface

HMAC Module

Signal	Width	In/Out	Description
clk_i	1	Input	System clock input.
rst_ni	1	Input	Active-low reset signal.
tl_i	Custom	Input	Transaction Layer Input interface.
tl_o	Custom	Output	Transaction Layer Output interface.
alert_rx_i	NumAlerts	Input	Alert signals received by HMAC.
alert_tx_o	NumAlerts	Output	Alert signals transmitted by HMAC.
intr_hmac_done_o	1	Output	Interrupt output indicating HMAC operation completion.
intr_fifo_empty_o	1	Output	Interrupt indicating FIFO has become empty.
intr_hmac_err_o	1	Output	Interrupt output for HMAC errors.
idle_o	4	Output	Indicates idle state.

Internal Signals

Signal	Width	Description
reg2hw	Custom	Register to hardware interface.
hw2reg	Custom	Hardware to register feedback interface.
secret_key	256	Storage for the secret key in HMAC operations.
wipe_secret	1	Control signal to clear secret key from memory.
fifo_rvalid	1	Valid signal indicating FIFO read data availability.
msg_fifo_req	1	Request signal for FIFO message access.

Type Aliases

Alias	Type	Description
sel_rdata_t	enum logic [1:0]	Enum for selecting data sources.
round_t	enum logic	Enum for defining hash rounds.
fifoctl_state_e	enum logic [2:0]	Enum for FIFO control state machine..

Timing

- **Latency**: The HMAC and SHA-256 core processes data in a pipelined fashion, with typical latency associated with block hashing and data FIFO operations.
- **Signal Behavior**: All inputs and outputs are synchronous to the positive edge of the system clock. The reset operation occurs synchronously on the negative edge of `rst_ni`.

Usage

1. **Initialization**: Ensure all configuration registers are set. Load the secret key using the `reg2hw` interface.
2. **Data Loading**: Load data into the message FIFO. Ensure the `msg_allowed` signal marks readiness.
3. **Operation Commencement**: Start the HMAC operation by asserting control signals (`reg_hash_start`).
4. **Completion**: Monitor `intr_hmac_done_o` for operation completion. Check error codes via `intr_hmac_err_o` if necessary.
5. **Idle Check**: Confirm `idle_o` for status before initiating new operations.

This module is designed to be integrated into larger SoCs for secure message processing. Care must be taken to handle configuration, key management, and error scenarios for secure operation.

Functional Description (Generated by funcgen)

Functional Description of Verilog Design Modules

Module: hmac (File: hmac.sv)

Purpose

The `hmac` module implements the HMAC-SHA256 algorithm, designed for cryptographic message authentication. It processes input messages to generate a

secure hash using a secret key, providing integrity and authentication verification.

Parameters

- **AlertAsyncOn**: A parameter used to manage asynchronous alerts, with a default value as a repeated set bit pattern (`{NumAlerts{1'b1}}`).

Ports

- **clk_i (input, 1-bit)**: Clock input for synchronous operations.
- **rst_ni (input, 1-bit)**: Active-low reset input to initialize the module.
- **tl_i (input, `tlul_pkg::tl_h2d_t`)**: Input interface for transaction layer input to the hardware design.
- **tl_o (output, `tlul_pkg::tl_d2h_t`)**: Output interface from the module to the host.
- **alert_rx_i (input, `prim_alert_pkg::alert_rx_t [NumAlerts-1:0]`)**: Input alert signal for receiving alerts.
- **alert_tx_o (output, `prim_alert_pkg::alert_tx_t [NumAlerts-1:0]`)**: Output alert signal for alert transmission.
- **intr_hmac_done_o (output, logic)**: Indicates HMAC operation completion.
- **intr_fifo_empty_o (output, logic)**: Indicates HMAC FIFO is empty.
- **intr_hmac_err_o (output, logic)**: Indicates an HMAC error.
- **idle_o (output, `prim_mubi_pkg::mubi4_t`)**: Indicates if the HMAC core is idle.

Internal Signals

- **reg2hw/hw2reg**: Register-to-hardware and hardware-to-register signal buses for interconnecting sub-modules and managing control/status.
- **secret_key (logic [255:0])**: Stores the secret key for HMAC computation.
- **fifo signals**: Various signals to manage FIFO read/write operations, their validity, fullness, and depth.
- **msg_fifo signals**: Manage FIFO message inputs and signal flows, including error handling.
- **Control signals**: `sha_en`, `hash_start`, `cfg_block`, among others, aid in controlling the HMAC operation process.

Functionality

- **Sequential Logic**:
  - Handles the reset operation to initialize internal registers and signals.
  - Manages the control of message flow through FIFOs and updates state upon operations such as `wipe_secret`.

  - Monitors conditions to control HMAC enable states, configuration locks, and interrupts.
- **Combinational Logic**:
  - Checks alerts and errors, assigns status to outputs, and manages input/output data paths.
  - Sets up signal pathways based on configurations of SHA and HMAC processes.
- **State Machines**:
  - Controls interaction of incoming/outgoing messages, manages sequenced states of hashing processes, and computes alongside the SHA algorithm.
  - Encodes states for padding and processing of data in HMAC operations.

Instantiations

- Includes instantiations of sub-modules such as `prim_intr_hw`, `prim_packer`, `sha2`, and others for FIFO, interrupts, and packaging purposes.
- Instantiates and connects `hmac_core` and `sha2` modules, which encapsulate the core HMAC computation logic and SHA-256 algorithm, respectively.

Module: hmac_core (File: hmac_core.sv)

Purpose

The `hmac_core` module implements the core functionalities of the HMAC algorithm, directly controlling operational sequencing and SHA interactions.

Parameters

- None explicitly defined. Operates with context-specific internal definitions.

Ports

- **clk_i (input, 1-bit)**: Clock signal input for synchronization.
- **rst_ni (input, 1-bit)**: Reset signal input to reinitialize states.
- **secret_key (input, [255:0])**: Input for the 256-bit secret key.
- **wipe_secret (input, 1-bit)**: Control input for the secret wiping process.
- **wipe_v (input, [31:0])**: Wipe value input.
- **hmac_en (input, 1-bit)**: Enable signal for HMAC operation.
- **hash_done, sha_hash_start, sha_hash_process (outputs, 1-bit each)**: Control interfaces for hash lifecycle.

- **sha_rvalid, sha_rready (output/input, 1-bit each)**: FIFO control signals for reading.
- **fifo interfaces (multi-bit inputs/outputs)**: Ports managing FIFO message data pathways.

Internal Signals

- **Operational flags**: Such as `hash_start`, `clear_fifo_wdata_sel`, managing computation sequences.
- **State indexes**: `txcount`, `round_` denote HMAC rounds and byte count markers.
- **Data selectors**: For input pad and output pad data selections and computations.

Functionality

- **Sequential Logic**:
  - Controls various states for incoming message storage and processing operations.
  - Shifts secret key input to generate internal pads and manage HMAC phases from `message` to `digest`.
- **Combinational Logic**:
  - Handles input conditions for FIFO and padding, setting HMAC operational flags tied to secret key properties.
- **State Machine**:
  - Defines states from receiving initial secret key, block processing conditions (`SelIPad`,`SelOPad`), and output FIFO placements.
  - Progresses through stages by detected conditions involving txcounts and round completions.

Instantiations

- Engages with FIFO and register instances for interfacing with external and internal processes, such as padding and preparation.

Module: hmac_reg_top (File: hmac_reg_top.sv)

Purpose

This module is a register top interface designed for handling register operations and interfacing between transaction layers and hardware signals. It manages register read/write operations for `hmac`.

Parameters

- Not explicitly defined. Utilizes constants for addressing and data width purposes.

Ports

- **clk_i, rst_ni (input, 1-bit each)**: Synchronous clock and reset signals.
- **Transaction layer interface ports (in/out)**: Manages transaction data flow through inputs and outputs.
- **Register access ports [reg2hw, hw2reg]**: Provide register-to-hardware connection for reading and writing register contents.
- **intg_err_o (output, 1-bit)**: Output for reporting register integrity errors.
- **devmode_i (input, 1-bit)**: Signal mode for device error handling.

Internal Signals

- **Register control signals**: For read/write enables, data addressing, including error flag indicators.
- **Integrity check signals**: For protection through checksums and transaction integrity checks.

Functionality

- **Sequential Logic**:
  - Initializes and holds states for registers, ensuring error-free transaction processing.
- **Combinational Logic**:
  - Drives the steering and routing logic for transactions, managing the flow and processing of commands.
  - Handles error detections in data integrity and performs subsequent alarm setting or error flagging.

Instantiations

- Instantiates `prim_subreg` modules for individual register field management and value storage.
- Includes a `tlul_adapter_reg` for interfacing between transaction layers and local registers.

Module: sha2 (File: sha2.sv)

Purpose

Implements the SHA-256 cryptographic hashing algorithm. This module processes message blocks and generates message digests, handling specific data

inputs.

Parameters

- Built on local parameter definitions, including `RoundWidth` for managing rounds in SHA computation.

Ports

- **clk_i, rst_ni**: Inputs for clock/reset signals.
- **wipe_secret, wipe_v**: Inputs for scrubbing data during any reset or wipe operations.
- **FIFO interface signals (in/out)**: Flow data and generate hashes via FIFO operations.
- **Control signals [sha_en, hash_start, hash_process]**: To initiate, run, and control hash processes.
- **Digest (output)**: Receives the hash output of message digestion.

Internal Signals

- Combinational signals to control hash computations and round logic counters.
- Intermediate signals for message separation into 512-bit blocks and control the processing of these.

Functionality

- **Sequential Logic**:
  - Handles message processing across crypto stages, operating through FIFO data.
- **State Machines**:
  - Maintains states for receiving, processing, and indicating complete chunks, adjusting based on readiness.

Instantiations: - Internal, utilizing a submodule `sha2_pad` for padding and controlling SHA-256 needs.

Module: sha2_pad (File: sha2_pad.sv)

Purpose

Complements the `sha2` design to apply SHA-specific padding, ensuring messages meet required block lengths before being processed in the SHA-256 engine.

Parameters

- Uses parameter definitions from included packages and does not introduce unique parameters.

Ports

- **Standard control and status signals**: Like `clk_i`, `rst_ni`, `wipe_secret`, and data relevant ports for padding actions.
- **Message length input**: For measuring and fitting initial message bits for hash-ready processing.

Internal Signals

- Contains specific padding and length determination signals to effectively manage the process.
- Maintains flags for processing continuation and adjustments in message input lengths.

Functionality

- **Sequential Logic**:
  - Manages and aligns incoming messages with padding protocols using FIFO controls.
- **Combinational Logic**:
  - Determines conditions for data padding and prepares control signals to assist in message segment management.
- **State Machine**:
  - Holds states transitioning from direct message feeds into appended data blocks with required padding for correct SHA hash execution.

Instantiations: - Structural, retaining its processing abilities directly managing padding without child module instantiations.

Inter-Module Connections

The overall architecture centers the `hmac` module as an integrative node that coordinates sub-modules: 1. **Data Flow**: The `hmac` module orchestrates message and control signal exchanges, with primary data flow from incoming messages processed through `hmac_core` and cryptographically hashed by the `sha2` module. 2. **Control Flow**: Includes and involves significant state management for the entire synchronization between hash execution and FIFO data exchanges, controlled via multiple logic flags and transaction signals. 3. **Hierarchy**: `hmac` is the top-level module, while `hmac_core` and `sha2` are crucial for HMAC operations, building the core data processing engines, under which `sha2_pad` ensures the SHA padding compliance.

This organizational and operational layout simplifies handling SHA and HMAC operations modularly, advancing structured cryptographic procedures in hardware design.