RC4 Module Specification

Introduction

The `rc4` module implements the RC4 stream cipher's Pseudo-Random Generation Algorithm (PRGA). RC4 is a widely used stream cipher known for its simplicity and speed in software. This module is designed to generate a keystream based on a given password input, adhering to the RC4 algorithm's specifications. The implementation includes key scheduling and keystream generation phases, following the standard RC4 protocol.

Architecture

The `rc4` module is structured around a finite state machine (FSM) that manages the key scheduling and keystream generation processes. The architecture consists of the following key components:

- **Key Scheduling State Machine (KSSM):** This FSM initializes and permutes the S array based on the input key. It transitions through several states (`KSS_KEYREAD`, `KSS_KEYSCHED1`, `KSS_KEYSCHED2`, `KSS_KEYSCHED3`) to complete the key scheduling process.
- **Keystream Generation:** Once the key scheduling is complete, the module enters the `KSS_CRYPTO` state, where it continuously generates the keystream by permuting the S array and producing output bytes.
- **Registers and Arrays:**
  - `key`: Stores the input key.
  - `S`: The permutation array used in the RC4 algorithm.
  - `i`, `j`: Indices used for array manipulation.
  - `discardCount`: Counter to discard the initial keystream values as per RFC 4345.

Interface

The interface of the `rc4` module is defined as follows:

Signal	Width	In/Out	Description
clk	1	In	Clock signal (wire)
rst	1	In	Reset signal, active high (wire)
password_input	8	In	Password input for key scheduling (wire)
output_ready	1	Out	Indicates when the output keystream is valid (reg)
K	8	Out	Output keystream byte (reg)

Internal Signals

Signal	Width	Description
key	8	Array storing the input key
S	8	Permutation array used in the RC4 algorithm
discardCount	11	Counter for discarding initial keystream values
KSState	4	Current state of the key scheduling FSM
i	8	Index for array manipulation
j	8	Index for array manipulation

Timing

The `rc4` module operates synchronously with the clock signal. The key scheduling process takes several clock cycles to complete, depending on the key size. The keystream generation begins after the key scheduling is complete, with the `output_ready` signal indicating when the output byte `K` is valid. The initial 1536 keystream values are discarded to enhance security, as recommended by RFC 4345.

Usage

To use the `rc4` module, follow these steps:

1. **Initialization:** Apply a reset signal to initialize the module. The reset signal should be held high for at least one clock cycle.
2. **Key Input:** Provide the password input byte-by-byte through the `password_input` signal. The module will read the key and transition through the key scheduling states.
3. **Keystream Generation:** Once the key scheduling is complete, the module will automatically begin generating the keystream. Monitor the `output_ready` signal to determine when the output byte `K` is valid.
4. **Continuous Operation:** The module will continue to generate the keystream indefinitely, allowing for continuous data encryption or decryption.

This specification provides a comprehensive overview of the `rc4` module, enabling hardware engineers to understand and implement the design from scratch.

Functional Description (Generated by funcgen)

RC4 Verilog Module Description

Module Name: rc4

- **Source File**: rc4.v

Purpose The `rc4` module implements the RC4 PRGA (Pseudo-Random Generation Algorithm) stream cipher. This cryptographic algorithm outputs a pseudorandom byte stream, which is commonly used for data encryption.

Parameters

- The module does not explicitly define parameters, though it uses a `KEY_SIZE` parameter inferred from an included file (`rc4.inc`).

Ports

- **clk** (input, 1-bit): Clock signal driving the module's sequential logic.
- **rst** (input, 1-bit): Active-high reset signal used to initialize the internal state of the module.
- **password_input** (input, 8-bit): Input for the encryption key, used during the key-scheduling phase.
- **output_ready** (output, 1-bit): Signal indicating the validity of the output data.
- **K** (output, 8-bit): Byte of the keystream generated by the PRGA, used in encryption or decryption.

Internal Signals

- **output_ready** (reg, 1-bit): Internal signal tracking the readiness of the output keystream byte.
- **key** (reg array, 8-bit x `KEY_SIZE`): Holds the input key for key scheduling.
- **S** (reg array, 8-bit x 256): Permutation array used in the RC4 algorithm.
- **discardCount** (reg, 11-bit): Counter to discard the initial values of the keystream for improved security, as per RFC 4345.
- **KSState** (reg, 4-bit): Current state of the key-scheduling and PRGA process.
- **i, j** (reg, 8-bit each): Indices used in the algorithm to manipulate the permutation array `S`.

Functionality

Sequential Logic The module predominantly employs synchronous logic driven by the `clk` signal, managing states and transitions through a finite state machine (FSM):

1. **Reset Behavior**: On reset activation, the module initializes indices (`i`, `j`) and sets the FSM to the key-reading state (`KSS_KEYREAD`). `output_ready` is cleared.
2. **State Machine**: The main FSM orchestrates the RC4 process:
    - **KSS_KEYREAD**: Reads input keys into the `key` array until it is filled.

- **KSS_KEYSCHED1**: Initializes array `S` where `S[i]` is set to `i`.
- **KSS_KEYSCHED2**: Performs initial permutation of `S` using the key.
- **KSS_KEYSCHED3**: Finishes permutation setup using nested swaps.
- **KSS_CRYPTO**: Generates the pseudorandom keystream. Values of `S[i]` and `S[j]` are swapped, and keystream byte `K` is calculated as `S[(S[i] + S[j]) mod 256]`.

3. **Output Logic**: The module signals a valid keystream byte on `output_ready` after discarding the initial 1536 values, enhancing security.

Combinational Logic

- Calculations of indices and swaps within the state transitions (`j` updates, `K` computation) involve critical combinational logic.

State Machines

- The FSM transitions manage key reading, scheduling, and keystream generation. Its states (`KSS_KEYREAD`, `KSS_KEYSCHED1`, `KSS_KEYSCHED2`, `KSS_KEYSCHED3`, `KSS_CRYPTO`) systematically transition upon completing actions.

Instantiations

- No explicit sub-module instantiations in the given design indicate self-contained functionality.

Inter-Module Connections

- The `rc4` module encompasses internal logic without sub-modular breakdown, following a self-sufficient design paradigm that facilitates streamlined use in various encryption tasks. The array `key` and the state of the `S` array form an implicit internal hierarchy central to the algorithm's completion.

The module's infrastructure efficiently manages the flow from key input to keystream output across a well-defined FSM, ensuring clear state transitions aligned with the RC4 algorithm's specification.