**Advanced Encryption Standard**

**Rijndael IP Core**

*Author: Rudolf Usselmann rudi@asics.ws www.asics.ws*

Rev. 1.1 November 12, 2002

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Rev.** | **Date** | **Author** | **Description** |
| 1.0 | 11/9/02 | Rudolf Usselmann | Initial Release |
| 1.1 | 11/12/2002 | RU | Fixed Several Typos |
|  |  |  |  |
|  |  |  |  |

**Table of Contents**

[List of Figures iii](#_Toc199889545)

[List of Tables iv](#_Toc199889546)

[1.0 Introduction 5](#_Toc199889547)

[2.0 Architecture 6](#_Toc199889548)

[3.0 Operation 6](#_Toc199889549)

[4.0 Dummy State Machine 7](#_Toc199889550)

[4.1 S\_IDLE 8](#_Toc199889551)

[4.2 S\_RUN 8](#_Toc199889552)

[4.3 S\_WAIT 9](#_Toc199889553)

[4.4 S\_DONE 9](#_Toc199889554)

[5.0 Clock 9](#_Toc199889555)

[6.0 Clock Domains 9](#_Toc199889556)

[7.0 Reset Domains 10](#_Toc199889557)

[8.0 Module Hierarchy 10](#_Toc199889558)

[9.0 IO Ports 11](#_Toc199889559)

[10.0 RTL Parameters 12](#_Toc199889560)

# List of Figures

[Figure 1 Cipher Core Architecture Overview. 6](#_Toc199889561)

[Figure 2 AES cipher core interface timing. 7](#_Toc199889562)

[Figure 3 dummy\_sm module state machine diagram. 8](#_Toc199889563)

# List of Tables

[Table 1 Clock Requirements 9](#_Toc199889564)

[Table 2 aes\_ciper\_top module IO ports 11](#_Toc199889565)

[Table 3 aes\_key\_expand\_128 module IO ports 12](#_Toc199889566)

[Table 4 aes\_sbox module IO ports 12](#_Toc199889567)

[Table 5 aes\_rcon module IO ports 12](#_Toc199889568)

[Table 6 dummy\_sm module IO ports 12](#_Toc199889569)

[Table 7 aes\_cipher\_top module default RTL parameters 13](#_Toc199889570)

# Introduction

Simple AES/Rijndael IP Core. I have tried to create an implementation of this standard that would fit into a low-cost FPGA, like the Spartan IIe series from Xilinx, and still would provide reasonably fast performance.

This implementation is with a 128-bit key expansion module only. Implementations with different key sizes (192 & 256 bits) and performance parameters (such as a fully pipelined ultra-high-speed version) are commercially available from ASICS.ws (www.asics.ws).

This document will describe the interface to the IP core. It will not talk about the AES standard itself.

# Architecture

The AES Rijndael core consists of two blocks: 1) The AES Cipher block which performs encryption; 2) The AES Inverse Cipher block which performs decryption. Both blocks instantiate the same key expansion block. Figure 1 illustrates the overall architecture of the AES Cipher core.

A diagram of a process

AI-generated content may be incorrect.

Figure 1 Cipher Core Architecture Overview.

The AES cipher core consists of a key expansion module, an initial permutation module, a round permutation module, and a final permutation module. The round permutation module will loop internally to perform 10 iterations (for 128-bit keys).

# Operation

The forward cipher block can perform a complete encrypt sequence in 12 clock cycles (10 cycles for the 10 rounds, plus one cycle for initial key expansion, and one cycle for the output stage).

The forward cipher block accepts a key and the plain text at the beginning of each encrypt sequence. The beginning is always indicated by asserting the ‘ld’ pin high. When the core completes the encryption sequence it will assert the ‘done’ signal for one clock cycle to indicate the completion. The user might choose to ignore the ‘done’ output and time the completion of the encryption sequence externally. Figure 2 shows the waveform of AES core.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 2 Waveform of AES core.

# Dummy State Machine

The dummy\_sm module is a dummy state machine. It sits between a single cycle **trigger** pulse and whatever logic you want to wrap around it, sequencing through a fixed “idle → run → wait → done” pipeline before returning to **idle.** All state transitions occur on the rising edge of **clk** and the state machine is synchronously reset to **S\_IDLE** when **rst** is asserted. Figure 3 shows the flow of the state machine.

A diagram of a diagram

AI-generated content may be incorrect.

Figure 3 dummy\_sm module state machine diagram.

## S\_IDLE

* **Purpose:** Hold the machine in a quiescent, ready-for-work condition.
* **Actions:**
  + Keeps internal state registers unchanged.
  + Waits for either rst or a new trigger pulse.
* **Transition Conditions:**
  + **From S\_IDLE to S\_RUN:** trigger is high (when the trigger is high, it indicates that an external trigger for the AES operation).
  + **From S\_IDLE to S\_IDLE:** trigger is low (if the trigger remains low, wait for the trigger to be active).

## S\_RUN

* **Purpose:** Start the active portion of the cycle. In real logic this is where you would launch whatever task needs to “run.”
* **Actions:**
  + - Performs a single-cycle “kick-off” of the requested task.
    - No branch decisions are made here; the machine simply advances.
* **Transition Conditions:**
  + - **From S\_RUN to S\_WAIT:** Always, on the very next rising edge of **clk**.

## S\_WAIT

* **Purpose:** Provide an intermediate wait or latency slot between run and done. This could model a pipeline stage, a busy timer, etc.
* **Actions:**
  + Maintains current outputs or internal counters (implementation-dependent).
  + Prepares to assert completion on the following cycle.
* **Transition Conditions:**
  + **From S\_WAIT to S\_DONE:** Always, on the next rising edge of **clk**.

## S\_DONE

* **Purpose:** Signal that the requested operation finished and return the FSM to its idle state.
* **Actions:**
  + Optionally raise a done flag or interrupt.
  + Clear any temporary resources before looping back to idle.
* **Transition Conditions:**
  + **From S\_DONE to S\_IDLE:** Always, on the next rising edge of **clk**.

# Clock

The AES has been tested with a clock as fast as 200 MHz (Table 1).

Table 1 Clock Requirements

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Source** | **Rates (MHz)** | | **Description** |
|  |  | **Max** | **Min** |  |
| clk | (System) | 200 MHz |  | System clock |

# Clock Domains

In digital design, clock domains play a crucial role in defining how different components within a system synchronize and interact. In the case of the AES core, the clocking mechanism is relatively simple yet effective, as it operates primarily within a single clock domain. Modules namely aes\_cipher\_top, aes\_key\_expand\_128, aes\_rcon and dummy\_sm are driven by a single input clock clk. This common clock source propagates through all the connected sub-modules, ensuring synchronized operation throughout the AES core. aes\_sbox is a combinational logic module with no clock input.

# Reset Domains

The AES core operates under a single reset domain, ensuring module namely aes\_cipher\_top, and dummy\_sm are reset simultaneously for synchronized operation. The single reset domain supports reliable data transmission and enhances system stability, ensuring smooth error recovery. aes\_sbox, aes\_rcon, and aes\_key\_expand\_128 are combinational logic modules with no reset input.

# Module Hierarchy

The AES core is organized in three clean hierarchy levels:

* **Top-level module** aes\_cipher\_top – wraps the complete cipher
  + Instance u0 of aes\_key\_expand\_128: expands the 128-bit user key into 44 round words.
  + Sixteen instances (us00, us01, us02, us03, us10, us11, us12, us13, us20, us21, us22, us23, us30, us31, us32, us33) of aes\_sbox: perform the byte-parallel SubBytes step on the 4 × 4 state matrix each cycle.
  + Instance u\_dummy\_sm of dummy\_sm: tiny state machine that sequences idle → run → wait → done after every ld start pulse.
* **Second level module** aes\_key\_expand\_128 – builds the round-key schedule
  + Four instances (u0, u1, u2, u3) of aes\_sbox: realise the SubWord substitution used in key expansion.
  + Instance r0 of aes\_rcon: supplies the 10 AES-128 round constants (Rcon[i]).
* **Leaf modules** – pure functional blocks
  + aes\_sbox: combinational Rijndael S-box (inverse path unused in this design).
  + aes\_rcon: small sequential LUT/counter that outputs successive round constants.
  + dummy\_sm: example 4-state finite-state machine for simple control or test‐bench driving.

This structure keeps control (dummy\_sm), key scheduling (aes\_key\_expand\_128) and data-path transformations (aes\_sbox) strictly separate, enabling easy verification, reuse and potential upgrades (e.g., wider keys or different controllers) without touching the core cipher logic.

# IO Ports

The AES-128 core is made up of five synthesizable Verilog modules. Each module exposes a small, self-contained interface so it can be verified and reused independently or stitched together into a larger System on Chip.

Table 2 through Table 6 provide a concise, module-by-module guide to every external signal in the AES-128 core.

Table 2 covers the top-level wrapper, aes\_cipher\_top, enumerating the clocks, resets, data/key buses, start pulse, and completion flag that a system integrator must wire up to feed plaintext and capture ciphertext.

Table 3 then moves inside the design to document the aes\_key\_expand\_128 key-schedule unit, explaining how the user key is loaded and how its four 32-bit round-key words emerge each cycle.

Table 4 records the ultra-simple two-port interface of the combinational aes\_sbox byte-substitution function that is instanced throughout the cipher data path.

The single-purpose aes\_rcon block, which generates the Rijndael round constants used by the key scheduler, is described in Table 5.

Finally, Table 6 outlines the clock, reset, trigger, and state outputs of dummy\_sm, the four-state demonstration controller included for test-bench or integration examples.

Table 2 aes\_ciper\_top module IO ports

|  |  |  |  |
| --- | --- | --- | --- |
| **Port** | **Width** | **Direction** | **Description** |
| clk | 1 | Input | System clock |
| rst | 1 | Input | Synchronous, active-high reset |
| ld | 1 | Input | Single-cycle “start” pulse; latches key & text\_in |
| done | 1 | Output | Goes high for one cycle when the 10-round cipher finishes |
| key | 128 | Input | Secret key, captured when ld is asserted |
| text\_in | 128 | Input | Plain-text (encryption) or cipher-text (decryption) block |
| text\_out | 128 | Output | Cipher-text (encryption) or plain-text (decryption) result |

Table 3 aes\_key\_expand\_128 module IO ports

|  |  |  |  |
| --- | --- | --- | --- |
| **Port** | **Width** | **Direction** | **Description** |
| clk | 1 | Input | System clock |
| kld | 1 | Input | Pulse high to (re-)load the user key |
| key | 128 | Input | 128-bit user key |
| wo\_0 | 32 | Output | First 32-bit word of the current round-key |
| wo\_1 | 32 | Output | Second 32-bit word of the current round-key |
| wo\_2 | 32 | Output | Third 32-bit word of the current round-key |
| wo\_3 | 32 | Output | Fourth 32-bit word of the current round-key |

Table 4 aes\_sbox module IO ports

|  |  |  |  |
| --- | --- | --- | --- |
| **Port** | **Width** | **Direction** | **Description** |
| a | 8 | Input | Byte to be substituted |
| d | 8 | Output | S-box output byte (S(a)) |

Table 5 aes\_rcon module IO ports

|  |  |  |  |
| --- | --- | --- | --- |
| **Port** | **Width** | **Direction** | **Description** |
| clk | 1 | Input | System clock |
| kld | 1 | Input | Pulse high to reset the internal counter |
| out | 32 | Output | Current Rcon[i] constant (placed on the MSB of the expanding word) |

Table 6 dummy\_sm module IO ports

|  |  |  |  |
| --- | --- | --- | --- |
| **Port** | **Width** | **Direction** | **Description** |
| clk | 1 | Input | System clock |
| rst | 1 | Input | Active-high synchronous reset |
| trigger | 1 | Input | Start pulse |
| state | 2 | Output | Encoded FSM state |

# RTL Parameters

RTL parameters are fundamental to shaping an AES core’s behavior and resource profile. They let designers fine-tune everything from supported key lengths and round counts to pipeline depth and S-box style. This parameter-driven flexibility ensures the design remains both portable and scalable across a broad spectrum of security-critical applications. Table 7 shows default RTL parameters for aes\_cipher\_top module. No RTL parameters in aes\_key\_expand\_128, aes\_sbox, aes\_rcon, dummy\_sm.

Table 7 aes\_cipher\_top module default RTL parameters

|  |  |  |
| --- | --- | --- |
| **RTL Parameter Name** | **Default RTL Parameter Value** | **Description** |
| DUMMY\_PARAM0 | 0 | Dummy top level RTL parameter 0. |
| DUMMY\_PARAM1 | 32’hDEADBEEF | Dummy top level RTL parameter 1. |
| DUMMY\_PARAM2 | 8'd42 | Dummy top level RTL parameter 2. |