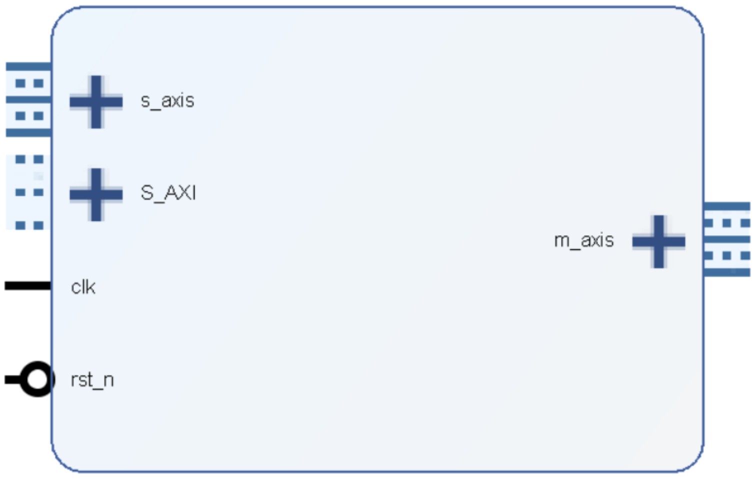

AES-256-CTR MODE ENCRYPTION/ DECRYPTION CORE

VHDL Implementation

ABSTRACT
Open-source pipelined VHDL Implementation of AES 256 CTR mode encryption and decryption.

Author: Stian K. Endresen

1 Table of Contents

2 Introduction

The AES-256-CTR Encryption Core offers a ready-made open-source VHDL implementation of CTR encryption and decryption with AES256. The design exposes generics for tailoring the design to the users needs.

2.1 Features

- A ready-made FPGA CTR encryption/decryption solution with built-in keygen.
- AXI4-Lite interface that allows for configuring and reconfiguring the key and IV.
- 128-bit AXI4-Stream interfaces for the plaintext and ciphertext.
- A compile-time customizable CTR Counter Width.
- A user-configurable number of AES Cores, giving a high degree of customization over the tradeoff between resource usage and throughput.
- An optional, configurable keystream buffer for precomputing keystream blocks to be ready when plaintext data arrives, and for avoiding throughput loss imposed by wait-cycles when using AXI4-Stream Data Width Convertors on the input or output.

3 CTR-mode Encryption Theory

3.1 Overview

CTR is one of the most common modes of AES encryption. It requires a key (256 bits for AES-256) and an Initialization Vector (IV). The width of the IV matches the block cipher width (128 bits with AES). The IV is divided into a nonce and a counter. A common choice is 96 bits for the nonce and 32 bits for the counter. The main two considerations when using CTR is:

- The same key/nonce pair should never be reused for more than one plaintext stream.
- With a given key/nonce pair, one should never encrypt more than 2^{CW} AES-blocks, where CW is the Counter Width in bits. Since each AES-block is 128 bits, this equates to 68 GB with a 32-bit counter.

Failure to adhere to any of these requirements will result in a critical security violation. Furthermore, while CTR provides excellent data confidentiality, it lacks any form of authentication, and thus cannot detect if the ciphertext is tampered with.

3.2 CTR mode details

CTR-mode encryption works by generating a *keystream* from the key and IV. $Ciphertext = Keystream\ XOR\ Plaintext$. The keystream is generated in blocks of 128 bits/16 bytes. To generate the first block, the (once||counter) is sent into the AES Core, and to get the second block, the (nonce||counter+1) is used as inputs, and (nonce||counter+2) for the third block, etc. An illustration of this is provided below. A common choice is to set the initial counter value to 0 at the start of the encryption, but other initial counter values are allowed.

To decrypt the ciphertext, the exact same procedure is used as when encrypting. $Plaintext = Keystream\ XOR\ Ciphertext$. In mathematical terms, this means that encryption and decryption are symmetric operations:

$$ciphertext = CTR\_encrypt(\ plaintext\ )$$

$$plaintext = CTR\_encrypt(\ ciphertext\ )$$

This symmetry between encryption and decryption means that the same hardware instance can be used for both encryption and decryption without modification, even without switching operation mode.

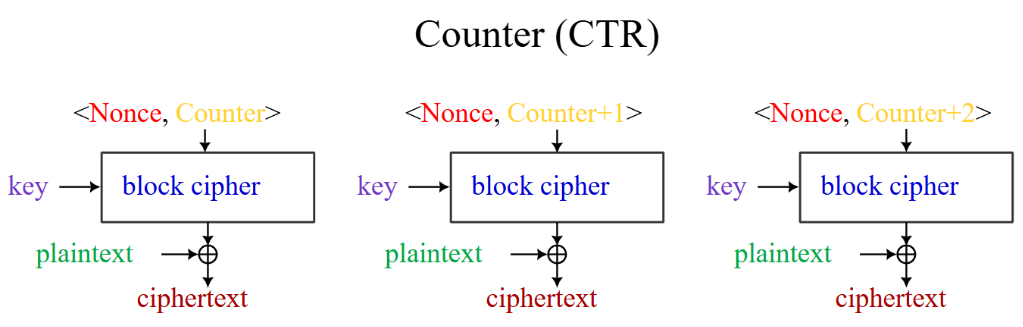

Figure 1: The workings of CTR-mode encryption illustrated *[Image credit: A.M. Rowsell and Epachamo[i]].*

4 Documentation of Implemented Design

4.1 Ports and Interfaces

The high-level interfaces of the implemented design are illustrated in Figure 2. A documentation of the purpose and operation of the inputs and outputs is given below.

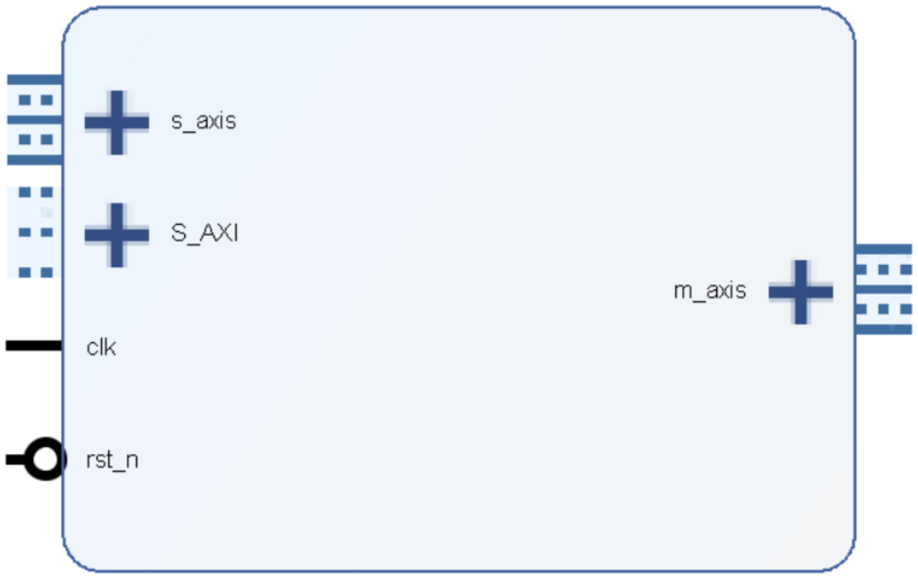

Figure 2: Illustration of the ports and interfaces for the AES-256-CTR Core.

As shown by S_AXI in the block design in Figure 2, the encryption core exposes an AXI4 Lite interface for setting the key, IV and control signals for the encryption core, as well as a readback of status signals. See chapter 4.2 and 4.3 for a detailed description of this.

The s_axis and m_axis interfaces are both 128-bit interfaces used for the plaintext and ciphertext. When encrypting data, plaintext is sent to the s_axis interface, and ciphertext is being generated on m_axis. When decrypting data, ciphertext is inputted on s_axis while the decrypted plaintext is generated on m_axis.

The reset signal rst_n is active low, and is used by both the AXI4 Lite interface, the AXI4-Stream interface and the internal AES256 encryption process.

4.2 Register Space

The following table describes the configuration registers:

Register Offset (hex)	Register Name	Reset Value (hex)	Access Type	Description
0x00	Control Register	0x0000_0000	R/W	See section 4.2.1 and 4.3 for a description of the control register.
0x04	Status Register	0x0000_0000	R	See section 4.2.1 and 4.3 for a description of the status register.
0x08	Key Part 0	0x0000_0000	R/W	Bits 0 to 31 of the 256-bit encryption key.
0x0C	Key Part 1	0x0000_0000	R/W	Bits 32 to 63 of the 256-bit encryption key.
0x10	Key Part 2	0x0000_0000	R/W	Bits 64 to 95 of the 256-bit encryption key.
0x14	Key Part 3	0x0000_0000	R/W	Bits 96 to 127 of the 256-bit encryption key.
0x18	Key Part 4	0x0000_0000	R/W	Bits 128 to 159 of the 256-bit encryption key.
0x1C	Key Part 5	0x0000_0000	R/W	Bits 160 to 191 of the 256-bit encryption key.
0x20	Key Part 6	0x0000_0000	R/W	Bits 192 to 223 of the 256-bit encryption key.
0x24	Key Part 7	0x0000_0000	R/W	Bits 224 to 255 of the 256-bit encryption key.
0x28	IV Part 0	0x0000_0000	R/W	Bits 0 to 31 of the 128-bit IV. Contains the counter.
0x2C	IV Part 1	0x0000_0000	R/W	Bits 32 to 63 of the 128-bit IV.
0x30	IV Part 2	0x0000_0000	R/W	Bits 64 to 95 of the 128-bit IV.
0x3C	IV Part 3	0x0000_0000	R/W	Bits 96 to 127 of the 128-bit IV.

4.2.1 Config Register and Control Register

Control Register

Bit	Default value	Signal name
0	0	load_key_and_iv
1	0	tx_raw_keystream
2-31	0	Not in Use

Status Register

Bit	Default value	Signal name
0	0	key_ready
1	0	tx_raw_keystream
2-31	0	Not in Use

4.3 Description of Operation

4.3.1 Key expansion

Before encryption or decryption can occur, a key/IV pair must be loaded. This is done by setting the `load_key_and_iv` bit in the control register to 1 for at least one cycle while the desired key and IV are loaded in their respective. Both the nonce and the initial counter value are set from the provided IV. When `load_key_and_iv` in the control register is asserted, `key_ready` in the status register is immediately set to 0. When `load_key_and_iv` is de-asserted, the key and IV pair are loaded into the device and key expansion starts. When the key expansion completes after 84 clock cycles, the `key_ready` signal in the status register is set to 1 and the AES core is ready for encrypting/decrypting. The AXI registers containing the key can now be set to zero, as they are unused until `load_key_and_iv` is re-asserted. The AXI registers containing the nonce-part of the IV, however, must not be changed during encryption or decryption.

If `load_key_and_iv` is set to 1 during key expansion or normal operation, the AES Core immediately aborts its current operation in preparation for key expansion with the new key. A new key and IV pair can be loaded as many times as desired. The recommended way of doing this is by first setting the `load_key_and_iv` bit in the control register high, and then write new values to the key and IV registers before finally de-asserting the `load_key_and_iv` bit in the control register.

4.3.2 Normal operation

Following completion of key expansion, it will take 60 cycles to generate the first keystream blocks. When a keystream block is ready, the s_axis interface will accept incoming plaintext data and xor with the keystream to encrypt. The resulting chiphertext is available on m_axis on the next cycle. For decryption, the same procedure and data flow as for encryption is used.

4.3.3 Key and IV management

The counter in the IV is auto-incremented whenever a block is encrypted. Readback of the active counter value is not supported. To change the key or IV, one must repeat the steps in section 4.3.1.

4.3.4 Transmit keystream mode

The CTR Core supports a special operation mode designed for situations where the limiting factor on throughput is the speed at which data can be moved between the PL and PS. If the `tx_raw_keystream` bit in the control register is set to 1, the core will ignore the s_axis interface and transmit the keystream directly on the m_axis interface. The PL must perform XOR between the plaintext and keystream to generate the ciphertext (or vice verca for decryption), but the amount of data that must move between the PL and PS is cut in half. The `tx_raw_keystream` bit in the status register is a readback of the value of `tx_raw_keystream` in the control register.

Note that if this approach is used, care must be taken when changing key/IV pair, that every keystream block from the old encryption session is flushed out from FIFO buffers in the FPGA etc. to avoid mixing the keystreams.

4.4 Generics

The design exposes three generics to allow customizing the design to meet the users' needs.

4.4.1 IV_COUNTER_WIDTH

Allowed values: [32-128]

Purpose: Controls the width of the counter for the IV, which determines when the counter overflows and wraps around. When set to 32, the counter is entirely contained within IV Part 0 (address 0x28). With greater counter widths, the counter spills over into the LSB-section of IV Part 1 (address 0x2C), etc.

4.4.2 NUM_AES_CORES

Allowed values: [1-5, 8, 15]

Purpose: The main parameter for specifying the tradeoff between throughput and resource usage. A higher number of cores increases throughput and resource usage. With the maximum 15 cores, one 128-bit plaintext block is encrypted every clock cycle. See section 4.5 for more details.

4.4.3 KEYSTREAM_BUFFER_SIZE

Allowed values: [≥ 0]

Purpose: Adds a buffer for storing up keystream blocks in advance, to be ready to encrypt a sudden influx of plaintext blocks. E.g. a value of 20 will precompute and store up to 20 128-bit blocks.

If Data Width Convertors are used on the input or output of the CTR encryption core, it Is recommended to set the keystream buffer to 3 blocks or more to avoid a throughput reduction (see section 4.5.1).

When using 15 AES cores, the value of the KEYSTREAM_BUFFER_SIZE generic will be ignored, since one AES-block is already encrypted every clock cycle.

4.5 Performance and Utilization

The tradeoff between throughput and resource utilization is highly configurable. By increasing the generic NUM_AES_CORES, several AES Cores will be used, increasing both throughput and resource usage. Allowed values for NUM_AES_CORES are {1-5, 8, 15}. At 15 cores, a full 128-bit block is encrypted every clock cycle.

The implemented design is synthesized for a zynq7000 to estimate the resource usage The result is shown below, as well as an estimated throughput for each configuration. The synthesized design reports maximum clock rates in the range 250 – 300 MHz.

Number of cores	LUTs	Slice Registers	F7 Muxes	F8 Muxes	Block RAM Tiles	Throughput multiplier	Throughput at 250 MHz
1	3232	3259	768	352	0	1	2.133 Gbps
2	4202	3921	1152	480	0	1.875	4.0 Gbps
3	5101	4585	1280	608	0	3	6.4 Gbps
4	6153	5229	1537	736	0	3.75	8.0 Gbps
5	6780	5881	1793	864	0	5	10.667 Gbps
8	9316	7422	2310	1120	0	7.5	16.0 Gbps
15	16226	11929	4096	2016	71	15	32.0 Gbps

4.5.1 Throughput considerations

If data is not always available on the input AXIS interface, or not always read at the output AXIS interface (like when AXIS Data Width Convertors are used), the KEYSTREAM_BUFFER_SIZE generic must be set to 3 or more to avoid reducing the throughput. The reasoning for this is as explained below.

The implemented design pipelines four 128-bit keystream blocks at once, meaning that 4 blocks of data can be encrypted or decrypted every 60/32/20/16/12/8/4 clock cycles with 1/2/3/4/5/8/15 cores. The keystream generator will halt keystream generation when a keystream block is ready but s_axis is empty or m_axis is blocking. This becomes important if data width convertors are used on the input or output, as with e.g. 32-to-128-bit convertors on the input, plaintext blocks are only available to the CTR encryptor every 4 cycles. Since all four keystream blocks must be used before new keystream blocks are generated, this imposes 9 idle cycles, decreasing throughput.

This issue is solved by adding a 3-block wide FIFO buffer for the keystream blocks, which is achieved by setting the KEYSTREAM_BUFFER_SIZE generic to 3. A 3-block keystream buffer uses an additional 150 LUTs and 400 slice registers. Regardless of the number of cores or the data width conversion ratio, 3 blocks will suffice to counteract the added delay from data width convertors, but a larger keystream buffer will achieve the same purpose.

5 Acknowledgements

This design builds on the open-source VHDL implementation of the AES256-Core by Aleksandar Lilic, containing both key generation and AES-encryption. Compared to his implementation, this design features:

- A pipelined AES Core, giving 4x throughput with no meaningful increase in resource usage.
- A customizable number of AES-Cores, exposing a user-configurable tradeoff between throughput and resource usage.
- A standard AXI4-Stream interfaces for the plaintext and ciphertext.
    - Subsequently, the old *data_loading* and *aes256_loading* modules are no longer used.
- A standard AXI-Lite slave interface for writing key/iv/config and reading status
- Implemented CTR-mode encryption on top of the AES Core.

The AXI-Lite slave interface is based on the design by mzeghers (https://github.com/mzeghers/hdl-axi-regs), though some modifications are performed. Notably, bresp and rresp now return SLVERR when incorrect addresses are given, and bvalid is only driven high when applicable.

6 Footnotes

[i] Image by A.M. Rowsell and Epachamo, taken from: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#/media/File:BlockCipherModesofOperation.svg, license: CC BY-SA 4.0