Speedster7t Cryptographic Engine User Guide (UG104)

Speedster FPGAs

Preliminary Data

Achronix
Data Acceleration

Copyrights, Trademarks and Disclaimers

Preliminary Data

This document contains preliminary information and is subject to change without notice. Information provided herein is based on internal engineering specifications and/or initial characterization data.

Achronix Semiconductor Corporation

2903 Bunker Hill Lane
Santa Clara, CA 95054
USA

Website: www.achronix.com
E-mail : info@achronix.com

Table of Contents

Chapter - 1: Description

The Cryptographic engine, **ACX_AESX_GMC_K**, supports data encryption/decryption and implements an AES algorithm using Rijndael encoding and decoding in compliance with the NIST Advanced Encryption Standard. The encryption is suitable for a variety of applications in the public and private domain.

The Advance Encryption Standard (AES) is a symmetric block cipher chosen by the US government to protect classified information. Symmetric, also known as *secret key*, ciphers use the same key for encrypting and decrypting so the sender and receiver must both know and use the same secret key. Compared to the DES and triple DES algorithms, AES provides a higher level of security because it has a larger key size and is also faster. In addition, DES has become vulnerable to brute-force attacks.

The Cryptographic engine core is pre-placed and pre-routed. Although it is implemented in the fabric, it can be considered a hard IP core because the placement and routing cannot be modified.

All of the inputs are synchronous to the clock signal, `clk`. The Cryptographic engine processes 128-bit blocks of messages using a 128-bit fixed-length key. The data input interface is 128 bits wide but the data can be input with byte resolution using the `ibyte` and `last_w` inputs. The `last_w` signal indicates the last word being input. The `ibyte` signal is ignored until `last_w` is asserted, indicating the number of valid bytes in the last word minus 1 (counted from the MSB). So `ibyte` = "0000" means that only the first byte in the incoming word is valid and `ibyte`="1111" means that all bytes are valid.

Example

There is an example reference design for the Cryptographic engine included in the Speedster 2D NoC Reference Design. This design can be freely obtained by contacting Achronix at support@achronix.com

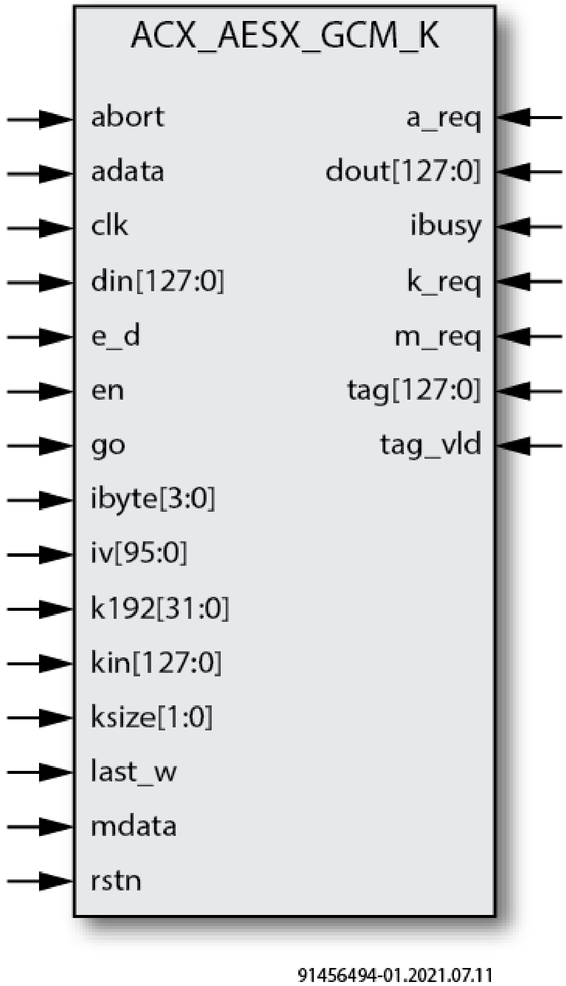

Figure 1: ***ACX_AESX_GMC_K Symbol***

Ports

Table 1: ***Port Description***

Name	Direction	Width	Description
`rstn`	Input	1	Active low asynchronous reset.
`clk`	Input	1	Clock signal.
`en`	Input	1	Synchronous enable signal.
`go`	Input	1	Starts cryptographic operation when = 1.
`abort`	Input	1	Aborts current operation when = 1.
`e_d`	Input	1	Mode signal: • Encryption when = 0 • Decryption when = 1
`kin[127:0]`	Input	128	Key data input.
`ksize[1:0]`	Input	2	Input key size. Not user programmable.
`k192[31:0]`	Input	31	Unexpanded key.
`din[127:0]`	Input	128	Input data: • Contains *Additional* input data when `adata` = 1 • Contains *Message* input data when `mdata` = 1 • `adata` and `mdata` are mutually exclusive cannot be 0 at the same time
`iv[95:0]`	Input	96	Initialization Vector.
`adata`	Input	1	Additional data is input when `adata` = 1.
`mdata`	Input	1	Message data is input when `mdata` = 1.
`ibyte[3:0]`	Input	4	Indicates the number of valid bytes in the last `din` word – 1. Valid when `last_w` is asserted. (1) • `ibyte` = 4'h0 = 1 byte • `ibyte` = 4'hf = 16 bytes, (full 128-bit word)
`last_w`	Input	1	When = 1, last Additional or Message data word is input. Validates `ibyte[3:0]` input.
`k_req`	Output	1	When = 1, the unexpanded key is requested.

Name	Direction	Width	Description
a_req	Output	1	When = 1, Additional data is requested.
m_req	Output	1	When = 1, Message data is requested.
ibusy	Output	1	When = 1, the core is in the initialization process.
dout[127:0]	Output	128	Processed message data output.
tag[127:0]	Output	128	Authenticated tag value output.
tag_vld	Output	1	Authenticated tag value valid output.

Table Notes

1. ibyte is scaled differently from many other last byte values. It is equal to (last din word – 1). If moving between other interfaces with mod signals, be aware of this scaling difference.

Chapter - 2: Operation

The **ACX_AESX_GMC_K** core supports both encryption and decryption according to the AES algorithm.

The rising edge on the `go` port triggers the beginning of a cryptographic operation using the `key` input as the key. The `en` signal must be asserted one cycle before the `go` signal. The `mdata` and `adata` signals must remain stable.

When the core is started, it requests the unexpanded key (one 128-bit word) by raising `k_req`. The application must assert the kin input on the same cycle that `k_req` is asserted, (not on the following cycle). After 14 cycles from `ibusy` being asserted, the core is ready to accept message data or additional data.

This is known as the initialization phase and it is indicated by the `ibusy` signal being asserted.

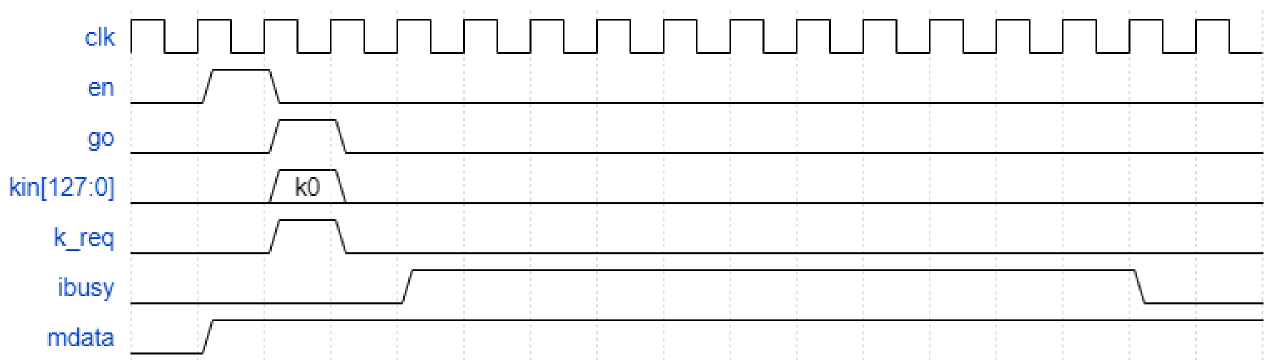

Figure 2: ***Initialization Phase***

After initialization, the core asserts `m_req` on the third clock cycle after the falling edge of `ibusy` if `mdata` is asserted, indicating that the core is now accepting message data on every clock cycle. The application must apply data to the core on the assertion of `m_req`, (not on the cycle following `m_req`). Therefore it is suggested that the application count cycles from the de-assertion of `ibusy` to ensure that the first word of data is applied on the same cycle that `m_req` is asserted.

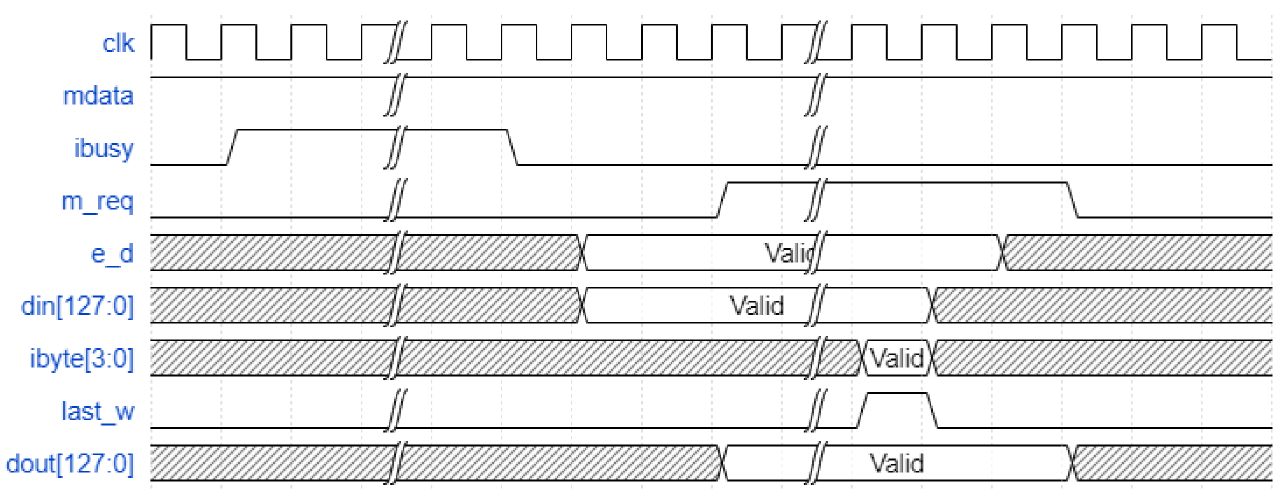

Figure 3: ***Message Data Input and Output***

The core asserts a_req on the third clock cycle after the falling edge of ibusy if adata is asserted. The type of data requested is indicated by the a_req and m_req signals for additional and message data respectively.

The output is synchronous as there are flops on the output. The encrypted or decrypted message data is the result of the AES counter operation XORed with the incoming data, as shown above.

Also, the related dout signal is two cycles behind din and so continues for two cycles after last_w. The e_d signal only needs to be valid when data is being input because decryption only affects the authentication tag calculation.

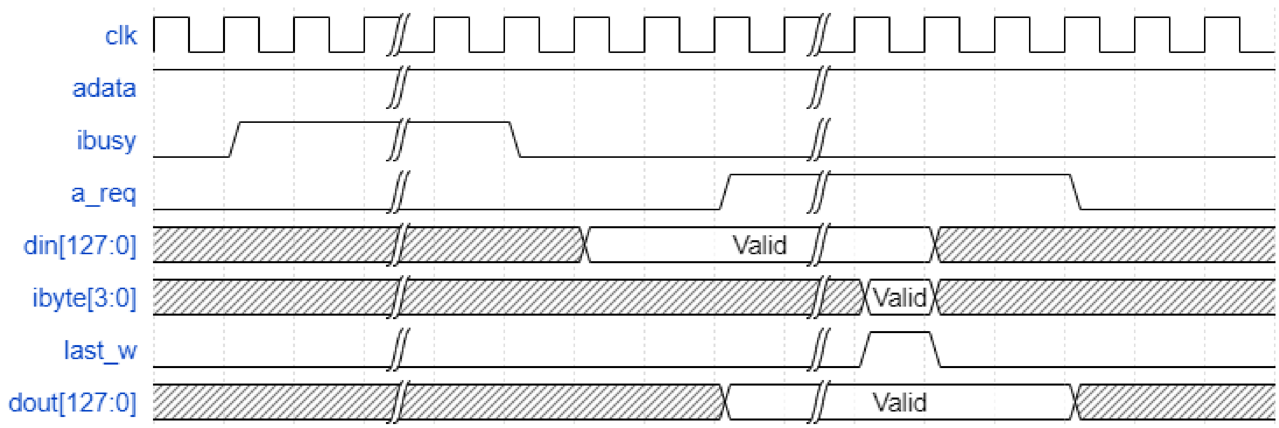

Figure 4: ***Additional Data Input and Output***

Chapter - 3: Usage

The Cryptographic engine, **ACX_AESX_GMC_K**, must be present in any design for the Speedster AC7t1550 device. The ACX_AESX_GMC_K module must be instantiated in a user design for this device. The core may be instantiated as detailed below, or a bypass version which bypasses the ACX_AESX_GMC_K core for applications that do not require data encryption and decryption may, instead, be instantiated.

There can be only one instance of the ACX_AESX_GMC_K module in the user design.

Synthesis

To instantiate either the core or the bypass version, the synthesis must include the Speedster AC7t1550 device synthesis library file. The core is then included as part of the Speedster AC7t1550 synthesis library.

```
# Configure path to ACE library files
set ACE_INSTALL_DIR $::env(ACE_INSTALL_DIR)
# Include AC7t1550 synthesis library file, which includes the ACX_AESX_GMC_K core
add_file -verilog "$ACE_INSTALL_DIR/libraries/device_models/AC7t1550_synplify.v"
```

Simulation

The ACE simulation model also supports the ACX_AESX_GCM_K core. To simulate the core, ensure that the Speedster AC7t1550 simulation device file is included in the simulation file list. The core is included as part of the Speedster AC7t1550 simulation device library.

```
# Include AC7t1550 device simulation library, which includes ACX_AESX_GCM_K core
$ACE_INSTALL_DIR/libraries/device_models/AC7t1550_simmodels.v
```

Chapter - 4: Templates

Verilog Functional Core

```
// Instantiate the Cryptographic Core
ACX_AESX_GCM_K i_ACX_AESX_GCM_K (
    .clk        (user_clk),
    .rstn       (user_rstn),
    .en         (user_en),
    .go         (user_go),
    .abort      (user_abort),
    .ksize      (user_ksize),
    .k192       (user_k192),
    .kin        (user_kin),
    .iv         (user_iv),
    .e_d        (user_e_d),
    .adata      (user_adata),
    .mdata      (user_mdata),
    .k_req      (user_k_req),
    .a_req      (user_a_req),
    .m_req      (user_m_req),
    .din        (user_din),
    .ibyte      (user_ibyte),
    .last_w     (user_last_w),
    .dout       (user_dout),
    .tag        (user_tag),
    .tag_vld    (user_tag_vld),
    .ibusy      (user_ibusy)
);
```

Verilog Core Bypass

If the Cryptographic engine in the Speedster AC7t1550 device is not required, the engine can be bypassed as shown below:

```
    // ----------------------------------------------------------------------
    // Support for the AC7t1550 device
    // ----------------------------------------------------------------------
    // If this design is intended to be targeted to the ac7t1550 device,
    // (which includes the pseudo-hard IP Cryptographic engine), then it is necessary to
    // instantiate the core in the code, even if unused
    // If not required for an AC7t1550 design, then instantiate a bypass
    // instance of the core as shown below
    // For a design that demonstrates full use of the core, please see the
    // Speedster_2D_noc_ref_design_RD22/ac7t1550 design
    // ----------------------------------------------------------------------
    // The define ACX_DEVICE is set as follows :
    //      In simulation by the /sim/<simulator>/Makefile
    //      In GUI synthesis by the /src/syn/<project>.prj file
    //      In batch synthesis by the /src/constraints/synplify_options.tcl file
    // ----------------------------------------------------------------------
`ifdef ACX_DEVICE_AC7t1550
    ACX_AESX_GCM_K_BYPASS ();
`endif
```

Chapter - 5: Implementation

Support for the Cryptographic engine has been integrated into ACE via partition flow. The flow automatically takes care of importing the Cryptographic engine as a partition along with the relevant .EPDB file.

> **Note**
>
> Because the Cryptographic engine is imported as a partition, incremental flow cannot be used on the Speedster AC7t1550 device.

The Cryptographic engine is placed in the South East corner of the device as shown below (see page 15) (highlighted in Blue). The placement is fixed and cannot be modified by the user.

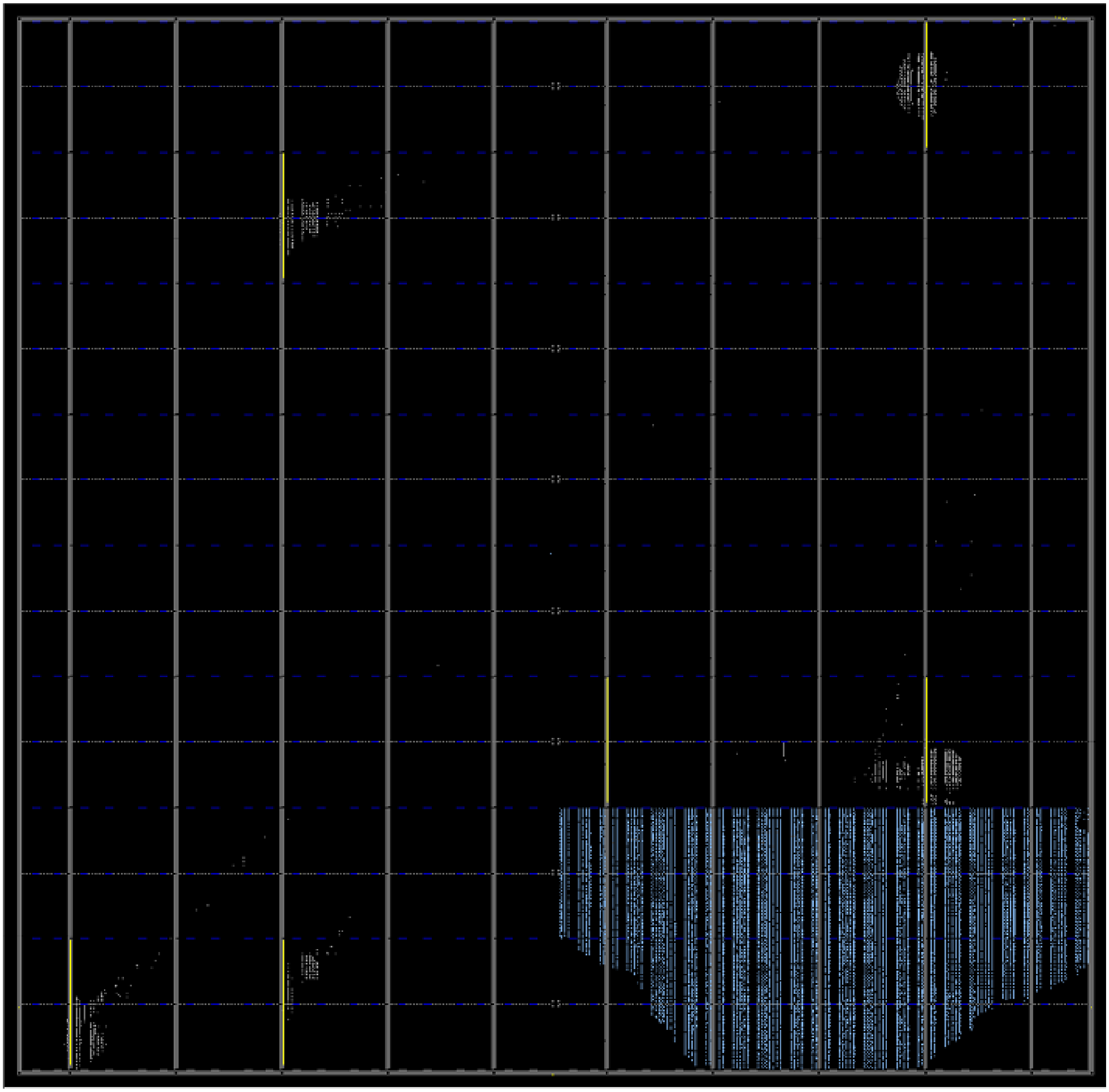

Figure 5: ***Cryptographic Engine Placement***

Chapter - 6: Bitstream Generation

The bitstream for the Speedster AC7t1550 must be encrypted. Use the default Achronix AES key that is included in the Speedster AC7t1550 device overlay package at key index 0. Alternately, users can substitute their own AES keys by burning the E-Fuses for the upper three AES keys into the device.

Default Key

```
bcdd1d62ad64c599807cfc1e1e35baa573fb51192fcfd2c89623051dc3dc521a
```

The AES key must be saved in a text file to be read by ACE during the bitstream generation phase. In the example reference design for the Cryptographic engine, the key is stored in `/src/mem_init_files/aes_key.txt`

ACE Options

The AES Key is input to ACE for encryption of the bitstream via implementation options as shown below:

```
set_impl_option bitstream_encrypted 1
set_impl_option bitstream_encryption_aes_key_file “<path_to_aes.txt> "
set_impl_option bitstream_encryption_key_index 2
set_impl_option bitstream_encryption_key_type 1
set_impl_option bitstream_encryption_same_key 1
```

In the example reference design for the Cryptographic engine, the above options are set in the `/src/constraints/ace_options.tcl` file.

Revision History

Version	Date	Description
1.0	17 Sep 2021	Initial release.