Using the Speedster7t AC7t1550 FPGA (AN028)

Achronix Data Acceleration

April 05, 2023 **Application Note**

Introduction

The Achronix Speedster®7t AC7t1550 is a high-performance FPGA specifically designed to support extremely high bandwidth requirements for demanding applications including data-center workloads and networking infrastructure. Additionally, the Speedster7t AC7t1550 FPGA provides a bi-directional AES-GCM cryptographic engine for encryption/decryption of data within the design. For more details about all features available in the Speedster7t AC7t1550 FPGA, refer to the *Speedster7t FPGA Datasheet* (DS015). This application note explains key details specific to the Speedtser7t AC7t1550 FPGA that must be considered. Specifically, this application note covers the requirements of including the AES-GCM cryptographic engine in the FPGA design, as well as the requirements for an encrypted bitstream.

Instantiating the AES Cryptographic Engine

A black-box representation of the Speedster7t FPGA cryptographic engine must be instantiated within the user design when using the Speedster7t AC7t1550 FPGA; however, it is not necessary to use this capability. Only one instantiation of the cryptographic engine is possible and can only be used with the AC7t1550 — this particular engine in the fabric is not supported in any of the other Speedster7t FPGA family members. 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 cryptographic engine, ACX_AESX_GMC_K, must be instantiated in any design for the Speedster7t AC7t1550 FPGA. The core may be instantiated such that it uses the encryption/decryption functionality, or it may be instantiated such that it bypasses the ACX_AESX_GMC_K core for applications that do not require data encryption and decryption.

Configuring the AES Cryptographic Engine Soft IP

To configure and generate the wrapper file for the soft IP of cryptographic engine:

1. Change to the IP Configuration Perspective in ACE.
2. Under the IP Libraries window, expand the list under **Core** → **Encryption**.

3. Double-click **AES 25G Cryptographic Engine** to open up the configurator:

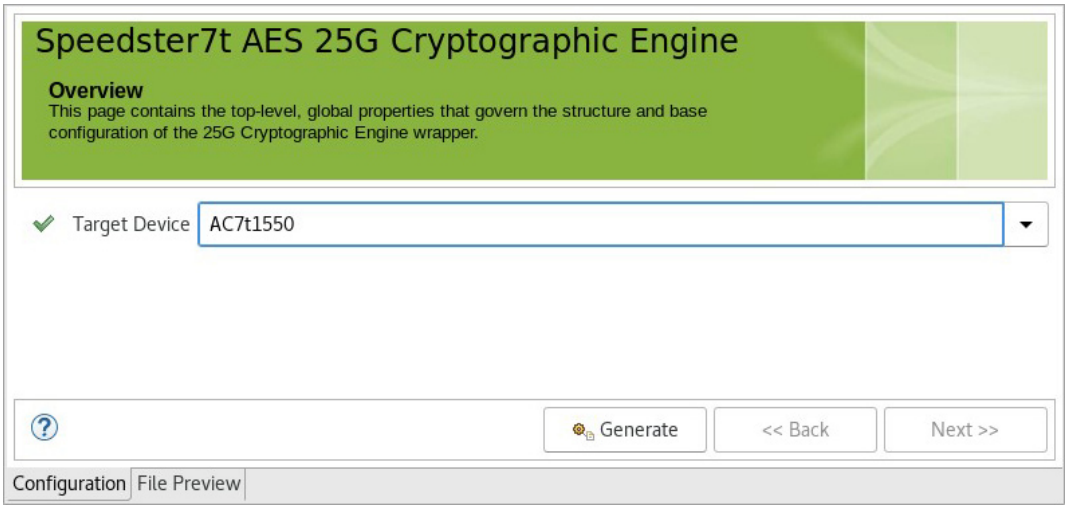

Figure 1: ***AES 25G Cryptographic Engine Configuration***

4. Set the target device to **AC7t1550** and save the configuration
5. Click **Generate** to trigger ACE to generate the RTL wrapper file that must be instantiated in the top-level RTL design file.
   The following is a block diagram of the generated soft IP:

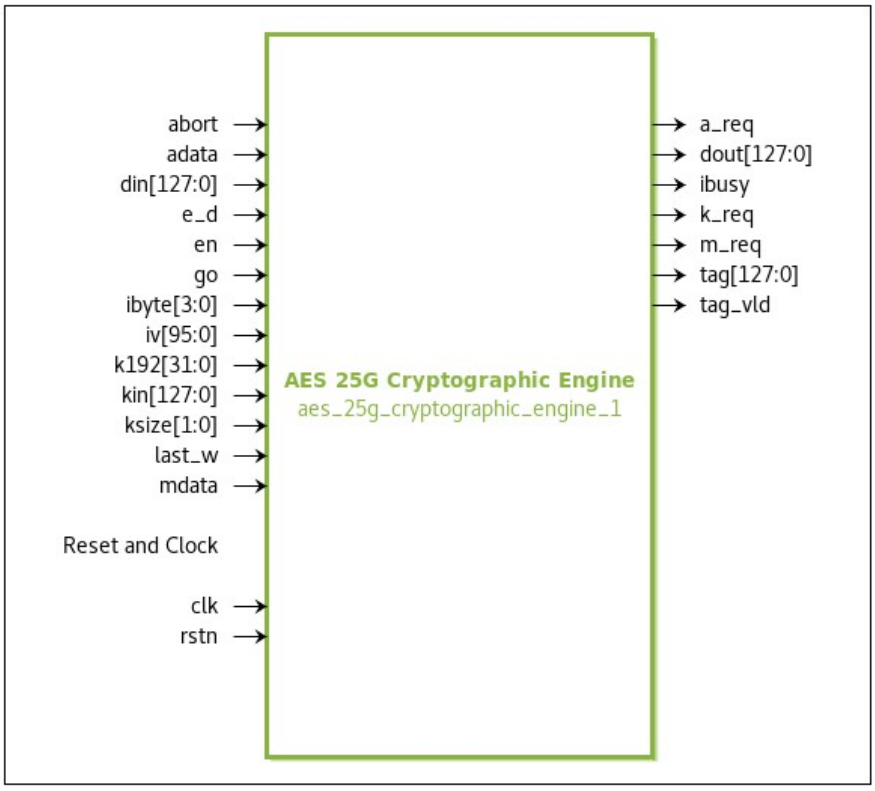

Figure 2: ***AES 25G Cryptographic Engine Soft IP Diagram***

Additionally, there are some practical aspects to be aware of. Because the Speedster7t AC7t1550 FPGA requires the use of encrypted bitstreams, if it is preferred to use JTAG TCL commands to read or write control and status registers (CSRs) in the interface subsystems, or to send transactions on the 2D NoC, the RTL design is required to include the Achronix device manager (ADM). The ADM allows communication with the FPGA via the JTAG TCL commands when using encrypted bitstreams. For more details on including the ADM in a user design, refer to the device manager section in the *Speedster7t Soft IP User Guide* (UG103).

Using the AES Cryptographic Engine

The AES cryptographic engine must be instantiated in the FPGA fabric, and the design can use the functionality of the encryption/decryption, or the design can simply put the soft IP in bypass mode. For details on how to connect a design to the AES cryptographic engine soft IP, refer to the *Speedster7t Cryptographic Engine User Guide* (UG104). This user guide contains descriptions of the input and output signals of the soft IP, how the core works including timing diagrams, information on simulating with the soft IP, and the flow used for implementation.

While the cryptographic engine is required to be instantiated in the RTL, it is not required to be used. If the design does not need any encryption/decryption on the data path, simply put the cryptographic engine in bypass mode. To do this, simply instantiate the bypass version of the cryptographic engine. An example follows.

```
AES Cryptographic Engine in bypass mode

ACX_AESX_GCM_K_BYPASS ();
```

Implementing Designs in the Speedster7t AC7t1550 FPGA

After instantiating the generated soft IP of the AES cryptographic engine, or the bypass version, ACX_AESX_GCM_K_BYPASS, in the RTL, the next step is to synthesize the design using Synplify Pro. The cryptographic engine has been integrated into ACE via the partition flow. This flow automatically takes care of importing the cryptographic engine as a partition along with the relevant `.epdb` file. The placement of the cryptographic engine soft IP is fixed and cannot be modified by the user.

Resources Available When Using the Speedster7t AC7t1550 FPGA

Because the Speedster7t AC7t1550 FPGA requires a single instance of the AES cryptographic engine in the FPGA fabric, there are resources in the FPGA that are consumed by the soft IP. These resources cannot be used by the rest of the design in the FPGA. The AES cryptographic engine is placed in the south-east corner of the FPGA fabric. This region is blocked off for the cryptographic core engine, and no user design can be placed in that region. This area consumes the following resources that are not available for the rest of the FPGA design:

- 86,400 LUTs
- 172,800 flip-flops
- 21,600 ALUs
- 320 BRAMs, LRAMs, and MLPs
- 10 NAPs:
  - NAP[6,1], NAP[6,2]
  - NAP[7,1], NAP[7,2]
  - NAP[8,1], NAP[8,2]
  - NAP[9,1], NAP[9,2]
  - NAP[10,1], NAP[10,2]

> **Note**
>
> Other NAPs on rows 1 and 2 of the 2D NoC can be used (columns 1-5), and transactions can traverse across those rows.

For details on the resources available to the Speedster7t AC7t1550 FPGA, refer to the *Speedster7t FPGA Datasheet* (DS015).

Generating Encrypted Bitstreams for the Speedster7t AC7t1550 FPGA

The Speedster7t AC7t1550 FPGA is required in order to generate and use encrypted bitstreams. Initially, the default Achronix AES key included in the Speedster7t AC7t1550 FPGA overlay package at key index 0 may be used. This can be a good starting point to test out encrypted bitstreams using a known key. Ultimately, custom AES keys may be substituted by burning the eFuses for the upper three AES keys into the FPGA. Generating encrypted bitstreams can be accomplished with a few simple settings, either set in the ACE GUI or set as implementation options on the TCL command line:

1. Enable encrypted bitstreams.
2. Set the filename path for the 256-bit AES encryption key (this can be a full path or a relative path to the current ACE project).
3. Select the decryption key type: black key (0) or red key (1).
4. Select the eFuse key index 0, 1, 2, or 3, where 0 is the default Achronix key.
5. Enable the "enforce same key" option if programming multiple encrypted bitstreams.

For example, the following commands can be used to set options for encrypted bitstreams:

```
set_impl_option -project <ace project name> -impl impl_1 bitstream_encrypted "1"
set_impl_option -project <ace project name> -impl impl_1 bitstream_encryption_aes_key_file
"key_files/aes.txt"
set_impl_option -project <ace project name> -impl impl_1 bitstream_encryption_key_type "1"
set_impl_option -project <ace project name> -impl impl_1 bitstream_encryption_key_index "0"
set_impl_option -project <ace project name> -impl impl_1 bitstream_encryption_same_key "0"
```

Programming and Debug with the Speedster7t AC7t1550 FPGA

Because the Speedster7t AC7t1550 FPGA requires encrypted bitstreams, there are important steps to be aware of when programming and debugging designs on the device. If it is desired to use custom encryption keys, the encryption keys must be programmed into the eFuses.

Programming the Encryption Keys Into the Speedster7t AC7t1550 FPGA eFuses

The following eFuse programming steps are a one-time process, as an eFuse can only be programmed once. These steps are only necessary if it is desired to write a new key value into an eFuse that was not previously programmed with encryption keys.

1. Apply power to the Speedster7t FPGA from a powered-off state or initiate a FCU reset by asserting the FCU_CONFIG_RSTN pin. Refer to the Speedster7t AC7t1500 FPGA Pin Table for the specific ball number.
2. Establish a JTAG connection. For detailed instructions on how to establish a JTAG connection, please refer to the "Configuring External Connections to Hardware" chapter in the "Tasks" section of the *ACE User Guide* (UG070) and to issue commands, refer to *Runtime Programming of Speedster FPGAs* (AN025).
3. Issue the following commands in the ACE Tcl console with the AES key value.

   ```
   jtag::write_aes_encryption_key_efuse $jtag_id <E-Fuse Key Index> <256-bit AES Encryption Key>
   ```

   The eFuse key index and 256-bit AES encryption key values are set in the ACE options, and an example can be seen in the section Generating Encrypted Bitstreams (see page 5) above.
4. Reset the FPGA by cycling the power to the device.

Programming a Speedster7t AC7t1550 FPGA Encrypted Bitstream

After writing the eFuses, proceed to program the encrypted bitstream:

1. Establish a JTAG connection. For detailed instructions on how to establish a JTAG connection, please refer to the "Configuring External Connections to Hardware" chapter in the "Tasks" section of the *ACE User Guide* (UG070) and to issue commands, refer to *Runtime Programming of Speedster FPGAs* (AN025).

2. Program the encrypted bitsteam.

> **Note**
> In the following command, the `-encrypted` switch is not needed as it is built into the Speedster7t AC7t1550 FPGA library.

```
ac7t1550::program_hex_file <path to bitstream>
```

3. Verify the encrypted bitstream has been configured. To verify this, check that the FCU_CONFIG_USER_MODE ball is high indicating that the device has transitioned into user mode.

> **Note**
> FCU_CONFIG_USER_MODE only transitions from 0 to 1 when the programmed encrypted bitstream is a full bitstream and not a stage0 or partial reconfiguration.

Refer to the Speedster7t AC7t1500 FPGA Pin Table for the specific ball number.

Debugging with the Speedster7t AC7t1550 FPGA

As mentioned above, if it is preferred to use JTAG TCL commands to read or write control and status registers (CSRs) in the interface subsystems, or to send transactions on the 2D NoC, the RTL design is required to include the Achronix device manager (ADM). The ADM allows communication with the FPGA via the JTAG TCL commands when using encrypted bitstreams. For more details on including the ADM in a user design, refer to the device manager section in the *Speedster7t Soft IP User Guide* (UG103).

If using Snapshot to debug a design on the Speedster7t AC7t1550 FPGA, there are no limitations due to encrypted bitstreams. When the design does not use the Achronix device manager (ADM), `ACX_SNAPSHOT` can be instantiated and used to help debug. If the ADM is used in the design, `ACX_SNAPSHOT_UNIT` is needed for connecting to Snapshot. For further details on how to use Snapshot, refer to the *Snapshot User Guide* (UG016).

Revision History

Version	Date	Description
1.0	05 Apr 2023	• Initial Achronix release

Achronix Semiconductor Corporation

2903 Bunker Hill Lane
Santa Clara, CA 95054
USA

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

Notice of Disclaimer