AVR1318: Using the XMEGA built-in AES accelerator

Features

- **Full compliance with AES (FIPS Publication 197, 2002)**
  - **Both encryption and decryption procedures**
- **128-bit Key and State memory**
- **XOR load option to State memory useful for cipher block coding**
- **Sequential access to State and Key memories**
- **Optional Interrupt- and DMA request on AES complete**

1 Introduction

The XMEGA™ AES Crypto Module supports the Advanced Encryption Standard (AES), and can perform encryption and decryption. The module supports a key length of 128 bits. The 128-bit key block and 128-bit data block (plaintext or ciphertext) must be loaded into the Key and State memory in the AES Crypto Module. The AES uses 375 clock cycles to execute one encryption/decryption after the Key and State memory is loaded and the mode of operation is selected.

This application note describes the basic functionality of the XMEGA AES with code examples to get up and running quickly. A driver interface written in C is included as well.

Advanced usage, such as Direct Memory Access and the XMEGA Event System, is outside the scope of this application note. Please refer to the device datasheets and other relevant application notes for details.

8-bit AVR® Microcontrollers

Application Note

Rev. 8106A-AVR-04/08

2 Theory

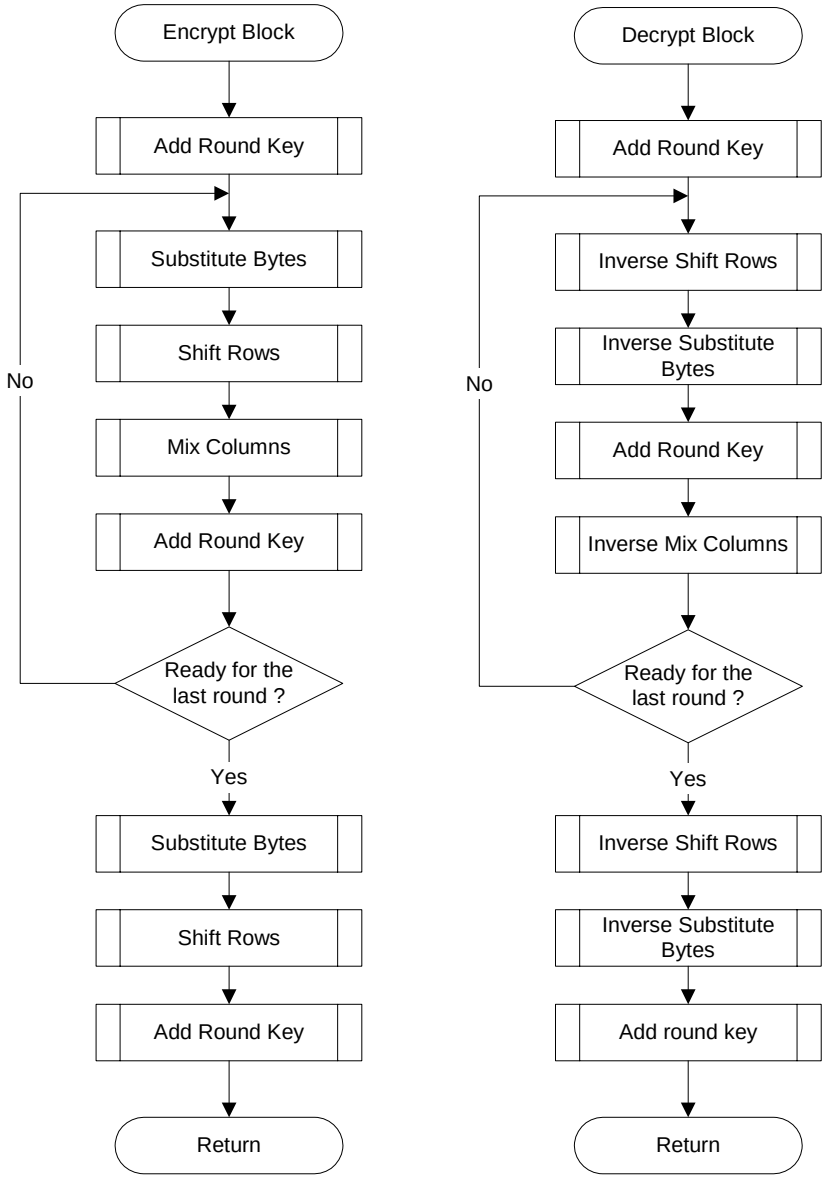
Cryptography is the art or science of keeping information secret and is based on either hiding the cryptographic method or securing the cryptographic key. Algorithms only based on the secrecy of the method used are mainly of historical interest and do not meet the needs of the real world. Modern algorithms use a key to control encryption and decryption. Without the matching key, the scrambled message or data cannot be arranged into plaintext.

Algorithms based on cryptographic keys are divided in two classes; symmetric and asymmetric. Symmetric algorithms use the same key for encryption and decryption while asymmetric algorithms use different keys. AES is a symmetric key algorithm.

2.1 Advanced Encryption Standard – AES

This section is not intended to be a detailed description of the AES algorithm, but a brief overview. For more details the reader should study the AES specification. The AES algorithm uses functions that are based on finite field arithmetic. The AES algorithm has a fixed block size of 128 bits, while the length of the key can be 128, 192 or 256 bits depending on the desired security. The flow of the AES algorithm is illustrated in Figure 2-1. See AES specification for explanation of the different operations in the flowchart.

Figure 2-1 Encryption and Decryption flowcharts



The number of rounds the algorithm needs to run is depended on the key length.

2.2 Cipher Block Chaining (CBC)

AES is a block cipher, meaning that the algorithm operates on fixed size blocks of data. For known input block and a constant encryption key, the output will always be the same. This information may provide useful for somebody wanting to attack the cipher system.

There are methods commonly used, which cause identical plaintext blocks being encrypted to different ciphertext blocks. One such methode called Cipher Block Chaining (CBC).

CBC is a method of connecting the cipher blocks such that leading blocks influence all trailing blocks. This is achieved by first performing an XOR operation on the

plaintext block and the previous ciphertext block before encrypting the result. This increases the number of plaintext bits one ciphertext bit depends on.

Figure 2-2. CBC Encryption

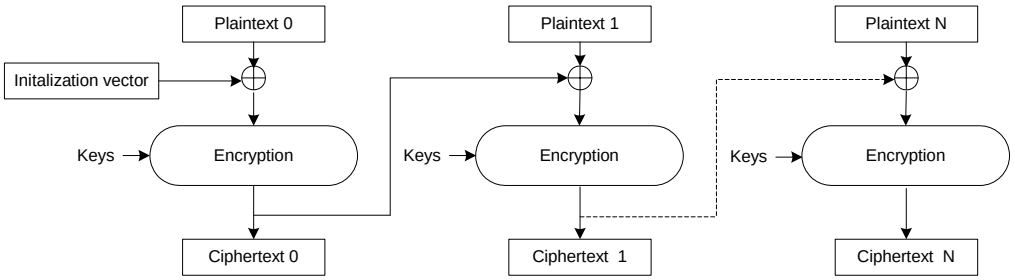
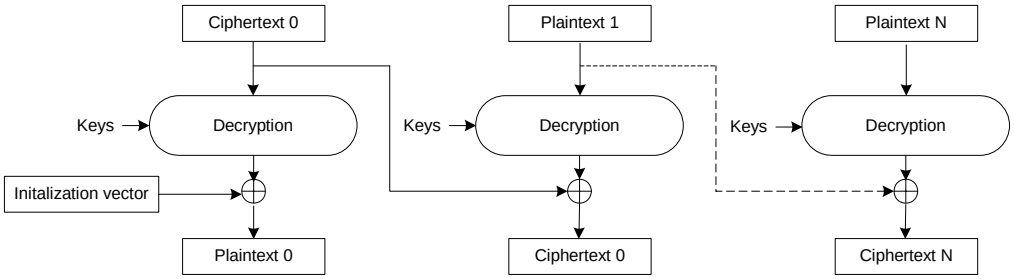


Figure 2-3. CBC Decryption



3 AES Crypto Module

The following subsections contain an introduction on how to operate the AES Crypto Module. The different feature the AES Crypto Module supports is also presented.

The XMEGA AES Crypto Module supports an AES key length of 128-bits. The service of the AES Crypto Module can be executed through an interrupt mechanism or polling.

The DMA can also be setup to handle the AES crypto module, but this is outside the scope of this application note. For more information, please refer to the device datasheet or the application note “AVR1304: Using the XMEGA DMA Controller”.

3.1 Key and State memory

The AES Crypto Module contains two 128 bits memories needed to keep the AES plaintext/ciphertext and the Key. Note that in the AES Crypto Module the following definition of the Key is used:

- In encryption mode, the Key is the one defined in the AES standard.
- In decryption mode, the Key is the last subkey of the Expanded Key defined in the AES standard.

The State and Key memory can be written / read sequentially byte by byte through the AES State Register (STATE) or AES Key Register (KEY). Both the State and Key memory have two 4-bit address pointers that address the memory for read and write access. The appropriate pointer is automatically incremented after an access to the AES memory. It is only possible to access the Key and State memories while the AES start bit (START) in the AES Control register (CTRL) is zero.

NOTE: Both the Key and State memory must be completely loaded before an encryption/decryption can start, if not the AES Error flag (ERROR) in the status register (STATUS) is set.

3.2 Encryption

To execute an AES encryption using the AES Crypto Module the following should be done.

- Enable/disable AES interrupts, by setting/clearing the Interrupt priority and enable bits (INTLVL) in the Interrupt Control register (INTCTRL).
- Select the AES encryption direction, by clearing the decrypt bit (decrypt) in the control register (CTRL).
- Load the AES key into the AES Key memory
- Load the data block into the AES State memory
- Start encryption, by setting the start bit (START) in the control register (CTRL).

When the encryption is completed, the AES State Ready Interrupt Flag (SRIF) in the AES Status register (STATUS) is set. If the interrupt mechanism is used an interrupt is generated. The AES State memory will after an encryption is completed contain the generated ciphertext while the AES Key memory will contain the last subkey of the Expanded Key defined in the AES standard.

3.3 Decryption

To execute an AES decryption using the AES Crypto Module the following should be done.

- Enable/disable AES interrupts, by setting/clearing the Interrupt priority and enable bits (INTLVL) in the Interrupt Control register (INTCTRL).
- Select the AES decryption direction, by setting the decrypt bit (decrypt) in the control register (CTRL).
- Load the last subkey of the Expanded Key defined in the AES standard into the AES Key memory.
- Load the data block into the AES State memory
- Start decryption, by setting the start bit (START) in the control register (CTRL).

When the decryption is completed, the AES State Ready Interrupt Flag (SRIF) in the AES Status register (STATUS) is set. If the interrupt mechanism is used an interrupt is generated. The AES State memory will after a decryption is completed contain the generated plaintext while the AES Key memory will contain the original Key defined in the AES standard.

NOTE: The last subkey of the Expanded Key that is needed to do a decryption can be generated in two optional ways. It can be generated by a Key Expansion procedure executed in software or by the AES Crypto module. The AES Crypto Module can generate the last subkey of the Expanded Key by processing a dummy data block in encryption mode, using the original Key. After the end of the encryption, the Key memory contains the last subkey.

3.4 AES State XOR load

When the AES State XOR load feature is enabled the loaded value to the AES State memory is bitwise XORed with the current value of the AES Stat memory. To enable AES State XOR feature set the XOR bit (XOR) in the AES Control register (CTRL). This feature is very useful when CBC or other Cipher Modes shall be executed.

3.5 AES Auto Start Trigger

When the AES Auto Start Trigger feature is enabled the encryption/decryption will automatically start when the State register is fully loaded.

4 Driver Implementation

This application note includes a source code package with a basic AES driver implemented in C. It is written for the IAR Embedded Workbench® compiler.

The AES driver supports encryption and decryptions of single block data and CBC. Both interrupted and a polled version of the driver is supported.

Note that this AES driver is not intended for use with high-performance code. It is designed as a library to get started using the AES accelerator. Please refer to the driver source code and device datasheet for more details.

4.1 Files

The source code package consists of three files:

- *AES_driver.c* – AES accelerator driver source file.
- *AES_driver.h* – DES accelerator driver header file.
- *AES_example_polled.c* – Example code using the polled driver.
- *AES_example_interrupt*.c – Example code using the interrupt driver.

For a complete overview of the available driver interface functions and their use, please refer to the source code documentation.

4.2 Doxygen Documentation

All source code is prepared for automatic documentation generation using Doxygen. Doxygen is a tool for generating documentation from source code by analyzing the source code and using special keywords. For more details about Doxygen please visit http://www.doxygen.org. Precompiled Doxygen documentation is also supplied with the source code accompanying this application note, available from the *readme.html* file in the source code folder.

Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site
www.atmel.com

Technical Support
avr@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Request
www.atmel.com/literature