AVR1507: Xplain training - XMEGA Crypto Engines

8-bit AVR® Microcontrollers

Application Note

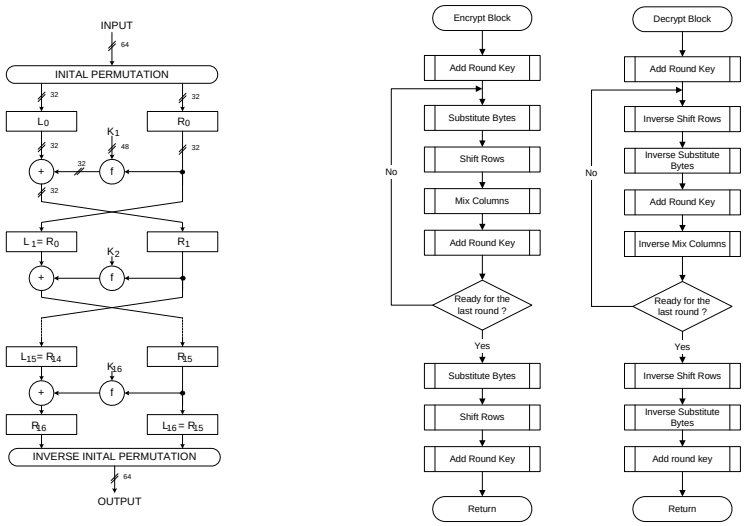
Prerequisites

- **Required knowledge**
  - **Use of Atmel® AVR® Studio®**
  - **Atmel XMEGA™ Basics C training**
- **Required Software**
  - **Atmel AVR Studio latest version**
  - **WinAVR latest version**
  - **XMEGA-Crypto training files**
- **Required Hardware**
  - **Xplain Evaluation board**
  - **JTAGICE mkII w/latest firmware**
- **Estimated completion time**
  - **1.5 hours**

1 Introduction

This training will focus on practical implementation of the mentioned encryption algorithms, and explain how to easily implement them using Atmel XMEGA's powerful hardware support. Details of the mathematics behind the DES and AES algorithms are beyond the scope of this training and will not be covered.

Figure 1-1. DES and AES Flowcharts



Rev. 8316A-AVR-06/10

2 Crypto Overview

This section provides an overview of the basic Crypto configuration options and functionality in the Atmel XMEGA.

2.1 DES – Data Encryption Standard

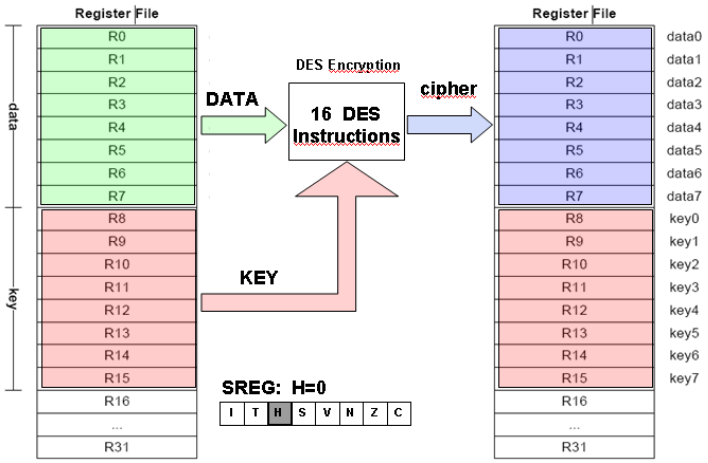
The XMEGA has an additional instruction added to the instruction set; the DES instruction. This instruction uses key data stored in General Purpose register R8 - R16 as a key to encrypt or decrypt the data stored in register R0 - R7. Executing one DES instruction performs one round in the DES algorithm. The DES algorithm uses 16 stages, thus 16 DES instructions must be executed in increasing order to form the correct DES ciphertext or plaintext. Valid arguments to the DES instruction are numbers in the range 0x00 to 0x0F, but will only output correct result if used in the correct sequence, that is: 0x00, 0x01, …. , 0x0E, 0x0F.

2.1.1 DES Encryption

Encrypting data using the DES algorithm is very simple on the XMEGA, and does not require the user to know anything about the mathematics and operation of the encryption method.

The practical implementation relies on 4 steps only:

Figure 2-1. Registers used for encrypting



1. Load plaintext data into registers R0-R7
2. Load key data into registers R8-R15
3. Clear “H” Flag in SREG
4. Execute 16 DES instructions DES 0x00-DES 0x0F

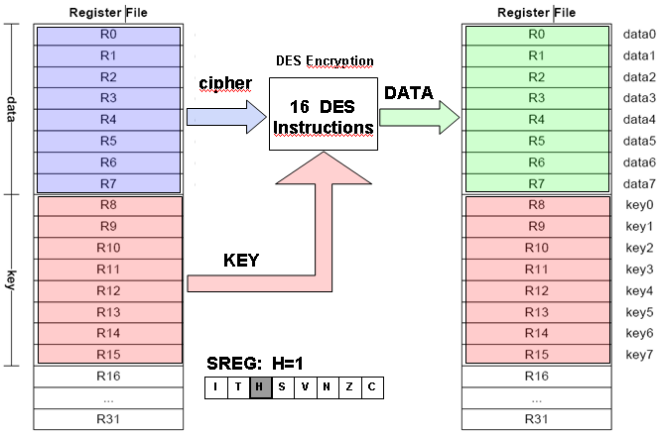
The ciphertext will now be available in registers R0..R7. While the DES instruction is not directly supported by compilers at this time, an assembly library that performs the correct DES encryption and decryption is supplied with this application note and is ready for use in application software written in C or assembly.

2.1.2 DES Decryption

Decrypting data width the DES algorithm is just as simple as encrypting. The same steps apply, only this time the “H” flag in SREG must be set:

1. Load ciphertext data into registers R0-R7
2. Load key data into registers R8-R15
3. Set “H” Flag in SREG
4. Execute 16 DES instructions DES 0x00-DES 0x0F

Figure 2-2. Registers used for decryption



The plaintext data is now available in registers R0 - R7.

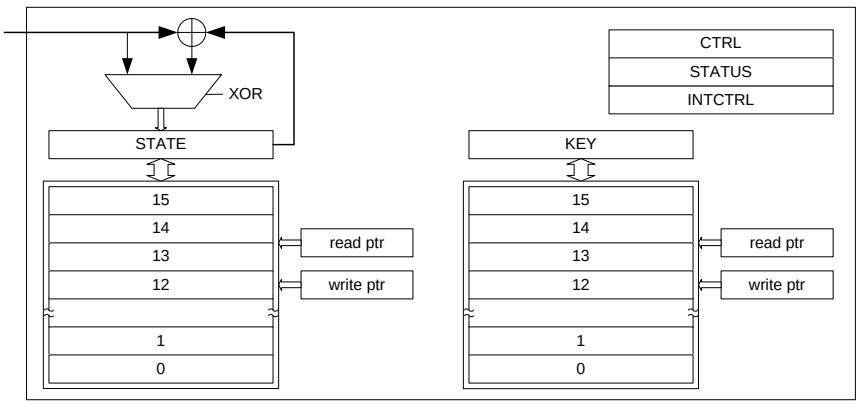
2.2 AES – Advanced Encryption Standard

Unlike the DES instruction, The AES is a hardware module with data registers, control and status registers. It can also generate interrupts when a conversion is completed.

The module contains:

1. Control Register CTRL, used for resetting the module, starting a conversion, select encrypting/decrypting and whether XOR mode is to be used when loading data to the state memory.
2. Status Register STATUS, contains the flags; ERROR and SRIF (short for State Ready Interrupt Flag). ERROR is set if the STATE or KEY registers while the START bit is set. It is also set if a conversion is started when the memories are not fully read or written. SRIF is set when a conversion is complete
3. Key Memory. 16 bytes of memory accessed through the KEY register
4. State Memory. 16 bytes of memory accessed through the STATE register
5. Interrupt Control Register (INTCTRL) used to enable and set interrupt level

Figure 2-3. AES Module



More information about the AES module and the registers involved can be found in the XMEGA manual.

3 Overview

This hands-on session covers some of the basic Atmel XMEGA Crypto features:

Task 1: DES encryption and decryption

This task shows how to use the DES application note library to encrypt and decrypt a simple message.

Task 2: 3DES encryption and decryption

This task shows how to use Triple DES. We will use the 3DES library function, but also make our own 3DES using three normal DES calls.

Task 3: AES encryption and decryption

This task shows how easy it is to use the AES functions to control the AES module, and encrypt and decrypt a message. We will also look at the number of cycles it takes to encrypt and decrypt a message using AES.

Task 4: AES and CBC

This task will add CBC to the AES encryption. In this task you need to look up the functions in the AES library file to find out the correct syntax. We will also look at the number of cycles it takes to encrypt and decrypt a message using CBC AES, and compare with the previous task.

- This pictogram indicates optional tasks and questions. Answers will be given in the wrap-up session at the end of this training.

4 Task 1: DES encryption and decryption

This task will introduce you to the DES application note and how to use its library functions to quickly implement DES encryption and decryption in applications.

The goal for this task is that you know how to:

- Use the application note library functions for DES encryption and decryption
- See what happens if decrypting with slightly different key than used to encrypt
- Find out which of the 64-bit code is used as parity bits


Todo

1. In Atmel AVR Studio, open the project `DES.aps` and look at the file `task1.c`

In this task we will use three strings to store in-data, out-data and cipher-data. This allows for a simple check to make sure that the decrypted message is identical to in-data.

Note that two DES key strings are defined. This is done to allow a simple method to decrypt the message with a slightly different key than the encryption key to see how this affects the output.

2. Study the code and get familiar with the structure

All the program does is encrypt a message, decrypt it and verify that the input and output is the same.

3. Build and run the code in AVR Studio and verify that in-data and out-data are indeed the same

4. Try changing one of the values in the decryptkey values and see how this affects the decryption

- The DES standard uses only 56 bit as the actual key. 8 bits in the decryptkey is used for parity check. Find out where these parity bits are located within the 64-bit decryptkey

  HINT: *Toggling these bits will not affect the output from the DES decryption algorithm.*

5 Task 2: 3DES Encryption and Decryption

Traditional DES encryption is 56-bit and although it provides good security this is no longer recommended for highly secure applications. Triple Data Encryption Standard (3DES), which is based on using DES three times, increases the key strength from 56 bits to 168 bits. This makes it much more secure, but also slower, as you are in fact executing three operations on each data block.

The goal for this task is that you know how to:

- Use the application note 3DES library functions to encrypt and decrypt a message
- Use the DES library to “manually” make a 3DES implementation

1. Open the project `3DES.aps` and take a look at the `Task2.c`

The code use the functions provided by the DES application note.

2. Try to understand the code in `Task2.c`. Then compile and run it. Use the watch window to keep an eye on cipher-data and out-data after running the code

Since 3DES is based on running DES three times, your task is to implement 3DES using the DES functions from the previous task. 3DES encryption is done by an encryption step followed by a decryption step and then a final encryption step. Each step uses its own key.

3. In the main function you will see the `DEC_Encrypt` and `DEC_Decrypt` function calls. Verify that they are placed in the correct order for decrypting

4. Insert the missing parameters. You may use the code in the previous task as a reference

5. Verify that the data decoded by the `DES_Encrypt` and `DES_Decrypt` functions are equal to the initial in-data and also the data decrypted from the `DES_3DES_Encrypt` and `DES_3DES_Decrypt` functions

- Is the number of cycles needed to execute the `DES_3DES_Encrypt` three times more than for the `DES_Encrypt` function?

6 Task 3: AES encryption and decryption

In this task we will look at the AES module and how to use the software library in Application Note AVR1318 to implement AES encrypting and decrypting.

The goal for this task is that you know how to:

- Use the application note AVR1318 AES library functions to encrypt and decrypt a message
- Know which key to use for both encryption and decryption
- Measure how many clock cycles are needed to execute an AES encoding and decoding

1. Open the project `AES.aps` and take a look at the `task3.c`. Look through the code and familiarize yourself with the functions used

2. Compile and run the program

3. Add watches to the variables `indata`, `cipherdata`, `outdata`, `aeskey` and `lastsubkey`

4. Step through the program and look at how the content of the variables change

Since the AES is a hardware module, the AES software can easily be written in C without any inline assembly or calls to assembly functions. You may have noticed that single stepping into the functions in the DES code in the previous tasks was impossible. The reason for this is that these are written in assembly, and Atmel AVR Studio / GCC do not allow single stepping into such assembly functions.

5. In the Processor view you will see that there is a Cycle Counter which seems not to be working. This because the JTAGICE mkII does not support this feature. We do, however, want to measure the number of cycles it takes to do an AES encryption and decryption. To do this we will temporarily change debug platform. In the “Debug” menu, select “Platform and device”, and choose to use “Simulator 2” instead of “JTAGICE mkII” for debugging. Simulator 2 is a cycle accurate simulator.

6. Measure how many cycles the encryption and decryption takes. Note that it is possible to reset the cycle counter by right clicking it

- What is the purpose of the `AES_lastsubkey_generate()` function?
- Why is Cycle Counting not possible on JTAGICE mkII (or Atmel AVR Dragon)?

7 Task 4: Using AES in CBC mode

Abstract:

In the cipher-block chaining (CBC) mode, each block of plaintext is XORed with the previous ciphertext block before being encrypted. This way, each ciphertext block is dependent on all plaintext blocks processed up to that point. Also, to make each message unique, an initialization vector must be used in the first block.

In this task we will look at how CBC (Cipher Block Chaining) can be implemented.

The goal for this task is that you know how to:

- Look up the driver file and find the syntax of its functions
- Encrypt three message blocks using CBC mode
- Decrypt the same three message blocks using CBC

In the previous tasks all code has already been written. This is no longer the case. Most of the code is provided, but you need to put it enter the arguments for the encryption and decryption functions, and make sure the correct arguments are passed to each function call.


Todo

1. Open the project `CBC.aps` and take a look at the `task4.c`. Try to roughly understand the code

2. In the `main()`-function, we have two function calls that are not complete. Complete with code for the encryption and decryption routine. Compile and run the code

3. Change to Simulator and measure the number of cycles the encryption and decryption function uses

- Can you explain the difference in cycles used for encoding and decoding?

8 Summary

In this hands-on training we have seen how easy it is to implement DES and AES encryption in your applications. No knowledge of the algorithm is needed to be able to take full advantage of these features!

9 Resources

- Atmel XMEGA Manual and Datasheets.
    - http://www.atmel.com/xmega
- Atmel AVR Studio with help files
    - http://www.atmel.com/products/AVR/
- WINAVR GCC compiler
    - http://winavr.sourceforge.net/
- Atmel IAR Embedded Workbench® compiler
    - http://www.iar.com/

10 Atmel Technical Support Center

Atmel has several support channels available:

• Web portal:	http://support.atmel.no/	All Atmel microcontrollers
• Email:	avr@atmel.com	All Atmel AVR products
• Email:	avr32@atmel.com	All 32-bit AVR products

Please register on the web portal to gain access to the following services:

- Access to a rich FAQ database
- Easy submission of technical support requests
- History of all your past support requests
- Register to receive Atmel microcontrollers' newsletters
- Get information about available trainings and training material

Headquarters

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

International

Atmel Asia
Unit 1-5 & 16, 19/F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
Hong Kong
Tel: (852) 2245-6100
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

8316A-AVR-06/10