

AES256XTSSTGIP Reference Design

1	Introduction	1
2	Hardware Overview.....	1
2.1	AES256XTSSTGIP	2
2.2	LAXi2Reg	2
2.3	AsyncAXiReg	3
2.4	UserReg	3
2.4.1	Encryption key setting	7
2.4.2	Tweakable key setting.....	8
2.4.3	parameter setting	9
2.4.4	Encryption	11
2.4.5	Decryption	12
3	CPU Firmware	13
3.1	Set encryption key	13
3.2	Set tweakable key.....	14
3.3	Set encryption/decryption IV	14
3.4	Show Data Memory	15
3.5	Fill Plain Data Memory.....	16
3.6	Encrypt	17
3.7	Fill Cipher Data Memory	18
3.8	Decrypt	18
4	Revision History.....	19

AES256XTSSTGIP Reference Design

Rev1.00 25-Aug-2023

1 Introduction

This document describes the details of AES256XTSSTGIP, AES256XTSSTG2XIP and AES256XTSSTG4XIP reference design. This document will focus on AES256XTSSTGIP, which can serve as a reference for the other modules in the series. The only difference is the width of the data. In this reference design, AES256XTSSTGIP are used to encrypt and decrypt data between two memories in FPGA. User can fill memory with plain or cipher data patterns, set encryption key, tweakable key, Initialization Vector (IV) and control test operation via serial console on test PC. More details of the hardware design and CPU firmware are described as follows.

2 Hardware Overview

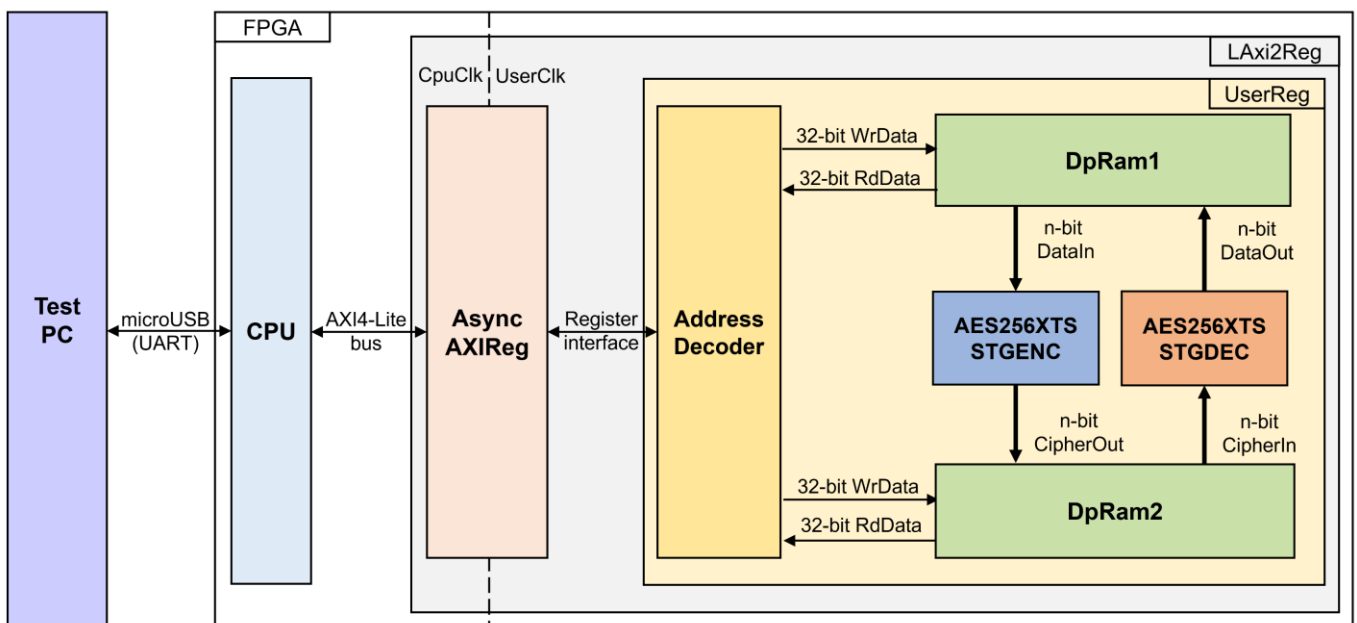


Figure 2-1 AES256XTSSTGIP reference design block diagram

In this test environment, two AES256XTSSTGIP, called AES256XTSSTGENC and AES256XTSSTGDEC, interface with two dual-port RAMs with asymmetric ports, which are DpRam1 and DpRam2, as shown in Figure 2-1. AES256XTSSTGIP and two RAMs are sub-modules in UserReg module within LAXi2Reg. CPU system is designed to interface with FPGA logic through AXI4 Lite bus and interface with user through serial console in test PC.

For user control interface, there are registers in UserReg to store parameters from user such as encryption key, tweakable key, Initialization Vector (IV), number of data to encrypt or decrypt. Input parameters are received from user via serial console.

For user data interface, UserReg is designed to be able to write pattern data or read data in RAMs following user's command. DpRam1 is used to store the plain data from user which will be input data for AES256XTSSTGENC in encryption mode or the decrypted data which is output data from AES256XTSSTGDEC in decryption mode. DpRam2 is used to store the cipher data from user which will be input data for AES256XTSSTGDEC in decryption mode or encrypted data which is output data from AES256XTSSTGENC in encryption mode. The details of AES256XTSSTGIP are described as follows.

2.1 AES256XTSSTGIP

AES256XTSSTGIP(AES256XTS-StorageIP) Series consists of encryption module and decryption module. Each product in the series has a different data width, the data width is represented by n. The detailed of the products within the series is below.

AES256XTSSTGIP:

- Support data width 128-bit alignment. (n=128)
- AES256XTSSTGENC is encrypted module for AES256XTSSTGIP.
- AES256XTSSTGDEC is decrypted module for AES256XTSSTGIP.

AES256XTSSTG2XIP:

- Support data width 256-bit alignment. (n=256)
- AES256XTSSTG2XENC is encrypted module for AES256XTSSTG2XIP.
- AES256XTSSTG2XDEC is decrypted module for AES256XTSSTG2XIP.

AES256XTSSTG4XIP:

- Support data width 512-bit alignment. (n=512)
- AES256XTSSTG4XENC is encrypted module for AES256XTSSTG4XIP.
- AES256XTSSTG4XDEC is decrypted module for AES256XTSSTG4XIP.

Because CPU system and AES256XTSSTGIP run in different clock domain, AsyncAXIReg module inside LAXi2Reg is designed as asynchronous circuit to support clock-crossing operation. Also, AsyncAXIReg converts AXI4-Lite bus signal which is standard bus in CPU system to be register interface. The details of LAXi2Reg module are described as follows.

2.2 LAXi2Reg

LAXi2Reg module is connected to CPU through AXI4-Lite bus. The hardware registers are mapped to CPU memory address, as shown in Table 2-1. The control and status registers for CPU access are designed in LAXi2Reg.

LAXi2Reg consists of AsyncAxiReg and UserReg. AsyncAxiReg is designed to convert the AXI4-Lite signals to be the simple register interface which has 32-bit data bus size (similar to AXI4-Lite data bus size). Additionally, as shown in Figure 2-1, there are two clock domains applied in this block, i.e., CpuClk which is used to interface with CPU through AXI4-Lite bus and UserClk which is user clock domain for AES256XTSSTGENC/AES256XTSSTGDEC. AsyncAxiReg includes asynchronous circuit between CpuClk and UserClk.

UserReg includes the register file of the parameters and the status signals of test logics, including dual-port rams and AES256XTSSTGIP. Both data interface and control interface of AES256XTSSTGIP are connected to UserReg. More details of AsyncAxiReg and UserReg are described as follows.

2.3 AsyncAxiReg

This module is designed to convert the signal interface of AXI4-Lite to be register interface. Also, it enables two clock domains, CpuClk and UserClk domain, to communicate.

To write register, RegWrEn is asserted to '1' with the valid signal of RegAddr (Register address in 32-bit unit), RegWrData (write data of the register), and RegWrByteEn (the byte enable of this access: bit[0] is write enable for RegWrData[7:0], bit[1] is used for RegWrData[15:8], ..., and bit[3] is used for RegWrData[31:24]).

To read register, AsyncAxiReg asserts RegRdReq='1' with the valid value of RegAddr (the register address in 32-bit unit). After that, the module waits until RegRdValid is asserted to '1' to get the read data through RegRdData signal at the same clock.

2.4 UserReg

This module is designed to write/read data in RAMs, control and check status of AES256XTSSTGENC and AES256XTSSTGDEC corresponding with write register access or read register request from AsyncAxiReg module. Memory map inside UserReg module is shown in Table 2-1. Timing diagram of register interface is shown in Figure 2-2.

Table 2-1 Register map Definition

Address offset	Register Name	Rd/Wr	Description
0x0000	ADDR_A1_REG	Rd/Wr	[12:4] – address A of DpRam1 (rAddrA1[12:4]).
0x0004	ADDR_A2_REG	Rd/Wr	[12:4] – address A of DpRam2 (rAddrA2[12:4]).
0x0008	ENC_DATAINCNT_REG	Rd	[2] – AES256XTSSTGENC EKeyIn busy flag (wEncEKeyInBusy). [1] – AES256XTSSTGENC TKeyIn busy flag (wEncTKeyInBusy). [0] – AES256XTSSTGENC busy flag (wEncBusy).
		Wr	[15:0] – length of plain data for encryption (rEncDataInCnt[15:0]). rEnclnitStart is asserted to '1' and encryption process is started.
0x000C	DEC_DATAINCNT_REG	Rd	[2] – AES256XTSSTGDEC EKeyIn busy flag (wDecEKeyInBusy). [1] – AES256XTSSTGDEC TKeyIn busy flag (wDecTKeyInBusy). [0] – AES256XTSSTGDEC busy flag (wDecBusy).
		Wr	[15:0] – length of cipher data for decryption (rDecDataInCnt[15:0]). rDeclnitStart is asserted to '1' and decryption process is started.
0x0010	ENC_IVINC_REG	Wr	[0] – AES256XTSSTGENC IvIncrement (rEnclvIncrement)

0x0014	DEC_IVINC_REG	Wr	[0] – AES256XTSSTGDEC IvIncrement (rDecIvIncrement)
0x0018	ENC_VER_REG	Rd	[31:0] – AES256XTSSTGENC IP version (wEncVersion).
0x001C	DEC_VER_REG	Rd	[31:0] – AES256XTSSTGDEC IP version (wDecVersion).
0x0020	ENC_EKEYIN_0_REG	Rd/Wr	[31:0] – AES256XTSSTGENC Ekey (rEncEKeyIn[31:0]). When rEncEKeyIn[31:0] is set, rEncEKeyInValid is asserted to '1' and encryption key setting is started.
0x0024	ENC_EKEYIN_1_REG	Rd/Wr	[31:0] – AES256XTSSTGENC Ekey (rEncEKeyIn[63:32]).
0x0028	ENC_EKEYIN_2_REG	Rd/Wr	[31:0] – AES256XTSSTGENC Ekey (rEncEKeyIn[95:64]).
0x002C	ENC_EKEYIN_3_REG	Rd/Wr	[31:0] – AES256XTSSTGENC Ekey (rEncEKeyIn[127:96]).
0x0030	ENC_EKEYIN_4_REG	Rd/Wr	[31:0] – AES256XTSSTGENC Ekey (rEncEKeyIn[159:128]).
0x0034	ENC_EKEYIN_5_REG	Rd/Wr	[31:0] – AES256XTSSTGENC Ekey (rEncEKeyIn[191:160]).
0x0038	ENC_EKEYIN_6_REG	Rd/Wr	[31:0] – AES256XTSSTGENC Ekey (rEncEKeyIn[223:192]).
0x003C	ENC_EKEYIN_7_REG	Rd/Wr	[31:0] – AES256XTSSTGENC Ekey (rEncEKeyIn[255:224]).
0x0040	DEC_EKEYIN_0_REG	Rd/Wr	[31:0] – AES256XTSSTGDEC Ekey (rDecEKeyIn[31:0]). When rDecEKeyIn [31:0] is set, rDecEKeyInValid is asserted to '1' and encryption key setting is started.
0x0044	DEC_EKEYIN_1_REG	Rd/Wr	[31:0] – AES256XTSSTGDEC Ekey (rDecEKeyIn[63:32]).
0x0048	DEC_EKEYIN_2_REG	Rd/Wr	[31:0] – AES256XTSSTGDEC Ekey (rDecEKeyIn[95:64]).
0x004C	DEC_EKEYIN_3_REG	Rd/Wr	[31:0] – AES256XTSSTGDEC Ekey (rDecEKeyIn[127:96]).
0x0050	DEC_EKEYIN_4_REG	Rd/Wr	[31:0] – AES256XTSSTGDEC Ekey (rDecEKeyIn[159:128]).
0x0054	DEC_EKEYIN_5_REG	Rd/Wr	[31:0] – AES256XTSSTGDEC Ekey (rDecEKeyIn[191:160]).
0x0058	DEC_EKEYIN_6_REG	Rd/Wr	[31:0] – AES256XTSSTGDEC Ekey (rDecEKeyIn[223:192]).
0x005C	DEC_EKEYIN_7_REG	Rd/Wr	[31:0] – AES256XTSSTGDEC Ekey (rDecEKeyIn[255:224]).

0x0060	ENC_TKEYIN_0_REG	Rd/Wr	[31:0] – AES256XTSSTGENC Tkey (rEncTKeyIn[31:0]). When rEncTKeyIn[31:0] is set, rEncTKeyInValid is asserted to '1' and tweakable key setting is started.
0x0064	ENC_TKEYIN_1_REG	Rd/Wr	[31:0] – AES256XTSSTGENC Tkey (rEncTKeyIn[63:32]).
0x0068	ENC_TKEYIN_2_REG	Rd/Wr	[31:0] – AES256XTSSTGENC Tkey (rEncTKeyIn[95:64]).
0x006C	ENC_TKEYIN_3_REG	Rd/Wr	[31:0] – AES256XTSSTGENC Tkey (rEncTKeyIn[127:96]).
0x0070	ENC_TKEYIN_4_REG	Rd/Wr	[31:0] – AES256XTSSTGENC Tkey (rEncTKeyIn[159:128]).
0x0074	ENC_TKEYIN_5_REG	Rd/Wr	[31:0] – AES256XTSSTGENC Tkey (rEncTKeyIn[191:160]).
0x0078	ENC_TKEYIN_6_REG	Rd/Wr	[31:0] – AES256XTSSTGENC Tkey (rEncTKeyIn[223:192]).
0x007C	ENC_TKEYIN_7_REG	Rd/Wr	[31:0] – AES256XTSSTGENC Tkey (rEncTKeyIn[255:224]).
0x0080	DEC_TKEYIN_0_REG	Rd/Wr	[31:0] – AES256XTSSTGDEC Tkey (rDecTKeyIn[31:0]). When rDecTKeyIn [31:0] is set, rDecTKeyInValid is asserted to '1' and tweakable key setting is started.
0x0084	DEC_TKEYIN_1_REG	Rd/Wr	[31:0] – AES256XTSSTGDEC Tkey (rDecTKeyIn[63:32]).
0x0088	DEC_TKEYIN_2_REG	Rd/Wr	[31:0] – AES256XTSSTGDEC Tkey (rDecTKeyIn[95:64]).
0x008C	DEC_TKEYIN_3_REG	Rd/Wr	[31:0] – AES256XTSSTGDEC Tkey (rDecTKeyIn[127:96]).
0x0090	DEC_TKEYIN_4_REG	Rd/Wr	[31:0] – AES256XTSSTGDEC Tkey (rDecTKeyIn[159:128]).
0x0094	DEC_TKEYIN_5_REG	Rd/Wr	[31:0] – AES256XTSSTGDEC Tkey (rDecTKeyIn[191:160]).
0x0098	DEC_TKEYIN_6_REG	Rd/Wr	[31:0] – AES256XTSSTGDEC Tkey (rDecTKeyIn[223:192]).
0x009C	DEC_TKEYIN_7_REG	Rd/Wr	[31:0] – AES256XTSSTGDEC Tkey (rDecTKeyIn[255:224]).
0x00A0	ENC_IVIN_0_REG	Rd/Wr	[31:0] – AES256XTSSTGENC IV (rEnclvIn[31:0]).
0x00A4	ENC_IVIN_1_REG	Rd/Wr	[31:0] – AES256XTSSTGENC IV (rEnclvIn[63:32]).
0x00A8	ENC_IVIN_2_REG	Rd/Wr	[31:0] – AES256XTSSTGENC IV (rEnclvIn[95:64]).
0x00AC	ENC_IVIN_3_REG	Rd/Wr	[31:0] – AES256XTSSTGENC IV (rEnclvIn[127:96]).
0x00C0	DEC_IVIN_0_REG	Rd/Wr	[31:0] – AES256XTSSTGDEC IV (rDecIvIn[31:0]).
0x00C4	DEC_IVIN_1_REG	Rd/Wr	[31:0] – AES256XTSSTGDEC IV (rDecIvIn[63:32]).

0x00C8	DEC_IVIN_2_REG	Rd/Wr	[31:0] – AES256XTSSTGDEC IV (rDeclvIn[95:64]).
0x00CC	DEC_IVIN_3_REG	Rd/Wr	[31:0] – AES256XTSSTGDEC IV (rDeclvIn[127:96]).
0x2000-0x3FFF	PLAIN_BASE_ADDR	Rd/Wr	[31:0] – Plain data in DpRam1 (wRdDataB1).
0x4000-0x5FFF	CIPHER_BASE_ADDR	Rd/Wr	[31:0] – Cipher data in DpRam2 (wRdDataB2).

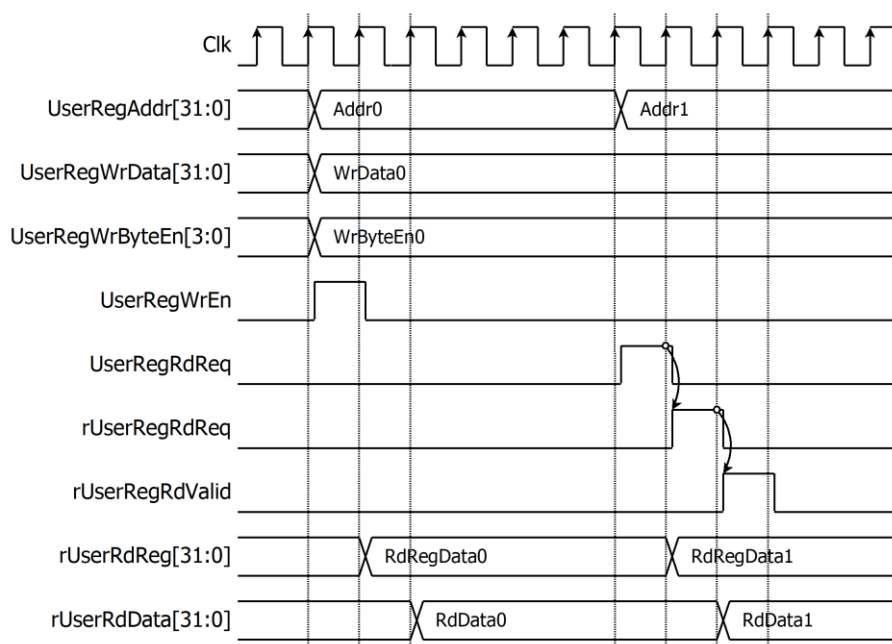


Figure 2-2 Register interface timing diagram

To read register, the multiplexer is designed to select the read data within each address area. UserRegAddr[12:2] is applied in each register area to select the data. Next, the address decoder uses UserRegAddr[15:13] to select the read data from each area for returning to CPU. As shown in Figure 2-2, read data is valid in next two clock cycles. When UserRegRdReq is active, rUserRegRdReq is asserted to ‘1’. Then rUserRdValid is active with the valid read value of UserRegAddr.

To write register, UserRegWrEn is asserted to ‘1’ with the valid of UserRegAddr. UserRegAddr[15:13] is used to decode that CPU accesses dual-port ram (DpRam) or internal register area. When CPU accesses DpRam (UserRegAddr[15:13]=“001” or “010”), UserRegAddr[12:2] is set to be the address of DpRam. For example, when UserRegAddr[15:0]=0x2004 and UserRegWrEn=‘1’, DpRam1 will be filled with UserRegWrData at Address 0x01. Otherwise, UserRegWrData is loaded to internal register which has matched UserRegAddr[12:2]. For example, rAddrA1 is loaded by UserRegWrData when UserRegAddr=0x0000.

UserRegWrByteEn signal is used when CPU firmware needs to access DpRam by using 32-bit, 16-bit or 8-bit pointer. UserRegWrByteEn[3:0] is mapped to Byte Write Enable port of DpRam.

In this reference design, there are three main operations which are parameter setting, encryption and decryption. Each operation is described as follows.

2.4.1 Encryption key setting

For AES256XTSSTGENC encryption key setting, rEncEKeyIn is set by writing register UserRegAddr=0x0020-0x003F as shown in Figure 2-3. When ENC_EKEYIN_0_REG is written (UserRegAddr=0x0020), rEncEKeyIn[31:0] is set. Then rEncEKeyInValid is asserted to '1' as shown in Figure 2-3. AES256XTSSTGENC encryption key setting takes 14 clocks cycles to finish.

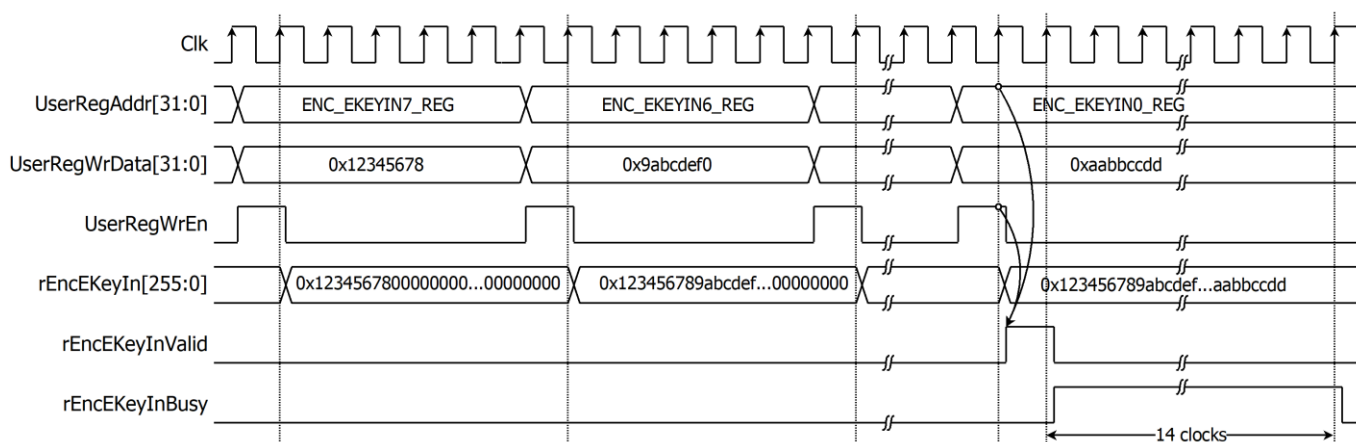


Figure 2-3 Timing diagram of AES256XTSSTGENC encryption key setting process

For AES256XTSSTGDEC encryption key setting, rDecEKeyIn is set by writing register UserRegAddr=0x0040-0x005F as shown in Figure 2-4. When DEC_EKEYIN_0_REG is written (UserRegAddr=0x0040), rDecEKeyIn[31:0] is set. Then rDecEKeyInValid is asserted to '1' as shown in Figure 2-4. AES256XTSSTGDEC encryption key setting takes 14 clocks cycles to finish.

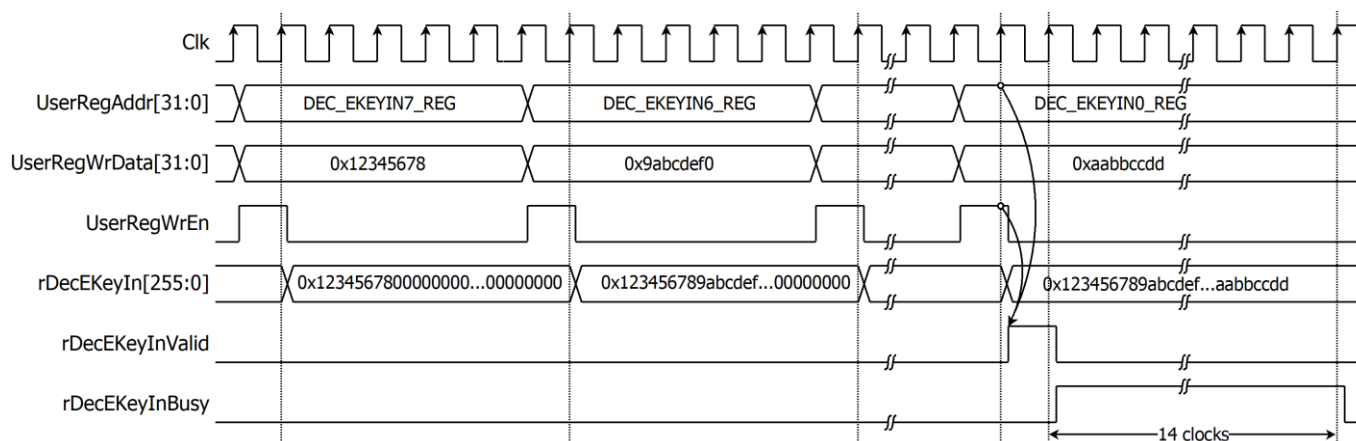


Figure 2-4 Timing diagram of AES256XTSSTGDEC decryption key setting process

2.4.2 Tweakable key setting

For AES256XTSSTGENC tweakable key setting, rEncTKeyIn is set by writing register UserRegAddr=0x0060-0x007F as shown in Figure 2-5. When ENC_TKEYIN_0_REG is written (UserRegAddr=0x0060), rEncTKeyIn[31:0] is set. Then rEncTKeyInValid is asserted to '1' as shown in Figure 2-3. AES256XTSSTGENC tweakable key setting takes 14 clocks cycles to finish.

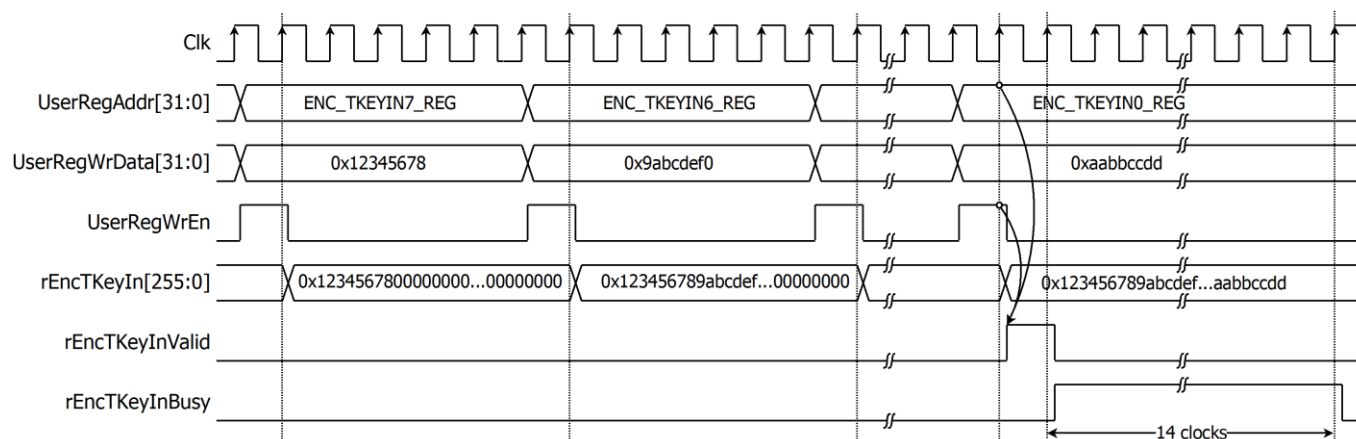


Figure 2-5 Timing diagram of AES256XTSSTGENC tweakable key setting process

For AES256XTSSTGDEC tweakable key setting, rDecTKeyIn is set by writing register UserRegAddr=0x0080-0x009F as shown in Figure 2-6. When DEC_TKEYIN_0_REG is written (UserRegAddr=0x0080), rDecTKeyIn[31:0] is set. Then rDecTKeyInValid is asserted to '1' as shown in Figure 2-4. AES256XTSSTGDEC tweakable key setting takes 14 clocks cycles to finish.

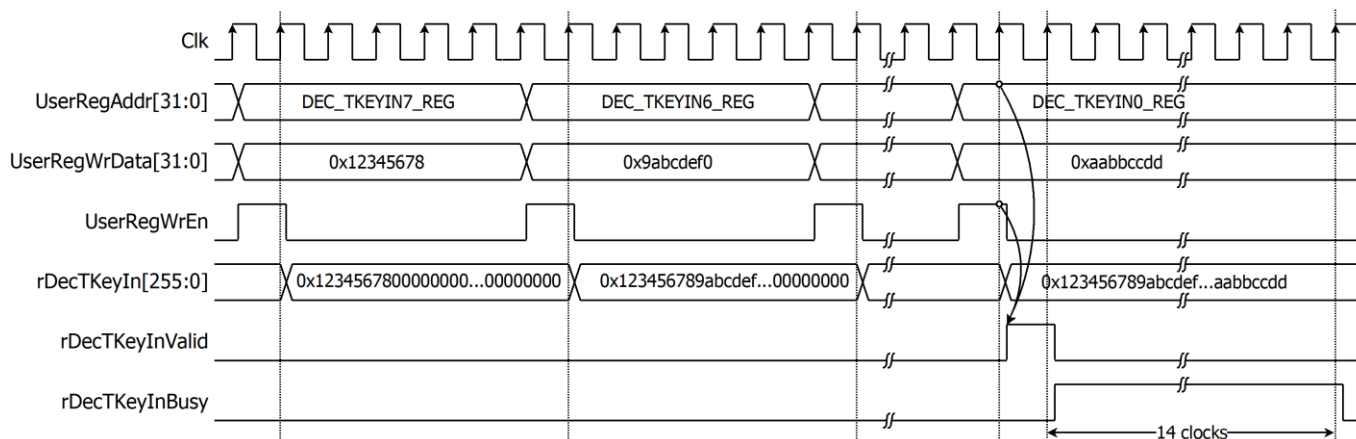


Figure 2-6 Timing diagram of AES256XTSSTGDEC tweakable key setting process

2.4.3 parameter setting

For encryption process, encryption IV is set by writing ENC_IVIN_0_REG to ENC_IVIN_3_REG. plain data, which will be input data for AES256XTSSTGENC, are stored in DpRam1. Before starting encryption process, address A of DpRam1 (rAddrA1) is set to 0x00 to access the first plain data. Encrypted data from AES256XTSSTGENC is stored in DpRam2. Before starting encryption process, address A of DpRam2 (rAddrA2) is set to 0x00 for writing the first output encrypted data. IvIncrement is regulated through rEncIvIncrement, which can be set by writing to the ENC_IVINC_REG. For setting length of plain data, rEncDataInCnt, ENC_DATAINCNT_REG is written and then rEncStart is asserted to '1' to start encryption as shown in Figure 2-7.

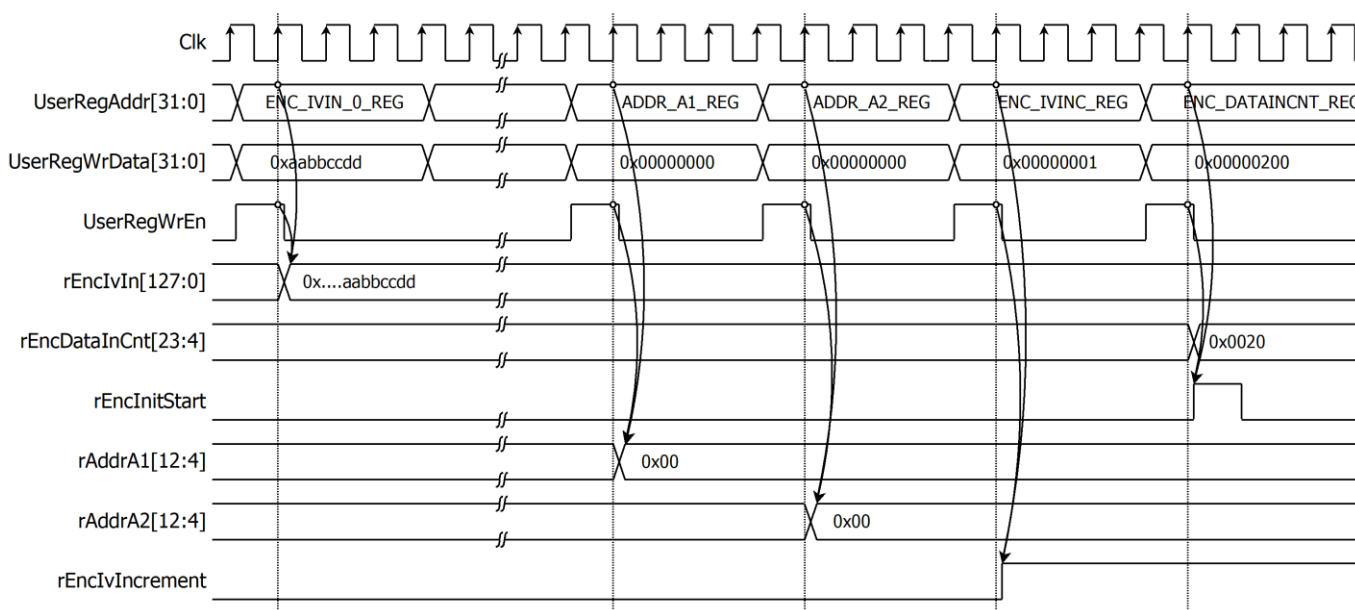


Figure 2-7 Timing diagram example of encryption parameter setting

For decryption process, decryption IV is set by writing DEC_IVIN_0_REG to DEC_IVIN_3_REG. cipher data, which will be input data for AES256XTSSTGDEC, are stored in DpRam2. Before starting decryption process, address A of DpRam2 (rAddrA2) is set to 0x00 to access the first cipher data. Decrypted data from AES256XTSSTGDEC is stored in DpRam1. Before starting decryption process, address A of DpRam1 (rAddrA1) is set to 0x00 for writing the first output decrypted data. IvIncrement is regulated through rDecIvIncrement, which can be set by writing to the DEC_IVINC_REG. For setting length of cipher data, rDecDataInCnt, DEC_DATAINCNT_REG is written and then rDecStart is asserted to '1' to start decryption as shown in Figure 2-8.

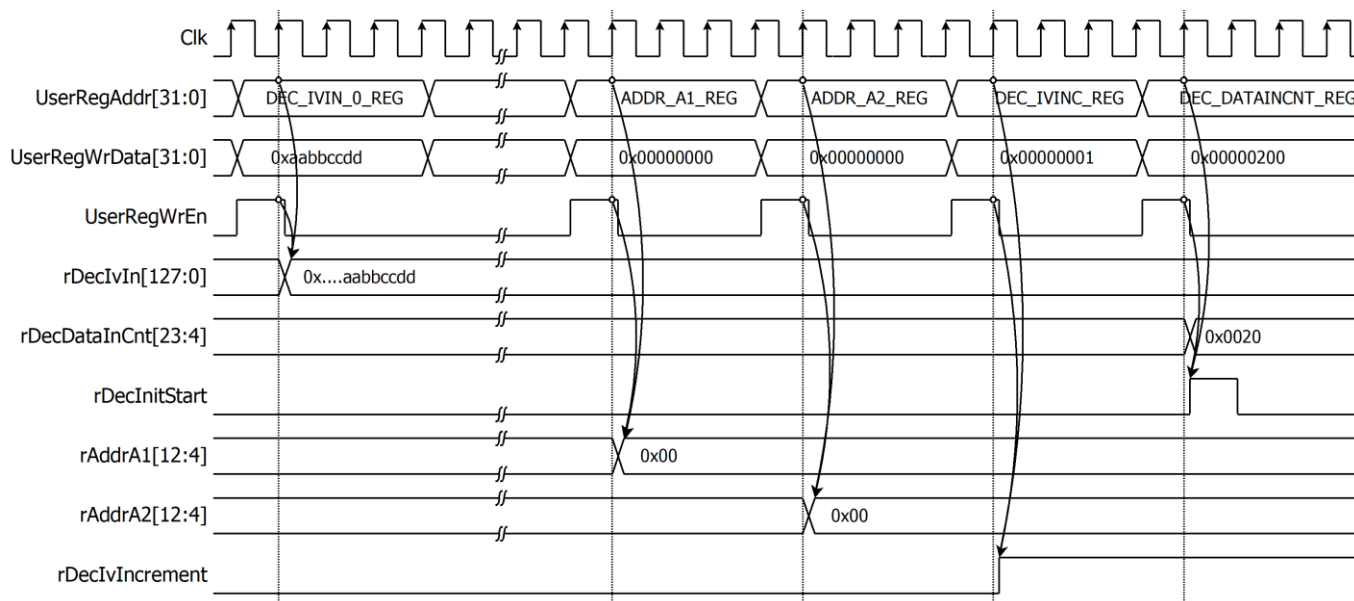


Figure 2-8 Timing diagram example of decryption parameter setting

2.4.4 Encryption

For encryption process, n-bit plain data, which will be input data for AES256XTSSTGENC, are stored in DpRam1. Before starting encryption process, address A of DpRam1 (rAddrA1) is set to 0x00 to access the first plain data. DpRam1 has wRdEnA1 as the read enable signal of the ram and wRdEnA1 is asserted to '1' when wEncDataInReady='1' and rEncDataInCnt>0. rEncDataInValid is delayed one clock cycle from wRdEnA1. As shown in Figure 2-9, when wRdEnA1 is active, rEncDataInCnt is decreased by 1 and rAddrA1 is increased by 1 to access next n-bit plain data.

Before starting encryption process, address A of DpRam2 (rAddrA2) is set to 0x00 for writing the first encrypted data. When wEncDataOutValid is active, wEncDataOut is loaded to DpRam2. Then rAddrA2 is increased by 1 prepared for next encrypted output as shown in Figure 2-9.

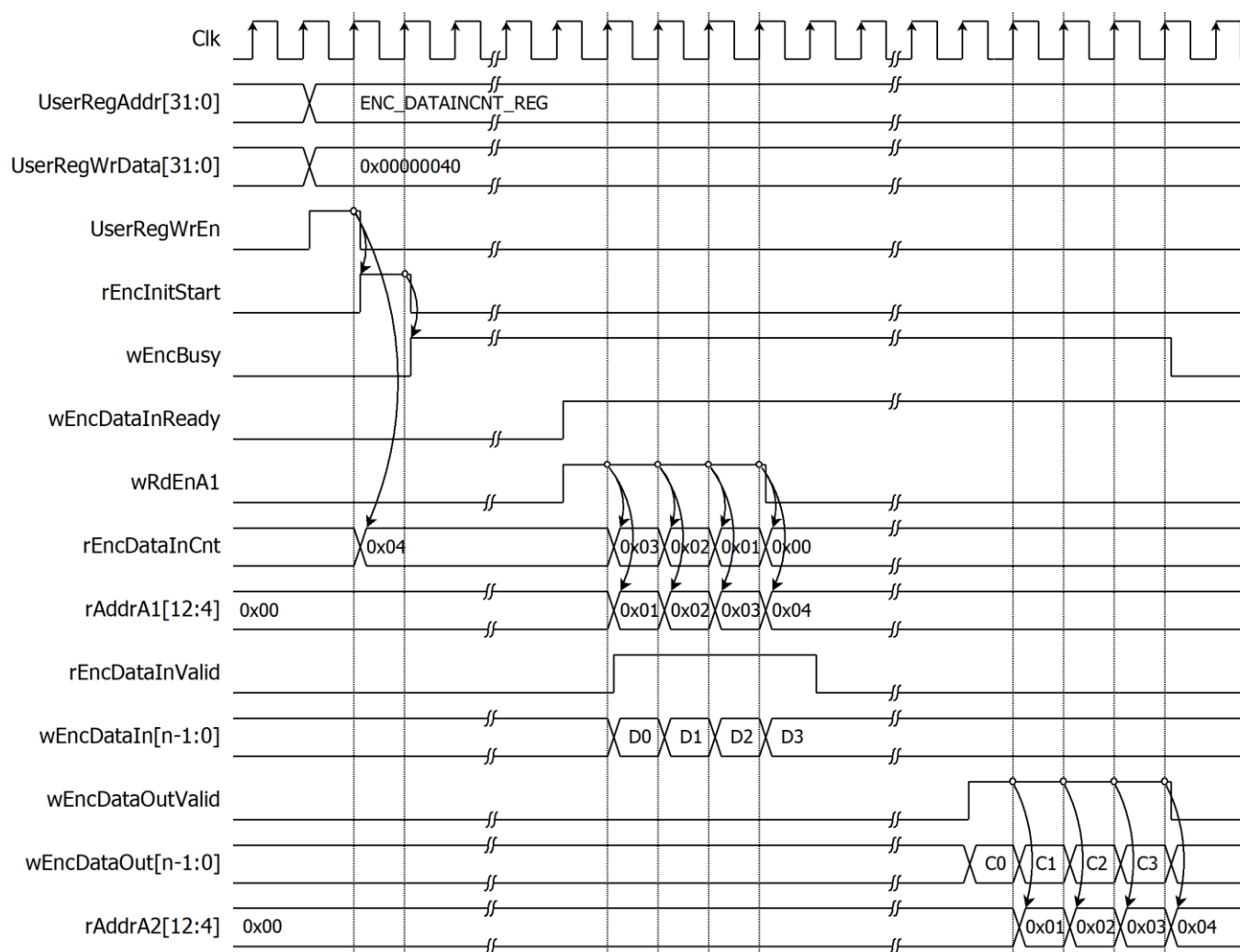


Figure 2-9 : Example of data encryption timing diagram

2.4.5 Decryption

For decryption process, n-bit cipher data, which will be input data for AES256XTSSTGDEC, are stored in DpRam2. Before starting decryption process, address A of DpRam2 (rAddrA2) is set to 0x00 to access the first cipher data. DpRam2 has wRdEnA2 as the read enable signal of the ram and wRdEnA2 is asserted to '1' when wDecDataInReady='1' and rDecDataInCnt>0. rDecDataInValid is delayed one clock cycle from wRdEnA2. As shown in Figure 2-10, when wRdEnA2 is active, rDecDataInCnt is decreased by 1 and rAddrA2 is increased by 1 to access next n-bit cipher data.

Before starting decryption process, address A of DpRam1 (rAddrA1) is set to 0x00 for writing the first decrypted data. When wDecDataOutValid is active, wDecDataOut is loaded to DpRam1. Then rAddrA1 is increased by 1 prepared for next decrypted output as shown in Figure 2-10.

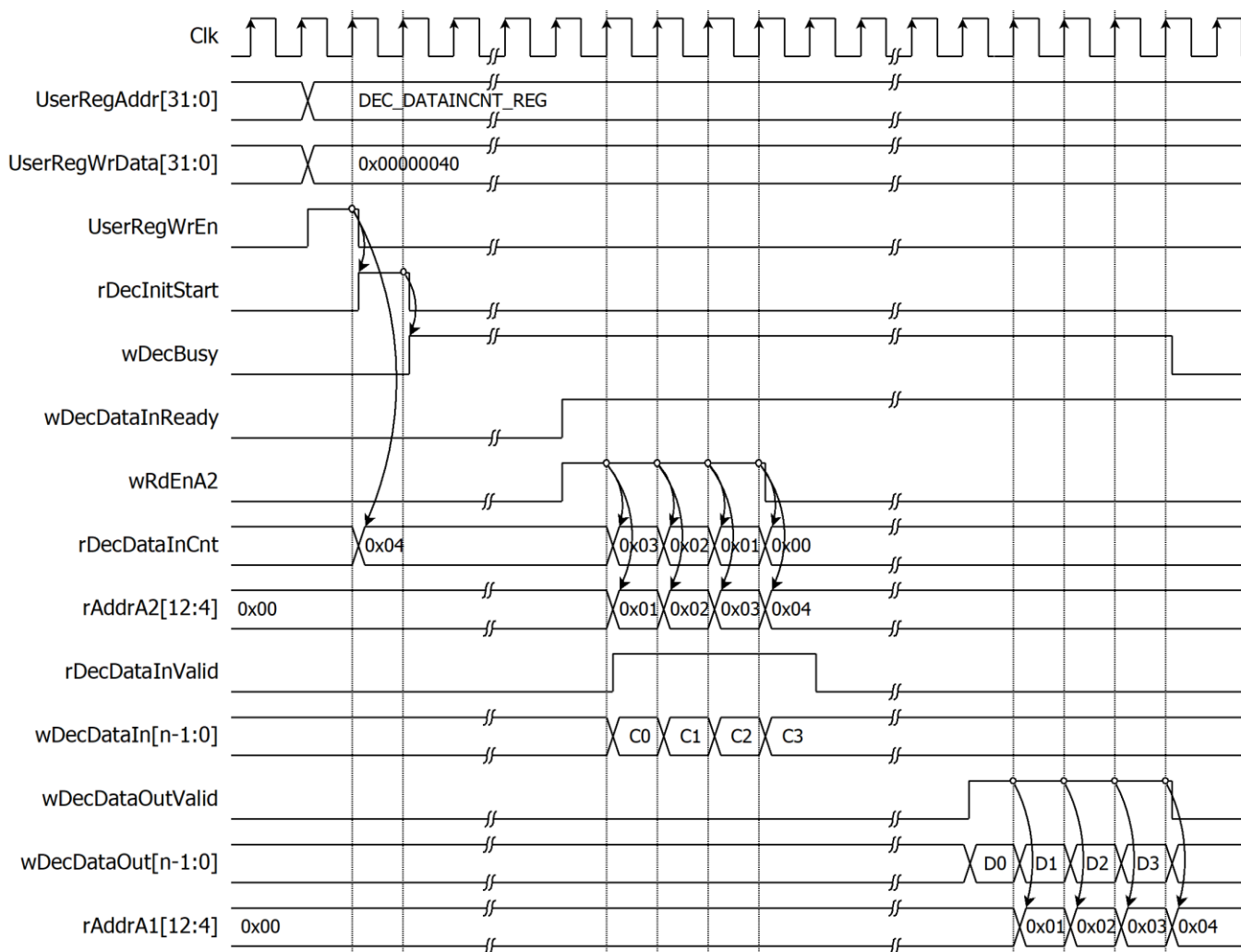


Figure 2-10 Example of data decryption timing diagram

3 CPU Firmware

After system boot-up, CPU initializes its peripherals such as UART and Timer and shows IP version of AES256XTSSTGIP. Then main menu is displayed. Main function runs in an infinite loop to show the main menu and get keyboard input from user. User can select each menu via serial console that will call the related functions. After functions finished running, the main menu is displayed again. More details of the sequence in each menu are described as follows.

3.1 Set encryption key

This menu is used to set encryption key for encryption module and decryption module. `set_key_or_iv` function is called to change Ekey set. User can input “enter” key to skip input Ekey and not change current key for encryption module or to skip input Ekey and change key similar to encryption module for decryption module. `ENC_EKEYIN_0_REG`, `DEC_EKEYIN_0_REG`, `length_hex=64`, “rEncEKeyIn”, “rDecEKeyIn” are input parameters. `set_key_or_iv` function is described in Table 3-1.

Table 3-1 set_key_or_iv function

<pre>void set_key_or_iv(unsigned int *enc_addr, unsigned int *dec_addr, unsigned int length_hex, char *enc_label, char *dec_label)</pre>	
Parameter	<p><code>enc_addr</code>: register for set key or iv of encryption module</p> <p><code>dec_addr</code>: register for set key or iv of decryption module</p> <p><code>length_hex</code>: length string of key or iv</p> <p><code>enc_label</code>: name of signal for encryption module</p> <p><code>dec_label</code>: name of signal for decryption module</p>
Return value	None
Description	<p>This function displays current key or iv on the console. Receive input new key or iv in hexadecimal string format or “enter” key to skip. User’s input will be ignored if it is not hexadecimal format. Then the function monitors busy status until busy status is not active. Then set the registers.</p>

3.2 Set tweakable key

This menu is used to set tweakable key for encryption module and decryption module. `set_key_or_iv` function is called to change Tkey set. User can input “enter” key to skip input Tkey and not change current key for encryption module or to skip input Tkey and change key similar to encryption module for decryption module. `ENC_TKEYIN_0_REG`, `DEC_TKEYIN_0_REG`, `length_hex=64`, “rEncTKeyIn”, “rDecTKeyIn” are input parameters.

3.3 Set encryption/decryption IV

This menu is used to set Initialization Vector (IV) for encryption module and decryption module. `set_key_or_iv` function is called to change iv set. User can input “enter” key to skip input iv and not change current key for encryption module or to skip input iv and change key similar to encryption module for decryption module. `ENC_IVIN_0_REG`, `DEC_IVIN_0_REG`, `length_hex=64`, “rEnclvIn”, “rDecIvIn” are input parameters.

3.4 Show Data Memory

This menu is used to show data in memory. User can set the number of data to show on console in byte unit. The sequence of the firmware is as follows.

- 1) Receive the length of data in decimal format or “enter” key to use default value. The length of data to show is stored in length.
- 2) Call show_data function to show data in memory with PLAIN_BASE_ADDR and length as input parameters. show_data function is described in Table 3-2.

Table 3-2 show_data function

void show_data(unsigned int base_addr, unsigned int length)	
Parameter	base_addr: base address to access memory length: length of data to show
Return value	None
Description	This function shows data in both DpRam1 and DpRam2 corresponding to length. CPU firmware reads data in DpRam1, starting at PLAIN_BASE_ADDR, and shows on console. Also, CPU firmware reads data in DpRam2, starting at CIPHER_BASE_ADDR, and shows on console. If length is too large, length will be set to the size of memory.

3.5 Fill Plain Data Memory

This menu is used to fill DpRam1 with plain data pattern. The sequence of the firmware is as follows.

- 1) Receive the length of plain data in decimal format in 512-byte unite or “enter” key to set length of data to default value. The number of data to encrypt is stored in enc_data_cnt.
- 2) Call fill_data function to choose data pattern and fill DpRam1 with selected pattern. The input parameter of fill_data is PLAIN_BASE_ADDR which is base address for DpRam1 and enc_data_cnt.
- 3) Call show_data function to show data in both DpRam1 and DpRam2 corresponding to enc_data_cnt.

According to Table 3-3, there are four data pattern choices to fill memory that are zero pattern, 8-bit counter pattern, 16-bit counter pattern and 32-bit counter pattern. For zero pattern, user can select this choice to clear data memory. It makes user see the output of encryption/decryption more clearly.

Table 3-3 fill_data function

void fill_data(unsigned int base_addr, unsigned int data_length)	
Parameter	base_addr: base address to access memory data_length: length of data to fill data pattern
Return value	None
Description	This function fills the selected pattern in memory corresponding to base_addr. Before filling memory, the function shows data pattern choices on serial console and waits for user’s input. For invalid choice, the console will display the error message and wait until user input new valid choice. Then selected data pattern is filled into memory by the length of data (data length) and filled zeros into the remaining memory from address base_addr+0x0000 to base_addr+0x1FFF.

3.6 Encrypt

This menu is used to start encryption process with current encryption parameters. The sequence of the firmware is as follows.

- 1) Receive parameter for IvIncrement stored in enc_ivinc_en.
- 2) Call aes_command function to set the starting address, IvIncrement value, number of data to encrypt and check status. ENC_IVINC_REG, ENC_CNT_REG, enc_ivinc_en and enc_data_cnt are input parameters. aes_command function is described in Table 3-7.
- 3) Print plain data length and call show_data function to show data in memory by the number of plain data.

Table 3-7 aes_command function

void aes_command(unsigned int *IVINC_REG, unsigned int *DATACNT_REG, unsigned int ivinc_en, unsigned int data_cnt)	
Parameter	IVINC_REG: register for control IvIncrement DATACNT_REG: register for data length ivinc_en: parameter for IvIncrement data_cnt: the number of data to operate
Return value	None
Description	This function sets the starting address of DpRam1 and DpRam2 by writing register ADDR_A1_REG and ADDR_A2_REG respectively. And set IvIncrement parameter by writing register IVINC_REG with ivinc_en. Then DATACNT_REG is written with data_cnt to set the number of data to operate. Also, DATACNT_REG is monitored to check whether operation finish. When operation finished, *DATACNT_REG=0 which means AES256XTSSTGIP is not busy.

3.7 Fill Cipher Data Memory

This menu is used to fill DpRam2 with cipher data pattern. The sequence of the firmware is as follows.

- 1) Receive the length of cipher data in decimal format in 512-byte unite or “enter” key to set length of data to default value. The number of data to decrypt is stored in dec_data_cnt.
- 2) Call fill_data function to choose data pattern and fill DpRam2 with selected pattern. The input parameter of fill_data is CIPHER_BASE_ADDR which is base address for DpRam2 and dec_data_cnt.
- 3) Call show_data function to show data in both DpRam1 and DpRam2 corresponding to dec_data_cnt.

3.8 Decrypt

This menu is used to start decryption process with current decryption parameters. The sequence of the firmware is as follows.

- 1) Receive parameter for IvIncrement stored in dec_ivinc_en.
- 2) Call aes_command function to set the starting address, IvIncrement value, number of data to decrypt and check status. DEC_IVINC_REG, DEC_CNT_REG, dec_ivinc_en and dec_data_cnt are input parameters.
- 3) Print cipher data length and call show_data function to show data in memory by the number of cipher data.

4 Revision History

Revision	Date	Description
1.00	25-Aug-2023	Initial version release