AES256GCM 10G25G Reference Design

AES256GCM10G25G Reference Design

Rev1.03   29-Aug-2023

1. Introduction

This document describes the detail of AES256GCM10G25GIP reference design. In this reference design, AES256GCM10G25GIP is used to encrypt/decrypt data between two memories in FPGA and provide authentication tag. User can fill memory with Additional Authenticated Data (AAD), plain or cipher data patterns, set 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.
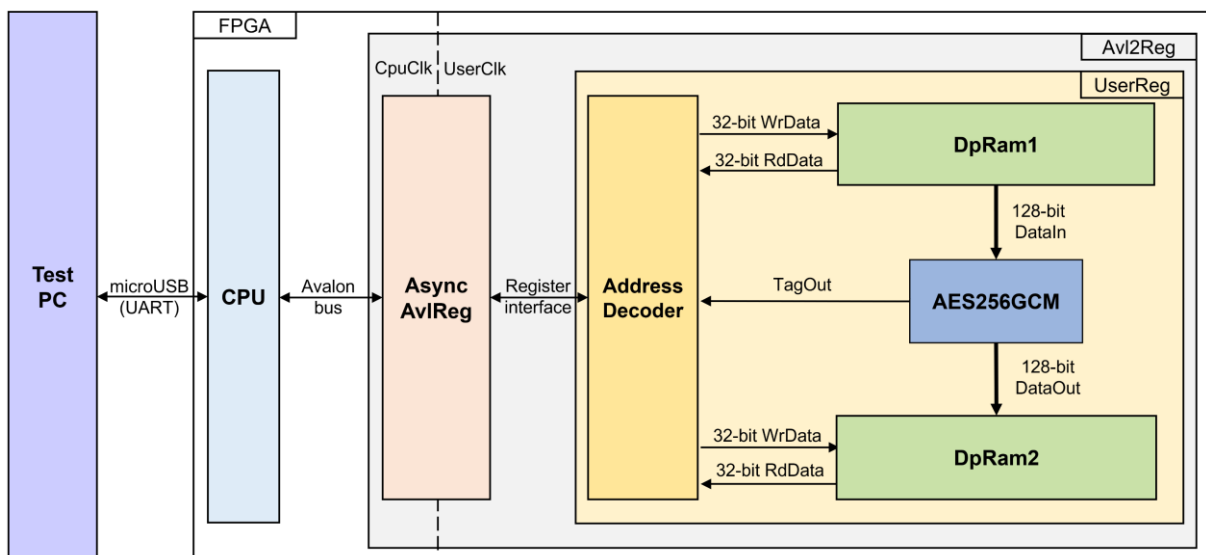
2. Hardware Overview

Figure 2-1 AES256GCM10G25GIP reference design block diagram

In this test environment, the AES256GCM10G25GIP, called AES256GCM, interface with two dual-port RAMs with asymmetric ports, which are DpRam1 and DpRam2, as shown in Figure 2-1. AES256GCM and two RAMs are sub-modules in UserReg module within Avl2Reg. CPU system is designed to interface with FPGA logic through Avalon-MM 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, IV, number of AAD and 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 and read encryption and decryption tag. DpRam1 is used to store AAD and the DataIn from user which will be input data for AES256GCM. DpRam2 is used to store output data from AES256GCM. Authentication tag is stored in registers which user can read.

Because CPU system and AES256GCM run in different clock domain, AsyncAvlReg module inside Avl2Reg is designed as asynchronous circuit to support clock-crossing operation. Also, AsyncAvlReg converts Avalon-MM bus signal which is standard bus in CPU system to be register interface. The details of Avl2Reg module are described as follows.

2.1 Avl2Reg

Avl2Reg module is connected to CPU through Avalon-MM 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 Avl2Reg.

Avl2Reg consists of AsyncAvlReg and UserReg. AsyncAvlReg is designed to convert the Avalon-MM signals to be the simple register interface which has 32-bit data bus size (similar to Avalon-MM 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 Avalon-MM bus and UserClk which is user clock domain for AES256GCM. AsyncAvlReg 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 AES256GCM. Both data interface and control interface of AES256GCM are connected to UserReg. More details of AsyncAvlReg and UserReg are described as follows.

2.2 AsyncAvlReg

This module is designed to convert the signal interface of Avalon-MM 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, AsyncAvlReg 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.

2.3 UserReg

This module is designed to write/read data in RAMs, read tag, control and check status of AES256GCM corresponding with write register access or read register request from AsyncAvlReg 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	[10:4] – address A of DpRam1 (rAddrA1[10:4]).
0x0004	ADDR_A2_REG	Rd/Wr	[10:4] – address A of DpRam2 (rAddrA2[10:4]).
0x0008	AADINCNT_REG	Wr	[15:0] – length of AAD for encryption/decryption (rAadInCount[15:0]).
0x000C	DATAINCNT_REG	Rd	[1] – AES256GCM KeyIn busy flag (wKeyInBusy). [0] – AES256GCM busy flag (wBusy).
		Wr	[15:0] – length of plain data for encryption/ decryption (rDataInCnt[15:0]). rStart is asserted to '1' and encryption/decryption process is started.
0x0010	VER_REG	Rd	[31:0] – AES256GCM IP version (wVersion).
0x0014	DECEN_REG	Rd/Wr	[0] – AES256GCM mode of operation. (rDecryptionEn)
0x0018	BYPASS_REG	Rd/Wr	[0] – AES256GCM Bypass mode of operation. (rBypass)
0x0020	KEYIN_0_REG	Rd/Wr	[31:0] – Encryption/Decryption key (rKeyIn[31:0]).
0x0024	KEYIN_1_REG	Rd/Wr	[31:0] – Encryption/Decryption key (rKeyIn[63:32]).
0x0028	KEYIN_2_REG	Rd/Wr	[31:0] – Encryption/Decryption key (rKeyIn[95:64]).
0x002C	KEYIN_3_REG	Rd/Wr	[31:0] – Encryption/Decryption key (rKeyIn[127:96]).
0x0030	KEYIN_4_REG	Rd/Wr	[31:0] – Encryption/Decryption key (rKeyIn[159:128]).
0x0034	KEYIN_5_REG	Rd/Wr	[31:0] – Encryption/Decryption key (rKeyIn[191:160]).
0x0038	KEYIN_6_REG	Rd/Wr	[31:0] – Encryption/Decryption key (rKeyIn[223:192]).
0x003C	KEYIN_7_REG	Rd/Wr	[31:0] – Encryption/Decryption key (rKeyIn[255:224]).
0x0060	IVIN_0_REG	Rd/Wr	[31:0] – Encryption/Decryption IV (rIvIn[31:0]).
0x0064	IVIN_1_REG	Rd/Wr	[31:0] – Encryption/Decryption IV (rIvIn[63:32]).
0x0068	IVIN_2_REG	Rd/Wr	[31:0] – Encryption/Decryption IV (rIvIn[95:64]).

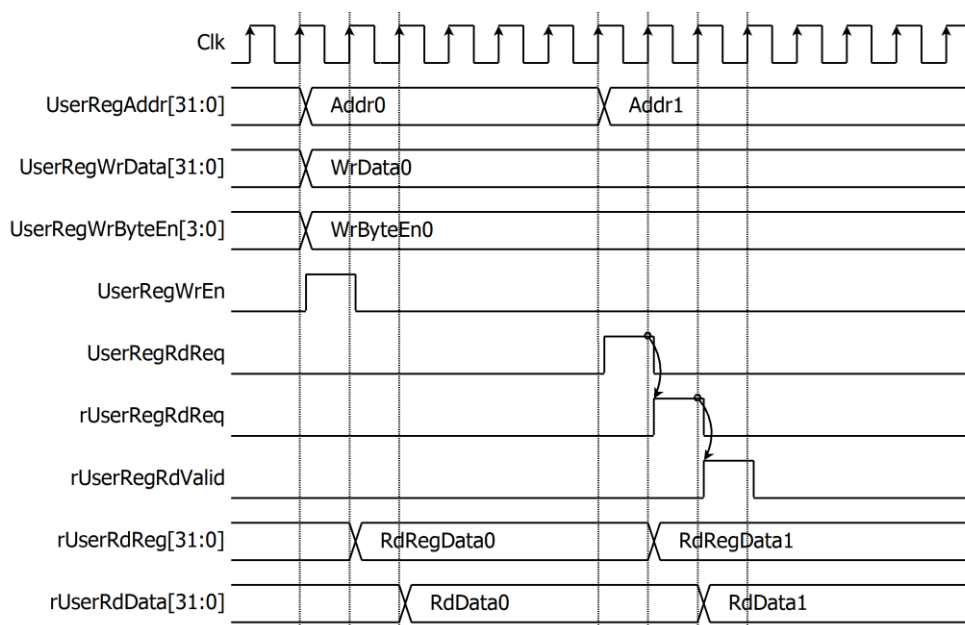| 0x0080	TAG_0_REG	Rd	[31:0] – Authentication tag (rTagOut[31:0]).
0x0084	TAG_1_REG	Rd	[31:0] – Authentication tag (rTagOut[63:32]).
0x0088	TAG_2_REG	Rd	[31:0] – Authentication tag (rTagOut[95:64]).
0x008C	TAG_3_REG	Rd	[31:0] – Authentication tag (rTagOut[127:96]).
0x2000~0x27FF	DATAIN_ADDR	Rd/Wr	[31:0] – Data in DpRam1 (wRdDataB1).
0x4000~0x47FF	DATAOUT_ADDR	Rd/Wr	[31:0] – 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[10:2] is applied in each register area to select the data. Next, the address decoder uses UserRegAddr[15:11] 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[15:11] is used to decode that CPU accesses dual-port ram (DpRam) or internal register area. When CPU accesses DpRam (UserRegAddr[15:11]="00100" or "01000"), UserRegAddr[10:2] 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[10:2]. 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/decryption/bypass. Each operation is described as follows.

2.3.1 Key setting

For key setting, rKeyIn is set by writing register UserRegAddr=0x0020-0x003F as shown in Table 2-1. When KEYIN_0_REG is written (UserRegAddr=0x0020), rKeyIn[31:0] is set. Then rKeyInValid is asserted to '1' as shown in Figure 2-3.
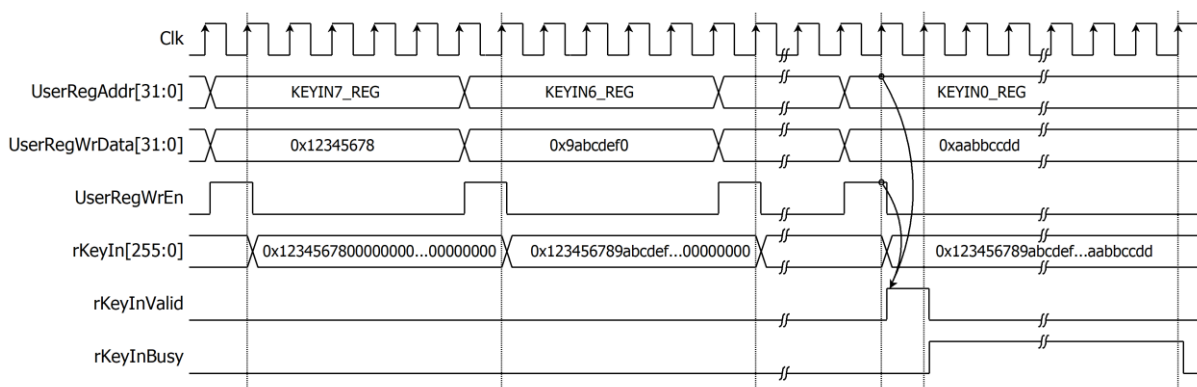

Figure 2-3 Timing diagram of key setting process

2.3.2 Parameter setting

For parameter setting, encryption/decryption mode is set by DECEN_REG. Bypass mode is set by BYPASS_REG. IV and length of AAD are set by writing IVIN_0_REG to IVIN_2_REG and AADCNT_REG, respectively. AAD and DataIn, which will be input data for AES256GCM, are stored in DpRam1, address A of DpRam1 (rAddrA1) is set to 0x00 to access the first AAD data by writing ADDR_A1_REG. DataOut from AES256GCM is stored in DpRam2. Before starting encryption/decryption process, address A of DpRam2 (rAddrA2) is set to 0x00 for writing the first output encrypted/decrypted data by writing ADDR_A2_REG. For setting length of DataIn, rDataInCnt, DATAINCNT_REG is written and then rStart is asserted to '1' to start encryption/decryption as shown in Figure 2-4.
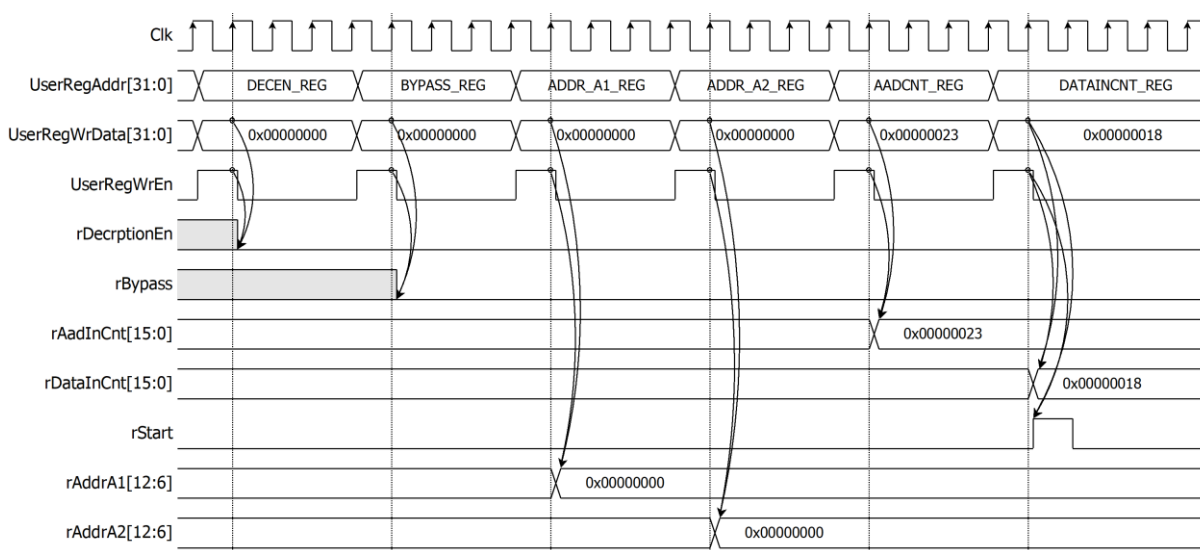

Figure 2-4 Timing diagram example of parameter setting

2.3.3 Encryption/Decryption/Bypass

DpRam1 has wRdEnA1 as the read enable signal of the ram and wRdEnA1 is asserted to '1' when rStart='1' or wDataInRd='1'. For encryption/decryption process, rAddrA1 is set to 0x00 to access the first DataIn. When wRdEnA1 is active, rAddrA1 is increased by 1 to access next 512-bit DataIn. When wDataOutValid is active, wDataOut is loaded to DpRam2. Then rAddrA2 is increased by 1 prepared for next DataOut as shown in Figure 2-5. Authentication tag (rTagOut) is stored in register when wTagOutValid is active as shown in Table 2-1. User can access the authentication tag by reading TAG_0_REG to TAG_7_REG.
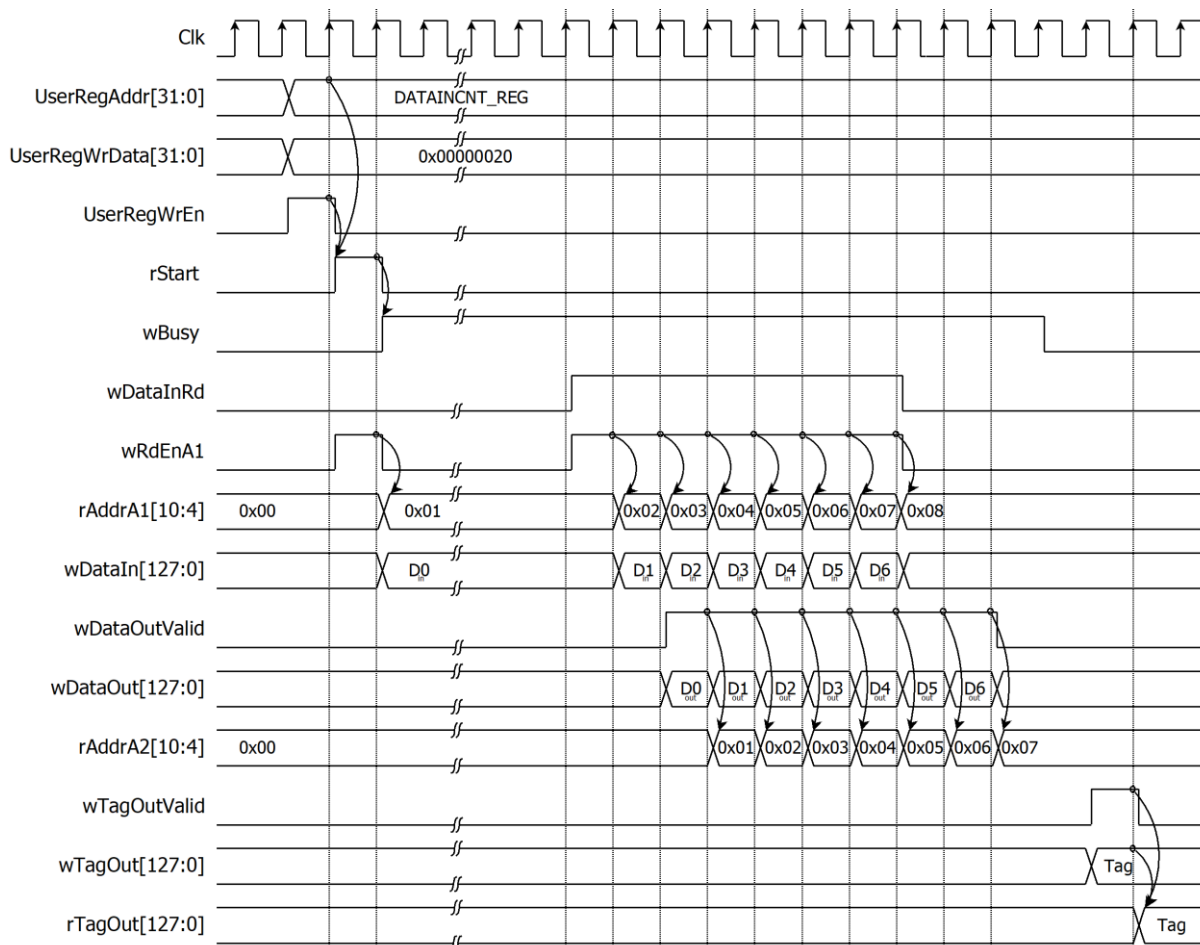

Figure 2-5 Timing diagram of encryption/decryption/bypass process

Note: For bypass mode, wTagOutValid will not be active, and rTagOut is not valid.

3. CPU Firmware

After system boot-up, CPU initializes its peripherals such as UART and Timer and shows IP version of AES256GCM. 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/decryption key

This menu is used to set encryption/decryption key. set_key_or_iv function is called to change key set. User can input "enter" key to skip input key and not change current key for module. KEYIN_0_REG, 64-character, "KeyIn" are input parameters. set_key_or_iv function is described in Table 3-1.

Table 3-1 change_key_set function

void set_key_or_iv(unsigned int *start_addr, unsigned int length_hex, char *label)	
Parameter	start_addr: register for set key or iv of module
	length_hex: length character of key or iv
	label: name of signal for module
Return value	None
Description	This function displays current key or iv on the console. Receive input new key or iv in hexadecimal 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.

3.2 Set encryption/decryption IV

This menu is used to set Initialization Vector (IV). set_key_or_iv function is called to change iv set. User can input "enter" key to skip input iv and not change current iv. IVIN_0_REG, 24-character, "IvIn" are input parameters.

3.3 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. If user input '0', the length of data to show will be default value. The length of data to show is stored in length.

2) Call show_data function to show data in memory with DATAIN_ADDR and length as its 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 DATAIN_ADDR, and shows on console. Also, CPU firmware reads data in DpRam2, starting at DATAOUT_ADDR, and shows on console. If length is too large, length will be set to the size of memory.

3.4 Fill AAD Memory

This menu is used to set the length of AAD and fill DpRam1 with AAD pattern. The sequence of the firmware is as follows.

1) Receive the length of AAD in byte unit or "enter" key to set AAD length to zero. The length of AAD is stored in aad_cnt and calculates the first address for fill data is stored at start_addr.

2) Call fill_data function to choose AAD pattern and fill DpRam1 with selected pattern. The input parameters of fill_data are 0x01, DATAIN_ADDR which is the first address for AAD in DpRam1, and aad_cnt. fill_data function is described in Table 3-3.

3) Call show_data function to show AAD data for both DpRam1 and DpRam2 corresponding to aad_cnt and data_cnt.

Table 3-3 fill_data function

void fill_data(unsigned int zero_padding, unsigned int base_addr, unsigned int data_length)	
Parameter	zero_padding: zero padding to 128-bit, 1 to fill right-padding with zeros and 0 is ignored.
	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 (zeros, 8-bit, 16-bit, 32-bit counter) 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).

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 DataIn in byte unit or "enter" key to set length of data to zero. The number of DataIn is stored in data_cnt.

2) Call fill_data function to choose data pattern and fill DpRam1 with selected pattern. The input parameter of fill_data is 0x00, start_addr which is base address for DpRam1 and data_cnt.

3) Call show_data function to show data in both DpRam1 and DpRam2 corresponding to aad_cnt and 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 pattern and 32-bit pattern. For zero pattern, user can select this choice to clear data memory. It makes user see the output of encryption/decryption more clearly.

3.6 Encrypt Data

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

1) Call aes_command function to set mode of operation, the starting address, number of AAD and DataIn to encrypt and check status. aes_command function is described in Table 3-4.

2) Print AAD length and DataIn length in byte unit. Call show_data function to show data in memory according to aad_cnt and data_cnt.

3) Call show_tag function to display encryption tag. show_tag function is described in Table 3-5.

Table 3-4 aes_command function

void aes_command(unsigned int mode, unsigned int *AADCNT_REG, unsigned int aad_cnt, unsigned int *DATACNT_REG, unsigned int data_cnt)	
Parameter	mode: mode of operation, 0 for encryption, 1 for decryption, 2 for bypass
	AADCNT_REG: register for AAD length
	aad_cnt: the number of AAD to operate
	DATACNT_REG: register for data length
	data_cnt: the number of data to operate
Return value	None
Description	This function set encryption/decryption mode by writing register DECEN_REG and set Bypass mode by writing register BYPASS_REG, sets the number of AAD by writing register AADCNT_REG with aad_cnt and sets the starting address of DpRam1 and DpRam2 by writing register ADDR_A1_REG and ADDR_A2_REG, respectively. 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, *CMD_REG=0 which means AES256GCM is not busy (*DATACNT_REG = 0).

Table 3-5 show_tag function

void show_tag(unsigned int *TAG_REG)	
Parameter	TAG_REG: register for reading authentication tag
Return value	None
Description	This function reads authentication tag from TAG_REG to TAG_REG+0x000F and prints tag on serial console.

3.7 Decrypt Data

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

1) Call aes_command function to set mode of operation, the starting address, number of AAD and DataIn to decrypt and check status.

2) Print AAD length and DataIn length in byte unit. Call show_data function to show data in memory according to aad_cnt and data_cnt.

3) Call show_tag function to display decryption tag.

3.8 Bypass Data

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

1) Call aes_command function to set mode of operation, the starting address, number of AAD and DataIn to bypass and check status.

2) Print AAD length and DataIn length in byte unit. Call show_data function to show data in memory according to aad_cnt and data_cnt.

3.9 Clone Memory

This menu is used to clone DpRam2 to DpRam1.

3.10 Loop verification

This menu is used to verify both encryption and decryption with the current parameters by calling loop_verify function which is described in Table 3-6.

Table 3-6 loop_verify function

int loop_verify(unsigned int aad_cnt, unsigned int data_cnt)	
Parameter	aad_cnt: the number of AAD for encryption and decryption
	data_cnt: the number of data for encryption and decryption
Return value	Int: 0 for verification succeeded and -1 for verification failed
Description	This function reads and stores plain data from DpRam1 for verification. Data in DpRam2 is cleared. aes_command function is called to encrypt plain data with current parameters. Then copy the data in DpRam2 to DpRam1 and aes_command function is called to decrypt data with current parameters. The decrypted data is compared with plain data before encryption process and decryption tag is compared with encryption tag. Verification result will be printed on serial console.

4. Revision History

Revision	Date	Description
1.00	01-Oct-2022	Initial version release
1.02	27-Oct-2022	Update description for new design
1.03	20-Jan-2023	- Add Bypass feature. - Improve performance.