ChaCha20Poly1305-IP Reference Design

 © Design Gateway Co., Ltd.

ChaCha20Poly1305-IP Reference Design

Rev1.01 31-Jul-2026

1 Introduction

This document describes the detail of ChaCha20Poly1305-IP reference design. In this reference design, ChaCha20Poly1305-IP is used to encrypt and decrypt data between two memories in FPGA and provide authentication tag. User can fill memory with Additional Authenticated Data (AAD), plain or cipher data, set encryption/decryption key, Initialization Vector (IV), and control the test operation and monitor results via serial console on a test PC. More details of the hardware design and CPU firmware are described as follows.

2 Hardware Overview

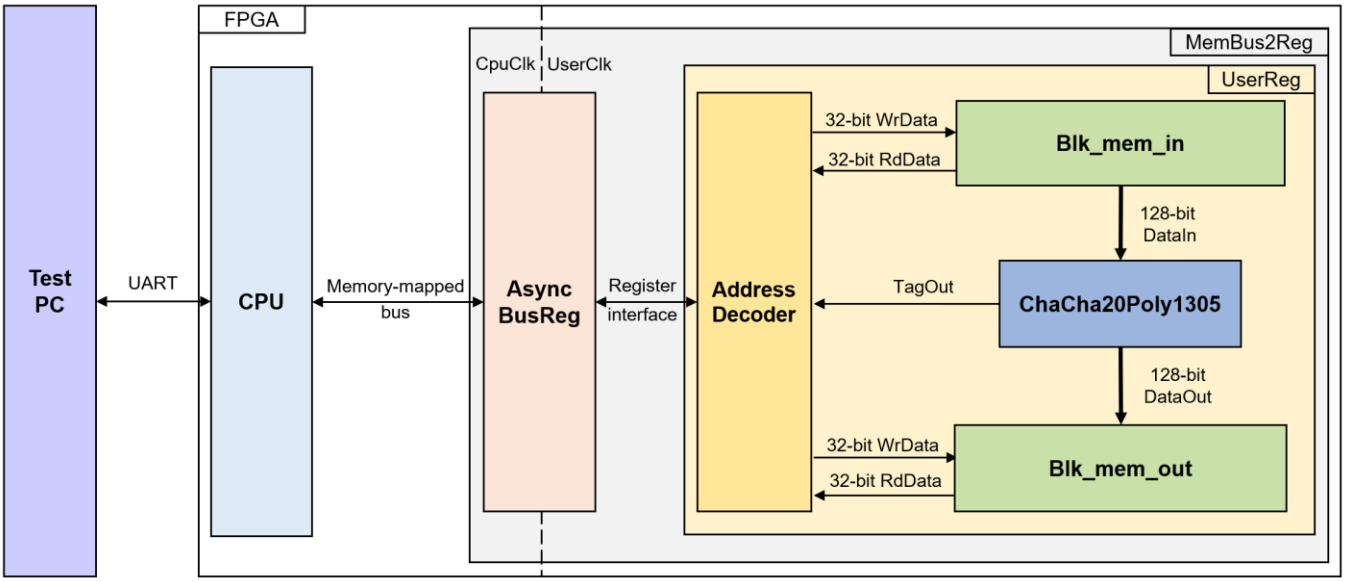

Figure 1 ChaCha20Poly1305-IP reference design block diagram

In this test environment, the ChaCha20Poly1305-IP interfaces with two dual-port RAMs with asymmetric ports, which are Blk_mem_in and Blk_mem_out, as shown in Figure 1. ChaCha20Poly1305-IP and two RAMs are sub-modules in UserReg module within MemBus2Reg. CPU system is designed to interface with FPGA logic through memory-mapped bus (Avalon-MM) and interface with user through serial console on test PC.

For user control interface, there are registers in UserReg to store parameters from user such as encryption and decryption keys, initialization vector (IV), the 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 or read data in RAMs following user's command and read authentication tag. Blk_mem_in is used to store AAD and the DataIn from user which will be input data for ChaCha20Poly1305-IP. Blk_mem_out is used to store output data from ChaCha20Poly1305-IP. Authentication tag is stored in registers which user can read.

Because CPU system and ChaCha20Poly1305-IP run in different clock domain, AsyncBusReg module inside MemBus2Reg is designed as asynchronous circuit to support clock-crossing operation. Also, AsyncBusReg converts memory-mapped bus signal which is standard bus in CPU system to be register interface. The details of MemBus2Reg module are described as follows.

 © Design Gateway Co., Ltd.

2.1 MemBus2Reg Module

The MemBus2Reg module interfaces with the CPU through a memory-mapped bus, such as Avalon-MM. The hardware registers within MemBus2Reg are mapped to specific CPU memory addresses, as shown in Table 1. These registers include control and status registers that enable the CPU to access and manage the module.

MemBus2Reg consists of two main sub-modules: AsyncBusReg and UserReg. The AsyncBusReg sub-module is responsible for converting the signals from the memory-mapped bus into a simple register interface that uses a 32-bit data bus, maintaining consistency with the bus's data size. As shown in Figure 1, the MemBus2Reg module operates with two clock domains: CpuClk, which interfaces with the CPU, and UserClk, which operates in the user-defined clock domain. The AsyncBusReg sub-module includes circuitry to handle asynchronous communication between these two clock domains.

UserReg includes the register file of the parameters and the status signals of test logics, including dual-port RAMs and ChaCha20Poly1305-IP. Both data interface and control interface of ChaCha20Poly1305-IP are connected to UserReg. More details of AsyncBusReg and UserReg are described as follows.

2.2 AsyncBusReg

This module is designed to convert the signal interface of a memory-mapped bus into a 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, AsyncBusReg 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.

 © Design Gateway Co., Ltd.

2.3 UserReg

This module is designed to write/read data in RAMs, read tag, control and check status of ChaCha20Poly1305-IP corresponding with write register access or read register request from AsyncBusReg module. Memory map inside UserReg module is shown in Table 1. Timing diagram of register interface is shown in Figure 2.

Table 1 Register Map Definition

Address offset	Register Name	Rd/Wr	Description
0x0000	STATUS_ADDR	Rd	[0] – ChaCha20Poly1305-IP busy flag (!rOperationEn).
0x0100	PARAMS_ADDR	Rd/Wr	[2] – Enable Speed test mode (rSpeedTestEn). [1] – Enable Bypass mode (rBypassEn) [0] – Enable Decrypt mode (rDecryptEn)
0x0200	AAD_CNT_ADDR0	Rd/Wr	[31:0] – length of AAD for encryption/decryption (rAadCnt [31:0]).
0x0204	AAD_CNT_ADDR1	Rd/Wr	[31:0] – length of AAD for encryption/decryption (rAadCnt [63:32]).
0x0300	DATA_CNT_ADDR0	Rd/Wr	[31:0] – length of Plaintext or Ciphertext (rDataCnt [31:0]).
0x0304	DATA_CNT_ADDR1	Rd/Wr	[31:0] – length of Plaintext or Ciphertext (rDataCnt [63:32]).
0x0400	INPUT_RD_ADDR	Wr	[0] – Set rOperationEn to ‘1’ then KeyValid will be asserted when KeyReady = ‘1’ to start operation.
0x0500	KEY_IN_ADDR0	Rd/Wr	[31:0] – Encryption/Decryption key (rKeyIn[31:0]).
0x0504	KEY_IN_ADDR1	Rd/Wr	[31:0] – Encryption/Decryption key (rKeyIn[63:32]).
0x0508	KEY_IN_ADDR2	Rd/Wr	[31:0] – Encryption/Decryption key (rKeyIn[95:64]).
0x050C	KEY_IN_ADDR3	Rd/Wr	[31:0] – Encryption/Decryption key (rKeyIn[127:96]).
0x0510	KEY_IN_ADDR4	Rd/Wr	[31:0] – Encryption/Decryption key (rKeyIn[159:128]).
0x0514	KEY_IN_ADDR5	Rd/Wr	[31:0] – Encryption/Decryption key (rKeyIn[191:160]).
0x0518	KEY_IN_ADDR6	Rd/Wr	[31:0] – Encryption/Decryption key (rKeyIn[223:192]).
0x051C	KEY_IN_ADDR7	Rd/Wr	[31:0] – Encryption/Decryption key (rKeyIn[255:224]).
0x0600	IV_IN_ADDR0	Rd/Wr	[31:0] – Encryption/Decryption IV (rIvIn[31:0]).
0x0604	IV_IN_ADDR1	Rd/Wr	[31:0] – Encryption/Decryption IV (rIvIn[63:32]).
0x0608	IV_IN_ADDR2	Rd/Wr	[31:0] – Encryption/Decryption IV (rIvIn[95:64]).
0x0700	TAG_OUT_ADDR0	Rd	[31:0] – Authentication tag (rTagOut[31:0]).
0x0704	TAG_OUT_ADDR1	Rd	[31:0] – Authentication tag (rTagOut[63:32]).
0x0708	TAG_OUT_ADDR2	Rd	[31:0] – Authentication tag (rTagOut[95:64]).
0x070C	TAG_OUT_ADDR3	Rd	[31:0] – Authentication tag (rTagOut[127:96]).
0x0800	IPVERSION_REG	Rd	[31:0] – ChaCha20Poly1305-IP version (wVersion).
0x4000~0x4FFF	DATA_IN_ADDR	Rd/Wr	[31:0] – Data in Blk_mem_in (wRdDataB1).
0x8000~0x8FFF	DATA_OUT_ADDR	Rd/Wr	[31:0] – Data in Blk_mem_out (wRdDataB2).

 © Design Gateway Co., Ltd.

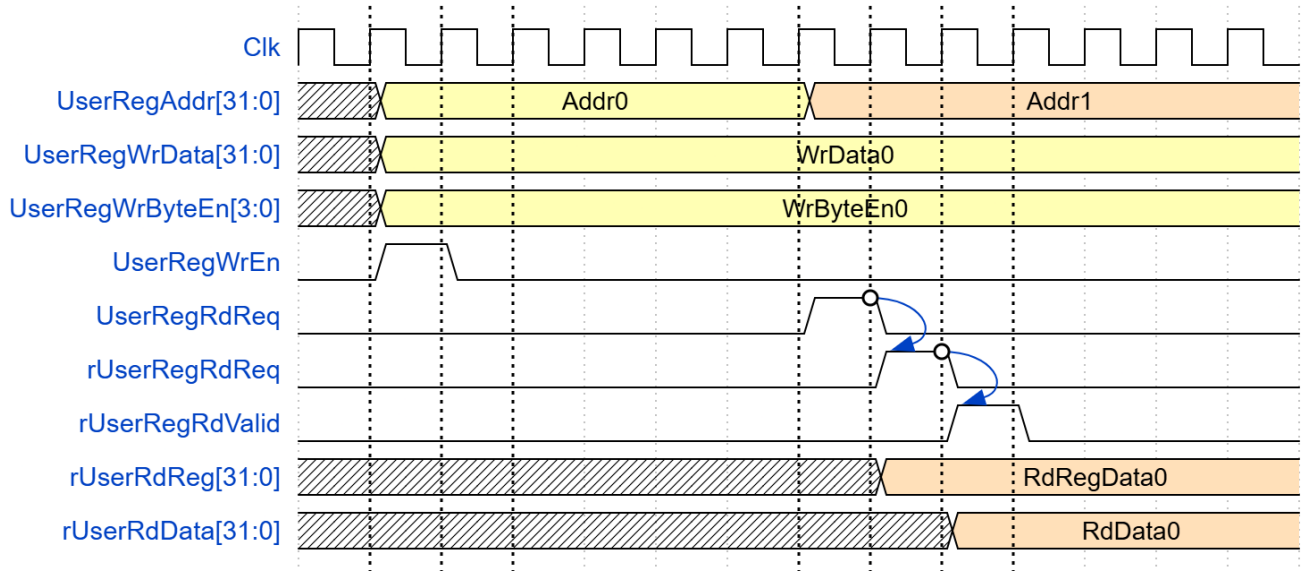

Figure 2 Register interface timing diagram

To read register, one 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, 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]=0x4004 and UserRegWrEn='1', DpRam will be filled with UserRegWrData at Address 0x01. Otherwise, UserRegWrData is loaded to internal register which has matched UserRegAddr[10:2]. For example, rAadCnt is loaded by UserRegWrData when UserRegAddr=0x0200.

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/IV Setting

For key configuration, rKeyIn is configured by writing to registers KEY_IN_ADDR7 through KEY_IN_ADDR0, and rIvIn is configured by writing to registers IV_IN_ADDR2 through IV_IN_ADDR0. The timing diagram is shown in Figure 3.

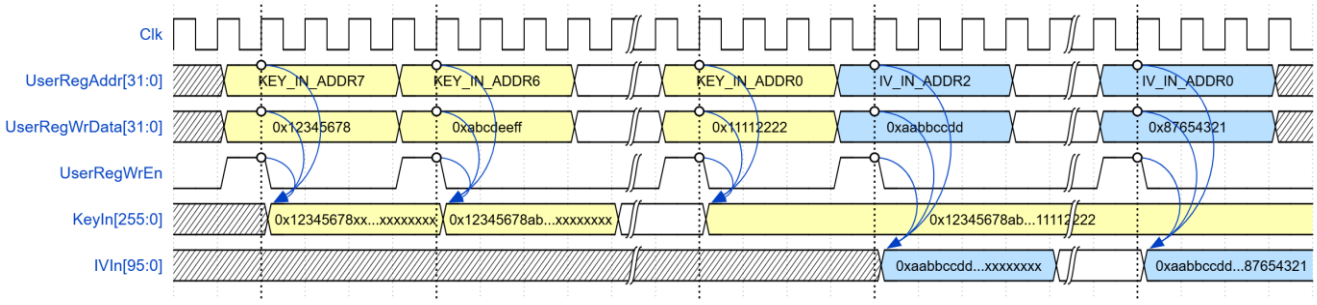

Figure 3 Timing diagram of Key/IV setting process

 © Design Gateway Co., Ltd.

2.3.2 Parameter Setting

The operation mode is configured through the PARAMS_ADDR register.

- 0x00: Encryption mode
- 0x01: Decryption mode
- 0x02: Bypass mode
- 0x04: SpeedTest mode

The AAD length is set by writing to AAD_CNT_ADDR0 and AAD_CNT_ADDR1. The Data length is set by writing to DATA_CNT_ADDR0 and DATA_CNT_ADDR1. The timing diagram is shown in Figure 4.

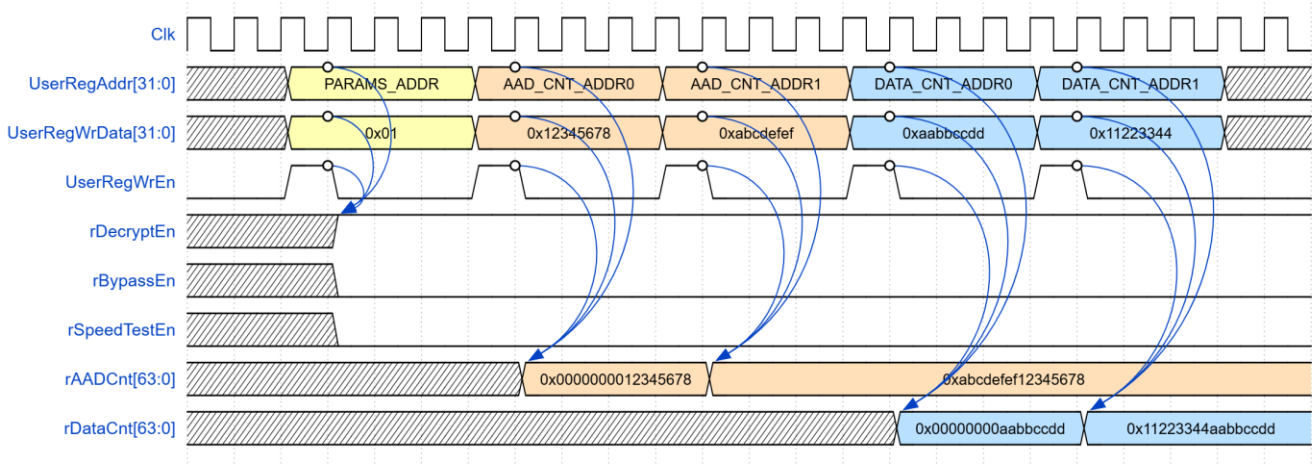

Figure 4 Timing diagram example of parameter setting

 © Design Gateway Co., Ltd.

2.3.3 Encryption/Decryption/Bypass Operation

For encryption, decryption, or bypass operations, the rDecryptEn, rBypassEn, and rSpeedTestEn registers are used to define the operation type. wBramInAddrB is set to 0x00 to access the first 128-bit block of DataIn. When either wAADOutValid or wDataOutValid becomes active, wDataOut is stored in Blk_mem_out RAM, then rBramOutAddrB and wBramInAddrB is incremented by 1 to prepare for the next data as shown in Figure 5.

The operation begins by writing '1' to bit 0 of the INPUT_RD_ADDR register (which sets rOperationEn to '1'). The signal wKeyInValid becomes active when both rOperationEn and wKeyInReady are '1'. Once wKeyInValid is asserted, the ChaCha20Poly1305-IP will be ready to receive data within a few clock cycles.

The authentication tag is stored in a register (rTagOut) after wTagOutValid is active, and the user can access this tag by reading TAG_OUT_ADDR0 to TAG_OUT_ADDR3.

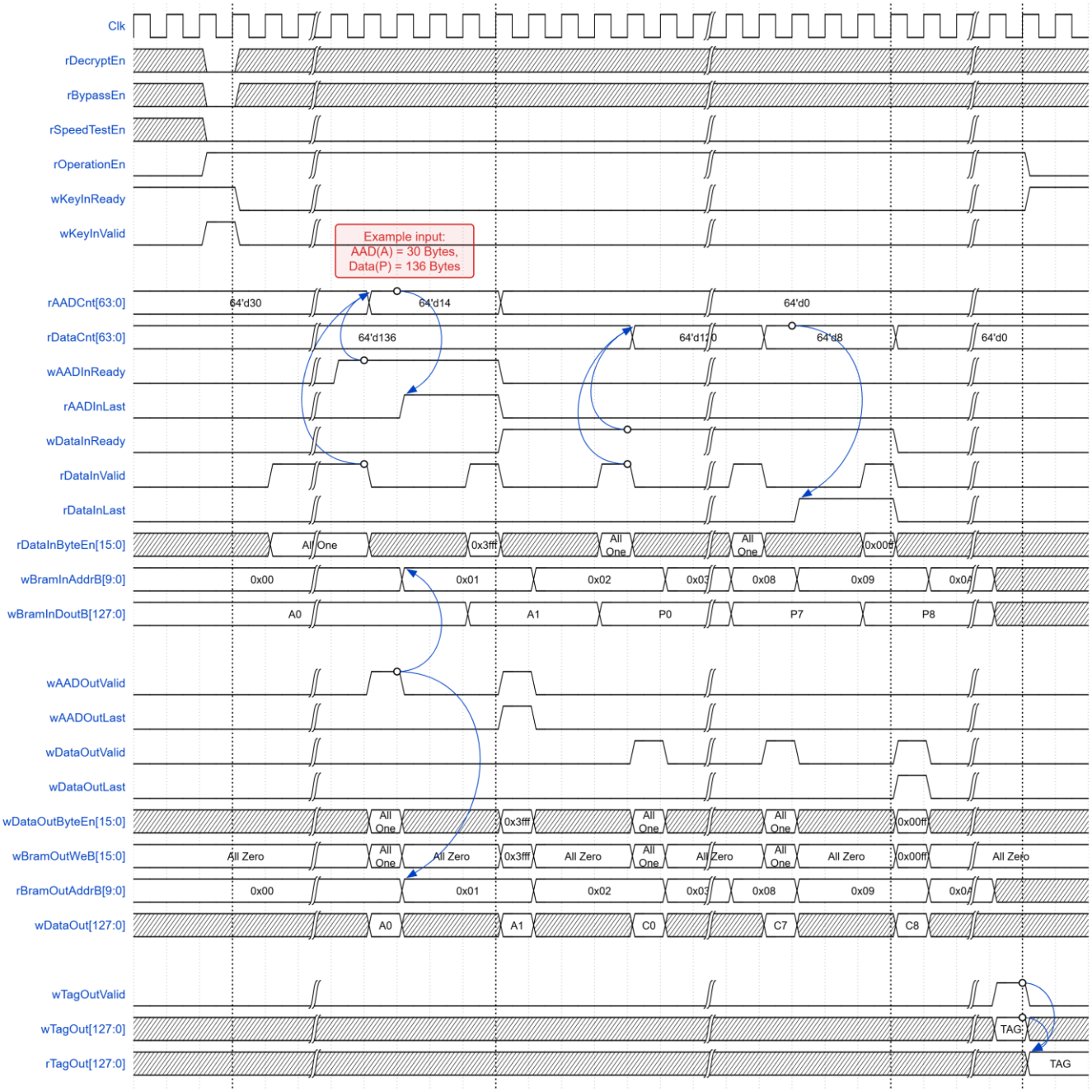

Figure 5 Example timing diagram of encryption mode

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

 © Design Gateway Co., Ltd.

2.3.4 SpeedTest Operation

When the operation begins in SpeedTest mode, wDataIn is fixed at 0, and all control signals such as rDataInValid, rAADInLast, and rDataInLast behave as shown in Figure 6. The wDataCnt signal counts the number of bytes that the IP core receives as DataIn. The rAADInLast and rDataInLast signals become active when their conditions are met, as indicated in the timing diagram.

The authentication tag is calculated using the ChaCha20Poly1305-IP encryption mode, and the user can access this tag by reading from TAG_OUT_ADDR0 to TAG_OUT_ADDR3.

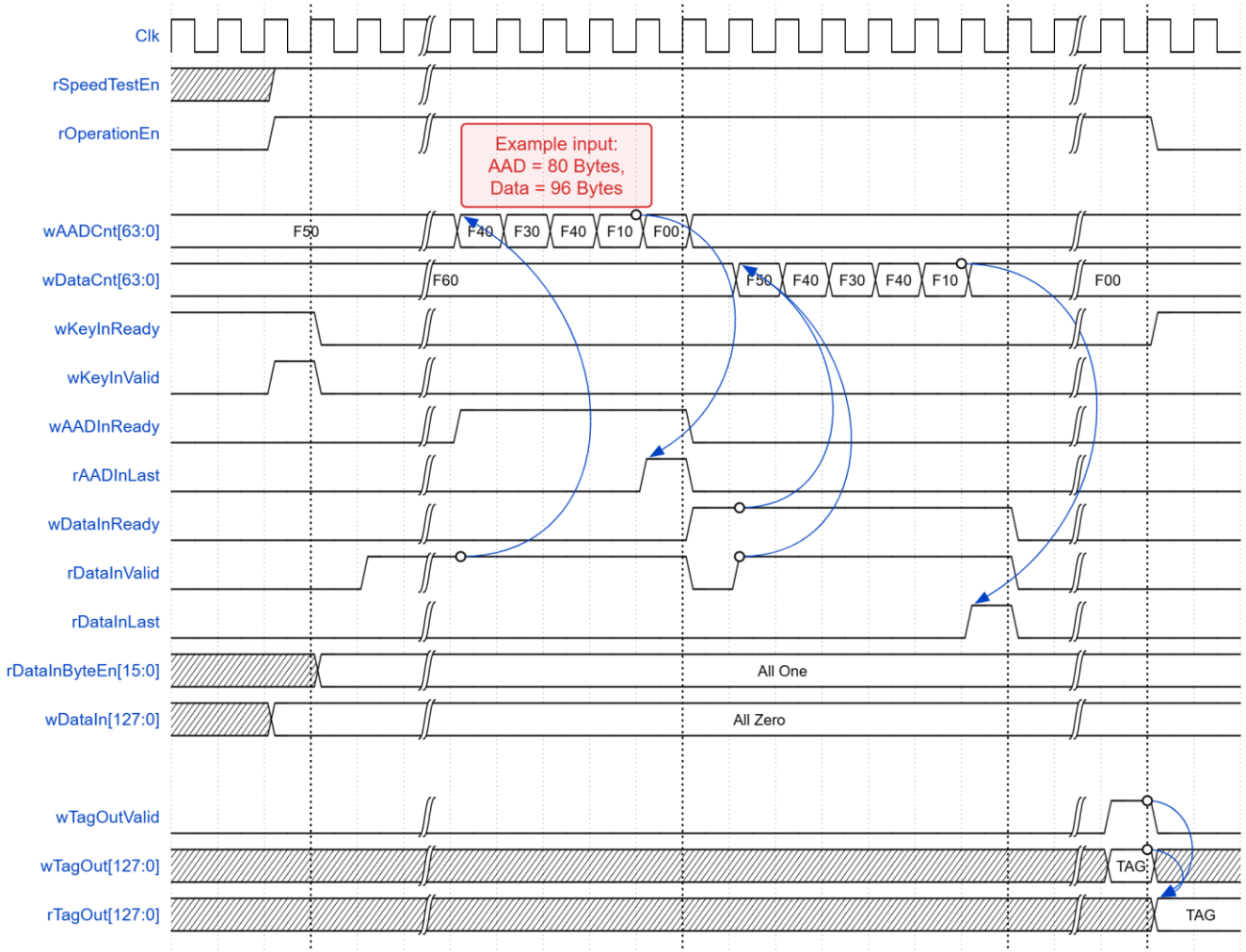

Figure 6 Example timing diagram of SpeedTest mode

 © Design Gateway Co., Ltd.

3 CPU Firmware

After system boot-up, the CPU initializes its peripherals, such as the UART and Timer, and displays the version of the ChaCha20Poly1305-IP. The init_params() function is called during startup to load the default parameters. The main() function then calls user_interface(), which continuously displays the main menu and receives keyboard input from the user. Selecting a menu item calls the corresponding function or operation. After the selected operation finishes, the main menu is displayed again.

Table 2 init_params function

void init_params()	
Parameter	None
Return value	None
Description	This function initializes the ChaCha20Poly1305-IP with default test parameters from RFC 8439. It writes a 256-bit test key to the KEY_IN_ADDR registers, writes a 96-bit test IV to the IV_IN_ADDR registers, and fills the DATA_IN_ADDR memory with the test AAD and input Data. The AAD and Data are padded to 16-byte boundaries in memory. Their original byte lengths are written to the variables referenced by aad_size and data_size.

3.1 Change Mode

This menu is used to set operation mode from user input by writing down PARAMS_ADDR register. Because SpeedTest mode uses arbitrary virtual AAD and Data sizes that are not backed by real buffer memory, the firmware saves the current AAD and Data length whenever the user switches into SpeedTest mode, and restores those saved lengths whenever the user switches back to any other mode (Encryption/Decryption/Bypass mode). The key, IV, AAD, and data contents themselves are left untouched by SpeedTest mode.

 © Design Gateway Co., Ltd.

3.2 Edit Encryption/Decryption Key

This menu is used to set the encryption/decryption key. The user is prompted to enter a new key in hexadecimal format. The firmware calls getHex() with KEY_IN_ADDR as the destination address and a maximum input length of 64 hexadecimal characters. Pressing 'Enter' without entering a value, or pressing 'q', leaves the current key unchanged. If the entered key is shorter than 256 bits, the remaining key registers are filled with zeros.

Table 3 getHex function

uint8_t getHex(uint32_t *str, uint64_t *strLen, uint64_t Max_length)	
Parameter	str: Pointer to where converted hex values will be stored strLen: Pointer to where the number of bytes written will be stored Max_length: Maximum length allowed for input in hexadecimal characters
Return value	0: Successful input - user pressed Enter 1: 'q' key pressed (cancel operation)
Description	This function continuously reads hexadecimal characters from UART and stores them in an array until the number of input characters reaches Max_length. The function terminates when 'Enter' or 'q' is pressed. When the user presses Enter, the function processes all stored data by taking 4 bytes (8 hex characters) at a time, converting each group to a 32-bit unsigned integer with endian swapping, and storing the results directly in the provided uint32_t *str. If there is an odd number of hex digits, the function pads with '0' on the right. If 'q' is pressed, the function terminates without writing any values.

3.3 Edit Encryption/Decryption IV

This menu is used to set the Initialization Vector (IV). The user is prompted to enter a new IV in hexadecimal format. The firmware calls getHex() with IV_IN_ADDR as the destination address and a maximum input length of 24 hexadecimal characters. Pressing 'Enter' without entering a value, or pressing 'q', leaves the current IV unchanged. If the entered IV is shorter than 96 bits, the remaining IV registers are filled with zeros.

3.4 Edit AAD & Data Memory

This menu is used to set the length and fill the memory for AAD and input data. The behavior differs based on the current operation mode. The sequence of the firmware is as follows:

For SpeedTest mode:

1) Prompt the user to enter the AAD length in bytes.
   - Pressing 'Enter' without entering a value sets the AAD length to zero.
   - Pressing 'q' keeps the previous AAD length
2) Round the AAD length up to the next 16-byte boundary.
3) Prompt the user to enter the Data length in bytes.
   - Pressing 'Enter' without entering a value sets the Data length to zero.
   - Pressing 'q' keeps the previous Data length.
4) Round the Data length up to the next 16-byte boundary.

For Other Modes (Encryption/Decryption/Bypass mode):

1) Prompt the user to enter the AAD value in hexadecimal format. Call getHex() with DATA_IN_ADDR as the destination address and MEM_SIZE16k as the maximum number of hexadecimal characters.
   - Pressing 'Enter' without entering a value sets the AAD length to zero.
   - Pressing 'q' cancels the AAD and Data editing sequence.
2) If the AAD length is non-zero and is not a multiple of 16 bytes, write zeros after the AAD value up to the next 16-byte boundary.
3) Prompt the user to enter the Data value in hexadecimal format. The Data is stored after the padded AAD area.
   - Pressing 'Enter' without entering a value sets the Data length to zero.
   - Pressing 'q' clears the current input attempt and displays the Data prompt again.
4) If the Data length is not a multiple of 16 bytes, clear the memory following the entered Data, up to the previous Data length.

 © Design Gateway Co., Ltd.

3.5 Show AAD & Data Memory

This menu is used to display the contents of Input Data memory. User can specify the number of bytes to display. The sequence of the firmware is as follows:

1) Check current mode - if in SpeedTest mode, skip display (return to main menu) as no actual data is stored in this mode.
2) Prompt user to enter the number of bytes to display, or press 'Enter' to display all data.
3) Call print_col() function to display memory contents starting from DATA_IN_ADDR with the specified length.

Table 4 print_col function

void print_col(const uint8_t *col, const uint64_t col_len)	
Parameter	col: Pointer to the data array to display col_len: Number of bytes to display
Return value	None
Description	This function displays memory contents in a formatted hexadecimal table. It prints a header row showing column positions (0 through F) in cyan color, followed by rows of data. Each row starts with an 8-digit hexadecimal address offset in cyan, followed by up to 16 bytes of data in hexadecimal format with spacing after every 4 bytes for readability.

 © Design Gateway Co., Ltd.

3.6 Execute Operation

This menu is used to start the operation process based on the currently selected mode. The firmware executes encryption, decryption, bypass, or speed test operation.

The sequence is as follows:

1) Write the AAD length to the AAD_CNT_ADDR registers and the Data length to the DATA_CNT_ADDR registers.
2) Start the timer.
3) Write 1 to the INPUT_RD_ADDR register to start processing.
4) Poll bit 0 of the STATUS_ADDR register until operation ends.
5) Stop the timer and calculate the execution time.
6) Display the operation result and calculated throughput.

   For SpeedTest mode:

   a) Display generated 128-bit authentication tag from TAG_OUT_ADDR registers
   b) Show execution time and calculated throughput in Mbps

   For Other Modes (Encryption/Decryption/Bypass mode):

   a) Display side-by-side comparison of Input Data and Output Data using print_col_diff() function
   b) Display generated 128-bit authentication tag from TAG_OUT_ADDR registers (not applicable in bypass mode)
   c) Show execution time and calculated throughput in Mbps

Table 5 print_col_diff function

void print_col_diff(const uint8_t *col1, const uint8_t *col2, const uint64_t col_len)	
Parameter	col1: Pointer to first byte array (Input Data) col2: Pointer to second byte array (Output Data) col_len: Number of bytes to display
Return value	None
Description	This function displays two memory regions side-by-side for comparison. It prints a double-width header showing column positions for both data columns in cyan. Each row begins with an 8-digit hexadecimal address offset in cyan, followed by up to 16 bytes from col1 in hexadecimal format, then spacing, then up to 16 bytes from col2 with the same formatting. This allows direct visual comparison between input and output data.

 © Design Gateway Co., Ltd.

4 Revision History

Revision	Date (D-M-Y)	Description
1.01	31-Jul-26	Update description for new design
1.00	14-Oct-25	Initial version release

 © Design Gateway Co., Ltd.