Acc For SM4 IP User Guide

IP Introduction

Version: Ver 1.0

Environment: Vivado 2018.3

Design Code: verilog HDL

Technical reference:

Recorder:Ge Wen Jie
Time: 2022.10.4

目录

一、IP 核概述

SM4(acc_for_sm4) 加密 IP 用于对输入明文按照国密 SM4 加密算法进行对称加密；

acc_for_sm4 为一 AXI 总线标准封装的 IP，具有三种不同的封装形式：

1.**直接电路封装**：该封装方式的 IP 可嵌入至 SOC 的中间数据通路中；

2.**AXI-Lite 封装**：该封装方式的 IP 可应用于低功耗、低数据带宽、低资源占用率的场合，面向功能的实现；

3.**AXI-Stream 封装**： 应用于高带宽、高总线利用率的场合；

本册介绍第二种 AXI-Lite 封装形式的 SM4 加密 IP（acc_for_sm4）；

一、IP 核特点

AXI-Lite 标准接口；

一路可控外部中断信号（上升沿触发）；

支持中断控制与软件查询两种通用工作方式；

最大时钟频率 169MHz；

加密数据最大输出频率 1.76MHz；

极低的资源消耗；

极低的功耗；

具有加密和解密模式；

单次 SM4 加密消息输入 128bits；

单次 SM4 加密密钥输入 128bits；

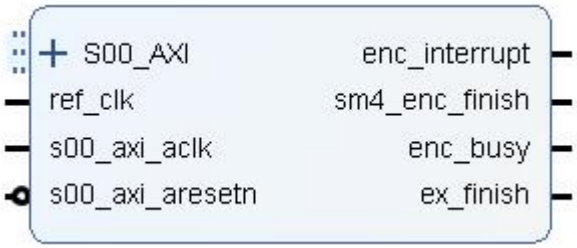
单次 SM4 加密密文输出 128bits；

二、IP 核最低要求资源配置（以 Zynq 为标准)

LUT	Block Ram	IO	F7 Mux	F8_Mux	Slice
451	3	0	64	32	212

100MHz 下功耗参考：0.008W

三、IP 核引脚说明



S00_AXI:AXI-Lite 接口通信线；
s00_axi_aclk: AXI 参考时钟；
s00_axi_aresetn: AXI 复位信号；
ref_clk: sm4 编解码参考时钟；
enc_interrupt: 编解码完成中断信号（上升沿有效）；
sm4_enc_finish: 编解码完成信号（高电平有效）；
enc_busy: 繁忙标志位（高电平有效），高电平时说明正在处理一次编解码事件；
ex_finish: 密钥拓展完成信号（高电平有效）；

四、寄存器及相关信号说明

4.1 寄存器说明

register	offset	width	describe
CTL0	0x00	32	控制寄存器 0，控制 SM4 IP 的工作状态；
ENC_BUFFERx(x=0,1,2,3)	0x04~0x10	32	128bit 明文数据输入寄存器；
KEY_BUFFERx(x=0,1,2,3)	0x14~0x20	32	128bit 密钥输入寄存器；
RES_BUFFERx(x=0,1,2,3)	0x24~0x30	32	128bit 加密/解密结果输出寄存器;
IFG0	0x34	32	状态标志寄存器，暂存 SM4 IP 的工作状态；

4.2 CTL0 寄存器信号说明

Bit	function	type	default	describe
31-5	Reserved	RW	0h	保留位 默认为 0
3	sm4_mode	RW	0h	工作模式标志位； 1b = 解码模式； 0b = 编码模式；
2	sm4_valid	RW	0h	数据有效信号； 1b = 明文寄存器与密钥寄存器中的数据有效； 0b = 明文寄存器与密钥寄存器中的数据无效，忽略 AXI 总线的相关数据输入；
1	en	RW	0h	使能信号； 1b = 使能； 0b = 禁能；
0	nrst	RW	0h	复位信号； 1b = 正常工作； 0b = 复位；

4.3 ENC_BUFFERx(x=0,1,2,3)信号说明

ENC_BUFFER 暂存 128bits 的明文输入，ENC_BUFFER0~ENC_BUFFER3 以每个寄存器 32bit 形式由低到高存储 128bit；

Bit	function	type	default	describe
31-0	ENC_BUFFER	RW	0h	以 32bit 为单位存储 128bit 的明文数据

4.4 KEY_BUFFERx(x=0,1,2,3)信号说明

KEY_BUFFER 暂存 128bits 的密钥输入，KEY_BUFFER0~KEY_BUFFER3 以每个寄存器 32bit 形式由低到高存储 128bit；

Bit	function	type	default	describe
31-0	KEY_BUFFER	RW	0h	以 32bit 为单位存储 128bit 的密钥数据

4.5 RES_BUFFERx(x=0,1,2,3)信号说明

RES_BUFFER 暂存 128bits 的加解密结果输出，RES_BUFFER0~RES_BUFFER3 以每个寄存器 32bit 形式由低到高存储 128bit；

Bit	function	type	default	describe
31-0	RES_BUFFER	RW	0h	以 32bit 为单位存储 128bit 的加解密结果数据

4.6 IFG0 信号说明

Bit	function	type	default	describe
31-5	Reserved	R	0h	保留位 默认为 0
3	ex_finish	R	0h	拓展密钥结束标志位； 1b = 密钥拓展完成； 0b = 密钥拓展进行中；
2	enc_interrupt	R	0h	中断有效信号； 1b = 外部中断有效； 0b = 外部中断无效，需忽略；
1	enc_busy	R	0h	加/解密繁忙标志位； 1b = 繁忙； 0b = 空闲；
0	sm4_enc_finish	R	0h	加/解密完成标志位； 1b = 加/解密完成； 0b = 加/解密数据未完成；

五、工作模式接收

SM4 IP 共有两种工作模式分别为编码模式和解码模式，可通过设置 CTL0 的 sm4_mode 比特位进行切换；

在不同的编解码模式切换顺序组合上，其操作也有相应不同；

5.1 仅用作编码模式（且中途不替换密钥）

此时只需将 CTL0 的 sm4_mode 比特位复位，并写入一次 key 密钥数据；

之后写入明文信息，然后将 valid 信号拉高再拉低即可触发一次编码操作；

当 sm4_enc_finish 信号为高时则一次编码完成，此时 RES_BUFFER 存储的编码值有效；

5.2 仅用作编码模式（且中途替换密钥）

将 CTL0 的 sm4_mode 比特位复位，并写入一次 key 密钥数据；

之后写入明文信息，然后将 valid 信号拉高再拉低即可触发一次编码操作；
当 sm4_enc_finish 信号为高时则一次编码完成，此时 RES_BUFFER 存储的编码值有效；
若下次加密需要替换密钥，则直接将新的 key 密钥数据写入，然后重复上述的操作；
*注：这种情况下的操作方式与编码模式相同，仅多了密钥的重新写入操作；

5.3 仅用作解码模式（且中途不替换密钥）

首先将 CTL0 的 sm4_mode 比特位置位；
写入一次 Key 密钥数据；
将 valid 信号拉高再拉低触发一次密钥拓展操作（内部的密钥拓展暂存模块会存储密钥的拓展值）；
当 ex_finish 信号为高时则一次拓展密钥完成，之后若不替换密钥，则无需再进行上述操作了；
写入明文信息，然后将 valid 信号拉高再拉低即可触发一次编码操作；
当 sm4_enc_finish 信号为高时则一次编码完成，此时 RES_BUFFER 存储的解码值有效；

5.4 仅用作解码模式（且中途替换密钥）

首先将 CTL0 的 sm4_mode 比特位置位；
写入一次 Key 密钥数据；
将 valid 信号拉高再拉低触发一次密钥拓展操作（内部的密钥拓展暂存模块会存储密钥的拓展值）；
当 ex_finish 信号为高时则一次拓展密钥完成，之后若不替换密钥，则无需再进行上述操作了；
写入明文信息，然后将 valid 信号拉高再拉低即可触发一次编码操作；
当 sm4_enc_finish 信号为高时则一次编码完成，此时 RES_BUFFER 存储的解码值有效；
重复上述操作，写入新的密钥值和加密明文；

5.5 由编码模式切换为解密模式（且中途不替换密钥）

由于在加密过程中，密钥拓展模块会对拓展密钥进行暂存，因此解密过程和 5.1 提到的加密过程相似；
首先将 CTL0 的 sm4_mode 比特位置位，切换至解密模式；
写入明文信息，然后将 valid 信号拉高再拉低即可触发一次编码操作；
当 sm4_enc_finish 信号为高时则一次编码完成，此时 RES_BUFFER 存储的解码值有效；

5.6 由编码模式切换为解密模式（且中途替换密钥）

首先将 CTL0 的 sm4_mode 比特位置位，切换至解密模式；
写入一次 Key 密钥数据；
将 valid 信号拉高再拉低触发一次密钥拓展操作（内部的密钥拓展暂存模块会存储密钥的拓展值）；
当 ex_finish 信号为高时则一次拓展密钥完成，之后若不替换密钥，则无需再进行上述操作了；
写入明文信息，然后将 valid 信号拉高再拉低即可触发一次编码操作；
当 sm4_enc_finish 信号为高时则一次编码完成，此时 RES_BUFFER 存储的解码值有效；

5.7 由解密模式切换为编码模式（且中途不替换密钥）

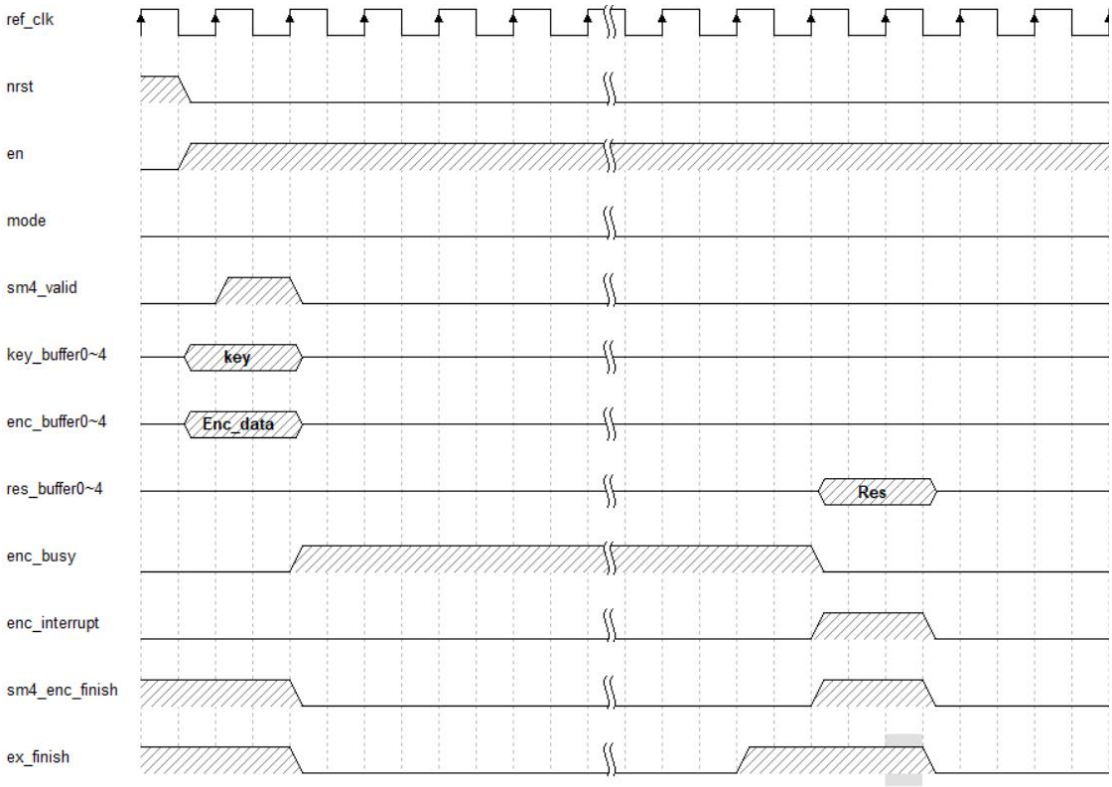
首先将 CTL0 的 sm4_mode 比特位复位，切换至编码（加密）模式；
写入明文信息，然后将 valid 信号拉高再拉低即可触发一次编码操作；
当 sm4_enc_finish 信号为高时则一次编码完成，此时 RES_BUFFER 存储的编码值有效；

5.8 由解密模式切换为编码模式（且中途替换密钥）

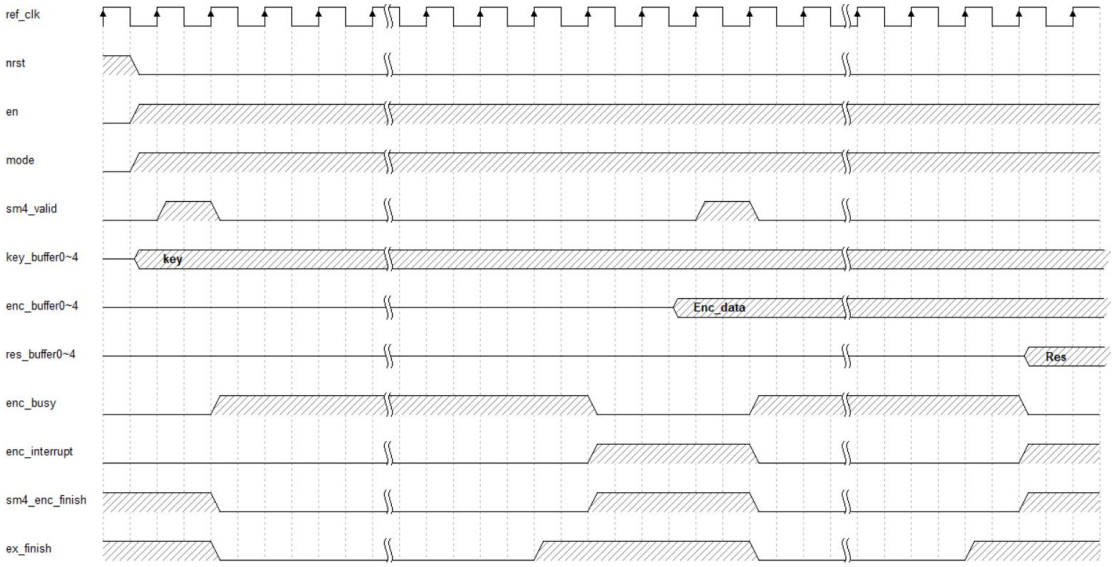
首先将 CTL0 的 sm4_mode 比特位复位，切换至编码（加密）模式；
写入一次 key 密钥数据；
之后写入明文信息，然后将 valid 信号拉高再拉低即可触发一次编码操作；
当 sm4_enc_finish 信号为高时则一次编码完成，此时 RES_BUFFER 存储的编码值有效；

六、信号时序图

6.1 编码模式



6.2 解码模式


ref_clk
nrst
en
mode
sm4_valid
key_buffer0~4
key
enc_buffer0~4
Enc_data
res_buffer0~4
Res
enc_busy
enc_interrupt
sm4_enc_finish
ex_finish

七、软件 API 接口

7.1 SM4_ACC_reset

SM4 IP 软件接口函数说明			
名称	SM4_ACC_reset	**参数**	Base_addr：SM4 IP 核基地址
返回值	无		
描述	SM4 IP 复位函数；		

7.2 SM4_ACC_reset

SM4 IP 软件接口函数说明			
名称	SM4_ACC_enable	**参数**	Base_addr：SM4 IP 核基地址
返回值	无		
描述	SM4 IP 使能函数；		

7.3 SM4_ACC_disable

SM4 IP 软件接口函数说明			
名称	SM4_ACC_disable	**参数**	Base_addr：SM4 IP 核基地址
返回值	无		
描述	SM4 IP 失能函数；		

7.4 SM4_ACC_SetAs_encoder

SM4 IP 软件接口函数说明			
名称	SM4_ACC_SetAs_encoder	**参数**	Base_addr：SM4 IP 核基地址
返回值	无		
描述	将对应地址的 SM4 IP 模块配置为编码模式；		

7.5 SM4_ACC_SetAs_decoder

SM4 IP 软件接口函数说明			
名称	SM4_ACC_SetAs_decoder	**参数**	Base_addr：SM4 IP 核基地址
返回值	无		
描述	将对应地址的 SM4 IP 模块配置为解密模式；		

7.6 SM4_ACC_event_start

SM4 IP 软件接口函数说明			
名称	SM4_ACC_event_start	**参数**	Base_addr：SM4 IP 核基地址
返回值	无		
描述	开始一次加密或解密事件；		

7.7 SM4_ACC_write_plaintxt_byte

SM4 IP 软件接口函数说明			
名称	SM4_ACC_write_plaintxt_byte	**参数**	Base_addr：SM4 IP 核基地址 plaintxt_buffer: 明文缓冲区指针
返回值	无		
描述	将 plaintxt_buffer 所指向的缓冲区中的 16 字节（128bits）明文写入 SM4 IP 的明文缓冲区；		

7.8 SM4_ACC_read_plaintxt_byte

SM4 IP 软件接口函数说明			
名称	SM4_ACC_read_plaintxt_byte	**参数**	Base_addr：SM4 IP 核基地址 plaintxt_buffer: 明文缓冲区指针
返回值	无		
描述	将 SM4 IP 的明文缓冲区数据读出；		

7.9 SM4_ACC_write_key_byte

SM4 IP 软件接口函数说明			
名称	SM4_ACC_write_key_byte	**参数**	Base_addr：SM4 IP 核基地址 key_buffer: 密钥缓冲区地址
返回值	无		
描述	将 key_buffer 所指向的缓冲区中的 16 字节（128bits）密钥写入 SM4 IP 的明文缓冲区；		

7.10 SM4_ACC_read_key_byte

SM4 IP 软件接口函数说明			
名称	SM4_ACC_read_key_byte	**参数**	Base_addr：SM4 IP 核基地址 key_buffer:密钥缓冲区地址
返回值	无		
描述	将 SM4 IP 的密钥缓冲区数据读出；		

7.11 SM4_ACC_read_result_byte

SM4 IP 软件接口函数说明			
名称	SM4_ACC_read_result_byte	**参数**	Base_addr：SM4 IP 核基地址 result_buffer:加密/解密结果缓冲区地址
返回值	无		
描述	将加密/解密结果读出；		

7.12 SM4_ACC_encoder_txt

SM4 IP 软件接口函数说明			
名称	SM4_ACC_encoder_txt	**参数**	Base_addr：SM4 IP 核基地址 txt：加密/解密明文缓冲区 result：加密/解密结果缓冲区 lenth：缓冲区长度
返回值	无		
描述	将 txt 所指向的明文缓冲区中 lenth 长度的明文进行加密/解密操作，并将结果写入 result 缓冲区中；		

7.13 SM4_ACC_read_CTL0

SM4 IP 软件接口函数说明			
名称	SM4_ACC_read_CTL0	**参数**	Base_addr：SM4 IP 核基地址
返回值	8 位整型		
描述	将 CTL0 寄存器中低四位内容读出，并以 8 位整型返回；		

7.14 SM4_ACC_read_IFG0

SM4 IP 软件接口函数说明			
名称	SM4_ACC_read_IFG0	**参数**	Base_addr：SM4 IP 核基地址
返回值	8 位整型		
描述	将 IFG0 寄存器中低四位内容读出，并以 8 位整型返回；		

7.15 SM4_ACC_check_busy

SM4 IP 软件接口函数说明			
名称	SM4_ACC_check_busy	**参数**	Base_addr：SM4 IP 核基地址
返回值	8 位整型		
描述	检测 SM4 IP 是否繁忙，若繁忙则返回 1，反之返回 0；		

7.16 SM4_ACC_check_encoder_finish

SM4 IP 软件接口函数说明			
名称	SM4_ACC_check_encoder_finish	**参数**	Base_addr：SM4 IP 核基地址
返回值	8 位整型		
描述	检测 SM4 IP 加解密过程是否完成，若完成则返回 1，反之返回 0；		

7.17 SM4_ACC_check_extend_finish

SM4 IP 软件接口函数说明			
名称	SM4_ACC_check_extend_finish	**参数**	Base_addr：SM4 IP 核基地址
返回值	8 位整型		
描述	检测 SM4 IP 密钥拓展过程是否完成，若完成则返回 1，反之返回 0；		

八、应用示例

```
/*
 * main.c
 *
 *  Created on: 2022 年 9 月 28 日
 *      Author: Ge Wen Jie
 *  Describe:SM4 IP test project
 *  global parameter:
 *  txt: the plain text to encoder
 *  sm4_key_buffer: sm4 key input
 *  enc_result: sm4 encoder result buffer
 *  dec_result: sm4 decoder result buffer
 */

#include "../user/headfile.h"


#define SM4_ENCODER_DATA_BUFFER_MAXLEN 128
#define SM4_ENCODER_RESULT_BUFFER_MAXLEN (SM4_ENCODER_DATA_BUFFER_MAXLEN)

uint32 sm4_key_buffer[16] = {0x11,0x11,0x11,0x11,0x22,0x22,0x22,0x22,0x33,0x33,0x33,0x33,0x44,0x44,0x44,0x44};
char txt[SM4_ENCODER_DATA_BUFFER_MAXLEN] = "nihaonihaonihaonihaonihaonihaonihaonihaogoodwayhouse";
char enc_result[SM4_ENCODER_RESULT_BUFFER_MAXLEN]={0x00};
char dec_result[SM4_ENCODER_DATA_BUFFER_MAXLEN]={0x00};


void print_sm4_buffer(uint32 * buffer)
{
int i;
for(i=0;i<4;i++)
{
xil_printf("%x,",*(buffer+i));
}
xil_printf("\n");
}

int main(){

xil_printf("\n------------------------SM4 Test Start-------------------------\n");
```

```
int i;
xil_printf("\nencoder Key(Hex):");
for(i=0;i<16;i++)
{
xil_printf("%x,",sm4_key_buffer[i]);
}
xil_printf("\nencoder txt(Chr):%s\n",txt);
SM4_ACC_reset(SM4_ACC_0);
SM4_ACC_enable(SM4_ACC_0);
SM4_ACC_SetAs_encoder(SM4_ACC_0);
SM4_Complie_plaintxt(txt);
SM4_ACC_write_key_byte(SM4_ACC_0,sm4_key_buffer);
SM4_ACC_encoder_txt(SM4_ACC_0,txt,enc_result,SM4_ENCODER_DATA_BUFFER_MAXLEN);
xil_printf("encoder result(Hex):\n");
for(i=0;i<sizeof(enc_result);i++)
{
xil_printf("%x,",enc_result[i]);
if((i!=0)&&(i%16 == 0))
{
xil_printf("\n");
}
}
SM4_ACC_SetAs_decoder(SM4_ACC_0);
SM4_ACC_encoder_txt(SM4_ACC_0,enc_result,dec_result,SM4_ENCODER_DATA_BUFFER_MAXL
EN);
xil_printf("\n");
xil_printf("decoder txt(Chr):");
xil_printf(dec_result);

xil_printf("\n------------------------SM4 Test End-------------------------\n:");
while(1);

return 0;
}
```