2020 年新工科联盟-Xilinx 暑期学校团队项目设计文档

作品名称	基于 SEA Board 的 AES 加密算法在 AWS 物联网中的应用
板卡型号	SEA 开发板——Xilinx xc7s15ftgb196-1 芯片
所在班级	A 班
成员姓名、学号、学校	杜娟　06017111　东南大学 李可欣　06017109　东南大学
Github 链接	https://github.com/KexinLi-design/Xlinx-SummerSchool-AWS_IoT-AES_Algorithm.git

第一部分

设计概述 /Design Introduction

1.1 设计目的

为了满足当下在信息安全领域的广泛需求，我们设计的是一个在物联网信息传输中的 AES 加密、解密装置。由 FPGA 的 ADC 串口输入待加密的外部信号（明文），通过拨码开关随时变更来调整加密密钥，同时我们利用 FPGA 的强大算力对明文和密钥进行 AES 算法，得到加密后的密文。然后，将加密后的数据通过 QSPI 串行通信接口发送到 ESP32，最后将 ESP32 读取的数据传输到 AWS 云端，实现外部信号的加密可视化。

数据加密的基本过程就是对原来为明文的文件或数据按某种算法进行处理，使其成为不可读的一段代码为“密文”，使其只能在输入相应的密钥之后才能显示出原容，通过这样的途径来达到保护数据不被非法人窃取、阅读的目的。加密过程的逆过程为解密，即将该编码信息转化为其原来数据的过程。我们希望应用所学的知识，利用 SEA Board 的 FPGA 和 ESP32 综合实现 AES 加密及解密过程。同时，该项目让我们加强了对信息加密算法的了解，同时增强了自己的实践能力。

1.2 应用领域及适用范围

该项目涉及领域的应用前景十分广泛，AES 算法的研究从理论到应用，已经深入到了信息安全技术的各个领域，深入研究与开发新的 AES 算法实现和应用具有重要的理论和实践意义。随着密码技术的高速发展，高级加密标准 AES(Rijndae1)算法将逐渐取代 DES 在 IPSec、SSL 和 ATM 中的使用，并广泛应用于虚拟专用网、远程访问服务器(RAS)、SONET(同步光网络)、高速 ATM / Ethernet 路由器、卫星通信、移动通信、电子金融业务等领域。此外，网络保密系统 、财政保密、电子游戏保密等方面也将采用 AES 加密算法，将现有的关于 AES 研究成果与其他领域的相关技术与应用相结合，从应用的角度拓展数据加密技术，从而获得新的应用，是 AES 算法(Rijndea1)的发展方向。

AES 标准用来替代原先的 DES（Data Encryption Standard），已经被多方分析且广为全世界所使用。经过五年的甄选流程，高级加密标准由美国国家标准与技术研究院 （NIST）于 2001 年 11 月 26 日发布于 FIPS PUB 197，并在 2002 年 5 月 26 日成为有效的标准。2006 年，高级加密标准已然成为对称密钥加密中最流行的算法之一。高级加密标准算法从很多方面解决了令人担忧的问题。

该设计适用于对文本信息、数字信号等进行加密，从电脑上输入一段信息，再人为设定一个秘钥，就会得到输出的加密信息；如果得知秘钥和加密后的信息，也可以对加密信息进行解密。

1.3 设计主要功能及特色

本设计充分考虑和综合了 FPGA 和 ESP32 各自的优点，实现的主要功能和创新点如下，

（1）利用 FPGA 强大算力实现外部信号的采集和加密、解密算法的计算，采用 ESP32 板载 WiFi 模块实现 AWS IoT 功能；

（2）该设计不仅完成了基于 FPGA 的 AES 算法验证，而且将 AES 算法充分使用在物联网信息传输加密、解密过程中，很好地迎合了当下人们的需求；

（3）在 AWS 平台上搭建通讯环境实现物联网，使得该项目与主流物联网平台紧密结合。

1.4 成员贡献

小组成员	主要分工	贡献百分比
杜娟	完成 FPGA 解密程序，书写 FPGA、AES 算法部分报告，综合大报告	50%
李可欣	完成 FPGA 加密程序，完成 Ardiuno IDE 中 AWS IoT 平台的搭建，书写 ESP32 部分报告，修改报告，github 上传整理	50%

1.5 展示照片

硬件系统如下图 1.1 所示，由 SEA Board 和 JTAG 下载器组成。

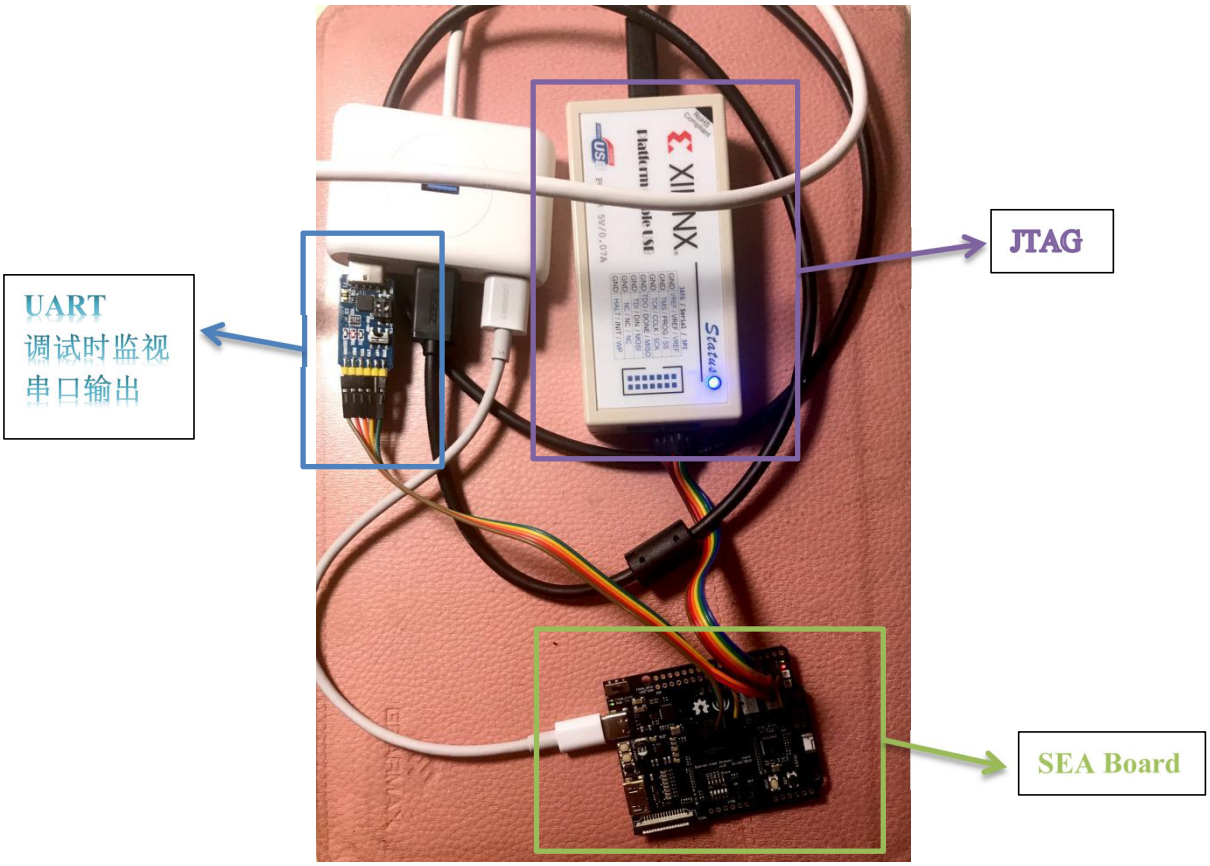


图 1.1 硬件平台的搭建

AWS IoT 平台的搭建如下图 1.2 所示，展示了加密全过程，从通过 FPGA 输入明文、密钥，到在 AWS 上显示密文。

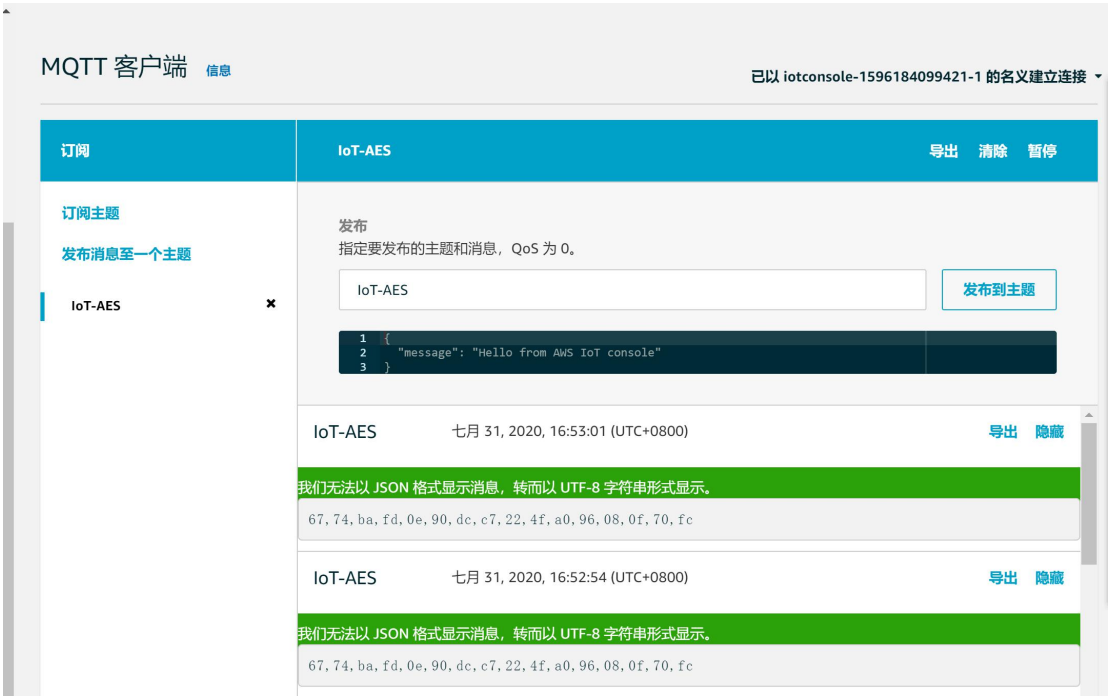


图 1.2 AWS IoT 平台的搭建

第二部分

系统组成及功能说明 /System Construction & Function Description

2. 加密模块

首先，我们设计基于 FPGA 的加密模块，模块示意图如下图 2.1 所示，主要模块有 key_128test、light_Sensor、aes_cipher_top、QSPI_slave 等，最终进行数据的云传输。最终，我们全部完成了下述框图内全过程，按计划完美完成任务。

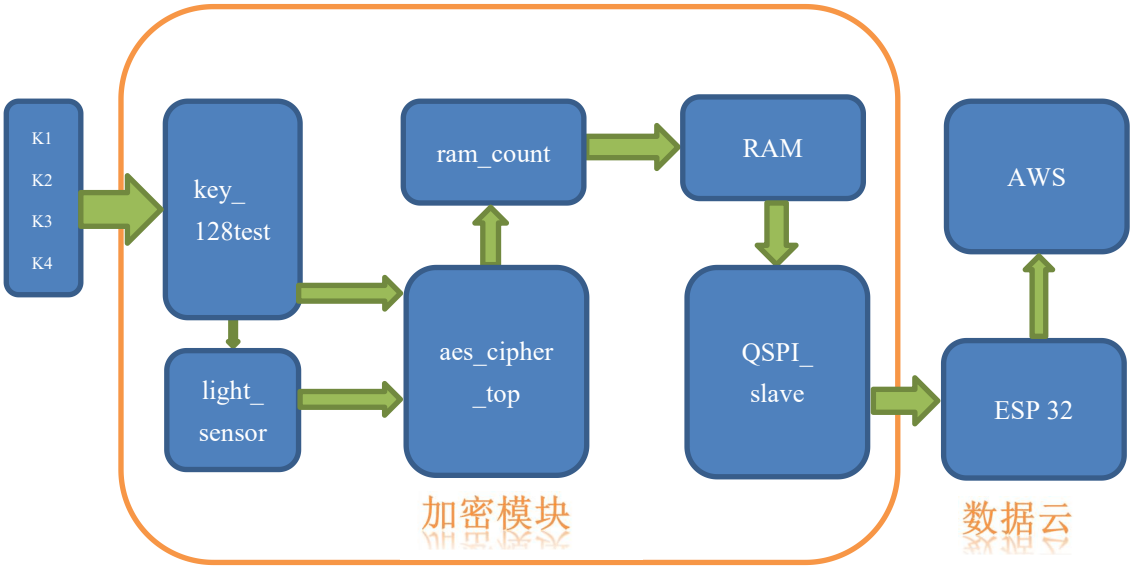


图 2.1 加密模块示意图

该系统组成如下所述：

（1）4 位秘钥由 4 个开关（K_1,K_2,K_3,K_4）人为输入,复位键低电平复位后，key_128test 模块开始加载 4 位秘钥并转化成 128 位的 AES 算法秘钥，秘钥转化完成后同时输出使能信号和 128 位算法秘钥；

（2）秘钥使能后 light_sensor 开始加载要加密的数据，并将 8bit 数据扩展成为 128bit 的 AES 算法数据，数据扩展完成后同时将 128bit 算法数据和 Id 使能信号输出到 AES 算法模块；

（3）aes_cipher_top 算法模块收到 Id 使能信号后，根据此时的 128bit 秘钥将 128bit 数据转化为 128 位 AES 加密数据，并将数据存入 ram_count 中；ram_count 将 128bit 数据拆分成每组 8bit 的数据，并以此存储到 RAM 中；

（4）存储到 RAM 中的起始地址为 8'b0000_0010，由于数据一共 16 组所以 RAM 的尾地址为 8'b0001_0001；

（5）最后通过 QSPI_slave 模块和外部 ESP32 进行数据交互，交互的通信协议为 QSPI；

（6）ESP32 再和 AWS 连接，最后加密后的信息传到 AWS 云上供用户读取。

2.1 key_128 模块

key_128test 模块是根据拨码开关转化秘钥的模块，拨码开关值为 1 则转化为 ffff_ffff，拨码开关为 0 则转化为 0000_0000。例如：K4_K3_K2_K1 = 1100，则转换为 128 位 ffff_ffff_ffff_ffff_0000_0000_0000_0000。该模块示意图如图 2.2 所示。

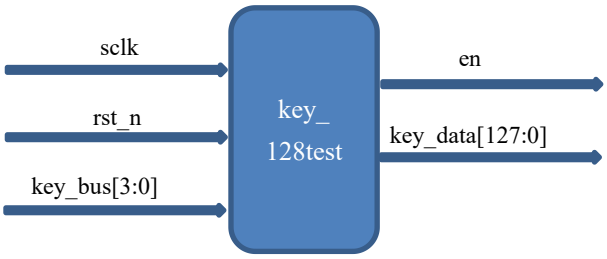


图 2.2 key_128test 模块示意图

其中，该模块各输入、输出说明如下，rst_n：复位开关；sclk：系统时钟；en：拨码开关数据转化完成后的输出使能端；key_bus：4bit 拨码开关数据输入端口；key_data：128bit 秘钥数据输出端口。

2.2 light_sensor 模块

light_sensor 模块在复位后进入等待状态，当 KEY_DONE 使能后表示秘钥已经加载完毕，这时候开始输出 AD 扩展数据。AD 模块在复位键复位后就开始读取 AD 数据，但是 AD 数据不进行扩展输出，在接收到 KEY_DONE = 1 后开始输出 128bit 传感器数据（由于硬件条件的限制，我们没有 ADC 模块，因此将需要加密的明文写在代码中）。在这个模块内部包含了一个 PLL 的 IP 核，这个核输出了整个程序所有需要的时钟，包括一路 15MHz 时钟、一路 15MHz + 90°相移时钟、一路 50MHz 时钟和一路 25MHz 时钟。并且内部还包含一个简单的判断模块，如果模拟量输入值大于 8'b0001_1111,则控制 led 灯使能点亮 led，小于阈值则 led 熄灭，这样也可以从外部判断我们的 AD 设备是否正常运行。模块数据扩展完毕后，同时输出 128 位传感器扩展程序和使能 data128_en 端。该模块示意图如图 2.3 所示。

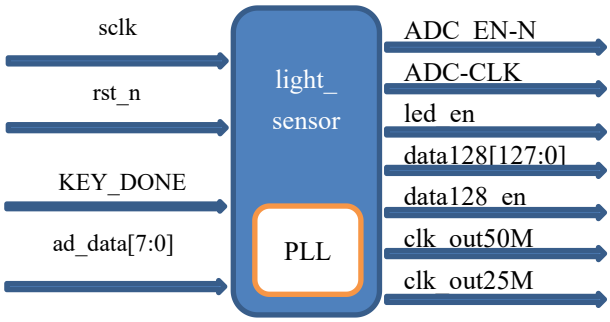


图 2.3 light_sensor 模块示意图

其中，该模块各输入、输出说明如下，

rst_n：复位开关；sclk：系统时钟；en：拨码开关数据转化完成后的输出使能端；KEY_DONE：秘钥使能输入端口，也是 light_sensor 模块的使能控制端，高电平输入和开始输出 AD 加密数据；ad_data：ADC_8bit 数据输入端口；

ADC_EN_N：ADC 数据输出使能控制端口;ADC_CLK：ADC 采样时钟，采用 datasheet 推荐的 15MHz 时钟；clk_out50M、clk_out15M：PLL 输出的 50MHz、25MHz 时钟信号;led_en：LED 灯使能控制信号；data128_en：128 位 ADC 加密数据输出使能；data128[127:0]：128 位 ADC 加密数据输出端。

2.3 aes_cipher_top 模块

前向密码块可以在 12 个时钟周期内执行完整的加密序列(10 轮为 10 个周期，加上一个周期用于初始密钥扩展，一个周期用于输出阶段)。aes_cipher_top 模块的时序示意图如图 2.4 所示。

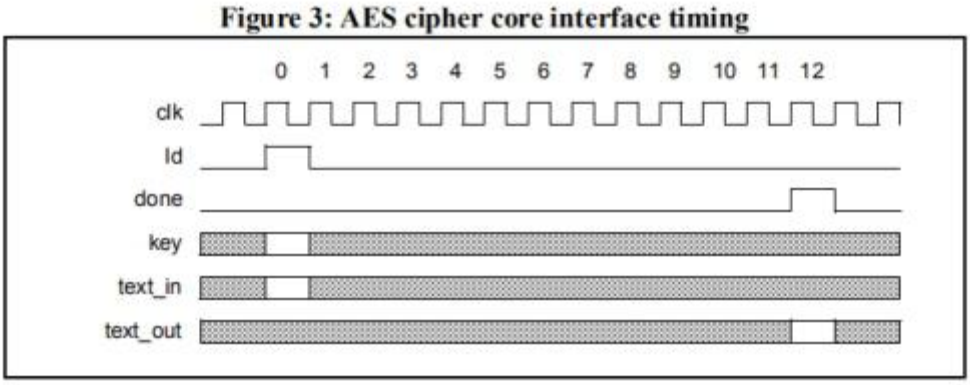


图 2.4 aes_cipher_top 模块时序图

前向密码块在每个加密序列的开头接受密钥和纯文本,开始总是通过断言' ld '引脚高来表示。当核心完成加密序列时，它将断言一个时钟周期内的“done”信号，以表示完成。用户可能选择忽略“done”输出和外部加密序列完成的时间，如图 2.5 所示。具体算法实现方式见第 4 小节。

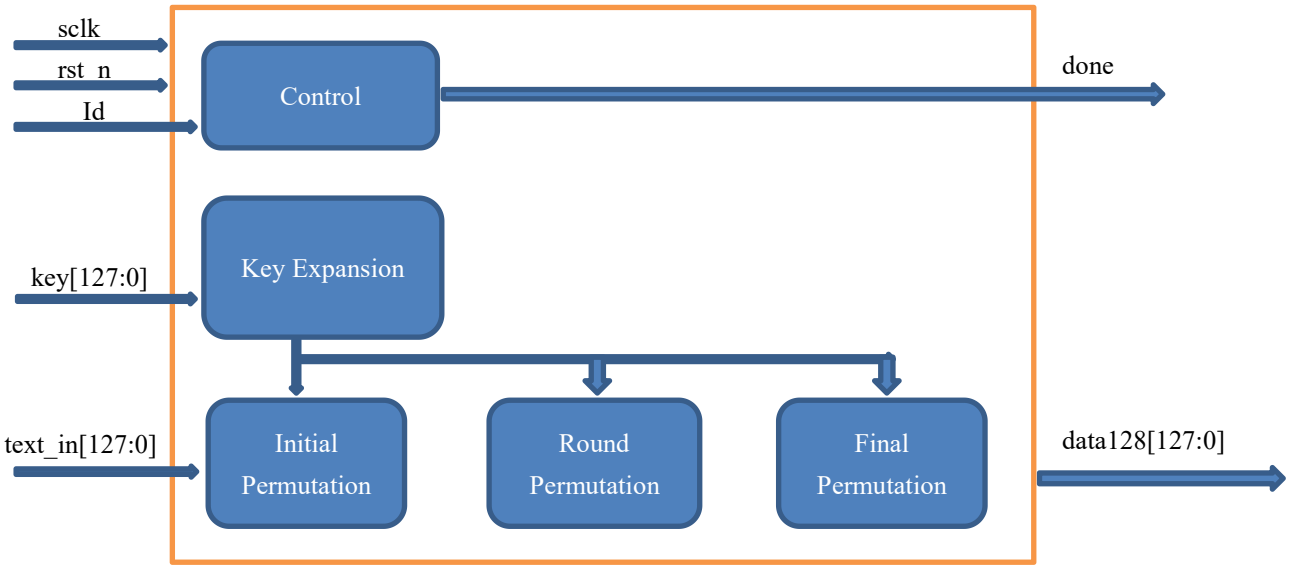


图 2.5 aes_cipher_top 模块示意图

2.6 ram_dount 模块

ram_dount 模块的主要功能是将 128bit 数据拆分成 16 组 8bit 的数据，然后依次存入 RAM 中，模块在 aes_done 拉高后，将此时的 ram_128test_in 数据存入到模块内部的寄存器中，然后顺序读出；后半部分控制相应的 ram 写使能和地址数据，模块示意图如图 2.6 所示。

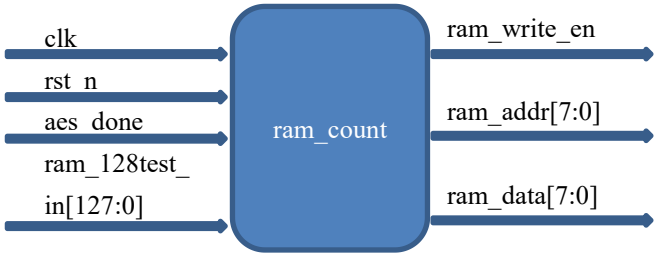


图 2.6 ram_dount 模块示意图

其中，该模块各输入、输出说明如下，

rst_n：复位开关；sclk：系统时钟；aes_done：表示 aes 算法数据处理完毕，已输出加密后的数据，此时模块开始运行；ram_128test_in：经过 aes 加密后的数据输入端口；

ram_write_en：RAM 写使能；ram_addr：RAM 地址输出端口；ram_data：RAM 数据输出端口。

2.7 QSPI_slave 模块

QSPI_slave 模块主要是标准的 QSPI 通信协议，端口直连 FPGA 内部的 RAM，通过 QSPI 发送相应的地址和读使能就能直接读取 RAM 中的数据，模块示意图如图 2.7 所示。

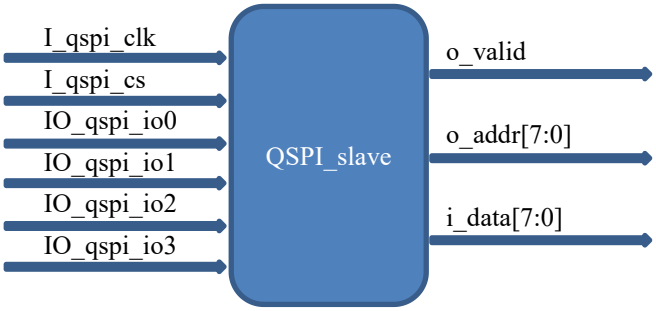


图 2.7 QSPI_slave 模块示意图

其中，该模块各输入、输出说明如下，

o_data：RAM 数据输入端口，本设计中未使用；o_valid：RAM 读使能控制信号；o_addr：RAM 读地址控制信号；i_data：RAM 数据输出端口，本模块在这个端口读取 RAM 内部的读数据。

2.8 ESP32 与 AWS IoT 加密互动模块

ESP32 是一款比较便宜、便携的微控制器开发板，可同时提供 WiFi 和蓝牙连接能力。开发板核心模块 ESP32 Module 可直接用于手持设备、可穿戴设备和众多物联网 IoT 设备的应用。在该项目中，我们使用 ESP32 作为终端设备，通过 QSPI 模块连接 FPGA 读取经过 AES 算法加密后的密码，并将该信息通过 HTTP 和 WiFi 网络以 MQTT Publish 发送给 AWS IoT Core 服务。具体的实现流程图 2.8 如下:

图 2.8 互动模块模块示意图

2.8.1 ESP32_AWS_aes 模块

ESP32_AWS_aes 模块主要是通过 Ardiuno IDE 编程，通过 ESP 开发使用的 WiFi 库和 AWS-IoT 库来连接个人 AWS IoT Core，从而实现物联网加密传输。端口直连 ESP32 密码输出，通过 ESP32_AWS_aes 模块发送相应的地址 8’b0000_0010 ~8’b0001_0001 的数据给 MQTT 测试主题，进行密码的 Subscibe 订阅。ESP32_AWS_aes 模块示意图如下图 2.9 所示，

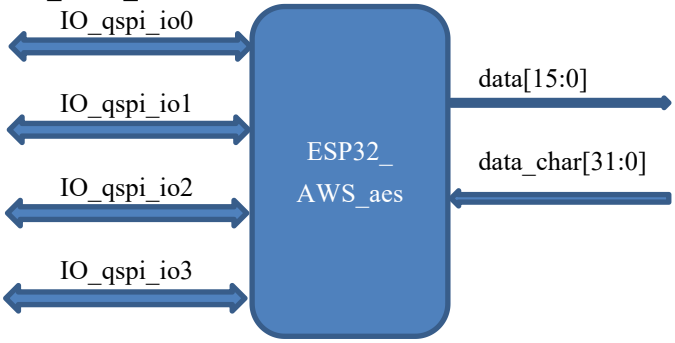


图 2.9 ESP32_AWS_aes 模块示意图

其中，data1[15:0]代表 AWS IoT 平台输出的加密后信息形式，data_char[31:0]代表向 ESP32 输入的待解密密码。

2.8.2 Ardiuno IDE 编程

（1）ESP32 库的加入

库的加载方式详见网页 https://www.arduino.cc/en/Guide/Libraries，在本案例中，我们加入 SEA-ESP32 编程所需的 esp32-hal-qspi 库和 sea_esp32_qspi 库，还有与 AWS 通讯所需的 WiFi 库和 AWS-IoT 库，加入库的情况可以由主程序头文件显示，如下图 2.10。

```
#include "sea_esp32_qspi.h"
#include "string.h"

#include "AWS_IOT.h"
#include "WiFi.h"
```

图 2.10 ESP32 库的加入

（2）引入 pubSubTest 示例修改

我们编程时采取在示例基础上进行修改的方法，点击文件->示例->AWS-IoT->pubSubTest 得到程序示例，如下图 2.11 所示。

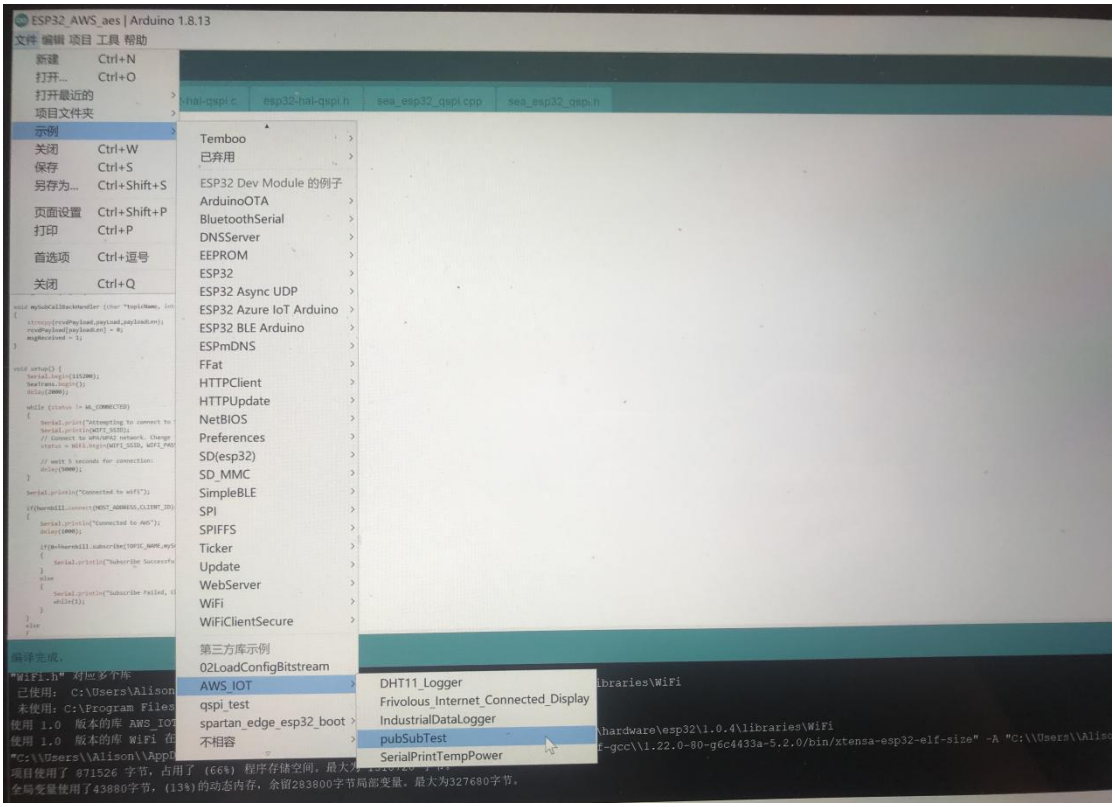


图 2.11 pubSubTest 示例的引入

在该示例中，我们加入将 IO_qsip_io 转化为 data1[16:0]形式密码输出的代码，并且将 data1[16:0]送给 AWS IoT 的 MQTT 测试，即读取 8’b0000_0010~8’b0001_0001 地址下的数据。

若 8’b0000_0010~8’b0001_0001 地址下依次存放 0x0f, 0x0c, 0x00, 0x0f, 0x08, 0x09, 0x06, 0x0a, 0x00, 0x04, 0x0f, 0x02, 0x02, 0x0c, 0x07, 0x0d, 0x0c, 0x09, 0x00, 0x00, 0x0e, 0x0f, 0x0d, 0x0b, 0x0a, 0x07, 0x04, 0x06, 0x07，则 data1[16:0]密码输出为 fc, 70, 0f, 08, 96, a0, 4f, 22 ,c7 ,dc ,90 ,0e ,fd ,ba ,74 ,67。因此，我们使用 C 语言中 sprintf(data_char,"%.2x”)函数来实现该功能。

2.8.3 AWS IoT Core 连接

（1）AWS IoT Core 平台创建物品、证书、策略

在 AWS 上创建物品的方式见 https://www.exploreembedded.com/wiki/Secure_IOT_with_AWS_and_Hornbill_ESP32 和 https://exploreembedded.com/wiki/AWS_IOT_with_Arduino_ESP32。在该项目中，我们创建物品，生成证书和策略分别如下图 2.12 所示。

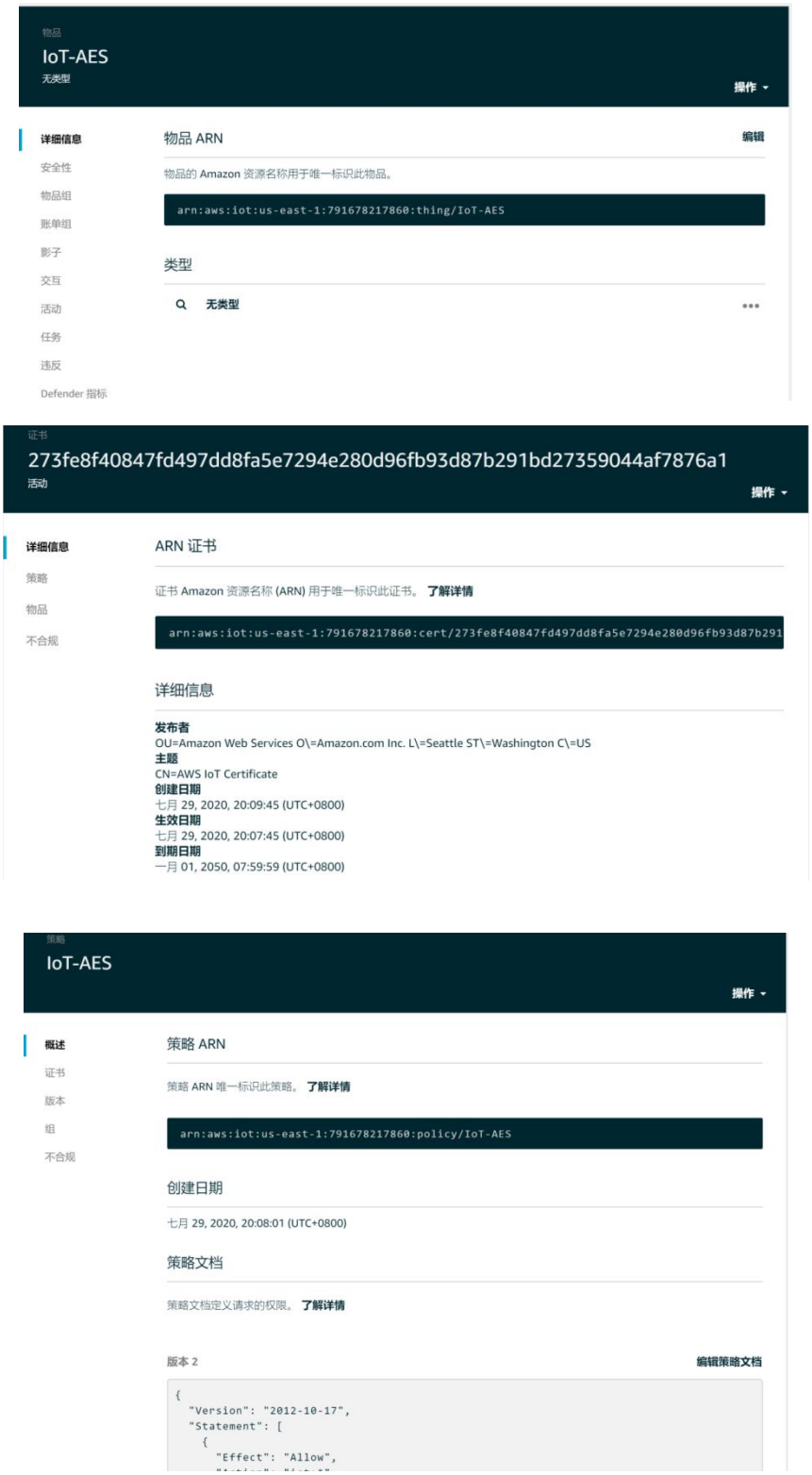


图 2.12 创建物品，生成证书和策略

（2）修改 Ardiuno IDE 程序库文件等

为了使 ESP32 和 AWS 平台实现互联，我们需要先将 ESP32 连接入本地的 WiFi，再将 AWS IoT 库中的 certification 改为我们所申请物品的相关信息。主要修改程序中如下图 2.13 中字段。

```
char WIFI_SSID[]="your Wifi SSID";
char WIFI_PASSWORD[]="Wifi Password";
char HOST_ADDRESS[]="AWS host address";
char CLIENT_ID[]= "client id";
```

图 2.13 修改 Ardiuno IDE 程序

其中，WIFI_SSID 和 WIFI_PASSWORD 与所在本地局域网 ID 与密码一致，HOST_ADDRESS 代表 AWS 平台上制定物品的 HTTP 交互节点（查看方式如下图 2.14 所示），CLIENT_ID 可以任意填写。

图 2.14 HTTP 交互节点查看方式

接着，需要打开 AWS-IoT 库中的 aws_iot_certficates.c 文件，将证书修改成 AWS 平台创建物品的证书信息，包括 aws_root_ca_pem[]，certificate_pem_crt[]和 private_pem_key[]。

将所有内容修改完毕后，点击编译上传，即可实现 ESP32 和 AWS 平台的物联网连接，连接情况可以再 Ardiuno IDE 的串口监视器中查看。

2.8.4 MQTT 实现 AWS IoT 数据收发（Publish & Subscribe）

在左侧栏目中选择测试，即可打开 MQTT 客户端，选择订阅主题，需要与程序中的 TOPIC_NAME 准确对应（程序中如 char TOPIC_NAME[]= "IoT-AES";），如下图 2.15 所示。

图 2.15 MQTT 客户端订阅主题的设定

在订阅主题中，能收发如下图 2.16 基本信息，则代表连接成功。

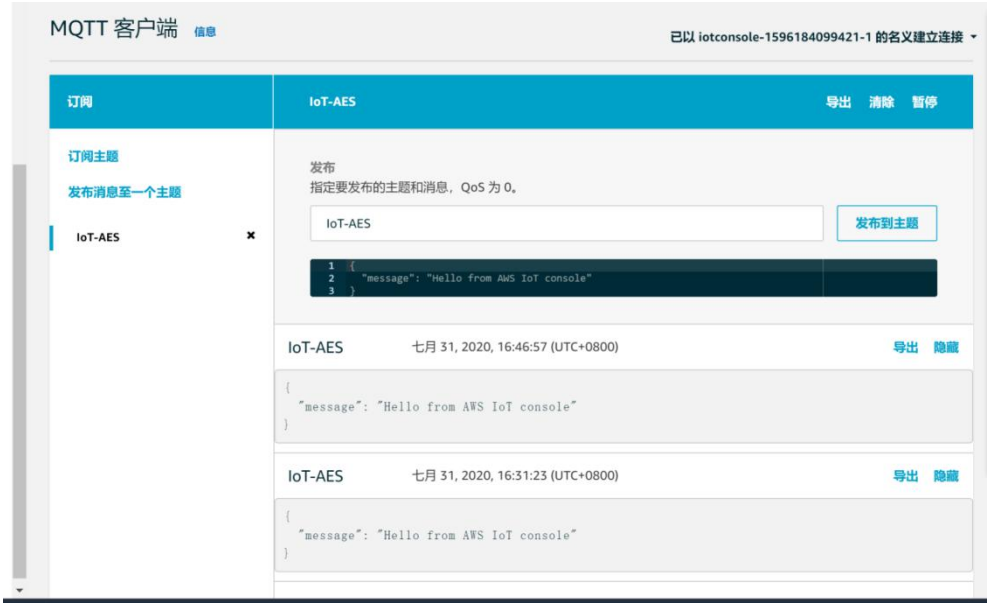


图 2.16 订阅主题连接成功

2.8.5 综合调试

在该项目中，综合调试分为如下步骤：

（1）FPGA 侧，编译 QSPI 从机顶层模块，分配好管脚以及电平，然后再次编译生成下载文件。

（2）将程序下载到 FPGA 上，上电后用串口工具进行调试。

（3）FPGA 侧调试成功后，在 Arduino 端对 ESP32 的 QSPI 主机进行配置和调试，QSPI 接口不变。

（4）从 MQTT 输出 ESP32 内接收相应地址的加密后密码。

2.9 加密模块 IO 接口

在 Vivado 中设置引脚如下图 2.17 所示。

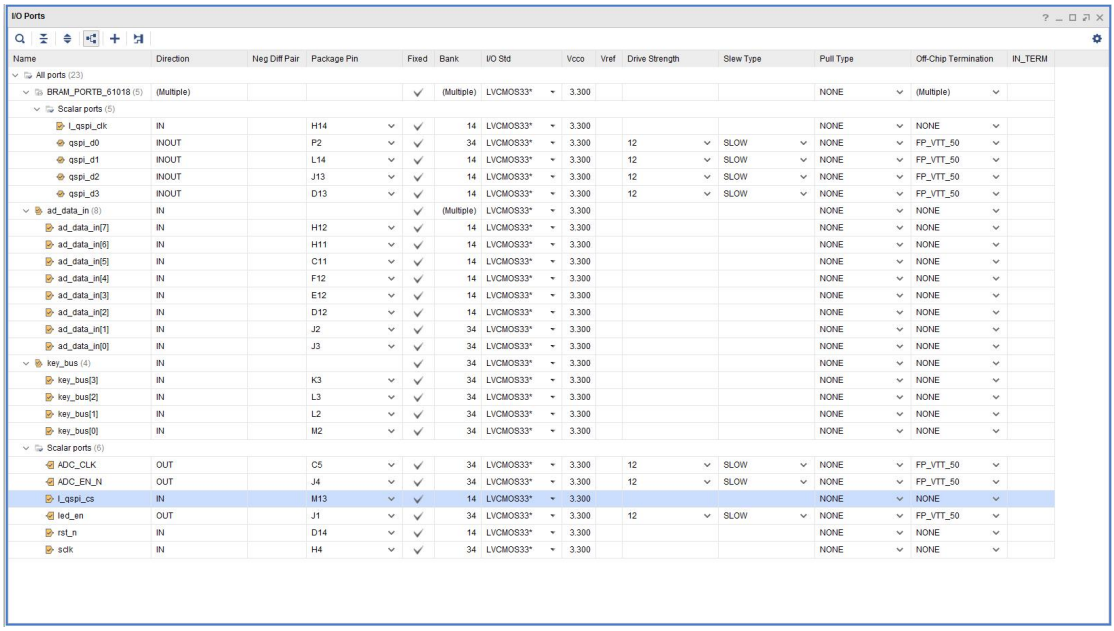


图 2.17 Vivado 引脚设置

3. 解密模块

我们可以通过 AWS 平台实现实时消息发送，将密码传给 ESP32 对设备进行远程控制，实现实时解密。再将密文从 ESP32 传送入 QSPI_slave 模块。

复位键低电平复位后，key_128test 模块开始加载 4 位秘钥并转化成 128 位的 AES 算法秘钥，秘钥转化完成后同时输出使能信号和 128 位算法秘钥；秘钥使能后 aes 解密模块（aes_inv_cipher_top）输出 kdone 信号标志密钥加载完毕，此时允许解密数据加载到 aes 解密模块。

用户在 AWS 上输入需要解密的信息，AWS 将其传输到 ESP32。然后，ESP32 通过 QSPI 通信协议，将 14*8bit 数据写入 RAM，ram_count 将 8bit 数据链接成 128bit 的 AES 解密数据，并在 kdone 使能后，使能 Id 信号和输出 128bit 解密数据到 aes 解密模块。

解密模块的示意图如下图 3.1 所示。

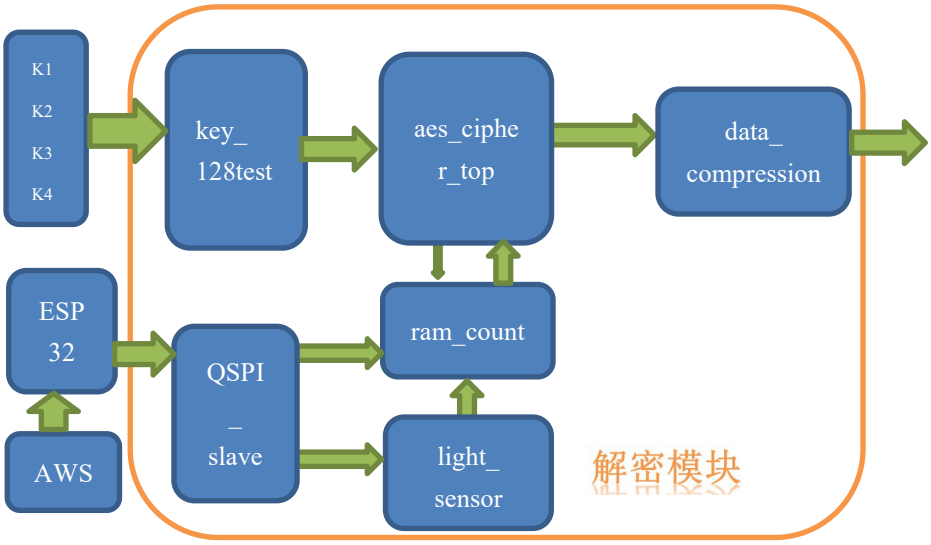


图 3.1 解密模块示意图

aes_inv _cipher_top 解密模块收到 Id 使能信号后，根据此时的 128bit 秘钥将 128bit 解密数据转化为 128 位原始扩展数据，并将数据输出到 data_ compression 模块中且使能 done 信号；

data_ compression 模块将 128bit 解密后的原始扩展数据，压缩为 8bit 的原始数据，并将其输出到外部 IO，同时拉高 DONE 输出使能标志信号和 en 控制信号；

LED_control 模块在接收到 en 信号后将 raw_data 读入到内部 men 中,并判断是否大于阈值,若大于则点亮 led 灯,小于则熄灭,此模块主要做测试使用.

在此，解密模块的功能框图和模块设计细节不再赘述，解密模块 IO 接口在 Vivado 中设置引脚如下图 3.2 所示。

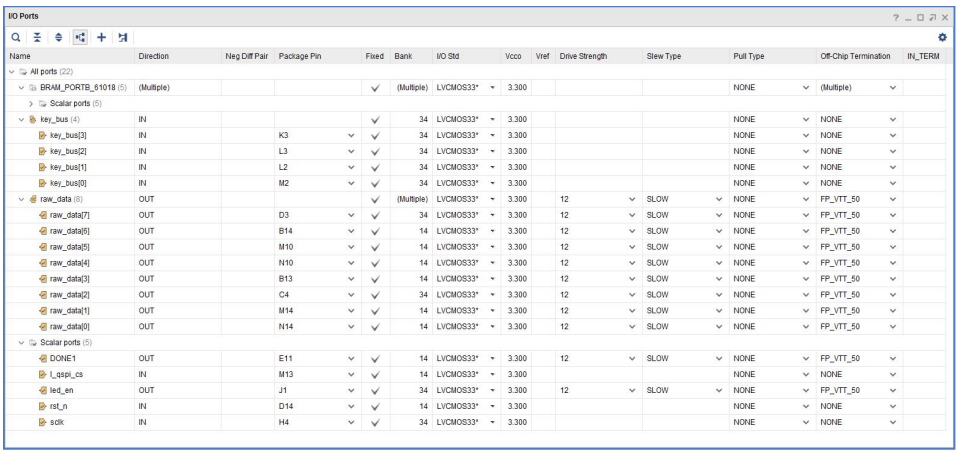


图 3.2 Vivado 引脚设置

4. AES 加密、解密算法

4.1 AES 算法简介

AES 加密数据块和密钥长度可以是 128b、192b、256b 中的任意一个。AES 加密有很多轮的重复和变换。大致步骤如下：

①密钥扩展（Key Expansion)；

②初始轮（InitialRound)；

③重复轮（Rounds），每一重复轮又包括字节间减法运算（SubBytes）、行移位（ShiftRows）、列混合（MixColumns)、轮密钥加法运算（AddRoundKey)等操作；

④最终轮（Final Round)，最终轮没有列混合操作（MixColumns)。

AES 密码与分组密码 Rijndael 基本上完全一致，Rijndael 分组大小和密钥大小都可以为 128 位、192 位和 256 位。然而 AES 只要求分组大小为 128 位，因此只有分组长度为 128Bit 的 Rijndael 才称为 AES 算法。本文只对分组大小 128 位，密钥长度也为 128 位的 Rijndael 算法进行分析。密钥长度为 192 位和 256 位的处理方式和 128 位的处理方式类似，只不过密钥长度每增加 64 位，算法的循环次数就增加 2 轮，128 位循环 10 轮、192 位循环 12 轮、256 位循环 14 轮。

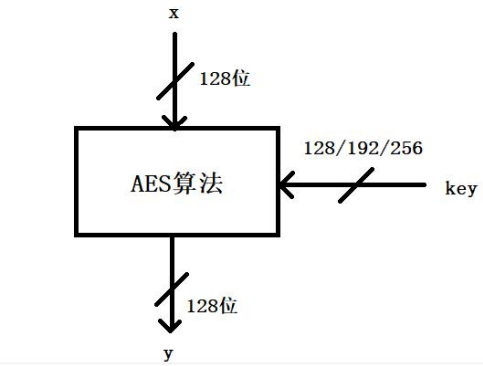


图 4.1 AES 算法示意图

AES 算法主要有四种操作处理，分别是密钥加法层(也叫轮密钥加，英文 Add Round Key)、字节代换层(SubByte)、行位移层(Shift Rows)、列混淆层(Mix Column)。而明文 x 和密钥 k 都是由 16 个字节组成的数据(当然密钥还支持 192 位和 256 位的长度，暂时不考虑)，它是按照字节的先后顺序从上到下、从左到右进行排列的。而加密出的密文读取顺序也是按照这个顺序读取的，相当于将数组还原成字符串的模样了，然后再解密的时候又是按照 4 • 4 数组处理的。AES 算法在处理的轮数上只有最后一轮操作与前面的轮处理上有些许不同(最后一轮只是少了列混淆处理)，在轮处理开始前还单独进行了一次轮密钥加的处理。在处理轮数上，我们只考虑 128 位密钥的 10 轮处理，AES 算法的处理流程如图 4.2 所示。

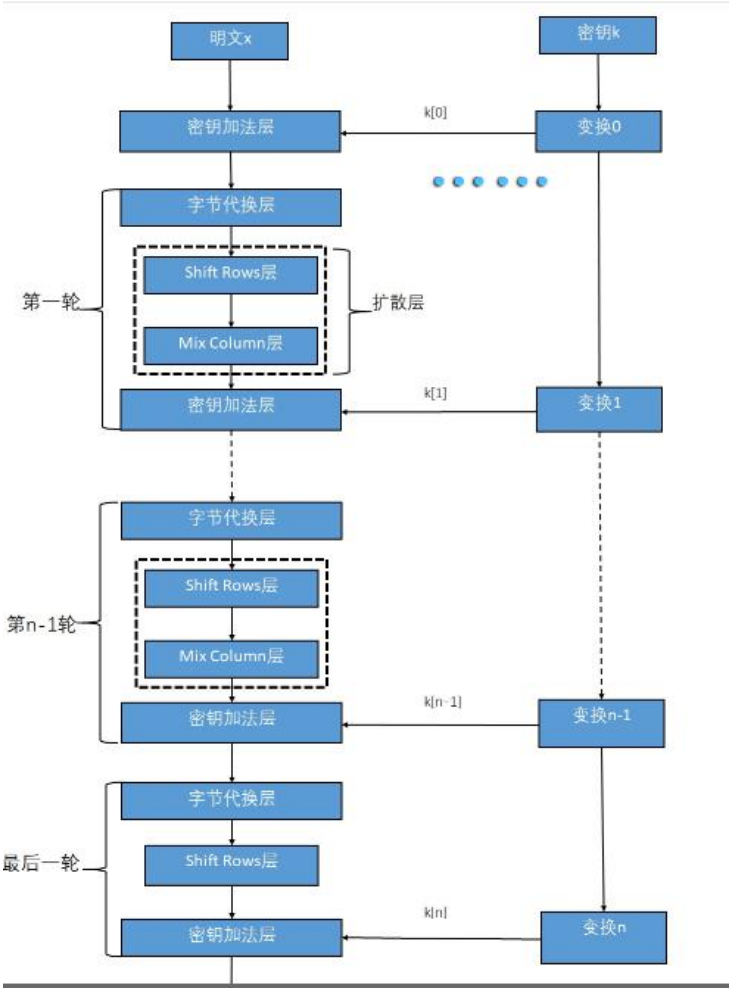


图 4.2 AES 算法的处理流程

4.2 秘钥加法层

在密钥加法层中有两个输入的参数，分别是明文和子密钥 k[0]，而且这两个输入都是 128 位的。k[0]实际上就等同于密钥 k，具体原因在密钥生成中进行介绍。我们前面在介绍扩展域加减法中提到过，在扩展域中加减法操作和异或运算等价，只需要将两个输入的数据进行按字节异或操作就会得到运算的结果，如图 4.3 所示。

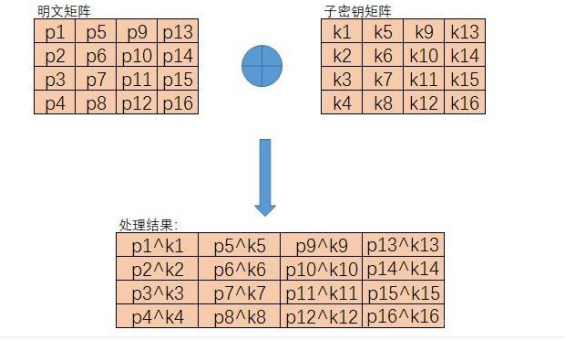


图 4.3 秘钥加法层

4.3 字节代换层

字节代换层的主要功能就是让输入的数据通过 S_box 表完成从一个字节到另一个字节的映射，这里的 S_box 表是通过某种方法计算出来的。S_box 表是一个拥有 256 个字节元素的数组，可以将其定义为

一维数组，也可以将其定义为 16 • 16 的二维数组，如果将其定义为二维数组，读取 S_box 数据的方法就是要将输入数据的每个字节的高四位作为第一个下标，低四位作为第二个下标。为了简化计算，我们将其视作一维数组即可。逆 S 盒与 S 盒对应，用于解密时对数据处理，我们对解密时的程序处理称作逆字节代换，只是使用的代换表盒加密时不同，如图 4.4 所示。

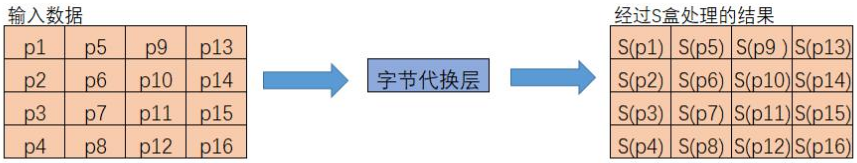


图 4.4 字节代换层

4.4 行位移

行位移操作最为简单，它是用来将输入数据作为一个 4 • 4 的字节矩阵进行处理的，然后将这个矩阵的字节进行位置上的置换。ShiftRows 子层属于 AES 手动的扩散层，目的是将单个位上的变换扩散到影响整个状态当，从而达到雪崩效应。在加密时行位移处理与解密时的处理相反，我们这里将解密时的处理称作逆行位移。它之所以称作行位移，是因为它只在 4 • 4 矩阵的行间进行操作，每行 4 字节的数据。在加密时，保持矩阵的第一行不变，第二行向左移动 8Bit(一个字节)、第三行向左移动 2 个字节、第四行向左移动 3 个字节。而在解密时恰恰相反，依然保持第一行不变，将第二行向右移动一个字节、第三行右移 2 个字节、第四行右移 3 个字节，到此操作结束。行位移算法如图 4.5 所示

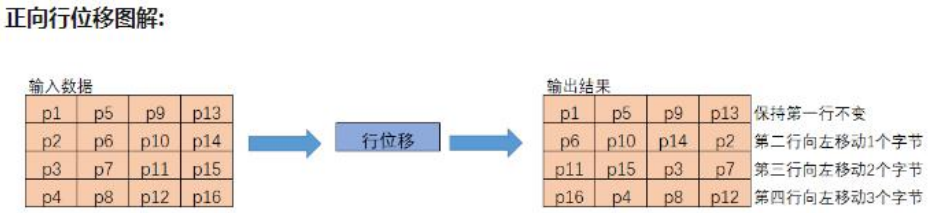


图 4.5 行位移

4.5 列混淆

列混淆子层是 AES 算法中最为复杂的部分，属于扩散层，列混淆操作是 AES 算法中主要的扩散元素，它混淆了输入矩阵的每一列，使输入的每个字节都会影响到 4 个输出字节。行位移子层和列混淆子层的组合使得经过三轮处理以后，矩阵的每个字节都依赖于 16 个明文字节成可能。其中包含了矩阵乘法、伽罗瓦域内加法和乘法的相关知识。

在加密的正向列混淆中，我们要将输入的 4 •4 矩阵左乘一个给定的 4 •4 矩阵。而它们之间的加法、乘法都在扩展域 GF(2^8)中进行。

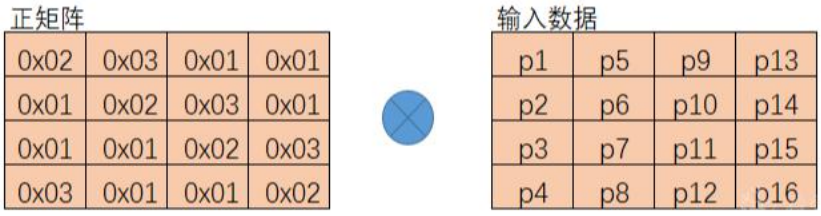


图 4.6 列混淆

4.6 子密钥的生成

子密钥的生成是以列为单位进行的，一列是 32Bit，四列组成子密钥共 128Bit。生成子密钥的数量比 AES 算法的轮数多一个，因为第一个密钥加法层进行密钥漂白时也需要子密钥。密钥漂白是指在 AES

的输入盒输出中都使用的子密钥的 XOR 加法。子密钥在图中都存储在 W[0]、W[1]、...、W[43]的扩展密钥数组之中。k1~k16 表示原始密钥对应的字节，而图中子密钥 k0 与原始子密钥相同。在生成的扩展密钥中 W 的下标如果是 4 的倍数时(从零开始)需要对异或的参数进行 G 函数处理。密钥的生成过程如图 4.7 所示。

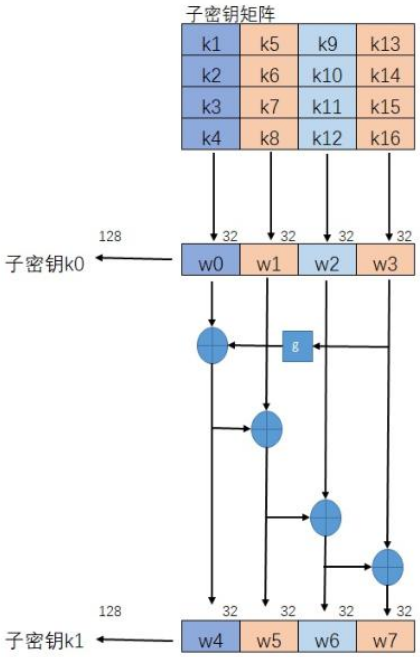


图 4.6 子密钥的生成

第三部分

完成情况及性能参数 /Final Design & Performance Parameters

5. 完成情况及性能参数

5.1 实现功能

实现了加密及解密代码的编写，仿真和下载验证。并通过 Vivado 编程，可以使用 SEA Board 对输入信息进行加密或解密。

（1）加密仿真

在加密部分，当输入 8 位明文 ad_data_in[7:0]=a5，转化为 128 位为 data128[127:0]=ffff0000ffff00000000ffff0000ffff，秘钥为 key[127:0]=ffff_ffff_ffff_ffff_ffff_ffff_ffff_ffff，得到加密后的密文应该为 fc70_0f08_96a0_4f22_c7dc_900e_fdba_7467。正确的密文参考答案由加密网页验证得到，如图 5.1 所示。

图 5.1 网页参考密文

在 Vivado 中进行加密模块仿真，得到的密文见 doutb[7:0]所示，从低位到高位依次读出密文为 fc70_0f08_96a0_4f22_c7dc_900e_fdba_7467，如图 5.2 所示。经验证，FPGA 中加密模块合理且正确。

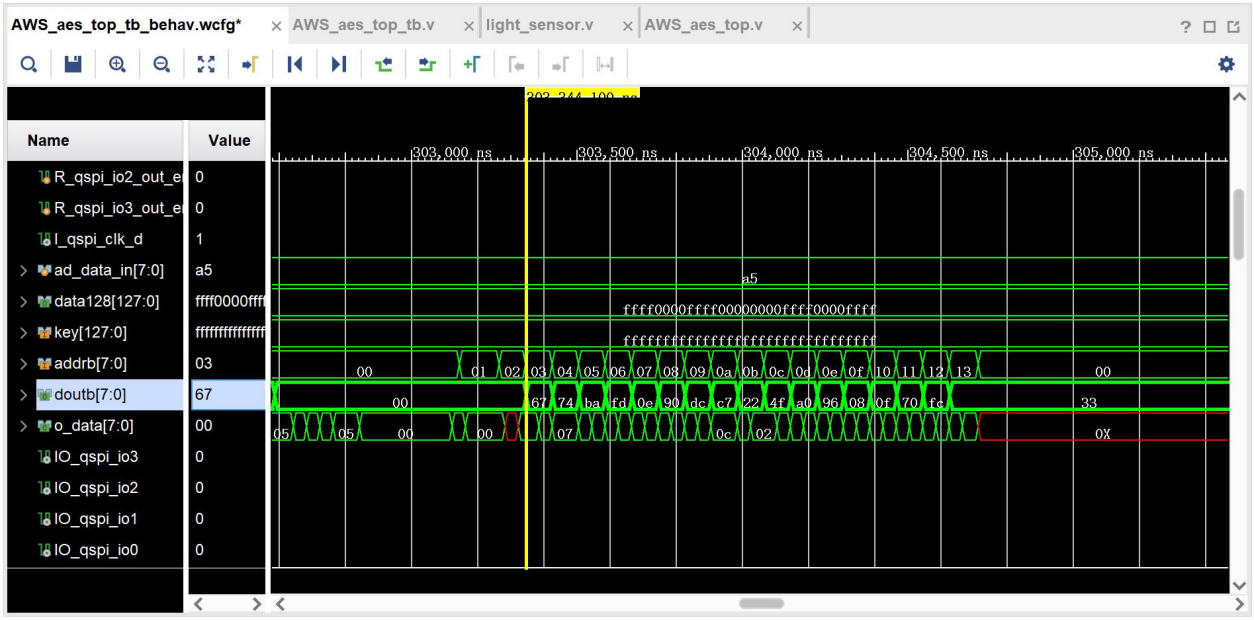


图 5.2 加密模块仿真

（2）解密仿真

在解密模块仿真，当输入密文为 fase_data[127:0]=c70_0f08_96a0_4f22_c7dc_900e_fdba_7467，得到对应的明文为 raw_data[7:0]=a5。经验证，FPGA 中解密模块合理且正确，如图 5.3 所示。

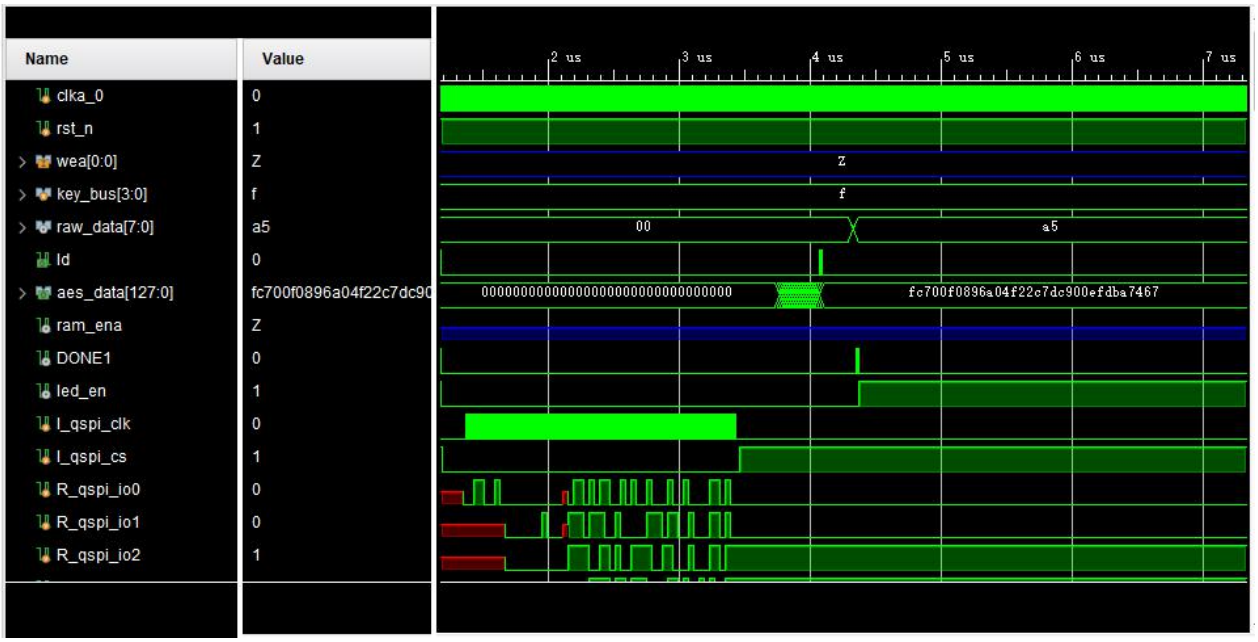


图 5.3 解密模块仿真

（3）实现 ESP32 与 Wifi 和 AWS 连接

从 Ardiuno IDE 的窗口监视器，我们可以发现 ESP32 与 Wifi 和 AWS 成功连接，如下图 5.4 所示。

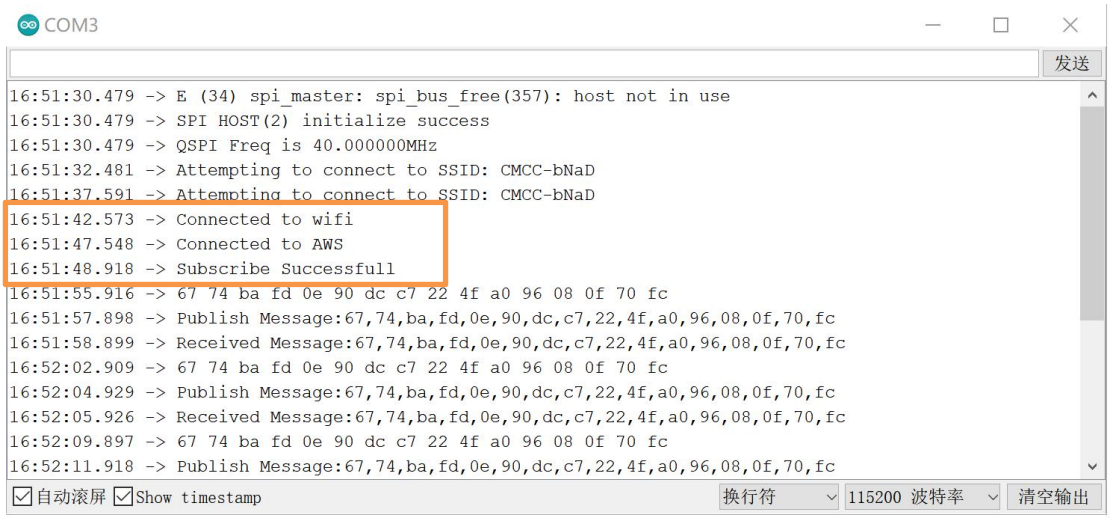


图 5.4 ESP32 与 Wifi 和 AWS 成功连接

（4）实现加密后密码输出

从客户端可以看出，当 FPGA 端输入明文 a5 和密钥 1111 后，AWS MQTT 客户端输出密码为 fc 70 0f 08 96 a0 4f 22 c7 dc 90 0e fd ba 74 67（如下图 5.5 所示），与预期结果相同，因此基本功能实现成功。

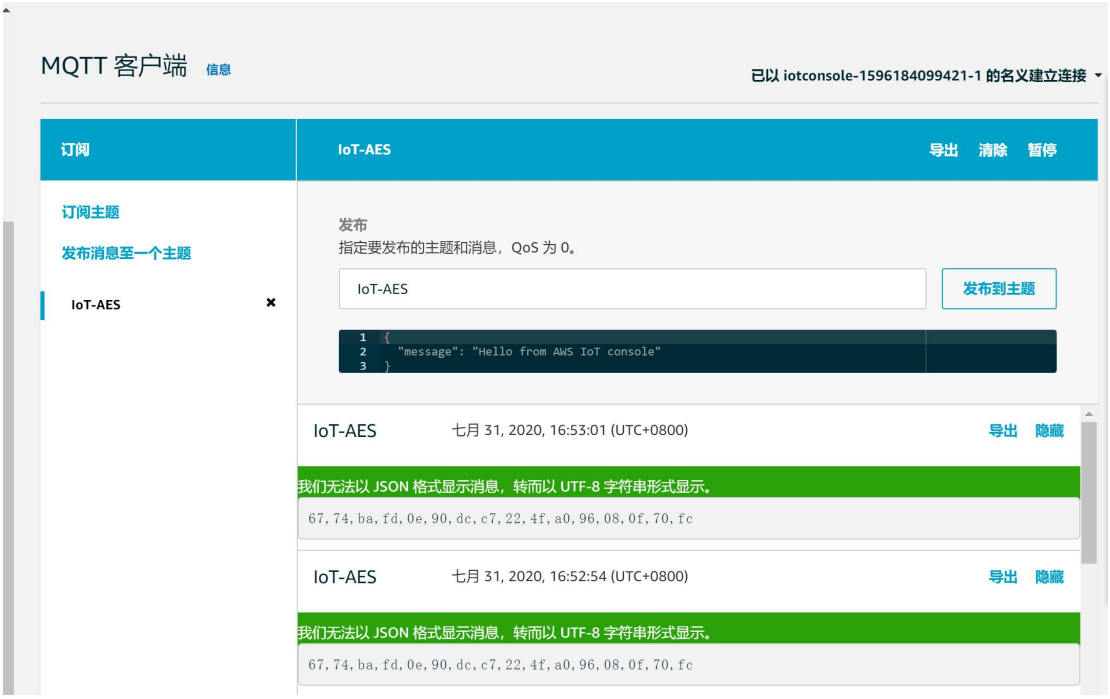
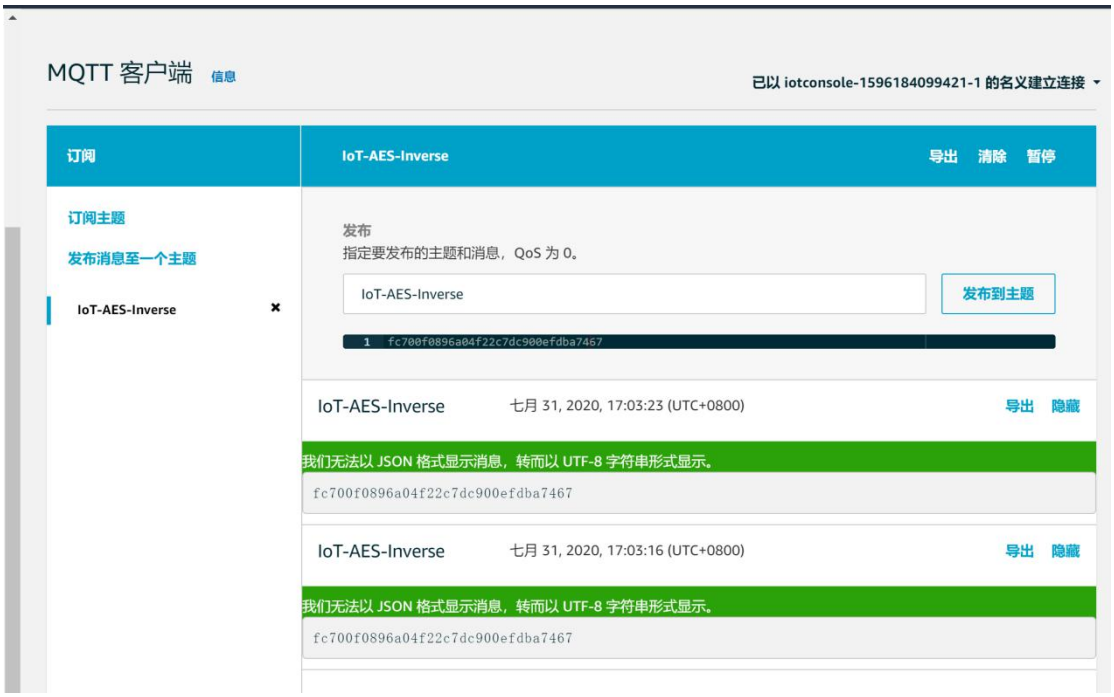


图 5.5 加密后密码输出

（5）实现解密密文 AWS 输入

我们可以实现通过 MQTT 向 ESP32 输入待解密的密码，如下图 5.6 所示。



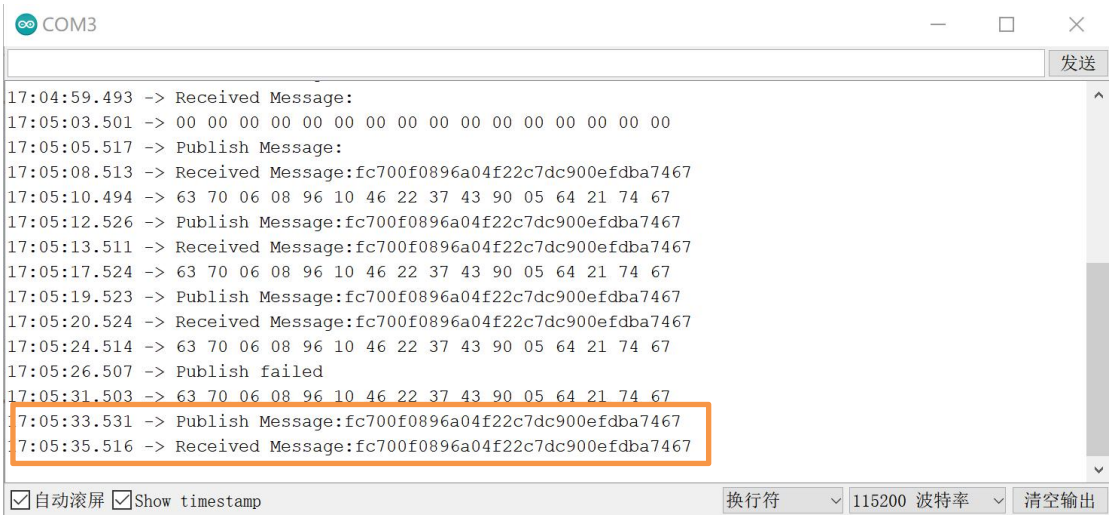


图 5.6 解密密文 AWS 输入

6. 性能参数

6.1 实时性

加密部分从输入待加密数据开始计时，到在 AWS IoT 平台读出数据，时间统计如下表 6.1 所示。

表 6.1 实时性统计表

进程	花费时间
FPGA bitstream 烧录 + Ardiuno 编译	1min 11s
Ardiuno 上传	50s
ESP32 连接成功后 MQTT 客户端收到加密信息	15s
更改需加密信息更新 MQTT 输出	5~10s

在 FPGA 和 ESP32 都完成烧录和连接后，真正的数据传输部分只需要 5~10s 左右，基本满足实时性的要求。因此，该系统是一个比较合格的物联网系统。解密为加密的逆过程，所需时间十分接近。

6.2 稳定性

经过反复试验，AWS 和 ESP32 的连接过程成功率超过 90%，若连接失败进行一次重连一般即可成功，由此证明系统稳定性较好。

6.3 创新性

我们基于 FPGA 的 AES 算法案例，拓展了其在物联网 AWS 平台上的应用，不仅完成了基于 FPGA 的 AES 算法验证，而且将 AES 算法充分使用在物联网信息传输加密、解密过程中，很好地迎合了当下人们的需求。并且该项目充分利用 FPGA 和 ESP32 两个平台各自的优点，实现功能的性能稳定、响应迅速。

第四部分

总结 /Conclusions

杜娟 06017111：

21 世纪，人们时刻身处信息的洪流之中，信息安全也成为极其重要的课题。为了保护人们的信息不被盗用和攻击，许多信息加密技术应运而生。在许多加密标准中，我们选择了 AES 加密标准来进行学习。因此，在此次的实习中，我们选择了 AES 算法进行物联网信息加密、解密作为最终的实习项目。

我们首先在网上查找 AES 算法和 AWS 物联网平台的相关资料，了解 AES 加密的基本概念与具体流程，学习了加密算法的相关知识，了解到了迦罗瓦域的乘法等数学知识。然后又找到了一些相关的范例进行学习，经过数次实验与纠错，最终成功编写完整的工程，并下载到 SEA Board 上进行实践。同时，我们还首次尝试将硬件与 AWS 云进行连接，通过 ESP32 接受由 FPGA 处理过的信息，再利用板载 WiFi 模块将数据发送至 AWS IoT Core 上。

我们在进行实验的过程中，也走过许多弯路，一开始使用 AWS 平台连接不成功，就尝试使用 upycraft，但也没有成功，然后又试图用 UART 进行通信，也没有成功，最终才通过 Arduino IDE 和 AWS 连接成功，可谓是一波三折。但是最终的结果没有让我们失望，我们的开发板成功的将输入信息进行加密和解密，并且经过 Matlab 程序和现有加密网站验证，结果非常准确，比较完美地完成了整个项目。

此次实习虽然时间很短，但是节奏很紧凑，每天的课程与作业让我们忙碌而充实，感谢 Xilinx 和学校提供给我们这样的学习和锻炼的机会，也感谢我们两人搭档互相之间一直以来的耐心、坚持与帮助，这次的学习合作因为有你而让人愉快。

李可欣 06017109：

我们组从选题、做题到最终完成，虽然中间历经坎坷，但结果是很美好的，如期按计划完成我们定题目时设想的所有功能，完成了一个完整的物联网加密系统。在最初的选题阶段，是我执意要选择这个看似冷门的项目。在互联网、大数据等等流行的今天，在这个倡导万物互联的时代，信息安全愈加凸显出其重要的地位。我选这道题的初衷在于，一方面我想结合加密算法、SEA Board 硬件平台和 AWS 物联网平台设计一款互联信息加密系统，这套系统可以推广到任何物联网设备上，并将发挥重要的作用；另一方面，我想利用在暑假学校中学到的知识和设计方法去实践我上一学期在计算机网络概论课程中学到的网络层、传输层和应用层知识。

在完成项目的过程中，我将项目分成 FPGA 和 ESP32 两部分。其中，我花了很多时间在 AES 算法的理解和书写，FPGA 程序的模块设计和物联网平台搭建上。由于我之前没有用 Ardiuno IDE 编程的经验，也没有基于 AWS 平台的开发经历，网上的开发资源也比较零散和混乱，物联网这部分的尝试对我来说极其困难。但在经过了三天的钻研和调试中，我最终克服了困难，成功将 FPGA，ESP32 和 AWS IoT 三者完美地串在了一起，它们各司其职，完成各自最擅长的部分。

我要特别感谢我的队长，我们合作过很多次，这次我们两也特别默契。由于她的 Ardiuno IDE ESP32 库下载有困难，所以在我独自完成物联网平台开发的过程中，她完善了 FPGA 加密、解密的过程设计，并且承担了大部分报告书写工作。

这次实习虽然只有半个月，但是我学到的是自己独立学习三个月可能都达不到的学习量，最终完成一个完整的项目也是我受益良多。这半个月说实话，真的感觉到非常辛苦，但是又很值得。希望我能带着在暑假学校中的收获，继续努力，充分利用在今后的学习和项目完成中！