# AWS-SHA256案例说明

## 摘要

本案例以SEA-Board（FPGA的型号为xc7s15ftgb196-1）为硬件开发平台，搭建一个具有数据采集、数据处理及数据通信功能的系统。

本案例使用SEA-Board板载陀螺仪模块进行数据的采集，通过FPGA中的SHA256模块进行数据处理产生256 bits哈希值，哈希值产生后上传至AWS-loT平台；本系统的核心为多平台的数据通信架构，该架构实现了FPGA + ESP32 + AWS-loT三平台的数据通信任务。

## 系统整体结构

本案例的核心是FPGA + ESP32 + AWS-loT三平台的数据通信架构，其中FPGA通过IIC读取板载陀螺仪模块采集的数据，FPGA和ESP32以QSPI总线进行数据通信，ESP32和AWS-loT平台以WiFi连接进行数据通信，通信方式及系统结构框图如图2-1所示。



**图2-1，系统结构框图**

本系统主要围绕FPGA和ESP32进行设计，本文将对FPGA和ESP32的关键模块功能及设计进行描述；关于板载陀螺仪的数据采集以及AWS-loT平台的配置可以参考官方手册或者教程。

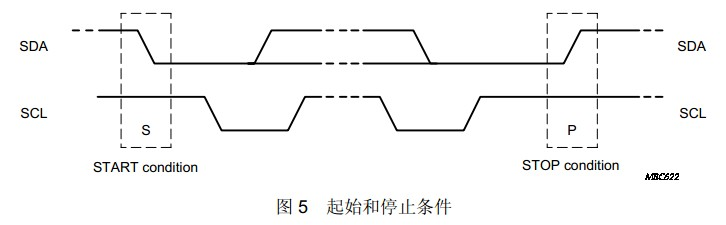
## FPGA侧各模块功能说明

本案例中， FPGA的主要功能有：

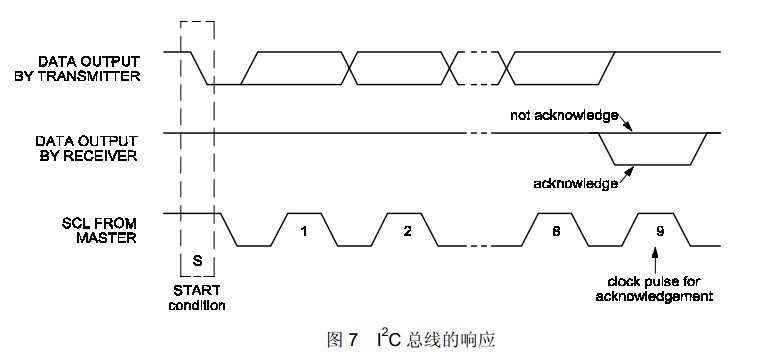
1. **陀螺仪模块数据的采集：**主要由IIC driver模块和Gyro driver模块共同实现，使用SEA-Board板载6轴传感器芯片LSM6D3TR进行数据采集，通过Gyro驱动模块完成传感器数据的读取；
2. **哈希值的产生：**由SHA256模块实现，将从传感器中读取的温度、姿态角数据共8Bytes数据，通过SHA256模块转化为256bits哈希值；
3. **QSPI从机的搭建：**由QSPI\_Slave模块实现，搭建QSPI从机，响应ESP32端QSPI主机的读写请求。

### IIC driver模块

IIC driver模块主要功能为：通过IIC总线，读取Gyro采集的数据，将数据以串转并，并将读取的传感器数据传递给Gyro driver模块作为模块数据的输入，IIC总线时序如图3-1和图3-2所示。



**图3-1，IIC起始和停止条件**



**图3-2，IIC总线响应**

### Gyro driver模块

Gyro driver模块主要功能为：通过IIC driver模块中采集的Gyro数据，解析为相关数据并将解析的传感器数据通过Gyro\_RAM传递给SHA256模块作为模块数据的输入，Gyro driver模块和SHA256模块的实际连接如图3-3所示。



**图3-3，Gyro driver模块和SHA256模块连接图**

该模块在数据采集使能为高电平时，通过IIC driver模块进行Gyro数据的读取，并将读取的数据以8bits的线宽并行写入到宽度同为8bits的RAM中进行数据的保存，Gyro\_ RAM的内存地址分布如图3-4所示。



**图3-4，**Gyro\_RAM**内存地址分布**

### SHA256模块



**图3-5，SHA256模块框图**

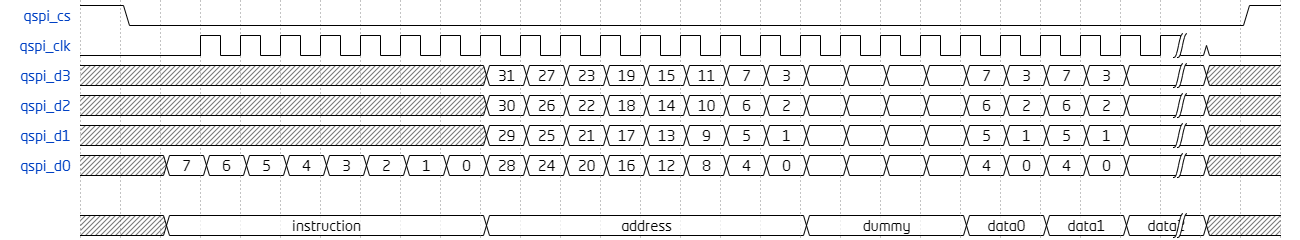
SHA256模块为FPGA设计中的核心数据处理模块，模块框图如图3-5所示；其主要功能为：读取Gyro\_RAM数据，将地址为00H ~ 07H的数据由低位至高位依次写入SHA256模块中，SHA256模块根据SHA256算法运算，将64 bits数据生成为256 bits哈希值，并将哈希值储存在与QSPI从机直接连接的RAM中，储存地址为10H ~ 2FH，内存数据处理如图3-6所示。



**图3-6，SHA256算法数据处理示意图**

### QSPI\_Slave模块

QSPI\_Slave模块主要功能为：接收并响应ESP32侧QSPI主机发送的指令及数据，通过QSPI通信协议进行RAM的读/写操作；在本案例中，主要响应ESP32侧QSPI主机发出的读数据指令，将RAM中地址为10H ~ 2FH的256 bits哈希值传输至ESP32中。关于QSPI\_Slave模块的使用，可以从SEA-Board相关手册中找到更详细的说明。QSPI通信时序如图3-7所示。



**图3-7，QSPI通信时序**

## ESP32侧功能说明

本案例中，ESP32的主要功能有：

1. **QSPI主机的搭建：**搭建QSPI主机，向FPGA中QSPI从机发送读取数据指令，从FPGA中读取256 bits哈希值；
2. **WiFi模块的配置：**连接ESP32至WiFi及AWS-loT平台，并将哈希值上传至AWS-loT平台。

### QSPI Master搭建

在ESP32开发平台中，添加QSPI主机库，并在主函数中进行QSPI主机的初始化；ESP32侧QSPI主机初始化完成后，可以调用QSPI库中读/写函数，向FPGA中的QSPI从机发送指令、地址、数据，获取256 bits哈希值，并以unsigned int形式存放在ESP32的数组中。

### WiFi模块的配置

调用AWS\_loT和WiFi库，将WiFi SSID和Password配置完成，通过串口工具助手反馈连接状态，连接完成后通过AWS loT平台相关证书及地址（详细参考AWS官方教程），连接至AWS loT平台，并通过串口助手确认连接状态；都连接成功时，ESSP32将会每5秒获取一次哈希值，并自动将哈希值上传至AWS loT平台上，可以通过AWS loT控制台中订阅相关主题，获取哈希值。

### ESP32程序流程

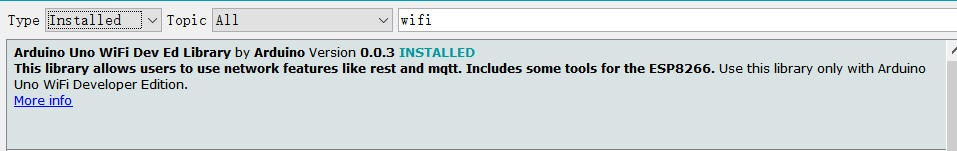


图4-1**，**ESP32 WiFi库

AWS-loT库：

<https://github.com/ExploreEmbedded/Hornbill-Examples/tree/master/arduino-esp32/AWS_IOT>

如图4-1和所示链接，在Arduino-IDE中添加AWS loT库和WiFi库，编译程序并下载至ESP32中。在AWS loT控制台中订阅相关设备的主题，获取实时上传的哈希值，程序流程如图4-2所示。



**图4-2，ESP32程序流程图**

## 功能验证

### 使用Vivado工具完成FPGA模块的开发

将案例Vivado工程打开，添加工程的约束文件：

# Clock signal 100 MHz

create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports { clk\_100MHz }];

set\_property -dict { PACKAGE\_PIN H4 IOSTANDARD LVCMOS33 } [get\_ports { clk\_100MHz }]; #IO\_L13P\_T2\_MRCC\_35 Sch=sysclk

#Gyro

set\_property -dict {PACKAGE\_PIN P12 IOSTANDARD LVCMOS33} [get\_ports Gyro\_IIC\_SCL];

set\_property -dict {PACKAGE\_PIN P13 IOSTANDARD LVCMOS33} [get\_ports Gyro\_IIC\_SDA];

set\_property PULLUP true [get\_ports {Gyro\_IIC\_SCL}]

set\_property PULLUP true [get\_ports {Gyro\_IIC\_SDA}]

set\_property -dict {PACKAGE\_PIN H1 IOSTANDARD LVCMOS33} [get\_ports VCC];

set\_property -dict {PACKAGE\_PIN H2 IOSTANDARD LVCMOS33} [get\_ports GND];

set\_property -dict {PACKAGE\_PIN N4 IOSTANDARD LVCMOS33} [get\_ports IIC\_OE];

set\_property -dict {PACKAGE\_PIN L13 IOSTANDARD LVCMOS33} [get\_ports VCCEN];

#QSPI

set\_property CLOCK\_DEDICATED\_ROUTE FALSE [get\_nets {I\_qspi\_clk\_IBUF}]

set\_property -dict {PACKAGE\_PIN P2 IOSTANDARD LVCMOS33} [get\_ports {qspi\_d0}]

set\_property -dict {PACKAGE\_PIN L14 IOSTANDARD LVCMOS33} [get\_ports {qspi\_d1}]

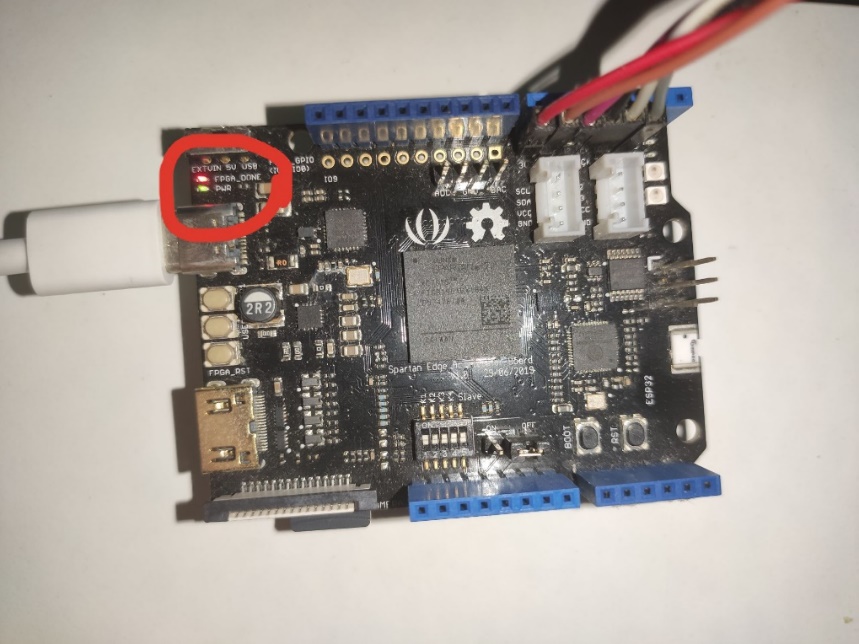
set\_property -dict {PACKAGE\_PIN J13 IOSTANDARD LVCMOS33} [get\_ports {qspi\_d2}]

set\_property -dict {PACKAGE\_PIN D13 IOSTANDARD LVCMOS33} [get\_ports {qspi\_d3}]

set\_property -dict {PACKAGE\_PIN M13 IOSTANDARD LVCMOS33} [get\_ports {I\_qspi\_cs}]

set\_property -dict {PACKAGE\_PIN H14 IOSTANDARD LVCMOS33} [get\_ports {I\_qspi\_clk}]

按照Vivado开发流程依次通过Synthesis, Implementation和Generate Bitstream，生成.bit文件，将生成的bitstream文件加载到FPGA中，可以看到SEA-Board开发板上FPGA\_DONE的LED灯亮，如图5-1所示。



**图5-1，**SEA-Board完成FPGA的下载

### 完成AWS-loT的配置，获取相关证书

该步骤省略，详细参考AWS loT平台关于证书的获取以及相关设备的连接教程，获取相关证书后用于ESP32程序中的连接配置。

官方链接：<https://docs.aws.amazon.com/iot/latest/developerguide/iot-gs.html>

### 使用Arduino-IDE完成ESP32的开发

将案例工程打开，并配置自己的WiFi和AWS loT证书和连接，通过验证并下载至ESP32中，下载完成，Arduino-IDE控制台如下所示。

esptool.py v2.6

Serial port COM7

Connecting.......

Chip is ESP32D0WDQ6 (revision 1)

Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None

MAC: a4:cf:12:24:b9:68

Uploading stub...

Running stub...

Stub running...

Changing baud rate to 921600

Changed.

Configuring flash size...

Auto-detected Flash size: 4MB

Compressed 8192 bytes to 47...

Writing at 0x0000e000... (100 %)

Wrote 8192 bytes (47 compressed) at 0x0000e000 in 0.0 seconds (effective 10923.0 kbit/s)...

Hash of data verified.

Compressed 17392 bytes to 11186...

Writing at 0x00001000... (100 %)

Wrote 17392 bytes (11186 compressed) at 0x00001000 in 0.2 seconds (effective 823.3 kbit/s)...

Hash of data verified.

Compressed 825056 bytes to 489705...

Writing at 0x00010000... (3 %)

……

Writing at 0x00084000... (100 %)

Wrote 825056 bytes (489705 compressed) at 0x00010000 in 9.2 seconds (effective 719.5 kbit/s)...

Hash of data verified.

Compressed 3072 bytes to 128...

Writing at 0x00008000... (100 %)

Wrote 3072 bytes (128 compressed) at 0x00008000 in 0.0 seconds (effective 4096.0 kbit/s)...

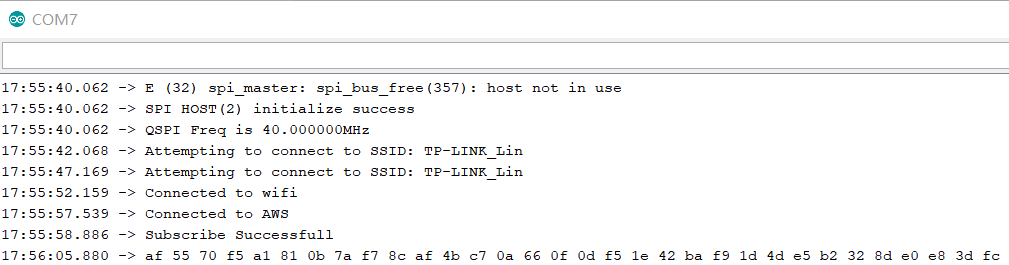
Hash of data verified.

Leaving...

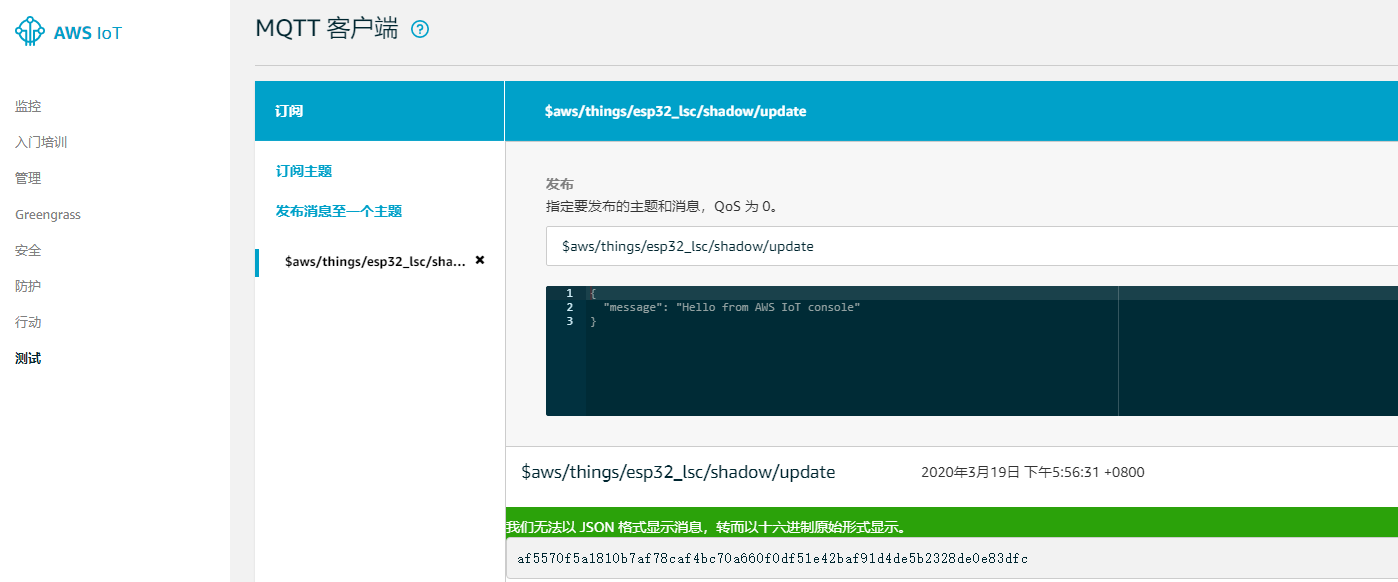
Hard resetting via RTS pin...

### 使用串口工具及AWS loT控制台观察结果

打开Arduino-IDE的串口工具，调整波特率为115200（此处在ESP32程序中可配置），可以观察到ESP32会自动连接WiFi和AWS loT平台；如果连接失败，请重新配置WiFi和AWS loT相关证书，并重新验证和下载。连接成功后，可以在串口工具中看到与AWS loT同步上传的256 bits 哈希值，如图5-2和图5-3所示。



**图5-2，**Arduino-IDE 串口工具显示连接状态及Hash值



**图5-3，**AWS loT控制台接收到Hash值