第一章 基于QSPI协议的微控制器和FPGA通信

1.1 QSPI通信协议简介

QSPI是Quad SPI的简写，表示4线SPI，是Motorola公司推出的SPI接口的扩展，比SPI应用更加广泛。其中共有6个接口使协处理器与主机互联。使用该接口，用户可以一次性传输包含多达16个8位或16位数据的传输队列。一旦传输启动，直到传输结束，都不需要CPU干预，极大的提高了传输效率。该协议在ColdFire系列MCU得到广泛应用。与SPI相比，QSPI的最大结构特点是以80字节的RAM代替了SPI的发送和接收数据寄存器。

图片包含 游戏机, 截图, 键盘

描述已自动生成

图 1 QSPI协议时序图

如图 1所示，QSPI协议中的6个数据线分别为nCS，SCLK，IO0，IO1，IO2，IO3。nCS接口被用来启动，当该接口置零时开始工作。SCLK是一个同步时钟计时，每一拍都传送一个数据。当进入命令输入阶段时，在IO0传递8位命令。在传输的过程中我们先传递高位后传递低位。当进入地址传输阶段，为了传递的速度更加快速，我们选择用四个传输口IO0，IO1，IO2，IO3分别传输8位数据的第0和4位，第5和1位，第6和2位，第7和3位。以此类推，就可以传输一个24位的地址在6拍之内。由于我们要通过QSPI对两端进行读和写入的操作，所以我们选择空出4拍的富余，让FPGA进行分析处理数据。在数据的传回阶段，我们同样的采取了，分4个传输口传输的方式，以此来加快传输的速度。在最后，我们并没有结束符，这是因为我们可以通过nCS来控制系统的运行。同时，当nCS拉低时，我们可以一直传输多位数据。

1.2 QSPI通信硬件连接[1]

SEA board 上搭载的Spartan-7 FPGA开发板XC7S15-FTGB196C的部分原理图如下图 2所示，其中标红的是FPGA上和QSPI通信相关的管脚。P2管脚对应FPGA\_QSPI\_D,H14管脚对应FPGA\_QSPI\_CLK,M13管脚对应FPGA\_QSPI\_CS,D13管脚对应FPGA\_QSPI\_HD,L14管脚对应FPGA\_QSPI\_Q,D13管脚对应FPGA\_QSPI\_HD,J13管脚对应FPGA\_QSPI\_WP。

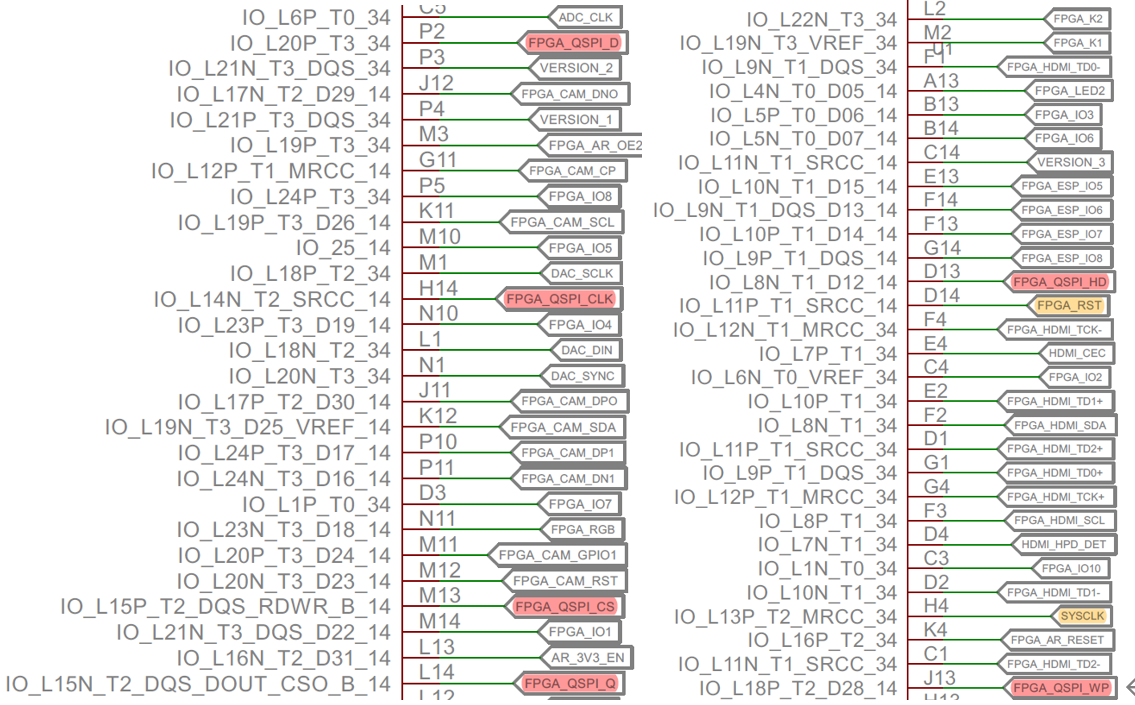


图 2 FPGA与QSPI通信相关的部分管脚原理图

图 3为ESP32-dD0WDQ6的部分原理图，其中标红的是QSPI通信使用的管脚。其中，GPIO21对应ESP\_QSPI\_HD,GPIO22对应ESP\_QSPI\_WP,GPIO19对应ESP\_QSPI\_Q,GPIO23对应ESP\_QSPI\_D,GPIO18对应ESP\_QSPI\_CLK,GPIO5对应ESP\_QSPI\_CS。

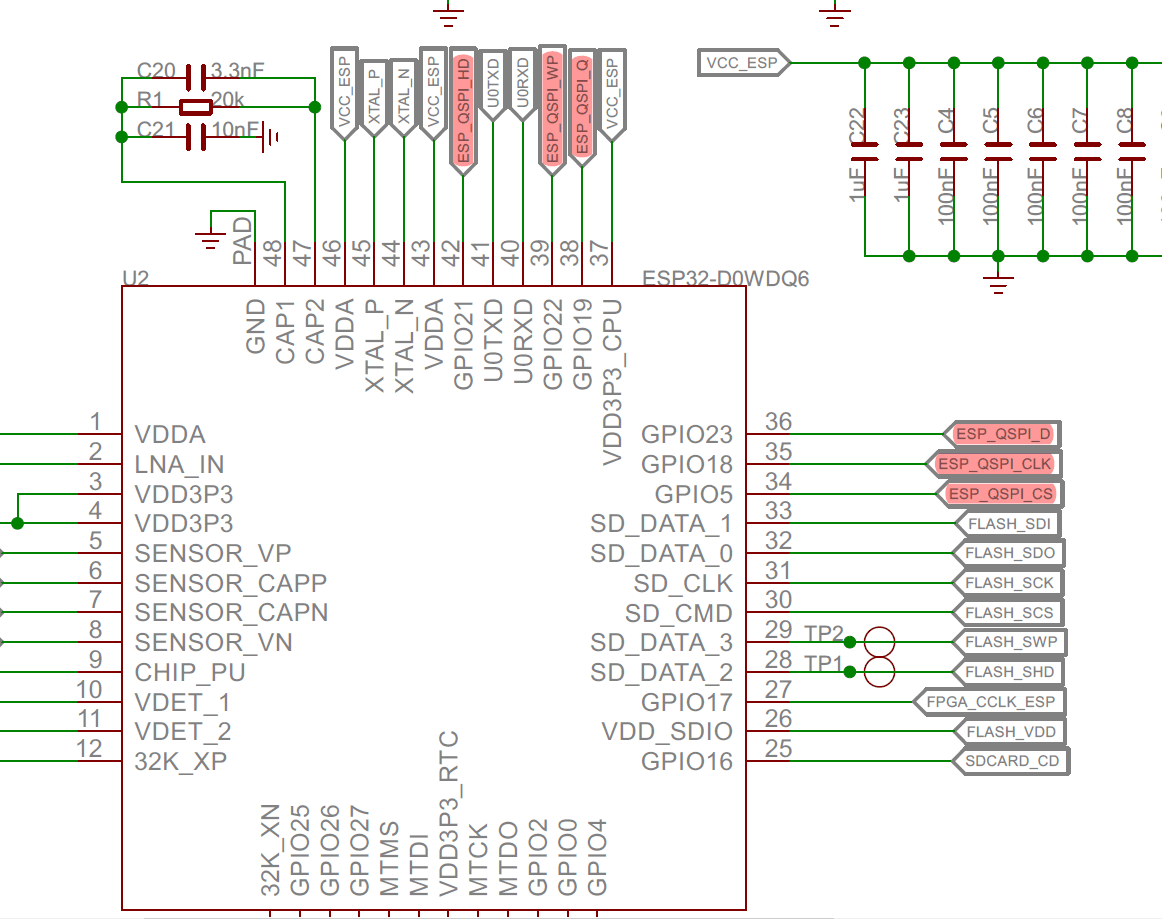


图 3 ESP32与QSPI通信相关的部分管脚原理图

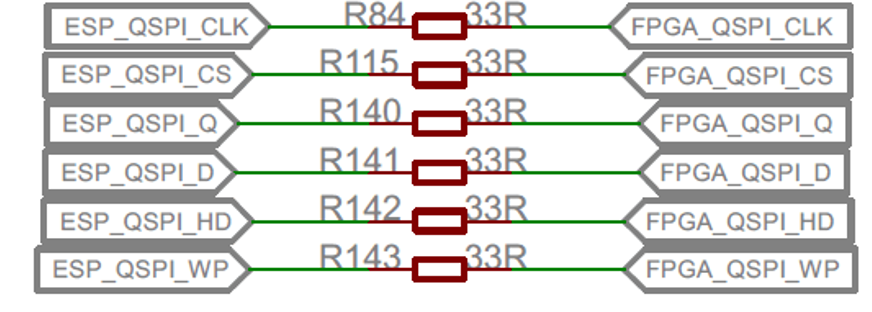


图 4 ESP32与FPGA的QSPI接口电路

图 4为ESP32连接FPGA时对应的管脚图，ESP与FPGA两端分别与一个电阻相连，以此来完成整个硬件电路的设计。

1.3 QSPI通信数据处理

**1.3.1 ESP32侧数据发送**

我们选择开发ESP32的软件为Arduino，本来打算利用MicroPython来做开发的，因为MicroPython的开发是逐语句的、实时的。但是SEA board公司提供的QSPI通讯库只提供了Arduino版本的，所以我从github上下载了QSPI通信的库[[1]](#footnote-1),加载到Ardunio中。主要的函数有以下两个：ESP32向存储器写数据的函数和ESP32向存储器读数据的函数，主要参数有两个部分是地址和数据。

**bool** sea\_esp32\_qspi::write(**uint16\_t** addr, **char** data[], **uint16\_t** length)

{

    qspiWritePolling(&this->qspi, addr, WR\_CMD,(**uint8\_t** \*)data, length);

}

**bool** sea\_esp32\_qspi::read(**uint16\_t** addr, **char** data[], **uint16\_t** length)

{

    qspiReadPolling(&this->qspi, addr, RD\_CMD, (**uint8\_t** \*)data, length);

}

**1.3.2 FPGA侧数据处理**

由于FPGA需要处理的数据是六条信号线上的生数据，我们自行编写状态机来分时处理不同类型的数据。由图 5所示，count为对SCLK上升沿的计数器，根据count的不同，在不同的时序阶段处理对应的数据。



图 5 FPGA处理QSPI数据的时序状态机

第二章 FPGA内嵌浮点数计算单元FPU

1.1 表示浮点数计算的方法[2]

在传递的数据中，共分为三个部分。第一部分为8位操作符，第二部分为两个16位数据。前八位用来表示4个不同的运算，分别用00000001来表示+，用00000010表示-，用00000100表示\*，用00001000表示/。之后的前16位里前八位用来表示浮点数的整数部分，后八位用来表示浮点数的小数部分。以此类推，输入浮点数据。

为了定点到浮点的转换，定点表示为有符号整数类型，这与Xilinx System Generator使用的数据类型一致。 定点值使用加权的二进制补码表示为以2的固定幂形式。定点数的二进制表示形式包含三个字段，如图5所示（尽管它仍然是加权的二进制补码）。

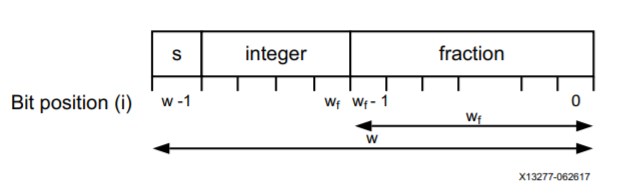


图 6 定点表示中的位字段

在图 6中，位的位置已标记为索引i。 基于此，定点数的值由下式给出：

例如，当指定总宽度为32和小数宽度为0时，将获得32位带符号整数表示形式。 在转换操作中采用舍入到最接近。 要提供符号位，整数字段的宽度必须至少为1，要求小数宽度不大于w-1。固定浮点运算符还可以选择执行32位和64位运算。位有符号和无符号整数转换，以将标准软件整数数据格式转换为浮点数。

1.2 判断并执行运算

先判断要进行的运算，后运用相对应的IPcore来执行接下来要进行的运算。

**1.2.1浮点数的加减运算**

浮点加法运算的实现包括以下几个步骤符号判断: 对阶、尾数加减操作、规格化、舍入操作、溢出判断。具体实现时通常把规格化、舍入操作、溢出判断作为一个步骤实现。浮点数的格式显然可以分为两部分, 即符号和数据的绝对值。若符号相同则符号不便绝对值相加; 若符号不同则须比较两绝对值的大小然后两绝对值作差运算。符号不同时首先判断和的符号, 显然若两浮点数的阶不同时和的符号当与阶数大的操作数相同; 若阶数不同则继续比较对阶操作首先比较两浮点数的阶数大小。然后需要两数对阶。对阶的原则是小阶对大阶，小阶对大阶的好处是，当小阶不同于大阶时，只需要移除小阶数的尾数部分的低位部分，加法流程图见图 7。

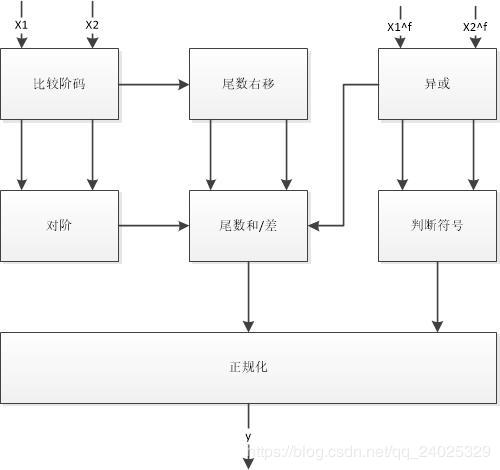


图 7 加/减运算流程

**1.2.2浮点数的乘法运算**

浮点数的乘法运算相对比较简单，只需要将两个操作数的符号位进行异或运算，再将阶码部分做和、尾数部分做积即可。同时需要检查操作数的运算结果是否有溢出问题。乘法流程见图 8:

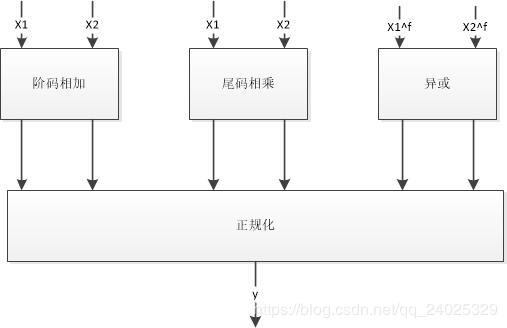


图 8 乘/除运算流程

参考文献

1. Seeed Studio, “Spartan Edge Accelerator Board v1.0”, Aug. 2019
2. XILINX, “Floating-Point Operator v7.1 LogiCore IP Product Guide”, Nov. 2019

1. <https://github.com/CutClassH/sea_esp32_qspi> [↑](#footnote-ref-1)