**STM32 与FPGA MVB通信说明**

1、FPGA----STM32 硬件连接如下：

FPGA管脚 STM32F103ZET6(144pin)

管脚序号 管脚名称 默认服用功能

//////////////////地址线

10 PF0 FSMC\_A0

11 PF1 FSMC\_A1

12 PF2 FSMC\_A2

13 PF3 FSMC\_A3

14 PF4 FSMC\_A4

15 PF5 FSMC\_A5

50 PF12 FSMC\_A6

53 PF13 FSMC\_A7

54 PF14 FSMC\_A8

55 PF15 FSMC\_A9

56 PG0 FSMC\_A10

57 PG1 FSMC\_A11

87 PG2 FSMC\_A12

88 PG3 FSMC\_A13

89 PG4 FMSC\_A14

90 PG5 FMSC\_A15

80 PD11 FSMC\_A16

81 PD12 FSMC\_A17

82 PD13 FSMC\_A18

2 PE3 FSMC\_A19

3 PE4 FSMC\_A20

4 PE5 FSMC\_A21

5 PE6 FSMC\_A22

1 PE2 FSMC\_A23

128 PG13 FSMC\_A24--------------X

129 PG14 FSMC\_A25--------------X

//////////////数据线

85 PD14 FSMC\_D0

86 PD15 FSMC\_D1

114 PD0 FSMC\_D2

115 PD1 FSMC\_D3

58 PE7 FSMC\_D4

59 PE8 FSMC\_D5

60 PE9 FSMC\_D6

63 PE10 FSMC\_D7

64 PE11 FSMC\_D8

65 PE12 FSMC\_D9

66 PE13 FSMC\_D10

67 PE14 FSMC\_D11

68 PE15 FMSC\_D12

77 PD8 FSMC\_D13

78 PD9 FSMC\_D14

79 PD10 FSMC\_D15

127 PG12 FSMC\_NE4 X

125 PG10 FSMC\_NE3/FSMC\_NCE4\_1 X

124 PG9 FSMC\_NE2/FSMC\_NCE3 X

141 PE0 FSMC\_NBL0 X }Nor/psram 信号

142 PE1 FSMC\_NBL1 X

117 PD3 FSMC\_CLK X

123 PD7 FSMC\_NE1/FSMC\_NCE2 ---CS 片选

118 PD4 FSMC\_NOE -----输出使能

119 PD5 FSMC\_NWE ---- 写使能 }公用信号

122 PD6 FSMC\_NWAIT

18 PF6 FSMC\_NIORD

19 PF7 FSMC\_NREG

20 PF8 FSMC\_NIOWR

21 PF9 FSMC\_CD

22 PF10 FSMC\_INTR

49 PF11 FSMC\_NIOS16

91 PG6 FSMC\_INT2

92 PG7 FSMC\_INT3

126 PG11 FSMC\_NCE4\_2

2、FPGA与STM32 总线通讯使用的寄存器地址

#define     UART\_RBR            (0x600001f8)//接收缓冲寄存器，只读寄存器，起始位之后，数据会以一定的波特率移入移位寄存器，等一个字节的数据接收完，会自动移入接收缓冲寄存

#define     UART\_THR            (0x600001f8)//发送保持寄存器，也叫只写寄存器，一次发送操作从写寄存器USART——THR开始。

#define     UART\_TEST           (0x600000fd)

#define     UART\_LSR            (0x600001fd)//FPGA的一个返回值。如果FPGA准备好，则会返回一个可操作指令。

//UART emulation寄存器状态定义

#define     UART\_DR\_OK         0x01

#define     UART\_THRE\_OK        0x20

#define   UART\_TEMT\_OK        0x40

   //常量定义

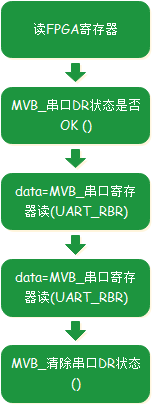
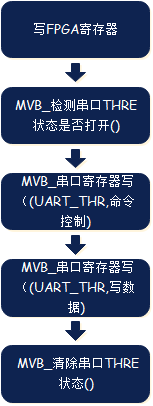
#define     MVB\_DR\_CHECK\_TIMES      500

#define     MVB\_THRE\_CHECK\_TIMES  500

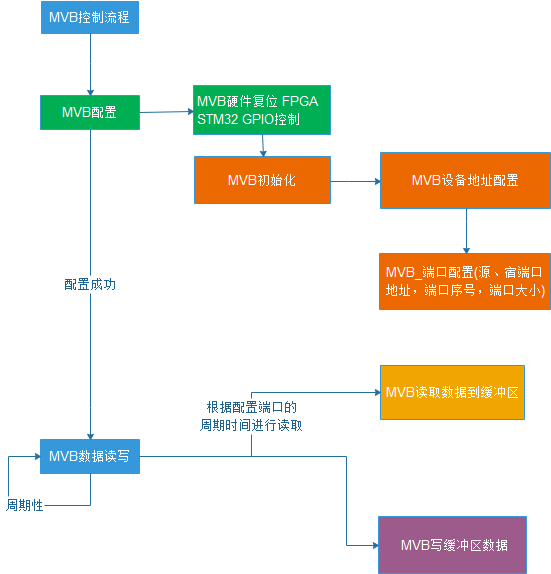
#define     RESET\_BYTE\_COUNTE        64

**MVB 通过读写 UART\_RBR、UART\_THR寄存器 通过不同命令启动不同的读写方式流程**

**MVB 使用8根数据线，读写寄存器数据都是1个字节，小端模式**



MVB工作流程图如下:



**宿端口、源端口 总共不超过31个大小**

**发送 、接收BUFFER 数据不超过250byte**

**下面是STM32与FPGA总线通讯协议的伪代码描述 访问寄存器使用数据线是8位**

**1、MVB\_串口寄存器读**(寄存器地址)

**2**、**MVB\_串口寄存器写**(寄存器地址,值)

**3**、**MVB\_串口状态检测**()

**{**

If(**MVB\_串口寄存器读**(UART\_LSR)&0x9E==0x90)

状态正常

Else

状态异常

**}**

**5、MVB\_清除串口()**

**{**

for(i=0;i<RESET\_BYTE\_COUNTE;i++) //64次

**MVB\_串口寄存器写**(UART\_THR,0x55);

**}**

**6、MVB\_检测串口THRE状态是否打开()**

**{**

for(i=0;i<MVB\_THRE\_CHECK\_TIMES;i++)//500次

{

LSR = **MVB\_串口寄存器读(**UART\_LSR);

if((LSR&UART\_THRE\_OK) == UART\_THRE\_OK)//0x01

{

return(TRUE);

}

}

**}**

**7、MVB\_清除串口THRE状态()**

**{**

LSR = **MVB\_串口寄存器读**(UART\_LSR);

LSR &= 0xdf;

**MVB\_串口寄存器写**(UART\_LSR, LSR);

**}**

**8 MVB\_清除串口DR状态()**

**{**

u8 LSR;

LSR = **MVB\_串口寄存器读**(UART\_LSR);

LSR &= 0xfe;

**MVB\_串口寄存器写**(UART\_LSR, LSR);

**}**

**9、MVB\_串口DR状态是否OK ()**

**{**

LSR = 0;

for(i=0;i<**MVB\_DR\_CHECK\_TIMES**;i++)

{

LSR = **MVB\_串口寄存器读**(UART\_LSR);

if((LSR&UART\_DR\_OK) == UART\_DR\_OK)//0x40

return(TRUE);

}

Return FALSE

**}**

10、**MVB\_停止通讯()**

**{**

If(**MVB\_检测THRE状态是否打开()** == OK)

{

**MVB\_串口寄存器写**(UART\_THR,’S’);

**MVB\_清除串口THRE状态()**

}else

Return FALSE;

If(**MVB\_串口DR状态是否OK () == OK)**

**{**

MVBDAddr = **MVB\_串口寄存器读**(UART\_RBR); //MVB设备地址低位

MVBDAddr += **MVB\_串口寄存器读**(UART\_RBR)<<8; //MVB设备地址高位

**MVB\_清除串口DR状态**();

**}**

MVBDAddr &= 0x3FFF;//don't change device address,switch off a and b lines

result = **MVB\_检测串口THRE状态是否打开**();

if(result == TRUE)

{

**MVB\_串口寄存器写**(UART\_THR ,'C'); //控制写命令 **MVB\_串口寄存器写**(UART\_THR, MVBDAddr); //mvb device address low

**MVB\_串口寄存器写**(UART\_THR (MVBDAddr>>8); //mvb device address high

**MVB\_清除串口THRE状态();**

}

else

{

return(FALSE);

}

**}**

**11 清除MVB（）**

**{**

**if(MVB\_串口状态检测**() **== FALSE )**

**return(FALSE);**

**MVB\_清除串口()**

**MVB\_停止通讯()**

**}**

**12、MVB\_启动通讯()**

**{**

If(**MVB\_检测串口THRE状态是否打开()** == OK)

{

**MVB\_串口寄存器写**(UART\_THR,’S’);

**MVB\_清除串口THRE状态()**

}else

Return FALSE;

If(**MVB\_串口DR状态是否OK () == OK)**

**{**

MVBDAddr = **MVB\_串口寄存器读**(UART\_RBR); //MVB设备地址低位

MVBDAddr += **MVB\_串口寄存器读**(UART\_RBR)<<8; //MVB设备地址高位

**MVB\_清除串口DR状态**();

**}**

MVBDAddr|= 0XC000;//don't change device address,switch on a and b lines

result = **MVB\_检测串口THRE状态是否打开**();

if(result == TRUE)

{

**MVB\_串口寄存器写**(UART\_THR ,'C'); //控制写命令 **MVB\_串口寄存器写**(UART\_THR, MVBDAddr); //mvb device address low

**MVB\_串口寄存器写**(UART\_THR (MVBDAddr>>8); //mvb device address high

**MVB\_清除串口THRE状态();**

}

else

{

return(FALSE);

}

**}**

**13、禁止MVB所有端口()**

**{**

for(i=0;i<MAXPORTNUM;i++)

{

**result =MVB\_检测串口THRE状态是否打开()**

}

if(result == TRUE)

{

**MVB\_串口寄存器写**(UART\_THR ,'C'); //控制写命令

**MVB\_串口寄存器写**(UART\_THR ,i');

**MVB\_串口寄存器写**(UART\_THR ,0x00);

**MVB\_串口寄存器写**(UART\_THR ,0x00);

delay\_ms(1);

**MVB\_清除串口THRE状态();**

}

**}**

/\* MvbPortAddr:MVB源、宿端口地址

\* PortNum：MVB端口序号

\* direction:端口是宿端口还是源端口 宿端口:(0) 源端口：(1)

\* length：端口大小

\*/

14、MVB\_端口配置(u16 MvbPortAddr,u8 PortNum,u16 direction,u16 length)

{

    u16 MVBPortCFG = 0;

    u8  result = 0;

    MVBPortCFG = (direction << 15) | (length << 12) | MvbPortAddr;

result = **MVB\_检测串口THRE状态是否打开**();

    if(result == TRUE)

    {

**MVB\_串口寄存器写（**(UART\_THR ,''H');

**MVB\_串口寄存器写**(UART\_THR ,PortNum);

**MVB\_串口寄存器写**(UART\_THR MVBPortCFG);

**MVB\_串口寄存器写**(UART\_THR MVBPortCFG>>8);

**MVB\_清除串口THRE状态();**

 }

    else

    {

        return(FALSE);

    }

    return(TRUE);

}

/\* MvbPortAddr:MVB源、宿端口地址

\* PortNum：MVB端口序号

\* direction:端口是宿端口还是源端口 宿端口:(0) 源端口：(1)

\* length：端口大小

\*/

u8 MVB\_端口配置信息确认(6MvbPortAddr,u8 PortNum,u16 direction,u16 length)

{

    u16 MVBPortCFG = 0;

    u16 MVBPortCFGcheck = 0;

    u8  result = 0;

MVBPortCFG = (direction << 15) | (length << 12) | MvbPortAddr;

result =  **MVB\_检测串口THRE状态是否打开**();     //FPGA发送和接受是否已经准备好。

    if(result == TRUE)

    {

**MVB\_串口寄存器写（**(UART\_THR ,'I');              //handle definition write

**MVB\_串口寄存器写（**(UART\_THR ,PortNum);          //handle number

**MVB\_清除串口DR状态**();

**MVB\_清除串口THRE状态();**

        if(**MVB\_串口DR状态是否OK ()** == TRUE)

        {

            MVBPortCFGcheck = **MVB\_串口寄存器读**(UART\_RBR)

            MVBPortCFGcheck += **MVB\_串口寄存器读**(UART\_RBR)<<8;

           // MVB\_clear\_DR();

            if (MVBPortCFG == MVBPortCFGcheck)

            {

                return(TRUE);

            }

            else

            {

                return(FALSE);

            }

        }

        else

        {

            return(FALSE);

        }

    }

    else

    {

        return(FALSE);

    }

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*MvbDeviceAddr MVB设备地址 默认0x00

\* InputLine 默认配置为0x03

\*/

u8  MVB设备地址配置(u16 MvbDeviceAddr,u16 InputLine)

{

    u16 MVBDAddr=0,temp=0;

    vu8 result = 0;

    MVBDAddr = (InputLine << 14) | 0x0000 | MvbDeviceAddr;//mvb device address,led,inputlines  1100 0000 0000 0000

    result = **MVB\_检测串口THRE状态是否打开**();

    if(result == TRUE)

    {

**MVB\_串口寄存器写（**UART\_THR ,'C');//control write

**MVB\_串口寄存器写（**UART\_THR ,MVBDAddr);

**MVB\_串口寄存器写（**UART\_THR ,MVBDAddr>>8);

**MVB\_清除串口THRE状态();**

    }

    else

    {

        return(FALSE);

    }

    result = **MVB\_检测串口THRE状态是否打开**();

    if(result == TRUE)

    {

**MVB\_串口寄存器写（**UART\_THR 'S');    //status read

**MVB\_清除串口THRE状态();**

    }

    else

    {

        return(FALSE);

    }

    result = **MVB\_串口DR状态是否OK ()**;

    if(result == TRUE)

    {

        temp = **MVB\_串口寄存器读**(UART\_RBR);

        temp += **MVB\_串口寄存器读**(UART\_RBR)<<8;

**MVB\_清除串口DR状态**();

}

    else

    {

        return(FALSE);

    }

    if(temp == MVBDAddr) //0x3000

    {

        return(TRUE);

    }

    else

    {

        return(FALSE);

    }

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \* PortNum :端口序号 源端口或者宿端口配置的序号

 \* PointerReceive:接收缓存的地址

 \* Len:读取数据长度(端口大小)默认读取32byte

 \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

u8  MVB读取数据到缓冲区(u8 PortNum,u8 \*PointerReceive,u8 Len)

{

    u8  i;

    u8  result ,BusStatus;

    result = **MVB\_检测串口THRE状态是否打开**();

    if(result == TRUE)

    {

**MVB\_串口寄存器写（**UART\_THR ,'G');

**MVB\_串口寄存器写（**UART\_THR ,PortNum);         //sink handle number

**MVB\_清除串口THRE状态();**

        result =  **MVB\_串口DR状态是否OK ()**;

        if(result == TRUE)

        {

**MVB\_清除串口DR状态 ();**

        BusStatus = **MVB\_串口寄存器读**(UART\_RBR);//mvb bus status value(A and B line)

            BusStatus &= 0xC0;

            if(BusStatus != 0)

            {

                for(i=0; i<32; i++)

                {

                    PointerReceive[i] = **MVB\_串口寄存器读**(UART\_RBR);

                }

                return(BusStatus);

            }

        }

    }

return(FALSE);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \* PortNum :端口序号 源端口或者宿端口配置的序号

 \* PointerReceive:发送数据指针地址

 \* Len:写数据长度(端口大小) 默认不够32byte 填充0x00

 \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

u8  MVB写缓冲区数据(u8 PortNum,u8 \*PointerSend,u8 Len)

{

    u8  i;

    u8  result;

    result = **MVB\_检测串口THRE状态是否打开**();

    if(result == TRUE)

    {

**MVB\_串口寄存器写（**UART\_THR ,'P');     //put data to mvb process data port

**MVB\_串口寄存器写（**UART\_THR ,PortNum);   //source handle number

        for(i=0; i<Len; i++)

        {

**MVB\_串口寄存器写（**UART\_THR ,\*(PointerSend+i));

        }

        for(i=0; i<MAXMVBBYTE-Len; i++)

        {

**MVB\_串口寄存器写（**UART\_THR ,0);

        }

**MVB\_清除串口THRE状态( );**

        return(TRUE);

    }

        return(FALSE);

}