liangkangnan的博客

深入浅出RISC-V调试

2020-03-21 | RISC-V | 22260

9.4k | 9 分钟

1.JTAG简介

目前RISC-V官方支持的调试方式是JTAG(Joint Test Action Group)，而ARM支持的调试方式有JTAG和SWD(Serial Wire Debug)这两种。

JTAG是一种国际标准的调试方式(IEEE1149.1)，而SWD是ARM开发的。

标准JTAG采用四线方式，分别是TCK、TMS、TDI和TDO，有一个可选的TRST引脚。

- TCK：测试时钟输入。
- TMS：测试模式选择。
- TDI：测试数据输入。
- TDO：测试数据输出。

在调试时需要用到一个工具，比如JLink或者CMSIS-DAP，对于这个工具，在这里称为JTAG主机(JTAG host)，而嵌入在芯片内部的JTAG称为JTAG从机(JTAG slave)，需要注意的是上面这些信号的输入输出方向是对于JTAG从机来说的。下文中如无特别说明，JTAG都是指JTAG从机。

一个JTAG主机可以同时对多个JTAG从机进行调试，这通过JTAG扫描链(JTAG Scan Chain)完成，如图1所示。

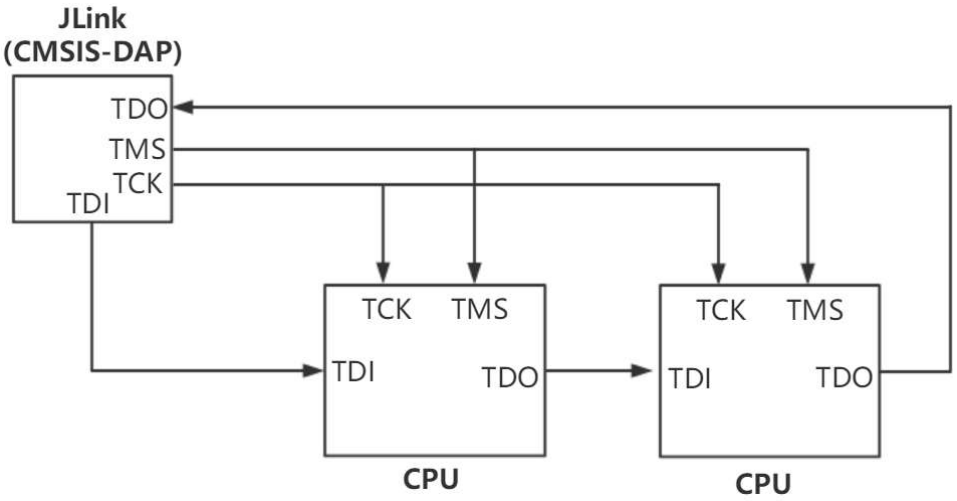

图1 一个JTAG主机连接多个JTAG从机

JTAG内部有一个TAP(Test Access Port)控制器(或者说状态机)，通过TCK和TMS信号来改变状态机的状态。这个状态机的核心是两路SCAN，分别是IR SCAN和DR SCAN，TAP状态机如图2所示。

多少个箭头就是多个状态转换函数：31个

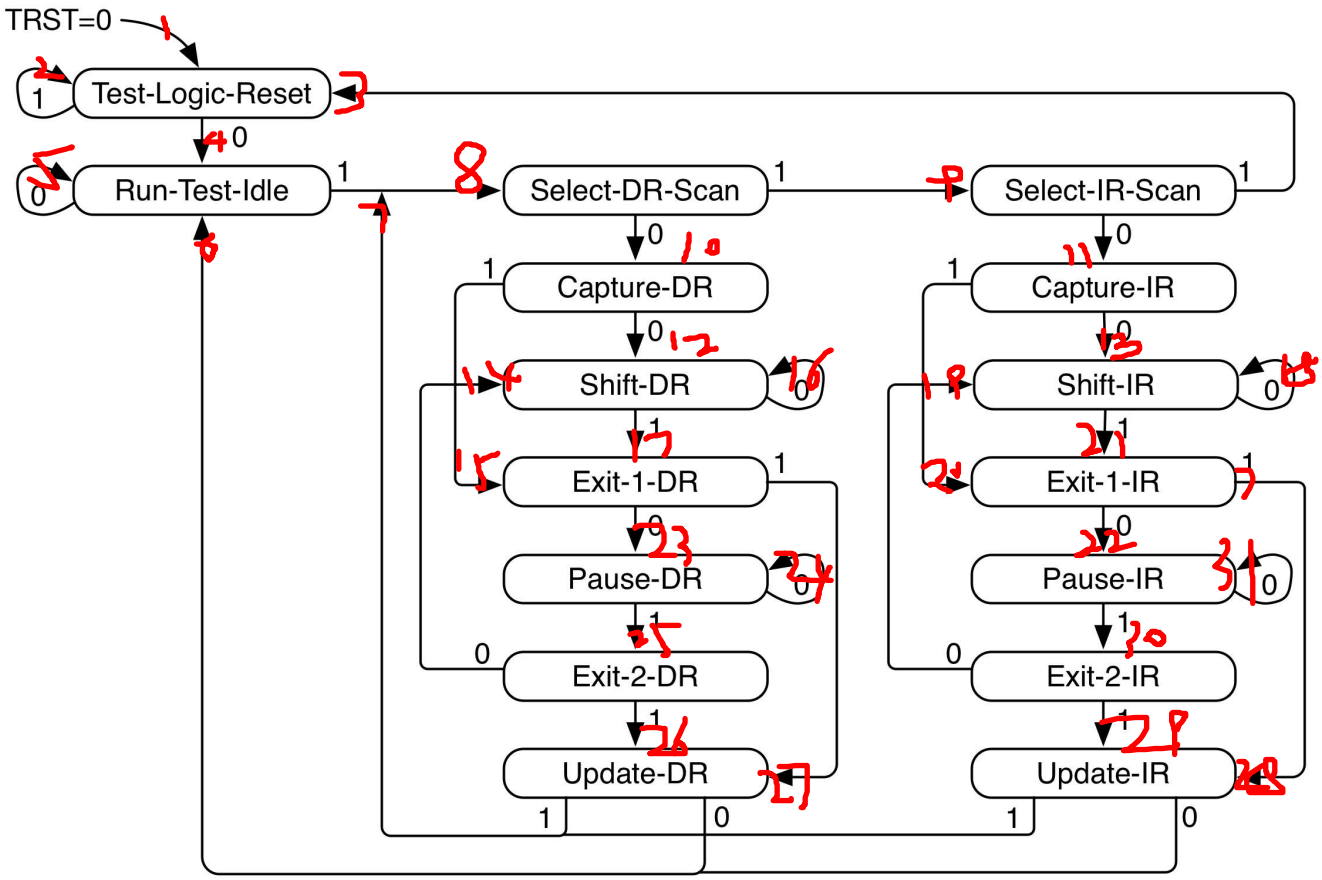

图2 TAP状态机

箭头上的0或1表示的是TMS信号的电平。JTAG在每一个TCK信号的上升沿采样TMS信号和TDI信号，决定状态机的状态是否发生变化，在每一个TCK信号的下降沿输出TDO信号。可以看到，无

论TAP目前处于哪一个状态，只要TMS保持高电平并持续5个TCK时钟，则TAP一定会回到Test-Logic-Reset状态。

JTAG内部有一个IR(instruction register)寄存器和多个DR(data register)寄存器，IR寄存器决定要访问的是哪一个DR寄存器。DR寄存器有IDCODE、BYPASS等。在Test-Logic-Reset状态下IR寄存器默认选择的是IDCODE这个DR寄存器。

JTAG主机通过IR SCAN设置IR寄存器的值，然后通过DR SCAN来读、写相应的DR寄存器。

2.RISC-V调试Spec

调试模块在CPU芯片设计里是最为不起眼的，但又是最为复杂的模块之一，大部分开源的处理器IP都没有调试模块。

下面的内容基于RISC-V debug spec 0.13版本。

目前RISC-V的官方调试上位机是openocd，调试工具可以是JLink或者CMSIS-DAP，RISC-V调试系统框架如图3所示。

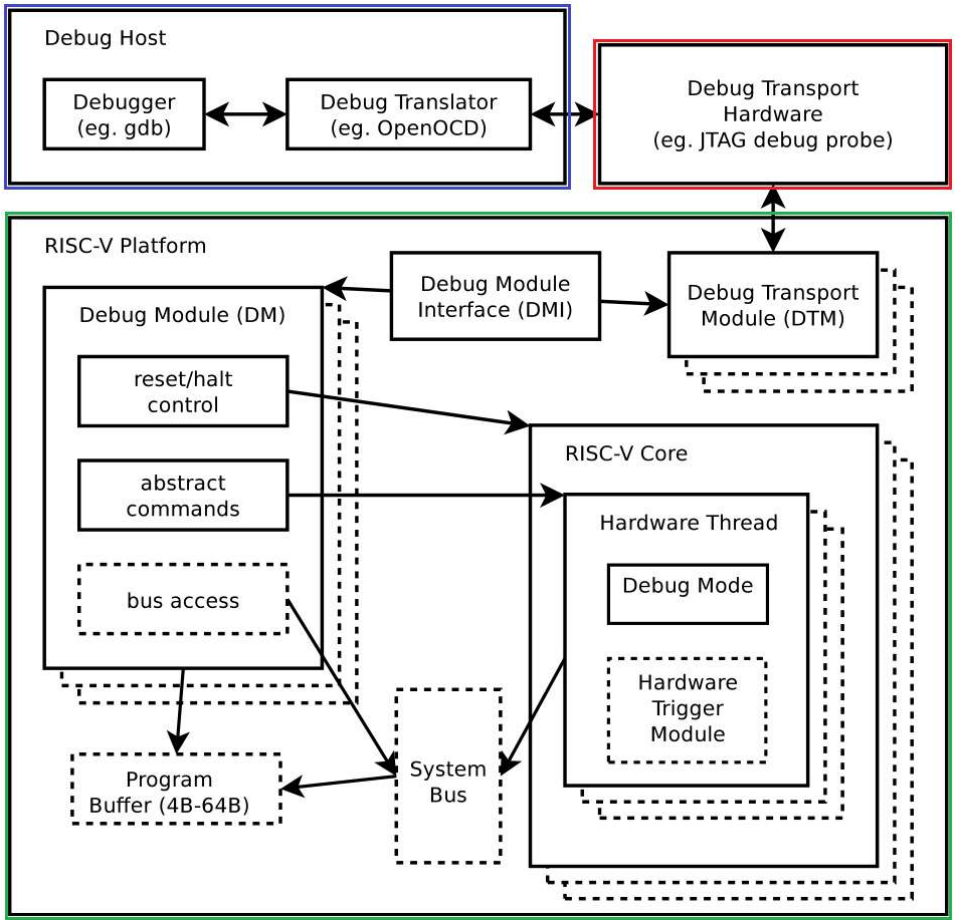

图3 RISC-V调试系统框架

可以看到主要分为3个部分，分别是Debug Host，可以理解为PC；Debug Hardware，可以理解为JLink或者CMSIS-DAP这样的调试工具；第三部分就是嵌入在芯片内部的调试模块。在调试模块内部，与调试工具直接交互的是DTM模块，DTM模块通过DMI接口与DM模块交互。

2.1DTM模块

在DTM模块里实现了一个TAP控制器(状态机)，其中IR寄存器的长度最少为5位，当TAP控制器复位时，IR的值默认为5’ b00001，即选择的是IDCODE寄存器。DTM模块的寄存器(DR寄存器)定义如图4所示。

Address	Name	Description
0x00	BYPASS	JTAG recommends this encoding
0x01	IDCODE	JTAG recommends this encoding
0x10	DTM Control and Status (`dtmcs`)	For Debugging
0x11	Debug Module Interface Access (`dmi`)	For Debugging
0x12	Reserved (BYPASS)	Reserved for future RISC-V debugging
0x13	Reserved (BYPASS)	Reserved for future RISC-V debugging
0x14	Reserved (BYPASS)	Reserved for future RISC-V debugging
0x15	Reserved (BYPASS)	Reserved for future RISC-V standards
0x16	Reserved (BYPASS)	Reserved for future RISC-V standards
0x17	Reserved (BYPASS)	Reserved for future RISC-V standards
0x1f	BYPASS	JTAG requires this encoding

图4 DTM寄存器

其中红色框起来的寄存器是必须要实现的。下面简单介绍一下这几个寄存器。

2.1.1 IDCODE寄存器(0x01)

当TAP状态机复位时，IR寄存器的值默认为0x01，即选择的是IDCODE寄存器。IDCODE寄存器的每一位含义如图5所示。IDCODE是只读寄存器。

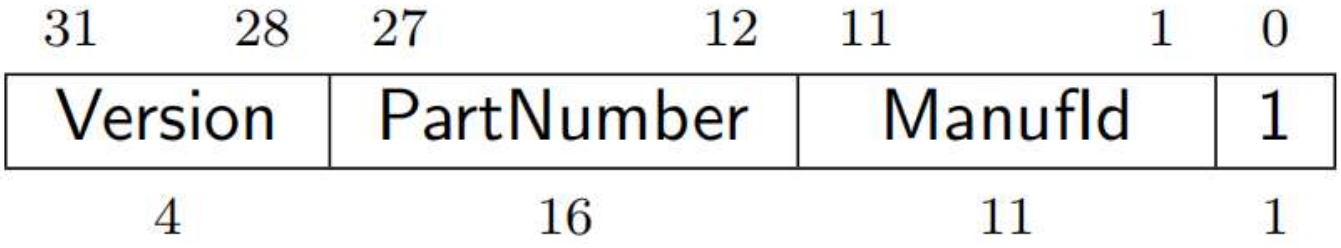

图5 IDCODE寄存器

- Version：只读，版本号，可为任意值。

- PartNumber：只读，可为任意值。
- ManufId：只读，厂商号，遵循JEP106标准分配，实际中可为任意值，只要不与已分配的厂商号冲突即可。

2.1.2 DTM控制和状态寄存器(dtmcs，0x10)

dtmcs寄存器的每一位含义如图6所示。

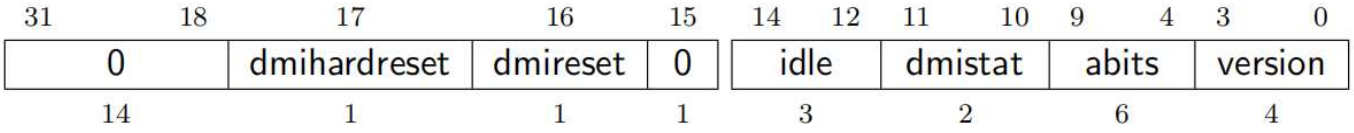

图6 dtmcs寄存器

- dmihardreset：DTM模块硬复位，写1有效。
- dmireset：清除出错，写1有效。
- idle：只读，JTAG 主机在Run-Test-Idle状态停留的时钟周期数，0表示不需要进入Run-Test-Idle状态，1表示进入Run-Test-Idle状态后可以马上进入下一个状态，以此类推。
- dmistat：只读，上一次操作的状态。0表示无出错，1或者2表示操作出错，3表示操作还未完成。
- abits：只读，dmi寄存器中address域的大小(位数)。
- version：只读，实现所对应的spec版本，0表示0.11版本，1表示0.13版本。

2.1.3 DM模块接口访问寄存器(dmi，0x11)

dmi寄存器的每一位含义如图7所示。

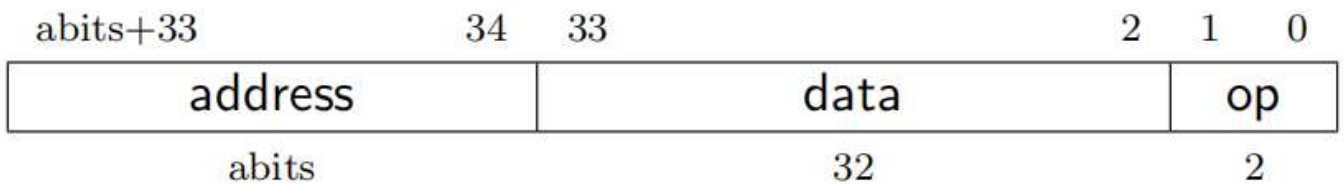

图7 dmi寄存器

- address：可读可写，DM寄存器的长度(位数)。
- data：可读可写，往DM寄存器读、写的数据，固定为32位。
- op：可读可写，读或者写这个域时有不同的含义。当写这个域时，写0表示忽略address和data的值，相当于nop操作；写1表示从address指定的寄存器读数据；写2表示把data的数据写到address指定的寄存器。写3为保留值。当读这个域时，0表示上一个操作正确完成；1为保留值；2表示上一个操作失败，这个状态是会被记住的，因此需要往dtmcs寄存器的dmireset域写1才能清除这个状态。3表示上一个操作还未完成。

在Update-DR状态时，DTM开始执行op指定的操作。在Capture-DR状态时，DTM更新data域。

2.1.4 BYPASS寄存器(0x1f)

只读，长度为1，值固定为0。

2.2DM模块

从图3可知，DM模块访问RISC-V Core有两种方式，一种是通过abstract command，另一种是通过system bus。abstract command方式是必须要实现的，system bus的方式是可选的。

DM模块的寄存器都为32位，定义如图8所示。

Address	Name
0x04	Abstract Data 0 (data0)
0x0f	Abstract Data 11 (data11)
0x10	Debug Module Control (dmcontrol)
0x11	Debug Module Status (dmstatus)
0x12	Hart Info (hartinfo)
0x13	Halt Summary 1 (haltsum1)
0x14	Hart Array Window Select (hawindowsel)
0x15	Hart Array Window (hawindow)
0x16	Abstract Control and Status (abstractcs)
0x17	Abstract Command (command)
0x18	Abstract Command Autoexec (abstractauto)
0x19	Configuration String Pointer 0 (confstrptr0)
0x1a	Configuration String Pointer 1 (confstrptr1)
0x1b	Configuration String Pointer 2 (confstrptr2)
0x1c	Configuration String Pointer 3 (confstrptr3)
0x1d	Next Debug Module (nextdm)
0x20	Program Buffer 0 (progbuf0)
0x2f	Program Buffer 15 (progbuf15)
0x30	Authentication Data (authdata)
0x34	Halt Summary 2 (haltsum2)
0x35	Halt Summary 3 (haltsum3)
0x37	System Bus Address 127:96 (sbaddress3)
0x38	System Bus Access Control and Status (sbcs)
0x39	System Bus Address 31:0 (sbaddress0)
0x3a	System Bus Address 63:32 (sbaddress1)
0x3b	System Bus Address 95:64 (sbaddress2)
0x3c	System Bus Data 31:0 (sbdata0)
0x3d	System Bus Data 63:32 (sbdata1)
0x3e	System Bus Data 95:64 (sbdata2)
0x3f	System Bus Data 127:96 (sbdata3)
0x40	Halt Summary 0 (haltsum0)

图8 DM寄存器

下面介绍一下红色框起来这几个重要的寄存器。

2.2.1 data寄存器(data0~data11，0x04~0x0f)

这12个寄存器是用于abstract command的数据寄存器，长度为32位，可读可写。

2.2.2 DM控制寄存器(dmcontrol，0x10)

dmcontrol寄存器的每一位含义如图9所示。

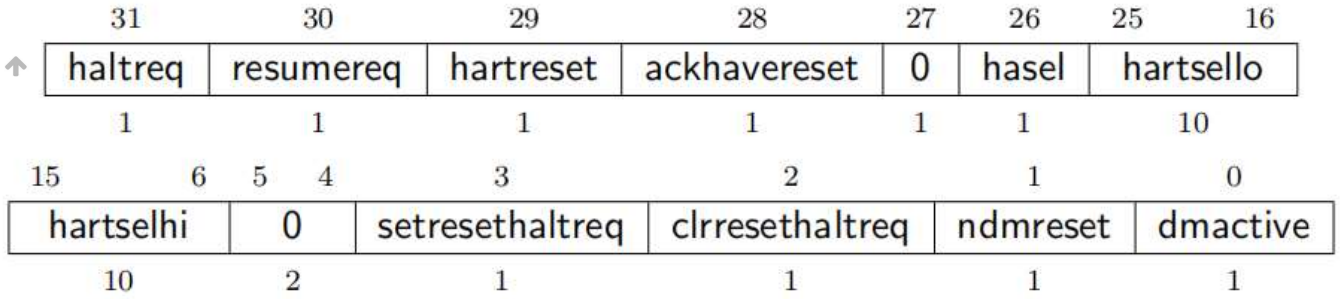

图9 dmcontrol寄存器

- haltreq：只写，写1表示halt(暂停)当前hart(hart表示CPU核，存在多核的情况)。
- resumereq：只能写1，写1表示resume(恢复)当前hart，即go。
- hartreset：可读可写，写1表示复位DM模块，写0表示撤销复位，这是一个可选的位。
- ackhavereset：只能写1，写1表示清除当前hart的havereset状态。
- hasel：可读可写，0表示当前只有一个已经被选择了的hart，1表示当前可能有多个已经被选择了的hart。
- hartsello：可读可写，当前选择的hart的低10位。1位表示一个hart。
- hartselhi：可读可写，当前选择的hart的高10位。1位表示一个hart。如果只有一个hart，那么hasel的值为0，hartsello的值为1，hartselhi的值为0。
- setresethaltreq：只能写1，写1表示当前选择的hart复位后处于harted状态。
- clrresethaltreq：只能写1，写1表示清除setresethaltreq的值。
- ndmreset：可读可写，写1表示复位整个系统，写0表示撤销复位。
- dmactive：可读可写，写0表示复位DM模块，写1表示让DM模块正常工作。正常调试时，此位必须为1。

2.2.3 DM状态寄存器(dmstatus，0x11)

dmstatus寄存器是一个只读寄存器，每一位含义如图10所示。

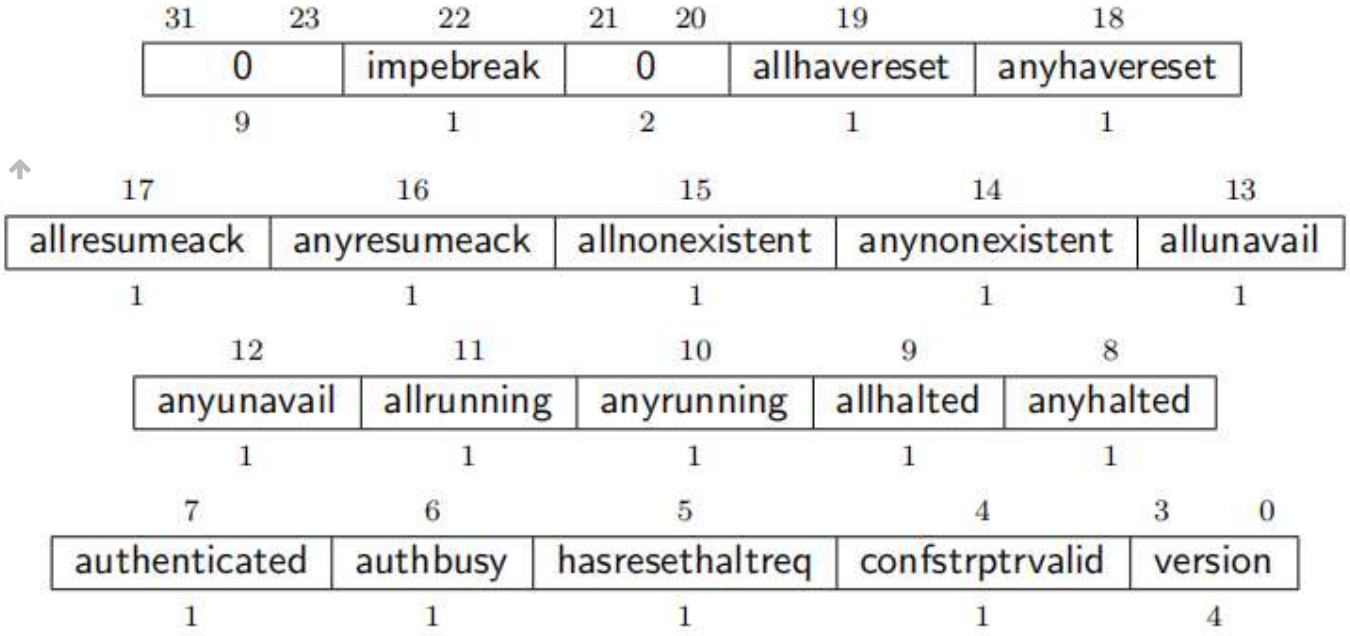

图10 dmstatus寄存器

- impebreak：1表示执行完progbuf的指令后自动插入一条ebreak指令，这样就可以节省一个progbuf。当progbufsize的值为1时，此值必须为1。
- allhavereset：1表示当前选择的hart已经复位。
- anyhavereset：1表示当前选择的hart至少有一个已经复位。
- allresumeack：1表示当前选择的所有hart已经应答上一次的resume请求。
- anyresumeack：1表示当前选择的hart至少有一个已经应答上一次的resume请求。
- allnonexistent：1表示当前选择的hart不存在于当前平台。
- anynonexistent：1表示至少有一个选择了的hart不存在于当前平台。
- allunavail：1表示当前选择的hart都不可用。
- anyunavail：1表示至少有一个选择了的hart不可用。
- allrunning：1表示当前选择的hart都处于running状态。
- anyrunning：1表示至少有一个选择了的hart处于running状态。
- allhalted：1表示当前选择的hart都处于halted状态。

- anyhalted：1表示至少有一个选择了的hart处于halted状态。
- authenticated：0表示使用DM模块之前需要进行认证，1表示已经通过认证。
- authbusy：0表示可以进行正常的认证，1表示认证处于忙状态。
- hasresethaltreq：1表示DM模块支持复位后处于halted状态，0表示不支持。
- confstrptrvalid：1表示confstrptr0~3寄存器保存了配置字符串的地址。
- version：0表示DM模块不存在，1表示DM模块的版本为0.11，2表示DM模块的版本为0.13。

2.2.4 abstract控制和状态寄存器(abstractcs，0x16)

abstractcs寄存器定义如图11所示。

31 29	28 24	23 13	12	11	10 8	7 4	3 0
0	progbufsize	0	busy	0	cmderr	0	datacount
3	5	11	1	1	3	4	4

图11 abstractcs寄存器

- progbufsize：只读，program buffer的个数，取值范围为0~16，每一个的大小为32位。
- busy：只读，1表示abstract命令正在执行，当写command寄存器后该位应该马上被置位直到命令执行完成。
- cmderr：可读、只能写1，cmderr的值仅当busy位为0时有效。0表示无错误，1表示正在操作command、abstractcs、data或者progbuf寄存器，2表示不支持当前命令，3表示执行命令时出现异常，4表示由于当前hart不可用，或者不是处于halted/running状态而不能被执行，5表示由于总线出错(对齐、访问大小、超时)导致的错误，7表示其他错误。写1清零cmderr。
- datacount：只读，所实现的data寄存器的个数。

2.2.5 abstract命令寄存器(command，0x17)

当写这个寄存器时，相应的操作就会被执行。command寄存器只能写，定义如图12所示。

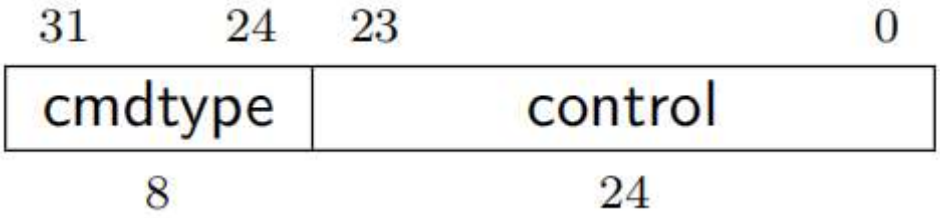

图12 command寄存器

- cmdtype：只写，命令类型，0为表示访问寄存器，1表示快速访问，2表示访问内存。
- control：只写，不同的命令类型有不同的含义，说明如下。

当cmdtype为0时，control定义如图13所示。

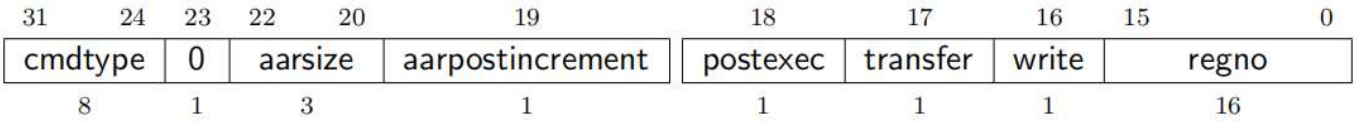

图13 访问寄存器

- cmdtype：值为0。
- aarsize：2表示访问寄存器的最低32位，3表示访问寄存器的最低64位，4表示访问寄存器的最低128位。如果大于实际寄存器的大小则此次访问是失败的。
- aarpostincrement：1表示成功访问寄存器后自动增加regno的值。
- postexec：1表示执行progbuf里的内容(指令)。
- transfer：0表示不执行write指定的操作，1表示执行write指定的操作。
- write：0表示从指定的寄存器拷贝数据到arg0指定的data寄存器。1表示从arg0指定的data寄存器拷贝数据到指定的寄存器。
- regno：要访问的寄存器。

综上，可知：

1. 当write=0，transfer=1时，从regno指定的寄存器拷贝数据到arg0对应的data寄存器。

2. 当write=1，transfer=1时，从arg0对应的data寄存器拷贝数据到regno指定的寄存器。

3. 当aarpostincrement=1时，将regno的值加1。

4. 当postexec=1时，执行progbuf寄存器里的指令。

arg对应的data寄存器如图14所示。

Argument Width	arg0/return value	arg1	arg2
32	data0	data1	data2
64	data0, data1	data2, data3	data4, data5
128	data0–data3	data4–data7	data8–data11

图14 arg对应的data寄存器

即当访问的寄存器位数为32位时，arg0对应data0寄存器，arg1对应data1寄存器，arg2对应data2寄存器。

当cmdtype为1时，control定义如图15所示。

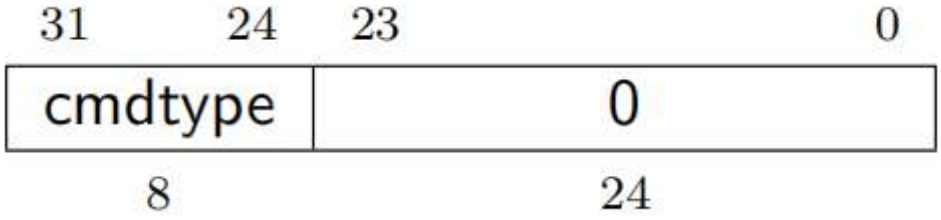

图15 快速访问

- cmdtyte：值为1。

此命令会执行以下操作：

1. halt住当前hart。

2. 执行progbuf寄存器里的指令。

3. resume当前hart。

当cmdtype为2时，control定义如图16所示。

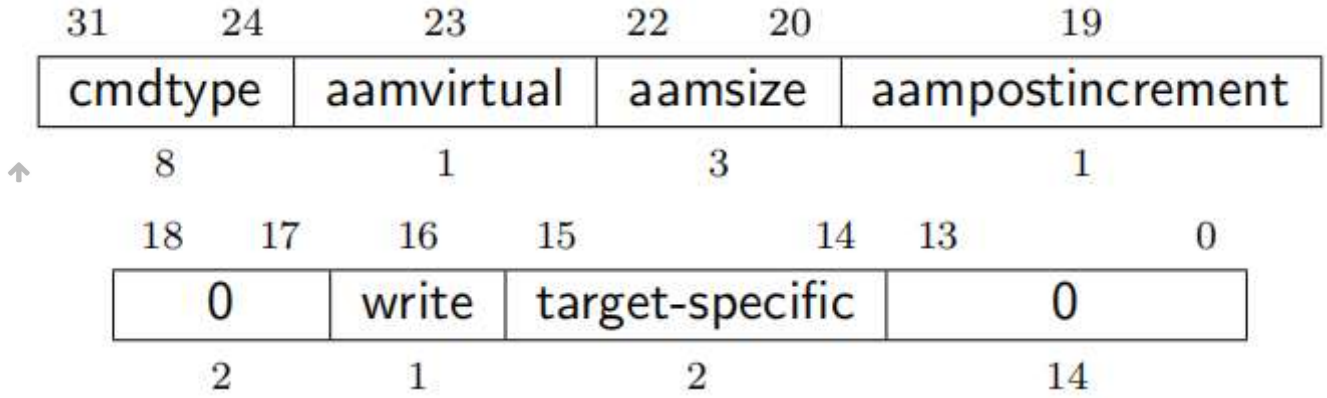

图16 访问内存

- cmdtype：值为2。
- aamvirtual：0表示访问的是物理地址，1表示访问的是虚拟地址。
- aamsize：0表示访问内存的低8位，1表示访问内存的低16位，2表示访问内存的低32位，3表示访问内存的低64位，4表示访问内存的低128位。
- aampostincrement：1表示访问成功后，将arg1对应的data寄存器的值加上aamsize对应的字节数。
- write：0表示从arg1指定的地址拷贝数据到arg0指定的data寄存器，1表示从arg0指定的data寄存器拷贝数据到arg1指定的地址。
- target-specific：保留。

综上，可知：

1. 当write=0时，从arg1指定的地址拷贝数据到arg0指定的data寄存器。
2. 当write=1时，从arg0指定的data寄存器拷贝数据到arg1指定的地址。
3. 当aampostincrement=1时，增加arg1对应的data寄存器的值。

2.2.6 系统总线访问控制和状态寄存器(sbcs，0x38)

sbcs寄存器定义如图17所示。

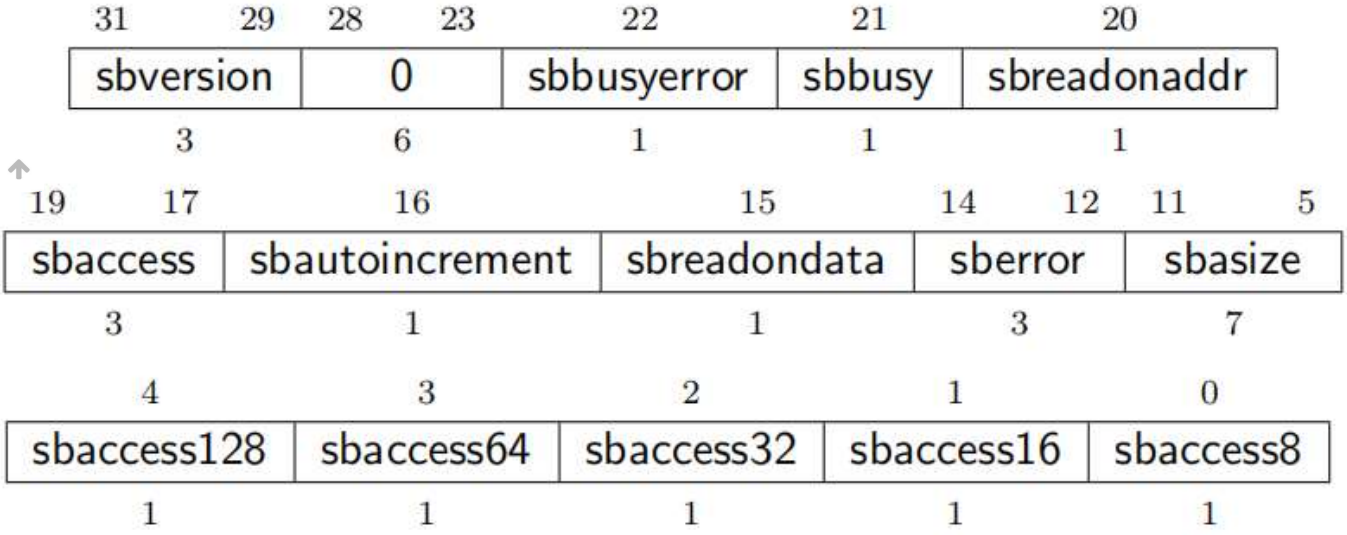

图17 sbcs寄存器

- sbversion：只读，0表示system bus是2018.1.1之前的版本，1表示当前debug spec的版本，即0.13版本。
- sbbusyerror：只读，写1清零，当debugger要进行system bus访问操作时，如果上一次的system bus访问还在进行中，此时会置位该位。
- sbbusy：只读，1表示system bus正在忙。在进行system bus访问前必须确保该位为0。
- sbreadonaddr：可读可写，1表示每次往sbaddress0寄存器写数据时，将会自动触发system bus从新的地址读取数据。
- sbaccess：可读可写，访问的数据宽度，0表示8位，1表示16位，2表示32位，3表示64位，4表示128位。
- sbautoincrement：可读可写，1表示每次system bus访问后自动将sbaddress的值加上sbaccess的大小(字节)。
- sbreadondata：可读可写，1表示每次从sbdata0寄存器读数据后将自动触发system bus从新的地址读取数据。
- sberror：可读，写1清零，0表示无错误，1表示超时，2表示访问地址错误，3表示地址对齐错误，4表示访问大小错误，7表示其他错误。
- sbasize：只读，system bus地址宽度(位数)，0表示不支持system bus访问。

- sbaccess128：只读，1表示system bus支持128位访问。
- sbaccess64：只读，1表示system bus支持64位访问。
- sbaccess32：只读，1表示system bus支持32位访问。
- sbaccess16：只读，1表示system bus支持16位访问。
- sbaccess8：只读，1表示system bus支持8位访问。

2.2.7 系统总线地址0寄存器(sbaddress0，0x39)

可读可写，如果sbcs寄存器中的sbasize的值为0，那么此寄存器可以不用实现。

当写该寄存器时，会执行以下流程：

1. 设置sbcs.sbbusy的值为1。
2. 从新的sbaddress地址读取数据。
3. 如果读取成功并且sbcs.sbautoincrement的值为1，则增加sbaddress的值。
4. 设置sbcs.sbbusy的值为0。

2.2.8 系统总线数据0寄存器(sbdata0，0x3c)

可读可写，如果sbcs寄存器中的所有sbaccessxx的值都为0，那么此寄存器可以不用实现。

当写该寄存器时，会执行以下流程：

1. 设置sbcs.sbbusy的值为1。
2. 将sbdata的值写到sbaddress指定的地址。
3. 如果写成功并且sbcs.sbautoincrement的值为1，则增加sbaddress的值。
4. 设置sbcs.sbbusy的值为0。

当读该寄存器时，会执行以下流程：

1. 准备返回读取的数据。

2. 设置sbcs.sbbusy的值为1。

3. 如果sbcs.sbautoincrement的值为1，则增加sbaddress的值。

4. 如果sbcs.sbreadondata的值为1，则开始下一次读操作。

5. 设置sbcs.sbbusy的值为0。

3.RISC-V调试上位机分析

RISC-V官方支持的调试器上位机是openocd。openocd是地表最强大(没有之一)的开源调试上位机，支持各种target(ARM(M、A系列)、FPGA、RISC-V等)，支持各种调试器(Jlink、CMSIS-DAP、FTDI等)，支持JTAG和SWD接口。

这里不打算详细分析整个openocd的实现，只是重点关注针对RISC-V平台的初始化、读写寄存器和读写内存这几个流程。

3.1 openocd启动过程

openocd启动时需要通过-f参数制定一个cfg文件，比如：

```
openocd.exe -f riscv.cfg
```

riscv.cfg文件的内容如下：

```
1  adapter_khz     1000
2  reset_config srst_only
3  adapter_nsrst_assert_width 100
4  interface cmsis-dap
5  transport select jtag
6  set _CHIPNAME riscv
7  jtag newtap $_CHIPNAME cpu -irlen 5 -expected-id 0x1e200a6d
8  set _TARGETNAME $_CHIPNAME.cpu
9  target create $_TARGETNAME riscv -chain-position $_TARGETNAME
```

- 第一行设置TCK的时钟为1000KHz。

- 第二行表示不支持通过TRST引脚复位，只支持TMS为高电平并持续5个TCK时钟这种方式的复位。
- 第三行是复位持续的延时。
- 第四行指定调试器为CMSIS-DAP。
- 第五行指定调试接口为JTAG。
- 第六行指定调试的target类型为riscv。
- 第七行指定生成一个IR寄存器长度为5位、IDCODE为0x1e200a6d的JTAG TAP。
- 第八、九行指定生成一个riscv target。

openocd启动时的主要流程如图18所示。

图18 openocd启动流程

下面重点关注一下examine target这个流程。

这里的target是指riscv，对于riscv，首先会读取dtmcontrol这个寄存器，因为openocd支持0.11和0.13版本的DTM，通过这个寄存器可以知道当前调试的DTM是哪一个版本。这里选择0.13版本来分析。通过读取dtmcontrol，还可以知道idle、abits这些参数。接下来会将dmcontrol这个寄存器的dmactive域写0后再写1来复位DM模块。接下来再读取dmstatus，判断version域是否为2。接下来还会读取sbcs和abstractcs寄存器，最后就是初始化每一个hart的寄存器。

3.2 read register过程

读寄存器时，先构建command寄存器的内容，首先将cmdtype的值设为0，aarsize的值设为2(寄存器的宽度为32位)，transfer的值设为1，regno的值设为要读的寄存器的number，其他值设为0，然后写到command寄存器里。然后一直读取abstractcs寄存器，直到abstractcs寄存器的busy位为0或者超时。然后再判断abstractcs寄存器的cmderr的值是否为0，如果不为0则表示此次读取寄存器失败，如果为0则继续读取data0寄存器，这样就可以得到想要读的寄存器的值。

3.3 write register过程

写寄存器时，先将需要写的值写到data0寄存器，然后构建command寄存器的内容，首先将cmdtype的值设为0，aarsize的值设为2(寄存器的宽度为32位)，transfer的值设为1，write的值设为1，regno的值设为要写的寄存器的number，其他值设为0，然后写到command寄存器里。然后一直读取abstractcs寄存器，直到abstractcs寄存器的busy位为0或者超时。然后再判断abstractcs寄存器的cmderr的值是否为0，如果不为0则表示此次写寄存器失败，如果为0则表示写寄存器成功。

3.4 read memory过程

如果progbufsize的值大于等于2，则会优先使用通过执行指令的方式来读取内存。这里不分析这种方式，而是分析使用system bus的方式。通过前面的分析可知，system bus有两个版本V0和V1，这里以V1版本来说明。

先将sbcs寄存器的sbreadonaddr的值设为1，sbaccess的值设为2(32位)，然后将要读内存的地址写入sbaddress0寄存器。接着读sbdata0寄存器，最后读sbcs寄存器，如果其中的sbbusy、sberror和sbbusyerror都为0，则从sbdata0读取到的内容就是要读的内存的值。

3.5 write memory过程

和read memory类似，同样以V1版本来说明。

先将要写的内存地址写到sbaddress0寄存器，然后将要写的数据写到data0寄存器，最后读sbcs寄存器，如果其中的sbbusy、sberror和sbbusyerror都为0，则此次写内存成功。

4.RISC-V JTAG的实现

通过在STM32F103C8T6上实现(模拟)RISC-V调试标准，进一步加深对RISC-V JTAG调试的理解。

使用STM32的四个GPIO作为JTAG信号的四根线，其中TCK所在的引脚设为外部中断，即上升沿和下降沿触发方式，实现了可以通过openocd以RISC-V的调试标准来访问STM32的寄存器和内存。程序流程如图19所示，完整的工程代码见[2]。verilog的实现见[3]。

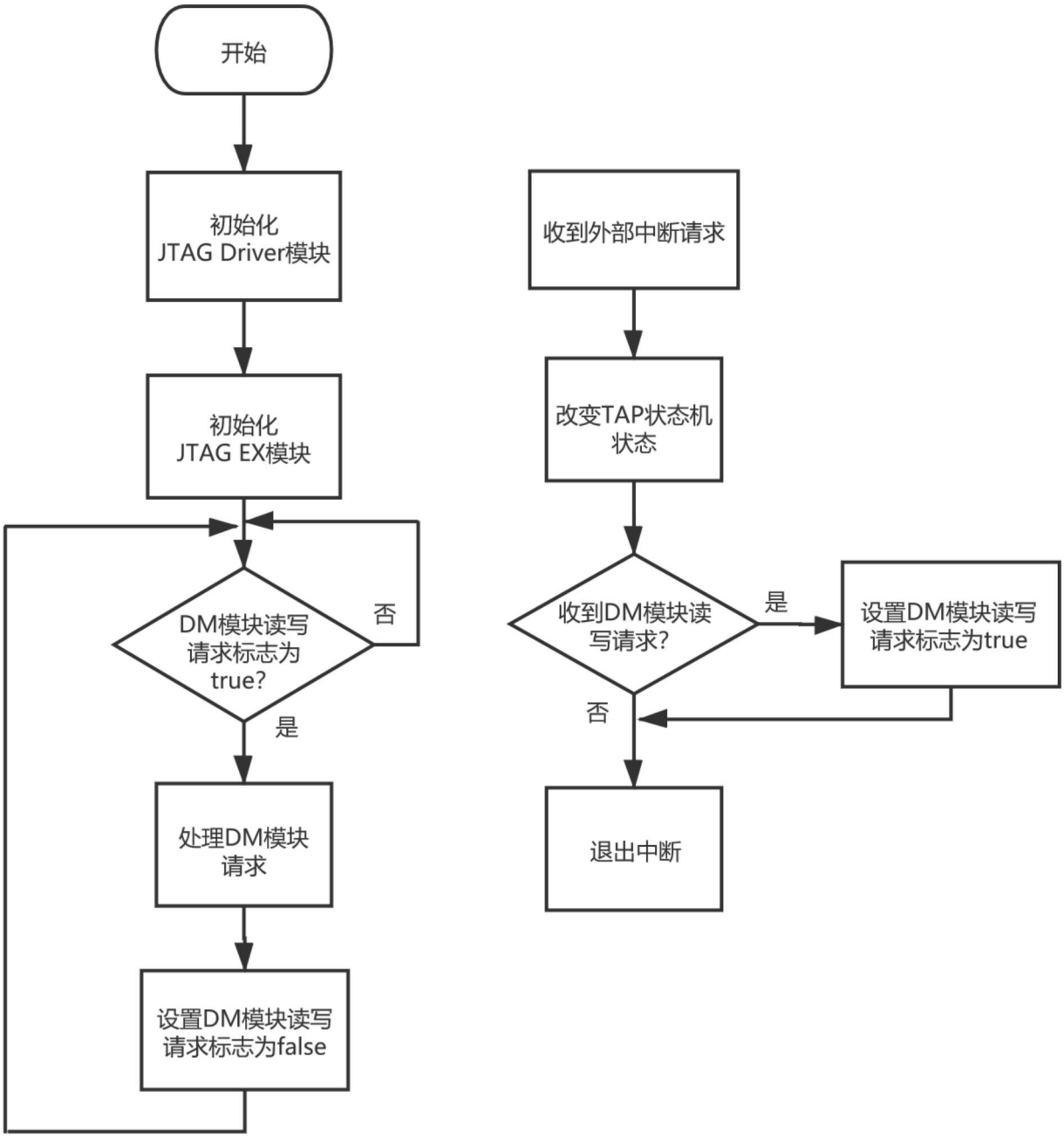

图19 JTAG实现的程序流程

5.参考资料

1. RISC-V External Debug Support Version 0.13。

2. 在STM32上模拟RISC-V JTAG的实现：stm32_riscv_jtag_slave。

3. 一个从零开始写的易懂的RISC-V处理器核：tinyriscv。

RISC-V # 调试

从零开始写RISC-V处理器 >