、

|  |  |
| --- | --- |
| Lausanne Micro Architecture Spec | |
| [备注] | |
|  |  |
| Security Classification | Confidential |
|  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Data | Author | Version | Comment |
| 2021-07-06 | Zhu rong | 0.1.0 | Initial Version. |
| 2021-07-08 | Zhu rong | 0.1.1 | 修改了会议纪要上的内容 |
| 2021-07-09 | Zhu rong | 0.1.2 | 增加了框图中信号线的标注 |
| 2021-07-16 | Zhu rong | 0.1.3 | 改正了时序，做法是将传入alu的PC来源改自if。更新了时序图与整体框图。修改了if、pc、alu的接口信号。 |
| 2021-07-16 | Zhu rong | 0.1.4 | 修改if模块和biu的接口。 |
| 2021-07-18 | Zhu rong | 0.1.5 | 增加了if、pc、de的框图并且标红了修改的地方。 |
| 2021-07-27 | Xu Suki | 0.1.6 | 上传在线编辑文档。 |
| 2021-07-27 | Zhu rong | 0.1.7 | 更改了if的名称为inst\_fetch，修正了inst\_fetch、pc、de的框图，修正了信号名变更后和rtl一致，去掉了inst\_fetch中的rstn信号和功能。 |
| 2021-07-27 | modi | 0.1.8 | 增添ALU、GPR模块MAS |
| 2021-08-01 | fengyi | 0.1.8 | 添加int中断模块 |
| 2021-08-03 | kasi | 0.1.9 | 添加JTAG debug |
| 2021-08-09 | modi | 0.2.0 | 增添WB、CSR模块MAS，修改GPR模块的端口 |

Lausanne Micro Architecture Spec

# Block CPU Overview

## Block CPU General Description

CPU模块的功能是实现RISC-V架构中的RV32I指令集。

## System-Level Block Diagram

如下图所示，流水线主要由两级组成，分别为取令和译码访存执行写回。



## Features

* CPU中包含了IFU、DE、GPR、CONTROL、BRANCH、LSU、WB、PC、BIU、INT、JTAG、CSR。
* CORE中包含了IFU、DE、GPR、CONTROL、BRANCH、LSU、WB、PC、CSR。
* CORE所需要的数据都是和biu进行交互获得的。
* 整个CPU属于一个时钟域下。
* CORE中除了IFU、PC以及GPR、CSR写入功能是时序逻辑，其他模块均是组合逻辑电路。
* 二级顺序流水线，顺序取值译码执行访存写回。（上一条指令完全执行完毕后才能执行下一条指令）
* JTAG介入方式：上一条指令完成后才能介入。
* 中断介入方式：上一条指令完成后才能介入。

### Unsupported Features

* 支持用户指令扩展（先不实现）
* 支持硬件乘法器（先不实现）
* 支持iCache（先不实现）
* 支持ITCM和DTCM（先不实现）
* 支持FastIO（先不实现）
* 支持单精度和双精度FPU（先不实现）
* 支持P扩展（先不实现）

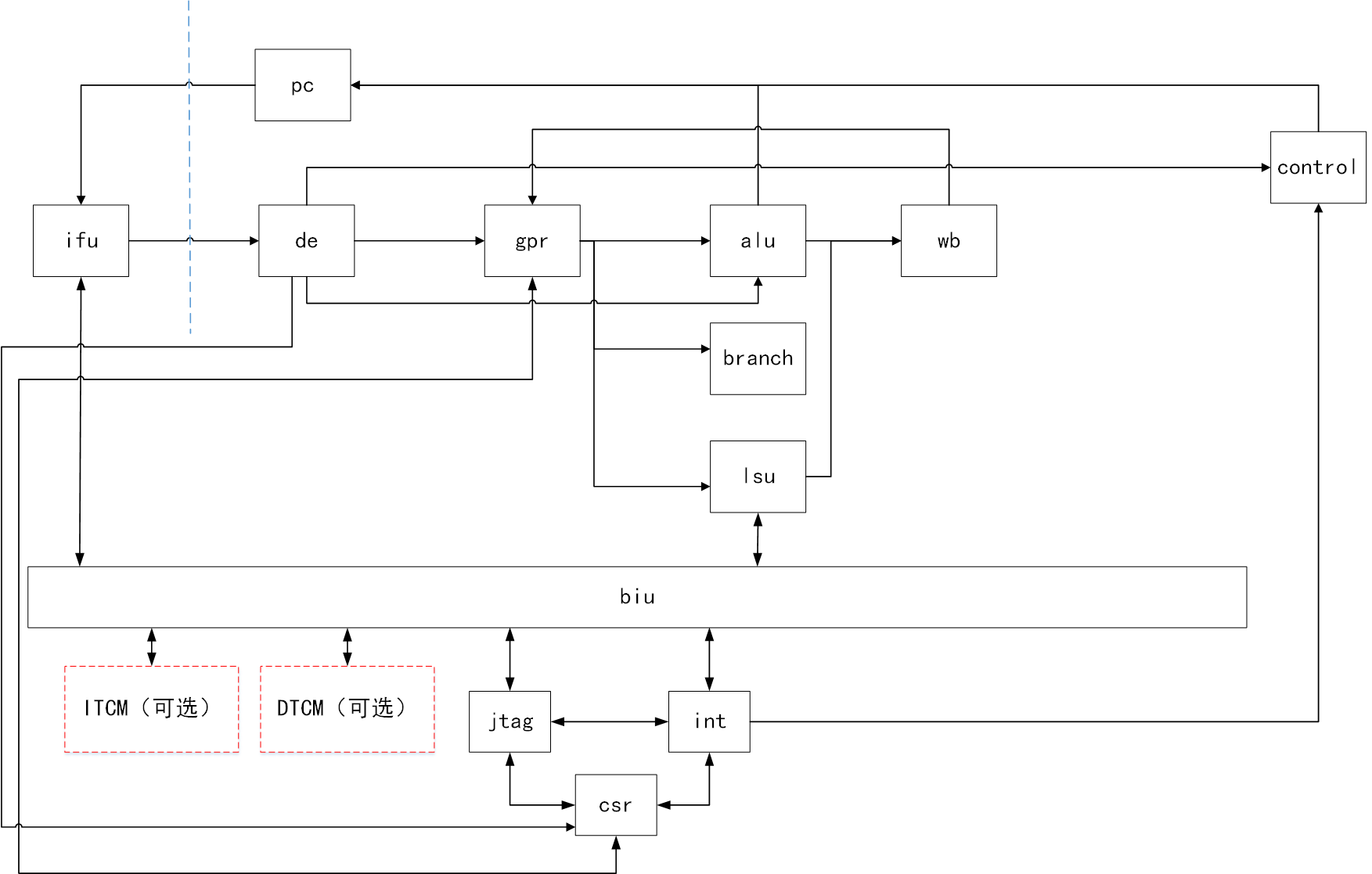
## Supported Standards

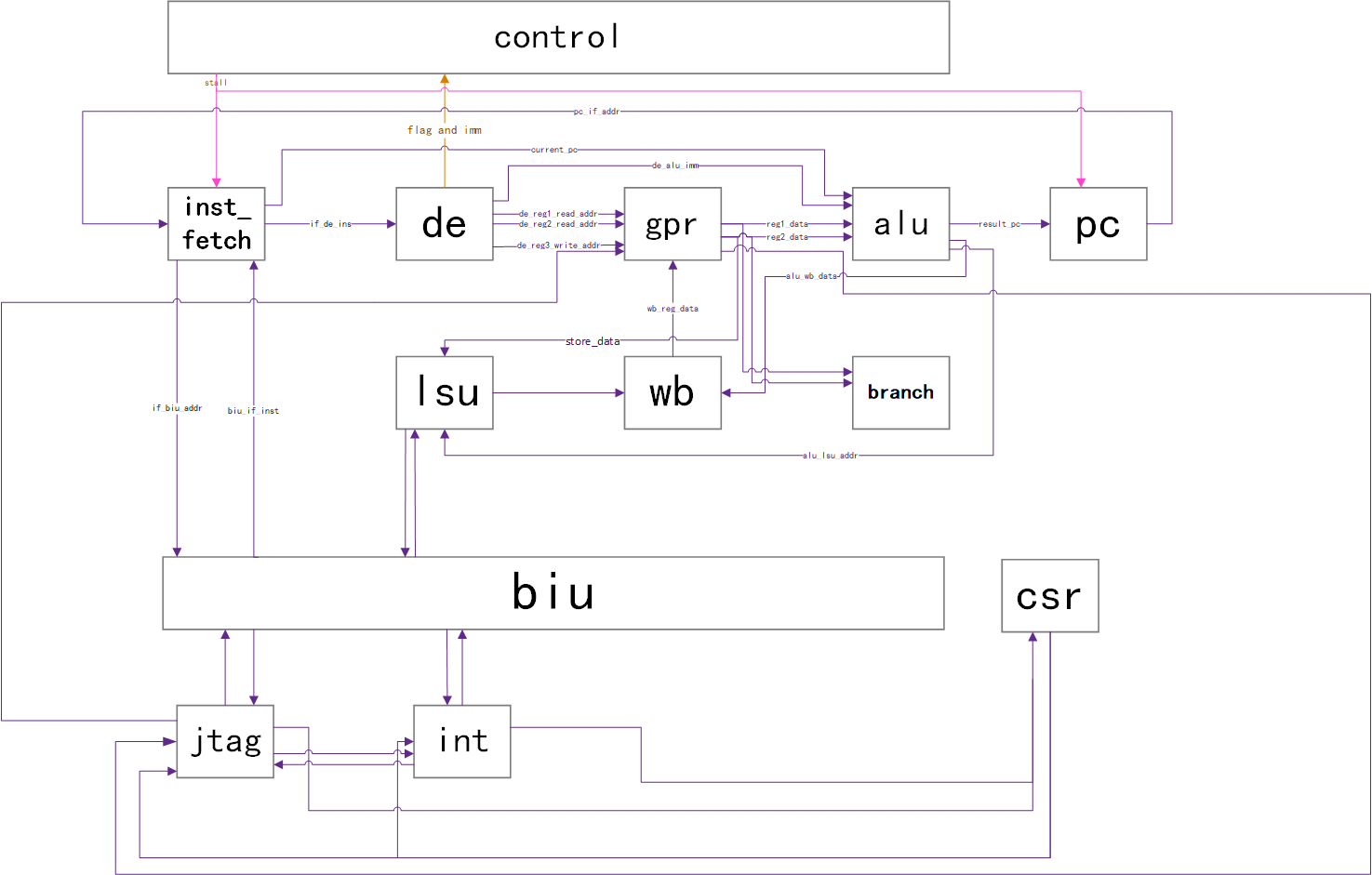
* 指令集：RV32E/IMAFDPC（先实现RV32I）
* 流水线：两级流水线
* 支持machine mode和user mode（先实现machine mode）
* 支持中断控制器（先实现基本功能，然后实现中断咬尾，中断向量化，基于优先级的中断嵌套）
* 支持JTAG调试接口（0.13版本）

# Architecture

## Overview of Architecture

### Block Diagram





### Block Descriptions

CPU模块完成以下功能:在取指模块内，我们将按照PC寄存器中传入的PC值去向总线要指令，总线再将取到的指令传回来，所取到的指令会传递给流水线缓冲器，PC寄存器会进行PC+4运算。其中，流水线缓冲器的作用为寄存当前取回来的数据，并负责执行流水线冲刷操作。译码模块会根据架构规范对指令进行译码以及根据不同的指令生成不同的立即数。译码出来的数据会直接传入通用寄存器模块，该模块会根据指令，取出相应的数据传入，分支判断模块以及ALU模块。如果传入的是跳转判断指令，分支判断模块会判断是否跳转，如果指令是直接跳转，跳转所需要的PC值就需要ALU运算后给出后传入判断模块后再传入PC寄存器，并冲刷流水线。如果是正常的运算指令，我们会使用ALU进行计算，计算后的结果会传入写回模块，进行写回寄存器操作。

如果执行的是访存操作，计算模块需要计算生成所需要的访存地址，如果是lw操作，计算模块生成的访存地址会传入访存这个接口模块，这个接口会去访问总线，去向总线要数据，要来的数据会进行写回寄存器的操作。如果是sw指令，计算模块会给访存地址，通用寄存器模块会给出访存的数据，之后将数据写入存储器中。

### Clock and Reset

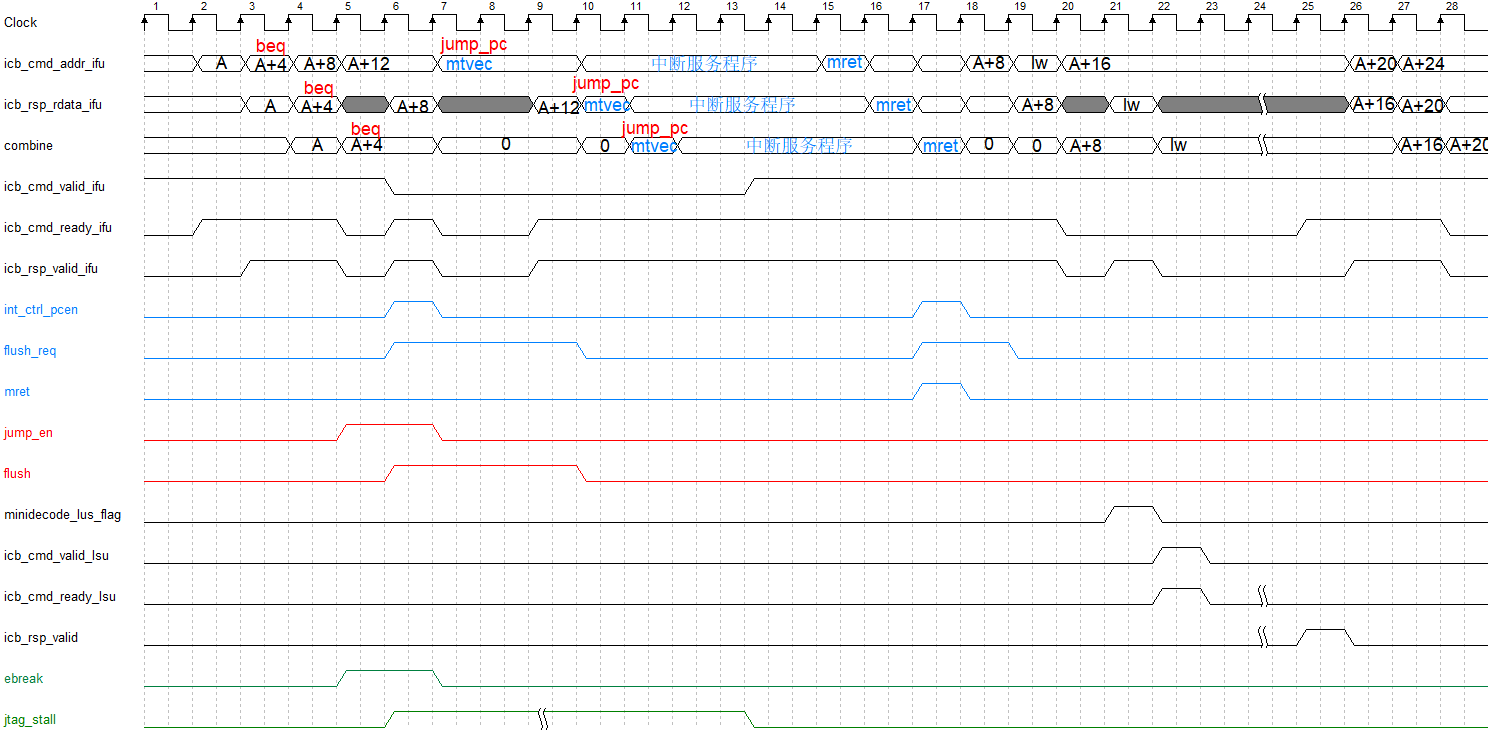
* 整个CPU属于一个时钟域下，除了流水线缓冲器、PC寄存器以及通用寄存器写入功能是时序逻辑，其他模块均是组合逻辑电路。

### Interfaces

|  |  |  |  |
| --- | --- | --- | --- |
| Signal | Direction | Width | Comments |
| clk | Input | 1 | clock |
| rstn | Input | 1 | reset, active low |
| bus\_signal | unknow | 1 | unknow |
| bus\_signal | unknow | 1 | unknow |
| bus\_signal | unknow | 1 | unknow |
| bus\_signal | unknow | 1 | unknow |
| control\_signal | unknow | 1 | unknow |
| control\_signal | unknow | 1 | unknow |
| control\_signal | unknow | 1 | unknow |

### Timing Diagram

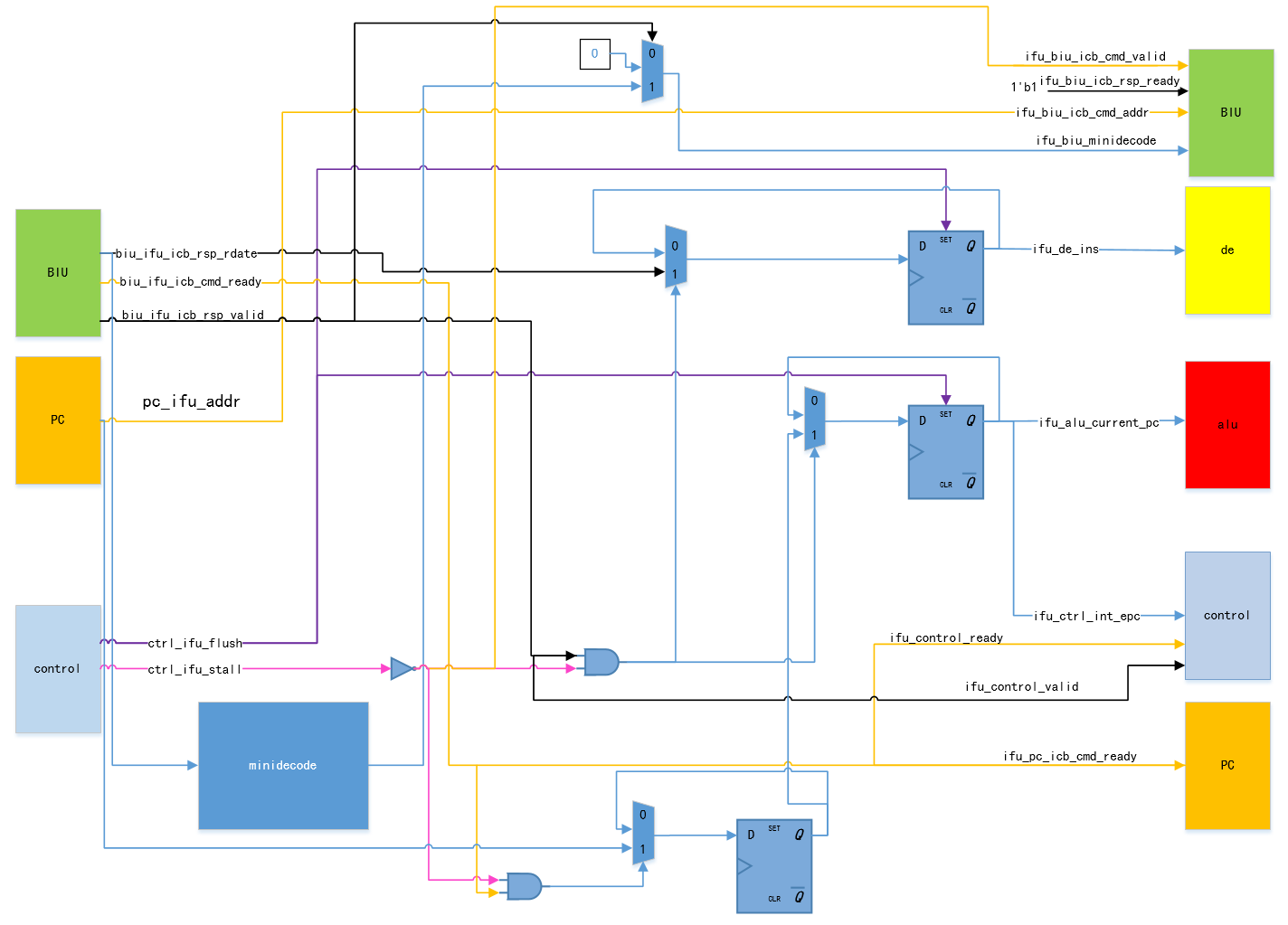
## 



（上图绿线为ebreak情况下的jtag\_stall信号，与其余信号无直接相关）

## Sub Block ifu Architecture

### Block Diagramd

s

### Block Descriptions

ifu 模块完成以下功能:

1.根据pc寄存器输出的pc地址，遵循icb协议和biu总线通信取回指令，并写入流水线寄存器中。

2.处理流水线冲刷，在接受到流水线冲刷信号后重置寄存器。

3.将流水线寄存器中的数据送给下一级流水线。

4.将目前执行的指令的PC值送给alu和控制模块中。

5.预译码出lsu的标志位给biu进行处理。

6.只有当biu\_ifu\_icb\_rsp\_valid为1且没有halt的时候，取回来的指令和正在执行的PC才能写入流水线寄存器。

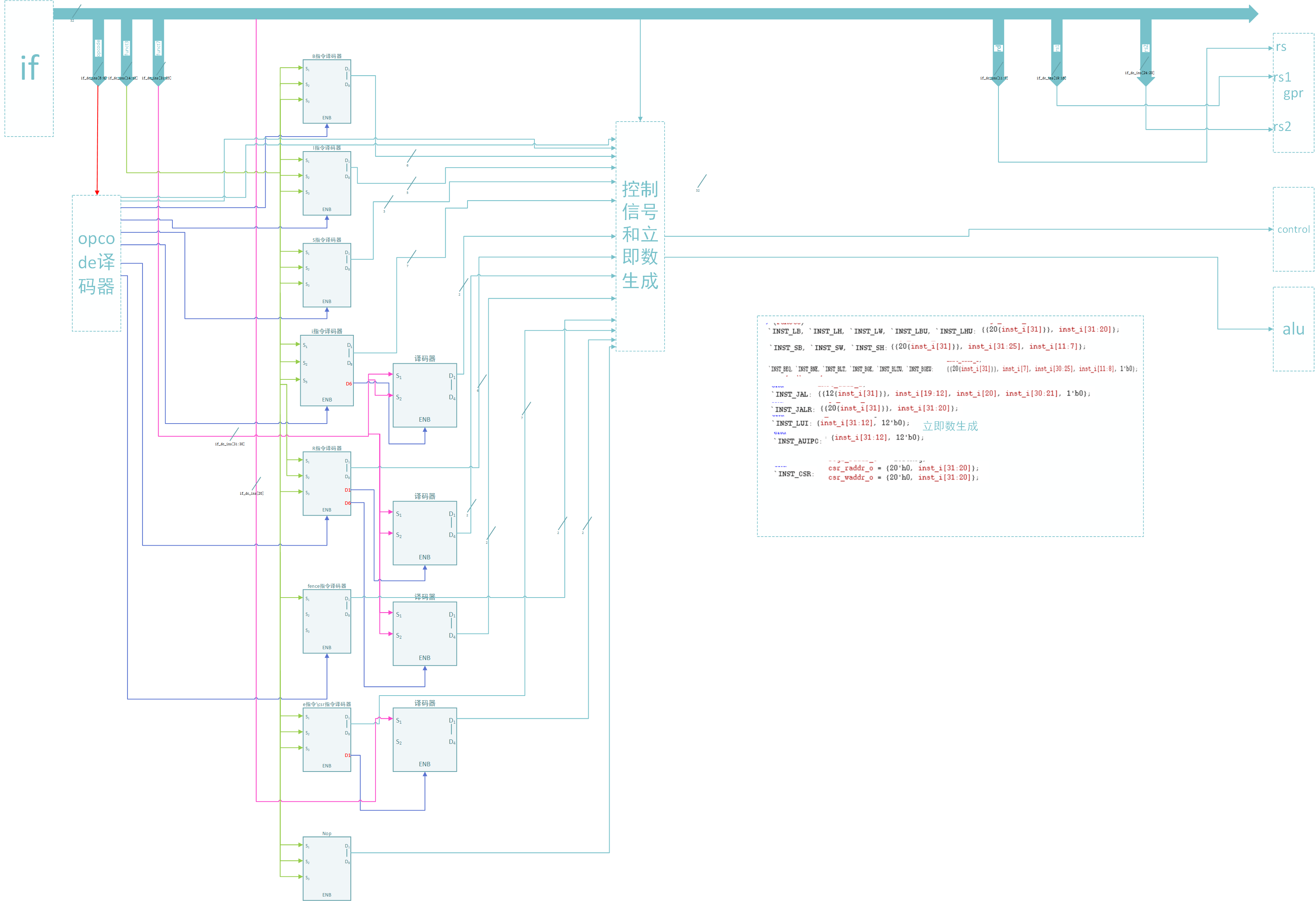
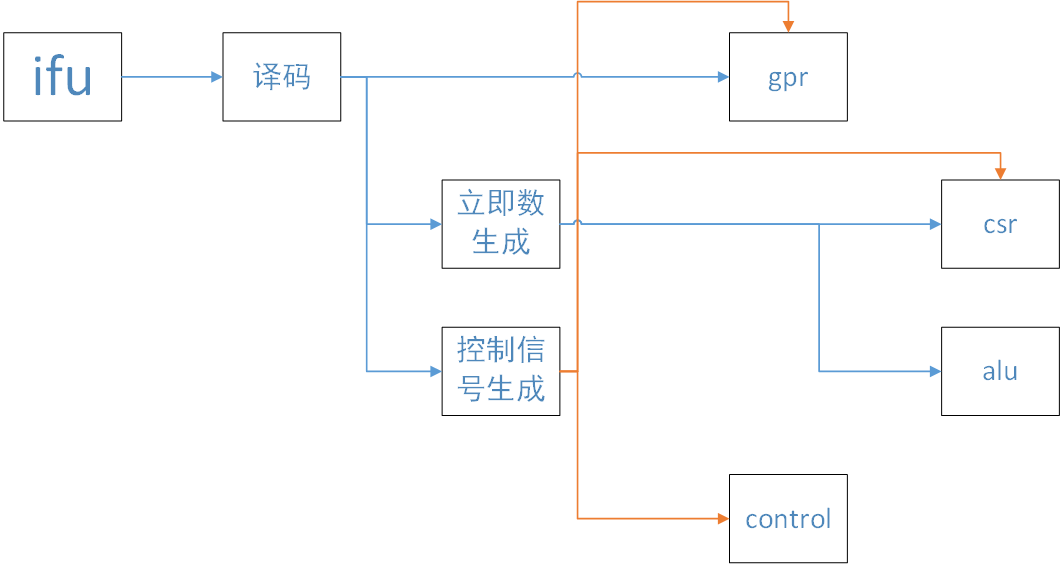
7.例化了decode模块，当前拍就进行预译码，生成lsu指令的flag，这个flag有效的时候，会让lsu抢占biu,biu不会给ifu ready信号，但数据是正常传输的。

### Interfaces

|  |  |  |  |
| --- | --- | --- | --- |
| Signal | Direction | Width | Comments |
| clk | Input | 1 | clock |
| With control | | | |
| ctrl\_ifu\_flush | Input | 1 | 接受冲刷信号（分支冲刷和中断冲刷） |
| ctrl\_ifu\_stall | Input | 1 | 流水线暂停信号（由于总线繁忙和jtag信号暂停） |
| ifu\_ctrl\_int\_epc | output | 32 | 中断信号作用之前的PC地址。（也就是中断信号看到的正在执行的指令）（打两拍之后的pc地址） |
| ifu\_ctrl\_ready | output | 1 | 将biu给的取指ready传递给control。（与biu\_ifu\_icb\_cmd\_ready一致） |
| ifu\_ctrl\_valid | output | 1 | 将biu给的取指内容valid传递给control。（与biu\_ifu\_icb\_rsp\_valid一致） |
| ifu\_ctrl\_minidecode | output | 1 | 告知中断模块目前执行的指令是ls指令。 |
| With pc | | | |
| pc\_ifu\_addr | input | 32 | PC模块通过ifu模块要发送给biu的PC地址。（地址加4） |
| ifu\_pc\_icb\_cmd\_ready | output | 1 | pc模块用这个信号去观测上一条指令是否正常取回，只有上一条取回后才会进行顺序取指的PC变动。（当为1时代表上一条指令是否正常取回，可以加4） |
| With biu | | | |
| biu\_ifu\_icb\_rsp\_rdate | input | 32 | 取到的指令。 |
| ifu\_biu\_icb\_cmd\_addr | output | 32 | 取指地址输出给biu。 |
| biu\_ifu\_icb\_cmd\_ready | input | 1 | icb握手协议，表明可以读取下一条PC。 |
| biu\_ifu\_icb\_rsp\_valid | input | 1 | icb握手协议，表明取回的指令内容数据有效。 |
| ifu\_biu\_icb\_cmd\_valid | output | 1 | icb握手协议，表明目前PC值有效，设计中为暂停流水线信号取反。 |
| ifu\_biu\_icb\_rsp\_ready | output | 1 | icb握手协议，表明接受了biu取回的指令内容，在本设计中恒为一。 |
| ifu\_biu\_minidecode | output | 1 | 用于提前译码控制biu，告知biu此指令是lsu相关指令，抢先占用biu总线。 |
| biu\_ifu\_icb\_cmd\_read | output | 1 | 用于告诉总线是对总线写操作还是读操作 |
| With alu | | | |
| ifu\_alu\_current\_pc | output | 32 | 目前执行指令的PC值，用于计算跳转PC。（打两拍之后的pc地址） |
| With de | | | |
| ifu\_de\_ins | output | 32 | 将biu取回的指令内容打拍输出。（取回的指令打一拍输出给译码模块） |

## Sub Block de Architecture

### Block Diagram



### Block Descriptions

de 模块完成以下功能:

1. 将指令按照架构规范进行解码。
2. 将解码后的数码类型传入控制模块。
3. 根据指令的类型去生成立即数。
4. 根据解码后的读写通用寄存器的地址，去进行读写GPR通用寄存器。
5. 根据解码后的访问csr的地址，去进行访问csr寄存器。

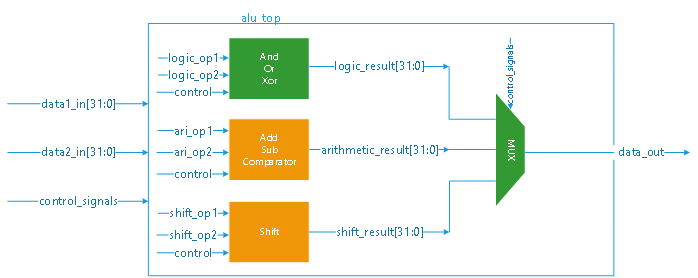
6.支持额外功能识别当前指令是ls相关指令，识别信号供其他模块使用。

### Interfaces

|  |  |  |  |
| --- | --- | --- | --- |
| Signal | Direction | Width | Comments |
| ifu\_de\_ins | Input | 32 | Instruct（取值模块给的打一拍的指令） |
| **With gpr** | | | |
| de\_reg1\_read\_addr | output | 5 | 读寄存器rs1地址 |
| de\_reg2\_read\_addr | output | 5 | 读寄存器rs2地址 |
| de\_reg3\_write\_addr | Input | 5 | 写寄存器rs3地址 |
| de\_gpr\_we | output |  | 写reg使能信号（需要写回的指令的写回信号） |
| **With alu** | | | |
| de\_alu\_imm | output | 32 | 生成的立即数 |
| de\_alu\_l\_flag | output | 1 | 代表是load指令 |
| de\_alu\_s\_flag | output | 1 | 代表是store指令 |
| de\_alu\_ls\_size | output | 1 | 代表 ls的位宽 |
| **With control** | | | |
| de\_ctrl\_reg2\_flg | output | 1 | 1表示运算时使用reg2\_data，0表示使用dc\_alu\_imm，对应的指令有addi，ori，xori，srli，srai，slli，slti，sltiu，lui，addi，auipc,sb,sh,sw,lb,lh,lw,lhu |
| de\_ctrl\_lui\_flg | output | 1 | 1表示LUI操作有效，0表示无效，对应的指令有lui |
| de\_ctrl\_shift\_flg | output | 1 | 1表示移位操作有效，0表示无效，对应的指令有srl，srli，sra，srai，sll，slli，lui |
| de\_ctrl\_shift\_right\_flg | output | 1 | 1表示进行右移操作，0表示左移操作，对应的指令同上 |
| de\_ctrl\_shift\_logic | output | 1 | 1表示进行逻辑移位，0表示进行算术移位，对应的指令同上 |
| de\_ctrl\_comparator\_flg | output | 1 | 1表示比较操作有效，0表示比较操作无效，对应的指令有slt，slti，sltu，sltiu |
| de\_ctrl\_and\_flg | output | 1 | 1表示按位与操作有效，0表示按位或操作无效，对应指令and，andi |
| de\_ctrl\_or\_flg | output | 1 | 1表示按位或操作有效，0表示按位或操作无效，对应的指令有or，ori |
| de\_ctrl\_xor\_flg | output | 1 | 1表示按位异或操作有效，0表示按位异或操作无效，对应的指令有xor，xori |
| de\_ctrl\_adder\_flg | output | 1 | 1表示进行加法操作，0表示加法操作无效，对应的指令有add，addi，sb，sh,sw,lb,lh,lw,lbu,lhu |
| de\_ctrl\_sub\_flg | output | 1 | 1表示进行减法操作，0表示减法操作无效，对应指令有sub |
| de\_ctrl\_auipc\_flg | output | 1 | 1表示进行pc加立即数操作，对应的指令有auipc |
| de\_ctrl\_br\_typ | output | 3 | 一共六种判断beq001、bne010、blt011、bge100、bltu101、bgeu110 |
| de\_ctrl\_lsu\_flg | output | 2 | 判断执行l指令还是s指令，01是l指令、10是s指令、00无效 |
| de\_ctrl\_int\_ebreak | output | 1 | ebreak instruction in IFU |
| de\_ctrl\_int\_ecall | output | 1 | ecall instruction in IFU |
| de\_ctrl\_int\_mret | output | 1 | mret signal flag |
| de\_ctrl\_illegal | output | 1 | 非法指令（进入中断服务程序） |
| **With csr** | | | |
| de\_csr\_zimm | output | 32 | 将高位补0后的zimm传给csr做处理（补０后的ＣＳＲ立即数） |
| de\_csr\_csrrw\_flag | output | 1 | csrrw csrrwi（将ｒｓ１中的数据写ｃｓｒ同时将ｃｓｒ数据写入ｒｄ） |
| de\_csr\_csrrs\_flag | output | 1 | csrrs csrrsi（rs1将CSR寄存器对应位至1操作） |
| de\_csr\_csrrc\_flag | output | 1 | csrrc csrrci（rs1将CSR寄存器对应位至０操作） |
| de\_csr\_imm\_en | output | 1 | csrrwi csrrsi csrrci 有效为1（有符号位拓展的ＣＳＲ指令） |
| de\_csr\_we | output | 1 | csr写使能（确认为CSR指令） |
| de\_csr\_idx | output | 32 | 读写csr的地址（ＣＳＲ寄存器的地址） |
| **额外功能** | | | |
| minidecode\_lsu\_flag | output | 1 | 用于minidecode例化的，在连接单独模块的时候不用关心（ls指令时为1） |

## Sub Block ALU Architecture

### Block Diagram



### Block Descriptions

alu 模块完成以下功能:

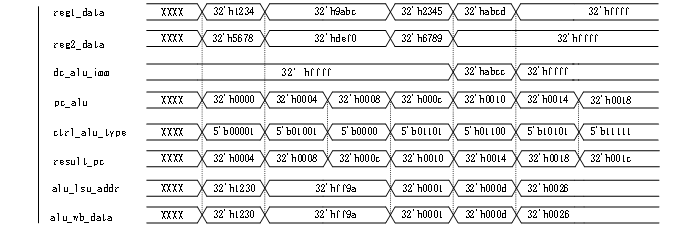
1. 进行RV32I支持的运算，包括SLL、SLLI、SRL、SRLI、SRA、SRAI、ADD、ADDI、SUB、LUI、AUIPC、XOR、XORI、OR、ORI、AND、ANDI、SLT、SLTI、SLTU、SLTIU

|  |  |
| --- | --- |
| 操作类型 | 数据处理细节 |
| AND | r1与r2位与，结果输出到两个输出端口，alu\_wb\_data，alu\_lsu\_addr，下面操作的结果输出相同。 |
| ANDI | r1与立即数位与 |
| OR | r1和r2按位取或 |
| ORI | r1和立即数按位或 |
| XOR | r1和r2按位异或 |
| XORI | r1和立即数异或 |
| SRL | r1右移x位，x为r2的低五位，r2的高位被忽略，空位补0。（逻辑右移） |
| SRLI | r1右移x位，x为立即数的低五位，立即数的高位被忽略，空位补0。（逻辑右移） |
| SRA | r1右移x位，移动位数x等于r2的低五位，r2的高位被忽略，空位用r1最高位填充。（算术右移） |
| SRAI | r1右移x位，移动位数x等于立即数的低五位，立即数的高位被忽略，空位用r1的最高位填充。（算术右移） |
| SLL | r1左移x位，移动的位数x等于r2的低五位，r2的高位被忽略，空位补0。（逻辑左移） |
| SLLI | r1左移x位，x等于立即数的低五位，立即数的高位被忽略，空位补0。（逻辑左移） |
| SLT | 若r1<r2，输出1，否则输出0，有符号比较。 |
| SLTI | 若r1<立即数，输出1，否则输出0，有符号比较。 |
| SLTU | 若r1<r2，输出1，否则输出0，无符号比较。 |
| SLTIU | 若r1<立即数，输出1，否则输出0，比较时视立即数为无符号数。 |
| LUI | 将立即数左移12位，低12位置零。（逻辑右移） |
| ADD | r1的值加上r2的值，忽略算术溢出。 |
| ADDI | r1的值加上立即数的值，结果写到rd，忽略算术溢出。 |
| SUB | r1减去r2，结果写到rd，忽略算术溢出。 |
| AUIPC | 立即数的值加上pc，结果写到rd。 |

### Interfaces

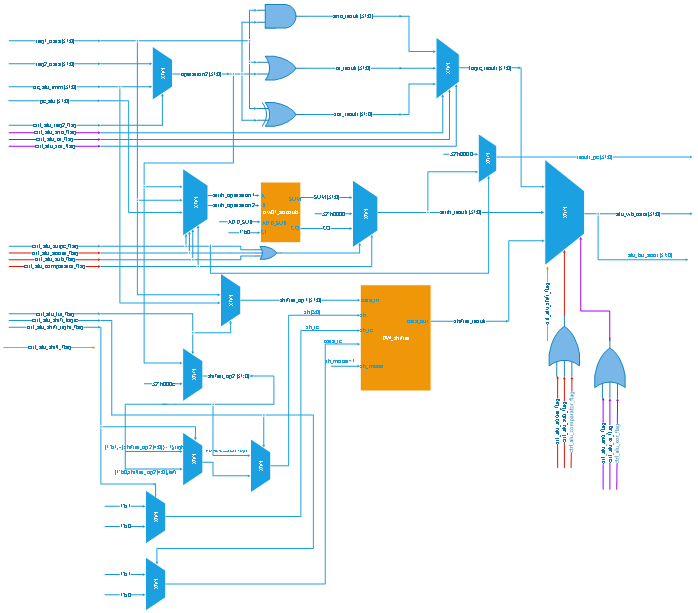
|  |  |  |  |
| --- | --- | --- | --- |
| **Signal** | **Direction** | **Width** | **Comments** |
| reg1\_data | Input | 32 | R1的数据（从ＧＰＲ中将数据读出） |
| reg2\_data | Input | 32 | R2的数据（从ＧＰＲ中将数据读出） |
| pc\_alu | Input | 32 | 目前的pc（打两拍之后的ｐｃ地址） |
| dc\_alu\_imm | Input | 32 | 生成的立即数（生成符号位拓展完成的立即数） |
| dc\_alu\_data\_vld | input | 1 | 数据有效（信号已经不在了） |
| ctrl\_alu\_reg2\_flag | input | 1 | 1表示运算时使用reg2\_data，0表示使用dc\_alu\_imm，对应的指令有addi，ori，xori，srli，srai，slli，slti，sltiu，lui，addi，auipc,sb,sh,sw,lb,lh,lw,lhu |
| ctrl\_alu\_lui\_flag | input | 1 | 1表示LUI操作有效，0表示无效，对应的指令有lui |
| ctrl\_alu\_shift\_flag | input | 1 | 1表示移位操作有效，0表示无效，对应的指令有srl，srli，sra，srai，sll，slli，lui |
| crtl\_alu\_shift\_right\_flag | input | 1 | 1表示进行右移操作，0表示左移操作，对应的指令同上 |
| ctrl\_alu\_shift\_logic | input | 1 | 1表示进行逻辑移位，0表示进行算术移位，对应的指令同上 |
| ctrl\_alu\_comparator\_flag | input | 1 | 1表示比较操作有效，0表示比较操作无效，对应的指令有slt，slti，sltu，sltiu |
| ctrl\_alu\_and\_flag | input | 1 | 1表示按位与操作有效，0表示按位或操作无效，对应指令add，addi |
| ctrl\_alu\_or\_flag | input | 1 | 1表示按位或操作有效，0表示按位或操作无效，对应的指令有or，ori |
| ctrl\_alu\_xor\_flag | input | 1 | 1表示按位异或操作有效，0表示按位异或操作无效，对应的指令有xor，xori |
| ctrl\_alu\_adder\_flag | input | 1 | 1表示进行加法操作，0表示加法操作无效，对应的指令有add，addi，sb,sh,sw,lb,lh,lw,lhu |
| ctrl\_alu\_sub\_flag | input | 1 | 1表示进行减法操作，0表示减法操作无效，对应指令有sub |
| ctrl\_alu\_auipc\_flag | input | 1 | 1表示进行pc加立即数操作，对应的指令有auipc |
| result\_pc | output | 32 | 运算后的pc0（已经优化掉） |
| result\_pc\_vld | output | 1 | result\_pc 数据有效（已经优化掉） |
| alu\_wb\_data | output | 32 | 运算后需要写回的数据（写回的数据） |
| alu\_wb\_data\_vld | output | 1 | alu\_wb\_data数据有效（已被优化） |
| alu\_lsu\_addr | output | 32 | 访存的地址（访存的地址） |
| alu\_lsu\_addr\_vld | output | 1 | alu\_lsu\_addr地址有效（已被优化） |

### Timing Digram



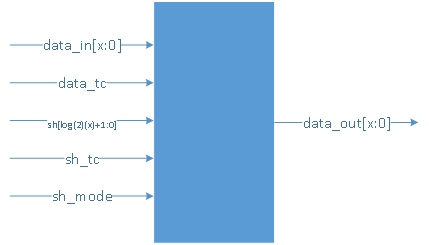
### FSM

### Design detail



### 2.4.7移位模块DW\_shifter

使用已有的模块进行7种移位操作，选用DW\_shifter库，其端口如下图所示，



端口信号说明如下表所示：

|  |  |  |  |
| --- | --- | --- | --- |
| Pin name | Data width | Direction | Function |
| data\_in | data\_width bits | input | input data |
| data\_tc | 1 bits | input | 0:unsigned data\_in  1:signed data\_in |
| sh | sh\_width bits | input | shifter control |
| sh\_tc | 1 bits | input | 0:unsigned sh  1:signed sh |
| sh\_mode | 1 bits | input | 0:barrel shifter  1:arithmetic shifter |
| data\_out | data\_width bits | output | output data |

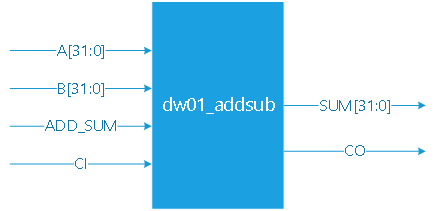
模块内部有一个可配置参数，inv\_mode，默认为0。下表列出inv\_mode的描述：

|  |  |
| --- | --- |
| Inv\_mode | Description |
| 0 | 正常输入，空位补0 |
| 1 | 正常输入，空位补1 |
| 2 | 反向输入，空位补0 |
| 3 | 反向输入，空位补1 |

ALU模块需要的移位操作对应的DW\_shfter真值表：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| operation | inv\_mode | sh\_tc | MSB sh | data\_tc | sh\_mode | operation |
| logic right | 0 | 1 | 1 | 0 | 1 | 右移sh的补码位，sh={1’b1,~{shifter\_operation[4:0]+1} |
| logic left | 0 | 0 | 0 | 0 | 1 | 左移sh的补码位，sh={1’b0,shifter\_operation[4:0]} |
| arith right | 0 | 1 | 1 | 1 | 1 | 右移sh的补码位，=sh={1’b1,~{shifter\_operation[4:0]+1} |

### 算术模块dw01\_addsub



端口说明：

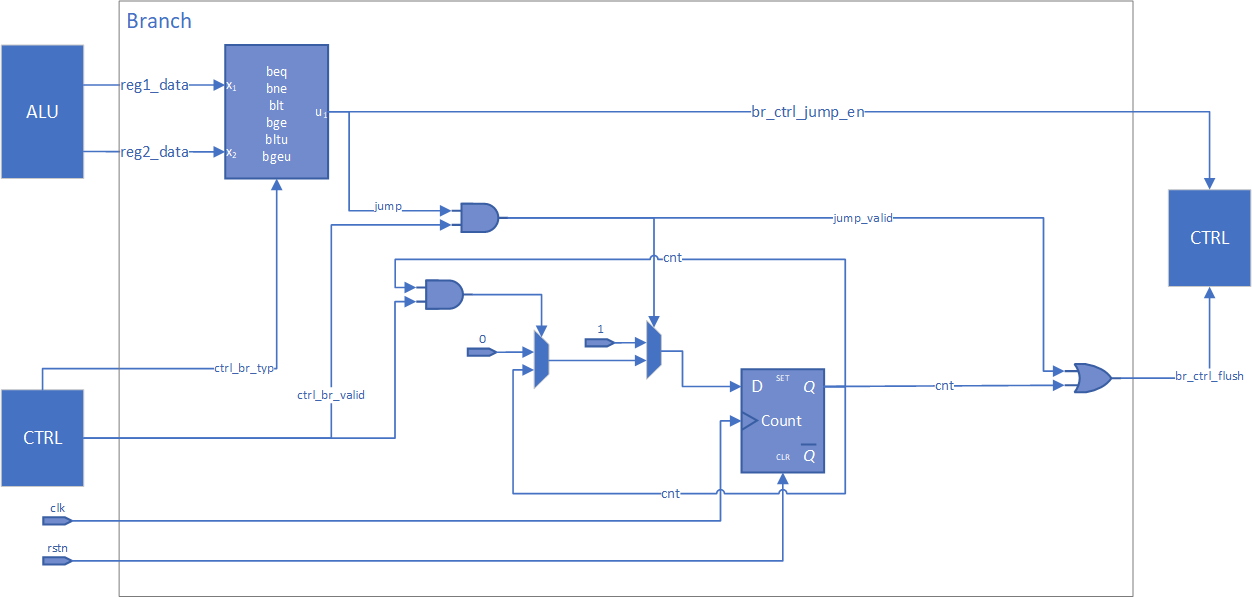
|  |  |  |  |
| --- | --- | --- | --- |
| Pin name | Data width | Direction | Function |
| A | data\_width bits | input | operation1 |
| B | data\_width bits | input | operation2 |
| Ci | 1bits | input | carry\_out from previous |
| ADD\_SUB | 1 bits | input | 0:A+B+CI  1:A-B-CI |
| SUM | data\_width bits | output | result |
| CO | 1bits | output | carry-out |

实现比较器：

如果A-B的进位输出为1，则A<B，否则A>=B

## Sub Block branch Architecture

### Block Diagram



### Block Descriptions

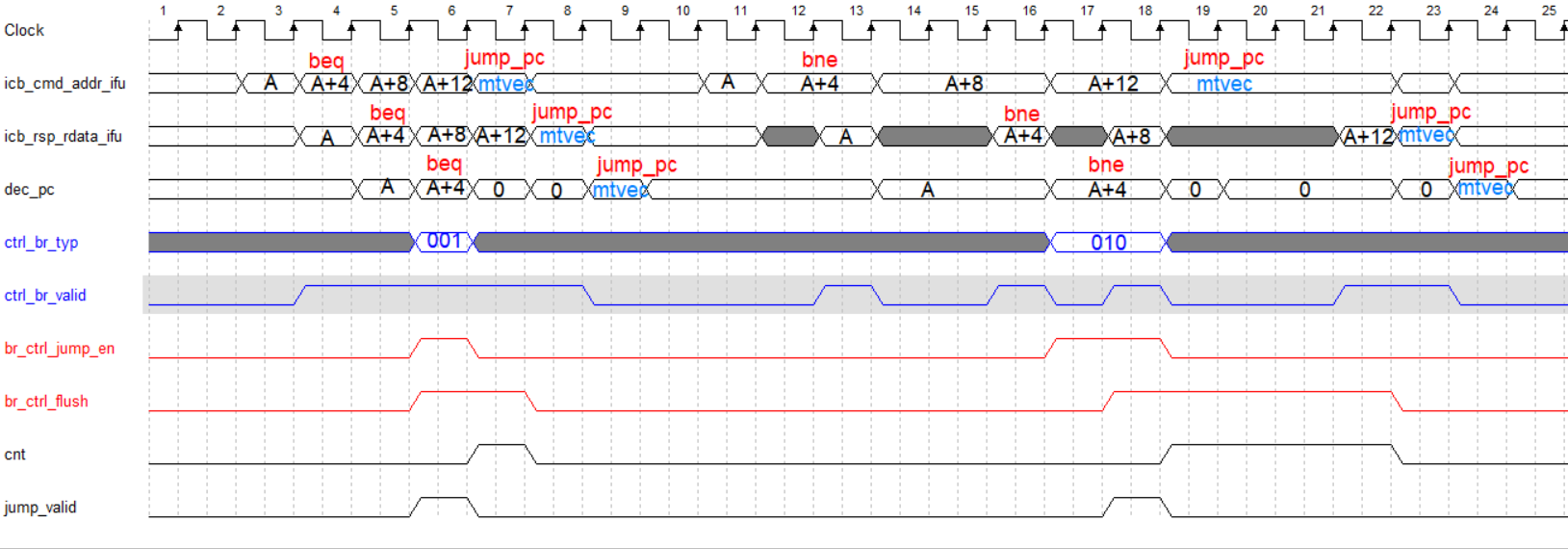
branch 模块完成以下功能:

1. 根据控制模块给出的判断需求以及通用寄存器读出的数据进行判断，进行六种判断中的一种判断，beq001、bne010、blt011、bge100、bltu101、bgeu110，
2. 判断的结果供控制模块使用，同时传回控制模块pc值跳转信号以及冲刷流水线信号。

### Interfaces

|  |  |  |  |
| --- | --- | --- | --- |
| Signal | Direction | Width | Comments |
| clk | Input | 1 | clock |
| rstn | Input | 1 | reset |
| ctrl\_br\_valid | Input | 1 | 与icb\_rsp\_valid\_ifu等同 |
| reg1\_data | Input | 32 | 判断数据1的值 |
| reg2\_data | Input | 32 | 判断数据2的值 |
| ctrl\_br\_typ | Input | 3 | 一共六种判断 |
| br\_ctrl\_jump\_en | output | 1 | 是否发生了跳转 |
| br\_ctrl\_flush | output | 1 | 冲刷跳转指令后的两个指令 |

### Timing digram



## Sub Block GPR Architecture

### Block Diagram



### Block Descriptions

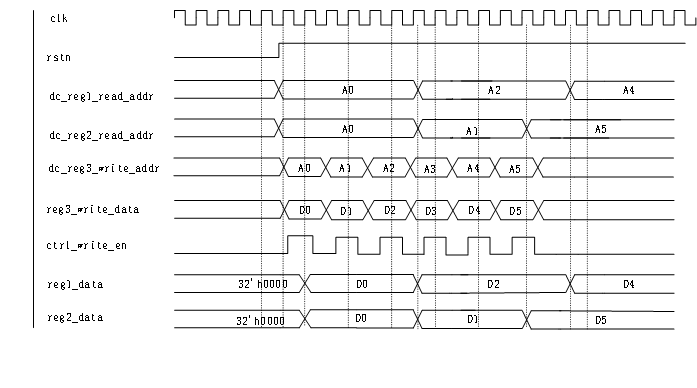
gpr 模块完成以下功能:

1. 根据读寄存器地址，读取寄存器中存储的数据，传给后续模块使用
2. 根据解码出的目的寄存器地址以及写回数据进行写回操作。

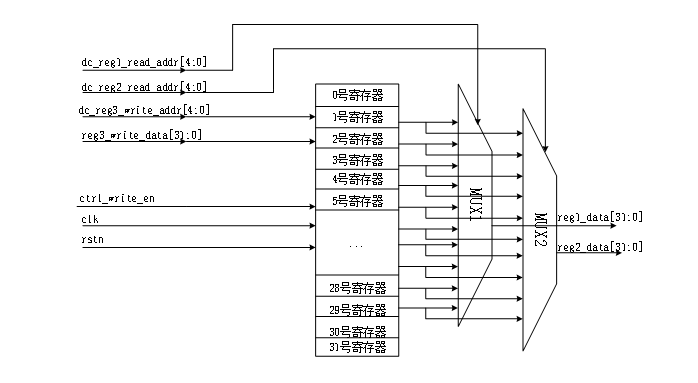
### Interfaces

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal** | **Direction** | **Width** | **Comments** |
| rstn | Input | 1 | reset, active low |
| clk | Input | 1 | clock |
| dc\_reg1\_read\_addr | Input | 5 | 读寄存器rs1地址 |
| dc\_reg2\_read\_addr | Input | 5 | 读寄存器rs2地址 |
| dc\_reg3\_write\_addr | Input | 5 | 写寄存器rs3地址 |
| wb\_gpr\_write\_data | Input | 32 | 写寄存器rs3数据 |
| ctrl\_write\_en | Input | 1 | 写寄存器rs3使能 |
| reg1\_data | output | 32 | 读出寄存器rs1值 |
| reg2\_data | output | 32 | 读出寄存器rs2值 |

### Timing digram



### Design detail



## Sub Block wb Architecture

### Block Diagram

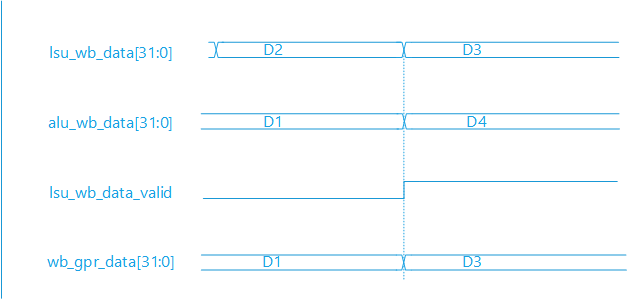
### Block Descriptions

wb 模块完成以下功能:

1. 整理lsu以及alu输入的写回的数据，将之写回到通用寄存器中。
2. lsu的数据有效信号来自lsu模块，alu的数据有效信号来自control模块，跳过wb模块直接送到grp模块由gpr模块判断优先级

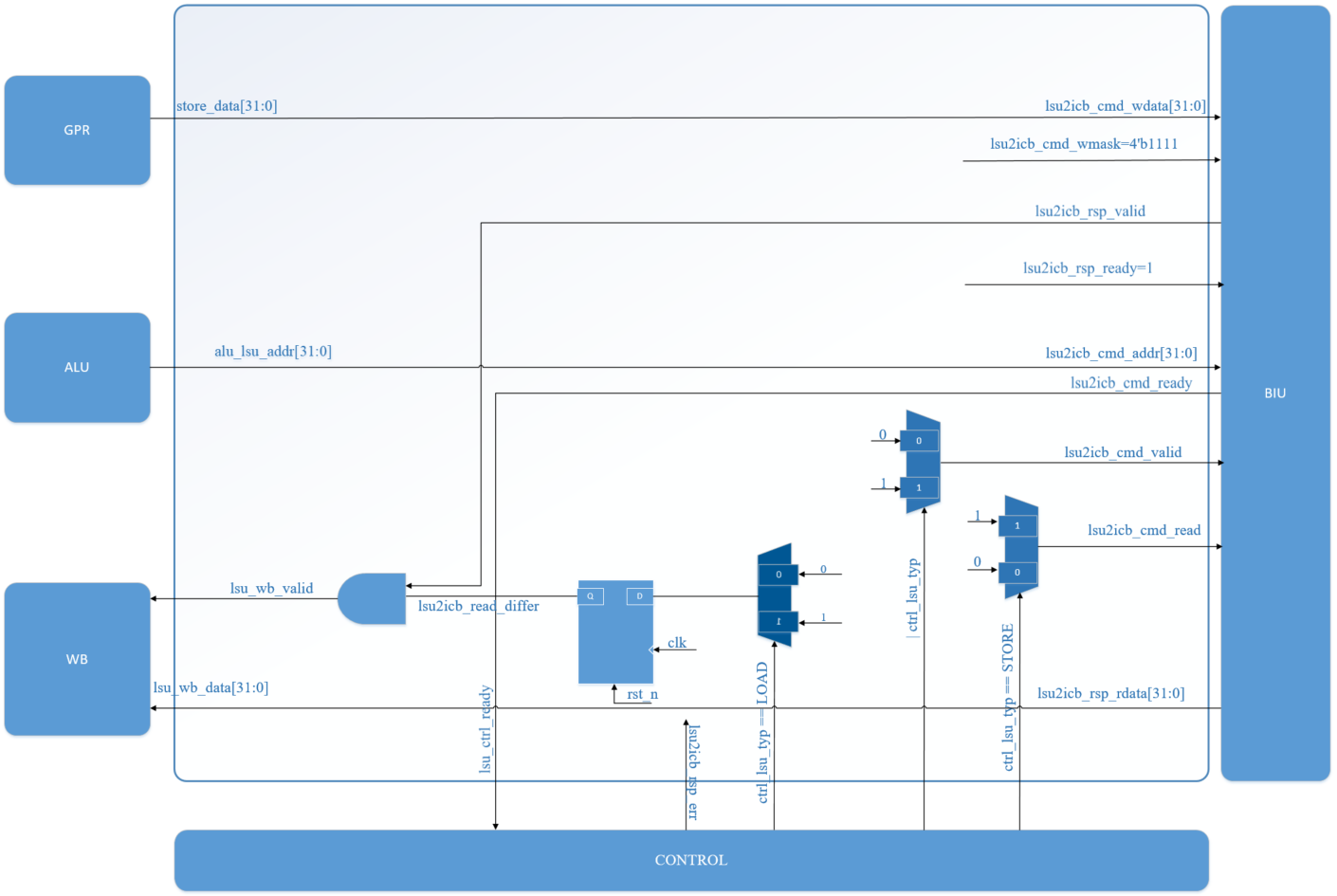
### Interfaces

|  |  |  |  |
| --- | --- | --- | --- |
| Signal | Direction | Width | Comments |
| alu\_wb\_data | Input | 32 | Alu给出的需要写回的数据 |
| lsu\_wb\_data | Input | 32 | lsu给出的需要写回的数据 |
| lsu\_wb\_data\_valid | input | 1 | lsu给出的数据有效信号 |
| wb\_reg\_data | output | 32 | 写回的数据 |



## Sub Block lsu Architecture

### Block Diagram



### Block Descriptions

lsu 模块完成以下功能:

1.当ctrl\_lsu\_typ=STORE时，此时的指令为STORE，将lsu2icb\_cmd\_valid拉高，lsu2icb\_cmd\_read拉低，此时lsu模块将GPR中的数据和ALU中的写入地址传递给BIU， 去进行写入操作。

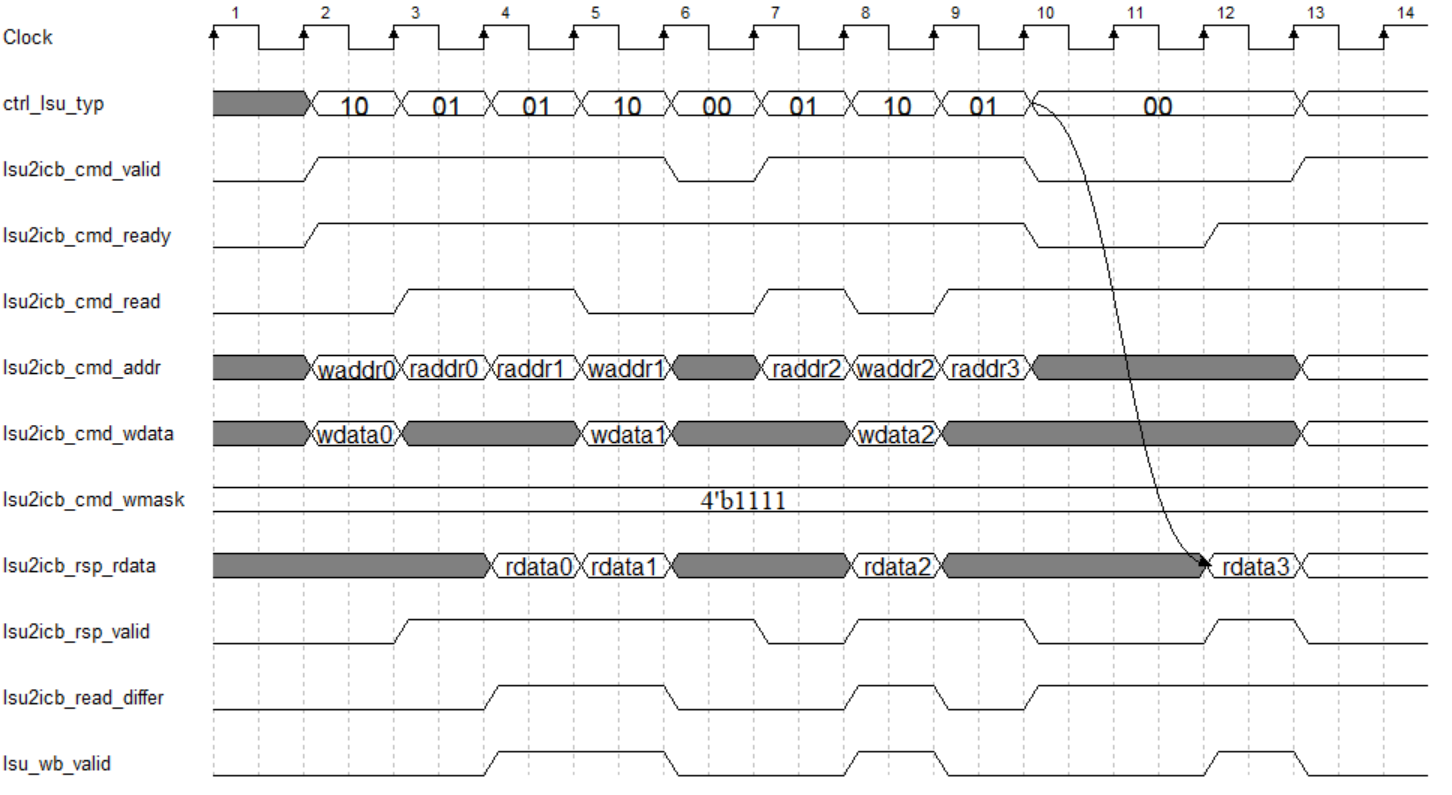
2.当ctrl\_lsu\_typ=LOAD时，此时的指令为LOAD，将lsu2icb\_cmd\_valid拉高，lsu2icb\_cmd\_read拉高，此时lsu模块将ALU中的读地址传递给BIU，根据读地址将读数据写回到WB中，同时将lsu\_wb\_valid拉高；若读取不成功，或需要多周期才能读取，则要将lsu2icb\_cmd\_ready拉低，从外部hold住流水线，保证没有新地址进来。lsu2icb\_rsp\_valid与lsu2icb\_read\_differ相与是为了生成只在读有效时产生的lsu\_wb\_valid信号。

3.当ctrl\_lsu\_typ=NULL时，此时的指令无效，将lsu2icb\_cmd\_valid拉低，既不做STORE操作也不做LOAD操作，直到下一个ctrl\_lsu\_typ有效。

### Interfaces

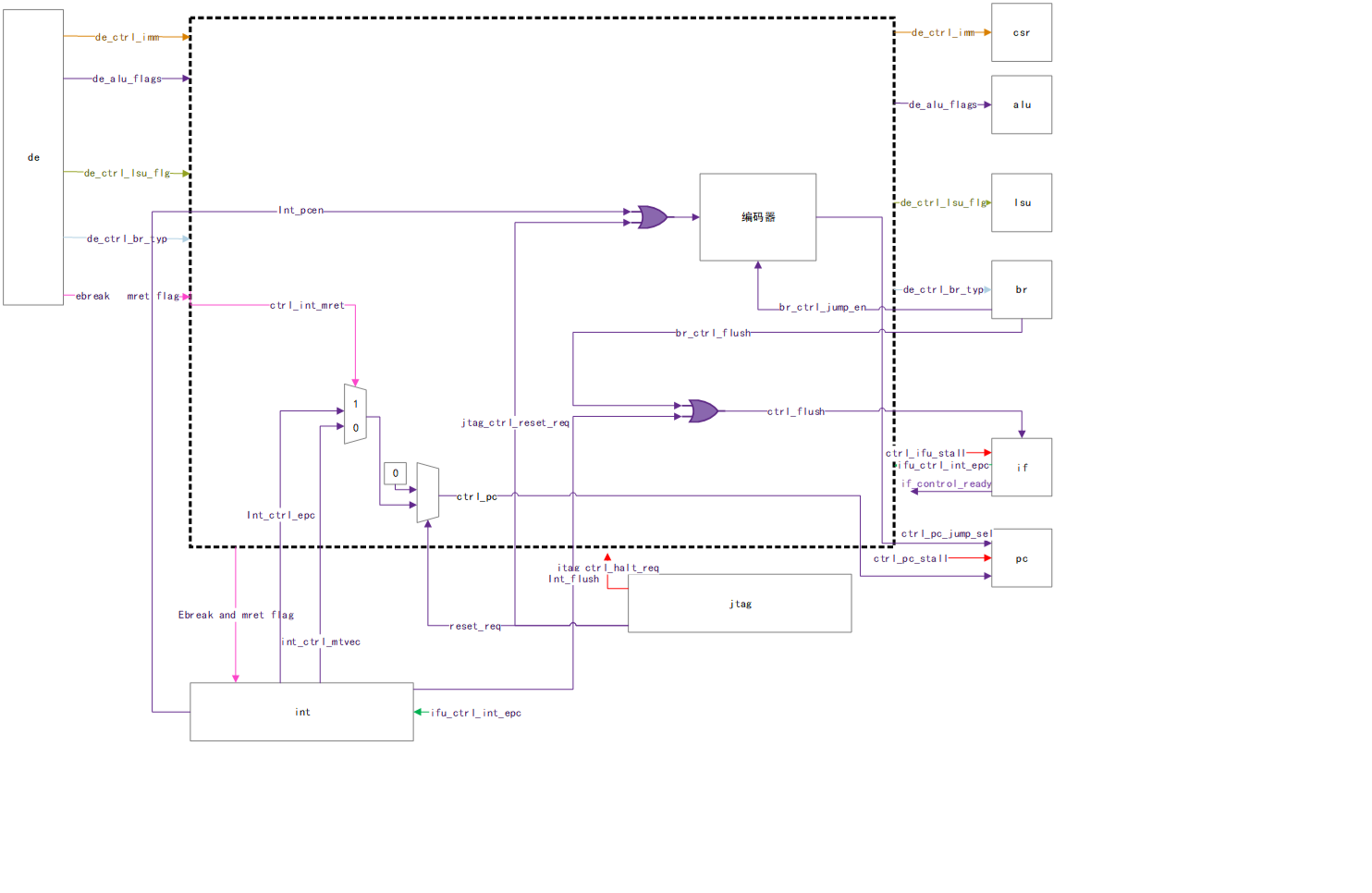
|  |  |  |  |
| --- | --- | --- | --- |
| Signal | Direction | Width | Comments |
| store\_data | input | 32 | 存储的数据 |
| alu\_lsu\_addr | input | 32 | 访存的地址 |
| lsu2icb\_rsp\_rdata | input | 32 | 读反馈的数据 |
| ctrl\_lsu\_typ | Input | 2 | 判断执行l指令还是s指令，01是l指令、10是s指令、00无效 |
| lsu2icb\_rsp\_valid | input | 1 | 从设备向主设备发送读写反馈请求信号 |
| lsu2icb\_cmd\_ready | input | 1 | 从设备向主设备返回读写接收信号 |
| lsu2icb\_rsp\_err | input | 1 | 读或写反馈错误标志 |
| lsu2icb\_cmd\_wdata | output | 32 | 写操作的数据 |
| lsu2icb\_cmd\_addr | output | 32 | 读写地址 |
| lsu\_wb\_data | output | 32 | 写回到wb中的数据 |
| lsu2icb\_cmd\_wmask | output | 4 | 写操作的字节掩码 |
| lsu\_wb\_valid | output | 1 | 写回请求信号 |
| lsu2icb\_cmd\_valid | output | 1 | 主设备向从设备发送读写请求信号 |
| lsu2icb\_cmd\_read | output | 1 | 读或写操作的指示 |
| lsu2icb\_rsp\_ready | output | 1 | 主设备向从设备返回读写反馈接收信号 |
| lsu\_ctrl\_ready | output | 1 | lsu给control模块的ready信号 |

### Timing digram



## Sub Block control Architecture

### Block Diagram



### Block Descriptions

control 模块完成以下功能:

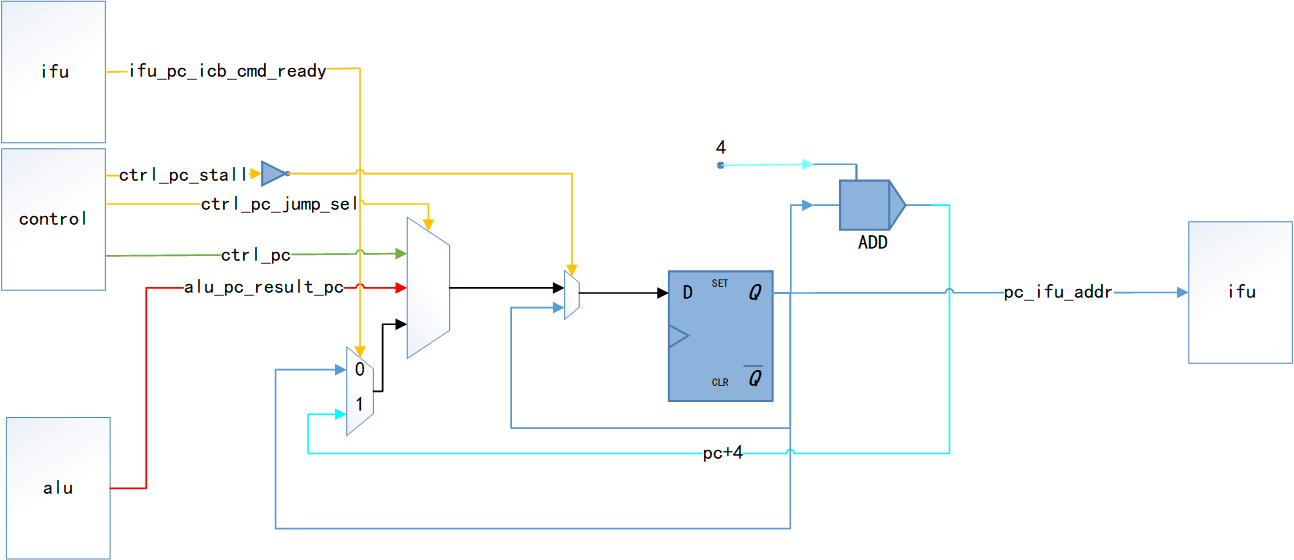
1. 传递JTAG流水线暂停信号
2. jtag的复位优先级相比于中断模块恢复现场的优先级更高，所以需要用PC的reset\_req作为MUX的选择信号，生成ctrl\_pc信号。
3. 中断模块会输入两个不同的PC值，需要根据目前执行的处理器指令不同而选定特定PC。在实现过程中，采用一个数据选择器，用mret作为控制信号。
4. 如果中断模块提交了中断PC有效信号或者JTAG提交了复位，则生成控制输出PC有效信号，这个信号和跳转模块跳转PC生效信号一并送入编码器进行编码，生成后的信号是PC内PC的MUX选择信号。
5. 将透传的信号送给相应的模块。
6. 生成flush信号，flush信号来源于跳转模块以及中断模块。
7. 生成PC选择信号，选择逻辑00代表ctrl\_pc,01代表分支跳转PC。其中ctrl\_pc包含了jtag的重置PC以及来自中断的跳转PC。

### Interfaces

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Signal | | Direction | | Width | | Comments | | |
| **With de** | | | | | | | | |
| de\_ctrl\_imm | | Input | | 32 | | 用于读写csr的地址 | | |
| de\_ctrl\_reg2\_flg | | Input | | 1 | | 1表示运算时使用reg2\_data，0表示使用dc\_alu\_imm，对应的指令有addi，ori，xori，srli，srai，slli，slti，sltiu，lui，addi，auipc | | |
| de\_ctrl\_lui\_flg | | Input | | 1 | | 1表示LUI操作有效，0表示无效，对应的指令有lui | | |
| de\_ctrl\_shift\_flg | | Input | | 1 | | 1表示移位操作有效，0表示无效，对应的指令有srl，srli，sra，srai，sll，slli，lui | | |
| de\_ctrl\_shift\_right\_flg | | Input | | 1 | | 1表示进行右移操作，0表示左移操作，对应的指令同上 | | |
| de\_ctrl\_shift\_logic | | Input | | 1 | | 1表示进行逻辑移位，0表示进行算术移位，对应的指令同上 | | |
| de\_ctrl\_comparator\_flg | | Input | | 1 | | 1表示比较操作有效，0表示比较操作无效，对应的指令有slt，slti，sltu，sltiu | | |
| de\_ctrl\_and\_flg | | Input | | 1 | | 1表示按位与操作有效，0表示按位或操作无效，对应指令and，andi | | |
| de\_ctrl\_or\_flg | | Input | | 1 | | 1表示按位或操作有效，0表示按位或操作无效，对应的指令有or，ori | | |
| de\_ctrl\_xor\_flg | | Input | | 1 | | 1表示按位异或操作有效，0表示按位异或操作无效，对应的指令有xor，xori | | |
| de\_ctrl\_adder\_flg | | Input | | 1 | | 1表示进行加法操作，0表示加法操作无效，对应的指令有add，addi， | | |
| de\_ctrl\_sub\_flg | | Input | | 1 | | 1表示进行减法操作，0表示减法操作无效，对应指令有sub | | |
| de\_ctrl\_auipc\_flg | | Input | | 1 | | 1表示进行pc加立即数操作，对应的指令有auipc | | |
| de\_ctrl\_lsu\_flg | | Input | | 2 | | 判断执行l指令还是s指令，01是l指令、10是s指令、00无效 | | |
| de\_ctrl\_br\_typ | | Input | | 3 | | 一共六种判断beq001、bne010、blt011、bge100、bltu101、bgeu110 | | |
| de\_ctrl\_int\_ebreak | | Input | | 1 | | ebreak instruction in IFU | | |
| de\_ctrl\_int\_mret | | Input | | 1 | | mret signal flag | | |
| **With if** | | | | | | | | |
| ctrl\_ifu\_flush | | output | | 1 | | 接受冲刷信号 | | |
| ctrl\_ifu\_stall | | output | | 1 | | 流水线暂停信号 | | |
| ifu\_ctrl\_int\_epc | | Input | | 32 | | pc of the current instruction before interrupt | | |
| ifu\_control\_ready | | input | | 1 | |  | | |
| ifu\_control\_valid | | input | | 1 | |  | | |
| ifu\_ctrl\_minidecode | | input | | 1 | |  | | |
| **With lsu** | | | | | | | | |
| ctrl\_lsu\_typ | | output | | 1 | | 判断执行l指令还是s指令，高电平是l指令 | | |
| **With branch** | | | | | | | | |
| ctrl\_br\_typ | | output | | 3 | | 一共六种判断beq001、bne010、blt011、bge100、bltu101、bgeu110 | | |
| br\_ctrl\_jump\_en | | input | | 1 | | 是否发生了跳转 | | |
| br\_ctrl\_flush | | input | | 1 | | branch输入的冲刷信号 | | |
| **With alu** | | | | | | | | |
| ctrl\_alu\_reg2\_flag | output | | | | 1 | | 1表示运算时使用reg2\_d  ata，0表示使用dc\_alu\_imm，对应的指令有addi，ori，xori，srli，srai，slli，slti，sltiu，lui，addi，auipc |
| ctrl\_alu\_lui\_flag | output | | | | 1 | | 1表示LUI操作有效，0表示无效，对应的指令有lui |
| ctrl\_alu\_shift\_flag | output | | | | 1 | | 1表示移位操作有效，0表示无效，对应的指令有srl，srli，sra，srai，sll，slli，lui |
| crtl\_alu\_shift\_right\_flag | output | | | | 1 | | 1表示进行右移操作，0表示左移操作，对应的指令同上 |
| ctrl\_alu\_shift\_logic | output | | | | 1 | | 1表示进行逻辑移位，0表示进行算术移位，对应的指令同上 |
| ctrl\_alu\_comparator\_flag | output | | | | 1 | | 1表示比较操作有效，0表示比较操作无效，对应的指令有slt，slti，sltu，sltiu |
| ctrl\_alu\_and\_flag | output | | | | 1 | | 1表示按位与操作有效，0表示按位或操作无效，对应指令add，addi |
| ctrl\_alu\_or\_flag | output | | | | 1 | | 1表示按位或操作有效，0表示按位或操作无效，对应的指令有or，ori |
| ctrl\_alu\_xor\_flag | output | | | | 1 | | 1表示按位异或操作有效，0表示按位异或操作无效，对应的指令有xor，xori |
| ctrl\_alu\_adder\_flag | output | | | | 1 | | 1表示进行加法操作，0表示加法操作无效，对应的指令有add，addi， |
| ctrl\_alu\_sub\_flag | output | | | | 1 | | 1表示进行减法操作，0表示减法操作无效，对应指令有sub |
| ctrl\_alu\_auipc\_flag | output | | | | 1 | | 1表示进行pc加立即数操作，对应的指令有auipc |
| **With pc** | | | | | | | | |
| ctrl\_pc\_jump\_sel | | output | | 2 | | 控制pc跳转选择信号 | | |
| ctrl\_pc | | output | | 32 | | control恢复现场的PC或重置的pc | | |
| ctrl\_pc\_stall | | output | | 1 | | 暂停流水线 | | |
| **With jtag** | | | | | | | | |
| jtag\_ctrl\_halt\_req | | input | | 1 | | ctrl block in core | | |
| jtag\_ctrl\_reset\_req | | input | | 1 | | ctrl block in core | | |
| **With int** | | | | | | | | |
| ctrl\_int\_valid | | output | 1 | | | flag of instruction last has been completely finished | | |
| ctrl\_int\_ready | | output | 1 | | | flag of instruction has been fatched | | |
| int\_ctrl\_mtvec | | input | 32 | | | interrupt service program pc transferred to control module | | |
| ctrl\_int\_epc | | output | 32 | | | pc of the current instruction before interrupt | | |
| ctrl\_int\_ebreak | | output | 1 | | | ebreak instruction in de | | |
| ctrl\_int\_ecall | | output | 1 | | | ecall instruction in de | | |
| ctrl\_int\_mret | | output | 1 | | | mret signal flag | | |
| int\_ctrl\_epc | | input | 32 | | | pc of the next instruction transfer to IFU | | |
| int\_ctrl\_pcen | | input | 1 | | | enable signal of mtecv | | |
| int\_ctrl\_flush\_req | | input | 1 | | | flush require | | |

## Sub Block pc Architecture

### Block Diagram



### Block Descriptions

pc 模块完成以下功能:

1. 处理直接跳转以及间接跳转的PC地址。（00选中ctrl\_pc，01选中alu\_pc\_result\_pc,ctrl\_pc包括了jtag的重置PC以及中断恢复现场的PC值）。
2. 将寄存的PC值送到取指模块。
3. 流水线暂停,此时PC值保持不变。
4. 只有当上一条指令回来后，我们才能进行PC+4,受控于ifu传来的ready。

### Interfaces

|  |  |  |  |
| --- | --- | --- | --- |
| Signal | Direction | Width | Comments |
| rstn | Input | 1 | reset, active low |
| clk | Input | 1 | clock |
| With Control | | | |
| ctrl\_pc\_jump\_sel | Input | 2 | 控制pc跳转选择信号-00选择控制给的PC，01选择跳转的pc. |
| ctrl\_pc | Input | 32 | control恢复现场的PC |
| ctrl\_pc\_stall | Input | 1 | 暂停流水线 |
| With alu | | | |
| alu\_pc\_result\_pc | Input | 32 | alu计算后的跳转pc地址 |
| With ifu | | | |
| pc\_ifu\_addr | output | 32 | 目前需要取指令的PC地址。 |
| ifu\_pc\_icb\_cmd\_ready | input | 1 | 从ifu模块中传递来的上一条指令完毕信号。 |

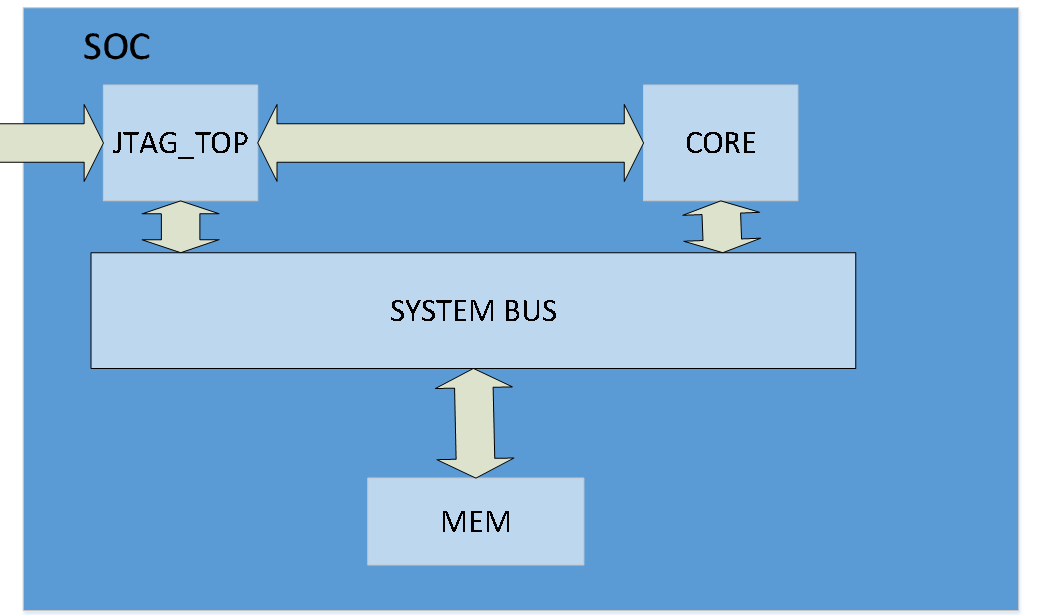
## Sub Block Jtag Architecture

* + 1. **Block Lausanne Debug Overview**

## **Block Lausanne Debug General Description**

Lausanne 调试模块需要实现对基于RV32I指令集的Lausanne Core的调试。

## **System-Level Block Diagram**

调试/JTAG顶层在SOC中的位置如上图所示。Jtag顶层与外界有5根接口(TDO, TDI, TMS, TCK, RST),其中TDO为output，其余都为input。JTAG顶层对内与CORE和RIB之间有通信。与core的通信用以访问寄存器，与RIB的通信用以访问MEM。

## **Features**

### **Supported Features**

* 支持所有的线程(hart)GPRs，CSRs都是可读可写的。
* 支持抽象命令对GPRs和CSRs的访问。
* 支持通过抽象命令(cmdtype=2)来将内存地址arg1中的数据拷贝到arg0中，或将arg0中的数据写入arg1地址的内存。
* 支持RV32
* 支持在默认情况下，调试器能检测到除内存映射和外围设备外所有的信息，也就是所有内存，寄存器的数据。
* 支持每个线程可以从第一个执行指令开始进行Debug，也就是通过reset PC=0。
* 支持一个RISC-V线程能支持一个断点，可以通过将断点的pc值的指令重写为ebreak，且将原指令存入CSR中debug专用的dscratch0寄存器中，从而进入debug mode，退出debug mode时，通过dscratch0恢复指令。
* 支持单步调试。先要暂停线程，通过往CSR寄存器中dcsr(0x7b0)的step中写1，线程就会在运行一条指令后暂停进入debug mode。
* 支持Debug功能独立于被使用到的DTM，也就是说DTM的结构不会对debug功能产生影响。

### **Unsupported Features**

* 不支持寄存器不需要暂停线程即可访问。

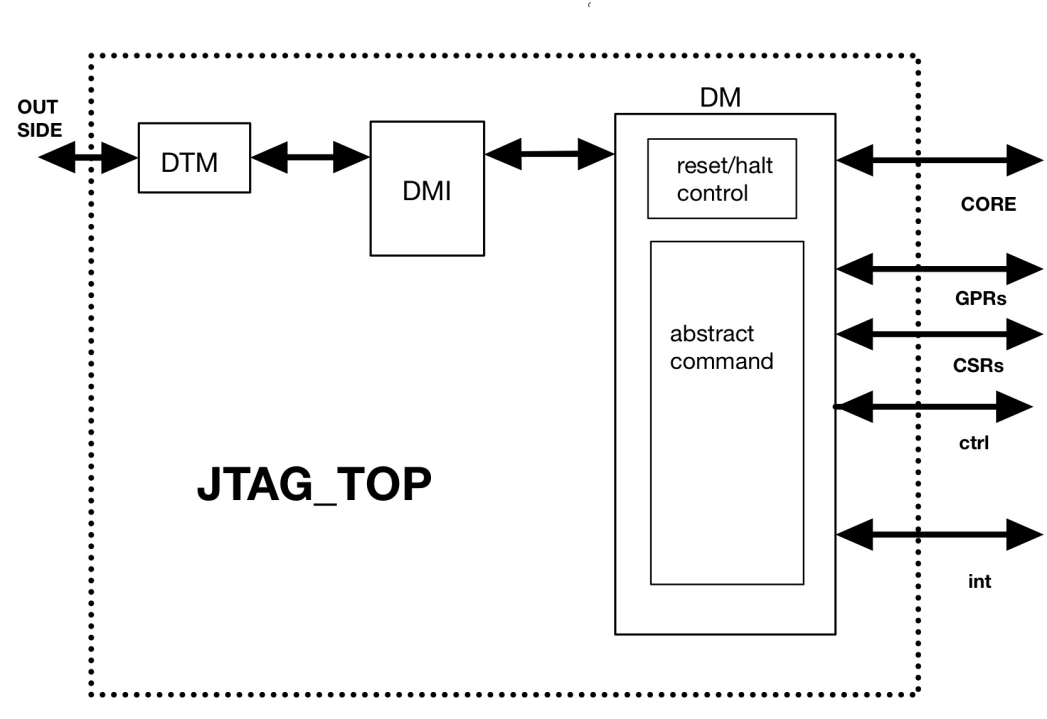
### **Supported Standards**

* RISC-V External Debug Support Version 0.13.2

## **Architecture**

### **Overview of Architecture**

### **Block Diagram**

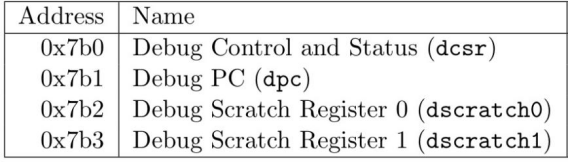


### **Block Overview**

该调试模块由调试主机内的调试器进行控制。主机通过标准jtag线连接到调试模块。输入的串流信号通过DTM转化成并行信号输出给DMI。DMI就是个握手模块，它的作用是连接DTM和DM两个不同时钟域的模块，其中DTM的时钟是jtag 4口之一的TCK，而DM的时钟就是core的时钟。当信号传到DM模块后，线程暂停进入debug模式，DM模块可以通过抽象命令(cmdtype=0)来访问GPRs和CSR，通过抽象命令(cmdtype=2)来访问内存。有4种方式进入debug模式：1. 执行ebreak指令。 2. 重置线程后直接暂停线程进入debug。3. 使用haltreq暂停线程。 4. 单步调试。

### **Modification required for core**

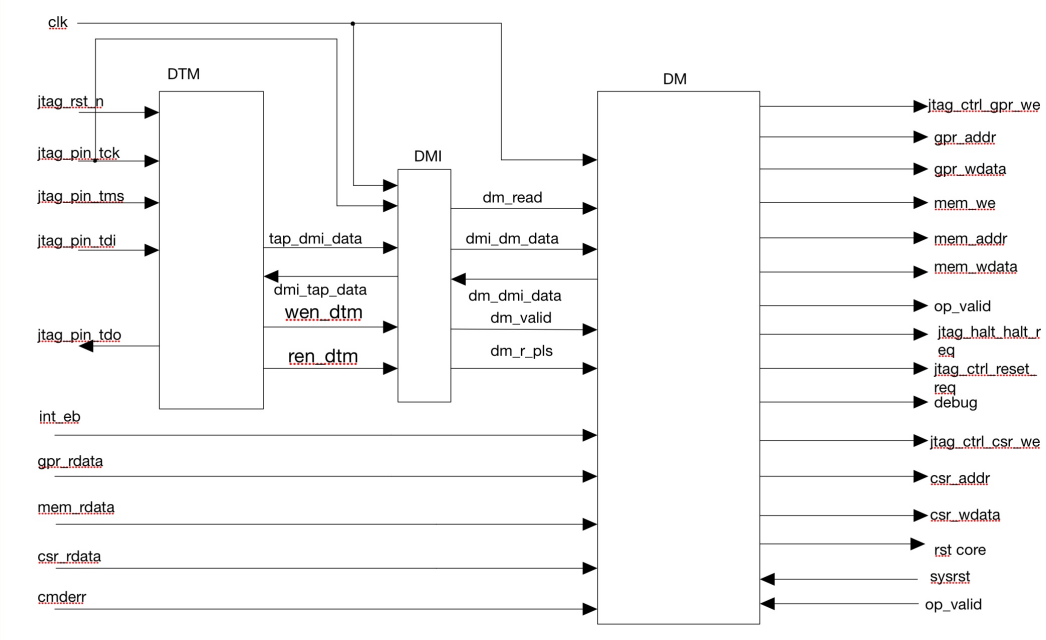
需要4个额外给debug使用的CSRs。

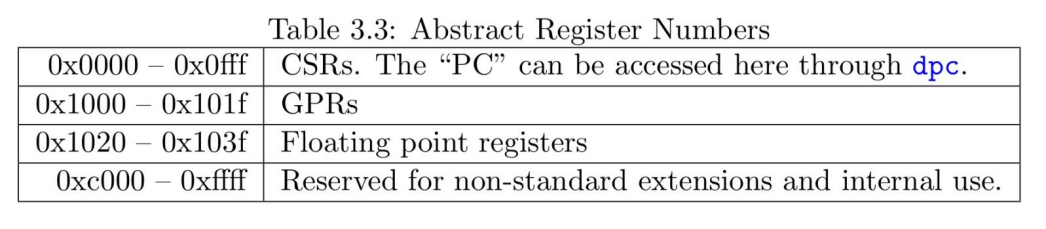


### **Parameters**

|  |  |  |
| --- | --- | --- |
| **Port\_name** | **Description** | **Default** |
| DMI\_ADDR\_BITS | Register address width in DM block | 6 |
| DMI\_DATA\_BITS | Data width of the registers in DM block | 32 |
| DMI\_OP\_BITS | OP width in dmi register of DTM block | 2 |

### **Design Detail**



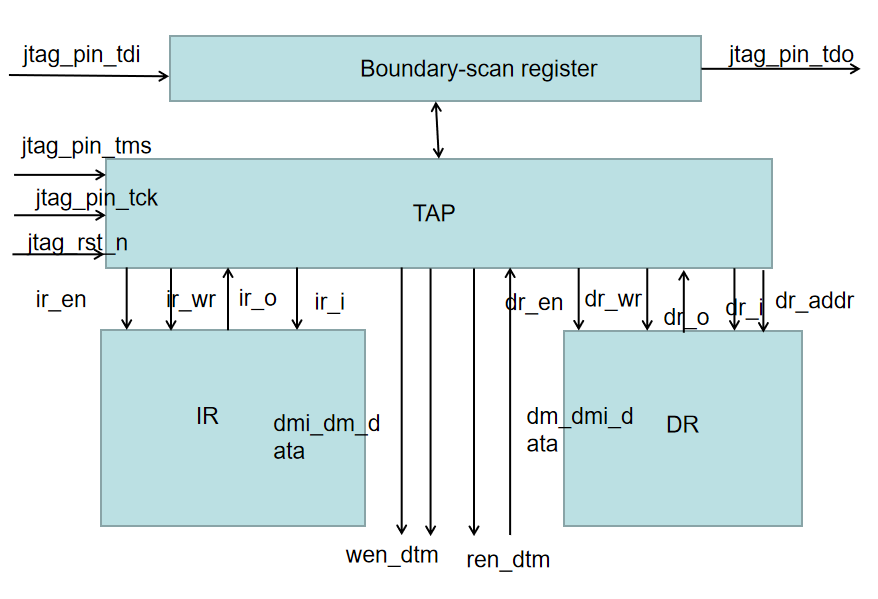


### **Interfaces**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Signal** | **Direction** | **Width** | **Comments** | **Connection** |
| clk | Input | 1 | clock | DMI, DM |
| jtag\_rst\_n | Input | 1 | reset, active low | DTM |
| jtag\_pin\_tck | Input | 1 | DTM clock | DTM |
| jtag\_pin\_tms | Input | 1 | control TAP FSM in DTM | DTM |
| jtag\_pin\_tdi | Input | 1 | DTM external input | DTM |
| jtag\_pin\_tdo | Output | 1 | DTM external output | DTM |
| jtag\_ctrl\_gpr\_we | Output | 1 | gpr write enable signal, low for read, high for write | ctrl |
| gpr\_addr | Output | 5 | gpr address | gpr |
| gpr\_wdata | Output | 32 | data required to be written into gpr | gpr |
| gpr\_rdata | Input | 32 | data read back from gpr | gpr |
| jtag\_ctrl\_csr\_we | Output | 1 | csr write enable signal, low for read, high for write | ctrl |
| csr\_addr | Output | x | csr address | csr |
| csr\_wdata | Output | 32 | data required to be written into csr | csr |
| csr\_rdata | Input | 32 | data read back from csr | csr |
| cmderr | Input | 1 | abstract command error | gpr, csr, sysbus |
| mem\_we | Output | 1 | memory write enable signal | sysbus |
| mem\_addr | Output | 32 | memory address sent to system bus | sysbus |
| mem\_wdata | Output | 32 | data needs to be written into a certain memory register | sysbus |
| mem\_rdata | Input | 32 | feedback or read data from a certain memory register | sysbus |
| op\_valid | Input | 1 | valid signal sent from system bus to confirm access permission | sysbus |
| op\_ready | Output | 1 | enable signal for the connection between jtag top and system bus | sysbus |
| systemerr | Input | 1 | system bus access error | sysbus |
| jtag\_ctrl\_halt\_req | Output | 1 | this halt signal sent to ctrl block in core to hold pc | ctrl |
| jtag\_ctrl\_reset\_req | Output | 1 | sent to ctrl to reset pc to zero | ctrl |
| int\_eb | Input | 1 | ebreak interrupt signal | int |
| debug | Output | 1 | entered debug mode signal | int |
| sysrst | Input | 1 | system reset signal | SOC |
| rstcore | Output | 1 | reset everything | core |

## **DTM Block Architecture**

### **Block Diagram**



### **Block Descriptions**

DTM模块作用是将TDI的串流输入数据变成并行输出给DMI模块的数据，其下主要有4个子模块，边界扫描寄存器（类似于移位寄存器），TAP状态机，IR，DR。

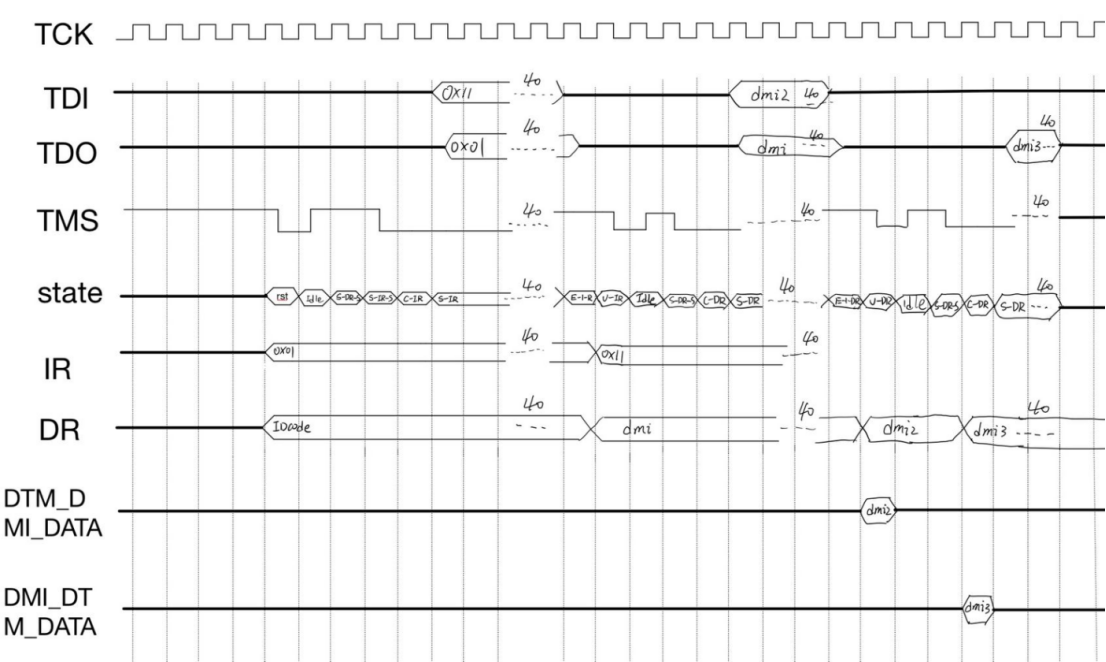
### **Parameters**

|  |  |  |
| --- | --- | --- |
| **Port\_name** | **Description** | **Default** |
| DMI\_ADDR\_BITS | the width of registers address domain of dmi register in DTM | 6 |
| DMI\_DATA\_BITS | the width of registers data domain of dmi register in DTM | 32 |
| DMI\_OP\_BITS | the op width of dmi register in DTM | 2 |

### **Interfaces**

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal** | **Direction** | **Width** | **Comments** |
| jtag\_rst\_n | Input | 1 | reset, active low |
| jtag\_pin\_tck | Input | 1 | DTM clock |
| jtag\_pin\_tms | Input | 1 | control TAP FSM in DTM |
| jtag\_pin\_tdi | Input | 1 | DTM external input |
| jtag\_pin\_tdo | Output | 1 | DTM external output |
| dmi\_dm\_data | Input | 40 | data consisted of address, 32-bit data and w/r signal sent from DMI |
| dm\_dmi\_data | Output | 34 | Data consisted of 32-bit data and feedback signal sent to DMI |
| wen\_dtm | Output | 1 | write to DMI enable signal to DMI |
| ren\_dtm | Output | 1 | read from DMI enable signal from DMI |

### **Timing Diagram**



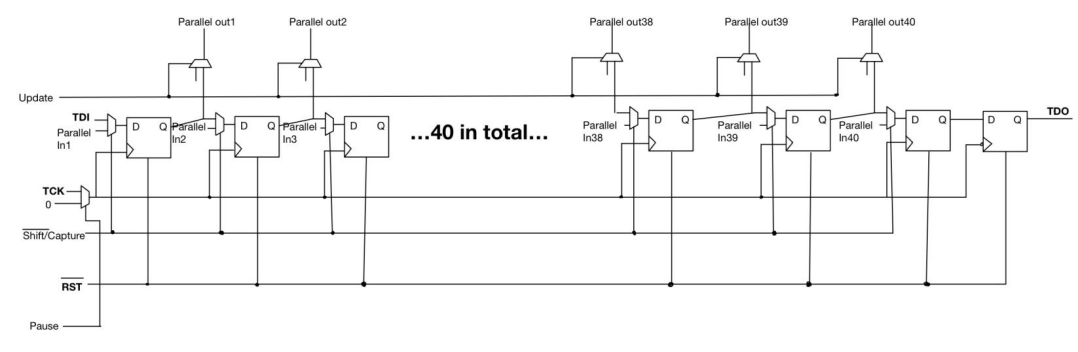
**Explanation:**

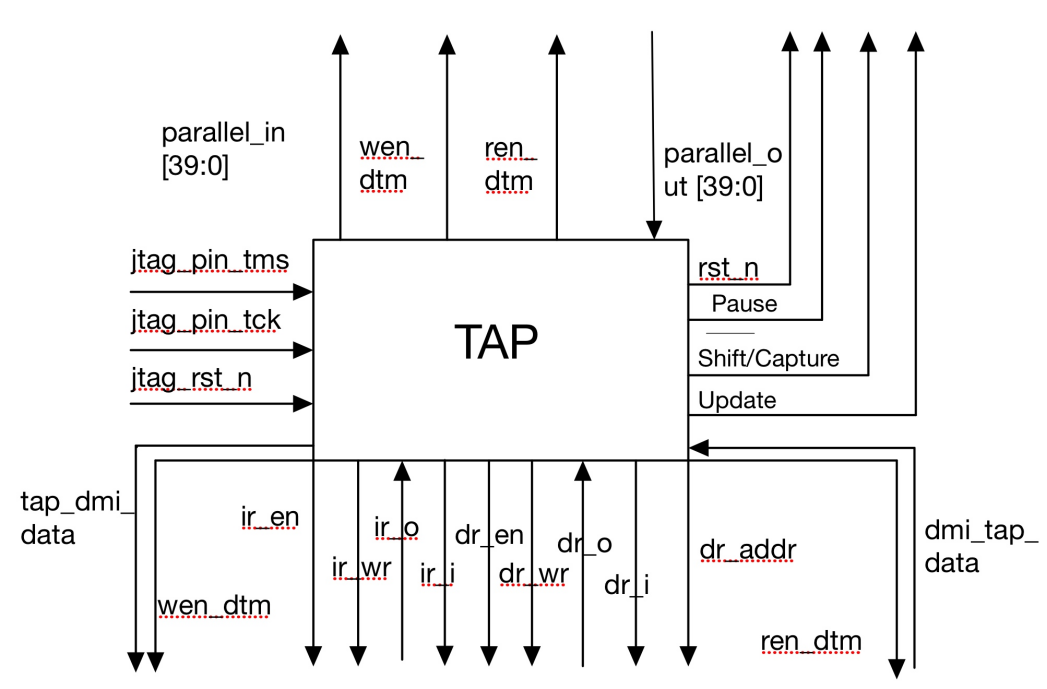
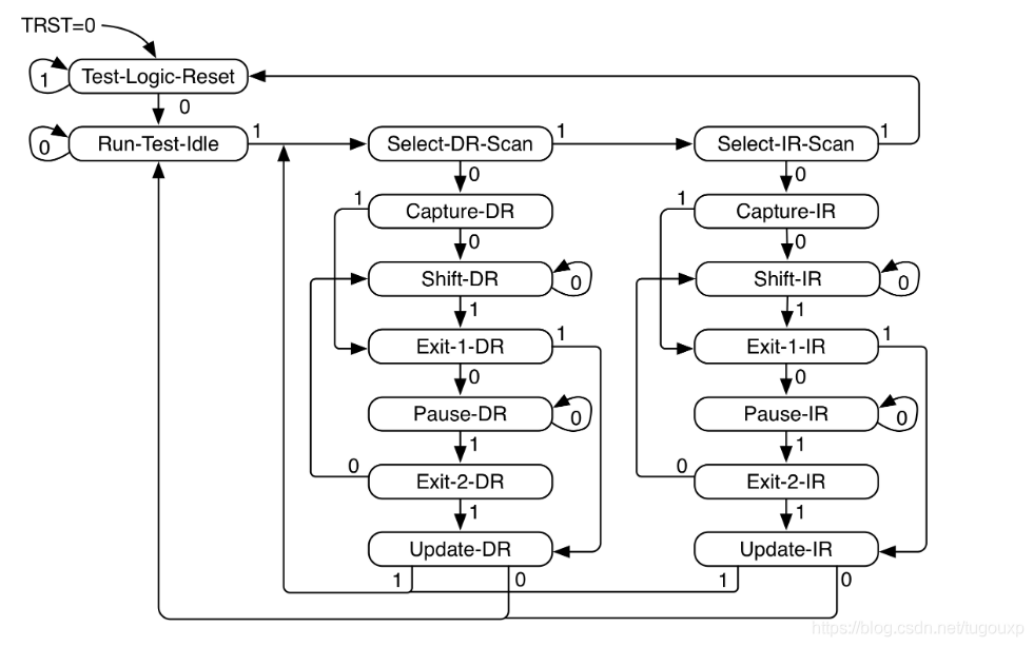
TMS控制TAP状态机状态，在第一个shift-IR的状态时，往移位寄存器中移位进新的IR值（就是dmi的地址），同时移位出IDCODE的地址，随后通过update-IR将dmi地址写入IR。随后选中IR中地址对应的DR(dmi)，往边界扫描寄存器中移位进新的dmi (dmi2)，在update-DR中将新的dmi (dmi2)更新进dmi寄存器，同时按照dmi2中的op域执行指定操作，此时会有信号从DTM\_DMI\_DATA中发出往DMI，在接下来的capture-DR状态中，将操作的反馈值写入dmi(dmi3)，同时将新写入的反馈值(dmi3)更新进边界扫描寄存器中，在随后的shift-DR中将其移位出去。

TMS为控制TAP FSM的输入信号。

### **Design Detail**

**Boundary scan register**

****

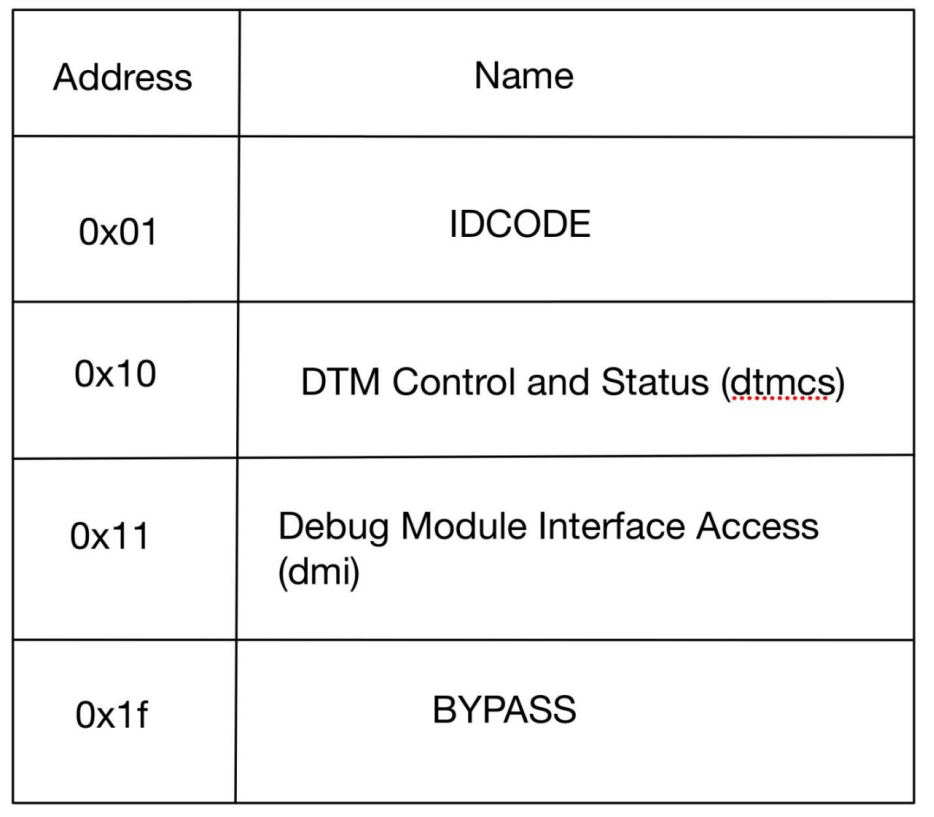
**TAP FSM**

注：此表格为当IR寄存器中地址指定为dmi时。

**IR&DR**

有且仅有一个的指令寄存器IR长度为5位，当TAP FSM复位时，IR的值默认为5’00001，即选择DR寄存器，IDCODE。

总共支持4个DR其地址和名字如下：

****

***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所示。IDCODE是只读寄存器。



IDCODE寄存器

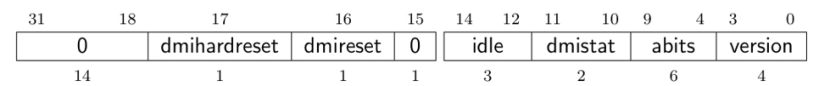
Version：只读，版本号，可为任意值。

PartNumber：只读，可为任意值。

Manufld：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。

***DTM控制和状态寄存器(dtmcs，0x10)***

dtmcs寄存器的每一位含义如图所示。



dtmcs寄存器

dmihardreset：DTM模块硬复位，写1有效。

dmireset：清除出错，写1有效。

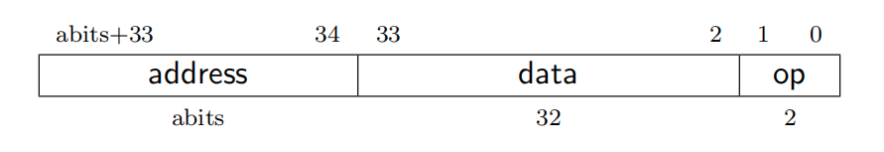
idle：只读，JTAG 主机在Run-Test-Idle状态停留的时钟周期数，0表示不需要进入Run-Test-Idle状态，1表示进入Run-Test-Idle状态后可以马上进入下一个状态，以此类推。

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。

***DM模块接口访问寄存器(dmi，0x11)***

dmi寄存器的每一位含义如图所示

dmi寄存器

这个寄存器允许对DMI的访问。

当更新DR时，DTM会按照op的状态进行操作，除非op sticky

当读取DR时，DTM更新从操作中获取的数据，并且如果op不为sticky，更新op。

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(update-DR)，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dm寄存器读数据；写2表示把data的数据写到address指定的dm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(sticky)。3表示上一个操作还未完成(sticky)。

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。

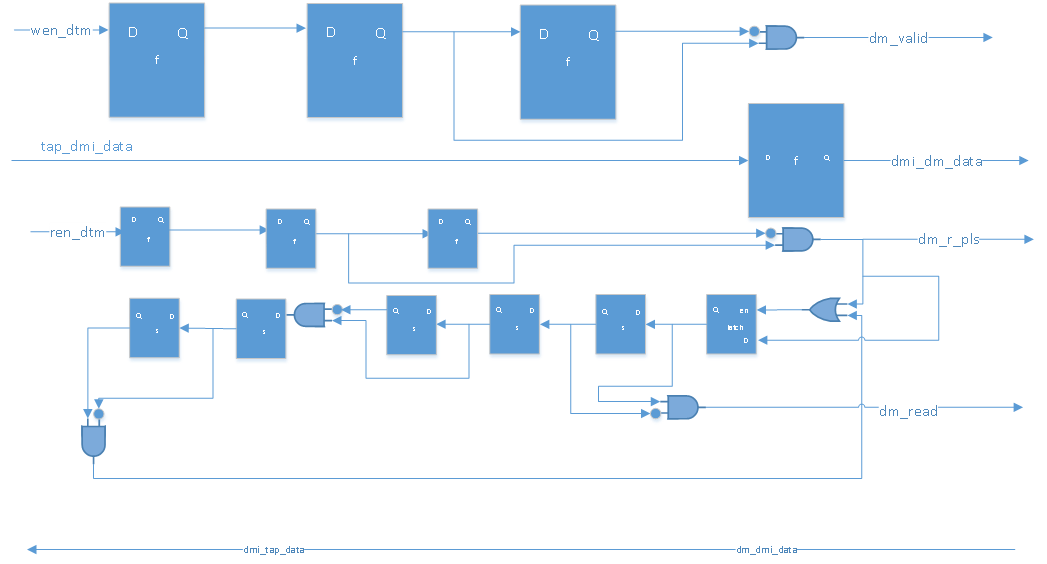
2.1.4 BYPASS寄存器(0x1f)

只读，长度为1，值固定为0。

**BYPASS (0x1f)**

为1位宽的寄存器，其内永远为0。

## **DMI Block Diagram**



### **Block Descriptions**

这是一个握手模块。因为DTM的时钟域是TCK，DM的时钟域是core的时钟，所以需要进行跨时钟域处理。

当有数据要从DTM传给DM时也就是TAP在update-DR状态时，会同时发wen\_dtm和tap\_dmi\_data。wen\_dtm会被快时钟打两拍，同步到快时钟域，变成dm\_valid信号，当dm\_valid置高位时，dmi\_dm\_data为有效输入。当数据要从DM传给DTM时，也就是TAP在capture状态时，会将ren\_dtm置高，ren\_dtm会被打两快时钟拍，变成一个脉冲，dm\_r\_pls，此信号会让dm中的输出数据被保存到寄存器中，随后会产生一个在pulse下降沿上升并保持直到下一个TCK上升沿的dm\_read，在该信号置高的时候，dm\_dmi\_data会被输出到DTM中的dmi data寄存器中。

### **Parameters**

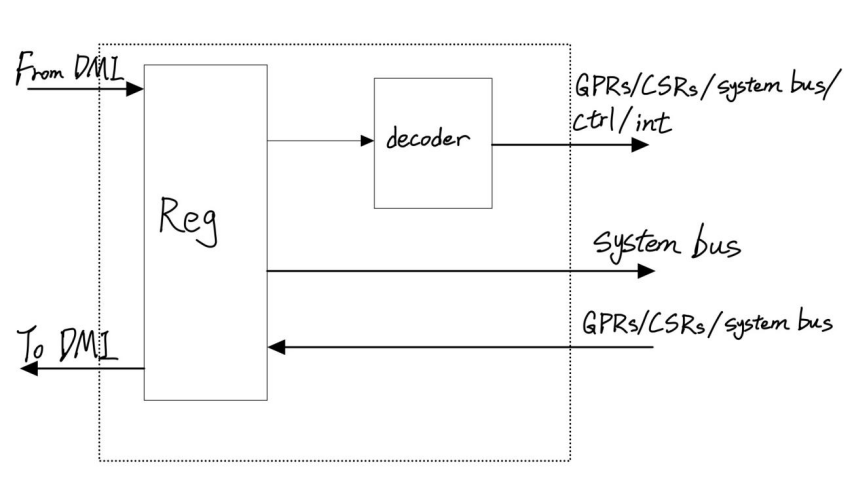
|  |  |  |
| --- | --- | --- |
| **Port\_name** | **Description** | **Default** |
| DMI\_ADDR\_BITS | the width of registers address domain of dmi register in DTM | 6 |
| DMI\_DATA\_BITS | the width of registers data domain of dmi register in DTM | 32 |
| DMI\_OP\_BITS | the op width of dmi register in DTM | 2 |

### **Interfaces**

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal** | **Direction** | **Width** | **Comments** |
| tck | Input | 1 | JTAG clock |
| clk | Input | 1 | system clk |
| tap\_dmi\_data | Input | 40 | Data sent from dmi in DTM to DMI |
| dmi\_tap\_data | Output | 32 | Data sent from DMI to dmi in DTM |
| dm\_dmi\_data | Input | 40 | Data sent from DM to DMI |
| dmi\_data\_data | Output | 40 | Data sent from DMI to DM |
| wen\_dtm | Input | 1 | write enable to DMI from DTM |
| dm\_valid | Output | 1 | write enable to DM from DMI |
| ren\_dtm | Input | 1 | read enable signal from DTM |
| dm\_r\_pls | Output | 1 | write enable signal to the reg in DM |
| dm\_read | Output | 1 | read enable signal to the reg in DM |

## **DM**

### **Brief diagram**



DM模块的寄存器都为32位，实现了0x04 data0, 0x08 data1, 0x10 dmcontrol, 0x11 dmstatus, 0x12 hartinfo, 0x16 abstractcs, 0x17 command

### **Interfaces**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Signal** | **Direction** | **Width** | **Comments** | **Connection** |
| clk | Input | 1 | system clock | /N |
| dmi\_dm\_data | Input | 40 | data consisted of address, 32-bit data and w/r signal sent from DMI | dmi |
| dm\_valid | Input | 1 | valid signal for dmi\_dm\_data | dmi |
| dm\_dmi\_data | Output | 34 | Data consisted of 32-bit data and feedback signal sent to DMI | dmi |
| dm\_r\_pls | Input | 1 | write enable to the reg in DM | dmi |
| dm\_read | Input | 1 | read enable to the reg in DM | dmi |
| jtag\_ctrl\_gpr\_we | Output | 1 | gpr write enable signal, low for read, high for write | ctrl |
| gpr\_addr | Output | 5 | gpr address | gpr |
| gpr\_wdata | Output | 32 | data required to be written into gpr | gpr |
| gpr\_rdata | Input | 32 | data read back from gpr | gpr |
| jtag\_ctrl\_csr\_we | Output | 1 | csr write enable signal, low for read, high for write | ctrl |
| csr\_addr | Output | x | csr address | csr |
| csr\_wdata | Output | 32 | data required to be written into csr | csr |
| csr\_rdata | Input | 32 | data read back from csr | csr |
| cmderr | Input | 1 | abstract command error | gpr, csr, sysbus |
| mem\_we | Output | 1 | memory write enable signal | sysbus |
| mem\_addr | Output | 32 | memory address sent to system bus | sysbus |
| mem\_wdata | Output | 32 | data needs to be written into a certain memory register | sysbus |
| mem\_rdata | Input | 32 | feedback or read data from a certain memory register | sysbus |
| op\_valid | Input | 1 | valid signal sent from system bus to confirm access permission | sysbus |
| op\_ready | Output | 1 | enable signal for the connection between jtag top and system bus | sysbus |
| jtag\_ctrl\_halt\_req | Output | 1 | this halt signal sent to ctrl block in core to hold pc | ctrl |
| jtag\_ctrl\_reset\_req | Output | 1 | sent to ctrl to reset pc to zero | ctrl |
| int\_eb | Input | 1 | ebreak interrupt signal | int |
| debug | Output | 1 | entered debug mode signal | int |
| sysrst | Input | 1 | system reset signal | SOC |
| rstcore | Output | 1 | reset everything | core |

### **Description for each DM register**

***2.2.1 data寄存器(data0, 0x04, data1, 0x05)***

这2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所示。

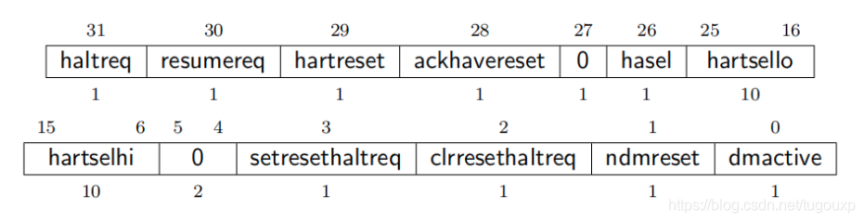


图 dmcontrol寄存器

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：只写，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。

ackhavereset：只能写1，写1表示清除当前hart的havereset状态。

hasel：只可以为0。

hartsello：该值为1。

hartselhi：该值为0。

setresethaltreq：只写，写1表示当前选择的hart复位清除后处于halted状态。

clrresethaltreq：只写，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)***

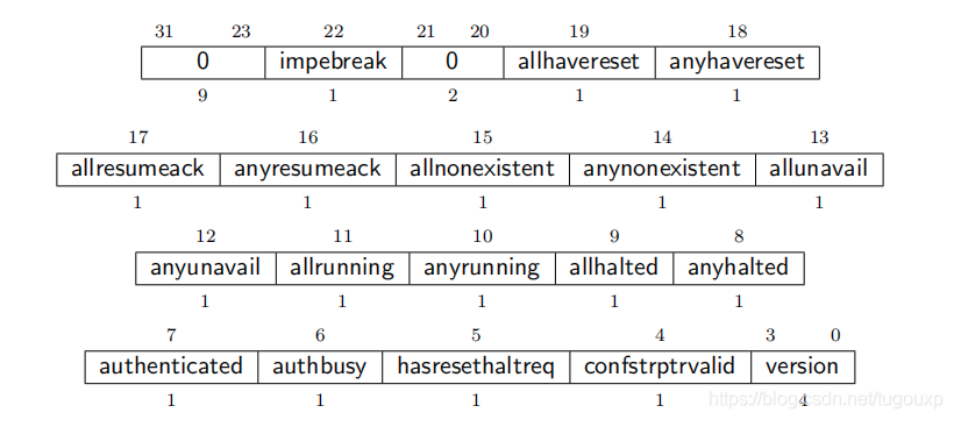
dmstatus寄存器是一个只读寄存器，每一位含义如图所示。

图 dmstatus寄存器

impebreak：控制pb的，不支持。

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：常为0。

anynonexistent：常为0。

allunavail：常为0。

anyunavail：常为0。

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：常为1。

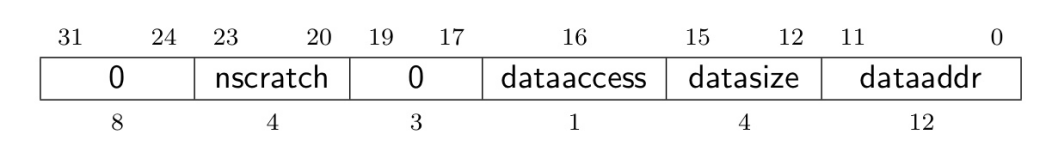
authbusy：常为0。

hasresethaltreq：常为1。

confstrptrvalid：常为0。

version：常为2。

***2.2.4 线程信息寄存器(hartinfo, 0x12)***

******

nscratch: 只读，因为CSR中仅支持1个dscratch，所以常设置为1。

dataaccess: 只读，常为0。

datasize: 只读， 常为0。

dataaddr: 只读，常为0。

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

abstractcs寄存器定义如图所示。



abstractcs寄存器

progbufsize：只读，常为0。

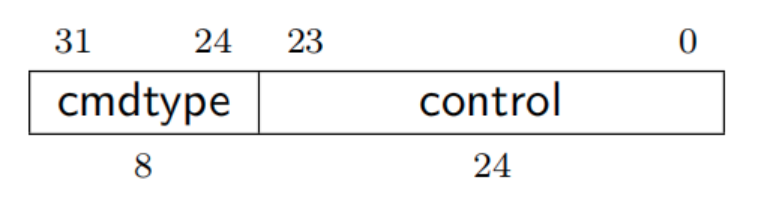
busy：只读，1表示abstract命令正在执行，当写command寄存器后该位应该马上被置位直到命令执行完成。

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，常为2。

***2.2.6 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，定义如图所示。

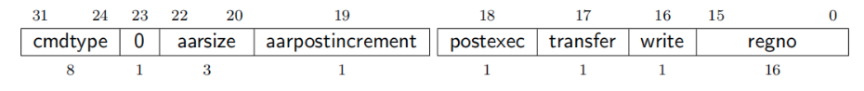


command寄存器

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定义如图所示。**



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：不支持，常为0。

postexec：不支持，常为0。

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：

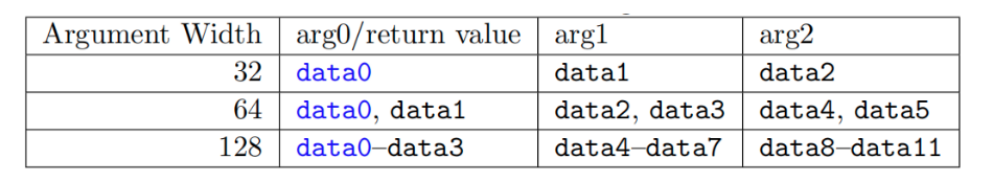
当write=0，transfer=1时，从regno指定的寄存器拷贝数据到arg0对应的data寄存器。

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。

当aarpostincrement=1时，将regno的值加1。

当postexec=1时，执行progbuf寄存器里的指令

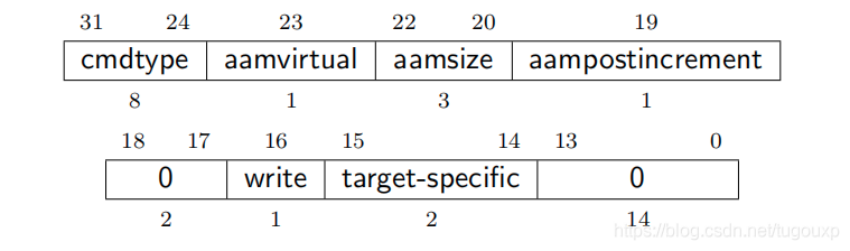
arg对应的data寄存器如图所示。



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2时，control定义如图所示。**



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，默认物理地址。

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：

当write=0时，从arg1指定的地址拷贝数据到arg0指定的data寄存器。

当write=1时，从arg0指定的data寄存器拷贝数据到arg1指定的地址。

当aampostincrement=1时，增加arg1对应的data寄存器的值。

### **Detailed Diagram**

**Explanation:**

首先，会有两根从DMI通入DM的信号，分别是dmi\_dm\_data[39:0]和dm\_valid。前者在PRE\_DE中被分成3根信号，从上到下分别表示DM寄存器的地址，DM寄存器需要写入的数据，读还是写的使能；后者dm\_valid (valid)则是dmi\_dm\_data信号有效的标志，置高位有效，低位无效。然后这4根信号通入DE\_REG中，从DE\_REG中出来4根信号，reg\_addr, reg\_wdata, reg\_wr, reg\_en，通给DM\_REG，其中注意当reg\_addr中指定的地址为0x16时，reg\_en会被置为低位，也就是说，事实上无法对0x16寄存器进行读写，但可以间接对0x16进行clear。

1. 对GPR和CSR的访问，首先reg\_addr中的值要为0x17，意味着对command寄存器进行操作，command为一个32位寄存器，其高8位，command[31:24]为0，意味着进入command type0，及寄存器的访问，确定好command type后就按照对应的command type进行对command的剩下低24位进行解码：command[23]常为0，没任何特殊意义；command[22:20]必须为2，意味着本SOC只支持32位操作；command[19]常设置为0，代表不支持regno自动增加； postexec[18]常设置为0，意味着不支持执行完抽象命令后进入program buffer；command[17]是对command[16]这个读写使能的使能，高位代表command[16]有效，低位则无效；command[16]是读写使能，高位为写，低位为读；command[15:0]是对需要操作的核的寄存器的地址，大于0xffff则是对GPR操作，小于0xffff则是对CSR操作。需要写就将data0(0x04)中的数据写入regno中指定的寄存器，需要读就将regno中指定寄存器的数据写入data0(0x04)中。然后下次JTAG再对data0进行访问就可以了。

对memory的访问，首先reg\_addr中的值要为0x17，意味着对command寄存器进行操作，command为一个32位寄存器，其高8位，command[31:24]为2，意味着进入command type2，及内存的访问，确定好command type后就按照对应的command type进行对command的剩下低24位进行解码：command[23]位常设置位0，因为只支持对物理地址的访问；command[22:20]位常设置为2，意味着本SOC只支持32位操作；command[19]位为自动增加memory地址，高位有效，每次增加4个字节，写入data1这个存内存地址的寄存器；command[16]为读写使能，高位写，低位读；command[15:0]无特殊含义，常设置为0。在进行memory操作时，busy置高，只有当op\_valid置高的时候才会清除busy。

## Sub Block Int Architecture

### Block Diagram

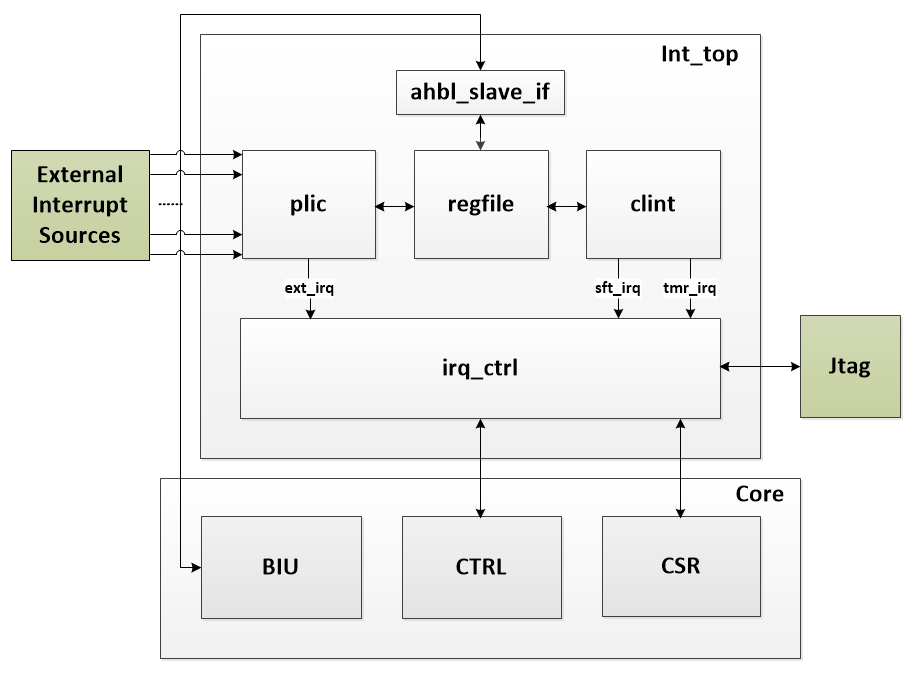


图2.11.1 int模块总架构示意图

### Block Descriptions

1. Interrupt顶层模块负责与核内的biu总线、ctrl、csr模块以及核外的Jtag模块作信息交互。并且接收soc层面传递来的多个外部中断源信号。
2. plic模块负责外部中断源信号之间的仲裁，最终仲裁出一个单比特外部中断信号给int\_ctrl模块。
3. clint模块负责软件中断和计时器中断信号的产生。
4. ahbl\_slave\_if模块负责将biu总线中ahbl\_bridge传输来的控制信号和数据信号进行对齐操作，将对齐后的数据传输给regfile模块读写。
5. regfile内部映射有多组寄存器，通过总线传输的交互信号读写中断仲裁及产生需要用到的IP、IE、优先级、域值、完成响应、计数、msip等信息。将这些寄存器的信息传输给plic和clint模块进行参与中断信号的生成。
6. Irq\_ctrl模块负责接收外部中断信号、计时器中断信号、软件中断信号、调试中断信号和ebreak指令中断并仲裁。与CSR寄存器模块交互负责硬件保存现场、保护断点、产生流水线冲刷以恢复断点的工作。

### Features

* 中断源数量可配置，最多支持255个外部中断源，中断源ID=0保留，表示没有中断，其对应的优先级为0，即传输给模块的外部中断信号一共256比特，第0位为1'b0。
* 中断优先级位宽可配，最高支持32级中断优先级（0~31），即PRIO\_WIDTH [max]=6，默认中断优先级数位0-7，即PRIO\_WIDTH [max]=3。
* 支持ebreak指令，由于ebreak指令为进入调试模式，该指令属于同步异常，因此记录的断点pc值为ebreak指令本身的pc，该pc存放在debug相关csr寄存器中。需要软件修改CSR寄存器中ebreak指令时断点的值为原pc+4，否则会造成死锁。
* 仅支持特权模式中的m\_mode，所有中断/异常执行都跳入同一个异常服务程序起始地址（mtvec所记录的pc）。
* ecause寄存器存储的中断原因有三种：软件中断、计时器中断、外部中断。异常原因目前仅有：ebreak、ecall。中断服务程序结束后并不会清除ecause中记录的中断原因。

### Unsupported Features

* 不支持用户模式和监督模式。
* 不支持不同中断自动分配对应的中断服务程序地址。（所有中断同一个服务程序起始地址，由软件根据中断类型进入到对应的服务程序）
* 不支持矢量中断和非矢量中断的区分。
* 不支持判断外部中断的触发类型。
* 不支持中断咬尾操作。
* 不支持中断嵌套。（RSIC架构规定中断响应时关闭全局中断，若想要实现中断嵌套需要软件打开全局中断，并将所嵌套的中断信息压入堆栈）。

### Supported Standards

* RSIC-V

### Parameters

|  |  |  |
| --- | --- | --- |
| **Port\_name** | **Description** | **Default** |
| EXT\_INT\_NUM | Number of external interrupt sources | 31 |
| PRIO\_WIDTH | Priority bit width | 3 |

### Interfaces

|  |  |  |  |
| --- | --- | --- | --- |
| Signal | Direction | Width | Comments |
| clk | Input | 1 | clock |
| rstn | Input | 1 | reset, active low |
| **With external** | | | |
| ext\_plic\_irq | Input | EXT\_INT\_NUM + 1 | Total external interrupt signals transfer to plic module. (ext\_plic\_irq[0] = 0) |
| **With timer** | | | |
| triggjtag\_irqer | input | 1 | always on clock trigger |
| **With biu** | | | |
| biu\_int\_haddr | Input | 32 | address of registers from ahbl bridge |
| biu\_int\_hwrite | Input | 1 | 1: write 0: read |
| biu\_int\_htrans | Input | 2 | 00: IDL 10: NONSEQ |
| biu\_int\_hsize | Input | 3 | always be 3’b010, 32 bits |
| biu\_int\_hwdata | Input | 32 | write data |
| biu\_int\_hburst | Input | 3 | always be 3’b000, SINGLE |
| biu\_int\_hsel | input | 1 | select signal of interrupt slave |
| biu\_int\_hready | input | 1 | hready from bus |
| int\_biu\_hreadyout | output | 1 | handshake signal |
| int\_biu\_hresp | output | 2 | response =OK |
| int\_biu\_hrdata | output | 32 | read data |
| **With jtag** | | | |
| jtag\_irq | input | 1 | debug mode now or debug interrupt |
| int\_jtag\_ebreak | output | 1 | ebreak instruction after arbitration |
| **With csr** | | | |
| counterstop | input | 1 | time stop flag from counterstop csr register |
| int\_csr\_ext | output | 1 | external interrupt for MIP register |
| int\_csr\_tmr | output | 1 | timer interrupt for MIP register |
| int\_csr\_sft | output | 1 | software interrupt for MIP register |
| csr\_int\_mie | Input | 1 | value of mie field in mstatus register |
| csr\_int\_meie | Input | 1 | value of meie field in mie register |
| csr\_int\_mtie | Input | 1 | value of mtie field in mie register |
| csr\_int\_msie | Input | 1 | value of msie field in mie register |
| csr\_int\_mtvec | Input | 32 | interrupt service program pc stored in mtvec register |
| csr\_int\_epc | Input | 32 | pc of the next instruction before interrupt |
| int\_csr\_ecasuse | Output | 32 | interrupt cause transfer to CSR register |
| int\_csr\_epc | Output | 32 | interrupt pc transfer to CSR register |
| int\_csr\_ena | Output | 1 | enable signal to modify epc, ecause and mstatus register |
| int\_csr\_mret | Output | 1 | mret enable signal to modify mstatus register back |
| int\_csr\_dcasuse | Output | 32 | ebreak cause transfer to CSR register |
| int\_csr\_dpc | Output | 32 | ebreak pc transfer to CSR register |
| int\_csr\_dena | output | 1 | enable signal to modify dcsr and dcause reristers |
| **With ctrl** | | | |
| ctrl\_int\_ready | intput | 1 | flag of biu ready |
| ctrl\_int\_valid | input | 1 | flag of biu valid |
| int\_ctrl\_mtvec | Output | 32 | interrupt service program pc transferred to control module |
| ctrl\_int\_epc | Input | 32 | pc of the current instruction before interrupt |
| ctrl\_int\_ebreak | Input | 1 | ebreak instruction in decode |
| ctrl\_int\_ecall | Input | 1 | ecall instruction in decode |
| ctrl\_int\_mret | Input | 1 | mret signal flag |
| int\_ctrl\_epc | Output | 32 | pc of the next instruction transfer to IFU |
| int\_ctrl\_pcen | output | 1 | enable signal of mtecv |
| int\_ctrl\_flush\_req | Output | 1 | flush require |

### Design Detail

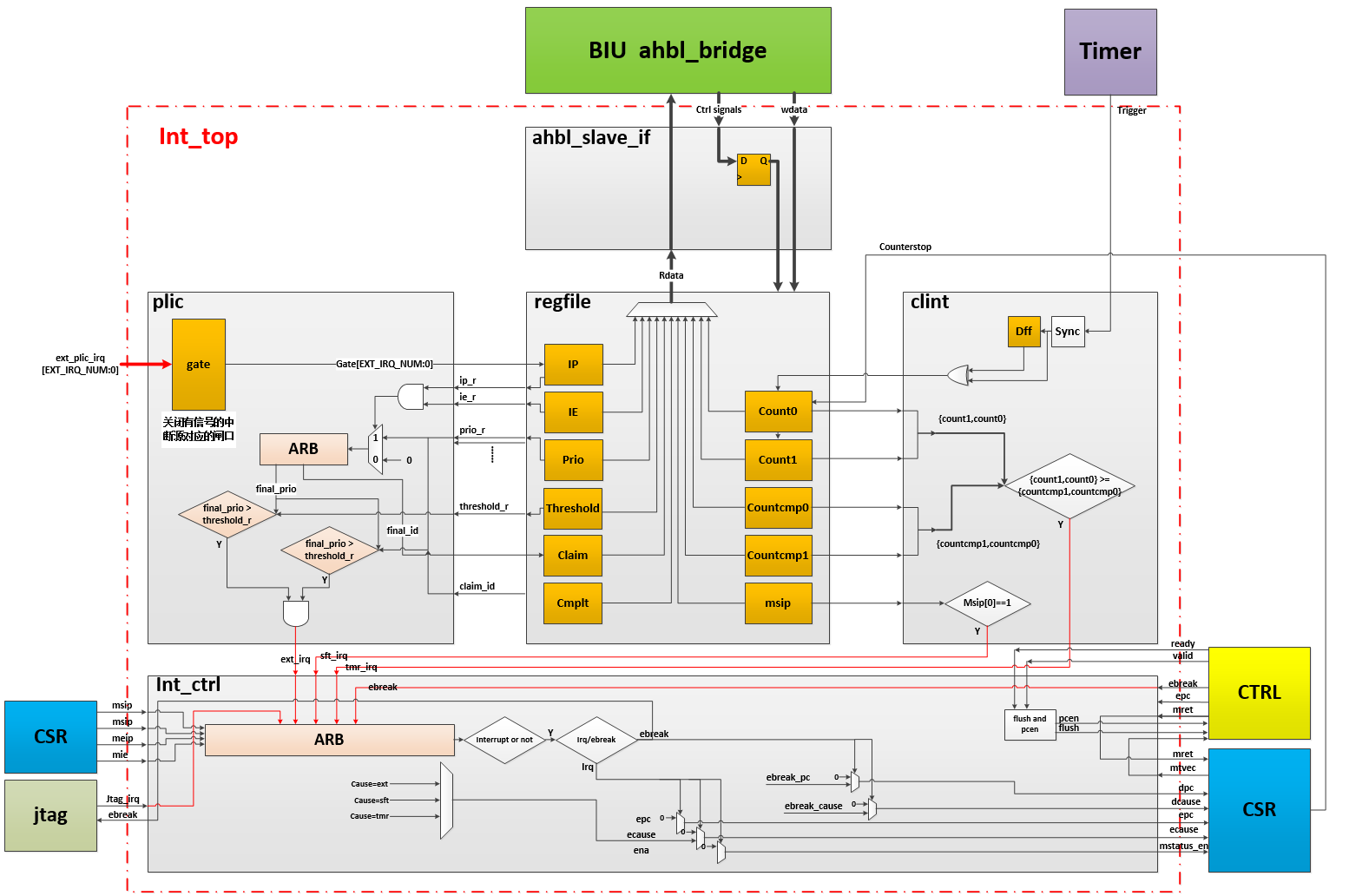


图2.11.2 int总模块设计细节示意图

**block plic design detail**

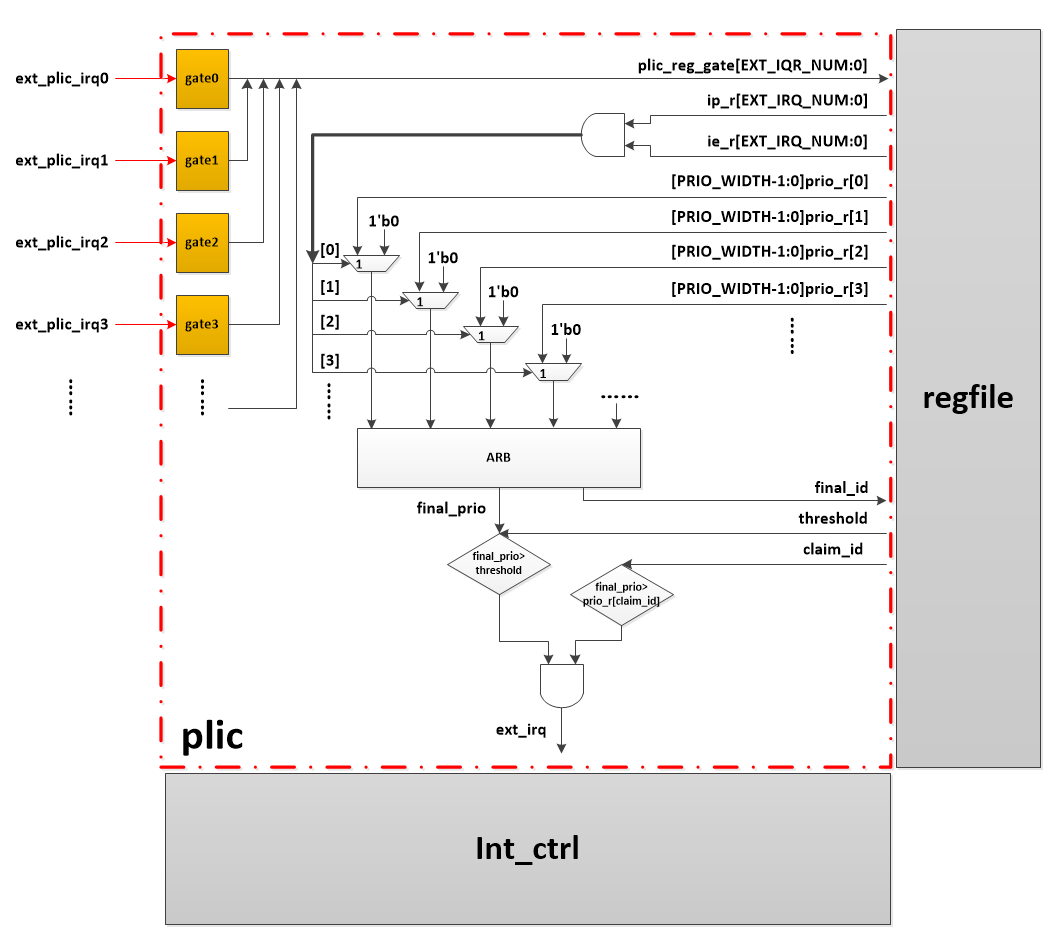


图2.11.3 plic模块设计细节示意图

**PLIC模块完成以下功能:**

接收来自多个外设的中断源信号（第0位为0），经过gate将所有不同触发方式的的中断信号转变为电平信号。并将每个中断源信号对应的gate传递给regfile模块中的ip寄存器组。gate还会接收来自regfile的cmplt\_id（上一条完成的中断的id），将该id对应的gate寄存器清零，以开始新一轮的中断仲裁。（gate寄存器软件不可读不可写，仅起到开关作用）

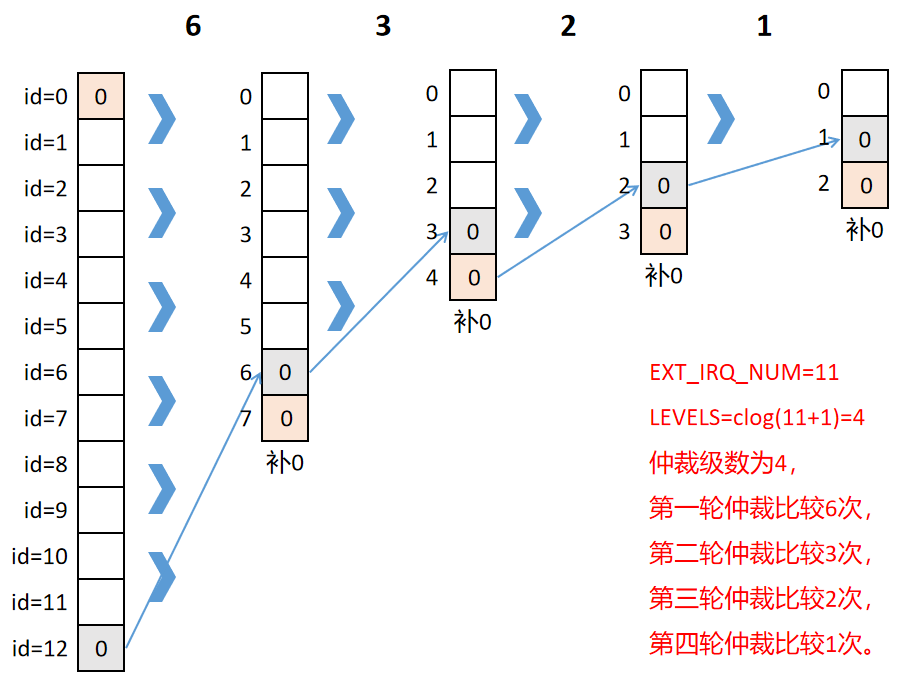
**仲裁过程：**

1. 接收regfile模块传输来的IP、IE、Theshold、prio、claim\_id寄存器值。每个中断源id对应的ip、ie寄存器值均为1时，其对应的中断优先级寄存器值有效，否则以优先级0进行后续仲裁。
2. 将各个中断源进行优先级仲裁，优先级数字越大，其优先级越高，优先级默认为8级（0-7），最高可配置32级优先级。当仲裁遇到优先级相同的情况，则取id较小的中断信号。优先级为0代表没有中断，id=0中断源的中断输入信号恒为0，当没有外部中断发生时，仲裁情况为n个优先级为0的中断进行仲裁，取id最小即id=0，因此没有外部中断时，仲裁的结果为优先级=0，id=0，表示没有中断，不会产生中断信号。
3. 将最终仲裁得到的中断信号，与优先级域值寄存器中的值和regfile传递的claim\_id对应的优先级进行比较，当仲裁的得到的优先级大于两者，则产生中断信号（即当前中断优先级大于域值且大于正在执行的中断的优先级）。

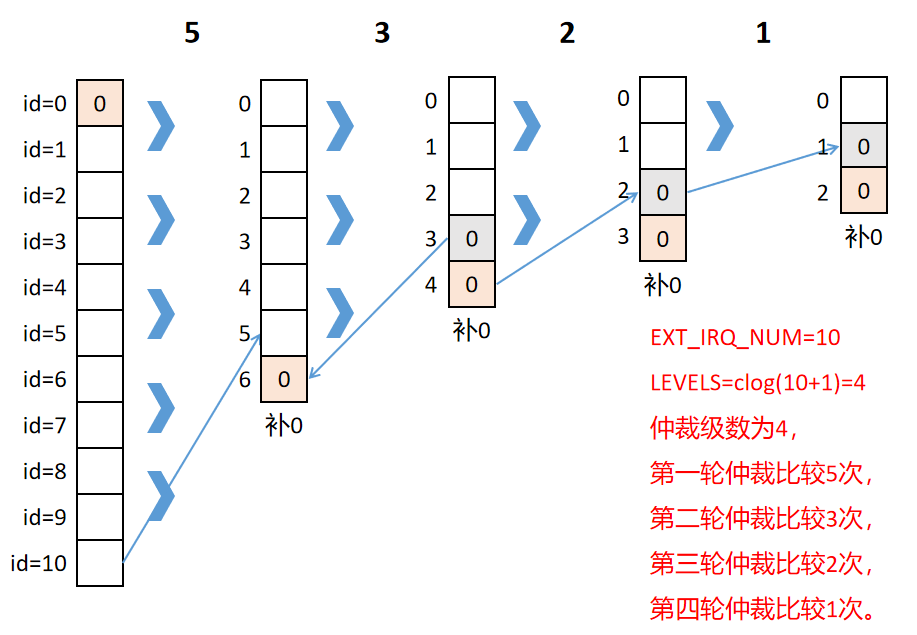
**仲裁硬件逻辑：**

N比特外部中断源信号，0位扩宽一位，大小为1'b0。若N+1为基数，则不变，若为偶数，则再高位扩宽一位，大小为1'b0，保证最后参与仲裁的数据位宽一定是个奇数，将扩宽后的位数记为N\_e。将该N\_e位信号两位两位比较，比较后多余的第一位挪到下一轮比较数据的次高位，并再最高位补0。该N\_e位信号仲裁级数level由log2(N+1)得到，每一级仲裁比较的次数由内部计算逻辑得到。如下述例子：

**eg1.**



**eg2.**



**\*每一轮仲裁后都会在最后补充一个0，是为了保证每轮仲裁完都有一个多余的数据可以用来移动至下一轮仲裁。如eg2中第3轮仲裁，若第3轮仲裁后没有补0，则第四轮仲裁就仅剩三个比特位，就无法两两比较。**

**block clint design detail**

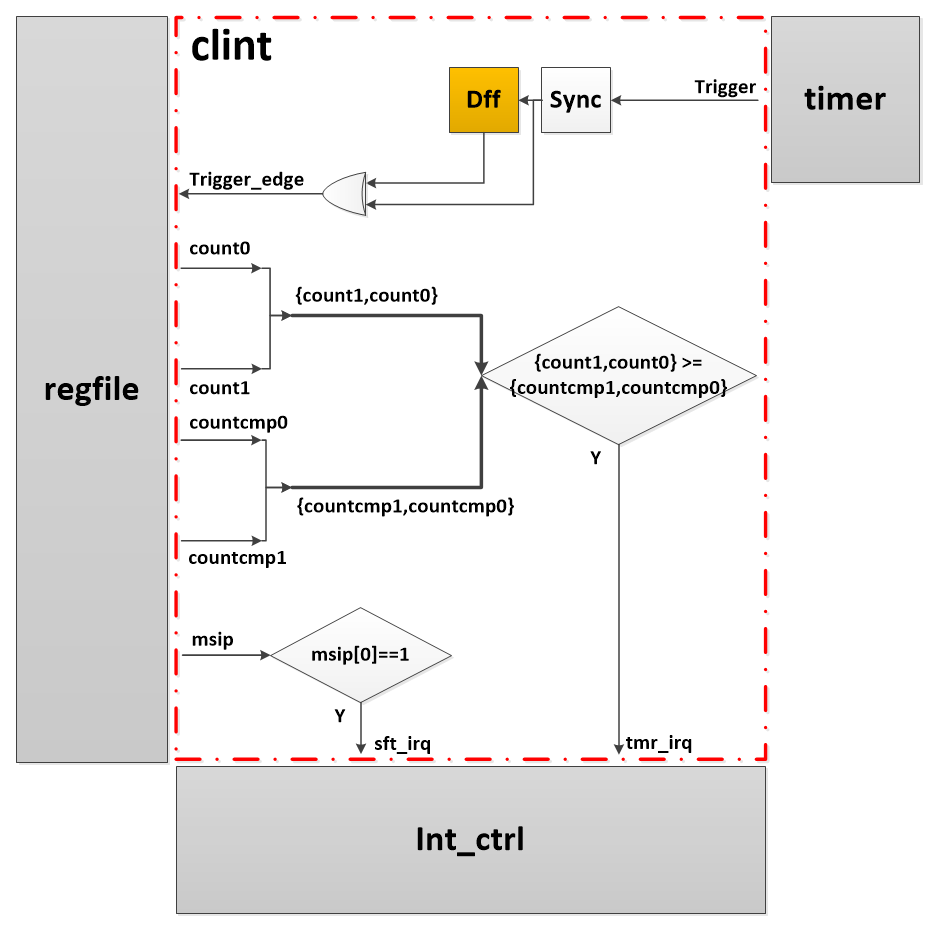


图2.11.4 clint模块设计细节示意图

**CLINT模块完成以下功能：**

1. 接收来自计时器模块的触发信号trigger，将该信号同步至系统时钟，然后经过采沿电路采其双沿，用该双沿信号进行计数，将计数值传输给regfile模块中的count0寄存器进行计数。（由于count1寄存器位计数的高32位，因此，count1的计数使能为count0计满的情况）
2. 接收来自regfile模块的count0，count1，countcmp0，countcmp1寄存器的值，当{count1,count0} >= {countcmp1,countcmp0}则产生计时器中断。一旦响应计时器中断，则需要软件手动改写countcmp寄存器中的值，使其大于count寄存器中的值，来消除中断。
3. 接收来自regfile模块中的msip寄存器的值，判断该寄存器值的0位，当第0位为1时，则软件中断信号置1。

**block regfile design detail**

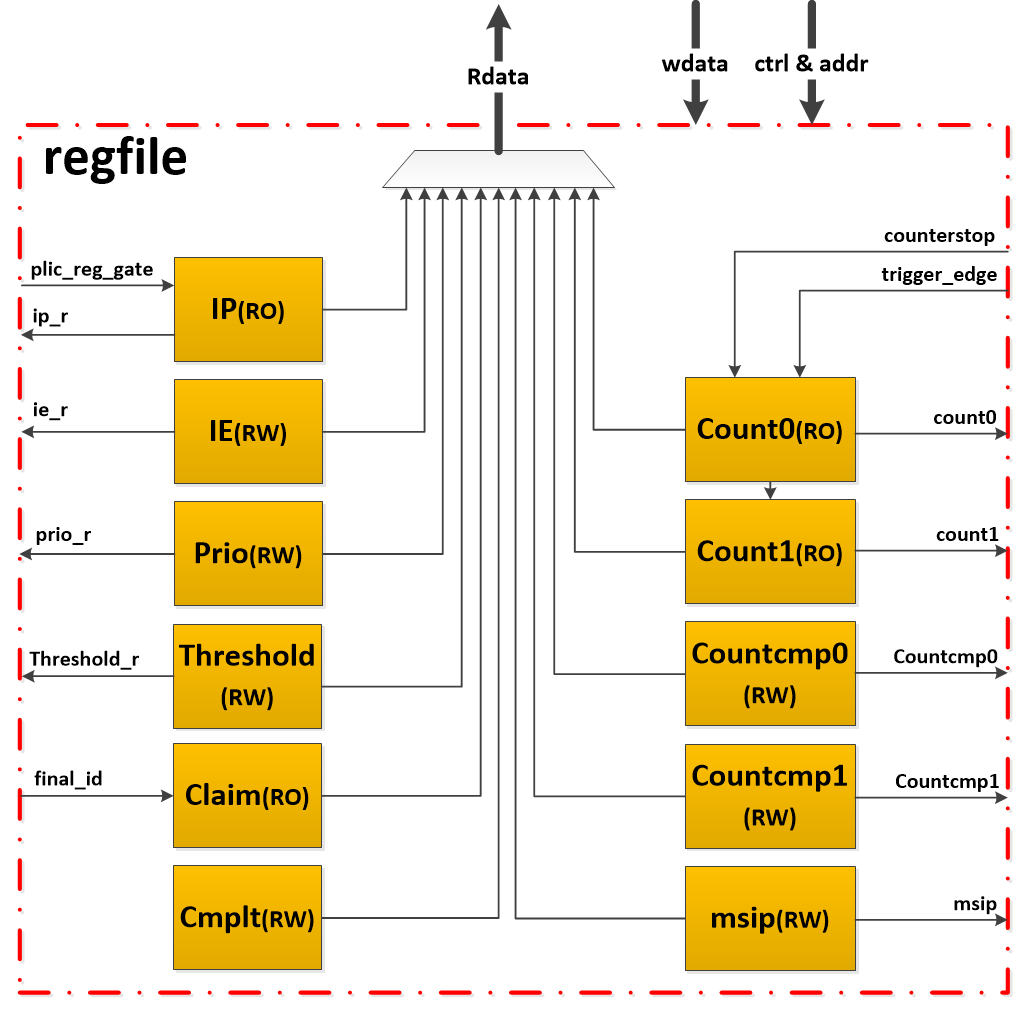


图2.11.5 regfile模块设计细节示意图

**regfile模块完成以下功能：**

映射ip（n个，每个中断对应一个ip寄存器），ie（8个，共256位对应最大256个外部中断），prio（n个，每个中断对应一个），threshold（机器模式仅一个），claim（机器模式进一个），cmplt（机器模式仅一个），count0，count1，countcmp0，countcmp1，msip寄存器，负责这些寄存器的软硬件读写。将寄存器的值通过地址判断，输出给ahbl\_slave\_if模块。该模块与biu总线的交互是通过ahbl\_slave\_if模块，以满足ahbl协议的时序要求。

有关所有寄存器的详细信息，见2.13.9。

**block int\_ctrl design detail**

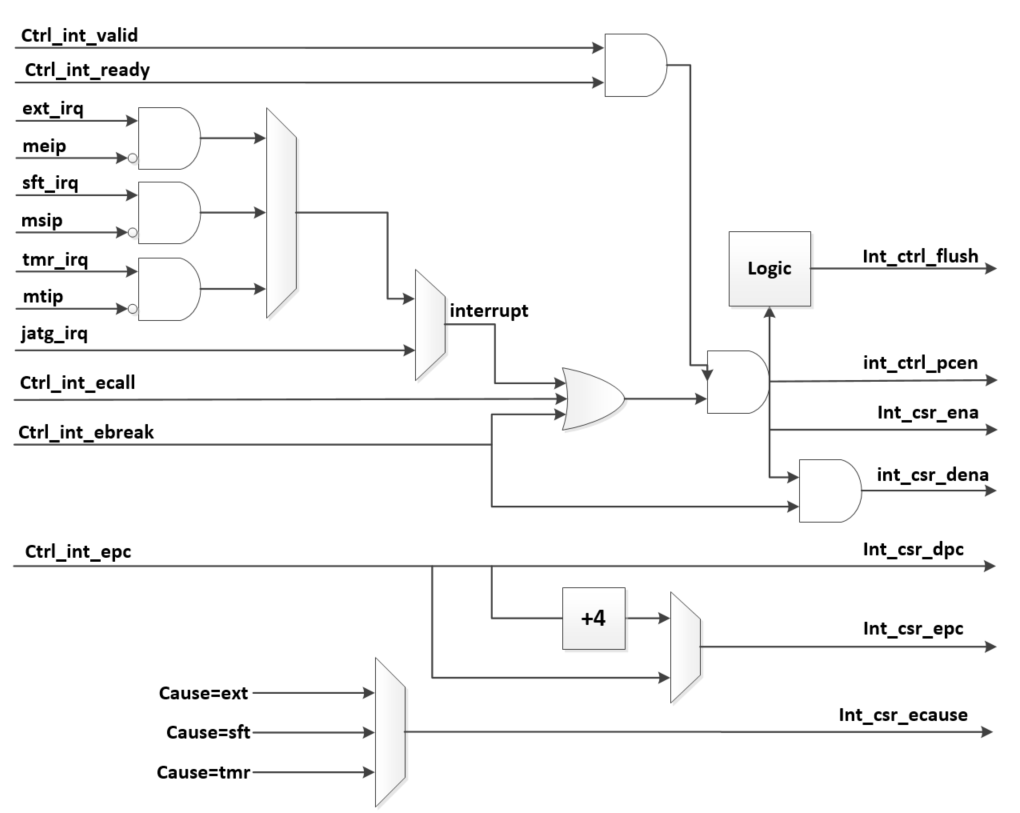


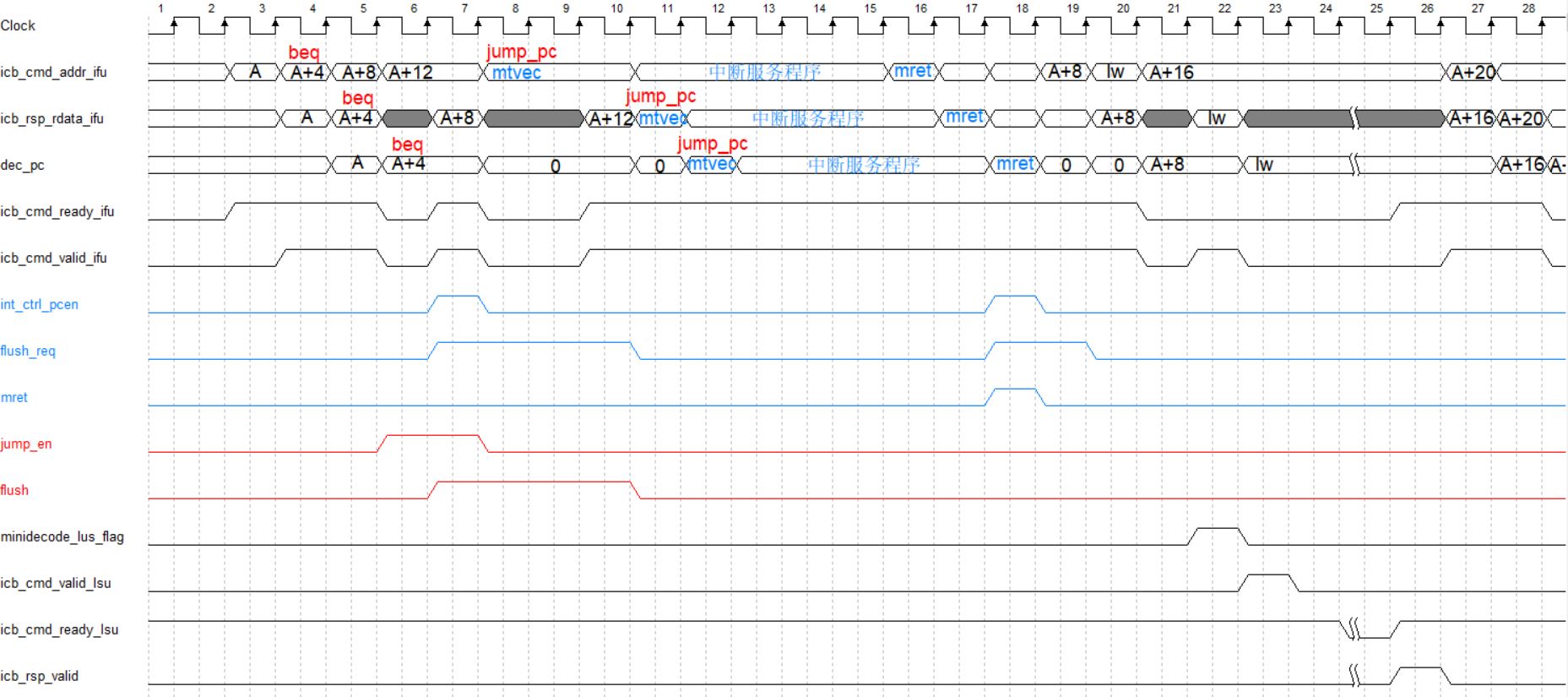
图2.11.5 regfile模块设计细节示意图（仅示意，修改多版有出入）

**int\_ctrl模块完成以下功能：**

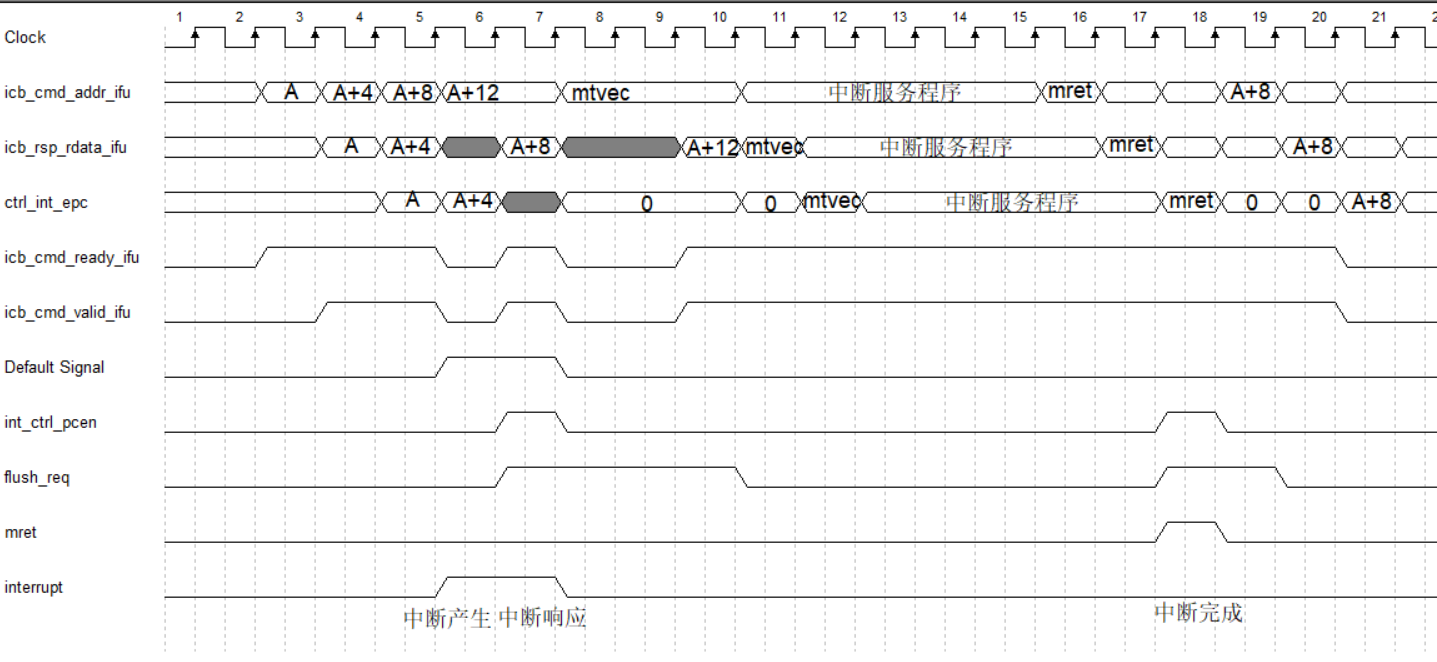
1. 接收来自PLIC、CLINT模块的三种中断信号，配合CSR寄存器中的屏蔽信号和全局中断信号，仲裁得到最终中断信号。将中断信号与dbg模块传递来的dbg\_irq仲裁，若无jtag中断信号，则将中断信号与ebreak信号做仲裁，得到最终的ebreak/中断信号。
2. 当仲裁得到最终中断信号有效后，会等待ctrl模块传递来的ready && valid信号为1，以确保上一条取指完成且下一条指令不是长指令。（长指令执行时，valid为1，ready为0，长指令执行过程中指令取回，会产生ready为1，valid为0，两者都为0的时候一定为非长指令执行的情况）。然后将pcen使能信号置高修改取指的pc为中断服务程序pc，并拉高流水线冲刷信号flush持续两个valid（冲刷下两条指令，而valid信号标志了一条指令的完成）。与此同时将ctrl模块传递来的epc，该中断对应的cause值，mststus修改使能信号传递给CSR寄存器模块进行保存现场以及修改mstatus寄存器的工作。（epc为32比特数据，表示断点pc，中断和异常的断点pc不同，中断的断点为当前执行pc+4，异常的断点为执行的pc本身）。
3. 中断结束后，也需要对核的pc进行修改并冲刷流水线，处理方式与中断相同。ctrl中传递mret信号给本模块，模块将mret信号传递给CSR寄存器模块，使其完成恢复现场的工作，并将CSR中mepc寄存器中的值传递给ctrl模块，冲刷流水线，恢复原先的正常指令流。值得注意的是，修改CSR寄存器的使能信号与pcen信号完全相同。
4. 若最产生的异常为ebreak，除了进行常规的修改pc冲刷流水线操作外，还会将该信号传递给jtag模块，并将ebreak对应的pc、dcaudse传输给csr寄存器，修改dcsr，dcause寄存器中的值，进入debug模式。idcause=3‘b001，dpc位ebreak指令本身的pc（因为ebreak属于异常）。
5. 若当前正在执行的是长指令（load，store），则不产生中断，直到长指令执行结束，此时ready和valid信号才会同时为1，中断才会响应。这是因为长指令发生时，由于该指令执行时间较长，容易产生数据相关性问题，总线将不进行常规的下一条指令取指，会等待该指令完成。因此流水线中取指流程在这一拍暂停了，此时模块如果冲刷两条指令的话，会将写入的中断服务程序pc也冲刷掉，造成错误。

**int\_ctrl timing diagram**

**总体时序：**

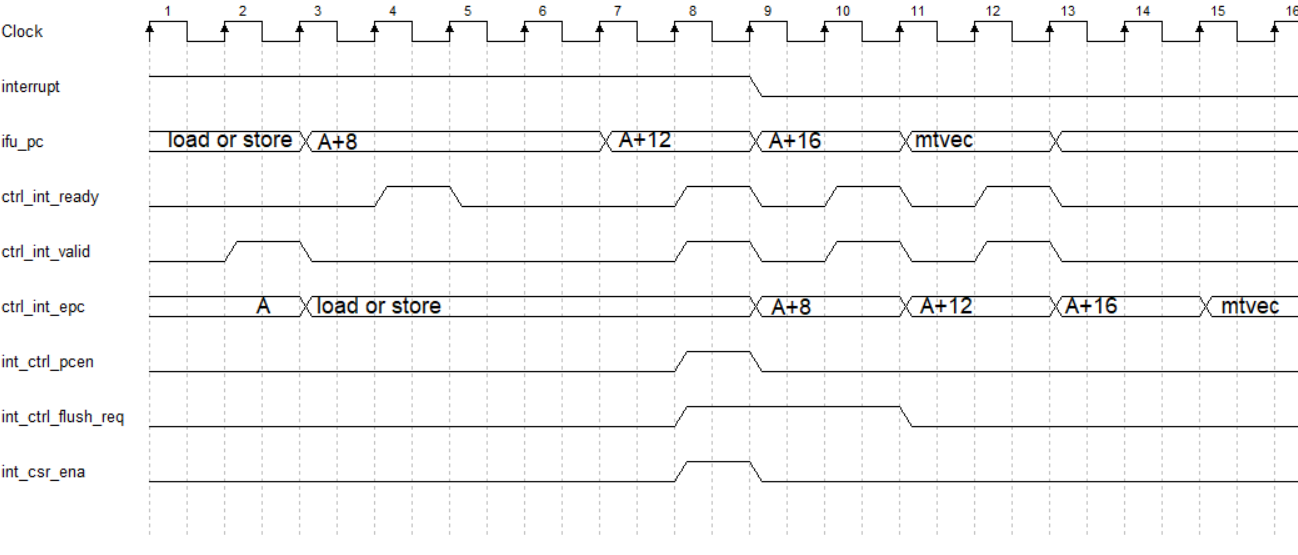
****

**一个标准的中断（或ebreak）响应流程：（\*interrupt信号外部不可见，表示当前有中断/异常）**



中断产生后，只有当ready && ready信号为1，才会传输pcen信号，即当正在进行的取指过程完成。pcen信号产生后取指即变成mtvec地址。由于长指令不进行中断，因此pcen信号为1时，valid信号一定也为1，则flush信号与pcen同时置1。flush信号将保持到下一次时钟上升沿采到valid高电平。当当前执行指令为mret时，同样会等待ready && valid为1，然后将pcen置高，flush保持两个valid。

**一个遇到长指令的中断（或ebreak）响应流程：（\*interrupt信号外部不可见，表示当前有中断/异常）**



中断产生后，等待ready && valid为1，才进行中断响应过程，pcen保持一个clk，flush信号保持两个valid。长指令执行过程中不存在ready && valid为1的情况，因此中断不会响应。

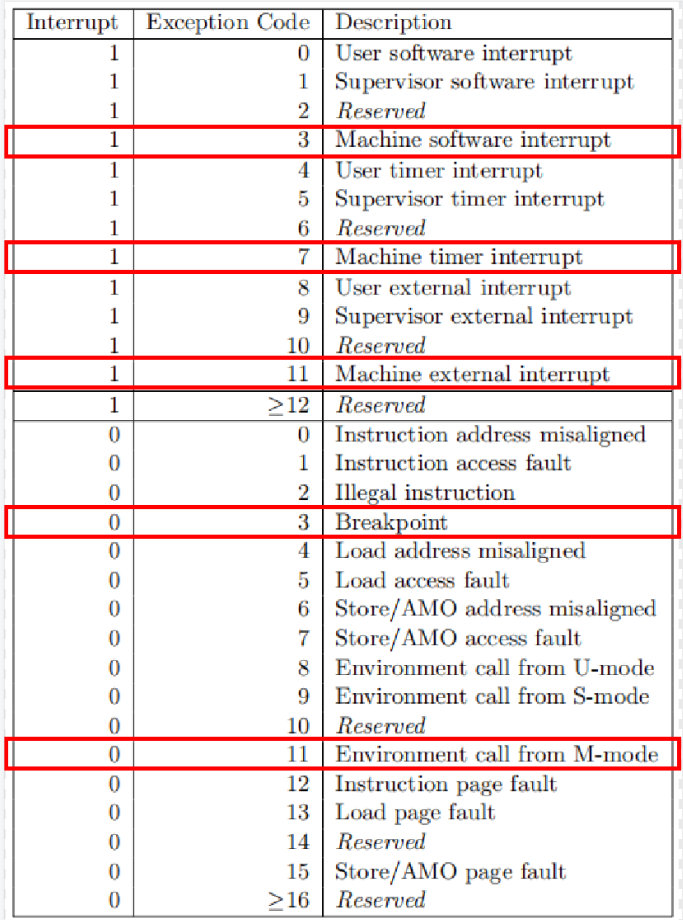
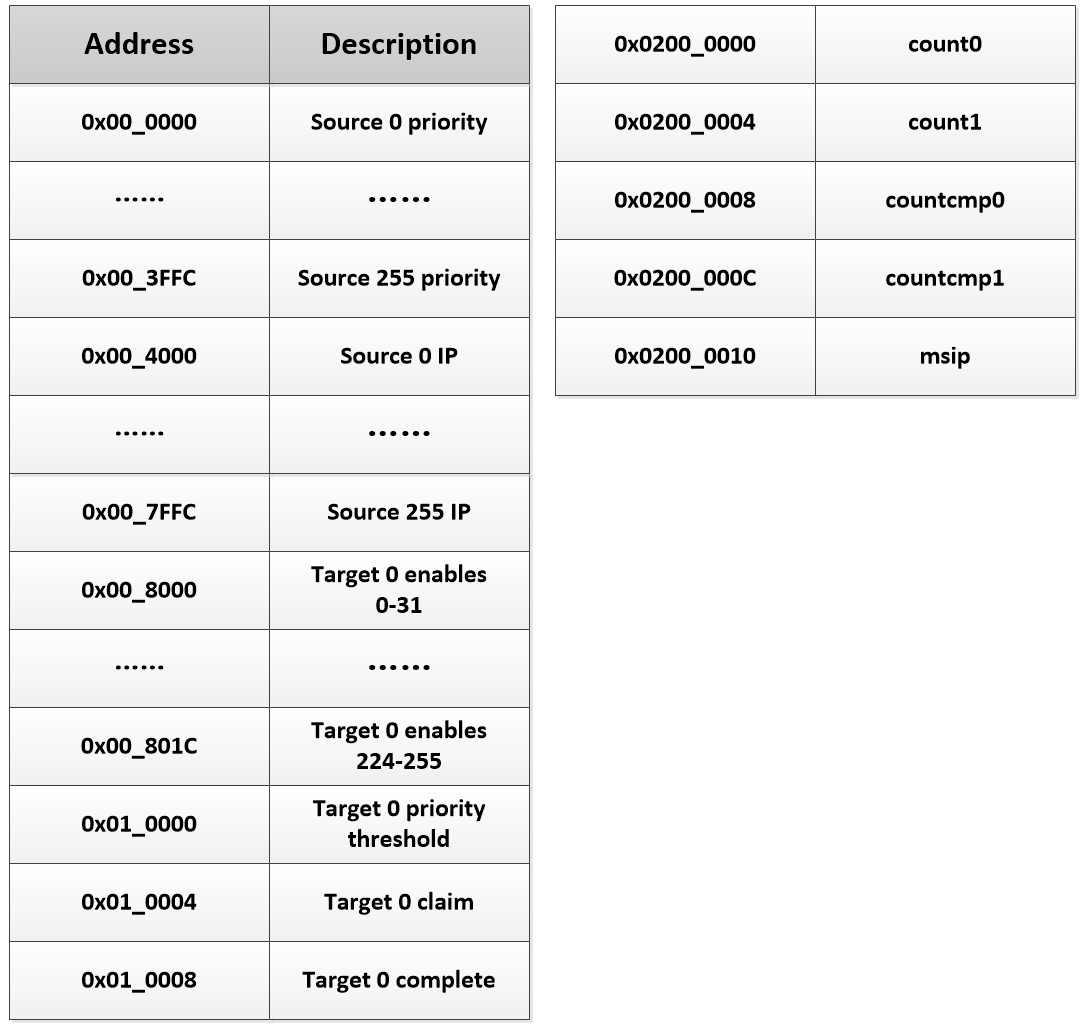


图2.12.6 rsic-V架构中规定的中断/异常id表单

**ahbl\_slave\_if模块完成以下功能：**

将总线上ahbl\_bridge传输的基于ahbl协议的相关信号，准换成regfile模块所使用的时序。将ahbl\_bridge传输的地址信号和多种控制信号寄存一拍，以符合ahbl的地址周期、数据周期传输模式。此外产生hresp、hready信号反馈给总线（hready恒为1，hresp恒为OK）。将总线传输的相关控制信号转换成寄存器组的读写使能信号。

### Registers



**IP寄存器：**

表示每个中断源的中断等待信号，当该plic模块中gate置1时，相对应的ip寄存器[0]置1，当中断被响应，则将该被响应中断对应的ip寄存器[0]置0，以开始下一轮的仲裁。

**IE寄存器：**

一共定义了8个IE寄存器，每个寄存器32比特代表了32个中断源的使能信号，总计最多支持255个外部中断源，中断源0预留，总计256个。软件可以一次性配置32个外部中断的使能信号。当IE寄存器响应比特位值为1时，表示该中断源信号有效，可以参与中断仲裁。

**prio寄存器：**

表示每个中断信号的优先级大小寄存器，当IE寄存器、IP寄存器对应的值为1时，该中断源的优先级有效。每个中断源对应了一个中断优先级寄存器。在寄存器表单中供预留了256个中断优先级寄存器。

**threshold0寄存器：**

表示中断目标0的优先级域值，当仲裁得到的中断源优先级大于该寄存器中的值，才会产生中断信号。由于目前仅支持机器模式，所以中断源仅有一个。在地址空间中预留了足够的空间，用于未来拓展更多的中断目标。

**claim0寄存器：**

表示中断目标0的中断响应寄存器，用于存放当前仲裁出的中断的id，注意这里仲裁出的中断id并不是正在响应的中断id，这是因为在中断响应的过程中会产生下一个正在等待的中断，也就是说这个寄存器中寄存的是实时仲裁后的中原源id。当软件决定响应中断时，该寄存器中保存的值会被软件读取，然后硬件会自行保存该读取的id，将gai该信号传输给IP寄存器关闭对应的ip。由于目前仅支持机器模式，所以中断源仅有一个。在地址空间中预留了足够的空间，用于未来拓展更多的中断目标。

**cmplt0寄存器：**

软件完成中断时，将完成的中断id写回该寄存器，寄存器将该id值，传输给plic模块打开对应的中断gate，使该中断可以进行下一次响应。由于目前仅支持机器模式，所以中断源仅有一个。在地址空间中预留了足够的空间，用于未来拓展更多的中断目标。

## Sub Block csr Architecture

### Block Diagram

### Block Descriptions

### Interfaces

|  |  |  |  |
| --- | --- | --- | --- |
| Signal | Direction | Width | Comments |
| rstn | Input | 1 | reset, active low |
| unknow | unknow | 32 |  |

## Appendix A Performance Power Area

*(Add after synthesis or physical design)*

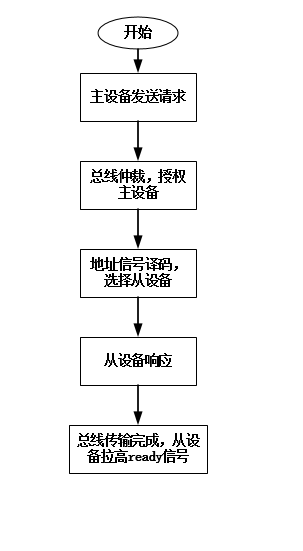
**Lausanne BIU Micro Architecture Spec**

# **Block BIU Overview**

## **Block BIU General Description**

BIU模块的功能是连接master和slave模块，实现实现处理器核的内部和外部通信。

## **System-Level Block Diagram**



## **Features**

* 采用和CPU相同的时钟；
* 支持多主多从连接，同一时刻只支持一主一从连接；
* 主设备采用优先级仲裁机制，优先级顺序：jtag，lsu，ifu；
* 分离的地址和数据通路，和master遵循icb协议；
* 根据各个master发出来的valid信号和cmd\_ready信号生成if2icb\_cmd\_ready, lsu2icb\_cmd\_ready, jtag2icb\_cmd\_ready三个信号；在每个时钟上升沿采三个ready信号，生成returned\_flag信号，用于判断返回数据返回哪个master；
* 和slave的接口支持ahbl总线接口；
* 当ifu模块传递来的minidecode\_lsu\_flag信号为高时，返回的if2icb\_rsp\_ready拉低，返回的数据和if2icb\_rsp\_vaild信号正常返回；
* HBURST只支持SINGLE传输，其它突发类型均不支持;
* HTRANS只支持IDLE和NONSEQ，其它传输类型均不支持，;
* HSIZE仅支持字传输，其它传输大小不支持;
* HRESP支持OKAY和 ERROR。

### **Unsupported Features**

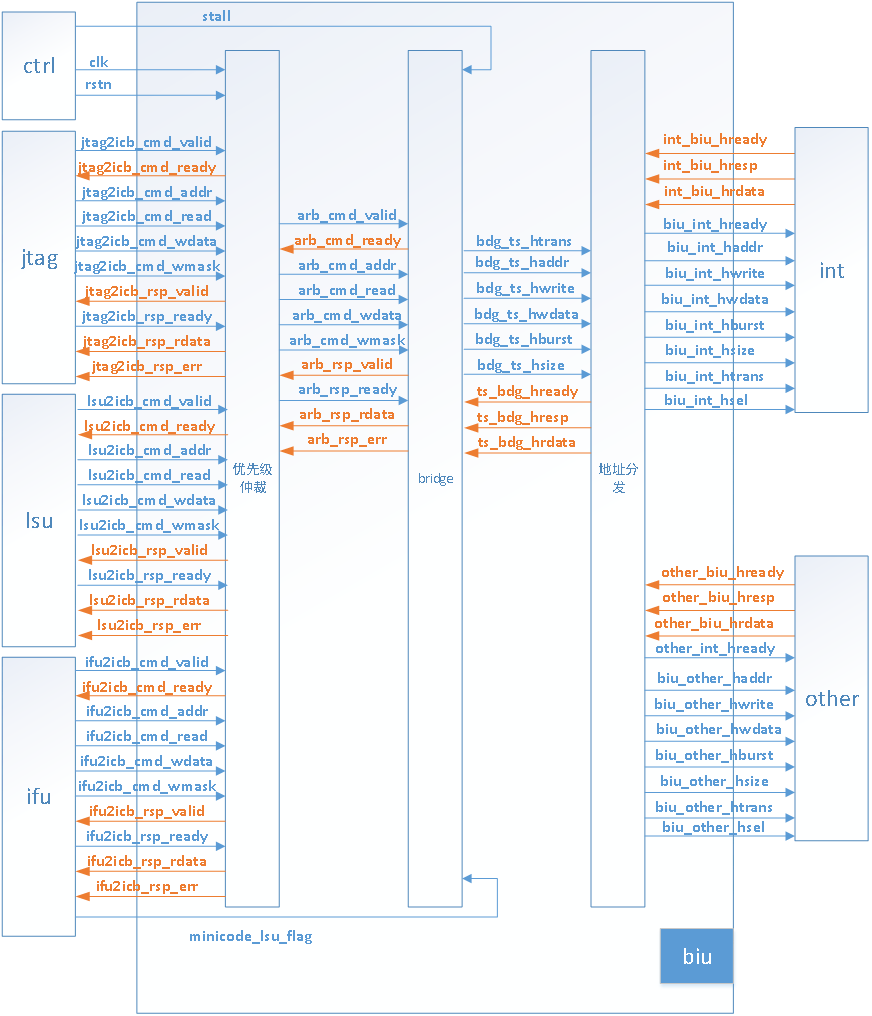
* 不支持乱序返回乱序完成；

## **Supported Standards**

* icb
* ahbl

# **Architecture**

## **Overview of Architecture**



### **Block Diagram**

### **Block Descriptions**

biu在同一个时钟域下，采用和CPU相同的时钟，分为三个模块，分别是arbiter，bridge，decoder模块，arbiter作用是优先级仲裁，bridge负责将信号转换，decoder负责地址分发。

### **Clock and Reset**

* 整个biu模块属于一个时钟域下，采用和CPU相同的时钟。

### **Interfaces**

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal** | **Direction** | **Width** | **Comments** |
| clk | Input | 1 | 时钟 |
| rstn | Input | 1 | 复位信号，低电平有效 |
| **master** | | | |
| icb\_cmd\_valid | Input | 1 | 主设备向从设备发送读写请求信号 |
| icb\_cmd\_ready | Output | 1 | 从设备向主设备返回读写接受信号 |
| icb\_cmd\_addr | Input | 32 | 读写地址 |
| icb\_cmd\_read | Input | 1 | 读或写操作的标志 |
| icb\_cmd\_wdata | Input | *32* | 写操作的数据 |
| icb\_cmd\_wmask | Input | 4 | 写操作的字节掩码 |
| icb\_rsp\_valid | Output | 1 | 从设备向主设备发送读写反馈请求信号 |
| icb\_rsp\_ready | Input | 1 | 主设备向从设备返回读写反馈接受标志 |
| icb\_rsp\_rdata | Output | 32 | 读反馈的数据 |
| icb\_rsp\_err | Output | *1* | 读或写反馈的错误标志 |
| **slave** | | | |
| slave\_biu\_hready | Output | 1 | 握手信号 |
| slave\_biu\_hresp | Output | 1 | 发给biu传输状态  0: ok,表示数据传输成功  1：error，表示数据返回失败 |
| slave\_biu\_hrdata | Output | 32 | 返回的数据 |
| biu\_slave\_hready | Input | 1 | 握手信号 |
| biu\_slave\_hsel | Input | 1 | 选中某一个slave |
| biu\_slave\_haddr | Input | 32 | 来自master传输的地址 |
| biu\_slave\_hwrite | Input | 1 | 写使能 |
| biu\_slave\_hwdata | Input | 32 | 来自master的写数据 |
| biu\_slave\_hburst | Input | 3 | 值为000，只支持单个字节传输（single） |
| biu\_slave\_hsize | Input | 3 | 值为010，数据传输宽度为32bits |
| biu\_slave\_htrans | Input | 2 | 传输类型，  00表示IDLE（空闲，总线没进行传输）状态，  10表示NONSEQ（一次单个数据传输，与上次传输无关）传输类型 |

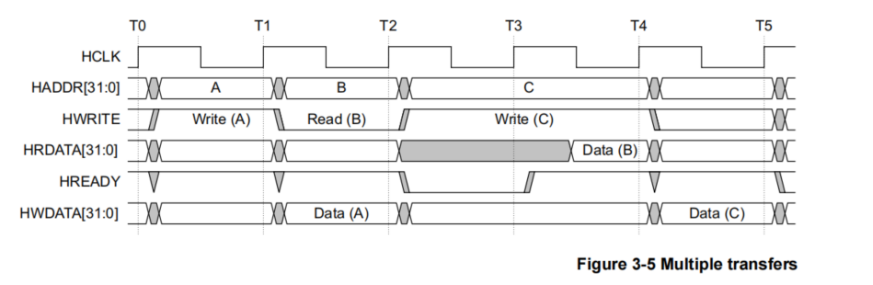
|  |  |  |  |
| --- | --- | --- | --- |
| **Signal** | **Direction** | **Width** | **Comments** |
| **ctrl** | | | |
| clk | Input | 1 | 时钟 |
| rstn | Input | 1 | 复位信号，低电平有效 |
| **jtag** | | | |
| jtag2icb\_cmd\_valid | Input | 1 | 主设备向从设备发送读写请求信号 |
| jtag2icb\_cmd\_ready | Output | 1 | 从设备向主设备返回读写接受信号 |
| jtag2icb\_cmd\_addr | Input | 32 | 读写地址 |
| jtag2icb\_cmd\_read | Input | 1 | 读或写操作的标志 |
| jtag2icb\_cmd\_wdata | Input | *32* | 写操作的数据 |
| jtag2icb\_cmd\_wmask | Input | 4 | 写操作的字节掩码 |
| jtag2icb\_rsp\_valid | Output | 1 | 从设备向主设备发送读写反馈请求信号 |
| jtag2icb\_rsp\_ready | Input | 1 | 主设备向从设备返回读写反馈接受标志 |
| jtag2icb\_rsp\_rdata | Output | 32 | 读反馈的数据 |
| jtag2icb\_rsp\_err | Output | *1* | 读或写反馈的错误标志 |
| **core - lsu** | | | |
| lsu2icb\_cmd\_valid | Input | 1 | 主设备向从设备发送读写请求信号 |
| lsu2icb\_cmd\_ready | Output | 1 | 从设备向主设备返回读写接受信号 |
| lsu2icb\_cmd\_addr | Input | 32 | 读写地址 |
| lsu2icb\_cmd\_read | Input | 1 | 读或写操作的标志 |
| lsu2icb\_cmd\_wdata | Input | *32* | 写操作的数据 |
| lsu2icb\_cmd\_wmask | Input | 4 | 写操作的字节掩码 |
| lsu2icb\_rsp\_valid | Output | 1 | 从设备向主设备发送读写反馈请求信号 |
| lsu2icb\_rsp\_ready | Input | 1 | 主设备向从设备返回读写反馈接受标志 |
| lsu2icb\_rsp\_rdata | Output | 32 | 读反馈的数据 |
| lsu2icb\_rsp\_err | Output | *1* | 读或写反馈的错误标志 |
| **core-if** | | | |
| if2icb\_cmd\_valid | Input | 1 | 主设备向从设备发送读写请求信号 |
| if2icb\_cmd\_ready | Output | 1 | 从设备向主设备返回读写接受信号 |
| if2icb\_cmd\_addr | Input | 32 | 读写地址 |
| if2icb\_cmd\_read | Input | 1 | 读或写操作的标志 |
| if2icb\_cmd\_wdata | Input | *32* | 写操作的数据 |
| if2icb\_cmd\_wmask | Input | 4 | 写操作的字节掩码 |
| minidecode\_lsu\_flag | Input | *1* | 暂停接收数据信号 |
| if2icb\_rsp\_valid | Output | 1 | 从设备向主设备发送读写反馈请求信号 |
| if2icb\_rsp\_ready | Input | 1 | 主设备向从设备返回读写反馈接受标志 |
| if2icb\_rsp\_rdata | Output | 32 | 读反馈的数据 |
| if2icb\_rsp\_err | Output | *1* | 读或写反馈的错误标志 |
| **int** | | | |
| int\_biu\_hready | Input | 1 | 握手信号 |
| int\_biu\_hresp | Input | 1 | 发给biu传输状态  0: ok,表示数据传输成功  1：error，表示数据返回失败 |
| int\_biu\_hrdata | Input | 32 | 返回的数据 |
| biu\_int\_hready | Output | 1 | 握手信号 |
| biu\_int\_hsel | Output | 1 | 选中slave |
| biu\_int\_haddr | Output | 32 | 来自master传输的地址 |
| biu\_int\_hwrite | Output | 1 | 写使能 |
| biu\_int\_hwdata | Output | 32 | 来自master的写数据 |
| biu\_int\_hburst | Output | 3 | 值为000，只支持单个字节传输（single） |
| biu\_int\_hsize | Output | 3 | 值为010，数据传输宽度为32bits |
| biu\_int\_htrans | Output | 2 | 传输类型，  00表示IDLE（空闲，总线没进行传输）状态，  10表示NONSEQ（一次单个数据传输，与上次传输无关）传输类型 |
| **other** | | | |
| other\_biu\_hready | Input | 1 | 握手信号 |
| other\_biu\_hresp | Input | 1 | 发给biu传输状态  0: ok,表示数据传输成功  1：error，表示数据返回失败 |
| other\_biu\_hrdata | Input | 32 | 返回的数据 |
| biu\_other\_hready | Output | 1 | 握手信号 |
| biu\_other\_hsel | Output | 1 | 选中slave |
| biu\_other\_haddr | Output | 32 | 来自master传输的地址 |
| biu\_other\_hwrite | Output | 1 | 写使能 |
| biu\_other\_hwdata | Output | 32 | 来自master的写数据 |
| biu\_other\_hburst | Output | 3 | 值为000，只支持单个字节传输（single） |
| biu\_other\_hsize | Output | 3 | 值为010，数据传输宽度为32bits |
| biu\_other\_htrans | Output | 2 | 传输类型，  00表示IDLE（空闲，总线没进行传输）状态，  10表示NONSEQ（一次单个数据传输，与上次传输无关）传输类型 |

### **Timing Diagram**

ICB

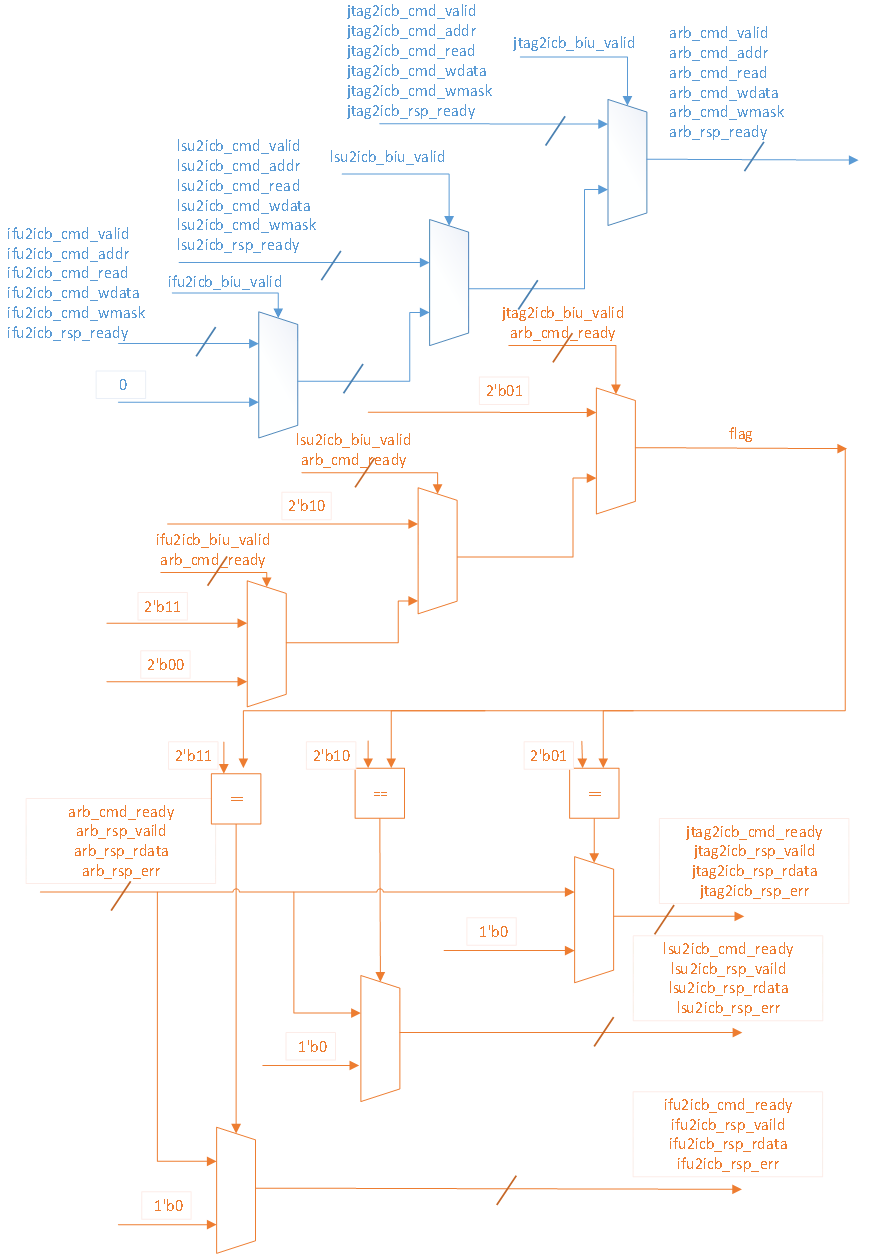
# 

AHBL



## **Sub Block Arbiter Architecture**

### **Arbiter Block Diagram**



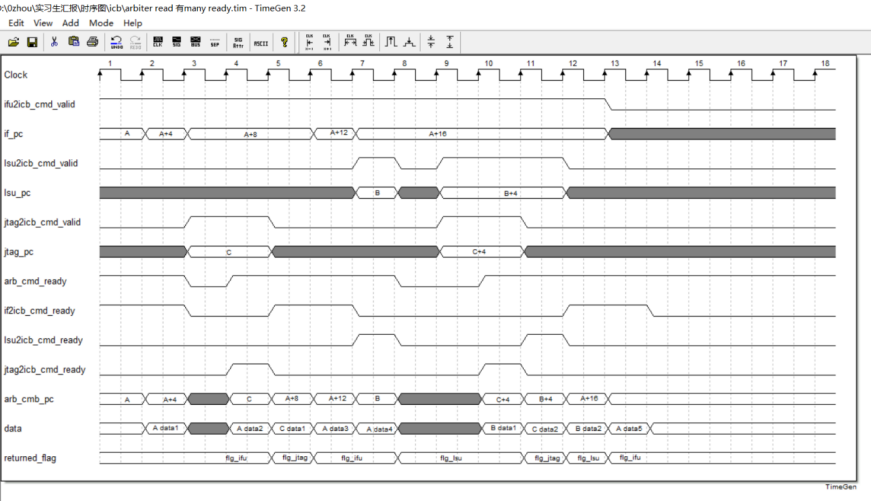
### **Block Descriptions**

master发出一个请求（valid）信号给仲裁器，采用优先级仲裁的方式，其中，jtag优先级最高，lsu优先级次之，ifu优先级最低，仲裁器接收来自某个master地址和数据等信号，同时将对应的master的ready信号拉高，并且将选中的master传来的数据传递给bridge；数据返回通道中，通过flag信号来判断将数据返回到哪个master模块。

### **Interfaces**

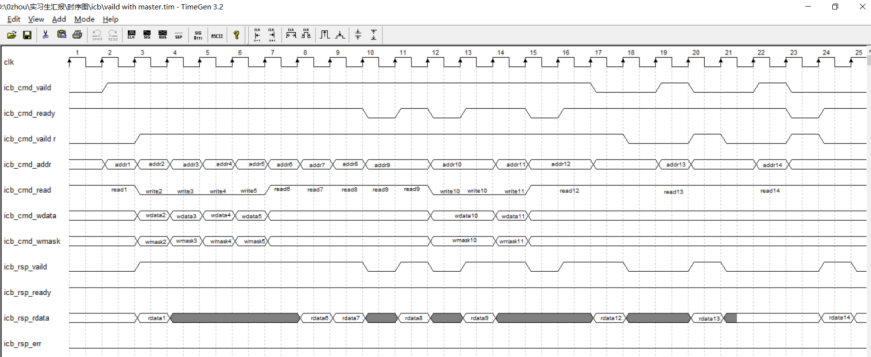
|  |  |  |  |
| --- | --- | --- | --- |
| **Signal** | **Direction** | **Width** | **Comments** |
| **jtag** | | | |
| jtag2icb\_cmd\_valid | Input | 1 | 主设备向从设备发送读写请求信号 |
| jtag2icb\_cmd\_ready | Output | 1 | 从设备向主设备返回读写接受信号 |
| jtag2icb\_cmd\_addr | Input | 32 | 读写地址 |
| jtag2icb\_cmd\_read | Input | 1 | 读或写操作的标志 |
| jtag2icb\_cmd\_wdata | Input | *32* | 写操作的数据 |
| jtag2icb\_cmd\_wmask | Input | 4 | 写操作的字节掩码 |
| jtag2icb\_rsp\_valid | Output | 1 | 从设备向主设备发送读写反馈请求信号 |
| jtag2icb\_rsp\_ready | Input | 1 | 主设备向从设备返回读写反馈接受标志 |
| jtag2icb\_rsp\_rdata | Output | 32 | 读反馈的数据 |
| jtag2icb\_rsp\_err | Output | *1* | 读或写反馈的错误标志 |
| **core - lsu** | | | |
| lsu2icb\_cmd\_valid | Input | 1 | 主设备向从设备发送读写请求信号 |
| lsu2icb\_cmd\_ready | Output | 1 | 从设备向主设备返回读写接受信号 |
| lsu2icb\_cmd\_addr | Input | 32 | 读写地址 |
| lsu2icb\_cmd\_read | Input | 1 | 读或写操作的标志 |
| lsu2icb\_cmd\_wdata | Input | *32* | 写操作的数据 |
| lsu2icb\_cmd\_wmask | Input | 4 | 写操作的字节掩码 |
| lsu2icb\_rsp\_valid | Output | 1 | 从设备向主设备发送读写反馈请求信号 |
| lsu2icb\_rsp\_ready | Input | 1 | 主设备向从设备返回读写反馈接受标志 |
| lsu2icb\_rsp\_rdata | Output | 32 | 读反馈的数据 |
| lsu2icb\_rsp\_err | Output | *1* | 读或写反馈的错误标志 |
| **core-if** | | | |
| if2icb\_cmd\_valid | Input | 1 | 主设备向从设备发送读写请求信号 |
| if2icb\_cmd\_ready | Output | 1 | 从设备向主设备返回读写接受信号 |
| if2icb\_cmd\_addr | Input | 32 | 读写地址 |
| if2icb\_cmd\_read | Input | 1 | 读或写操作的标志 |
| if2icb\_cmd\_wdata | Input | *32* | 写操作的数据 |
| if2icb\_cmd\_wmask | Input | 4 | 写操作的字节掩码 |
| if2icb\_rsp\_valid | Output | 1 | 从设备向主设备发送读写反馈请求信号 |
| if2icb\_rsp\_ready | Input | 1 | 主设备向从设备返回读写反馈接受标志 |
| if2icb\_rsp\_rdata | Output | 32 | 读反馈的数据 |
| if2icb\_rsp\_err | Output | *1* | 读或写反馈的错误标志 |
| **arb** | | | |
| arb\_cmd\_valid | Output | 1 | 仲裁出来的 主设备向从设备发送读写请求信号 |
| arb\_cmd\_ready | Input | 1 | 仲裁出来的 从设备向主设备返回读写接受信号 |
| arb\_cmd\_addr | Output | 32 | 仲裁出来的 读写地址 |
| arb\_cmd\_read | Output | 1 | 仲裁出来的 读或写操作的标志 |
| arb\_cmd\_wdata | Output | *32* | 仲裁出来的 写操作的数据 |
| arb\_cmd\_wmask | Output | 4 | 仲裁出来的 写操作的字节掩码 |
| arb\_rsp\_valid | Input | 1 | 从设备向主设备发送读写反馈请求信号 |
| arb\_rsp\_ready | Output | 1 | 主设备向从设备返回读写反馈接受标志 |
| arb\_rsp\_rdata | Input | 32 | 读反馈的数据 |
| arb\_rsp\_err | Input | *1* | 读或写反馈的错误标志 |

### **Timing Diagram**



仲裁模块时序图，cmd通道：

优先级仲裁机制，优先级顺序的是jtag，lsu，ifu模块，当优先级高的master发出valid信号的时候，在总线ready的时候可以优先传数据，优先级低的保持；比如，在第三个时钟上升沿的时候，ifu同时发起valid请求，arb\_cmd\_ready信号在第四个时钟上升沿才拉起来，第四拍接收jtag的地址信号，同时当拍jtag2icb\_cmd\_ready信号拉高，if模块的信号保持，直到第五个时钟上升沿的时候，jtag和lsu的valid信号都为低时，信号才被接收；若是读操作，data是应该在在下一拍同时ready为高时才被返回，此时需要一个returned\_flag信号，在时钟上升沿检测cmd\_valid信号和cmd\_ready信号，只有两个信号都为高时，显示对应master的flag。

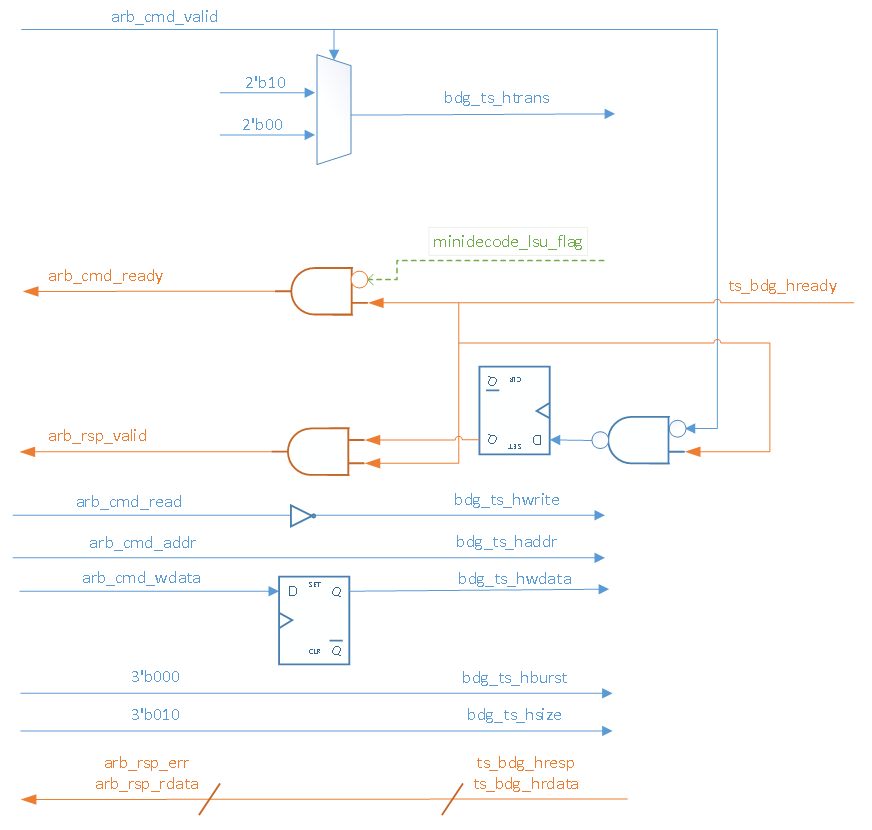


数据返回通道（rsp）：

由于和slave之间怎寻ahbl协议，在吃进去地址的下一拍数据才能返回，第二个到第九个时钟上升沿，icb\_cmd\_ready信号一直为高，无论是读是写，都是第一拍传地址，第二拍写入数据/读出数据，在数据传输完成的时候，icb\_rsp\_vaild信号拉高；看一下addr8的信号，在第九个时钟上升沿，cmd\_ready信号为高，地址传输给slave，然而，在下一个时钟cmd\_ready信号为低，数据没有返回，在第11个时钟时，cmd\_ready信号为高，数据隔了一拍才返回；返回通道的来自master的rsp\_ready信号一直拉高，重点要看的是rsp\_vaild信号，因为本设计的总线和master直接的接口信号采用icb总线协议，采用双向握手机制，而对于总线和slave之间的连接，采用ahbl总线协议，只有一个握手信号，所以对于数据返回通道的valid的信号，是通过cmd\_valid信号和cmd\_ready（两层含义：一个是数据已经传输完成，一个是可以接受新的地址信号）信号造出来的，首先在时钟上升沿采cmd\_valid为低时，cmd\_ready为高时，表明此时master 0占用总线，并没有数据传输，为idle状态，取反后即为数据传输状态，和cmd\_ready信号相与，得到的就是数据成功返回时的valid信号。

## **Sub Block Bridge Architecture**

### **Bridge Block Diagram**



### **Block Descriptions**

bridge将从仲裁模块传输来icb信号转化为AHBL接口信号，来自master的信号转换如下：htrans由valid信号转化而来，如果valid信号为高，htrans为2’b10，表示要进行一次单个数据传输；hwrite为来自主模块传输的写使能信号，通过read的信号翻转得来；haddr为来自主模块传输的地址；wdata为来自主模块传输的写数据；hburst赋值为3’b000，表示只支持单个字节传输；hsize赋值为3’b010，表示数据传输宽度为32比特；

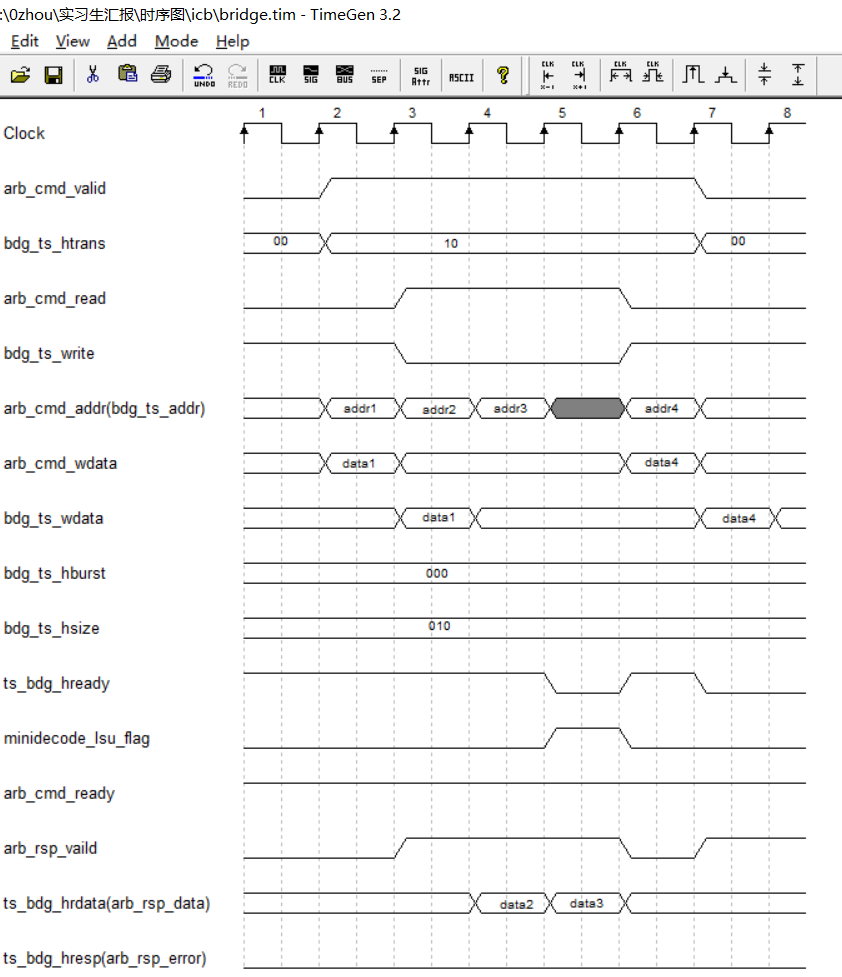
来自slave的信号转换如下：ts\_bdg\_hready和minidecode\_lsu\_flag信号控制arb\_cmd\_ready，当minidecode\_lsu\_flag信号拉高时，说明译码出来的指令是长指令，需要继续占用总线，ts\_bdg\_hready信号不返回master，总线由lsu占用；arb\_cmd\_valid和ts\_bdg\_hready信号共同控制返回通道的arb\_rsp\_valid信号，此处做了一个采沿处理，当vaild信号和hready信号同时为高的时候，数据返回给对应的slave；hresp转换为rsp\_err信号，如果hresp为1’b1的话，表示返回错误，rsp\_err信号拉高，否则拉低；rsp\_rdata为slave读反馈的数据。

转换完成后将数据传递给下一模块。

### **Interfaces**

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal** | **Direction** | **Width** | **Comments** |
| arb\_cmd\_valid | Input | 1 | 主设备向从设备发送读写请求信号 |
| arb\_cmd\_ready | Output | 1 | 从设备向主设备返回读写接受信号 |
| arb\_rsp\_valid | Output | 1 | 从设备向主设备发送读写反馈请求信号 |
| arb\_cmd\_read | Input | 1 | 读或写操作的标志 |
| arb\_cmd\_addr | Input | 32 | 读写地址 |
| arb\_cmd\_wdata | Input | 32 | 写操作的数据 |
| minidecode\_lsu\_flag | Input | *1* | 暂停接收数据信号 |
| arb\_rsp\_err | Output | 1 | 读或写反馈的错误标志  1表示返回错误  0表示传输完成且没有错误 |
| arb\_rsp\_rdata | Output | 32 | 读反馈的数据 |
| bdg\_ts\_htrans | Output | 2 | 传输类型，  00表示IDLE（空闲，总线没进行传输）状态，  10表示NONSEQ（一次单个数据传输，与上次传输无关）传输类型 |
| ts\_bdg\_hready | Input | 1 | 从设备反馈数据并可以进行下一次传输 |
| bdg\_ts\_hwrite | Output | 1 | 读或写操作的标志 |
| bdg\_ts\_haddr | Output | 32 | 访问地址 |
| bdg\_ts\_hwdata | Output | 32 | 写数据 |
| bdg\_ts\_hburst | Output | 3 | 值为000，只支持单个字节传输（single） |
| bdg\_ts\_hsize | Output | 3 | 值为010，数据传输宽度为32bits |
| ts\_bdg\_hresp | Input | 1 | 读或写反馈的标志  0: ok,表示数据传输成功  1：error，表示数据返回失败 |
| ts\_bdg\_hrdata | Input | 32 | 读反馈的数据 |

### **Timing Diagram**

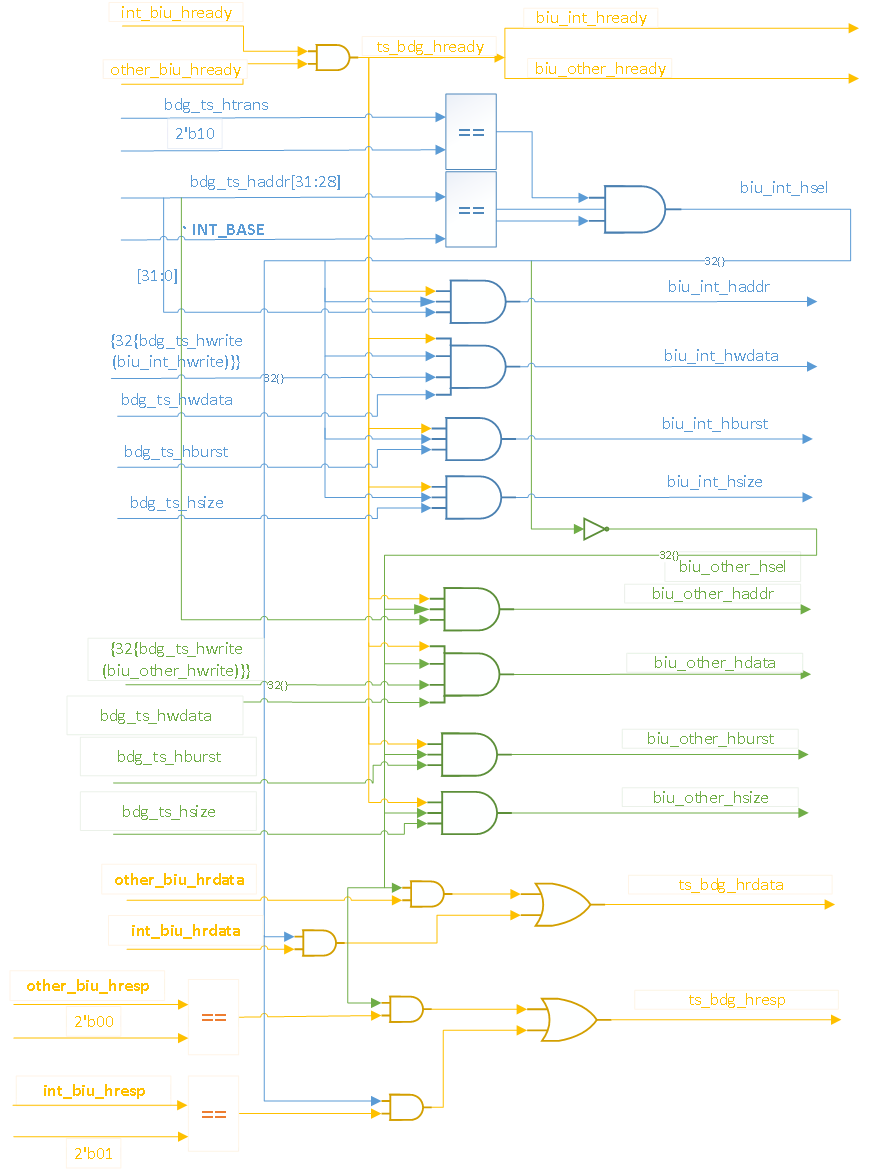


bridge时序图：

主要做了一个信号的转化，在这里需要注意的是minidecode\_lsu\_flag信号，此信号是来自于if的信号，说明上条指令是一条长指令（lw指令），还需要继续执行，不能吃进去新的pc地址，所以需要把hready信号拉低，如图所示：在第四个时钟周期，吃进去了ifu的addr3信号，在第五个时钟周期，data3取回，会传给ifu一个rsp\_valid信号，数据传给ifu的同时，会进入ifu的minidecode（组合逻辑）进行译码，发现是一个lw指令，需要继续执行，于是当拍ifu会将minidecode\_lsu\_flag信号拉高，此时总线在看到这个信号hready信号拉低，当拍并不吃新的地址，在下一拍ready重新拉高，接收lsu的valid信号，进行信号传输。

## **Sub Block Decoder Architecture**

### **Block Diagram**



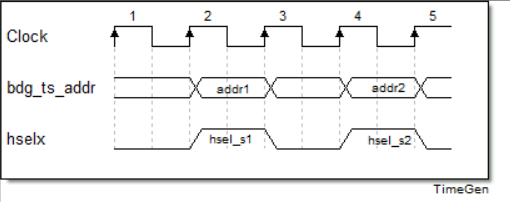
### **Block Descriptions**

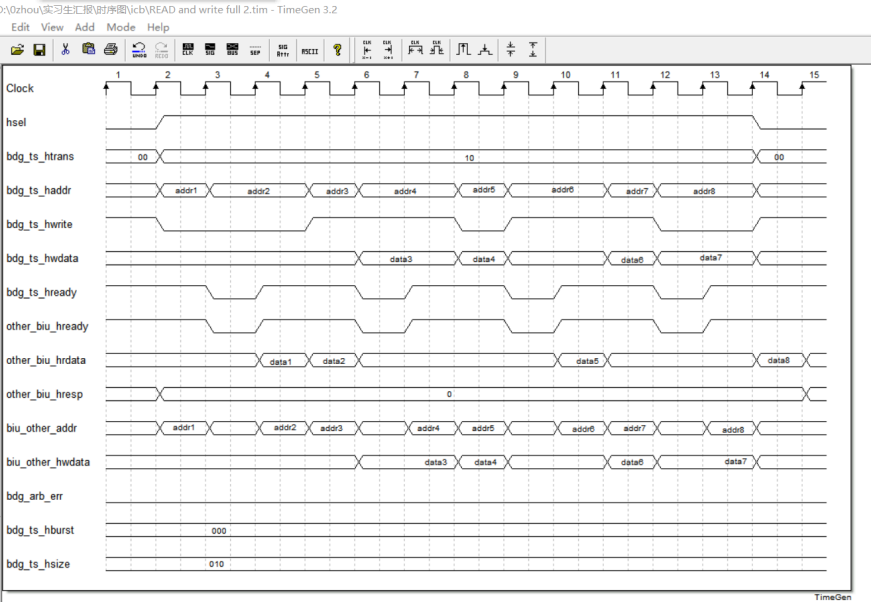
分发模块根据地址信号前四位来判断选哪个slave，如果地址是、INT BASE选中int模块，如果没有选中的情况，就选择other模块，选中slave后如果检测到slave的ready信号，会将传递给slave的htrans信号并赋值为2’b10，表示nonseq状态，即一次仅进行一个单数据传输，将地址传递给slave，slave根据传来的信号作出响应，将需要的指令取回并传递给master；如果写使能hwrite为高的话，写数据hwdata也被传递给slave，实现写功能；如果slave没有找到对应要取的指令，hresp会会返回一个1’b1的信号，表示发生错误，否则，传输完成后将返回一个1’b0，表示传输完成。

### **Interfaces**

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal** | **Direction** | **Width** | **Comments** |
| bdg\_ts\_htrans | Input | 2 | 传输类型，  00表示IDLE（空闲，总线没进行传输）状态，  10表示NONSEQ（一次单个数据传输，与上次传输无关）传输类型 |
| bdg\_ts\_hwrite | Input | 1 | 写使能 |
| bdg\_ts\_haddr | Input | 32 | 访问地址 |
| bdg\_ts\_hwdata | Input | 32 | 写数据 |
| bdg\_ts\_hburst | Input | 1 | 值为000，只支持单个字节传输（single） |
| bdg\_ts\_hsize | Input | 1 | 值为010，数据传输宽度为32bits |
| ts\_bdg\_hready | Output | 1 | 从设备反馈数据并可以进行下一次传输 |
| ts\_bdg\_hresp | Output | 1 | 发给传输状态  0: ok,表示数据传输成功  1：error，表示数据返回失败 |
| ts\_bdg\_hrdata | Output | 32 | 读反馈的数据 |
| **int** | | | |
| b\_biu\_hready | Input | 1 | 握手信号 |
| int\_biu\_hresp | Input | 1 | 发给biu传输状态  0: ok,表示数据传输成功  1：error，表示数据返回失败 |
| int\_biu\_hrdata | Input | 32 | 返回的数据 |
| biu\_int\_hready | Output | 1 | 握手信号 |
| biu\_int\_hsel | Output | 1 | 选中slave |
| biu\_int\_haddr | Output | 32 | 来自master传输的地址 |
| biu\_int\_hwrite | Output | 1 | 写使能 |
| biu\_int\_hwdata | Output | 32 | 来自master的写数据 |
| biu\_int\_hburst | Output | 3 | 值为000，只支持单个字节传输（single） |
| biu\_int\_hsize | Output | 3 | 值为010，数据传输宽度为32bits |
| biu\_int\_htrans | Output | 2 | 传输类型，  00表示IDLE（空闲，总线没进行传输）状态，  10表示NONSEQ（一次单个数据传输，与上次传输无关）传输类型 |
| **other** | | | |
| other\_biu\_hready | Input | 1 | 握手信号 |
| other\_biu\_hresp | Input | 1 | 发给biu传输状态  0: ok,表示数据传输成功  1：error，表示数据返回失败 |
| other\_biu\_hrdata | Input | 32 | 返回的数据 |
| biu\_other\_hready | Output | 1 | 握手信号 |
| biu\_other\_hsel | Output | 1 | 选中slave |
| biu\_other\_haddr | Output | 32 | 来自master传输的地址 |
| biu\_other\_hwrite | Output | 1 | 写使能 |
| biu\_other\_hwdata | Output | 32 | 来自master的写数据 |
| biu\_other\_hburst | Output | 3 | 值为000，只支持单个字节传输（single） |
| biu\_other\_hsize | Output | 3 | 值为010，数据传输宽度为32bits |
| biu\_other\_htrans | Output | 2 | 传输类型，  00表示IDLE（空闲，总线没进行传输）状态，  10表示NONSEQ（一次单个数据传输，与上次传输无关）传输类型 |

### **Timing Diagram**





和slave的接口时序：

首先是通过bdg\_ts\_addr信号前四位来选中某一个slave；信号传输遵循ahbl协议，最快是在吃进地址的下一拍返回数据，分析如下：信号1写使能为低，读数据，在第二个时钟周期ready为高，将addr1吃进去，master传来的信号变为信号2；在第三个时钟周期hready为低，表明slave并没有将数据取回，同时信号2保持；在第四个周期才将数据取回，hready信号拉高，将数据data1传出去，同时吃进去了addr2，信号2为读数据，master传来的信号变为信号3；在第五个时钟周期hready信号为高，读数据data2取回并传出，同时吃进去了信号3（hwrite为高，写数据）的addr3，master传来的信号变为信号4；同样的，在第六个时钟周期hready为低，说明slave没有准备好接收数据，data3和addr4要保持；在第七个时钟周期，hready信号为高，会将信号3的数据和信号4的地址吃进去；后面的过程类似。

### A.1 Timing

### A.2 Power

### A.3 Area

### A.4 Throughput and Latency