封面

CSR总线提议稿

* **修订记录**

|  |  |  |
| --- | --- | --- |
| **日期** | **修订人** | **说明** |
| 2019/09/24 | dmg | 初稿 |
|  |  |  |

目录

[1. 概述 4](#_Toc20748623)

[2. 接口时序 4](#_Toc20748624)

[3. 框图 5](#_Toc20748625)

[3.1 CSR特征 5](#_Toc20748626)

[4. 说明 6](#_Toc20748627)

[4.1 寄存器表格说明 6](#_Toc20748628)

[4.2 典型应用 11](#_Toc20748629)

[4.3 待定事项 14](#_Toc20748630)

## 概述

本文档描述CSR(control & status register)总线行为，CSR总线设计目标：

* 简化IP设计者寄存器读写操作；
* 为驱动开发者提供明确含义的寄存器文档，寄存器类型支持常见的SET\_REG(addr,data)，data = GET\_REG(addr)操作；

## 接口时序

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名** | **位宽** | **I/O** | **描述** |
| CSRValid | - | I | CSR信号有效 |
| CSRReady | - | O | CSR信号接收准备好 |
| bCSRWrite | - | I | CSR信号读写标志，1:写，0:读 |
| CSRAddr | [AW-1:0] | I | CSR信号地址 |
| CSRRdData | [DW-1:0] | O | CSR信号读数据 |
| CSRWrData | [DW-1:0] | I | CSR信号写数据 |
| CSRWrStrb | [DW/8-1:0] | I | CSR信号写数据字节选择控制 |

* CSR读



图 1 CSR读时序示意图

* CSR写



图 2 CSR写时序示意图

## 框图



图 3 CSR框图

### CSR特征

* 与IP核同时钟域，与SOC配置总线时钟域切换在CSRCdc模块完成；
* 支持CSRSlave模块任意拓展、级联；
* CSRSlaveReg模块中，Offset地址偏移量从0开始；每个寄存器绝对偏移量需逐级解读前级的CSRSlave模块地址分配。
* 支持pkg模式，大量寄存器配置操作可使用该模式；
* 支持shadow模式，当前IP还在运行时可提前写下次需要的配置；
* 支持memory读写操作

## 说明

### 寄存器表格说明

|  |  |  |  |
| --- | --- | --- | --- |
| **寄存器名** | CFG\_REG | **offset** | 0x0 |
| **type** | cfg | **spec** | - |
| **SW** | RW | **HW** | RW |
| **信号名** | **位段定义** | **default** | **comment** |
| **var\_name1** | [0] | **0x1** |  |
| **var\_name2** | [7:4] | **0x1\_0** | 图像高 |

对offset, type, spec, SW/HW，default分别做说明；



图 4 CSR SW/HW侧说明

##### offset

1. 第1个寄存器必填，后续可省略。自动化脚本工具会工具寄存器位宽，自动累加地址值；后续新填入了offset值，则从新的值继续累加。

2. offset是CSRSlaveReg模块自身的相对偏移量，从0x0地址开始；若需要知道对于第1级CSRSlave的偏移量，则逐级推到前级的CSRSlave模块地址分配； 若需要知道SOC系统的绝对偏移量，则再加上系统分配的基地址。

##### type

**type可填cfg, cmd, status, mem, slave, pkg， 下面对这几种类型分别作说明。**

###### cfg

```

//CSRSlaveReg.v

//PuCfg1

wire PuCfg1\_upen = (CSRValid && CSRReady) && bCSRWrite && CSRAddr==PuCfg1Offset;

wire [31:0] PuCfg1ValR;

reg  [31:0] rcPuCfg1Val;

always @(posedge Clk or negedge Rstn)

begin

    if( !Rstn )begin

        rcPuCfg1Val[7:0] <= default\_val;

    end

    else if( PuCfg1\_upen && CSRWrStrb[0] )begin

        rcPuCfg1Val[7:0] <= CSRWrData[7:0];

    end

~~else if( hw\_upen )begin//nearly useless~~

~~rcPuCfg1Val[7:0] <= hwset\_PuCfg1Val[7:0];~~

~~end~~

end

assign PuCfg1ValR = rcPuCfg1Val;

//IP.v

CSRSlave #() uCSRSlave

(

    ...

    //PuCfg1

    .PuCfg1ValR       (PuCfg1ValR),

    ...

);

```

如果寄存器类型为配置寄存器(cfg)，SW:RW, HW:RO; 通常软件端可读可写，硬件端只读，看到的代码片段如上。

代码片段删除线部分是硬件端指定可写(HW:RW)，先作保留，但很少使用，可忽略。

###### cmd+status

当寄存器类型为cmd+status时，CSRSlaveReg模块内部不包含实际的寄存器。

cmd是status状态寄存器的一个子集，都按照status状态寄存器作说明。

当status状态寄存器属性SW:WO, HW:RO, 表示软件端给硬件端发起一个脉冲信号(cmd)。

当status状态寄存器属性SW:RO, HW:WO, 表示硬件端拉一个电平信号到状态寄存器上，软件端可读。

cmd代码片段如下：

```

//CSRSlaveReg.v

//PuCmd1

wire        PuCmd1ValOEn = (CSRValid && CSRReady) && bCSRWrite && CSRAddr==PuCmd1Offset;

wire [31:0] PuCmd1ValO = CSRWrData;

//IP.v

CSRSlave #() uCSRSlave

(

    ...

    //PuCmd1

    .PuCmd1ValOEn       (PuCmd1ValOEn),

    .PuCmd1ValO         (PuCmd1ValO  ),

    ...

);

```

status作为debug寄存器时代码片段如下：

```

//CSRSlaveReg.v

//PuDbg1

wire [31:0] PuDbg1ValD;

reg  [31:0] rbcsr\_rddata;

always @\*

begin

    rbcsr\_rddata = 'b0;

    if( (CSRValid && CSRReady) && !bCSRWrite )begin

        case ( 1'b1 )

            CSRAddr==PuDbg1Offset:  rbcsr\_rddata = PuDbg1ValD;

            default: rbcsr\_rddata = 'b0;

        endcase

    end

end

assign CSRRdData = rbcsr\_rddata;

//IP.v

CSRSlave #() uCSRSlave

(

    ...

    //PuDbg1

    .PuDbg1ValD       (PuDbg1ValD), //input

    ...

);

```

###### slave

扩展一组CSR模块。

```

module CSRSlave

(

    CSRIf.Slave    CSRIfS ;

    CSRIf.Master   CSRIfM1;

    CSRIf.Master   CSRIfM2;

);

```

###### pkg

大量寄存器访问操作时，寄存器配置包先存放在外部存储中，然后通过访问外部数据效率更高的EMI(external memory interfale)总线去读取寄存器配置。

```

module CSRSlave

(

    CSRIf.Slave      CSRIfS  ;

    EmiRdIf.Master   EmiRdIfM;

);

```

###### mem

CSR总线的信号与mem比较契合，可以支持mem读写操作，暂时省略。

##### spec.

special选项，指定寄存器特殊类型。目前支持shadow+repeat。

* **shaodw**

shadow类型目的是掩盖寄存器配置的时间，但同时多了一个备份寄存器，增加了硬件开销。

当spec.选项值为shadow时，表示在当前PU正在工作时，也可以更新寄存器，更新的寄存器值先被掩蔽在另一寄存器中，当掩蔽寄存器更新使能(ShadowUpen)信号到来时，才把掩蔽寄存器的值赋给正在控制的配置寄存器。通常掩蔽寄存器更新使能(ShadowUpen)信号可接cmd发出的PU启动信号。

代码片段如下：

```

//CSRSlaveReg.v

//PuCfg1

wire PuCfg1\_upen = (CSRValid && CSRReady) && bCSRWrite && CSRAddr==PuCfg1Offset;

wire [31:0] PuCfg1ValR;

reg  [31:0] rcPuCfg1Valq;

reg  [31:0] rcPuCfg1Val;

always @(posedge Clk or negedge Rstn)

begin

    if( !Rstn )begin

        rcPuCfg1Valq[7:0] <= default\_val;

    end

    else if( PuCfg1\_upen && CSRWrStrb[0] )begin

        rcPuCfg1Valq[7:0] <= CSRWrData[7:0];

    end

    else if( hw\_upen )begin//nearly useless

        rcPuCfg1Valq[7:0] <= hwset\_PuCfg1Val[7:0];

    end

end

always @(posedge Clk or negedge Rstn)

begin

    if( !Rstn )begin

        rcPuCfg1Val <= default\_val;

    end

    else if( ShadowUpEn )begin

        rcPuCfg1Val <= rcPuCfg1Valq;

    end

end

assign PuCfg1ValR = rcPuCfg1Val;

//IP.v

CSRSlave #() uCSRSlave

(

    ...

    .ShadowUpEn       (ShadowUpEn),

    //PuCfg1

    .PuCfg1ValR       (PuCfg1ValR),

    ...

);

```

* **repeat**

比如需要N个系数的运算，每个系数的数值不同，但数据类型+位宽都是相同的。

此时可以指定repeat N，生成寄存器的脚本工具会自动扩展成N个寄存器，节省填写寄存器表格的时间。

填写说明

\*形式1：

|  |  |  |  |
| --- | --- | --- | --- |
| **寄存器名** | reg\_name | **offset** | 0x0 |
| **type** | cfg | **spec** | repeat 4 |
| **SW** | RW | **HW** | RO |
| **信号名** | **位段定义** | **default** | **comment** |
| **var\_name1** | [31:0] | **0** |  |

生成的信号为：

wire [3:0][31:0] reg\_name\_var\_name1 = {32’h0, 32’h0, 32’h0, 32’h0};

\*形式2：

|  |  |  |  |
| --- | --- | --- | --- |
| **寄存器名** | reg\_name | **offset** | 0x0 |
| **type** | cfg | **spec** | repeat 4, shadow |
| **SW** | RW | **HW** | RO |
| **信号名** | **位段定义** | **default** | **comment** |
| **var\_name1** | [31:0] | **0,0x1000,**  **0x2000** |  |

wire [3:0][31:0] reg\_name\_var\_name1 = {32’h2000, 32’h2000, 32’h1000, 32’h0};

\*形式3

|  |  |  |  |
| --- | --- | --- | --- |
| **寄存器名** | reg\_name | **offset** | 0x0 |
| **type** | cfg | **spec** | shadow, repeat 4 |
| **SW** | RW | **HW** | RO |
| **信号名** | **位段定义** | **default** | **comment** |
| **var\_name1** | [31:0] | 0x0000\_0000,  0x0000\_1000,  0x0000\_2000,  0x0000\_3000 |  |

wire [3:0][31:0] reg\_name\_var\_name1 = {32’h3000, 32’h2000, 32’h1000, 32’h0};

注1：对于spec项的shadow/repeat填写，与repeat N的default值填写，均遵从CSV（逗号分割符）规则；

注2：对于指定repeat N的default值，若填写的默认值数量小于N，则把填写的最后一个数值，复制多份到满足repeat N的指定。

##### SW/HW

软件端：可填写RW/RO/WO，指定该寄存器：可读可写/只读/只写。

硬件端：可填写RW/RO/WO，指定该寄存器：可读可写/只读/只写。

##### default

**支持%d, %x, 1024,0x4\_00（任意下划线）**

### 典型应用

##### cfg

|  |  |  |  |
| --- | --- | --- | --- |
| **寄存器名** | CFG\_REG | **offset** | 0x0 |
| **type** | cfg | **spec** | - |
| **SW** | RW | **HW** | RO |
| **信号名** | **位段定义** | **default** | **comment** |
| **var\_name1** | [0] | **0x1** |  |
| **var\_name2** | [7:4] | **0x1\_0** | 图像高 |

IP设计者看到CSR\_SLAVE模块端口:

```

CSRSlave #() uCSRSlave

(

    ...

    //CFG\_REG

.CFG\_REG\_var\_name1R       (CFG\_REG\_var\_name1R),

    .CFG\_REG\_var\_name2R       (CFG\_REG\_var\_name2R),

    ...

);

```

##### cmd

|  |  |  |  |
| --- | --- | --- | --- |
| **寄存器名** | CMD\_REG | **offset** | 0x10 |
| **type** | cmd | **spec** | - |
| **SW** | WO | **HW** | RO |
| **信号名** | **位段定义** | **default** | **comment** |
| **var\_name1** | [0] |  |  |
| **var\_name2** | [7:4] |  |  |

IP设计者看到CSR\_SLAVE模块端口:

```

CSRSlave #() uCSRSlave

(

    ...

    //CMD\_REG

.CMD\_REG\_var\_name1OEn     (CMD\_REG\_var\_name1OEn),

.CMD\_REG\_var\_name1O       (CMD\_REG\_var\_name1O),

.CMD\_REG\_var\_name2OEn     (CMD\_REG\_var\_name2OEn),

.CMD\_REG\_var\_name2O       (CMD\_REG\_var\_name2O),

    ...

);

```

##### dbg

|  |  |  |  |
| --- | --- | --- | --- |
| **寄存器名** | DBG\_REG | **offset** | 0x100 |
| **type** | status | **spec** | - |
| **SW** | RO | **HW** | WO |
| **信号名** | **位段定义** | **default** | **comment** |
| **var\_name1** | [0] |  |  |
| **var\_name2** | [7:4] |  |  |
| **var\_name3** | [31:16] |  |  |

IP设计者看到CSR\_SLAVE模块端口:

```

CSRSlave #() uCSRSlave

(

    ...

    //CMD\_REG

.DBG\_REG\_var\_name1D     (DBG\_REG\_var\_name1D),

.DBG\_REG\_var\_name2D     (DBG\_REG\_var\_name2D),

.DBG\_REG\_var\_name3D     (DBG\_REG\_var\_name3D),

    ...

);

```

##### intt

1. EXT\_INTT\_EN

|  |  |  |  |
| --- | --- | --- | --- |
| **寄存器名** | EXT\_INTT\_EN | **offset** | 0x0 |
| **type** | cfg | **spec** | - |
| **SW** | RW | **HW** | RO |
| **信号名** | **位段定义** | **default** | **comment** |
| **ExtInttEn** | [31:0] | **0xffff\_ffff** |  |

1. EXT\_INTT\_MASK

|  |  |  |  |
| --- | --- | --- | --- |
| **寄存器名** | EXT\_INTT\_MASK | **offset** | 0x0 |
| **type** | cfg | **spec** | - |
| **SW** | RW | **HW** | RO |
| **信号名** | **位段定义** | **default** | **comment** |
| **ExtInttMask** | [31:0] | **0x0000\_0000** | 1:屏蔽 |

1. EXT\_INTT\_SET

|  |  |  |  |
| --- | --- | --- | --- |
| **寄存器名** | EXT\_INTT\_SET | **offset** | 0x0 |
| **type** | status | **spec** | - |
| **SW** | WO | **HW** | RO |
| **信号名** | **位段定义** | **default** | **comment** |
| **ExtInttSet** | [31:0] | **0xffff\_ffff** |  |

1. EXT\_INTT\_STATUS

|  |  |  |  |
| --- | --- | --- | --- |
| **寄存器名** | EXT\_INTT\_STATUS | **offset** | 0x0 |
| **type** | status | **spec** | - |
| **SW** | RW | **HW** | RW |
| **信号名** | **位段定义** | **default** | **comment** |
| **ExtInttSta0** | [0] |  |  |
| **ExtInttSta31** | [31] |  |  |

```

integer i;

wire [31:0] ExtInttEnR;

wire [31:0] ExtInttMaskR;

wire [31:0] ExtInttSetO;

wire [31:0] ExtInttSetOEn;

wire [31:0] ExtInttStaO;

wire [31:0] ExtInttStaOEn;

reg  [31:0] rcExtInttSta;

reg  [31:0] rbhw\_setval;

reg  [31:0] rbsw\_setval;

always @\*

begin

    for ( i=0; i<32; i=i+1 )begin

        rbhw\_setval[i] = ExtInttEnR[i] && !ExtInttMaskR[i] && hw\_setval[i];

        rbsw\_setval[i] = ExtInttEnR[i] && !ExtInttMaskR[i] && ExtInttSetO[i];

    end

end

always @(posedge Clk or negedge Rstn)

begin

    if( !Rstn )begin

        rcExtInttSta <= 'b0;

    end

    for ( i=0; i<32; i=i+1 )begin

        if( hw\_seten[i] )begin

            rcExtInttSta[i] <= rbhw\_setval[i];

        end

        else if( ExtInttSetOEn[i] )begin

            rcExtInttSta[i] <= rbsw\_setval[i];

        end

        else if( ExtInttStaOEn[i] && ExtInttStaO[i] )begin

            rcExtInttSta[i] <= ~ExtInttStaO[i];

        end

    end//end of for

end

```

### 待定事项

##### AHB地址非字对齐确认

CSR为支持按byte写操作，需注意AHBSize=0(8bit)与AHBSize=2(32bit)时对应的功能。并且，假设AHBSize=2，AHBAddr=0x2，即地址非字对齐，需要确认此时AHB的准确行为。

##### pkg模式讨论

pkg时，需告诉EMI读地址与读长度，并踢起读命令。

一个简单的想法是：给出EMI读地址配置寄存器，给出EMI读长度配置寄存器，最后给出EMI启动寄存器(cmd)。

另一个想法是，对于CSRWrData的32bit，按

CSRWrData = {EMIRdCmd, EMIRdAddr[22:0], EMIRdLen[7:0]}拼接。

以上两种方法各有利弊，欢迎提出新的想法。

另外，需遵从一个思路，自动生成的寄存器文档，再次经过自动化脚本工具，应得到同样的结果。

++++++++++++++++++++++++++++++++++++++++++++++

##### csr pkg功能开发初步想法

说明在寄存器表格中，pkg对应的功能；

|  |  |  |  |
| --- | --- | --- | --- |
| **寄存器名** | imgp\_pkg | **offset** | 0x0 |
| **type** | pkg | **spec** | - |
| **SW** | WO | **HW** | RO |
| **信号名** | **位段定义** | **default** | **comment** |
|  |  |  | csr\_wdata对应pkg中的初始化指令(inst\_init)。 |

|  |  |  |  |
| --- | --- | --- | --- |
| **寄存器名** | imgp\_cfg | **offset** | 0x4 |
| **type** | cfg | **spec** | - |
| **SW** | RW | **HW** | RO |
| **信号名** | **位段定义** | **default** | **comment** |
| pkg\_addr | [31:0] | 0x0000\_0000 | 有csr pkg功能时，配置包在ddr中的基地址。 |

**\*csr\_pkg功能描述：**

1.要解决的问题：当有大量寄存器配置操作时，比如xm项目中ai指令流(约200KiB), ai权重数据(约0.5MiB), lsc table查找表(约16KiB);

2.解决方式：配置寄存器一般在amba框架下，是通过ahb或apb配置的，是比较低速的总线协议； 而axi是高性能访问ddr的总线协议，考虑把配置包放到ddr中，给硬件一个启动信号，让硬件通过类似axi协议从ddr读取配置包，并转换为csr总线配置内部子模块。

3.粗略思路：简单定义简易版指令动作，该指令动作包括: 初始化，配置，跳转，结束 这几种类型。配置包参考链表思路存储，不一定要全部连续的存在一片ddr空间内，可以多次读取，每次读取通过跳转指令指示下次读取位置，直到解析到结束指令才停止。

4.详细介绍：

(1)寄存器自动化脚本生成的rtl模块，只有一个csr slave接口，为保证自动化脚本简洁，考虑把配置包解析模块放在外面。

(2)csr配置包解析模块，需完成外部csr接口，内部pkg解析后产生的csr接口，两组接口2to1选择的mux功能。

(3)csr配置包解析模块启动，需要一个启动脉冲，一个初始化指令，一个初始化地址； 启动脉冲+初始化指令，由寄存器表格中type=pkg对应的寄存器项给出； 初始化地址，则由任一配置寄存器给出。

(4)配置包指令类型，在2.4.1小节详细介绍

(5)配置包解析rtl模块，见2.4.2小节

指令类型

1.初始化指令

2.配置动作指令

3.跳转动作指令

4.结束指令

5.ddr空间占用计算一览表

6.举例

**\*指令位段定义**

|  |  |  |  |
| --- | --- | --- | --- |
| [31:24] | [23:20] | [19:0] | 说明 |
| 保留 | optype | imm | - |

**\*初始化指令(inst\_init)**

|  |  |  |  |
| --- | --- | --- | --- |
| [31:24] | [23:20] | [19:0] | 说明 |
| 保留 | 4’b0000 | word\_4B\_len | 从ddr读多少个32bit，最大一次可读4MiB数据。 |

**\*配置动作指令(inst\_cfg)**

配置寄存器时考虑如下类型：

(1)单个寄存器，每次解析都是reg\_addr+reg\_data；

(2)多个寄存器，每次解析，reg\_addr递增4B；需要一个reg\_base\_addr;

(3)多个寄存器，每次解析，reg\_addr保持不变；需要一个reg\_base\_addr;

%定义optype=4’b01xx,

|  |  |  |  |
| --- | --- | --- | --- |
| [31:24] | [23:20] | [19:0] | 说明 |
| 保留 | 4’b0100 | reg\_num | 单个寄存器; reg\_num表示本次要解析多少个寄存器配置； |
| 保留 | 4’b0101 | reg\_num | 多个寄存器，地址递增； |
| 保留 | 4’b0110 | reg\_num | 多个寄存器，地址不变； |

%配置包中截取片段:

>>>

-单个寄存器 //optype=4'b0100;

inst\_cfg

@reg\_addr

reg\_data

@reg\_addr

reg\_data

…

#######################

-多个寄存器 //optype=4'b0101 or 4’b0110;

inst\_cfg

@reg\_addr

reg\_data

reg\_data

reg\_data

…

#######################

>>>

注1：inst\_cfg, reg\_addr, reg\_data都为32bit数据， 配置包中不会写@reg\_addr的’@’字符，这里仅用来说明。

注2：reg\_addr指相对于该模块的寄存器地址偏移，与总线地址偏移等无关。

**\*跳转动作指令(inst\_jmp)**

考虑绝对跳转，相对跳转

%定义optype=4’b001x,

|  |  |  |  |
| --- | --- | --- | --- |
| [31:24] | [23:20] | [19:0] | 说明 |
| 保留 | 4’b0010 | word\_4B\_len | 绝对跳转；指令后紧跟跳转地址；  word\_4B\_len表示从ddr读多少个32bit，最大一次可读4MiB数据。 |
| 保留 | 4’b0011 | word\_4B\_len | 相对跳转；指令后紧跟跳转偏移； |

%配置包中截取片段:

>>>

-绝对跳转 //optype=4'b0010;

inst\_jmp

addr\_base //跳向该地址

#######################

-相对跳转 //optype=4'b0011;

inst\_jmp

addr\_offset //偏移地址，涉及到解析包地址维护，可以通过上一次跳转地址+读取数据量计算得出； 也可以每次解析，地址累加，跳转时与前一种地址维护方式比较，可监测配置包出错。

#######################

>>>

**\*结束指令(inst\_exit)**

|  |  |  |  |
| --- | --- | --- | --- |
| [31:24] | [23:20] | [19:0] | 说明 |
| 保留 | 4’b1111 | - | 解析结束 |

**\*ddr空间占用计算一览表**

|  |  |  |  |
| --- | --- | --- | --- |
| 指令类型 | 寄存器数量 | 占用ddr空间(byte) | 备注 |
| 初始化 | - | 0 |  |
| 配置(单个寄存器) | N | 8\*N |  |
| 配置(多个寄存器) | N | 4+4\*N |  |
| 跳转 | - | 8 |  |
| 结束 | - | 4 |  |

**\*举例**

一个配置包的例子:

>>>

inst\_cfg

@reg\_addr

reg\_data

@reg\_addr

reg\_data

…

inst\_jmp

addr\_base

inst\_cfg

@reg\_addr

reg\_data

reg\_data

…

inst\_exit

>>>

stl\_csr\_pkg\_rd模块定义

**功能**： 支持csr pkg功能， 从ddr读配置包，并根据指令类型解析；

(1)考虑1：正在解析配置包时，若系统发起读写csr行为，可阻塞几个时钟周期，但需保证系统可正确读写csr动作。

*~~(2)考虑2：若apb数据位宽非32bit? 跳转指令寻址64bit地址空间？ //暂不重要~~*

(3)考虑3：为保证csr每周期写一次的吞吐率，解析数据需一次拿到64bit；

(4)考虑4：性能考虑，读超发，N个跳转指令紧接着存放；//会增加软件理解难度

**参数+接口**：省略时钟，复位信号；

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名** | **范围** | **典型值** | **描述** |
| CSR\_AW | [8,32] | 32 | csr总线地址位宽 |
| CSR\_DW | 32 | 32 | csr总线数据位宽 |
| BUS\_DW | 64,128,256.. | 128 | 读ddr请求的总线数据位宽 |
| **信号名** | **位宽** | **I/O** | **描述** |
| pkg\_start | 1 | I | 配置包启动脉冲信号 |
| pkg\_ inst\_init | CSR\_DW | I | 配置包初始化指令 |
| pkg\_base\_addr | 32 | I | 配置包初始化读，读的基地址 |
| pkg\_ongoing | 1 | O | 配置包解析，正在工作中 |
| pkg\_parse\_err | 1 | O | 配置包解析出错脉冲信号 |
| pkg\_parse\_err\_sta | 8 | O | 配置包解析出错状态，  [0]解析完成没有结束指令  [1]结束指令提前到来  [2]跳转指令超发数量，超过硬件限制；//推荐不超过4,，根据实现过程中存跳转指令的fifo深度决定；  [7:3]rsv； |
|  |  |  |  |
| sys\_csr\_if | - | if | csr总线的slave接口，来自系统侧 |
| cfg\_csr\_if | - | if | csr总线的master接口，接到自动化脚本生成的csr\_slave\*相关模块。 |
|  |  |  |  |
| bus\_ra\_valid | 1 | O | 读地址通道，读请求有效 |
| bus\_ra\_ready | 1 | I | 读地址通道，读请求接收准备好 |
| bus\_ra\_addr | 32 | O | 读地址通道，读请求地址 |
| bus\_ra\_bytelen | 32 | O | 读地址通道，读请求字节大小 |
| bus\_rd\_valid | 1 | I | 读数据通道，读返回数据有效 |
| bus\_rd\_ready | 1 | O | 读数据通道，读返回数据接收准备好 |
| bus\_rd\_data | BUS\_DW | I | 读数据通道，读返回数据 |

##### emi\_if定义

这里引出外部存储访问接口(externel memory interface)的说明。

1. emi基于axi的思路，支持burst，支持超发，地址/数据分离，读写通道分离。

2. emi简化axi的一些信号，直接面向IP设计者，表现为：

* 没有id信号;
* 读写端口非对称，可以只有读端口，只有写端口，M个读端口N个写端口；
* 支持多级EMI总线级联；
* 复位保护，保证总线干净；

在emi模块内部，接收各个IP模块读写端口后，进行仲裁、地址1KiB对齐割(addr\_split)、id分配后，再接上EMI2AXI等模块，访问外部数据。

##### 填写说明简记

\* **type: cfg, cmd, status, mem, slave, pkg**

**\* spec: shadow, repeat?**

**\* default\_val: 支持%d, %x, 0xff\_faaa\_f（任意下划线）**

**\* SW/HW： RO/WO/RW; 待支持R1C, W1C=WO(status)**

**\* offset：打头寄存器必填，后续可省略**

**\* 需支持byte写，offset按字对齐，一般 1字=32bit;**

**\* spec: repeat=N时，default填写: 遵从CSV(逗号分隔符)格式，当repeat数量为N时，支持N个数值CSV分隔填写，若填的默认值不足N个，则填的最后一个默认值复制扩展到相应数量。**

**\* 生成的信号名规则， “reg\_name”+”\_”+”var\_name”; 在工具内部会保证reg\_name不重复，所以生成的信号名也不会重复；**