硬件调试（HAD）参考手册

平头哥

声明：

平头哥半导体有限公司(Pingtouge Semiconductor Co., Ltd.)保留本文档的所有权利。
本文档的内容有可能发生更改、更新、删除、变动，恕不另行通知。

公司地址：杭州市西湖区西斗门路 3 号天堂软件园 A 座 15 层
邮政编码：310012
电话：0571-88157059
传真：0571-88157059-8888
E-mail：info@c-sky.com

版本历史

版本号	日期	说明	作者
V1.0	2011-12-02	介绍调试接口的定义和寄存器等相关内容。	平头哥半导体有限公司
V1.1	2017-04-13	增加通过 HAD 调试程序的示例	平头哥半导体有限公司
V1.2	2019-05-09	增加 E902 系列描述	平头哥半导体有限公司
V1.3	2019-10-15	增加 3.1~3.5 寄存器描述	平头哥半导体有限公司

目录

图目录

表目录

1 HAD 概述

HAD（Hardware Assisted Debug）模块存在于 C-SKY 所有系列 CPU 中，用户可以通过 HAD 模块下载可执行程序到存储器中并 CPU 开始执行，并可以实时了解到 CPU 的寄存器和存储器内容等当前状态，和其他片内设备信息。

HAD 模块通过 JTAG 接口与外部的调试器通信，JTAG 通信协议与 IEEE-1149.1 标准兼容，可以同已有的 JTAG 部件或独立的 JTAG 控制器集成在一起。在低成本的 CK8XX 系列 CPU（CK801，CK802，CK803S）和 E902 中，HAD 实现了使用 2 根信号线进行通信的 JTAG_2 调试接口以节省引脚，高性能的 CPU 中使用 JTAG_5 调试接口。

HAD 模块主要有以下特性：

- 使用标准的 JTAG 接口进行调试，同时支持两线制的 JTAG_2 调试接口
- 可以非侵入式获取 CPU 状态
- 可以设置多个内存硬断点
- 检查和设置 CPU 寄存器的值
- 检查和改变内存值
- 快速下载程序
- 可进行指令单步执行或多步执行
- 可以在 CPU 复位之后或在普通用户模式下进入调试模式
- 支持同步和异步的调试请求
- CK801，CK802 和 E902 调试单元支持直接访问内存，详见相关核的紧耦合 IP 手册

调试器对 HAD 中的资源的存取不会中断处理器的正常执行，对一些特定的对象的存取要求处理器处于调试模式下。如果需要使用处理器中的资源，HAD 控制逻辑会向处理器发出一个调试请求，处理器执行完当前的指令，保存流水线的信息，然后进入调试模式。调试请求会使处理器打断 STOP，DOZE 和 WAIT 命令的执行。

C-SKY CPU 的调试工作是调试软件（gdb），调试代理服务程序（DebugServer），调试器（CKLink）和 HAD 模块一起配合完成的，HAD 模块在整个 CPU 调试环境中的位置如图 1-1 所示。其中，调试软件和调试代理服务程序通过网络互联，调试代理服务程序与调试器通过 USB 连接，调试器与 CPU 的 HAD 模块以 JTAG 模式通信。

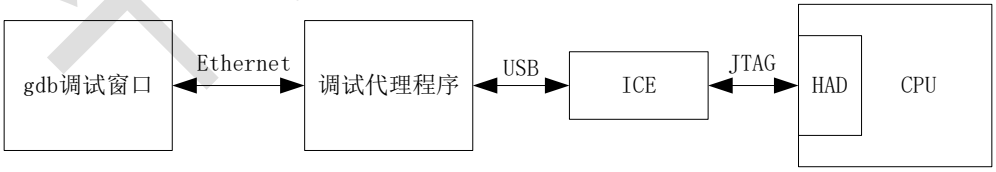

图 1-1 HAD 在整个 CPU 调试环境中的位置

2 HAD 的外部接口

HAD 模块与外部的接口主要有 JTAG 相关的接口信号和调试相关的接口信号。表 2-1 列出了 CSKY 各系列 CPU 中的 HAD 实现中的外部接口信号。

表 2-1 HAD 与外部的接口信号

信号名	方向	CK5xx CK6xx	CK801	CK802	CK803S	CK804	CK805	CK807	CK810	E902
(pad_biu_dbgrq_b)	输入		×	×	×	×	×			×
(pad_biu_brkrq_b[1:0])	输入		×	×	×	×	×	×	×	×
(biu_pad_debug_b /sysio_pad_dbg_b) (802/801)	输出				×	×	×			
pad_had_jdb_req_b	输入									
had_pad_jdb_ack_b	输出									
pad_had_jtg_tap_en	输入									
had_pad_jtg_tap_on	输出									
had_pad_jdb_pm[1:0]	输出									
pad_had_jtg_tclk	输入									
pad_had_jtg_trst_b	输入									
had_pad_jtg_tdo	输出		△	△						△
had_pad_jtg_tdo_en	输出		△	△						△
pad_had_jtg_tdi	输入		△	△						△
pad_had_jtg_tms	输入		△	△	×	×	×	×	×	△
pad_had_jtg_tms_i	输入	×	△	△						△
had_pad_jtg_tms_o	输出	×	△	△						△
had_pad_htg_tms_oe	输出	×	△	△						△
pad_hadjtag2_sel	输入	×	×	×						×

注：×表示 HAD 中没有该信号；△表示在某些配置下存在该信号，否则不存在

pad_biu_dbgrq_b

同步进入调试模式请求信号。低电平有效。该信号是 CPU 的外部输入信号，在 CPU 中被系统时钟同步之后输入到 HAD 中，HAD 使用该信号让 CPU 同步进入调试模式。拉低该信号与设置 HAD 寄存器 HCR 的 DR 位有相同的作用。

CK801，CK802，E902 中的 HAD 没有实现该信号。

pad_biu_ brkrq_b[1:0]

处理器断点调试请求信号。低电平有效。该信号是 CPU 的外部输入信号，在 CPU 中被系统时钟同步。如果该信号有效，并且 HAD 寄存器 CSR 中的 FDB 位为 1，则 CPU 进入调试模式，否则，CPU 将进入断点异常。设置 pad_biu_brkrq_b[0]为 0，将产生数据存取内存断点。设置 pad_had_brkrq_b[1:0]将产生指令取指内存断点。但是，如果该指令被取指之后但没有被执行，将不产生断点。

CK801，CK802，E902，CK807 和 CK810 中的 HAD 没有使用该信号。

biu_pad_dbg_b (sysio_pad_dbg_b)

低电平表示 CPU 处于调试模式。

pad_had_jdb_req_b 与 had_pad_jdb_ack_b

pad_had_jdb_req_b 信号为让 CPU 异步进入调试状态的请求信号，该信号至少要维持低电平两个 JTAG 时钟周期才能保证 CPU 进入调试状态并且能够调试程序。如果该信号维持低电平不足两个 JTAG 时钟周期，那么可能会出现 CPU 已进入调试状态但不能调试程序的情况，因为该信号还会用于使能 HAD 中的 TAP 状态机。

CK801，CK802，E902 中没有使用该信号，该信号指示作为 input 输入 HAD，内部没有实体逻辑。

在 CPU 进入调试状态之后，had_pad_jdb_ack_b 信号会维持两个 JTAG 时钟周期的低电平以作为应答。

pad_had_jtg_tap_en 与 had_pad_jtg_tap_on

pad_had_jtg_tap_en 信号为 HAD 中 TAP 状态机的使能信号，维持该信号为高电平至少一个 JTAG 时钟周期可以使能 HAD 中的 TAP 状态机。如果该信号在 CPU 上电之后一直无效，那么使用同步方式（如设置 HAD 寄存器 HCR 中的 DR 位，断点等）使 CPU 进入调试状态时可能无法调试程序。

在 TAP 状态机启动之后，had_pad_jtg_tap_on 信号将拉高。实际上，只要 pad_had_jtg_tap_en 信号有效，had_pad_jtg_tap_on 信号将一直有效。同时 had_pad_jtg_tap_on 信号还会在通过 pad_had_jdb_req_b 信号异步进入调试状态之后有效。

在 CK801 以及 CK802，E 902 中，pad_had_jdb_req_b 只作为输入信号，没有实际逻辑产生。

had_pad_jdb_pm[1:0]

had_pad_jdb_pm[1:0]信号指示 CPU 当前低功耗模式，可以通过该信号确定 CPU 是否已进入调试模式。具体如表 2.2 所示。

表 2-2 had_pad_jdb_pm 指示当前 CPU 状态

had_pad_jdb_pm[1:0]	说明
00	普通模式
01	低功耗模式（STOP，DOZE，WAIT）
10	调试模式
11	保留

pad_had_jtg_tclk

JTAG 时钟信号。该信号为外部输入信号，一般为调试器产生的时钟信号，要保证该时钟信号的频率至少要比 CPU 时钟信号的频率慢 2 倍才能保证 HAD 与 CPU 核之间的正常工作。

pad_had_jtg_trst_b

pad_had_jtg_trst_b 信号为 JTAG 复位信号，可以复位 TAP 状态机以及其他相关控制信号。

JTAG5 相关信号

pad_had_jtg_tdi 信号为 HAD 端 JTAG 串行输入信号，HAD 端在 JTAG 时钟信号 tclk 的上升沿对其采样，而外部调试器在 JTAG 时钟的下降沿设置该信号；

pad_had_jtg_tdo 信号为 HAD 端 JTAG 串行输出信号，HAD 端在 JTAG 时钟信号 tclk 的下降沿对其设置，而外部调试器在 JTAG 时钟的上升沿对其采样；

pad_had_jtg_tdo_en 信号表示 pad_had_jtg_tdo 信号有效，通常外部调试器监视该信号以确定 pad_had_jtg_tdo 信号是否有效，调试器也可以不观察该信号而通过调试器内部的 TAP 状态机的状态以决定对 pad_had_jtg_tdo 信号的采样。该信号主要用作在实现多个 TAP 状态机时，确定是哪个 JTAG 的输出；

pad_had_jtg_tms 信号为 JTAG 模式选择信号，该信号由调试器发出，用于控制 HAD 中

TAP 状态机的运作。

注意：CK5xx 和 CK6xx 系列中，JTAG_5 接口作为唯一的调试接口存在； CK801，CK802，CK803 和 E902 中，JTAG_5 调试接口作为一种可选的实现，在选在实现 JTAG_2 接口时，JTAG_5 相关信号将不存在；CK807 和 CK810 中，JTAG_2 接口和 JTAG_5 接口同时在 CPU 核中实现，这些信号将同时存在，只是 pad_had_jtg_tms 信号的功能被 JTAG_2 的 3 个信号替代。

JTAG2 相关信号

pad_had_jtg_tms_i 信号为 JTAG_2 接口串行数据输入信号，HAD 端在 JTAG 时钟信号 tclk 的上升沿对其采样，而外部调试器在 JTAG 时钟的下降沿设置该信号；

该信号在空闲时必须保持为高电平，同时空闲时 JTAG_2 接口的时钟信号最好停止。用户可以利用该信号同步复位 HAD 逻辑：在实现 JTAG_2 接口的 HAD 中，如果 JTAG_2 时钟信号一直有效，用户只需保持该信号为高电平状态并维持 80 个时钟周期即可同步复位 HAD。复位 HAD 后，HAD 的 TAP 状态机回到 RESET 态，同时 HAD 寄存器 HACR 复位到 0x82（指向 HAD 的 ID 寄存器）。

had_pad_jtg_tms_o 信号为 JTAG_2 接口串行数据输出信号，HAD 端在 JTAG 时钟信号 tclk 的下降沿对其设置，而外部调试器在 JTAG 时钟的上升沿对其采样；

had_pad_jtg_tms_oe 信号为 had_pad_jtg_tms_o 信号有效指示信号。CPU 外部应利用该信号将 pad_had_jtg_tms_i 和 had_pad_jtg_tms_o 信号合为一个双向端口信号。

pad_had_jtag2_sel

JTAG_2 接口选择信号。该信号用于在同时实现 JTAG_5 接口和 JTAG_2 接口的 CPU 中，选择两者之一用作调试接口。置高表示选择 JTAG_2 调试接口，置低表示选择 JTAG_5 调试接口。该信号目前在 CK803S，CK807 和 CK810 中的 HAD 有使用。

3 HAD 寄存器说明

在不同的 CSKY CPU 核中，HAD 寄存器的是否实现以及寄存器的宽度略有不同。CK5xx，CK6xx 与 CK8xx 之间的 HAD 的区别较大，在 CK5xx 和 CK6xx 的 HAD 中，存在 128 位的 CPUSCR 寄存器，而在 CK8xx 的 HAD 中，CPUSCR 被分解成 5 个独立的子寄存器。各系列 CPU 中的 HAD 寄存器列表及各寄存器的简单说明如表 3-1 和表 3-2 所示。

表 3-1 各系列 CPU 中的 HAD 寄存器列表

寄存器名称	CK5xx CK6xx	CK801	CK802	CK803S	CK804	CK805	CK807	CK810	E902
HACR	8 位	8 位	8 位	8 位	8 位	8 位	8 位	8 位	8 位
BSEL	×	32 位	32 位	×	×	×	×	×	32 位
ID	×	32 位	32 位	32 位	32 位	32 位	32 位	32 位	32 位
OTC	16 位	8 位	8 位	8 位	8 位	8 位	8 位	8 位	×
MBCA	16 位	8 位	8 位	8 位	8 位	8 位	8 位	8 位	×
BABA	32 位	32 位	32 位	32 位	32 位	32 位	32 位	32 位	32 位
BAMA	32 位	8 位	8 位	8 位	8 位	8 位	8 位	8 位	8 位
MBCB	16 位	[1]8 位	[1]8 位	8 位	8 位	8 位	8 位	8 位	×
BABB	32 位	[1]32 位	[1]32 位	32 位	32 位	32 位	32 位	32 位	[1]32 位
BAMB	32 位	[1]8 位	[1]8 位	8 位	8 位	8 位	8 位	8 位	[1]8 位

[2]BABx	×	[1]32 位	[1]32 位	[1]32 位	[1]32 位	[1]32 位	×	×	[1]32 位
[2]BAMx	×	[1]8 位	[1]8 位	[1]32 位	[1]32 位	[1]32 位	×	×	[1]8 位
PCFIFO	32 位	×	×	32 位	32 位	32 位	32 位	32 位	32 位
CPUSCR	128 位	×	×	×	×	×	×	×	×
[3]Bypass									
HCR	32 位	32 位	32 位	32 位	32 位	32 位	32 位	32 位	32 位
HSR	16 位	16 位	16 位	16 位	16 位	16 位	16 位	16 位	16 位
EHSR	32 位	×	×	×	×	×	×	×	×
WBBR	×	32 位	32 位	32 位	32 位	32 位	32 位	32 位	32 位
PSR	×	32 位	32 位	32 位	32 位	32 位	32 位	32 位	32 位
PC	×	32 位	32 位	32 位	32 位	32 位	32 位	32 位	32 位
IR	×	32 位	32 位	32 位	32 位	32 位	32 位	32 位	32 位
CSR	×	16 位	16 位	16 位	16 位	16 位	16 位	16 位	16 位
DADDR	32 位	32 位	32 位	32 位	32 位	32 位	32 位	32 位	×
DDATA	32 位	32 位	32 位	32 位	32 位	32 位	32 位	32 位	×
DDERx	×	×	[1]32 位	×	×	×	×	×	[1]32 位
EDCR	×	32 位	32 位	×	×	×	×	×	×
DPCR	×	32 位	32 位	×	×	×	×	×	×
[4]MBIR	×	×	32 位	×	×	×	×	×	×
[4]DACSR	×	32 位	32 位	×	×	×	×	×	32 位
[4]DATR	×	32 位	32 位	×	×	×	×	×	32 位
[4]DARWR	×	32 位	32 位	×	×	×	×	×	32 位

[1] 该类该寄存器是否实现是硬件可配置

[2] 该类寄存器在 CK801, CK802, CK803S, CK804, CK805, E902 中有 7 个，在其他核中没有

[3] Bypass 寄存器在所有处理器 HAD 中都存在，但该寄存器并没有实际意义，且宽度任意

[4] MBIR 只在 CK802 中存在；DACSR, DATR, DARWR 是 DDMA 功能的相关寄存器，只在 CK801，CK802，E902 中存在

表 3-2 各寄存器简要说明

寄存器名称	说明
HACR	HAD 命令寄存器（HAD Command Register）
BSEL	HAD 寄存器组选择寄存器（Bank Select Register）
OTC	跟踪调试计数寄存器（Trace Counter）
MBCA	内存硬断点计数寄存器 A（Memory Breakpoint Counter Register A)
BABA	内存硬断点基地址寄存器 A（Breakpoint Address Base Register A）
BAMA	内存硬断点地址掩码寄存器 A（Breakpoint Address Mask Register A）
MBCB	内存硬断点计数寄存器 B（Memory Breakpoint Counter Register B)
BABB	内存硬断点基地址寄存器 B（Breakpoint Address Base Register B）
BAMB	内存硬断点地址掩码寄存器 B（Breakpoint Address Mask Register B）
BABx	内存硬断点基地址寄存器 x（Breakpoint Address Base Register x）
BAMx	内存硬断点地址掩码寄存器 x（Breakpoint Address Mask Register x）
PCFIFO	程序地址指针缓存（Program Counter FIFO）
CPUSCR	CPU 扫描链寄存器（CPU Scan Chain Register）

Bypass	Bypass，空
HCR	HAD 控制寄存器（HAD Control Register）
HSR	HAD 状态寄存器（HAD Status Register）
EHSR	HAD 扩展状态寄存器（Extended HAD Status Register）
WBBR	回写总线寄存器（Write Back Bus Register）
PSR	处理器状态寄存器（Processor Status Regsiter）
PC	程序指针寄存器（Program Counter）
IR	指令寄存器（Instruction Register）
CSR	控制和状态寄存器（Control and Status Register）
DADDR	直接下载通道地址寄存器（Direct Download Channel Address Register）
DDATA	直接下载通道数据寄存器（Direct Download Channel Data Register）
DDERx	调试数据交换寄存器器（Debug Data Exchange Register）
EDCR	扩展调试控制寄存器（Extended Debug Control Register）
DPCR	调试性能分析计数器（Debug Profiling Counter Register）
MBIR	硬件断点触发指示寄存器（Memory Breakpoint Indicate Register）
DACSR	直接访问总线控制和状态寄存器（Direct Access Control and Status Register）
DATR	直接访问总线传输寄存器（Direct Access Trans Register）
DARWR	直接访问总线读写寄存器（Direct Access Read and Write Register）

3.1 HAD 命令寄存器

HACR 寄存器定义如图 3-1 所示。

7	6	5	4 ... 0
R/W	GO	EX	RS

图 3-1 HACR 寄存器定义

RS[4:0] – HAD 寄存器选择位

RS[4:0]指示当前的 HAD 寄存器读写操作是针对哪个 HAD 寄存器。具体如表 3-3 所示。请参考表 3-1，在各系列 CPU 中，对于没有实现的 HAD 寄存器，相应的 RS[4:0]值没有意义，读这些没有实现的寄存器时，读出的值是不确定的，写这些寄存器没有任何作用。

其中 RS[4:0]为 11111 时并不指向任何特定的 HAD 寄存器，而是指向上一次操作中的目标寄存器。在 E902 中，RS[4:0]的复位值为 00010（指向 ID 寄存器）。

表 3-3 HACR 中寄存器选择位

RS[4:0]	说明		
	Bank 0	Bank 1	Bank 2
00000	保留	BABC	EDCR
00001		BAMC	DPCR
00010	ID	BABD	
00011	OTC	BAMD	
00100	保留	BABE	
00101	保留	BAME	
00110	PCFIFO	BABF	

00111	BABA	BAMF	
01000	BABB	BABG	
01001	BAMA	BAMG	
01010	BAMB	BABH	
01011	CPUSCR	BAMH	
01100	Bypass（没有选择任何寄存器）	BABI	
01101	HCR	BAMI	
01110	HSR	保留	
01111	EHSR		
10000	保留		
10001	WBBR		
10010	PSR		
10011	PC		
10100	IR		
10101	CSR		
10110	DDERx		
10111	保留		
11000	DADDR	DACSR	
11001	DDATA	DATR	
11010	保留	DARWR	
11011	保留	MBIR	
11100	保留	MBEE	
11101	保留	MBSE	
11110	BSEL	BSEL	
11111	系统保留	保留	

EX – 指示 CPU 核离开调试模式

设置该位指示让 CPU 离开调试模式，设置了 EX 位必须同时设置 GO 位才有效。

在 CK8xx 系列 CPU 和 E902 中，设置 EX 位和 GO 位，同时 R/W 为 0（写操作），RS 选择的是 WBBR、PSR、PC、IR、CSR 或者 Bypass 寄存器，CPU 才会在当前 HAD 寄存器写操作之后离开调试模式，并跳转到 PC 寄存器的地址上开始执行程序。

0：停在调试模式

1：退出调试模式，跳到 PC 寄存器所指向的地址开始执行指令

GO – 调试模式下让 CPU 执行指令

该位用于指示 CPU 在调试模式下执行指令。指令执行完成之后再回到调试模式。

在 CK8xx 系列 CPU 和 E902 中，欲让 CPU 在调试模式下执行一条指令，需要将指令写入 IR 寄存器并设置 GO 位（HACR 中的 R/W = 0，RS = 10100）

0：停在调试模式

1：在调试模式下执行一条指令

R/W – Read/Write

该位表示当前的 JTAG 操作是读 HAD 寄存器还是写 HAD 寄存器。

0：写 HAD 寄存器

1：读 HAD 寄存器

3.2 HAD 控制寄存器

HAD 控制寄存器 HCR 定义如图 3-3 所示。

31 22	21	20	19 16	15	14	13	12 9	8 6	5 3	2 0
0	ADR	DDCEN	0	DR	0	TME	0	BCB	0	BCA

图 3-2 HCR 寄存器定义

BCA[4:0] / BCB[4:0] – 内存硬断点控制位

这些控制位控制断点的允许状态，限定对于断点的存或取的特性，读或写操作类型，用户级别，程序空间或数据空间等是否产生断点匹配信号。BCA[4:0]对应于内存硬断点 A 的设置，BCB[4:0]对应于内存硬断点 B 的设置。具体如表 3-4 所示。

在 E902，CK803，CK802 以及 CK801 中，硬断点 B 硬件是否实现是可配置的。

表 3-4 内存硬断点控制表

BCA[2:0] / BCB[2:0]	说明
000	禁止内存硬断点
001	任何指令取指和数据存取地址匹配都可以触发断点
010	指令的取指地址匹配可以触发断点
011	数据的存取地址匹配可以触发断点
100	转移指令的取指地址匹配可以触发断点
101	写入内存地址匹配可以触发断点
110	读取内存地址匹配可以触发断点
111	保留

TME – 程序追踪调试模式使能位

该位控制追踪调试模式的使能。

0：禁止追踪调试

1：允许追踪调试

DR – 同步调试请求位

设置该位之后，CPU 在非调试模式下执行完当前指令后进入调试模式。如果该位一直有效，那么即使设置 HACR 寄存器的 GO 和 EX 位让 CPU 退出调试模式，CPU 也会在退出调试模式执行一条指令之后马上又会返回调试模式。

0：没有请求同步进入调试模式

1：请求在当前指令退休之后进入调试模式

DDCEN – DDC 使能位

DDC 模式使能位。DDC（Direct Download Channel）模式允许用户快速下载 CPU 可执行文件到 CPU 程序存储器中。使用 DDC 模式下载程序相比以普通写存储器的方式下载程序更快，一般，DDC 模式下载速度有 1.5 MByte/s。使用 DDC 下载程序时首先要保证在调试模式下。DDCEN 位默认值为 0。

0：DDC 模式关闭

1：DDC 模式使能

ADR – 异步调试请求位

CPU 以异步方式进入调试模式使能位。设置该位之后，CPU 立即进入调试模式而不等待

当前指令执行完成。该位可用于在 CPU 执行程序僵死时使 CPU 强制进入调试模式。

0：没有请求异步进入调试模式

1：请求 CPU 立即强制进入调试模式

3.3 回写总线寄存器

WBBR 寄存器是 CPU 内部与调试器传送操作数的信息中介。如果想要读取内部寄存器或是存储器的值，那么就可以在调试模式下通过让 CPU 执行相应指令把值传送到 WBBR 寄存器中，然后调试器通过 JTAG 接口将该值读出。比如：让 CPU 执行一条 mov r0, r0 指令，那么 R0 的值就会保存在 WBBR 中。如果想将确定的值写入 CPU 内部寄存器或是存储器某个地址，那么就可以先将该值写入到 WBBR 寄存器中，然后让 CPU 执行相应指令，就能完成操作（此时需要 CSR 寄存器的 FFY 被设置，具体请参考 CSR 寄存器说明）。关于怎样在调试模式下让 CPU 执行确定的指令请参考 IR 寄存器说明。

3.4 指令寄存器

在进入调试模式后，用户可以输入任意有效的指令给指令寄存器，有效控制这些指令被执行。通过有目的的选择和执行指令，可以检测和改变内存和处理器内部寄存器的内容，这种调试方式是单步实现的。写 IR 寄存器时，HACR 寄存器中的 GO 必须为 1，IR 中的指令才会被执行。

退出调试模式时，由于 HACR 寄存器中 EX 位和 GO 位同时置为 1，此时 CPU 不会执行 IR 寄存器中的指令，而是跳到 PC 寄存器中的地址并退出调试模式。

在 E902 中实现的是 RV32EMC 指令集，该指令集为 16 位/32 位混编指令集，所以 E902 的 HAD 寄存器 IR 为 32 位的。并且在 E902 中，IR 寄存器是只写寄存器，读该寄存器将得到不确定的值。

3.5 控制和状态寄存器

CSR 寄存器保存了 CPU 进入调试模式时部分 CPU 的状态值，还有一些调试控制位。在进入调试模式后，用户可以改变 CSR 寄存器的值，无论是单步调试（不退出调试模式）还是退出调试模式（执行 GO+EX 命令）。在调试时，CSR 的几个有效位必须赋值正确。CSR 寄存器的定义如图 3-7 所示。

15 14	13 11	10	9	8	7	6 0
0	0	0	0	FFY	FDB	0

图 3-3 CSR 寄存器定义

FDB – 强制调试模式控制位（Force Debug Mode Bit）

该位控制处理器是否允许某种方式进入调试模式。一旦设置了该位，处理器就允许以某些方式进入调试模式，这和其他的同步调试请求信号是一样的效果。使用该位进入调试模式的方法包括 CPU 执行 EBREAK 指令进入调试模式等。

0：禁止某些方式进入调试模式，如禁止软件中 EBREAK 指令进入调试模式

1：允许某些方式进入调试模式，如允许软件中 EBREAK 指令进入调试模式

在 CK802 中，FDB 位不再控制 bkpt 是否进入调试。

FFY – Y 操作数前馈控制位（Feed Forword Y Operand Bit）

该位控制 WBBR 寄存器的值是否作为 WBBR 寄存器被更新后 CPU 执行的第一条指令的 Y 操作数。这样就可以比较容易的实现 CPU 内部寄存器的值的更新：初始化 WBBR 寄存器；设置 FFY 位为 1；执行一条对期望修改的寄存器的指令，比如要更新 R1 的值，那么就执行 mov R1, R1 指令；CPU 利用 WBBR 的值更新 R1 寄存器。

0：无操作

1：WBBR 作为 Y 操作数的值

4 调试接口与 HAD 寄存器读写

用户可以通过 HAD 的调试接口，即 JTAG_5 接口或者 JTAG_2 接口与 HAD 通信，以读写 HAD 寄存器。在 HAD 中，对于这两种接口都有一个对应的状态机控制其与外部逻辑的通信。

4.1 通过 JTAG_5 接口读写 HAD 寄存器

HAD 中的 JTAG_5 接口控制器使用了标准的 JTAG TAP 状态机，如图 4-1 所示。TAP 状态机在 tms 信号的控制下，每个 tclk 周期变一次。用户可以控制 tms 变换，从而控制 HAD 中的 TAP 状态机，以达到与 HAD 通信的目的。

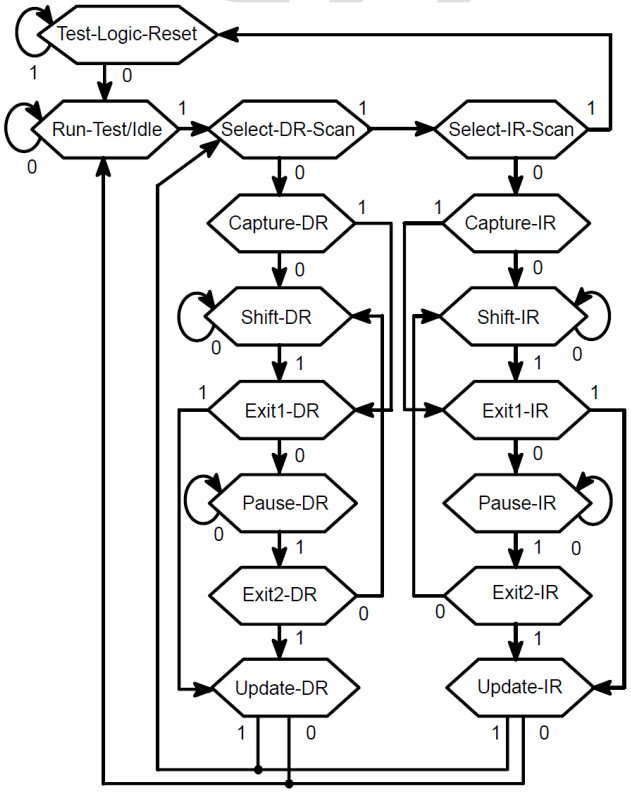

图 4-1 TAP 状态机

TAP 状态机控制着对串行输入信号 tdi 的采样和串行输出信号 tdo 的设置，状态机基本可以分为 IR 阶段和 DR 阶段，IR 阶段操作的是 HAD 中的 HACR 寄存器，而 DR 阶段操作的是 HAD 其他寄存器。具体的：

在“Shift-IR”状态中，HAD 在每个 TCK 的上升沿对 tdi 采样，循环 8 次得到一个 8 位的

数据，该数据在“Update_IR”状态时将写入到 HAD 寄存器 HACR 中，HACR 寄存器为 HAD 的命令寄存器（Command Register），HACR 的 RS 位指向接下来 DR 阶段操作的寄存器。

如果是读 HAD 寄存器，那么 HAD 在“Capture-DR”状态将相应的 HAD 寄存器的值加载到移位寄存器中，并在接下来“Shift-DR”状态中，在每个 TCK 的下降沿，从最低位开始，将数据从 TDO 移出；

如果是写 HAD 寄存器，类似于写 HACR 操作，在“Shift-DR”状态采样 TDI，在“Update-DR”状态将采样得到的值写入到相应的 HAD 寄存器。在通过 JTAG 接口串行移入和移出数据时，都是先移低位后移高位的。

通过 HAD 的 JTAG_5 接口写 HACR 寄存器，读写 HAD 其他寄存器的时序分别如图 4-2 和图 4-3 所示。

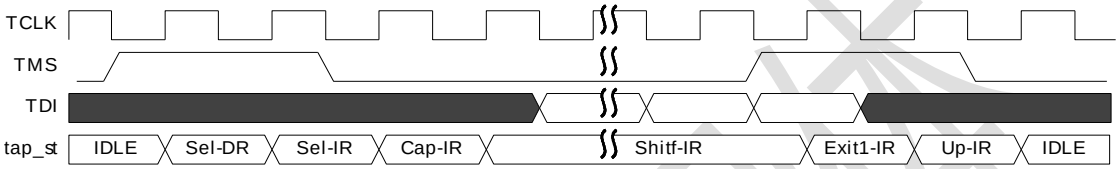

图 4-2 以 JTAG_5 接口写 HACR 寄存器时序

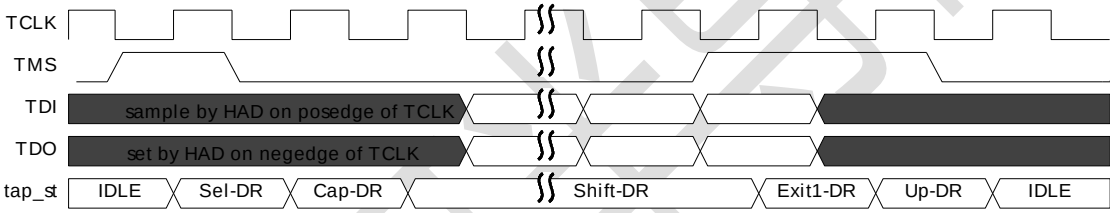

图 4-3 以 JTAG_5 接口读写 HAD 其它寄存器时序

4.2 通过 JTAG2 接口读写 HAD 寄存器

为节省 CPU 外部引脚，CSKY 定义了另一套调试接口，JTAG2 接口。JTAG2 调试接口包括 JTAG2 通信协议，JTAG2 接口控制器。JTAG2 接口定义为节省调试用引脚，支持多核调试，简化通信方式，能与 JTAG5 接口共存并能够被识别的一套调试接口。

JTAG2 接口中定义了两个接口信号，CLK 和 DAT，其中 DAT 为双向信号。如果是 JTAG2 和 JTAG5 调试接口共存于一个 CPU 核中，那么 JTAG2 的两个信号分别是复用 JTAG5 的 TCLK 和 TMS 信号。

JTAG2 通信协议中，数据总是在时钟的上升沿采样或者设置。数据是以数据包的形式传输的，一个数据包中包括：起始位（START），读写指示位（R/W），寄存器组选择位（RS[1:0]），数据信号方向反转位（Trn），数据同步位（SYNC），数据段（DATA），校验位（PAR）。读 HAD 寄存器和写 HAD 寄存器的数据包格式分别如图 4-4 和图 4-5 所示。

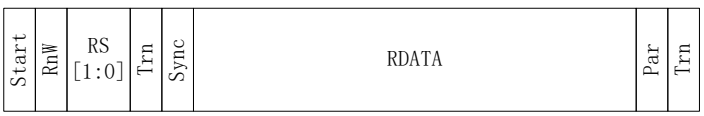

图 4-4 读 HAD 寄存器操作数据包格式

图 4-5 写 HAD 寄存器数据包格式

其中：

START 位为 DAT 信号线上一个周期的低电平；

R/W 为高表示读 HAD 寄存器操作，为低表示写 HAD 寄存器操作；

RS[1:0]表示寄存器组选择，

00：选择 CDIC（在 CPU 之外的调试控制器，为多核调试）命令寄存器

01：选择 CDIC 数据寄存器组

10：选择 HAD 命令寄存器 HACR

11：选择 HAD 数据寄存器组；

TRN 周期中为在读 HAD 寄存器操作中，预留给 HAD 一段时间用于将 DAT 数据信号的方向翻转，在该周期内，HAD 驱动数据信号为高，用户也应该驱动数据信号为高；

SYNC 位为一个周期的低电平，只在 HAD 寄存器读操作中存在，用户可根据该位判断何时开始读取 HAD 返回的数据；

DATA 数据段在读写 HAD 命令寄存器或者读写 CDIC 命令寄存器时固定为 8 位宽度，在读写 HAD 数据寄存器或者 CDIC 数据寄存器时固定为 32 位宽；

PAR 校验位为 DATA 数据段的奇校验；

在 HAD 中设计了一个状态机控制 HAD 通过 JTAG2 接口与外部逻辑通信的行为。状态机如图 4-6 所示。

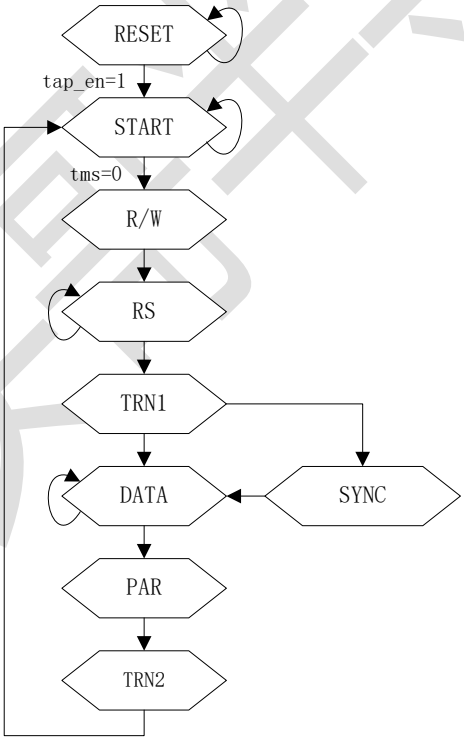

图 4-6 JTAG_2 接口通信控制状态机

通过 JTAG_2 接口读 HAD 寄存器和写 HAD 寄存器的时序分别如图 4-7 和图 4-8 所示。

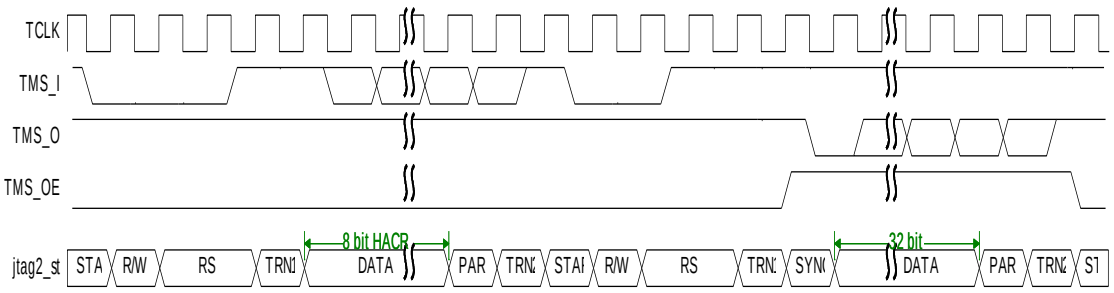

图 4-7 JTAG_2 接口读 HAD 寄存器时序

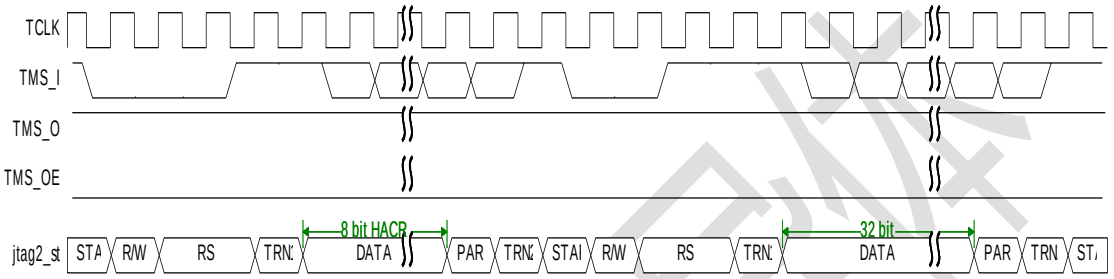

图 4-8 JTAG_2 接口写 HAD 寄存器时序

4.3 通过 TCIP 接口读写 HAD 寄存器

HAD 内部的寄存器，除了通过 JTAG 接口进行读写，也可以通过 CPU 直接进行读写，以提升调试效率。为了实现该功能，将 HAD 内部的处理器空间，映射到了 TCIP 接口上。

若处理器访问的 32 位地址空间的高 20 位为 0xE0011，则表示处理器对 HAD 内部寄存器的读写访问。

31 … 12	11 … 7	6 … 2	1	0
HAD IMG BASE	BANK SEL	INDEX		

图 4-9 HAD 寄存器 TCIP 映射地址说明

如上图，处理器地址 ADDR[21:12]用于判定是否命中 TCIP 中的 HAD 寄存器映射区域，若命中，则 ADDR[11:7]用于选择 HAD 寄存器的 BANK，类似 BSEL 寄存器的作用，ADDR[6:2]用于索引制定 BANK 中的具体寄存器，类似 HACR 中 RS[4:0]的作用。

因此，通过 TCIP 访问 HAD 寄存器，只需要一次 load 或者 store 操作即可，避免了通过 JTAG 操作时对 HACR，BSEL 等寄存器的多次读写操作。

目前在 CK801，CK802 上实现了上述 HAD 寄存器的 TCIP 映射功能。

4.4 调试接口的其他操作

4.4.1 通过 TRST 信号异步复位 HAD

在 5 线制的 JTAG5 接口中，TRST 信号可以异步复位 HAD，低电平有效，在 2 线制的 JTAG2 接口中，TRST 信号是可选的信号。拉低该信号可复位 HAD 逻辑。异步复位操作可以复位 HAD 中的通信控制状态机到复位状态，并会复位 HAD 的寄存器 HACR 为 0x82，即指向 HAD 的 ID

寄存器。

4.4.2 通过 JTAG2 接口同步复位 HAD

通过 JTAG2 接口可以同步复位 HAD 逻辑单元。参考第二章关于 TMS 信号的描述，用户拉高 TMS 信号并维持 TCLK 时钟信号至少 80 个周期，HAD 将会被同步复位，同步复位操作可以复位 HAD 中的 JTAG2 接口通信控制状态机到复位状态，并会复位 HAD 的寄存器 HACR 为 0x82，及复位后指向 HAD 的 ID 寄存器。

4.4.3 JTAG2 接口与 JTAG5 接口的识别

由于 JTAG2 接口复用了 JTAG5 接口的 TCLK 和 TMS 两个信号，用户在使用 HAD 之前需要首先知道目标 CPU 中 HAD 的调试接口类型，可通过以下方式识别：

1) 先认为目标 CPU 带有 JTAG2 调试接口；
2) 以 4.4.2 节中所描述的方式同步复位 HAD；
3) 以 JTAG2 接口中所定义的通信方式直接读取 HAD 的 DR 寄存器而不设置 IR 寄存器；
4) 用户判断读出的数据，如果是 0xFFFFFFFF，则认为目标 CPU 中实现的是 JTAG5 接口，否则为 JTAG2 接口，并且读出的数据就是 HAD 的 ID 寄存器的值；
5) 如果接口是 JTAG5 接口，用户需要在 TMS 信号线上发起一连串连续的 0 将 JTAG5 的状态机驱动到 IDLE 状态。

调试接口识别过程的时序如图 4-10 所示。从图中可以看出，对于 JTAG5 接口，接口识别时序对 JTAG5 的状态机没有影响，并且 JTAG5 对识别时序没有响应，而 JTAG2 接口却可以将其认为有效的命令，并将数据通过数据信号返回。

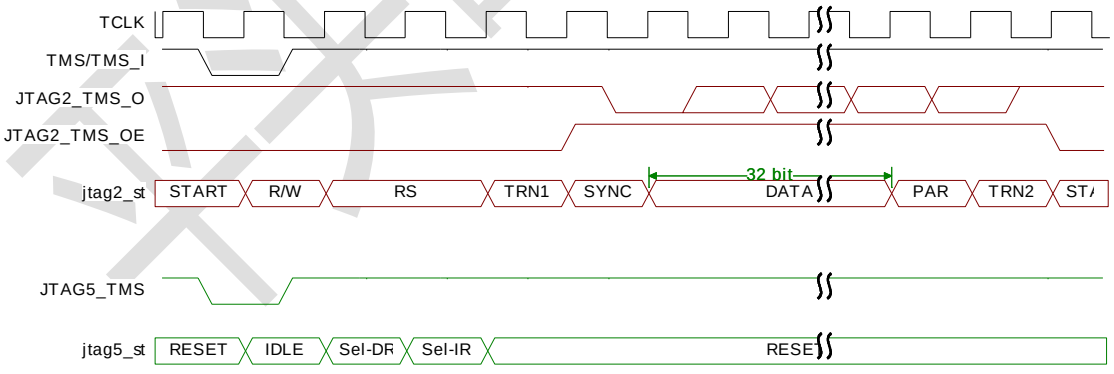

图 4-10 调试接口类型识别时序

5 通过 HAD 调试程序

设计 HAD 的目的就是为了方便调试。通过本章，用户可以了解怎样使 CPU 进入调试模式并通过 HAD 调试程序，以及通过 HAD 可以进行什么样的调试。

5.1 进入调试模式与检测 CPU 状态

进入调试模式的方法主要有以下 8 种。

1) 异步外部调试请求进入调试模式

HAD 的外部输入信号 pad_had_jdb_req_b 为调试请求信号，拉低该信号并维持至少两个 JTAG 时钟周期可使 CPU 异步进入调试模式并可通过 HAD 调试。

如果该信号在 CPU 复位信号有效时为低，并且的 CPU 复位信号拉起的时候仍然为低，那么 CPU 将不会执行任何指令而直接进入调试模式。这样进入调试模式后，HAD 中的 PC 和 PSR 寄存器的值将是不确定的，用户需要在让 CPU 退出调试模式之前设置好这些寄存器。

2) 同步外部调试请求进入调试模式

HAD 的外部输入信号 biu_had_sdb_req_b(pad_biu_dbgreq_b)被 CPU 时钟同步之后进入 HAD，如果设置了 HCR 寄存器中的 IDRE 位，则 HAD 请求 CPU 同步进入调试模式。

3) 设置 HCR 寄存器的 DR 位进入调试模式

可以通过调试器设置 HAD 的控制寄存器 HCR 中的 DR 位，以同步方式进入调试模式。

在不同 CPU 状态下设置 DR 位时，CPU 会有不同的行为进入调试模式：

1）在 CPU 复位时设置 DR 位

CPU 在执行完第一条指令之后进入调试模式

2）在正常指令执行状态下设置 DR 位

CPU 在执行完当前指令（当前指令退休）之后进入调试模式

3）在低功耗模式下设置 DR 位

此时 HAD 向 CPU 发起调试请求并同时请求退出低功耗模式，CPU 退出低功耗模式，执行完一条指令之后马上进入调试模式

4) 设置 HCR 寄存器的 ADR 位进入调试模式

可以通过调试器设置 HAD 的控制寄存器 HCR 中的 ADR 位，以异步方式进入调试模式。同第一种进入调试模式的方法一样，在进入调试模式后，HAD 中的 PC 和 PSR 寄存器的值将是不确定的，用户需要在让 CPU 退出调试模式之前设置好这些寄存器。

5) 软件调试请求进入调试模式

在正常执行中，如果 HAD 中 CRS 寄存器的 FDB 位置位，在软件中执行 bkpt 指令后，CPU 进入调试模式。

6) 外部内存断点请求进入调试模式

如果 CPU 外部信号 pad_biu_brkrq_b[1:0]有效时，且 HAD 寄存器 CSR 中的 FDB 位有效，CPU 产生内存断点进入调试模式。

7) 内存硬断点请求进入调试模式

内存硬断点模式开启后，当断点计数器递减到 0 时，处理器执行当前指令，如果该指令符合断点设置的要求，处理器进入调试模式。如果取指的地址触发断点，当这条已经取指的指令执行完成之后，CPU 进入调试模式；如果是 load/store 指令的存取地址触发断点，当该条指令执行完成之后 CPU 进入调试模式；当断点计数器为 0 时，如果当前指令触发了异常，CPU 将不会响应断点请求（除了指令取指地址匹配这种情况产生的调试请求），即使这条指令触发了断点。

8) 追踪调试请求进入调试模式

追踪开启后，若追踪计数器大于 0，CPU 每执行完一条指令，计数器递减 1。当追踪计数器递减到 0 时，在下一条指令执行完成之后，HAD 向 CPU 发出追踪调试请求，CPU 进入

调试模式，但如果这条指令触发了异常，CPU 将不会响应追踪调试请求。

注意：并不是所有的 CPU 都支持这 8 种进入调试模式的方法。具体见下表。

表 5-1 CPU 支持的调试请求方式

方式	CK5XX CK6XX	CK801	CK802	CK803S	CK804	CK805	CK807	CK810	E902
1（pad_had_jdb_req_b）		×	×						×
2（pad_biu_dbgreq_b）		×	×						×
3（DR 位）									
4（ADR 位）	×								
5（FDB + bkpt 指令）									
6（FDB + pad_biu_brkrq_b[1:0]）		×	×	×	×	×	×	×	×
7（内存断点请求）									
8（追踪调试请求）									×

在上述条件之一有效时，CPU 就会进入调试模式，用户需要检测 CPU 状态以确认当前 CPU 是否处于调试模式。检测 CPU 状态的操作可以通过读 HAD 状态寄存器 HSR 完成。HSR 寄存器中最低两位 PM[1:0]表示 CPU 状态。

5.2 读写 CPU 通用寄存器

以读 R1 寄存器为例：

1) 让 CPU 进入调试模式；
2) CK5xx 和 CK6xx 中需要保存 CPUSCR 寄存器，CK8xx/E902 中需要保存 CSR 寄存器；
3) 将 CSR 寄存器中的 FFY 位清零；
4) 向 IR 寄存器中写入“mov r1, r1”指令码，并同时设置 HACR 寄存器中的 GO 位；
5) 读 WBBR 寄存器，读出的值就是 CPU 寄存器 r1 中的内容；
6) 恢复 CK5xx 和 CK6xx 的 CPUSCR 寄存器中，恢复 CK8xx/E902 的 CSR 寄存器。

以写 R1 寄存器为例：

1) 让 CPU 进入调试模式；
2) CK5xx 和 CK6xx 中需要保存 CPUSCR 寄存器，CK8xx/E902 中需要保存 CSR 寄存器；
3) 将 CSR 寄存器中的 FFY 位置 1；
4) 将目标值写入 WBBR 寄存器；
5) 向 IR 寄存器中写入“mov r1, r1”指令码，并同时设置 HACR 寄存器中的 GO 位；
6) 恢复 CK5xx 和 CK6xx 的 CPUSCR 寄存器中，恢复 CK8xx/E902 的 CSR 寄存器。

对于 E902 而言，上述读写寄存器以 X1 为例，向 IR 寄存器写入的指令码为“mv x1, x1”。

5.3 读写 CPU 控制寄存器

5.3.1 通过 HAD 读 CPU 控制寄存器

前面已经实现了通过 HAD 读 CPU 通用寄存器，对于控制寄存器，由于没有控制寄存器更新控制寄存器的指令，所以要将控制寄存器的值转移到通用寄存器才可以通过读通用寄存器实现。不过，如果要通过通用寄存器间接实现，那么通用寄存器的内容就会被破坏，所以还要对使用到的通用寄存器做备份。用户一般需要按照以下步骤实现通过 HAD 读某个 CPU 控制寄存器，如 CR12：

1) 让 CPU 进入调试模式；
2) 通过 HAD 读取操作相关的 HAD 寄存器并在 HAD 外部做好备份。如 5.2 节中所述 ；
3) 通过 5.2 节所述的方法读出 CPU 通用寄存器 r0，并在 HAD 外部备份读出的 r0 寄存器的值；
4) 设置 HAD 寄存器 CSR 中的 FFY 位为 0，之后设置 HACR 寄存器中的 GO 位为 1，向 IR 寄存器中中写入“mfcr r0, cr12”指令码，不同的 CPU 中，该指令码不一定相同。CPU 执行这条指令，将用 cr12 寄存器的值更新通用寄存器 r0；
5) 通过 5.2 节所述的方法读出 CPU 通用寄存器 r0，因为 r0 已经在上一步被 cr12 的值更新，所以这里读出的 r0 的值就是 cr12 的值；
6) 再通过 5.2 节所述的写 CPU 通用寄存器的方法，用第 3）步保存的 r0 值写回到 CPU 通用寄存器 r0，达到恢复 r0 寄存器的目的；
7) 恢复 CPU 通用寄存器 r0 之后，再以第 2）步中备份的 HAD 寄存器写回到相应的 HAD 寄存器，恢复被修改过的 HAD 寄存器。

对于 E902 而言，上述操作中 cr12 寄存器可改为 mepc，作为临时变量保存的 r0 改为 x1，步骤 4 中向 IR 寄存器中写入的指令码改为“csrrs x1,mepc,x0”。

5.3.2 通过 HAD 写 CPU 控制寄存器

通过 HAD 对 CPU 控制寄存器的写操作也是需要借助 CPU 通用寄存器的。HAD 用户一般需要按照以下步骤实现通过 HAD 写某个 CPU 控制寄存器，如 CR17：

1) 让 CPU 进入调试模式；
2) 通过 HAD 读取操作相关的 HAD 寄存器并在 HAD 外部做好备份。如 5.2 节中所述；
3) 通过 5.2 节所述的方法读出 CPU 通用寄存器 r0，并在 HAD 外部备份读出的 r0 寄存器的值；
4) 通过 5.2 节的方法将希望写入控制寄存器 cr17 的值先写入 CPU 通用寄存器 r0；
5) 设置 HAD 寄存器 CSR 中的 FFY 位为 0，之后设置 HACR 寄存器中的 GO 位为 1，向 IR 寄存器中中写入“mtcr r0, cr17”指令码，不同的 CPU 中，该指令码不一定相同。

CPU 执行这条指令，将用通用寄存器 r0 的值更新控制寄存器 cr17；

6) 再通过 5.2 节所述的写 CPU 通用寄存器的方法，用第 3）步保存的 r0 值写回到 CPU 通用寄存器 r0，达到恢复 r0 寄存器的目的；

7) 恢复 CPU 通用寄存器 r0 之后，再以第 2）步中备份的 HAD 寄存器写回到响应的 HAD 寄存器，恢复被修改过的 HAD 寄存器.

对于 E902 而言，上述操作中 cr17 寄存器可改为 mie，作为临时变量保存的 r0 改为 x1，步骤 4 中向 IR 寄存器中写入的指令码改为“csrrw x0,mie,x1”。

5.4 通过 HAD 读写存储器

通过 HAD 读写存储器内容在调试中很有必要。读写存储器与读写 CPU 寄存器一样，都需要 CPU 执行指令的支持。同样的，存储器内容的读写也需要 HAD 寄存器 WBBR，CSR 和 IR 的支持。由于 HAD 操作过程中会改写这些寄存器，所以用户应该在操作之前备份这些寄存器，操作之后对其恢复。以下 4 节中主要说明如何通过 HAD 读写存储器内容的，最后一节说明了如何以 DDC 的方式快速下载程序到 CPU 程序存储器上。用户应该了解的是，存储器的读写和 CPU 寄存器的读写都是通过 HAD 寄存器的读写间接完成的。

5.4.1 通过 HAD 写入一个字的数据到存储器

如现在需要向存储器地址 Addr = 0xF0008000 中写入一个 32 位的字 Data = 0xAA55AA55，通常需要 CPU 执行 st 指令完成操作。用户一般需要按照一下步骤实现通过 HAD 写入一个字的数据到存储器中：

1) 让 CPU 进入调试模式；

2) 通过 HAD 读取操作相关的 HAD 寄存器并在 HAD 外部做好备份。如 5.2 节中所述；

3) 通过 5.2 节所述的方法读出 CPU 通用寄存器 r0，并在 HAD 外部备份读出的 r0 寄存器的值；

4) 通过 5.2 节所述的方法读出 CPU 通用寄存器 r1，并在 HAD 外部备份读出的 r1 寄存器的值；

5) 通过 5.2 节所述的方法将 Addr 的值 0xF0008000 写入到 CPU 通用寄存器 r0 中；

6) 通过 5.2 节所述的方法将 Data 的值 0xAA55AA55 写入到 CPU 通用寄存器 r1 中；

7) 设置 HAD 寄存器 CSR 中的 FFY 位为 0，之后设置 HACR 寄存器中的 GO 位为 1，向 IR 寄存器中中写入“st.w r1 (r0)”指令码，不同的 CPU 中，该指令码不一定相同。CPU 执行这条指令，将 r1 寄存器的值写入到寄存器 r0 所指向的存储器地址中；

8) 通过 5.2 节所述的写 CPU 通用寄存器的方法，用第 3）步保存的 r0 值写回到 CPU 通用寄存器 r0，恢复被改写的 r0 寄存器的到原来的值；

9) 通过 5.2 节所述的写 CPU 通用寄存器的方法，用第 4）步保存的 r1 值写回到 CPU 通用寄存器 r1，恢复被改写的 r1 寄存器的到原来的值；

10) 以第 2）步中备份的 HAD 寄存器写回到相应的 HAD 寄存器，恢复被修改过的 HAD 寄存器；

对于 E902 而言，上述操作中的 r0，r1 需要分别改为 x1, x2，步骤 7）中向 IR 寄存器写入的指令码需改为“sw x2,0x0(x1)”。

5.4.2 通过 HAD 将多个字一次写入连续的存储器地址上

同样的，写存储器操作需要 CPU 执行 st 指令，只是在向连续的地址上写多个数据时，还需要 CPU 在每写完一个数据之后修改一次写的存储器目标地址。一般 CPU 执行指令将存储地址的通用寄存器加上相应的值。如现在需要从存储器地址 Addr = 0xF0008000 开始连续写入 10 个字 Data[10]，用户一般需要按照以下步骤实现通过 HAD 将多个字一次写入到连续的存储器地址上：

1) 让 CPU 进入调试模式；

2) 通过 HAD 读取操作相关的 HAD 寄存器并在 HAD 外部做好备份。如 5.2 节中所述；

3) 通过 5.2 节所述的方法读出 CPU 通用寄存器 r0，并在 HAD 外部备份读出的 r0 寄存器的值；

4) 通过 5.2 节所述的方法读出 CPU 通用寄存器 r1，并在 HAD 外部备份读出的 r1 寄存器的值；

5) 通过 5.2 节所述的方法将 Addr 的值 0xF0008000 写入到 CPU 通用寄存器 r0 中；

6) 通过 5.2 节所述的方法将 Data[0]的值写入到 CPU 通用寄存器 r1 中；

7) 设置 HAD 寄存器 CSR 中的 FFY 位为 0，之后设置 HACR 寄存器中的 GO 位为 1，向 IR 寄存器中写入“st.w r1 (r0)”指令码，不同的 CPU 中，该指令码不一定相同。CPU 执行这条指令，将 r1 寄存器的值写入到寄存器 r0 所指向的存储器地址中；

8) 设置 HAD 寄存器 CSR 中的 FFY 位为 0，之后设置 HACR 寄存器中的 GO 位为 1，向 IR 寄存器中写入“addi r0, 4”指令码，不同的 CPU 中，该指令码不一定相同。CPU 执行这条指令，将 r0 寄存器中的值自加 4；

9) 重复指令步骤 6）到步骤 8）10 次，将 10 个字顺序写入到从地址 0xF0008000 开始的连续地址上；

10) 通过 5.2 节所述的写 CPU 通用寄存器的方法，用第 3）步保存的 r0 值写回到 CPU 通用寄存器 r0，恢复被改写的 r0 寄存器的到原来的值；

11) 通过 5.2 节所述的写 CPU 通用寄存器的方法，用第 4）步保存的 r1 值写回到 CPU 通用寄存器 r1，恢复被改写的 r1 寄存器的到原来的值；

12) 以第 2）步中备份的 HAD 寄存器写回到相应的 HAD 寄存器，恢复被修改过的 HAD 寄存器；

用户还需要注意的是，这个例子说明如何写多个 32 位的字，如果是写入多个 8 位的字节，那么在第 7）步骤中向 IR 寄存器写入的指令码应该是“st.b r1, (r0)”，而第 8）步骤中向 IR 寄存器写入的指令码就应该是“addi r0, 1”；如果是写入多个 16 位的半字，在第 7）步骤中向 IR 寄存器写入的指令码应该是“st.h r1, (r0)”，而第 8）步骤中向 IR 寄存器写入的指令

码就应该是“addi r0, 2”。

对于 E902 而言，上述操作中的 r0，r1 需要分别改为 x1, x2，步骤 7）中向 IR 寄存器写入的指令码需改为“sw x2,0x0(x1)”，步骤 8）中向 IR 寄存器写入的指令码需改为“addi x1,x1,4”。若需要多次写入多个字节的话，相应指令码为“sb x2,0x0(x1)”和“addi x1,x1,1”；若需要多次写入多个半字的话，相应的指令码为“sh x2,0x0(x1)”和“addi x1,x1,2”。

5.4.3 通过 HAD 从存储器中读取一个字节的数据

在写存储器时需要 CPU 执行 st 指令的支持，同样，在读存储器中读取数据时需要 CPU 指令 ld 指令的支持。如需要从存储器地址 0xF0008000 中读取一个字节，那么用户一般需要按照以下步骤实现：

1) 让 CPU 进入调试模式；

2) 通过 HAD 读取操作相关的 HAD 寄存器并在 HAD 外部做好备份。如 5.2 节中所述；

3) 通过 5.2 节所述的方法读出 CPU 通用寄存器 r0，并在 HAD 外部备份读出的 r0 寄存器的值；

4) 通过 5.2 节所述的方法读出 CPU 通用寄存器 r1，并在 HAD 外部备份读出的 r1 寄存器的值；

5) 通过 5.2 节所述的方法将 Addr 的值 0xF0008000 写入到 CPU 通用寄存器 r0 中；

6) 设置 HAD 寄存器 CSR 中的 FFY 位为 0，之后设置 HACR 寄存器中的 GO 位为 1，向 IR 寄存器中中写入“ld.b r1 (r0)”指令码，不同的 CPU 中，该指令码不一定相同。CPU 执行这条指令，将 r0 寄存器所指向的存储器地址中的数据读取出来并存入通用寄存器 r1 中；

7) 通过 5.2 节所述的方法读出 CPU 通用寄存器 r1，此时读出的值就是存储器地址 0xF0008000 中所存储的数据；

8) 通过 5.2 节所述的写 CPU 通用寄存器的方法，用第 3）步保存的 r0 值写回到 CPU 通用寄存器 r0，恢复被改写的 r0 寄存器的到原来的值；

9) 通过 5.2 节所述的写 CPU 通用寄存器的方法，用第 4）步保存的 r1 值写回到 CPU 通用寄存器 r1，恢复被改写的 r1 寄存器的到原来的值；

10) 以第 2）步中备份的 HAD 寄存器写回到相应的 HAD 寄存器，恢复被修改过的 HAD 寄存器；

对于 E902 而言，上述操作中的 r0，r1 需要分别改为 x1, x2，步骤 7）中向 IR 寄存器写入的指令码需改为“lb x2,0x0(x1)”。

5.4.4 通过 HAD 从连续的存储器地址中一次读取多个字节

通过 HAD 连续的从存储器中读取数据与连续的写入数据有类似的情况。如现在需要从

存储器地址 0xF0008000 出连续的读取 10 个字节，用户一般需要按照以下步骤实现：

1) 让 CPU 进入调试模式；

2) 通过 HAD 读取操作相关的 HAD 寄存器并在 HAD 外部做好备份。如 5.2 节中所述；

3) 通过 5.2 节所述的方法读出 CPU 通用寄存器 r0，并在 HAD 外部备份读出的 r0 寄存器的值；

4) 通过 5.2 节所述的方法读出 CPU 通用寄存器 r1，并在 HAD 外部备份读出的 r1 寄存器的值；

5) 通过 5.2 节所述的方法将 Addr 的值 0xF0008000 写入到 CPU 通用寄存器 r0 中；

6) 设置 HAD 寄存器 CSR 中的 FFY 位为 0，之后设置 HACR 寄存器中的 GO 位为 1，向 IR 寄存器中写入“ld.b r1 (r0)”指令码，不同的 CPU 中，该指令码不一定相同。CPU 执行这条指令，将 r0 寄存器所指向的存储器地址中的数据读取出来并存入通用寄存器 r1 中；

7) 通过 5.2 节所述的方法读出 CPU 通用寄存器 r1，此时读出的值就是存储器地址 0xF0008000 中所存储的数据；

8) 设置 HAD 寄存器 CSR 中的 FFY 位为 0，之后设置 HACR 寄存器中的 GO 位为 1，向 IR 寄存器中写入“addi r0, 1”指令码，不同的 CPU 中，该指令码不一定相同。CPU 执行这条指令，将 r0 寄存器中的值自加 1；

9) 重复指令步骤 6）到步骤 8）10 次，从存储器地址 0xF0008000 开始连续读取需要的数据；

10) 通过 5.2 节所述的写 CPU 通用寄存器的方法，用第 3）步保存的 r0 值写回到 CPU 通用寄存器 r0，恢复被改写的 r0 寄存器的到原来的值；

11) 通过 5.2 节所述的写 CPU 通用寄存器的方法，用第 4）步保存的 r1 值写回到 CPU 通用寄存器 r1，恢复被改写的 r1 寄存器的到原来的值；

12) 以第 2）步中备份的 HAD 寄存器写回到相应的 HAD 寄存器，恢复被修改过的 HAD 寄存器；

用户还需要注意的是，当按照 32 位的字或者 16 位的半字方式连续读存储的时候，第 7）与第 8）步骤中的指令码是不同的，这在 5.4.2 节有过说明。

对于 E902 而言，上述操作中的 r0，r1 需要分别改为 x1, x2，步骤 6）中向 IR 寄存器写入的指令码需改为“lb x2,0x0(x1)”，步骤 8）中向 IR 寄存器写入的指令码需改为“addi x1,x1,1”。

5.4.5 通过 HAD 以 DDC 方式快速下载程序

HAD 中与 DDC 相关的寄存器只有 DDC 地址寄存器 DADDR 和 DDC 数据寄存器 DDATA。在实现思路上，通过 DDC 方式写存储器与 5.4.2 节所述的基本是一样的，只是两者的实现步骤不一样。DDC 方式中，大部分复杂的工作都由 HAD 中 DDC 逻辑自动完成而不需要 HAD 用户关注，如：地址的递增，向 IR 寄存器写 st 指令码等。用户只需要提供一个基地址，并在之后连续向 HAD 中提供数据就可以完成向存储器连续写数据的操作，当然，用户还应该注

意在认为有需要的时候保存和恢复相关寄存器。

如需要以 DDC 方式快速向 CPU 程序存储器中写入编译好的二进制文件，用户一般需要按照以下步骤实现：

1) 使 CPU 进入调试模式；
2) 设置 HAD 控制寄存器 HCR 中的第 20 位 DDCEN 为 1，使能 DDC 功能；
3) 将程序存储器的基地址写入到 HAD 寄存器 DADDR 中；
4) 将二进制文件以 32 位字为单位写入到 HAD 寄存器 DDATA 中；
5) 连续重复第 4）步，直到二进制文件写完；

注意，DDC 方式也可以用在其他需要大批量写存储器的地方，但是 DDC 方式只能是以 32 位字为单位连续写入的。

对于 E902 而言，上述 DADDR 和 DDATA 寄存器实体均映射到 WBBR 寄存器，因此仍可实现 DDC 快速下载程序。