OTP JTAG Interface Design Specification

Version 0.1

Dec. 2017

**Revision History**

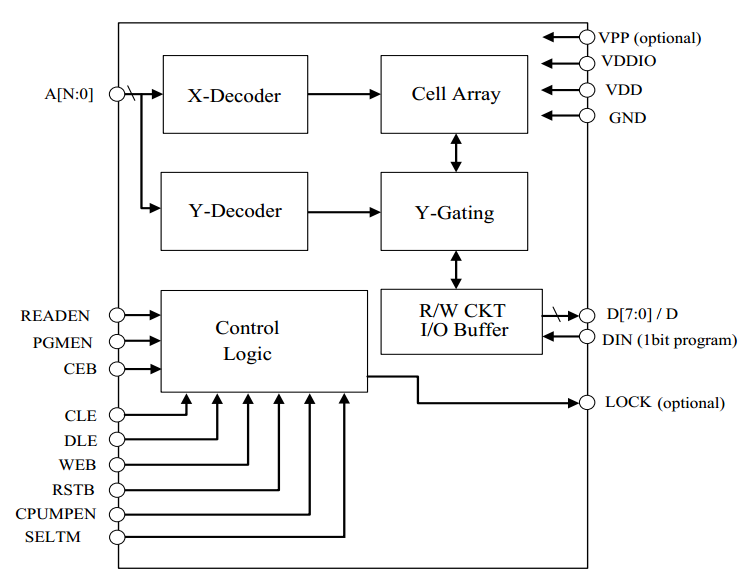
|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version | Author | Comments |
| 2017/12/26 | 0.1 | Hao Sun | Initial Version |
|  |  |  |  |

# Chapter 1. Product Overview

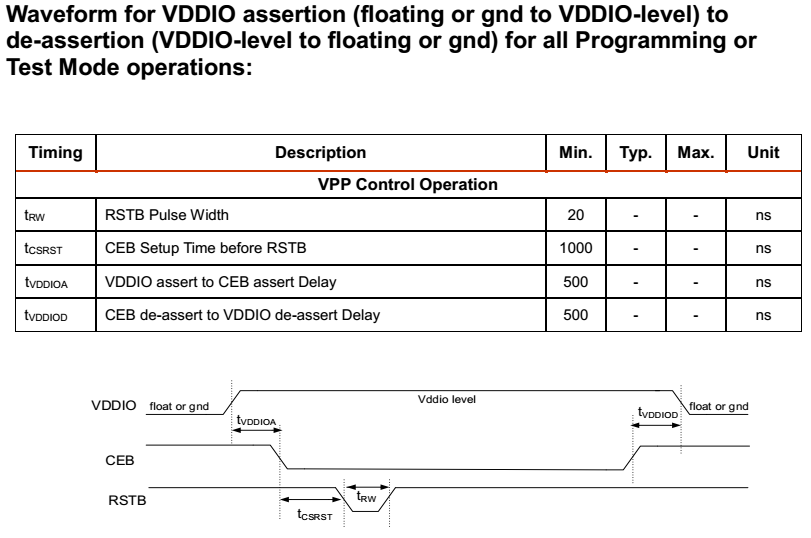
## Feature Support

1. Password reg: 每次硬件reset之后需要先通过IR选择password reg，并且输入对应的序列才行，序列匹配之后才能进行其他操作。每次复位只能输入一次password。用otp一个bit来选择是否用
2. 支持写OTP，读OTP，锁定OTP只能读不能写。不支持其他的TEST模式。
3. 需要有对应的来自OTP的disable控制bit，可以封掉JTAG。
4. 由于OTP没有烧写的时候全是0，如果再次烧入0的话时间的消耗是一样的，因此可以略过0只烧写1，这样可以节省时间。
5. 设置从0~3 四个command寄存器，并配有对应的状态寄存器.
6. OTP driver 每次只完成一笔操作，如果想利用多个command寄存器进行连续的写/读操作，则需要上层的控制状态机及时给出restart信号的对应的地址以及data。

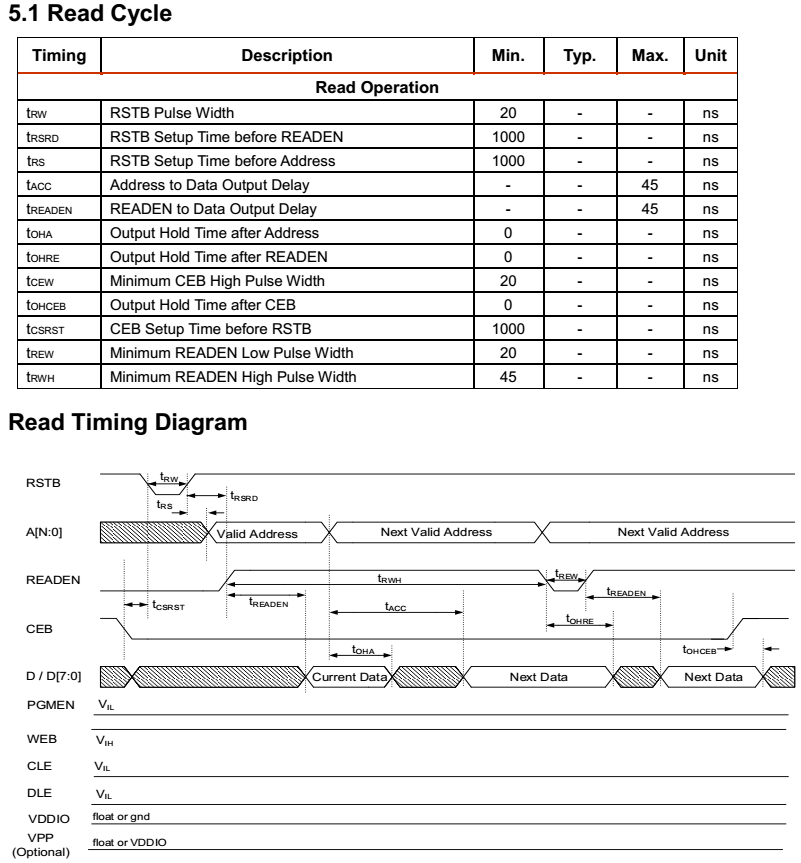
## OTP Diagram



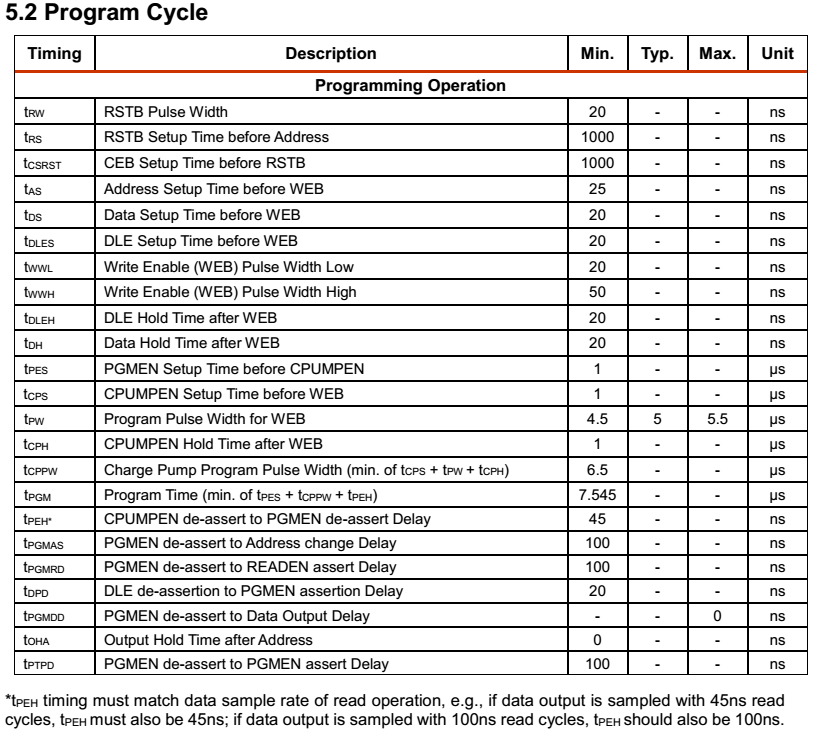
### 1.2.1. Initialization

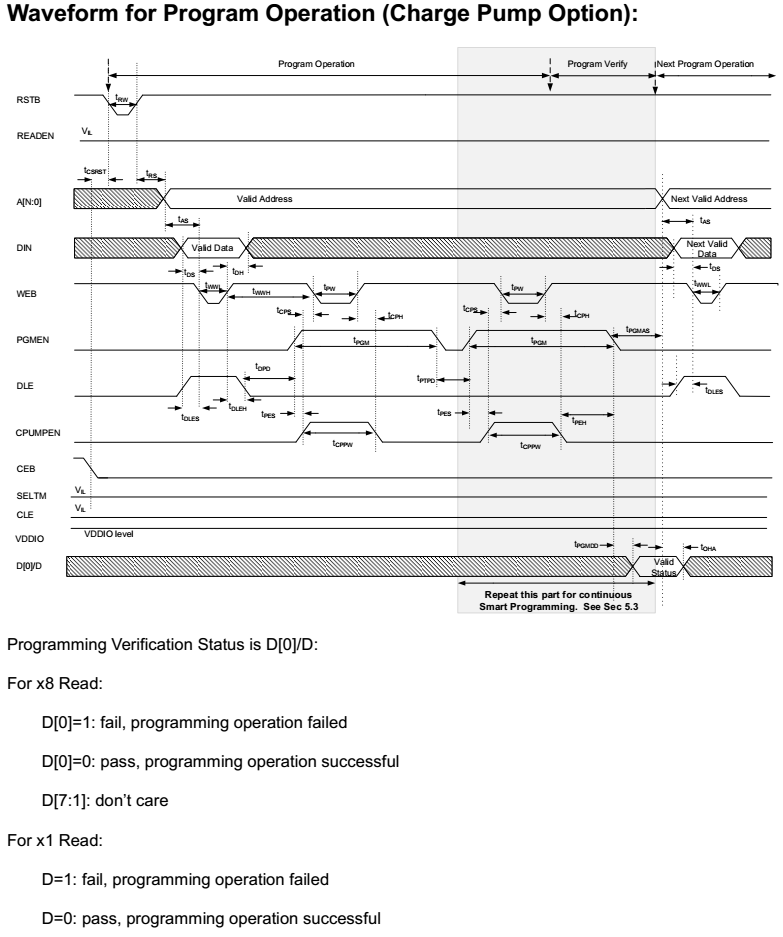


### 1.2.2. Read Operation



### 1.2.3. Program Operation





### 1.2.4. PGMLOCK Operation

## OTP Controller

### The otp controller diagram in Sirius:



### Planed otp controller diagram with jtag port:



## JTAG OTP Controller



### OTP Driver

#### OTP driver state machine:

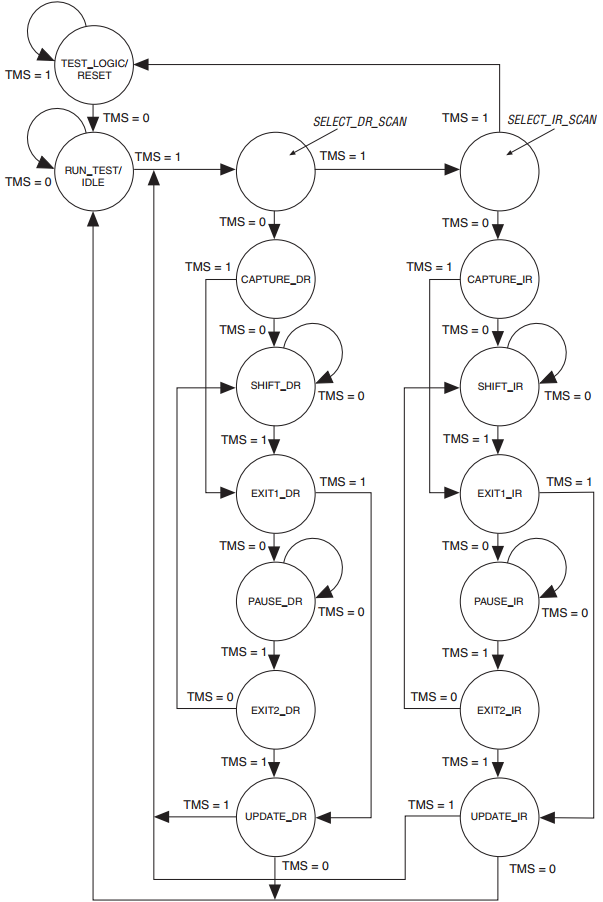


#### otp\_driver pins

|  |  |  |  |
| --- | --- | --- | --- |
| name | IO | width | description |
| i\_clk | I | 1 | ref clock |
| i\_reset\_n | I | 1 |  |
| i\_start | I | 1 |  |
| i\_restart | I | 1 |  |
| i\_hold | I | 1 |  |
| i\_read\_op | I | 1 |  |
| i\_write\_op | I | 1 |  |
| i\_lock\_op | I | 1 |  |
| i\_addr | I | 17 |  |
| i\_din | I | 1 |  |
| o\_dout | O | 8 |  |
| o\_w\_pass | O | 1 |  |
| o\_w\_fail | O | 1 |  |
| o\_locked | O | 1 |  |
| T\_CSRST | I | 8 | read/write/lock |
| T\_RW | I | 5 | read/write/lock |
| T\_LBV | I | 7 | read/write/lock |
| T\_RS\_LBV | I | 7 | read/write/lock |
| T\_READEN | I | 6 | read |
| T\_AS | I | 5 | write |
| T\_WWL | I | 5 | write/lock |
| T\_DLEH | I | 5 | write/lock(T\_CLEH) |
| T\_DPD | I | 5 | write/lock(T\_CPD) |
| T\_PES | I | 8 | write/lock |
| T\_CPS | I | 8 | write/lock |
| T\_PW | I | 9 | write/lock |
| T\_CPH | I | 8 | write/lock |
| T\_PEH | I | 5 | write/lock |
| T\_PTPD | I | 6 | write |
| T\_PGMAS | I | 6 | write |
|  |  |  |  |
| T\_TWS | I | 5 | lock |
| T\_PGML | I | 6 | lock |
| T\_PGRST | I | 8 | lock |
|  |  |  |  |
| otp\_D | I | 8 |  |
| otp\_LOCK | I | 1 |  |
| otp\_RST\_B | O | 1 |  |
| otp\_A | O | 17 |  |
| otp\_READEN | O | 1 |  |
| otp\_CEB | O | 1 |  |
| otp\_DIN | O | 1 |  |
| otp\_WEB | O | 1 |  |
| otp\_PGMEN | O | 1 |  |
| otp\_DLE | O | 1 |  |
| otp\_CPUMPEN | O | 1 |  |
| otp\_SELTM | O | 1 |  |
| otp\_CLE | O | 1 |  |
| otp\_VDDIO | O | 1 |  |

### TAP Controller

#### tap state machine



#### Password control state machine

password输入控制状态：

1. idle：reset之后进入本状态。如果i\_jtag\_disable为1，则直接跳入fail状态；如果 i\_password\_enable为0，则直接跳入pass状态；否则的话，如果IR寄存器中不是PASSWORD模式，则保持在此状态，且无法更新DR寄存器；如果IR寄存器中是PASSWORD模式，那么跳入到password状态。
2. password：这时如果IR寄存器中不是PASSWORD模式，那么直接跳入fail状态；如果password DR寄存器中更新的值是正确的密码，跳入pass模式；如果password DR寄存器是0，则保持在本状态；其他情况全都跳入fail。这时不论shift in什么数据，同时shift out状态。
3. pass：密码验证通过或者otp的控制bit没有要求输入密码，可以使用PASSWORD模式之外的所有模式，始终保持在本状态。如果IR进入PASSWORD模式，则shift in的数据不更新进password DR, shift out的数据表示验证通过或是不需要输入密码的状态。
4. fail：密码验证失败或者jtag接口被otp的控制bit封掉了，不能使用任何模式，始终保持在本状态。IR保持在PASSWORD模式不能更新，shift in的数据不更新进password DR，shift out的数据表示验证失败或是jtag接口被封掉的状态。

|  |  |
| --- | --- |
| Bit range | Description |
| [0] | need input password |
| [1] | verify pass or don’t need password |
| [2] | verify fail or jtag port is disable |
| [61:3] | 59’h0 |
| [62] | otp control bit i\_password\_enable |
| [63] | otp control bit i\_jtag\_disable |

#### otp\_tap\_ctrl pins

|  |  |  |  |
| --- | --- | --- | --- |
| name | IO | width | description |
| i\_reset\_n\_tck | I | 1 | reset synced with tck |
| i\_reset\_n\_tckn | I | 1 | reset synced with tckn |
| i\_tck | I | 1 | jtag port tck |
| i\_tckn | I | 1 | invert clock from tck |
| i\_tms | I | 1 | jtag port tms |
| i\_tdi | I | 1 | jtag port tdi |
| o\_tdo | O | 1 | jtag port tdo |
| i\_jtag\_disable | I | 1 | otp control bit, if this bit is 1, then the jtag port can’t be access |
| i\_password\_enable | I | 1 | otp control bit, if this bit is 1 and i\_jtag\_disable is 0, then password is need to shift in through the jtag port |
|  |  |  |  |
| o\_cmdX | O | 2 |  |
| o\_waddrX | O | 17 |  |
| o\_wdataX | O | 32 |  |
| o\_raddrX | O | 17 |  |
| o\_lockX | O | 1 | if want to lock the otp, lock0 and lock2 should be 1, lock1 and lock3 should be 0 |
| o\_processedX | O | 1 | status0.processed |
| i\_rdataX | I | 32 |  |
| i\_wlockX | I | 1 | status0.wlock write lock |
| i\_werrorX | I | 1 | status0.error some bit program fail, detail in error0\_all |
| i\_busyX | I | 1 | status0.busy ctrl FSM is working |
| i\_errorX\_all | I | 32 | detail error of all 32 bit |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

### Register Define

#### JTAG IR Mode

IR width = 5,

|  |  |
| --- | --- |
| MODE | VALUE |
| CMD | 5’b00001 |
| LOCK | 5’b00010 |
| OTP\_BYPASS | 5’b00011 |
| CONFIG | 5’b00100 |
| PASSWORD | 5’b01110 |
| BYPASS | 5’b11111 |

#### JTAG DR Format

DR width = 64

|  |  |
| --- | --- |
| MODE | Used Data Width |
| CMD/LOCK | [35:0] |
| OTP\_BYPASS | [26:0] |
| CONFIG | [33:0] |
| PASSWORD | [63:0] |

CMD/LOCK DR Format

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | CMD | ID | DATA | |  |
|  | [35:34] | [33:32] | [31:0] | |  |
|  | WDATA | id | wdata[31:0] | |  |
|  |  |  |  |  |  |
| shift in | WRITE | id | 0 | waddr[16:0] | CMD |
| shift in | WRITE | id | 32'ha5dead5a | | LOCK |
| shift out |  |  |  |  |  |
|  |  |  |  |  |  |
| shift in | RDATA | id | 0 | type[1:0] | |
| shift out | type | id | rdata[31:0] | |  |
|  |  |  |  |  |  |
|  | READ | id | 0 | raddr[16:0] | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| statusX | [7:0] | | | | | | |
| bit range | [7] | [6:5] | [4] | [3] | [2] | [1] | [0] |
| description | prcessedX | cmdX[1:0] | lockX | 1'b0 | busyX | wlockX | werrorX |

OTP\_BYPASS DR Format

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | [26:10] | [9] | [8] | [7] | [6] | [5] | [4] | [3] | [2] | [1] | [0] |
| shift in | A | CEB | CLE | CPUMPEN | DIN | DLE | PGMEN | READEN | RSTB | SELTM | WEB |
|  | [26:9] | | [8] | [7:0] | | | | | | | |
| shift out | 18'h0 | | LOCK | D | | | | | | | |

CONFIG DR Format

|  |  |
| --- | --- |
| [33:32] | [31:0] |
| WDATA | wdata |
| WRITE | waddr |
| RDATA | rdata |
| READ | raddr |

PASSWORD DR Format

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | [63:0] | | | | | |
| shift in | 64'hdeadbeefaa55cc33 | | | | | |
|  | [63] | [62] | [61:3] | [2] | [1] | [0] |
| shift out | jtag\_disable | passwd\_enable | 59'h0 | fail | pass | password |

#### CMD/CONFIG mode command

|  |  |  |
| --- | --- | --- |
| Command | Value | Description |
| WDATA | 2’b00 | shift in write data to WDATAX. |
| WRITE | 2’b01 | shift in write start addr, trigger program otp operate |
| RDATA | 2’b10 | shift out from RDATA\_id  type: 0->otp data; 1->command regs status; 2->program error detail. |
| READ | 2’b11 | trigger read otp operate, and fetch otp data to RDATAX  raddr: byte aligned, little endian. |

对OTP进行写操作的时候

## Operation flow

1. 切换到ATE模式；
2. reset后，访问控制状态机在IDLE状态，可以将IR切换到PASSWORD mode；然后访问控制状态机跳入PASSWORD状态，从低位开始shift in密码（deadbeefaa55cc33）到DR;
3. 如果密码输入正确的话，配置成功，并且无法再次输入密码，也不能再次切换成PASSWORD mode。如果密码输入错误的话，配置失败，并且无法再次输入密码，IR保持在PASSWORD mode。IR在PASSWORD模式下，往DR shift in 64’h0000000000000000则不认为有输入密码，这时tdo出来的数据为配置状态。
4. 验证成功后，切换到CONFIG模式，选通JTAG的otp\_driver，并根据需要对otp programmer/reader的参数进行配置。

Program otp:

1. 切换到CMD模式。
2. 读状态寄存器，得到四个命令寄存器的状态以及各命令寄存器上次操作的结果， CMDX的busy表示当前CMD正在处理中，这时processed一定是0，CMD处理完之后，processed会被置为1。当CMDX的processed为1时，并且没有error的时候，可以再次通过JTAG对CMDX进行配置。
3. 通过WDATA把数据写入寄存器，然后通过WRITE把要写otp的起始地址写入寄存器，并且自动将processed置为0，等待otp控制器处理当前CMD。
4. 状态机每次执行命令都会按照CMD0 -> CMD1 -> CMD2 -> CMD3 -> CMD0的顺序，直到所有的processed都为1。

Read otp:

1. 切换到CMD模式。
2. 读状态寄存器，得到四个命令寄存器的状态以及各命令寄存器上次操作的结果， CMDX的busy表示当前CMD正在处理中，这时processed一定是0，CMD处理完之后，processed会被置为1。当CMDX的processed为1时，并且没有error的时候，可以再次通过JTAG对CMDX进行配置。
3. 通过READ把RADDR写入寄存器，并且自动将processed置为0，等待otp控制器处理当前CMD。每条命令会从otp读出32bit
4. 状态机每次执行命令都会按照CMD0 -> CMD1 -> CMD2 -> CMD3 -> CMD0的顺序，直到所有的processed都为1。

Read data/status/error:

1. 切换到CMD模式。
2. 每次shift in命令时，tdo都会把上次READ的结果按照小端的顺序shift out出来：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| shift in | RDATA | id | 30’bxxxx | type[1:0] |
| shift out | type | id | rdata[31:0] | |

下面的是连续读取data/status/error的例子：

|  |  |  |
| --- | --- | --- |
|  | shift in | shift out |
| tap1 | {RDATA,2'h0,30'hxx,2'h0} |  |
| tap2 | {RDATA,2'h1,30'hxx,2'h0} | {2'h0,2'h0,rdata0[31:0]} |
| tap3 | {RDATA,2'h2,30'hxx,2'h0} | {2'h0,2'h1,rdata1[31:0]} |
| tap4 | {RDATA,2'h3,30'hxx,2'h0} | {2'h0,2'h2,rdata2[31:0]} |
| tap5 | {RDATA,2'hx,30'hxx,2'h1} | {2'h0,2'h3,rdata3[31:0]} |
| tap6 | {RDATA,2'h0,30'hxx,2'h2} | {2'h1,2'h3,status3[7:0],status2[7:0],status1[7:0],status0[7:0]} |
| tap7 | {RDATA,2'h1,30'hxx,2'h2} | {2'h0,2'h0,error0[31:0]} |
| tap8 | {RDATA,2'h2,30'hxx,2'h2} | {2'h0,2'h1,error1[31:0]} |
| tap9 | {RDATA,2'h3,30'hxx,2'h2} | {2'h0,2'h2,error2[31:0]} |
| tap10 | {RDATA,2'hx,30'hxx,2'hx} | {2'h0,2'h3,error3[31:0]} |

Lock otp:

1. 切换到LOCK模式。如果所有CMDX.processed都是1，进行以下操作。
2. 写入{WRITE,2’b10,32’ha5dead5a}。
3. 写入{WRITE,2’b00,32’ha5dead5a}启动控制器。