

ARM DAP

ARM DAP分为DP (debug port)和AP (access port)

DP一般和串口DP或JTAG DP

AP一般有AHB-AP, AXI-AP, APB-AP, JTAG-AP, mem-ap等

一般来讲AXI-AP, AHB-AP, APB-AP是执行MEM-AP结构，访问对应总线的memory system.

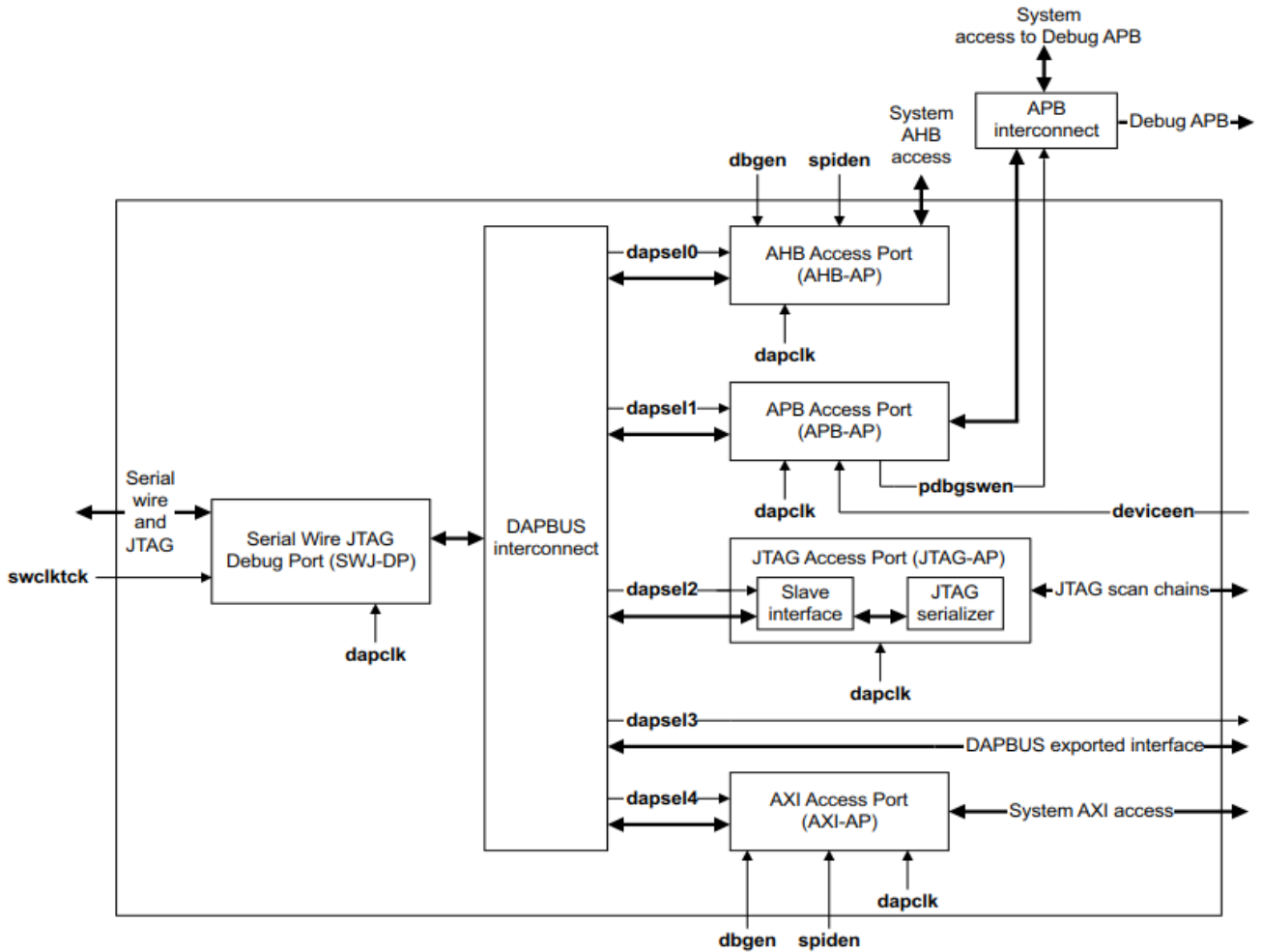


Figure 4-1 Structure of the CoreSight SoC-400 DAP components

1. JTAG DP IR command

Table 3-209 JTAG-DP register summary

4-bit IR instruction value ^a	8-bit IR instruction value ^b	JTAG-DP register	DR scan width	Description
0b1000	0b11111000	ABORT	35	JTAG-DP Abort Register, ABORT.
0b1010	0b11111010	DPACC	35	JTAG DP/AP Access Registers, DPACC/APACC.
0b1011	0b11111011	APACC	35	
0b1110	0b11111110	IDCODE	32	JTAG Device ID Code Register, IDCODE.
0b1111	0b11111111	BYPASS	1	JTAG Bypass Register, BYPASS.

- a. cxdapswjdp implemented with parameter IRLENS=0.
- b. cxdapswjdp implemented with parameter IRLENS=1.

2. DPACC & APACC TDR

通过DPACC TDR去访问DP register,通过APACC TDR去访问AP register
其定义如下：

TDR length为35bit, 低bit数据先传。

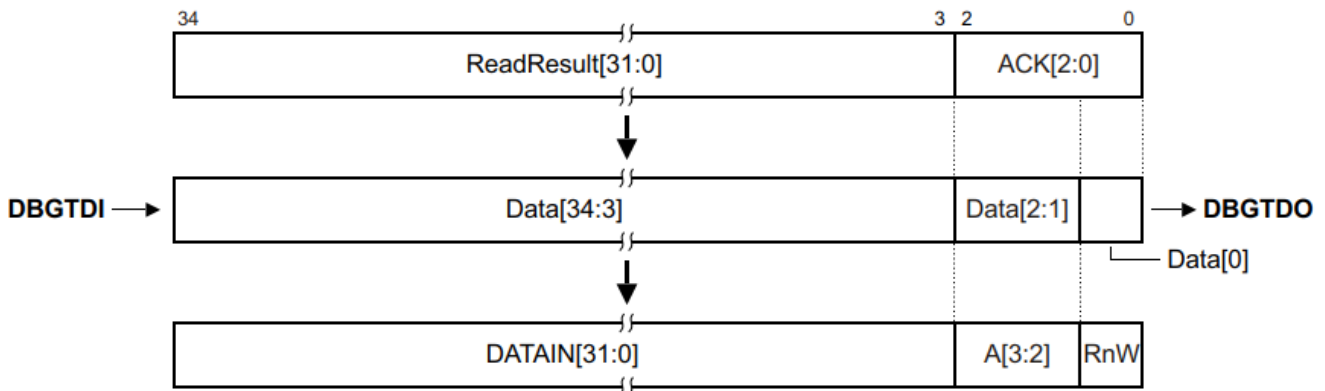
TDI输入数据格式定义如下：

- bit[34:3]传32bit写数据，
- bit[2:1]传DP或AP的地址A[3:2]
- bit[0] 表示读(1)或者写操作(0)

TDO输出数据格式定义如下：

- bit[34:3]传32bit读结果数据，
- bit[2:0]表示读响应ACK, 0b010 - OK/FAULT, 0b001 - WAIT

The operation of the DPACC and APACC registers is shown in the following figure:



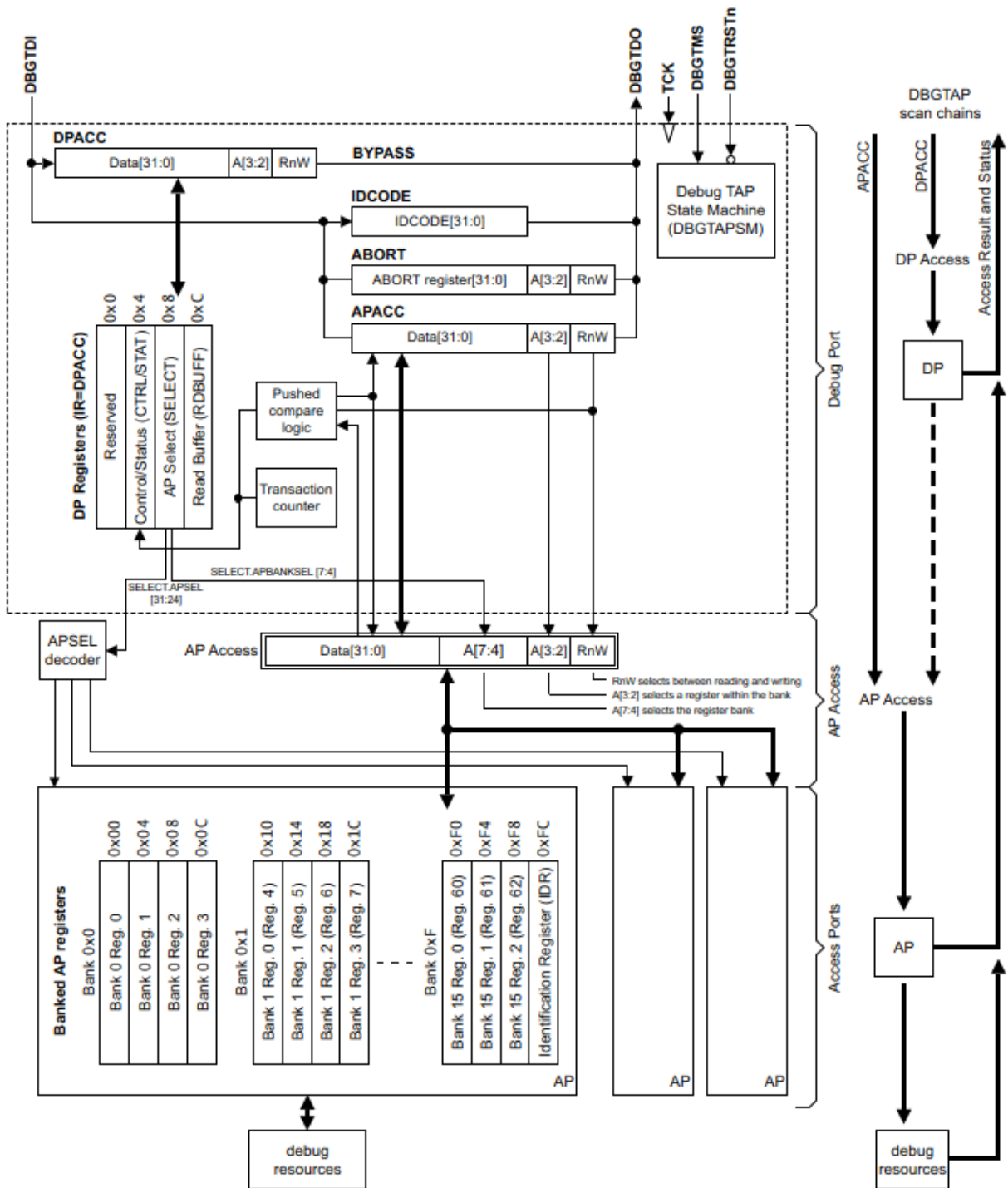


Figure A1-2 Structure of the DAP, showing DPv0 JTAG-DP accesses to a generic AP

在操作APACC时，会发起对应的AP操作，一般分为AP写和AP读操作，以下是AP读需要注意的地方。

如果是AP读的话，需要再读一次TDR，但是如果IR还是指定为APACC的话，就会再触发一次AP操作，所以此时可以临时用IR切换为DPACC，这样读结果不会重新AP操作。

通过DP.RDBUFF将AP读数据取出。

The purpose and behavior of **RDBUFF** is DATA LINK DEFINED:

JTAG-DP The Read Buffer is architecturally defined to provide a DP read operation that does not have any side effects. This definition allows a debugger to **insert a DP read of RDBUFF at the end of a sequence of operations, to return the final AP Read Result and ACK values.**

3. DP register summary

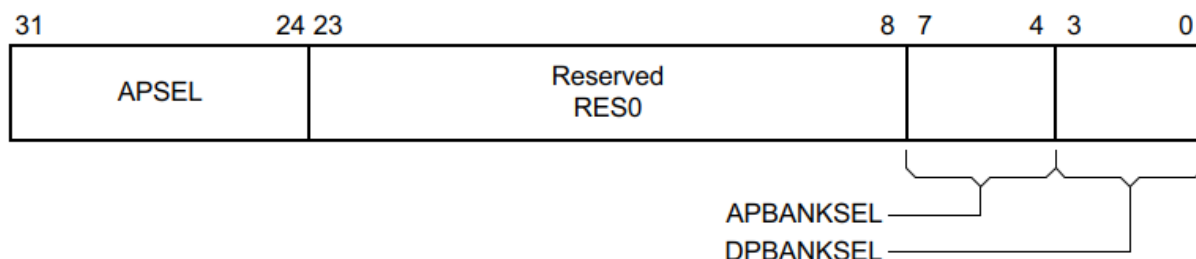
Table B3-5 JTAG-DP Register access summary.

Register	Access	Address (A ^a , SELECT.DPBANKSEL)		
		DPv0	DPv1	DPv2
CTRL/STAT	RW	0x4, -	0x4, 0x0	0x4, 0x0
DLCR	RW	-	0x4, 0x1	0x4, 0x1
DLPIDR	RO	-	-	0x4, 0x3
DPIDR	RO	-	0x0, x	0x0, x
EVENTSTAT	RO	-	-	0x4, 0x4
RDBUFF	RO	0xC, -	0xC, x	0xC, x
SELECT	WO ^b	0x8, -	0x8, x	0x8, x
TARGETID	RO	-	-	0x4, 0x2

- a. Bits [1:0] of the address are always 0b00.
- b. RW for DPv0

SELECT:

The **SELECT** bit assignments are:



4. AP register summary

4.1 AHB-AP register summary

- AP的地址[7:4]来自于 DP.SELECT寄存器的[7:4]
- AP的地址[3:2]来自于APACC TDR里面的A[3:2]地址

3.16.2 AHB-AP register summary

Table 3-205 shows the AHB-AP register summary.

Table 3-205 AHB-AP register summary

Offset	Type	Reset value	Name
0x00	RW	0x40000002	<i>AHB-AP Control/Status Word register, CSW, 0x00 on page 3-179, CSW.</i>
0x04	RW	0x00000000	<i>AHB-AP Transfer Address Register, TAR, 0x04 on page 3-181, TAR.</i>
0x08	-	-	Reserved, SBZ.
0x0C	RW	-	<i>AHB-AP Data Read/Write register, DRW, 0x0C on page 3-181, DRW.</i>
0x10	RW	-	<i>AHB-AP Banked Data registers, BD0-BD03, 0x10-0x1C on page 3-181.</i>
0x14	RW	-	
0x18	RW	-	
0x1C	RW	-	

Table 3-205 AHB-AP register summary (continued)

Offset	Type	Reset value	Name
0x20-0xF7	-	-	Reserved, SBZ.
0xF8	RO	IMPLEMENTATION DEFINED	<i>AHB-AP Debug Base Address register, ROMBASE, 0xF8 on page 3-182.</i>
0xFC	RO	0x64770001	<i>AHB-AP Identification Register, IDR, 0xFC on page 3-182, IDR.</i>

5. verilog example code

AHB-AP只能生成32bit的ahb地址，AXI-AP可以生成64bit的axi地址。

具体的APSEL可根据实际的dap与ap间的连接关系确定

当然也可以进行APSEL扫描，然后去读AP的0xFC地址，通过读出的Identification Register值来确认AP类型或AP是否存在。

```

module dap(*autoarg*/
);

task tck_cycle;
    input    t_tms;
    
```

```
input  t_tdi;
begin
  #1ns;
  // sample tdo
  sample_tdo = tdo;
  force tck_r = 1'b1;
  #20ns;
  force tms_r = t_tms;
  force tdi_r = t_tdi;
  force tck_r = 1'b0;
  #19ns;
end
endtask

task tms5;
begin
  repeat(5) tck_cycle(1'b1,1'b0);
end
  tck_cycle(1'b0, 1'b0); // rti
endtask

task shift_ir;
input  [7:0]      len;
input  [255:0]   data;
integer i=0;
begin
  tck_cycle(1'b1, 1'b0);
  tck_cycle(1'b1, 1'b0);
  tck_cycle(1'b0, 1'b0);
  tck_cycle(1'b0, 1'b0);

  shift_ret = 0;
  i=0;
  repeat(len-1) begin
    tck_cycle(1'b0, data[i]);
    shift_ret[i] = sample_tdo;
    i = i+1;
  end
  tck_cycle(1'b1, data[i]);
  tck_cycle(1'b1, 1'b0);
  tck_cycle(1'b0, 1'b0); // rti
  tck_cycle(1'b0, 1'b0); // rti

end
endtask

task shift_dr;
input  [7:0]      len;
input  [255:0]   data;
integer i=0;
begin
  tck_cycle(1'b1, 1'b0);
```

```

    tck_cycle(1'b0, 1'b0);
    tck_cycle(1'b0, 1'b0);

    shift_ret = 0;
    i=0;
    repeat(len-1) begin
        tck_cycle(1'b0, data[i]);
        shift_ret[i] = sample_tdo;
        i = i+1;
    end
    tck_cycle(1'b1, data[i]);
    tck_cycle(1'b1, 1'b0);
    tck_cycle(1'b0, 1'b0); // rti
    tck_cycle(1'b0, 1'b0); // rti

    $display("shift_dr ret = %h", shift_ret);
end
endtask

task ahbap_write;
    input  [7:0] ap_addr;
    input  [31:0] ap_data;

    begin

        // dpacc.sel ahb-ap
        shift_ir(4, 4'ha);
        #1000;
        //shift_dr(35, 35'h0); // 32b data, 2b A[3:2] (1:ctrl, 2:ap_sel, 3:read-
buffer), 1b RnW (0:write)
        shift_dr(35, {{8'h02,16'h0, ap_addr[7:4],4'h0} , 2'h2, 1'b0}); // ap_sel
= select ahb-ap (31:24 ap_sel, 7:4 ap_banksel)
        #1000;

        // dpacc.ctrl
        shift_ir(4, 4'ha);
        #1000;
        //shift_dr(35, 35'h0); // 32b data, 2b A[3:2] (1:ctrl, 2:ap_sel, 3:read-
buffer), 1b RnW (0:write)
        shift_dr(35, {32'h10000000 , 2'h1, 1'b0}); // CTRL/STAT.CDBGPWRUPREQ
        #1000;

        // apacc.read-buffer
        shift_ir(4, 4'hb);
        #1000;
        //shift_dr(35, 35'h0); // 32b data, 2b A[3:2] (dp_addr), 1b RnW
(0:write)
        shift_dr(35, {ap_data, ap_addr[3:2], 1'b0}); // read-buffer
        #1000;
    end
endtask

```

```

    end
endtask

task ahbap_read;
    input  [7:0] ap_addr;
    input  [31:0] ap_data;

    reg [31:0] xpacc_data;
    reg [2:0] ack;

    begin

        // dpacc.sel ahb-ap
        shift_ir(4, 4'ha);
        #1000;
        //shift_dr(35, 35'h0); // 32b data, 2b A[3:2] (1:ctrl, 2:ap-sel, 3:read-
buffer), 1b RnW (0:write)
        shift_dr(35, {{8'h02,16'h0, ap_addr[7:4],4'h0} , 2'h2, 1'b0}); // ap_sel
= select ahb-ap (31:24 ap_sel, 7:4 ap_banksel)
        #1000;

        // dpacc.ctrl
        shift_ir(4, 4'ha);
        #1000;
        //shift_dr(35, 35'h0); // 32b data, 2b A[3:2] (1:ctrl, 2:ap-sel, 3:read-
buffer), 1b RnW (0:write)
        shift_dr(35, {32'h10000000 , 2'h1, 1'b0}); // CTRL/STAT.CDBGPWRUPREQ
        #1000;

        // apacc.read-buffer
        shift_ir(4, 4'hb);
        #1000;
        //shift_dr(35, 35'h0); // 32b data, 2b A[3:2] (dp_addr), 1b RnW
(0:write)
        shift_dr(35, {ap_data, ap_addr[3:2], 1'b1}); // read-buffer
        #1000;

        // dpacc.reserved
        shift_ir(4, 4'ha);
        #1000;
        repeat(10) begin
            shift_dr(35, {32'h0 , 2'h0, 1'b0}); // for read ap buffer only, (ret
data is in [34:3])
            xpacc_data = shift_ret[34:3];
            ack = shift_ret[2:0];
            $display("----xpacc_data ret = %8h, ack = ", xpacc_data, ack);
            if (ack == 3'h2) break;
        end
        #1000;
    end
endtask

```

```
    end
endtask

task ahb_write;
    input [31:0] ahb_addr;
    input [31:0] ahb_data;
    begin
        ahbap_write(8'h4, ahb_addr);
        ahbap_write(8'hc, ahb_data);
    end
endtask

task ahb_read;
    input [31:0] ahb_addr;
    reg [31:0] ahb_data;
    begin
        ahbap_write(8'h4, ahb_addr);
        ahbap_read(8'hc, 32'h0);
        ahb_data = shift_ret[34:3];
        $display("ahb_read ret = %8h", ahb_data);
    end
endtask

initial begin
    #10000;
    trstn_r = 1'b1;
    #1000;
    tms5;

    // // idcode
    // shift_ir(4, 4'he);
    // #1000;
    // shift_dr(32, 32'h0);
    // #1000;

    ahb_write(32'h0680, 32'h3344);
    ahb_read(32'h0680);

    ahb_write(32'h0684, 32'h5566);
    ahb_read(32'h0684);

end
```

```
task ahb_read;
  input [31:0] ahb_addr;
  reg [31:0] ahb_data;
  begin
    // // idcode
    // shift_ir(4, 4'he);
    // #1000;
    // shift_dr(32, 32'h0);
    // #1000;
    ahbap_write(8'h4, ahb_addr);
    ahbap_read(8'hc, 32'h0);
    ahb_data = shift_ret[34:3];
    $display("ahb_read ret = %8h", ahb_data);
  end
endtask

endmodule
```