Pseudocode sequence for ARM Debug Interface using JTAG DAP:

1. Check DAP Port is accessible?
2. If DAP Port is locked, unlock it using the challenge/response mechanism
3. Initialise the DAP
4. Find the supported MEM-AP's
5. Configure DP for Selecting AP & AP's Register bank

   Before accessing AP's register , you need to configure DP's register for selecting the AP and AP's register bank
6. Read/Write Access of AP's Register
7. How to access CPU Register via APB-AP
8. How to access resource connected to system bus via AHB-AP
9. How to execute instruction via APB-AP
10. Error Handling
11. Reference

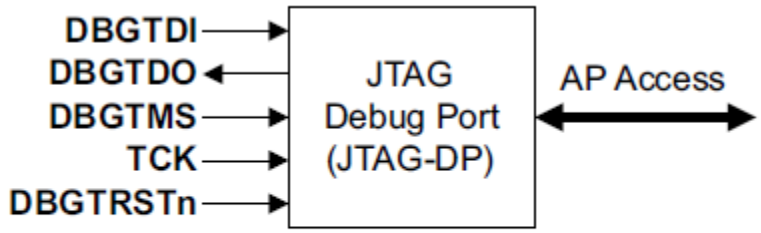

Figure 3-2 JTAG-DP physical connection

Bit stream : LSB shift out first in little endian format.

Example: While DBGTAPSM is in the Shift-IR state, the IR scan chain advances one bit for each rising edge of TCK. This means that on the first tick:

— The LSB of the IR scan chain is output on DBGTDO.
— Bit[1] of the IR scan chain is transferred to bit[0].
— Bit[2] of the IR scan chain is transferred to bit[1].
— Similarly, for every other bit n of the IR scan chain, bit[n] of the scan chain is transferred to bit[n-1].
— The value on DBGTDI is transferred to the MSB of the IR scan chain.

1. Check DAP Port is accessible: (Assumes TAP is accessible)

   Make Access to one of the DAP Register (DAP IDR) or CPU/System Memory Map Register or Addr via APB-AP which values know to be not all zeros, if its reads out all zeros concludes Port is locked.

2. To unlock the challenge/response mechanism:
   a. This is implementation specific.

3. Initialise the DAP
   a. Power Domain is enabled: Debug(Resouce or Component) + system
   b. Set CTRL/STAT.TRNMODE= 0b00 Normal operation. After a powerup reset, the value of this field is UNKNOWN.
   c. CSW.DeviceEn, Print the status of this signal – Print Warning message if its disabled
   d. CSW.DbgSwEnable Print the status of this signal
   e. Clear Sticky Error bit
   f. Enable Overrun Detection

   Code snippet: Click here

4. Find the Address offset of MEM-AP's(generalisation)
   a. Scan the list of AP's and find out MEM-AP using ID Register
   b. Check the Class ID value in IDR Register
   c. Store the AP ID offset for detected type id
   d. Code: Click here

5. Configure DP for Selecting AP & AP's Register bank
   a. • Use a DP (AP_SELECT)register write to set :
      i. — SELECT.APSEL to "Your AP ID"
      ii. — SELECT.APBANKSEL to "Your Reg. Bank ID".
         1. 0: 0x0-0xc,
         2. 1:0x10-0x1c,

3. 2:0x20-0x2c
4. 3:0x30-0x3c

b. Code: Click here

6. Read/Write Access of AP's Register
   a. The DP SELECT register addresses a MEM-AP with a connection to the Memory Map Register Resource.
   b. The AP TAR addresses the Address of the register of Resource.
   c. Perform an AP write to DRW with the Data to Write/Read:
      i. If AP is ready, so the DP returns an OK/FAULT ACK response. Check CSW.TrInProg for status.
      ii. Else Wait ACK Response -> retry the DPACC or APACC access.
   d. The TAR addresses the Address of the register of Resource, and the AP access consists of a write/read to the DRW.
      i. Therefore, the AP initiates a write/read to the Memory Map Register through its connection to the Resource
   e. Transfer of Write/Read completes.
   f. Code: Click here
7. How to access CPU Register via APB-AP
   a. To activate APB-AP : Configure DP's SELECT Register for APB-AP and APB-AP's Register bank for TAR
   b. Write TAR of APB-AP with CPU Register Address
   c. Write/Read to DRW of APB-AP will do a access to CPU Register
   d. Incase of Read access, capture the read response via Capture-DR /Shift-DR state
   e. Code: click here
8. How to access resource connected to system bus via AHB-AP
   a. To activate AHB-AP : Configure DP's SELECT Register for AHB-AP and AHB-AP's Register bank for TAR
   b. Write TAR of AHB-AP with CPU Register Address
   c. Write/Read to DRW of AHB-AP will do a access to resource connected to System Bus
   d. In-case of Read access, capture the read response via Capture-DR /Shift-DR state
   e. Code: click here

9. How to execute instruction via APB-AP

   For example: To issue an instruction STR R0, [R1] to CPU ; R0=0xABCDABCD, R1=0x80000000
   3 steps involved:
   - Load R0 register with 0xABCDABCD
   - Load R1 register with 0x08000000
   - Execute STR R0, [R1] instruction

   Cortex Debug Memory-Mapped Registers shall be used : DTRRX,ITR , DSCR , DTRTX , DRCR

   Steps: Enter the Debug state using DRCR

   1. Activate APB-AP
   2. Set the APB-AP TAR = DTRRX address to access DTRRX

a. Write 0x80001080 to APB-AP. TARregister.0x80001080 is the address for DTRRX register in the CPU's debug component. All CoreSight debug components are memory mapped according to the ROM table. Each entry in the ROM table specifies the offset address of each debug component from the base address of the ROM table. The offset address for the ARM core is 0x1000.The base address of the ROM table is 0x80000000. Adding 0x80000000 to 0x1000 gives the base address of the ARM core debug component.0x80is the offset address of the DTRRX register in the ARM core's debug component.

3. Set the APB-AP data to write: DRW=0xABCDABCD
    a. Write0xABCDABCDto APB-AP.DRWregister.
4. Execute CPU instruction to move the data in DTRRX register to R0 register:
    a. Write 0x80001084 to APB-AP's TAR register. 0x80001084 is the address for ITR register in the CPU's debug component.
    b. Write the opcode for 'MRC p14,0, r0, c0, c5, 0' to APB-Aps' DRW. Assume opcode is 0xXXXXXXXX. This operation initiates the value 0xXXXXXXXX to be written to the ITR register.
    c. CPU executes the instruction. When the instruction execution completes, the value 0xABCDABCD is stored to R0 register.
5. Set the APB-AP TAR = DTRRX address to access DTRRX:
    a. Write 0x80001080 to APB-AP's TAR register 0x80001080 is the address for DTRRX register in the CPU's debug component.
6. Set the APB-AP data to write: Write R1's value to DTRRX
    a. Write0x08000000to APB-AP.DRWregister. This operation initiates the value of 0x08000000to be written to DTRRXregister in the CPU via the Debug APB bus.
7. Execute CPU instruction to move the data in DTRRXregister to R1 register:
    a. First, write 0x80001084 to the APB-AP's TAR register.0x80001084 is the address for the ITR register in the CPU's debug component.
    b. Second, write the opcode for 'MRCp14,0, r1, c0, c5, 0' to APB-APs' DRW. Assume opcode is 0xXXXXXXXX. This operation initiates the value 0xXXXXXXXX to be written to the ITR register.
    c. CPU executes the instruction. When the instruction execution completes, the value 0x08000000 is stored to R1 register.
8. Set the APB-AP address to access: TAR = ITR address
    a. Write0x80001084to the APB-AP's TARregister.0x80001084is the address for the ITR register in the CPU's debug component.
9. Set the APB-AP data to write: Write the Opcode of STR R0,[R1] to DRW to load in to ITR
    a. • Write the opcode for 'STRR0, [R1]'to APB-AP.DRW. Assume opcode is 0xXXXXXXXX.
    b. • The CPU executes the instruction. When the instruction execution ccompletes, the value 0xABCDABCD is written to the memory address 0x08000000.
10. Check the status of Memory write operation using DSCR and exit the Debug state using DRDR.

10. Error Handling

a. Read and write errors

A read or write error can occur in the DAP or in the resource being accessed. In either case, when the error is detected, the Sticky Error flag CTRL/STAT.STICKYERR is set to 0b1.

For example, a read or write error might occur if the debugger makes an AP transaction request while the debug power domain is powered down. See Power and reset control on page B1-46 for information about power domains.

b. STICKYERR, Sticky flags and DP error responses

Sticky flags signal transaction errors and are persistent between transactions. When set, a sticky flag remains set

until the debugger actively clears it, even if the condition that caused the flag to be set no longer applies.

After performing a series of APACC transactions, a debugger must check the CTRL/STAT register to check if an

error occurred. If the debugger finds that a sticky flag is set, it clears the flag, and, if necessary, initiates extra

APACC transactions to determine why the sticky flag was set. Because the flags are sticky, the debugger does not

have to check the flags after every transaction, and must only check the CTRL/STAT register periodically, which

reduces the overhead of checking for errors.

When an error is flagged, the current transaction

JTAG-DP, all implementations

• Access is R/W1C.

• To clear STICKYCMP to 0b0, write 0b1 to it, which signals the DP to clear the flag and set it to 0b0. A single write of the CTRL/STAT register can be used to clear multiple flags if necessary.

STICKYCMP can also be cleared using the ABORT.STKERRCLR field.

B2 DP Reference Information

B2.2 DP register descriptions

ID030917 Non-Confidential

SW-DP, all implementations, and JTAG-DP, DPv1 and higher

• Access is RO/WI.

• To clear STICKYCMP to 0b0, write 0b1 to the ABORT.STKCMPCLR field in the ABORT register. A single write of the ABORT register can be used to clear multiple flags if necessary.

c. STICKYORUN,

d. TrInProg, bit[7]

Transfer in progress. This field has one of the following values:

0b0 The connection to the memory system is idle.

0b1 A transfer is in progress on the connection to the memory system.

After an ABORT operation, debug software can read this bit to check whether the aborted transaction completed.

11. References

- ARM IHI0031C_debug_interface_as.pdf
- ARM DDI0314H_coresight_components_trm.pdf
- https://simba-os.readthedocs.io/en/latest/library-reference/drivers/network/jtag_soft.html

- http://openocd.org/doc/pdf/openocd.pdf
- http://openocd.org/doc/doxygen/html/arm__adi__v5_8h_source.html
- Google Test Framework
  https://github.com/eerimoq/simba/blob/master/tst/drivers/software/network/jtag_soft/main.c
  https://paginas.fe.up.pt/~jmf/hibu2k2/contents/contents.htm

12.

```
Int main( void ){
    mem_ap_t mem_ap; /* list of mem ap */

   /* Init DAP DP */
    dap_dp_init();

   /* Scan and find out the AP offset for MEM-AP */
   dap_mem_ap_scan(&mem_ap);

  /* After finding valid mem-ap , select one accordingly */
  If (false == WriteResetReason( ResetReasonAddr ,  ValueToWrite )) {
      LOG(" Writing Reset Reason Failed ");
      Return ;
  }

  If (false == WriteMemMappedRegister(ResetRegisterAddr , ValueToWrite)){
          LOG("Reset Command Execution Failed");
          Return;
  }
  LOG("Reset Executed Successfully")
}
Bool WriteMemMappedRegister(Addr , Val ){

   If (AP_TYPE_INVALID  !=  mem_ap.apb)
   {
        dap_select_ap(mem_ap.apb) // this is implementation specific
        ap_write( MEM_AP_REG_TAR, Value=Addr)
        ap_write( MEM_AP_REG_DRW, Value=Val)
        ap_Capture(&AckDrw)
          return is_write_Done() ;
    }
  return false;
}
```

```
bool WriteResetReason(ResetReasonAddr, ValueToWrite){

    If (AP_TYPE_INVALID  !=  mem_ap.ahb)
    {
        dap_select_ap(mem_ap.ahb)  // this is implementation specific
        ap_write( MEM_AP_REG_TAR, Value=ResetReasonAddr)
        ap_write( MEM_AP_REG_DRW, Value=ValueToWrite)
          return is_write_Done() ;
    }
   return false;

}
```

dap_dp_init(){

```
   int dp_ctrl_stat=0x0;

   // Enable Power Domain
   dp_ctrl_stat |= CDBGPWRUPREQ;
   dp_ctrl_stat |= CSYSPWRUPREQ;

   dp_write(DP_CTRL_STAT, &dp_ctrl_stat)

   LOG("DAP: wait CDBGPWRUPACK & CSYSPWRUPACK ");
   dp_poll(DP_CTRL_STAT , (CDBGPWRUPACK|CSYSPWRUPACK))

   dp_ctrl_stat=0x0;

   // clear sticky error
   dp_ctrl_stat |= SSTICKYERR;

   // Enable Overrun Detection
   dp_ctrl_stat |= CORUNDETECT

   dp_write(DP_CTRL_STAT, &dp_ctrl_stat)
}
```

dp_write(reg, val){

```
    Ir_write(DPACC)
    switch(reg)
    {
         Case: DP_AP_SELECT:
                dr_write( val<<33 | (0x8 >> 2)<< 1)
    }

}
```

```
dap_select_ap(ap_num, ap_bank_id){

   dp_write(DP_AP_SELECT,( (ap_num<<24)|(ap_bank_id<<4)))

}

void    dap_mem_ap_scan(mem_ap_t * mem_ap)
{
    {
              int val = dap_find_memap(AP_TYPE_AHB_AP);
             if(val == AP_TYPE_INVALID  )  LOG("Inavlid  AHB AP Found ");
            mem_ap->ahb=val;

             val = dap_find_memap(AP_TYPE_APB_AP);
             if(val == AP_TYPE_INVALID  )  LOG("Inavlid  APB AP Found ");
             mem_ap->apb =val;

             val = dap_find_memap(AP_TYPE_AXI_AP);
             if(val == AP_TYPE_INVALID  ) LOG("Inavlid  AXI AP Found ");
             mem_ap->axi =val;

             val = dap_find_memap(AP_TYPE_JTAG_AP);
             if(val == AP_TYPE_INVALID  )  LOG("Inavlid  JTAG AP Found ");
             mem_ap->jtag =val;
     }
}

int dap_find_memap(type_to_find)

{

    /* Maximum AP number is 255 since the SELECT register is 8 bits */
    for (uint32_t ap_num = 0; ap_num <= 255; ap_num++) {
      uint32_t id_val = 0;
      retval = dap_select_ap(ap_num,0xF )
      retval = ap_read(AP_REG_IDR,&id_val)

     /* IDR bits:
      * 31-28 : Revision
      * 27-24 : JEDEC bank (0x4 for ARM)
      * 23-17 : JEDEC code (0x3B for ARM)
      * 16-13 : Class (0b1000=Mem-AP)
      * 12-8  : Reserved
      *  7-4  : AP Variant (non-zero for JTAG-AP)
      *  3-0  : AP Type (0=JTAG-AP 1=AHB-AP 2=APB-AP 4=AXI-AP)
      */
```

```
      if ((retval == ERROR_OK) &&                /* Register read success */
      ((id_val & IDR_JEP106) == IDR_JEP106_ARM) && /* Jedec codes match */
      ((id_val & IDR_CLASS) == AP_CLASS_MEM_AP) && /* Class Match */
      ((id_val & IDR_TYPE) == type_to_find)) {     /* type matches*/

         LOG("Found %s at AP index: %d (IDR=0x%08"  ")",
               (type_to_find == AP_TYPE_AHB_AP)  ? "AHB-AP"  :
               (type_to_find == AP_TYPE_APB_AP)  ? "APB-AP"  :
               (type_to_find == AP_TYPE_AXI_AP)  ? "AXI-AP"  :
               (type_to_find == AP_TYPE_JTAG_AP) ? "JTAG-AP" : "Unknown",
               ap_num, id_val);

         return ap_num;
      }
   }
   return AP_TYPE_INVALID  ;
}

#define DP_CTRL_STAT   BANK_REG(0x0, 0x4) /* DPv0+: rw */

#define SSTICKYERR   (1UL << 5)

#define CORUNDETECT   (1UL << 0)

#define CDBGPWRUPREQ   (1UL << 28)

#define CDBGPWRUPACK   (1UL << 29)

#define CSYSPWRUPREQ   (1UL << 30)

#define CSYSPWRUPACK   (1UL << 31)

/* MEM-AP register addresses */

#define MEM_AP_REG_CSW      0x00

#define MEM_AP_REG_TAR      0x04

#define MEM_AP_REG_TAR64    0x08       /* RW: Large Physical Address Extension */

#define MEM_AP_REG_DRW      0x0C       /* RW: Data Read/Write register */

#define MEM_AP_REG_BD0      0x10       /* RW: Banked Data register 0-3 */

#define MEM_AP_REG_BD1      0x14

#define MEM_AP_REG_BD2      0x18

#define MEM_AP_REG_BD3      0x1C

#define MEM_AP_REG_MBT      0x20       /* --: Memory Barrier Transfer register */
```

```
#define MEM_AP_REG_BASE64   0xF0       /* RO: Debug Base Address (LA) register */
#define MEM_AP_REG_CFG      0xF4       /* RO: Configuration register */
#define MEM_AP_REG_BASE     0xF8       /* RO: Debug Base Address register */
/* Generic AP register address */
#define AP_REG_IDR          0xFC       /* RO: Identification Register */
#define IDR_JEP106  (0x7FFUL << 17)
#define IDR_TYPE    (0xFUL << 0)
#define IDR_JEP106_ARM 0x04760000
#define IDR_CLASS   (0xFUL << 13)
#define DP_SELECT       BANK_REG(0x0, 0x8) /* DPv0+: JTAG: rw; SWD: wo */
/*
* MEM Access Port types
*/
typedef struct mem_ap {
   uint8_t jtag;  /* JTAG-AP - JTAG master for controlling other JTAG devices */
   uint8_t  ahb  ; /* AHB Memory-AP */
   uint8_t  apb  ;  /* APB Memory-AP */
   uint8_t  axi  ;  /* AXI Memory-AP */
} mem_ap_t;

/*
* Access Port classes
*/
enum ap_class {
AP_CLASS_NONE   = 0x00000,  /* No class defined */
AP_CLASS_MEM_AP = 0x10000,  /* MEM-AP */
};
/*
  * Access Port types
```

```
*/
enum ap_type {
  AP_TYPE_JTAG_AP = 0x0,  /* JTAG-AP - JTAG master for controlling other JTAG devices */
  AP_TYPE_AHB_AP  = 0x1,  /* AHB Memory-AP */
  AP_TYPE_APB_AP  = 0x2,  /* APB Memory-AP */
  AP_TYPE_AXI_AP  = 0x4,  /* AXI Memory-AP */
  AP_TYPE_INVALID  = -1,  /* Invalid Memory-AP */
};
```