Semiconductors

AN10039

ISP1582/83 Firmware Programming Guide

Rev. 02 — 3 January 2005 Application note

Document information

Info	Content
Keywords	isp1582; isp1583; peripheral controller; usb; universal serial bus
Abstract	This document explains the firmware programming of the ISP1582/83.

PHILIPS

Revision history

Rev	Date	Description
2.0	20050103	Second release: updated Section 2.6.
1.0	20040603	First release.

Contact information

For additional information, please visit: http://www.semiconductors.philips.com

For sales office addresses, please send an email to: sales.addresses@www.semiconductors.philips.com

1. Introduction

The ISP1582/83 is a Hi-Speed Universal Serial Bus (USB) peripheral controller that provides a flexible interface for a wide range of microcontrollers. The high-speed microcontroller interface increases system throughput and reduces processor utilization.

The ISP1582/83 can be configured to function, with a common USB bus enumeration process, as a Generic Direct Memory Access (GDMA) application or a mass storage application. This document is divided based on the USB enumeration and the two supported applications.

Fig 1 shows the ISP1582/83 firmware-programming model.

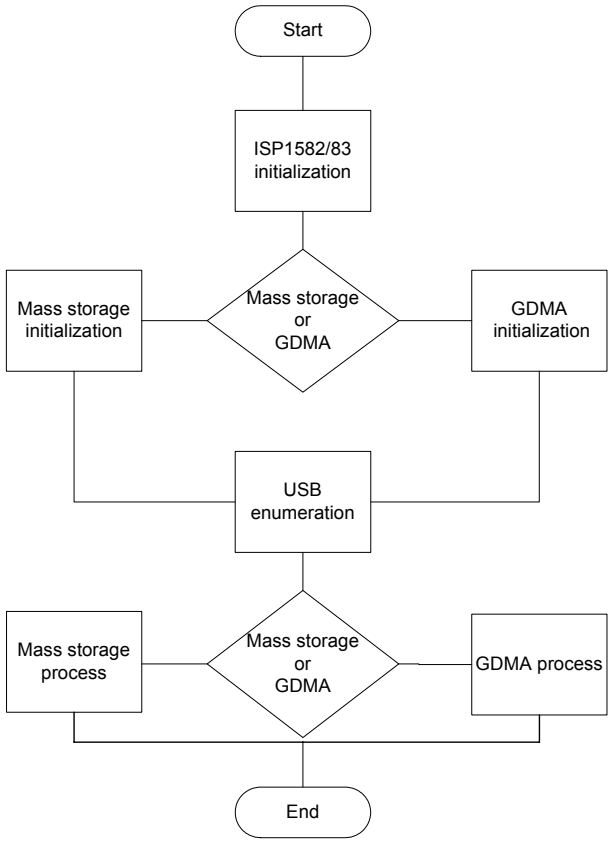


Fig 1. Flowchart of the ISP1582/83 firmware-programming model.

2. ISP1582/83 initialization routine

2.1 Initializing the ISP1582/83 registers

The initialization routine depends on the application targeted for the ISP1582/83. The routine has most of the initializations common to all applications, except for the DMA Configuration and DMA Hardware registers that determine the function of the ISP1582/83. Fig 2 shows the flowchart of the ISP1582/83 register initialization.

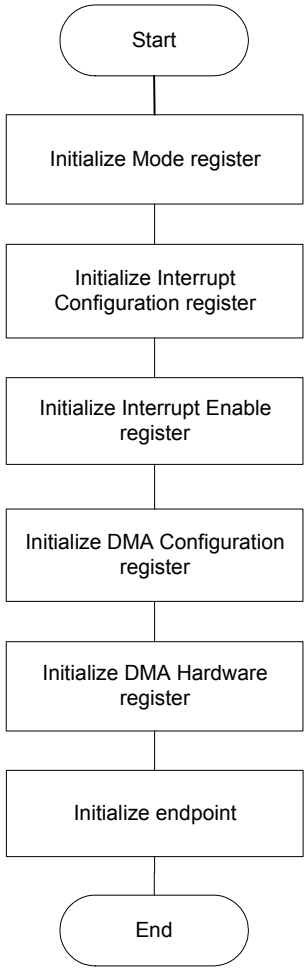


Fig 2. Register initialization.

2.2 Initializing the Mode register

After the power-on reset, the processor initializes the Mode register (see Table 1:). This register can be programmed, depending on the user application.

Table 1: Mode register: bit allocation

Bit	15	14	13	12	11	10	9	8
Symbol	TEST2	TEST1	TEST0		reserved		DMACLKON	VBUSSTAT
Reset	-	-	-	-	-	-	0	-
Bus reset	-	-	-	-	-	-	0	-
Access	R	R	R	R	R	R	R/W	R
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Symbol	CLKAON	SNDRSU	GOSUSP	SFRESET	GLINTENA	WKUPCS	PWRON	SOFTCT
Reset	0	0	0	0	0	0	0	0
Bus reset	0	0	0	0	unchanged	0	0	unchanged
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

The DMACLKON bit of the Mode register determines whether the clock should be supplied to the DMA core of the ISP1582/83. Disabling the bit at initialization saves power. The DMA clock can be switched on when DMA is required using the DMACLKON bit.

Power can also be saved by disabling clock CLKAON during suspend mode. It is recommended that you use this power saving mechanism to save power when the ISP1582/83 is configured to bus-powered applications. Next, set the global interrupt enable (bit GLINTENA) to enable the interrupt mechanism of the ISP1582/83. The ISP1582/83 allows you to wake up the system from suspend mode by toggling the chip select (pin CS_N) of the ISP1582/83.

The ISP1582/83 allows you to design systems to support the sleep mode in which power to the ISP1582/83 is turned off to save power. The PWRON bit in the Mode register configures the ISP1582/83 into two modes of operation when it is in the suspend state. When the bit is set to logic 0, the firmware must issue an unlock command to the ISP1582/83 before any write operation to a register can be performed. This prevents accidental writing to registers when the processor wakes up from the power saving mode.

The SoftConnect™ feature (bit SOFTCT) is used to connect the pull-up resistor to the DP line of the ISP1582/83. This gives the users complete authority on when to connect and disconnect from the USB bus. The SOFTCT bit is used together with V_{BUS} (bit VBUSSTAT) that reflects whether the USB cable is plugged in or out.

2.3 Initializing the Interrupt Configuration register

After setting the Mode register, the Interrupt Configuration register (see Table 2:) is initialized. This register controls how an interrupt is generated for the control pipe, IN endpoint and OUT endpoint.

Table 2: Interrupt Configuration register: bit allocation

Bit	7	6	5	4	3	2	1	0
Symbol	CDBGMOD[1:0]		DDBGMODIN[1:0]		DDBGMODOUT[1:0]		INTLVL	INTPOL
Reset	1	1	1	1	1	1	0	0
Bus reset	1	1	1	1	1	1	unchanged	unchanged
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

The interrupt generation depends on the type of mode setting specified in the register; see Table 3:. CDBGMOD is for the control pipe, DDBGMODIN is for the IN pipe, and DDBGMODOUT is for the OUT pipe. The register sets interrupt to the level or pulse trigger by using the INTLVL bit. The polarity of the interrupt signal is control by INTPOL.

Table 3: Debug mode settings

Value	CDBGMOD	DDBGMODIN	DDBGMODOUT
00h	Interrupt on all ACK and NAK	Interrupt on all ACK and NAK	Interrupt on all ACK, NYET and NAK
01h	Interrupt on all ACK	Interrupt on ACK	Interrupt on ACK and NYET
1Xh	Interrupt on all ACK and first NAK	Interrupt on all ACK and first NAK	Interrupt on all ACK, NYET and first NAK

The typical function of the Interrupt Configuration register is to enable the ACK interrupt.

The following figures show the CATC trace of the ACK interrupt with the red arrows indicating the interrupt signal that will be generated on the INT pin, when ACK condition occurs.

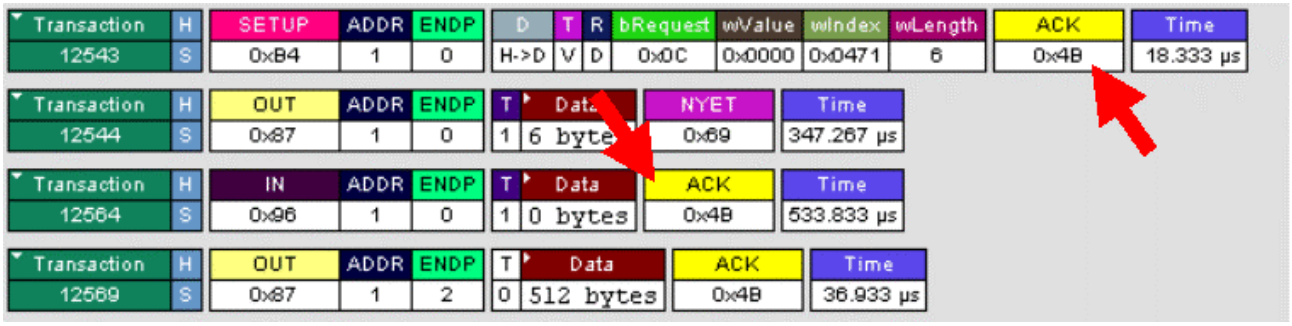


Fig 3. CATC capture of setup and OUT ACK interrupt.

The following sample code programs the IN endpoint interrupt to all ACK conditions:

```
Void Init_D14_SFR(void)
{
//Set the control pipe to ACK only interrupt
//Set the IN pipe to ACK only interrupt
//Set OUT pipe to ACK and NYET interrupt
D14_Cntrl_Reg.D14_INT_CONFIG = 0x54;
}
```

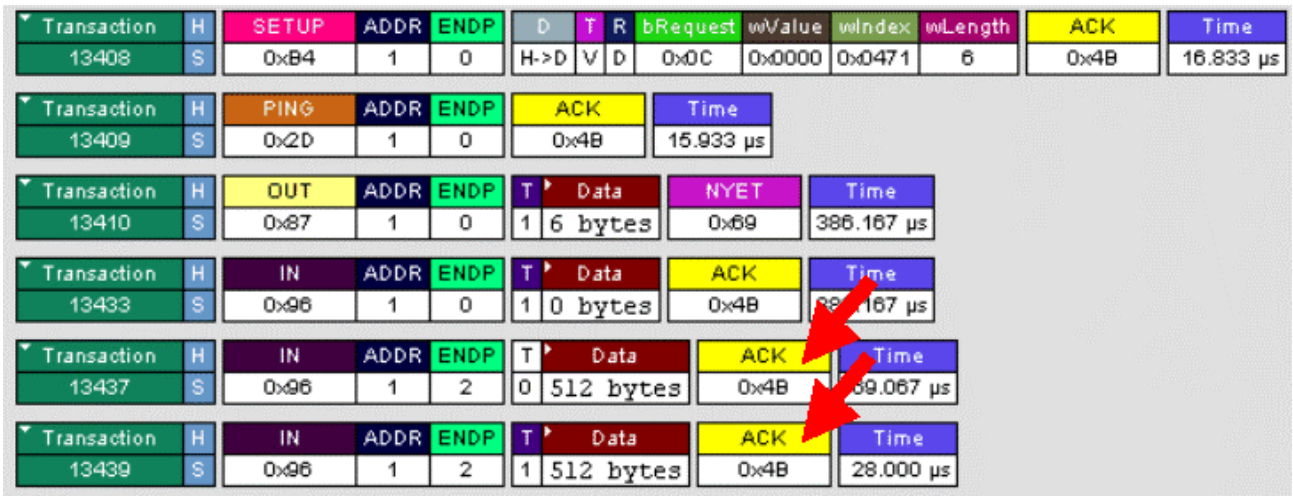


Fig 4. CATC capture of IN ACK interrupt.

The OUT endpoint is programmed to all ACK and NYET interrupts as shown in Fig 5.

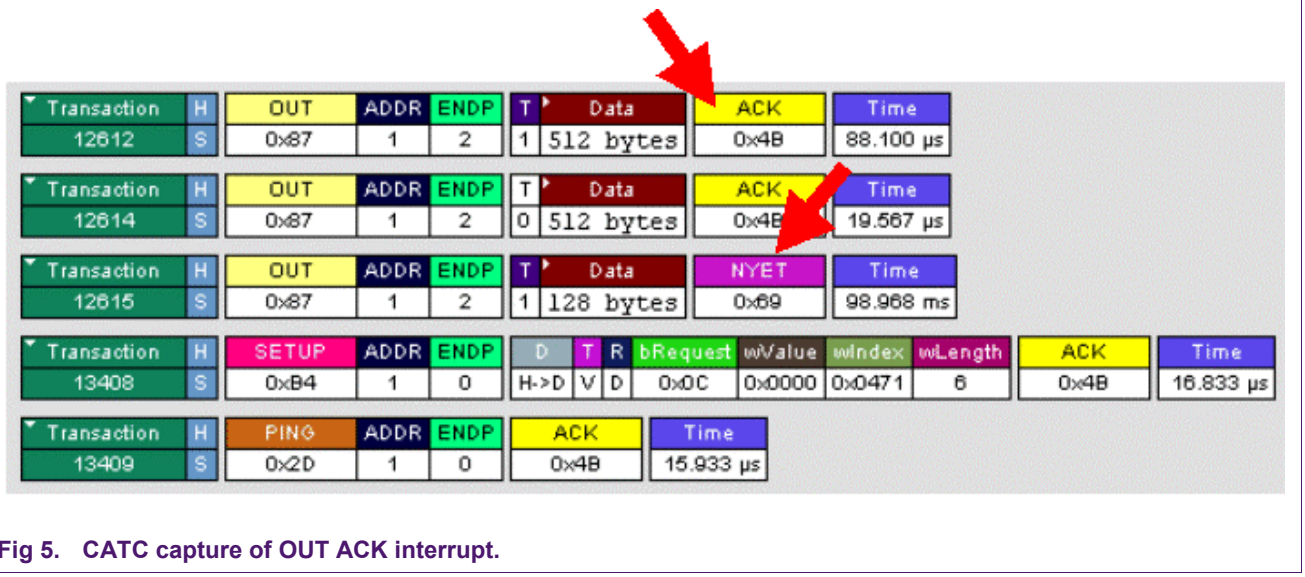


Fig 5. CATC capture of OUT ACK interrupt.

2.4 Initializing the Interrupt Enable register

Next is the initialization of the Interrupt Enable register (see Table 4:). Setting the respective bit to logic 1 will enable the interrupt signal to be generated on the INT pin of the ISP1582/83. Take into consideration the interrupt service routine because an AND operation must be performed on the bits in the Interrupt Reason register with the enable interrupt. The Interrupt Reason register reflects the status bit of the respective event, even if the enable bit is not set in the Interrupt Enable register.

Table 4: Interrupt Enable register: bit allocation

Bit	31	30	29	28	27	26	25	24
Symbol	reserved						IEP7TX	IEP7RX
Reset	-	-	-	-	-	-	0	0
Bus reset	-	-	-	-	-	-	0	0
Access	-	-	-	-	-	-	R/W	R/W
Bit	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Symbol	IEP6TX	IEP6RX	IEP5TX	IEP5RX	IEP4TX	IEP4RX	IEP3TX	IEP3RX
Reset	0	0	0	0	0	0	0	0
Bus reset	0	0	0	0	0	0	0	0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Symbol	IEP2TX	IEP2RX	IEP1TX	IEP1RX	IEP0TX	IEP0RX	reserved	IEP0 SETUP
Reset	0	0	0	0	0	0	-	0
Bus reset	0	0	0	0	0	0	-	0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Symbol	IEVBUS	IEDMA	IEHS_STA	IERESM	IESUSP	IEPSOF	IESOF	IEBRST
Reset	0	0	0	0	0	0	0	0
Bus reset	0	0	0	0	0	0	0	unchanged
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

The following is a sample code for the interrupt service routine:

```
Void Int_Ext_0(void) interrupt 0 using 1
{
       // Read in USB interrupt reason register
        USB_Int_Flag.VALUE = D14_Cntrl_Reg.D14_INT.VALUE;
        USB_Int_Flag.VALUE &= D14_INT_ENABLE;

       //Read in DMA interrupt register
        DMA_Int_Flag.VALUE = D14_Cntrl_Reg.D14_DMA_INT.VALUE;
        DMA_Int_Flag.VALUE &= D14_DMA_INT_ENABLE;

     //Clear DMA interrupt register
   D14_Cntrl_Reg.D14_DMA_INT.VALUE = DMA_Int_Flag.VALUE;
     // Clear USB interrupt reason register
   D14_Cntrl_Reg.D14_INT.VALUE = USB_Int_Flag.VALUE;
}
```

2.5 Initializing the DMA Configuration and Hardware registers

Next, you configure the DMA Configuration register (see Table 5:) and DMA Hardware register (see Table 6:) to either GDMA or mass storage application. Based on the application, some associated registers must be programmed.

Table 5: DMA Configuration register: bit allocation

Bit	15	14	13	12	11	10	9	8
Symbol	reserved		ATA_ MODE	DMA_MODE[1:0]			PIO_MODE[2:0]	
Reset	0	0	0	0	0	0	0	0
Bus reset	0	0	0	0	0	0	0	0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Symbol	DIS_XFER _CNT		reserved		MODE[1:0]		reserved	WIDTH
Reset	0	0	0	0	0	0	0	1
Bus reset	0	0	0	0	0	0	0	1
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Table 6: DMA Hardware register: bit allocation

Bit	7	6	5	4	3	2	1	0
Symbol	ENDIAN[1:0]		EOT_POL	MASTER	ACK_POL	DREQ_ POL	WRITE_ POL	READ_POL
Reset	0	0	0	0	0	1	0	0
Bus reset	0	0	0	0	0	1	0	0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

The ISP1582/83 can be configured to various operation modes, depending on the combination of the various bits in the DMA Configuration and DMA Hardware registers. Table 7: illustrates the valid combinations.

Table 7: DMA configuration

ATA mode	Master	DMA mode	PIO mode	DIS_XFER _CNT	Mode	Width	Description
0	0	Not used	Not used	0	0	1	GDMA Slave mode DMA Transfer Counter DIOR as read strobe DIOW as write strobe 16-bit data bus
		Not used	Not used	0	1	1	GDMA Slave mode DMA Transfer Counter DIOR as read strobe DACK as write strobe 16-bit data bus
		Not used	Not used	0	2	1	GDMA Slave mode DMA Transfer Counter DACK as read strobe DACK as write strobe 16-bit data bus

ATA mode	Master	DMA mode	PIO mode	DIS_XFER _CNT	Mode	Width	Description
		Not used	Not used	0	0	0	GDMA Slave mode DMA Transfer Counter DIOR as read strobe DIOW as write strobe 8-bit data bus
		Not used	Not used	0	1	0	GDMA Slave mode DMA Transfer Counter DIOR as read strobe DACK as write strobe 8-bit data bus
		Not used	Not used	0	2	0	GDMA Slave mode DMA Transfer Counter DACK as read strobe DACK as write strobe 8-bit data bus
		Not used	Not used	1	2	0	GDMA Slave mode EOT mode DACK as read strobe DACK as write strobe 8-bit data bus
1	0	0	0	Not used	Not used	1	ATA mode MDMA Mode 0 strobe timing PIO mode 0 timing
		1	1	Not used	Not used	1	ATA mode MDMA Mode 1 strobe timing PIO Mode 1 timing
		2	2	Not used	Not used	1	ATA mode MDMA Mode 2 strobe timing PIO Mode 2 timing
		2	3	Not used	Not used	1	ATA mode MDMA Mode 2 strobe timing PIO Mode 3 timing
		2	4	Not used	Not used	1	ATA mode MDMA Mode 2 strobe timing PIO Mode 4 timing
1	1	Not used	Not used	Not used	0	1	GDMA Master mode DMA Transfer Counter ATA MDMA 0 Strobe timing 16-bit data bus
		Not used	Not used	Not used	1	1	GDMA Master mode DMA Transfer Counter ATA MDMA 1 Strobe timing 16-bit data bus
		Not used	Not used	Not used	2	1	GDMA Master mode DMA Transfer Counter ATA MDMA 2 Strobe timing 16-bit data bus

ATA mode	Master	DMA mode	PIO mode	DIS_XFER_CNT	Mode	Width	Description
		Not used	Not used	Not used	0	0	GDMA Master mode DMA Transfer Counter ATA MDMA 0 Strobe timing 8-bit data bus
		Not used	Not used	Not used	1	0	GDMA Master mode DMA Transfer Counter ATA MDMA 1 Strobe timing 8-bit data bus
		Not used	Not used	Not used	2	0	GDMA Master mode DMA Transfer Counter ATA MDMA 2 Strobe timing 8-bit data bus
0	1	Invalid	Invalid	Invalid	Invalid	Invalid	Invalid Mode

When DIS_XFER_CNT is set to logic 1, the DMA transfer counter is not in use, and the DMA termination is done using the input signal of pin EOT. This mode is called the EOT mode. The polarity of pin EOT can be set using the EOT_POL bit in the DMA Hardware register.

2.6 Initializing the ISP1582/83 endpoint

The endpoint FIFO configuration is done using the Endpoint MaxPacketSize register (see Table 8:) and the Endpoint Type register (Table 9:), and indexed using the Endpoint Index register (see Table 10:).

For the same endpoint number, ensure that initialization is done for both the IN and OUT, even if only one side is used.

Table 8: Endpoint MaxPacketSize register: bit allocation

Bit	15	14	13	12	11	10	9	8
Symbol	reserved			NTRANS[1:0]		FFOSZ[10:8]		
Reset	-	-	-	0	0	0	0	0
Bus reset	-	-	-	0	0	0	0	0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Symbol	FFOSZ[7:0]							
Reset	0	0	0	0	0	0	0	0
Bus reset	0	0	0	0	0	0	0	0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Table 9: Endpoint Type register: bit allocation

Bit	15	14	13	12	11	10	9	8
Symbol	reserved							
Reset	-	-	-	-	-	-	-	-
Bus reset	-	-	-	-	-	-	-	-
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Symbol	reserved			NOEMPKT	ENABLE	DBLBUF	ENDPTYP[1:0]	
Reset	-	-	-	0	0	0	0	0
Bus reset	-	-	-	0	0	0	0	0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Table 10: Endpoint Index register: bit allocation

Bit	7	6	5	4	3	2	1	0
Symbol	reserved		EP0SETUP	ENDPIDX[3:0]				DIR
Reset	-	-	0	0	0	0	0	0
Bus reset	-	-	0	0	0	0	0	0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Initialization of the endpoint is done as depicted by the flowchart in Fig 6.

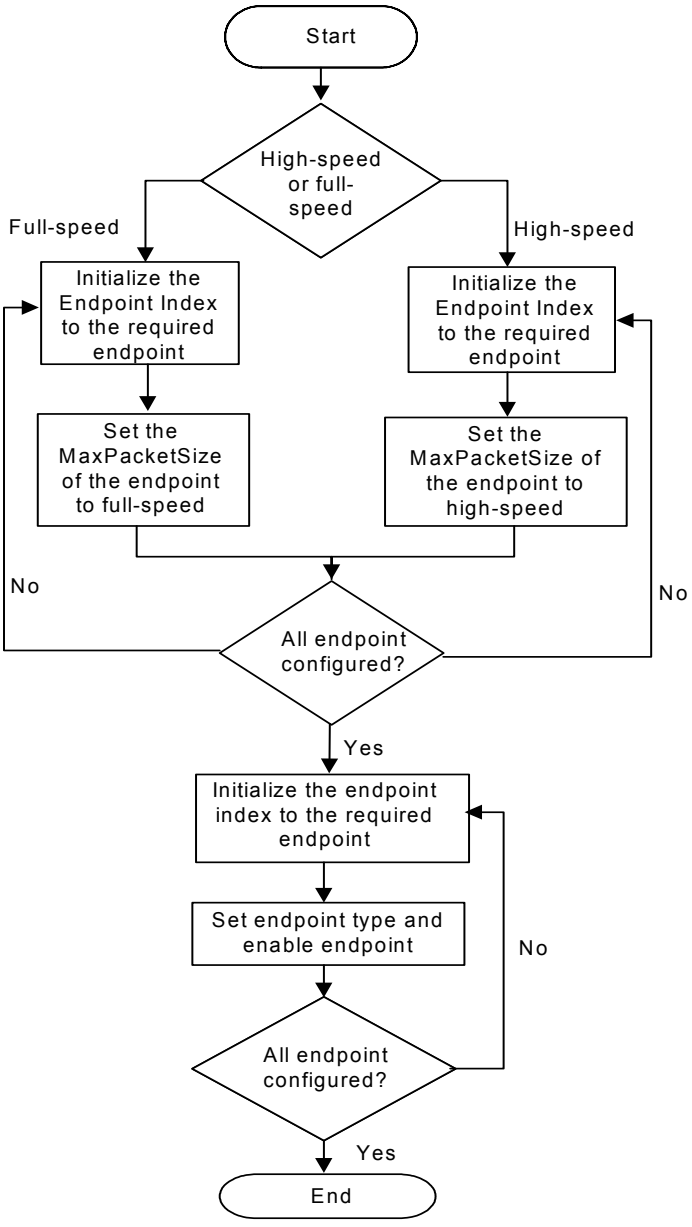


Fig 6. Initializing endpoint.

The initialization routine is used when:

- The system is powered up.
- A bus-reset event occurs.
- There is a change from full-speed to high-speed.

The following is a sample code for endpoint initialization:

```
void Init_Endpoint(void)
{

//check if device in full speed state
if(Kernel_Flag.BITS.HS_FS_State == FULL_SPEED)
```

```
{
//Bulk Out MaxPacketSize Endpoint
   D14_Cntrl_Reg.D14_ENDPT_INDEX = 4;
   D14_Cntrl_Reg.D14_ENDPT_MAXPKTSIZE.VALUE = 0x4000;

//Bulk In MaxPacketSize Endpoint
   D14_Cntrl_Reg.D14_ENDPT_INDEX = 5;
D14_Cntrl_Reg.D14_ENDPT_MAXPKTSIZE.VALUE = 0x4000;

   //Bulk Out Endpoint Type
D14_Cntrl_Reg.D14_ENDPT_INDEX = 4;
   D14_Cntrl_Reg.D14_ENDPT_TYPE.VALUE = 0x1600;

   //Bulk In Endpoint Type
D14_Cntrl_Reg.D14_ENDPT_INDEX = 5;
   D14_Cntrl_Reg.D14_ENDPT_TYPE.VALUE = 0x1600;

//enable FIFO
   D14_Cntrl_Reg.D14_ENDPT_INDEX = 4;
   D14_Cntrl_Reg.D14_ENDPT_TYPE.VALUE |= 0x0800;

   //enable FIFO
   D14_Cntrl_Reg.D14_ENDPT_INDEX = 5;
   D14_Cntrl_Reg.D14_ENDPT_TYPE.VALUE |= 0x0800;

}

//check if device in high speed
if(Kernel_Flag.BITS.HS_FS_State == HIGH_SPEED)
{

   //Bulk Out MaxPacketSize Endpoint
D14_Cntrl_Reg.D14_ENDPT_INDEX = 4;
   D14_Cntrl_Reg.D14_ENDPT_MAXPKTSIZE.VALUE = 0x0002;

   //Bulk In MaxPacketSize Endpoint
   D14_Cntrl_Reg.D14_ENDPT_INDEX = 5;
   D14_Cntrl_Reg.D14_ENDPT_MAXPKTSIZE.VALUE = 0x0002;

   //Bulk Out Endpoint Type
D14_Cntrl_Reg.D14_ENDPT_INDEX = 4;
D14_Cntrl_Reg.D14_ENDPT_TYPE.VALUE = 0x1600;

//Bulk In Endpoint Type
D14_Cntrl_Reg.D14_ENDPT_INDEX = 5;
   D14_Cntrl_Reg.D14_ENDPT_TYPE.VALUE = 0x1600;

//enable FIFO
   D14_Cntrl_Reg.D14_ENDPT_INDEX = 4;
   D14_Cntrl_Reg.D14_ENDPT_TYPE.VALUE |= 0x0800;

   //enable FIFO
   D14_Cntrl_Reg.D14_ENDPT_INDEX = 5;
```

```
    D14_Cntrl_Reg.D14_ENDPT_TYPE.VALUE |= 0x0800;
}
```

The initialization of the endpoint is done as shown in Fig 2 to allow the endpoint FIFO RAM to be configured. Ensure that the total endpoints FIFO size does not exceed 8 kB.

3. ISP1582/83 USB enumeration process

The USB enumeration process can be divided into three categories:

- Setup token with data IN stage; see Section 3.1
- Setup token with data OUT stage; see Section 3.2
- Setup token with no data stage; see Section 3.3.

Besides these categories, there is also a stalling procedure for the endpoint (see Section 3.4). Special attention must be given when stalling the endpoint of the ISP1582/83.

The endpoint Index register is used to reference the control endpoint and the setup endpoint. The setup endpoint is 8 bytes in length. It is used for storing the USB peripheral request from the host. The firmware only needs to program the EP0SETUP bit of the ISP1582/83 (see Table 10:) to reference the index to the setup endpoint.

The data from the setup or control endpoint, indexed by the endpoint index is read from the FIFO using the Data Port register; see Table 11:.

Table 11: Data Port register: bit allocation

Bit	15	14	13	12	11	10	9	8
Symbol	DATAPORT[15:8]							
Reset	0	0	0	0	0	0	0	0
Bus reset	0	0	0	0	0	0	0	0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Symbol	DATAPORT[7:0]							
Reset	0	0	0	0	0	0	0	0
Bus reset	0	0	0	0	0	0	0	0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

In the case of writing to the endpoint, the Data Port register is also used for the data transfer to the control IN endpoint. The control endpoint is controlled by the Control Function register (see Table 12:) that allows the firmware to clear the data in the OUT endpoint by using the CLBUF bit. The endpoint FIFO can be validated by either the Buffer Length register or the VENDP bit in the Control Function register. The Buffer Length register (see Table 13:) reflects the amount of data in the IN endpoint. As for the OUT endpoint FIFO, the buffer length is a means for the firmware to autovalidate the FIFO once the value is reached. It is used when the data to be sent in the IN endpoint is a short packet.

Table 12: Control Function register: bit allocation

Bit	7	6	5	4	3	2	1	0
Symbol		reserved		CLBUF	VENDP	DSEN	STATUS	STALL
Reset	-	-	-	0	0	0	0	0
Bus reset	-	-	-	0	0	0	0	0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Table 13: Buffer Length register: bit allocation

Bit	15	14	13	12	11	10	9	8
Symbol	DATACOUNT[15:8]							
Reset	0	0	0	0	0	0	0	0
Bus reset	0	0	0	0	0	0	0	0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Symbol	DATACOUNT[7:0]							
Reset	0	0	0	0	0	0	0	0
Bus reset	0	0	0	0	0	0	0	0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

In the control function, the DSEN bit is for the ISP1582/83 to proceed to the data stage. If the setup token is with a data OUT stage, the firmware must set the DSEN bit to the ISP1582/83 for the device to generate an ACK handshake to the OUT endpoint. This bit also governs the IN endpoint data. An IN endpoint after a setup token with the data IN stage will not be sent to the USB bus even when the VENDP bit is issued. Therefore, the DSEN bit must be set for the ISP1582/83 to proceed to the status stage of the setup. The status stage is achieved by setting the STATUS bit in the Control Function register. On setting the STATUS bit, the endpoint will generate a zero-length packet to the IN token and send an ACK for the OUT token. The status stage will not generate the control IN endpoint interrupt.

3.1 Setup token with data IN stage

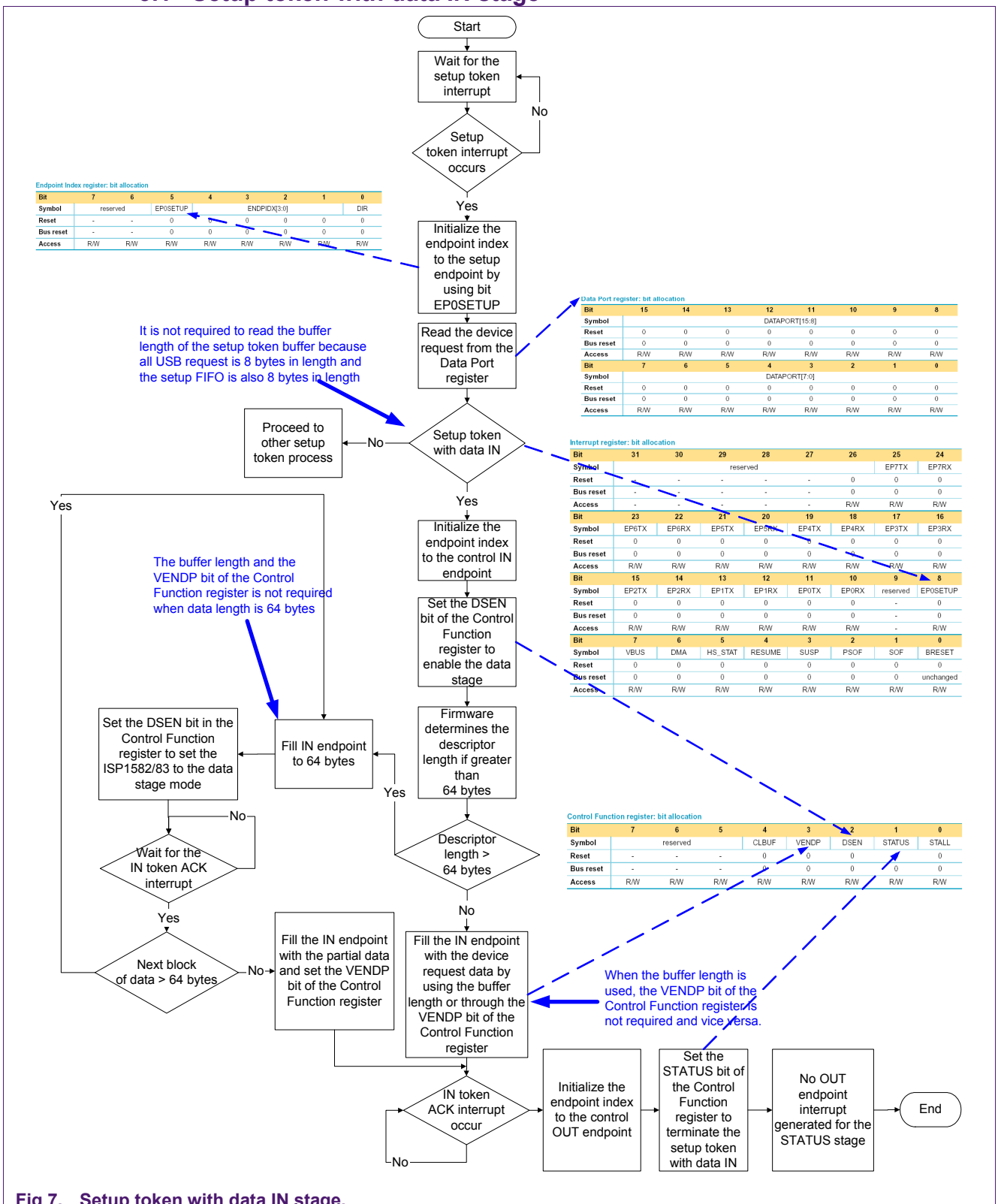


Fig 7. Setup token with data IN stage.

3.2 Setup token with data OUT stage

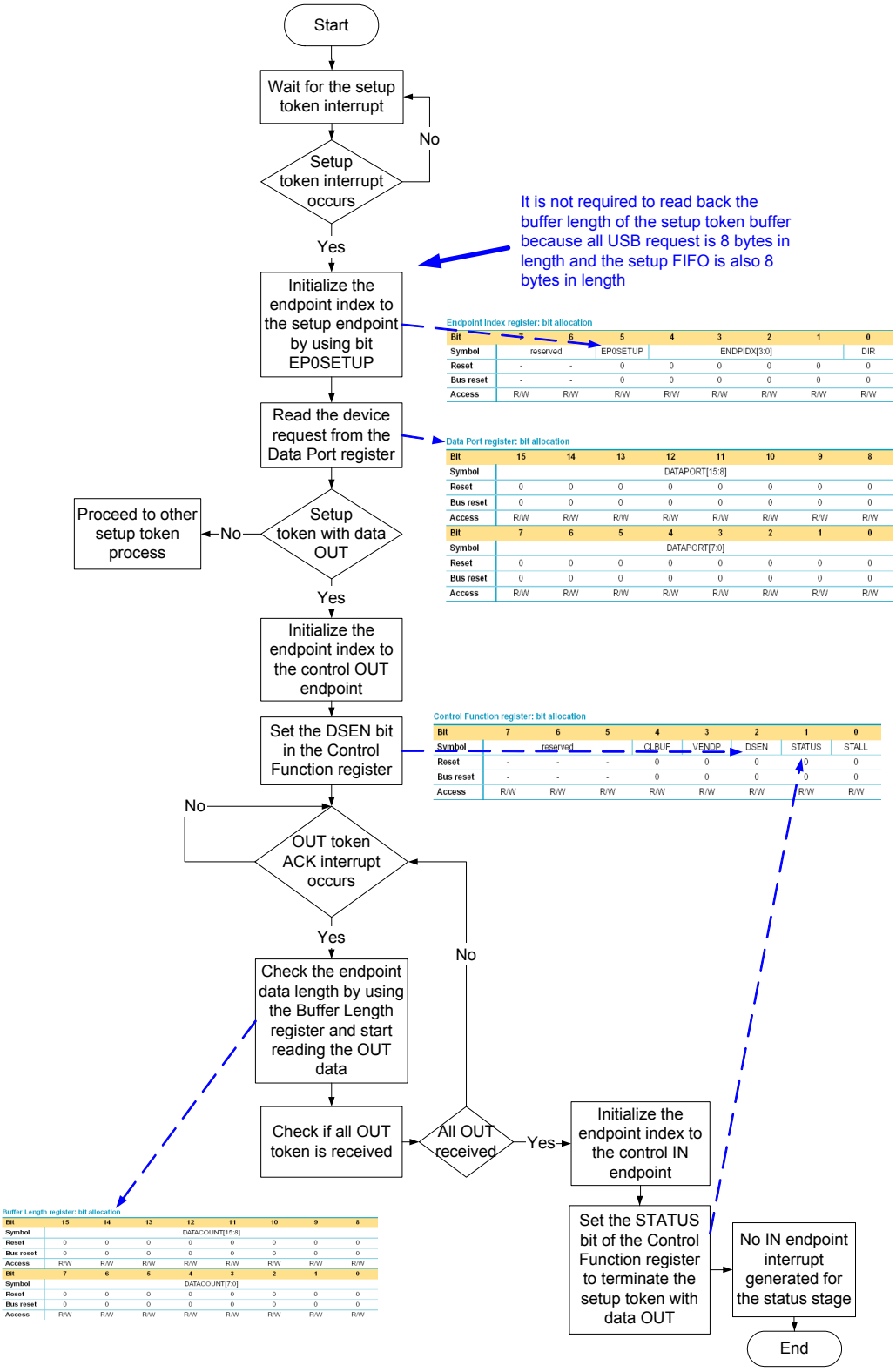


Fig 8. Setup token with data OUT stage.

3.3 Setup token with no data stage

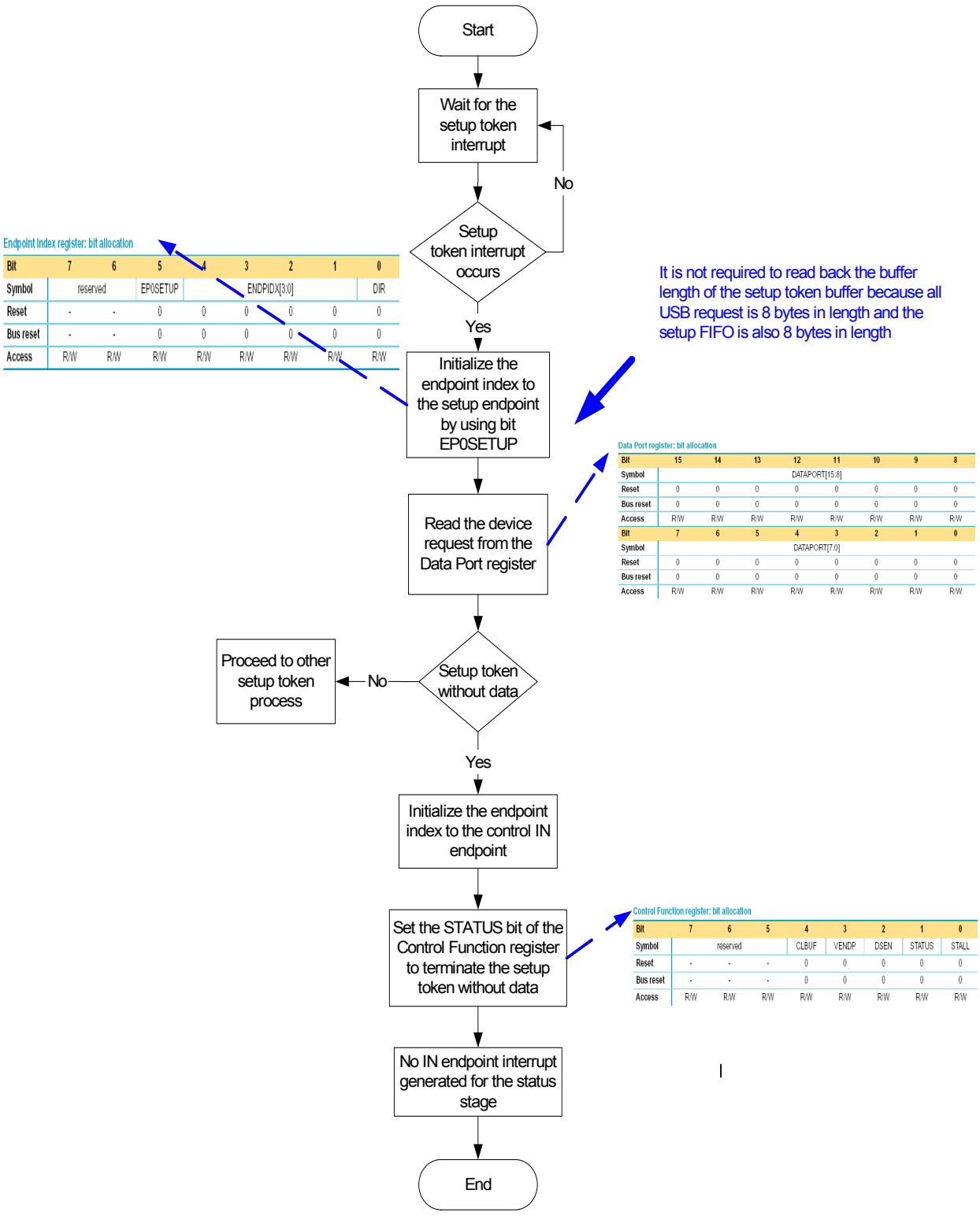


Fig 9. Setup token with no data stage.

3.4 Stalling setup token

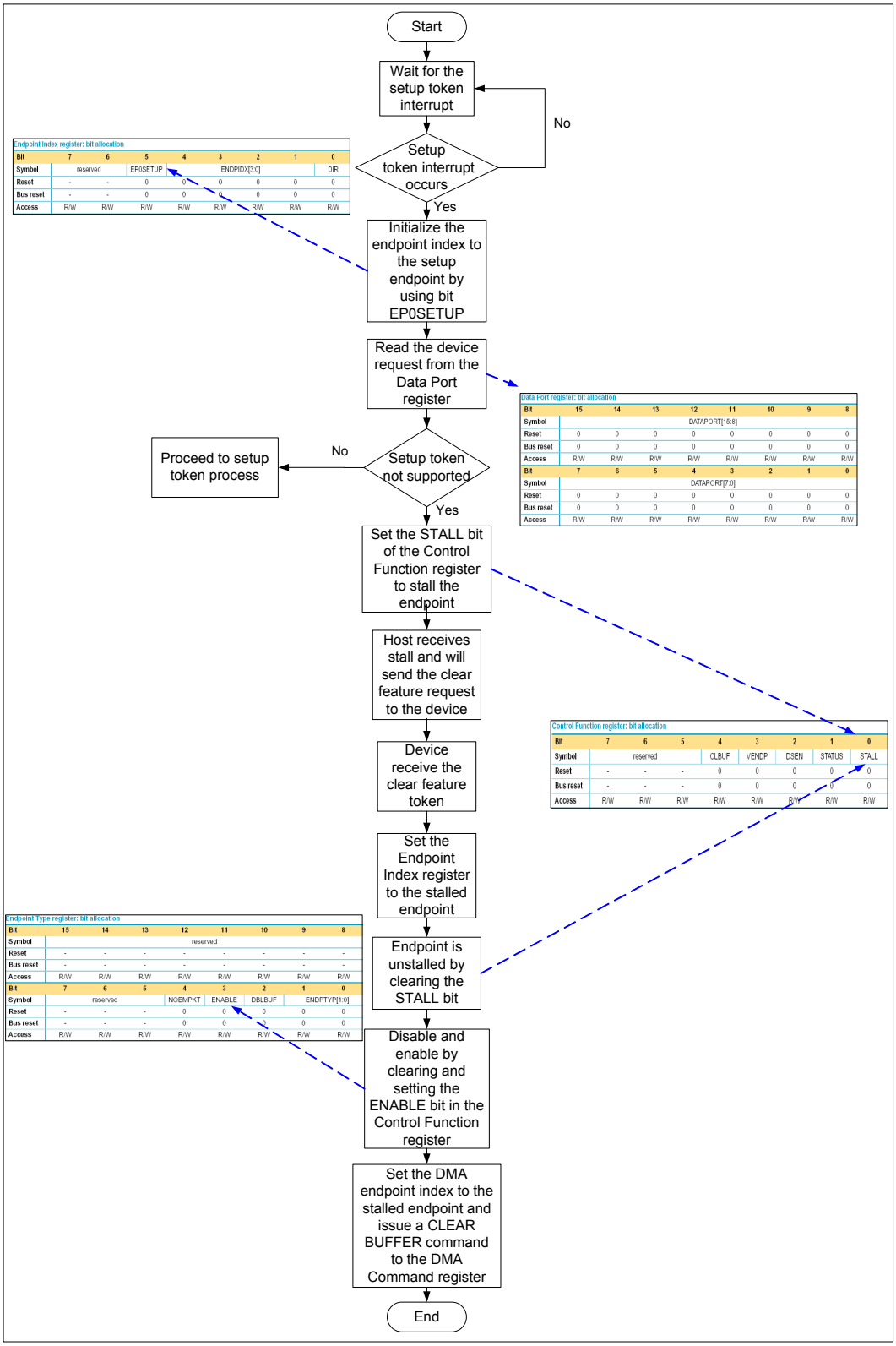


Fig 10. Stalling setup token.

4. Mass storage application

After successful enumeration, the respective device drivers come into play. For a mass storage application, the firmware must conform to the USB Mass Storage Class Bulk-Only Transport Specification and the ATA/ATAPI protocol. The host sends 31 bytes Command Block Wrapper (CBW). The ATAPI command is embedded in the CBW.

In the case of an ATAPI device, for each CBW received the ATAPI packet command is issued to the device, followed by the ATAPI command in the CBW.

In the case of an ATA device, the ATAPI command in the CBW must be translated into ATA command, before issuing to the device. The ATAPI command phase is followed by an optional data phase, which is followed by a status phase. If the command is successfully completed, the firmware has to report good status in the status field of the 13-byte Command Status Wrapper (CSW).

See Table 14: and Table 15: for CBW and CSW, respectively.

Table 14: Command Block Wrapper

Bit / Byte	7	6	5	4	3	2	1	0
30 to 15 (1Eh to 0Fh)	*CBWCB*							
14 (0Eh)	reserved (0)				*bCBWCBLength*			
13 (0Dh)	reserved (0)				*bCBWLUN*			
12 (0Ch)	*bmCBWFlags*							
11 to 8 (0Bh to 08h)	*dCBWDataTransferLength*							
7 to 4	*dCBWTag*							
3 to 0	*dCBWSignature*							

Table 15: Command Status Wrapper

Bit / Byte	7	6	5	4	3	2	1	0
12 (0Ch)	*bCSWStatus*							
11 to 8 (0Bh to 08h)	*dCSWDataResidue*							
7 to 4	*dCSWTag*							
3 to 0	*dCSWSignature*							

4.1 Mass storage protocol

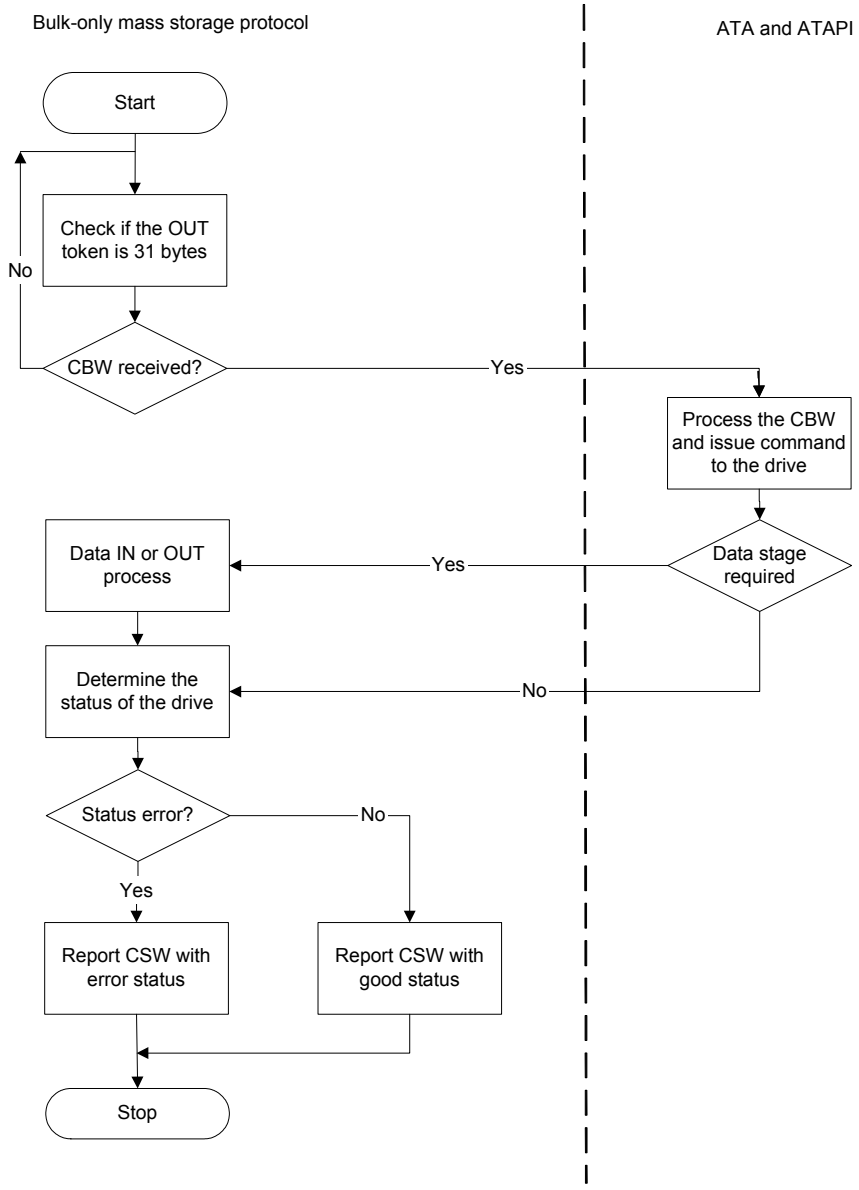


Fig 11. Mass storage protocol.

4.1.1 Extracting Command Block Wrapper

1. Initialize the Endpoint Index register to the respective OUT endpoint.
2. Read the Buffer Length register. A buffer length of 31 indicates a command packet is in the buffer.
3. Read the Data Port register to extract the CBW.
4. Check for valid CBW signature in the command block.

If the CBW signature is valid, the command is issued to the ATA or ATAPI device.

For an invalid command block, the respective OUT endpoint is stalled.

4.1.2 Handling CBW for ATA and ATAPI device

1. Initialize the drive select register (1F6) to the master or slave, based on the device present.
2. Initialize the Error or Feature register to 1 for a DMA transfer.
3. Initialize the ATAPI Byte Count LSB and ATAPI Byte Count MSB registers to the number of bytes to be transferred as specified in the CBW.
4. The ATAPI packet command is issued to the device by writing the Command or Status register.
5. Issue the 12-byte command block in CBW to the Task File Data register.
6. Check for the next phase.

If CBW.dCBWDataTransferLength is zero, then the next phase is status. If CBW.dCBWDataTransferLength is non-zero and USB_CBW.bmCBWFlags is 0x80, then the next phase is read. If CBW.dCBWDataTransferLength is non-zero and USB_CBW.bmCBWFlags is not equal to 0x80, then the next phase is write.

7. The read or write phase is followed by the status phase.

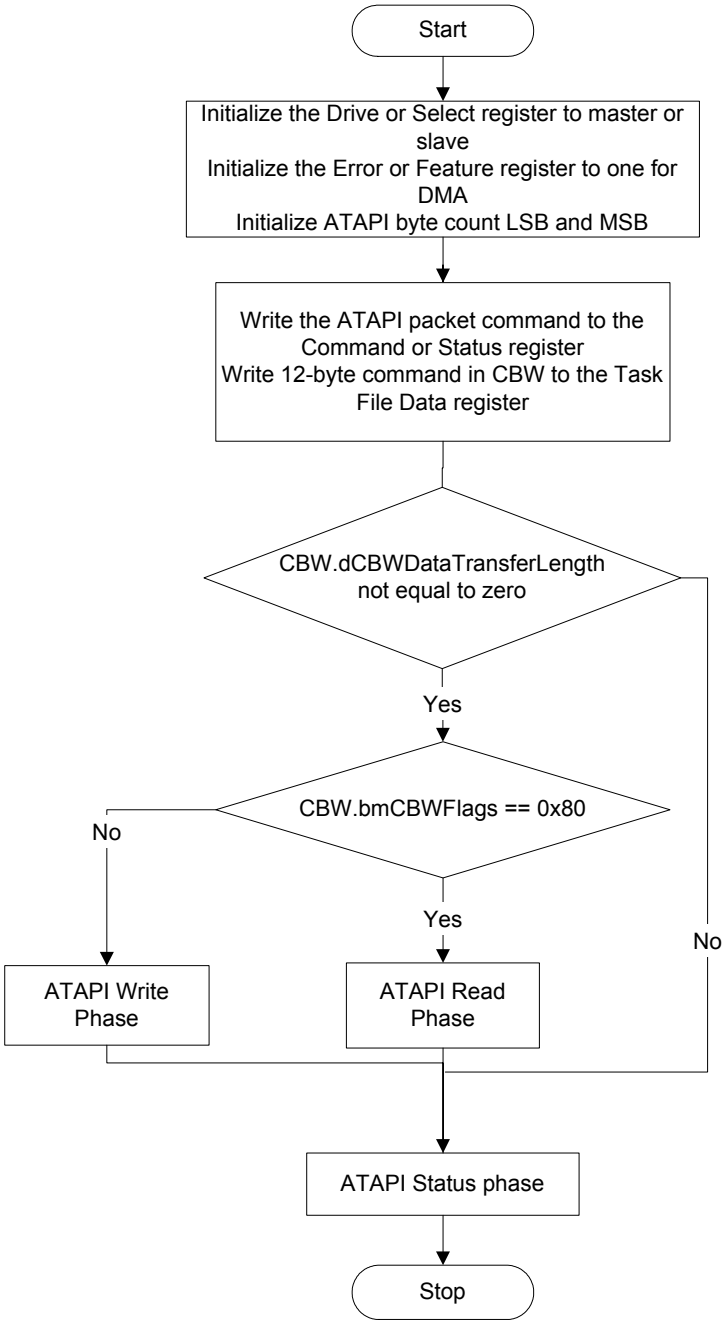


Fig 12. Handling CBW for ATAPI device.

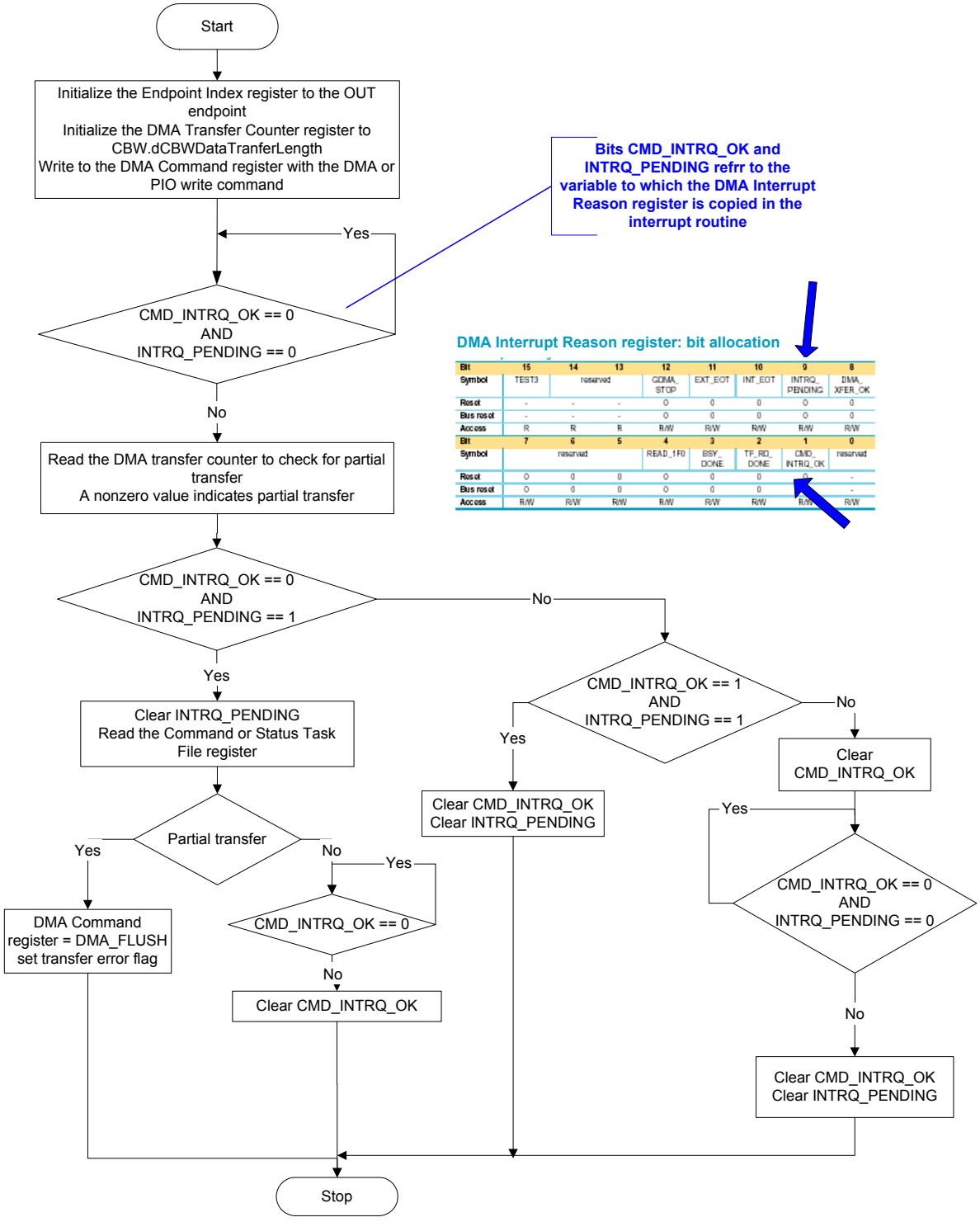


Fig 13. ATAPI write phase.

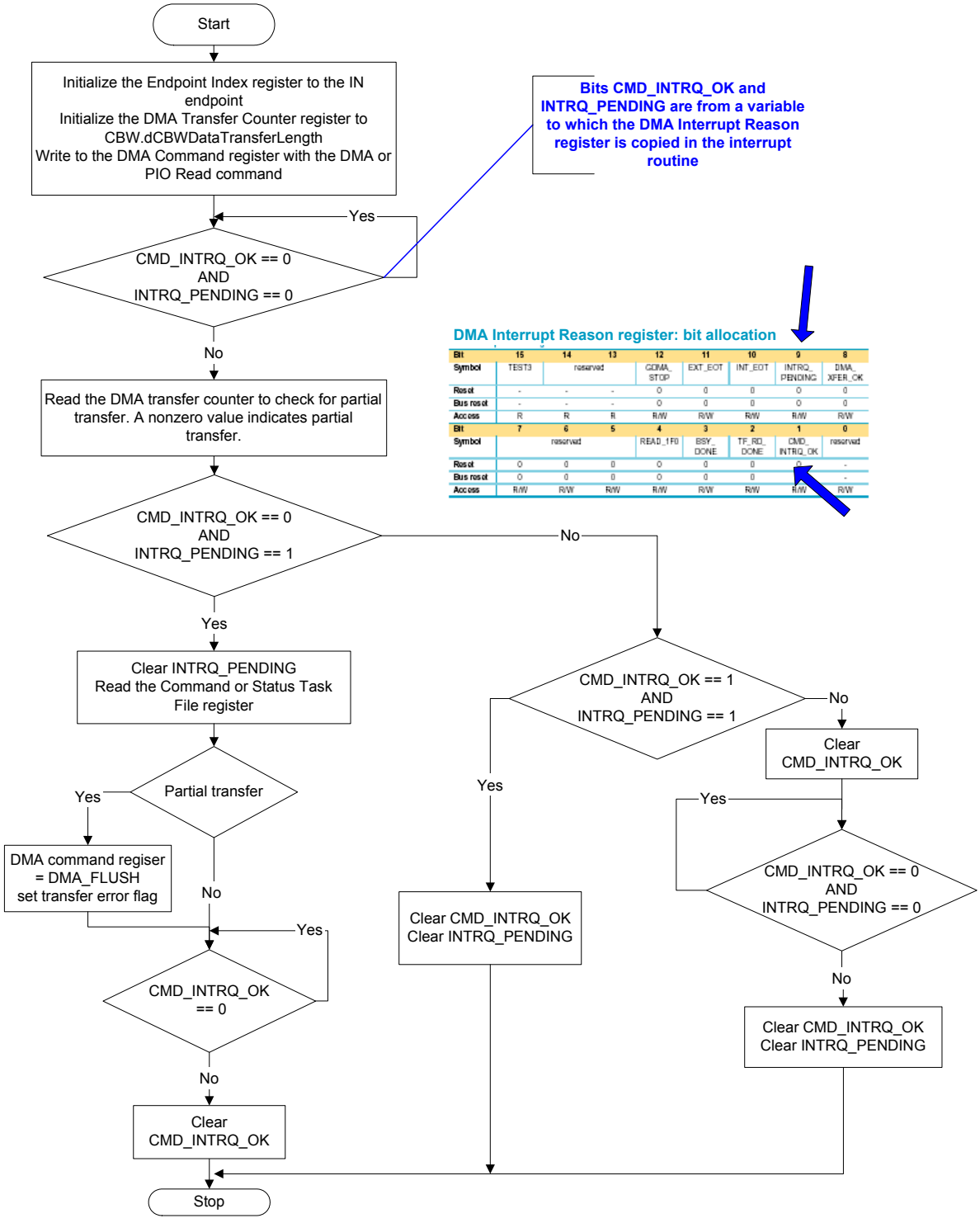


Fig 14. ATAPI read phase.

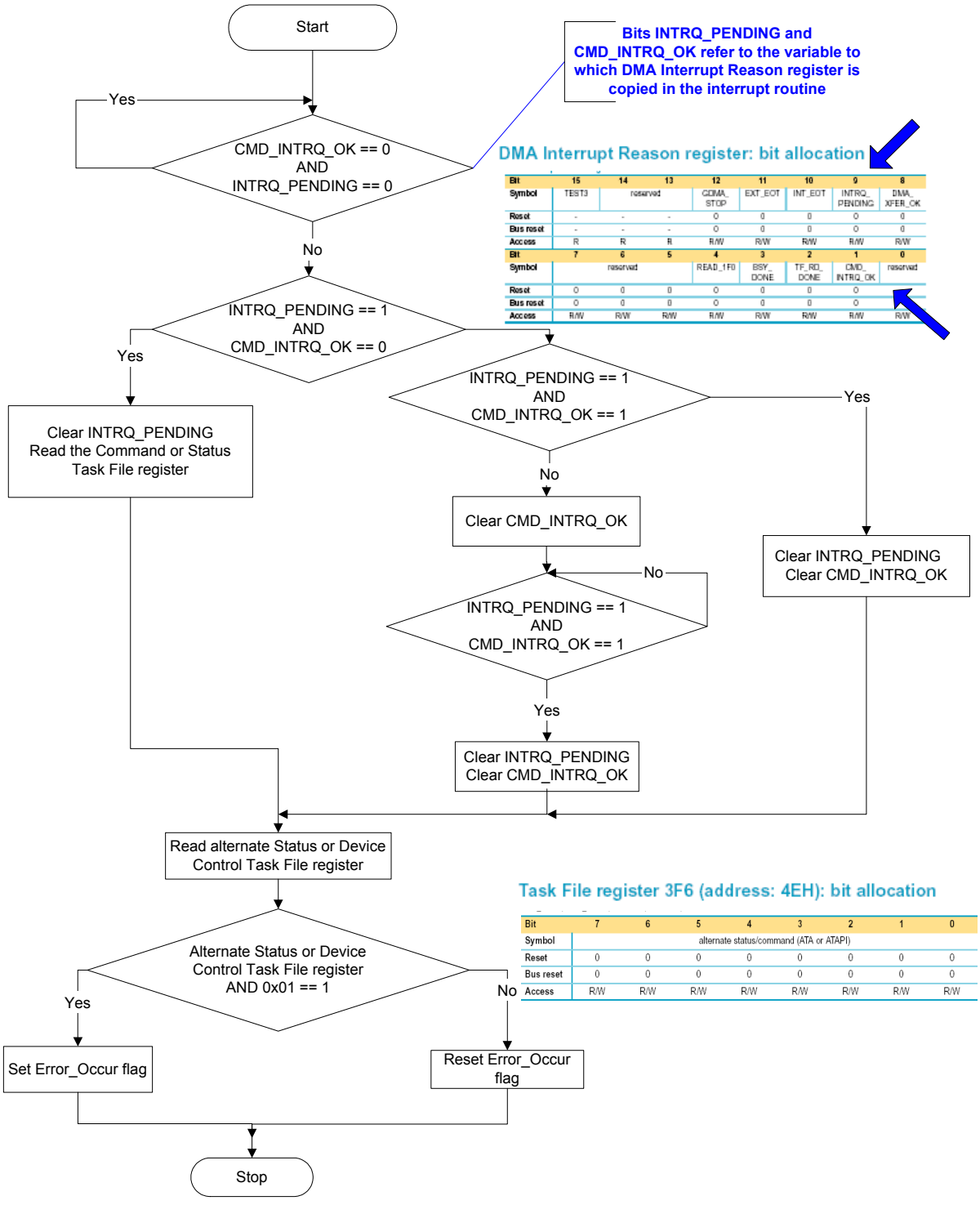


Fig 15. ATAPI status phase.

4.1.3 Handling invalid CBW

A command block is invalid if it has an invalid CBW signature.

1. Initialize the Endpoint Index register to the respective OUT endpoint.
2. Set the STALL bit in the Control Function register. When the host receives a stall, it will send a clear feature command. Wait for the setup token interrupt for the clear feature.
3. Initialize the Endpoint Index register to the respective OUT endpoint, and clear the STALL bit in the Control Function register.
4. Set and reset the ENABLE bit in the Endpoint Type register. This will reset the data toggle.
5. Initialize the Endpoint Index register to 1 and set the STATUS bit in the Control Function register
6. Wait till the EP0Tx interrupt is received.
7. Wait for the EP0SETUP interrupt, for the bulk-only mass storage reset command.
8. Initialize the Endpoint Index register, by setting the EP0SETUP bit in the Endpoint Index register.
9. Read the Setup buffer and reinitialize the ATA or ATAPI device, for a valid mass storage reset command.

4.1.4 Handling error

If the Error_Occur flag or transfer_error flag is set, error must be handled, by stalling the respective endpoint involved in the transfer.

- For the data OUT transfer, the respective OUT endpoint is stalled.
- For the data IN transfer, the respective IN endpoint is stalled.
- Clear the Error_Occur and transfer_error flag.

4.1.5 Handling CBW for an ATA device

CBW contains the ATAPI command embedded in the command block.

For an ATA device, the ATAPI packet command must be translated into an ATA command, before issuing to the device. The CBWCDB[0] field in the CBW contain the ATAPI command and the remaining field contains the command parameters. Based on the ATAPI command, the corresponding ATA command is selected and issued to the device.

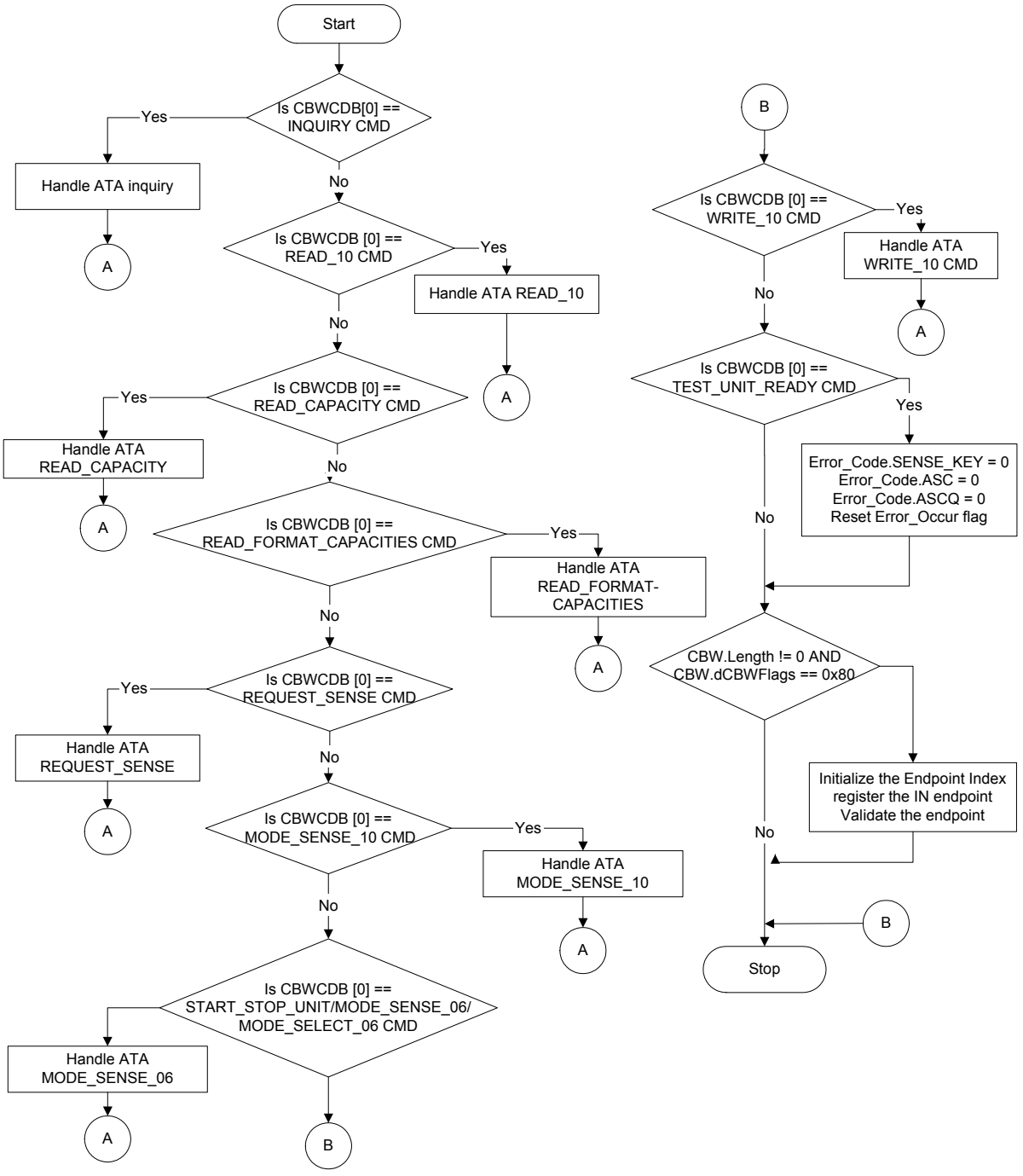
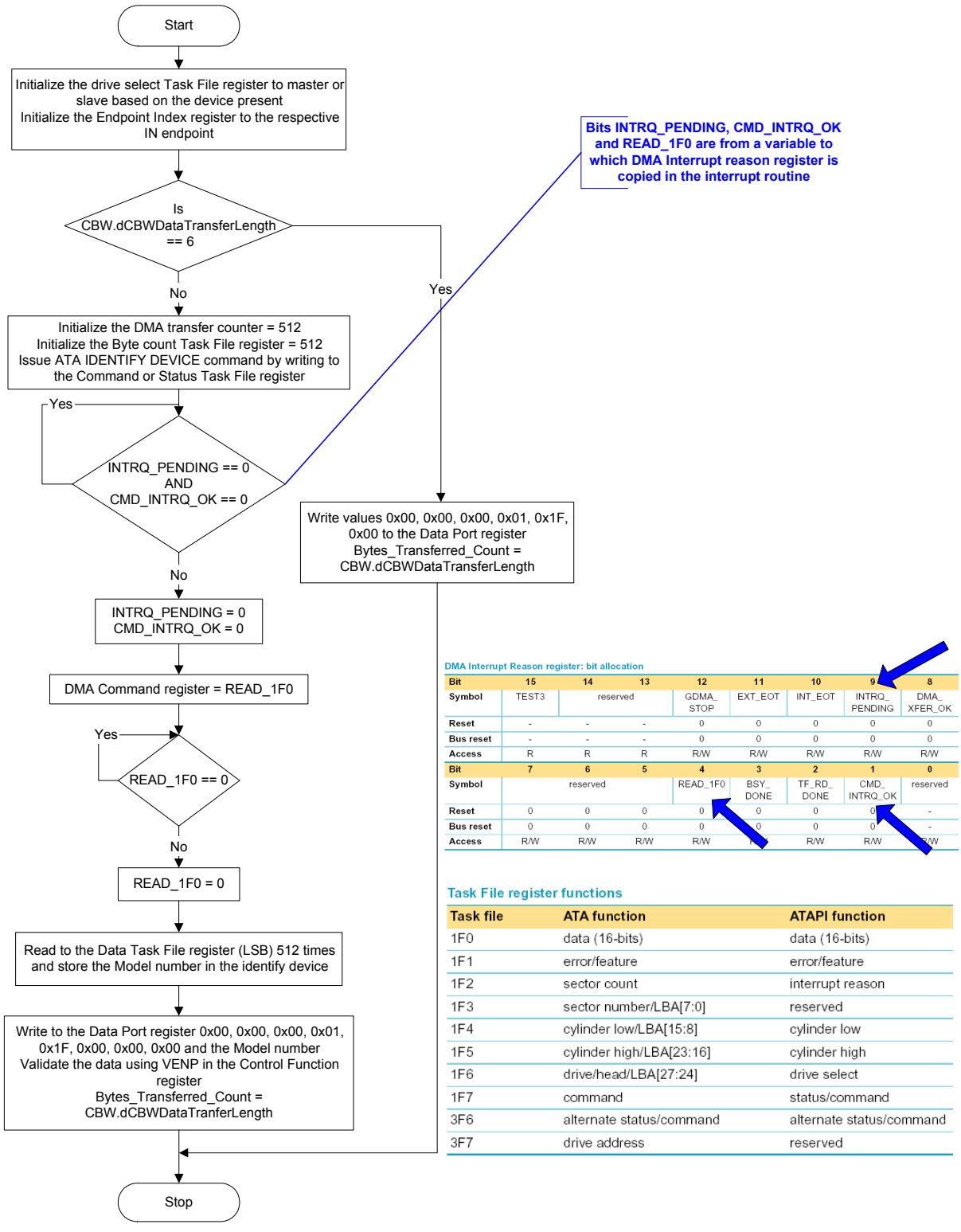


Fig 16. Handling ATA Command Translator.



DMA Interrupt Reason register: bit allocation

Bit	15	14	13	12	11	10	9	8
Symbol	TEST3	reserved		GDMA_ STOP	EXT_EOT	INT_EOT	INTRQ_ PENDING	DMA_ XFER_OK
Reset	-	-	-	0	0	0	0	0
Bus reset	-	-	-	0	0	0	0	0
Access	R	R	R	R/W	R/W	R/W	R/W	R/W
Bit	7	6	5	4	3	2	1	0
Symbol	reserved			READ_1F0	BSY_ DONE	TF_RD_ DONE	CMD_ INTRQ_OK	reserved
Reset	0	0	0	0	0	0	0	-
Bus reset	0	0	0	0	0	0	0	-
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Task File register functions

Task file	ATA function	ATAPI function
1F0	data (16-bits)	data (16-bits)
1F1	error/feature	error/feature
1F2	sector count	interrupt reason
1F3	sector number/LBA[7:0]	reserved
1F4	cylinder low/LBA[15:8]	cylinder low
1F5	cylinder high/LBA[23:16]	cylinder high
1F6	drive/head/LBA[27:24]	drive select
1F7	command	status/command
3F6	alternate status/command	alternate status/command
3F7	drive address	reserved

Fig 17. Handling ATA_INQUIRY.

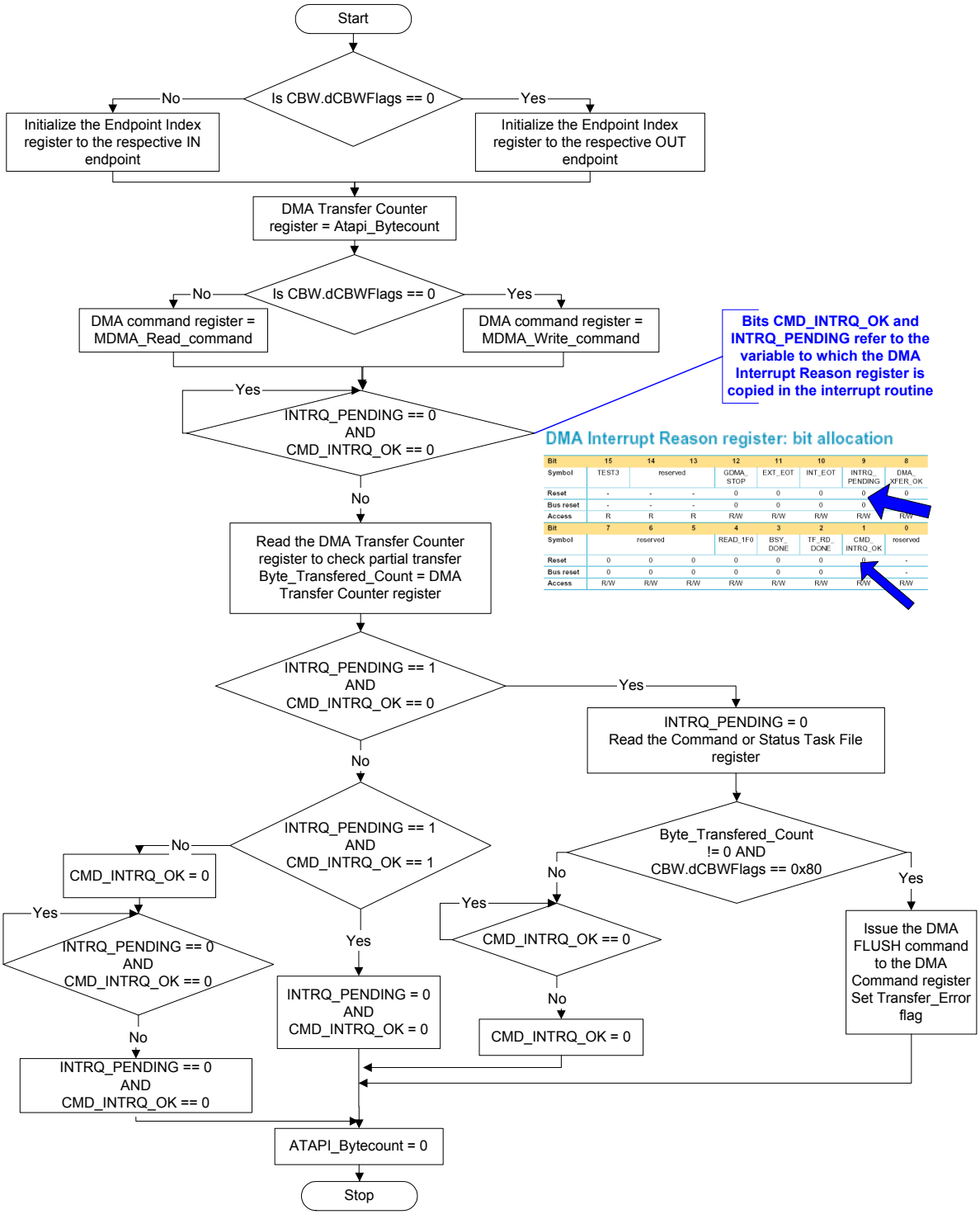
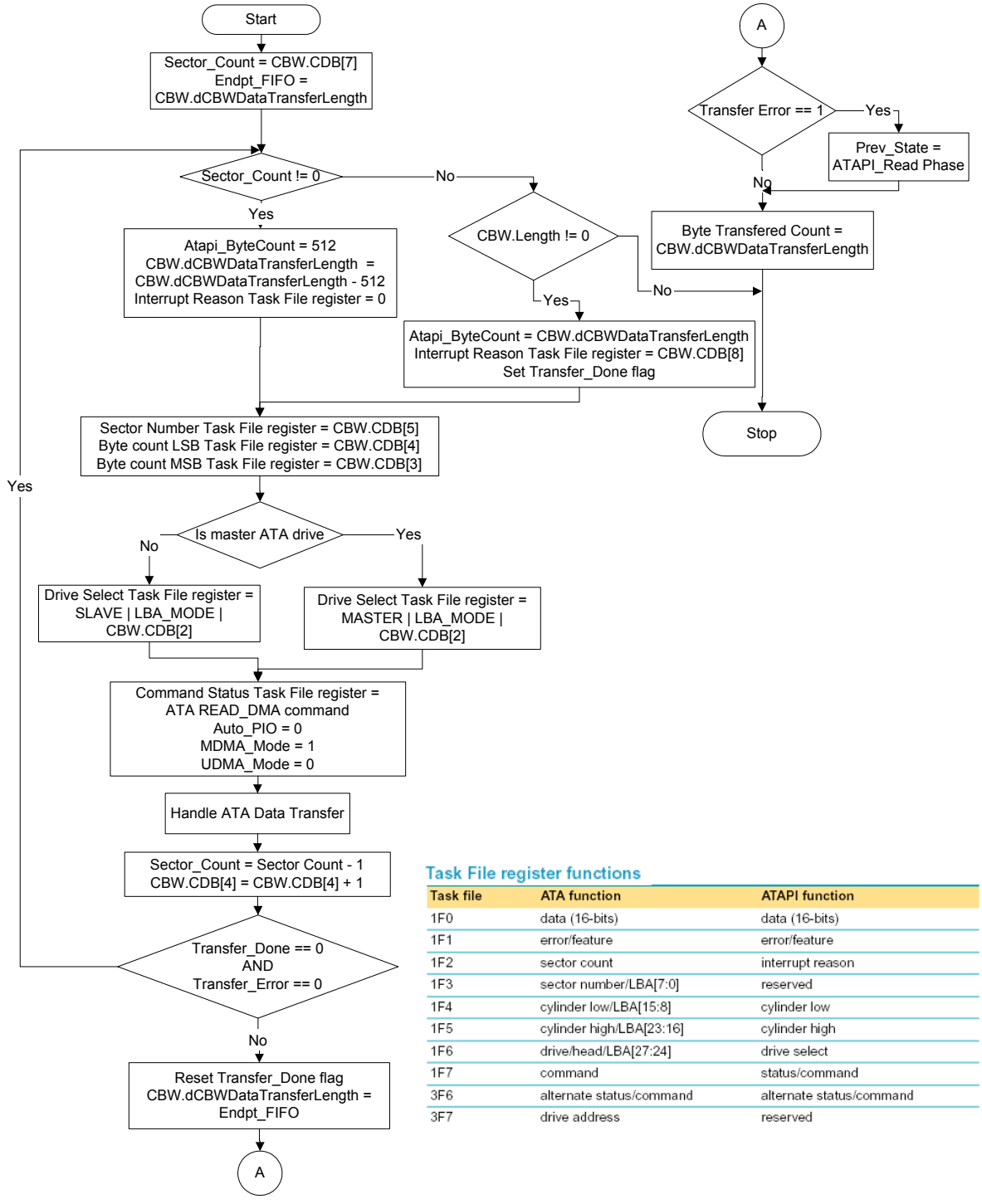


Fig 18. Handling ATA_DATA_TRANSFER.



Task File register functions

Task file	ATA function	ATAPI function
1F0	data (16-bits)	data (16-bits)
1F1	error/feature	error/feature
1F2	sector count	interrupt reason
1F3	sector number/LBA[7:0]	reserved
1F4	cylinder low/LBA[15:8]	cylinder low
1F5	cylinder high/LBA[23:16]	cylinder high
1F6	drive/head/LBA[27:24]	drive select
1F7	command	status/command
3F6	alternate status/command	alternate status/command
3F7	drive address	reserved

Fig 19. Handling ATA_READ_10.

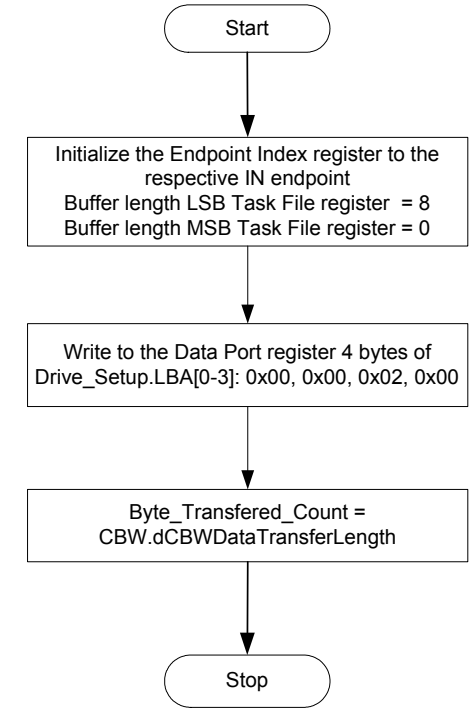


Fig 20. Handling ATA READ CAPACITY.

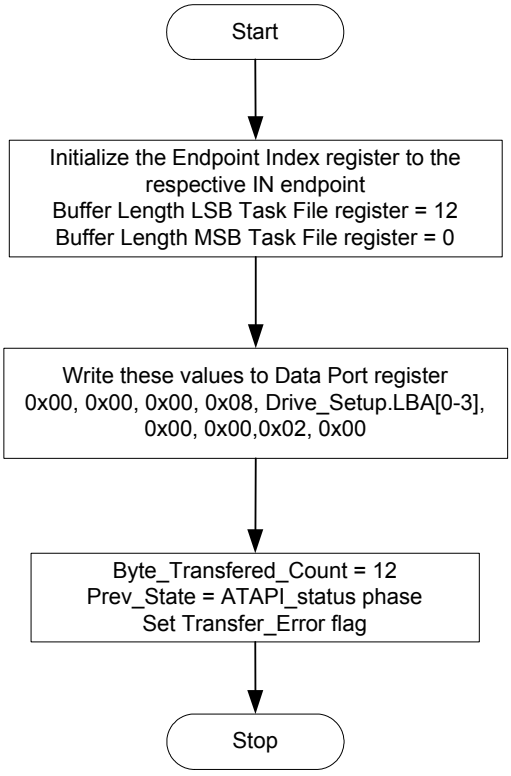
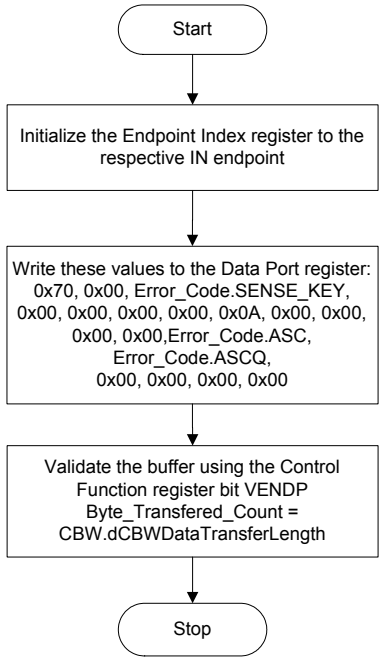


Fig 21. Handling ATA READ_FORMAT_CAPACITIES.



In the 8-bit mode, write to the LSB and then the MSB of the Data Port register

Data Port register: bit allocation

Bit	15	14	13	12	11	10	9	8
Symbol	DATAPORT[15:8]							
Reset	0	0	0	0	0	0	0	0
Bus reset	0	0	0	0	0	0	0	0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Bit	7	6	5	4	3	2	1	0
Symbol	DATAPORT[7:0]							
Reset	0	0	0	0	0	0	0	0
Bus reset	0	0	0	0	0	0	0	0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Control Function register: bit allocation

Bit	7	6	5	4	3	2	1	0
Symbol	reserved			CLBUF	VENDP	DSEN	STATUS	STALL
Reset	-	-	-	0	0	0	0	0
Bus reset	-	-	-	0	0	0	0	0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Fig 22. Handling ATA REQUEST_SENSE.

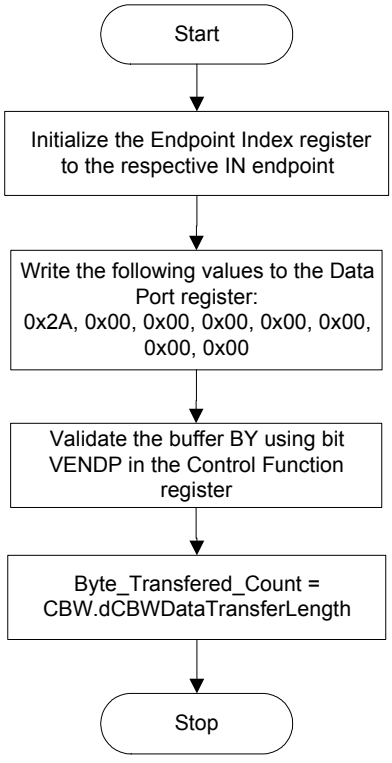


Fig 23. Handling ATA MODE_SENSE_10.

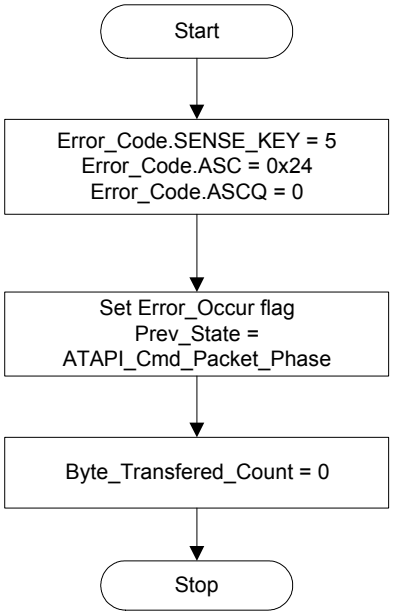
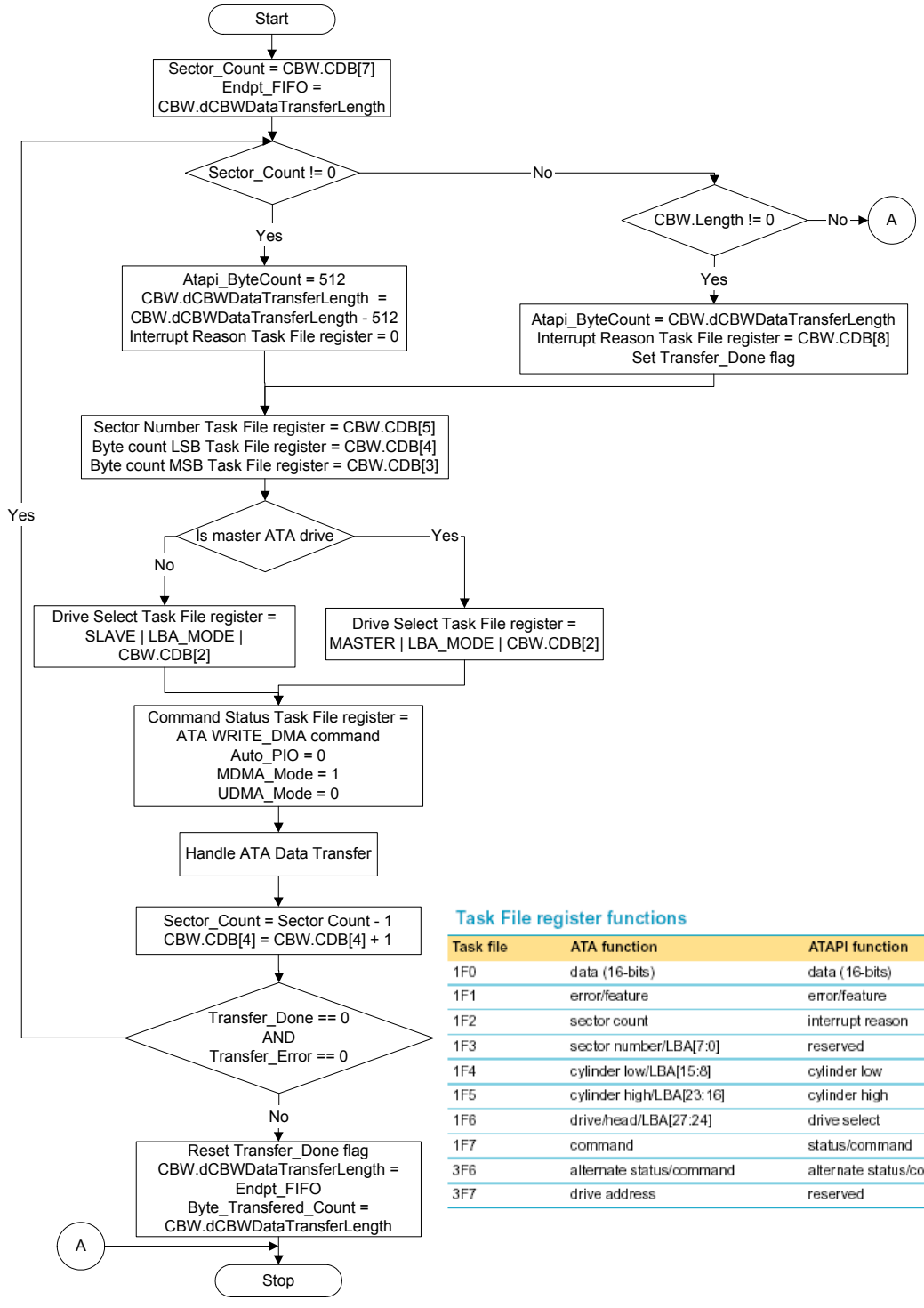


Fig 24. Handling ATA MODE_SENSE_06.



Task File register functions

Task file	ATA function	ATAPI function
1F0	data (16-bits)	data (16-bits)
1F1	error/feature	error/feature
1F2	sector count	interrupt reason
1F3	sector number/LBA[7:0]	reserved
1F4	cylinder low/LBA[15:8]	cylinder low
1F5	cylinder high/LBA[23:16]	cylinder high
1F6	drive/head/LBA[27:24]	drive select
1F7	command	status/command
3F6	alternate status/command	alternate status/command
3F7	drive address	reserved

Fig 25. Handling ATA WRITE_10.

5. GDMA application

The GDMA application supports the GDMA slave and the PIO modes. These modes are selected based on the vendor-specific command issued by the host. The host sends an 8-byte setup packet, followed by a 6-byte vendor-specific information. The number of bytes to be transferred, the direction of transfer, and the mode of transfer are embedded in the vendor-specific command.

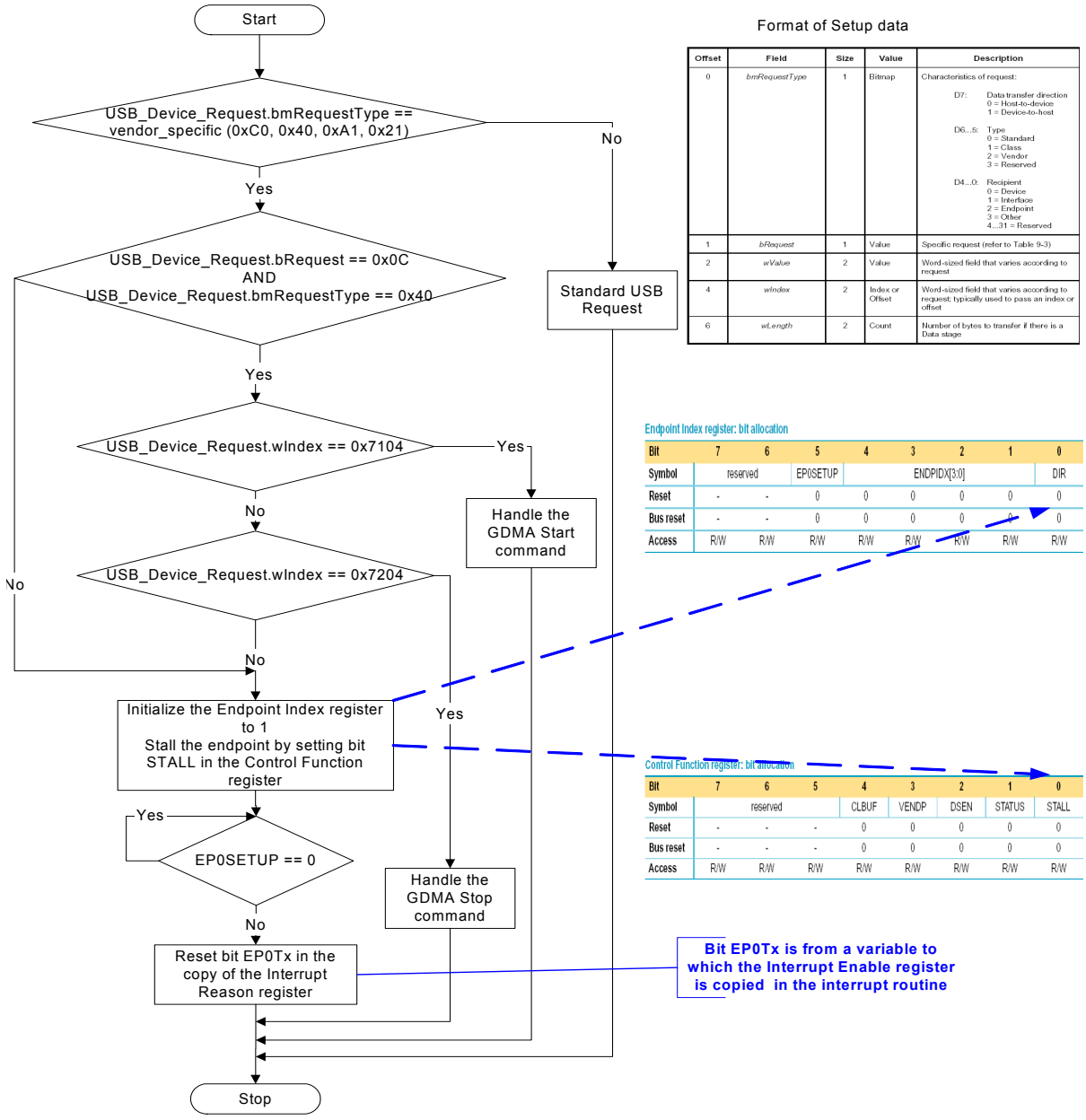


Fig 26. GDMA flowchart.

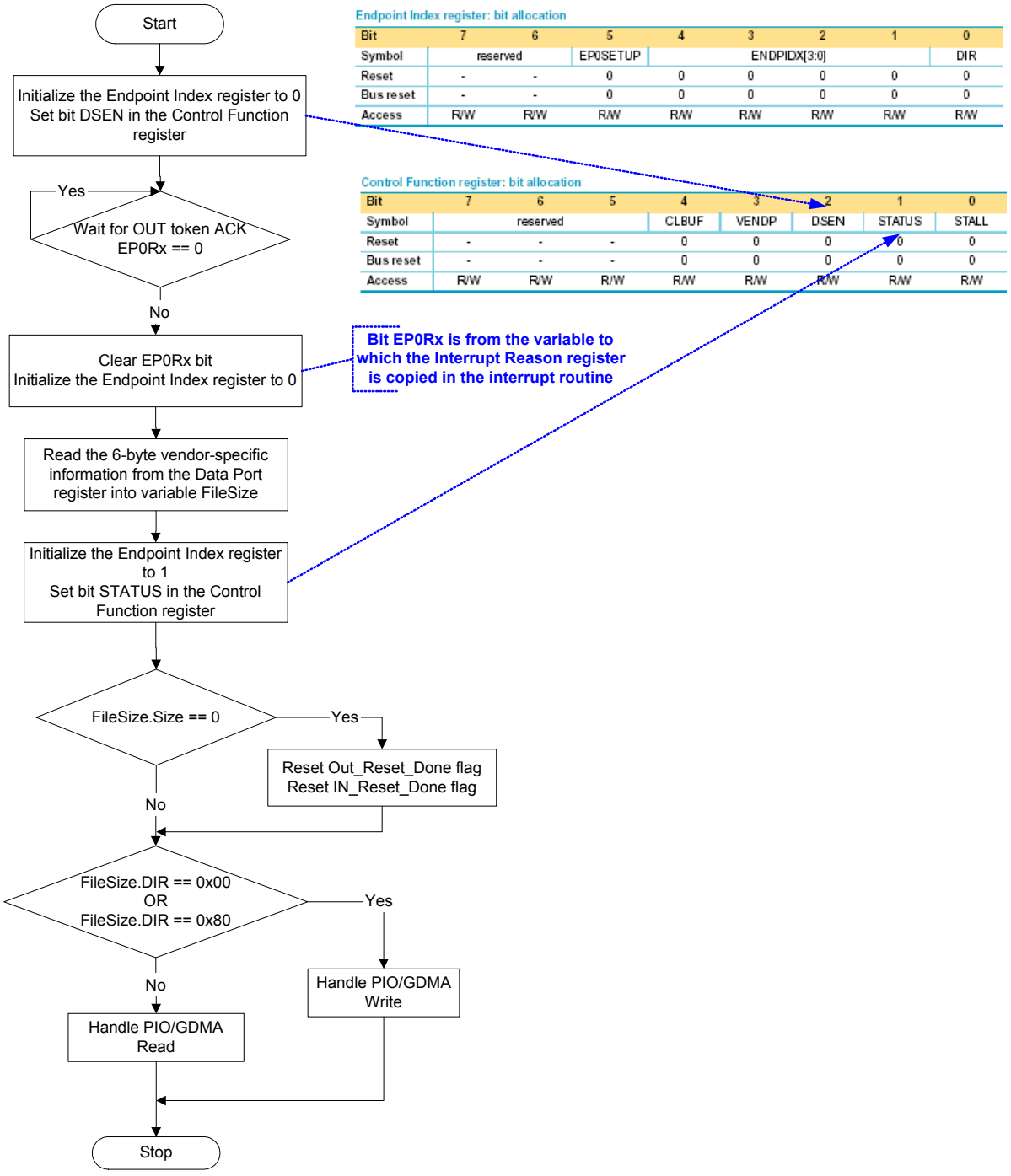
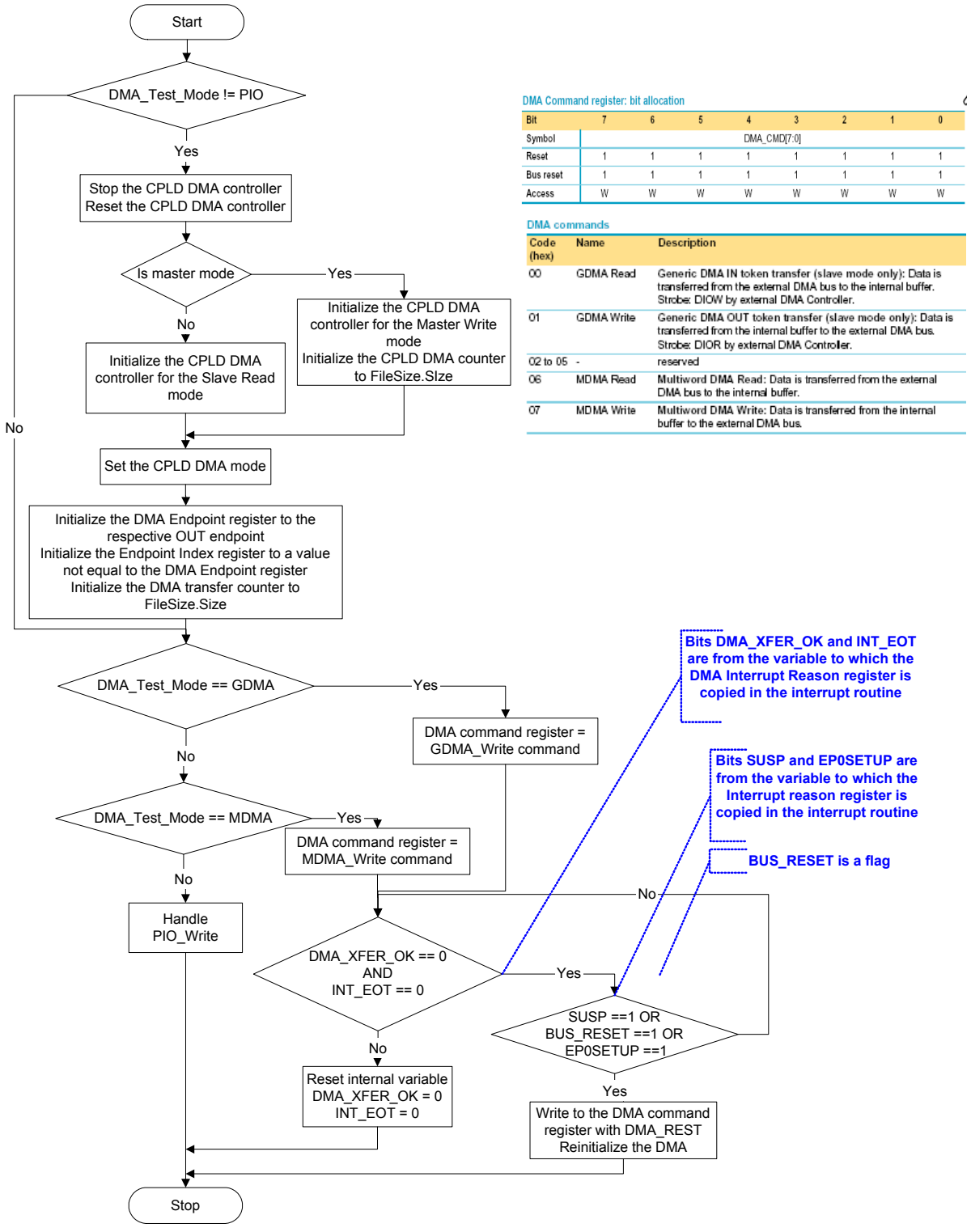


Fig 27. Handling GDMA start command.



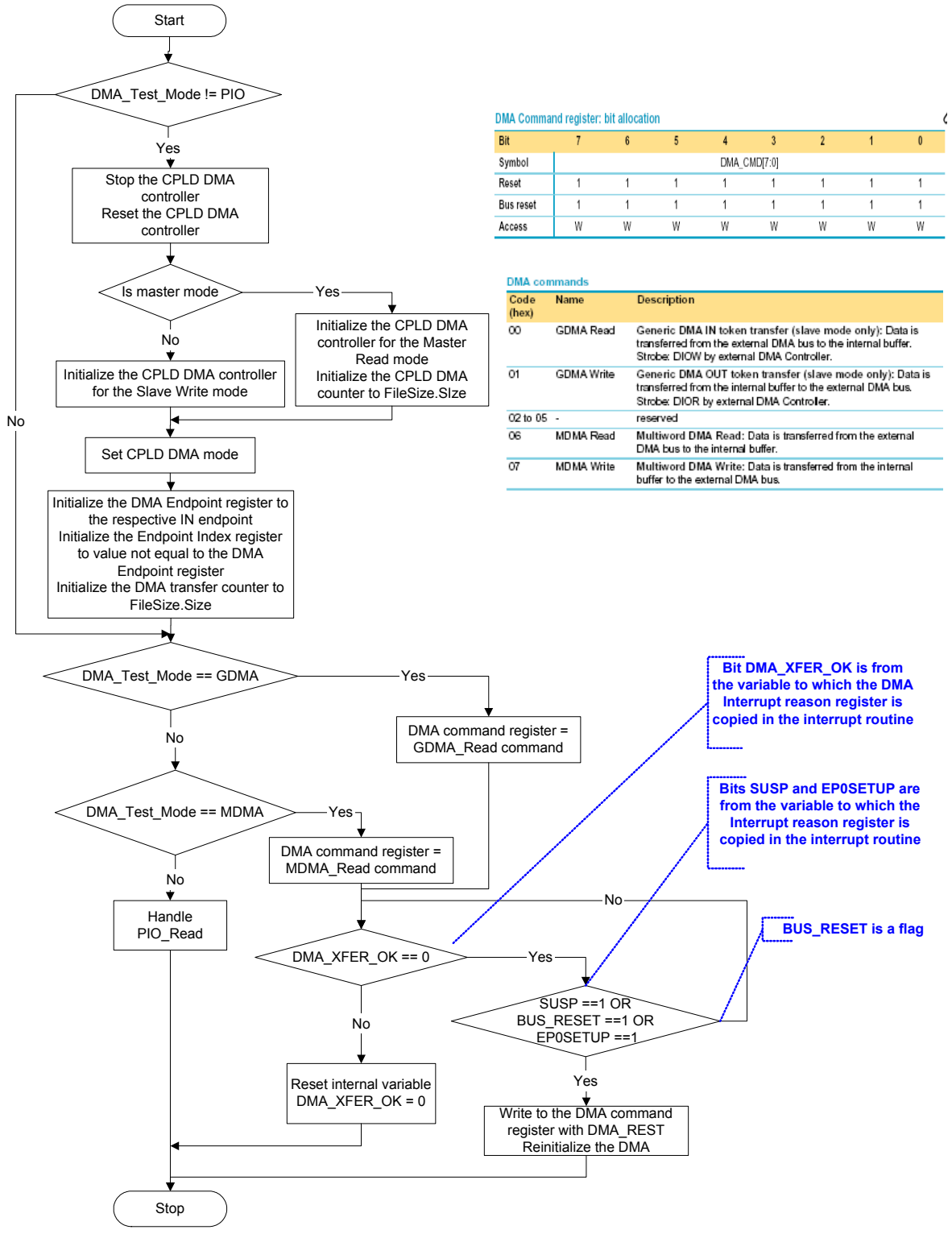
DMA Command register: bit allocation

Bit	7	6	5	4	3	2	1	0
Symbol	DMA_CMD[7:0]							
Reset	1	1	1	1	1	1	1	1
Bus reset	1	1	1	1	1	1	1	1
Access	W	W	W	W	W	W	W	W

DMA commands

Code (hex)	Name	Description
00	GDMA Read	Generic DMA IN token transfer (slave mode only): Data is transferred from the external DMA bus to the internal buffer. Strobe: DIOW by external DMA Controller.
01	GDMA Write	Generic DMA OUT token transfer (slave mode only): Data is transferred from the internal buffer to the external DMA bus. Strobe: DIOR by external DMA Controller.
02 to 05	-	reserved
06	MDMA Read	Multiword DMA Read: Data is transferred from the external DMA bus to the internal buffer.
07	MDMA Write	Multiword DMA Write: Data is transferred from the internal buffer to the external DMA bus.

Fig 28. Handling PIO or GDMA write.



DMA Command register: bit allocation

Bit	7	6	5	4	3	2	1	0
Symbol	DMA_CMD[7:0]							
Reset	1	1	1	1	1	1	1	1
Bus reset	1	1	1	1	1	1	1	1
Access	W	W	W	W	W	W	W	W

DMA commands

Code (hex)	Name	Description
00	GDMA Read	Generic DMA IN token transfer (slave mode only): Data is transferred from the external DMA bus to the internal buffer. Strobe: DIOW by external DMA Controller.
01	GDMA Write	Generic DMA OUT token transfer (slave mode only): Data is transferred from the internal buffer to the external DMA bus. Strobe: DIOR by external DMA Controller.
02 to 05	-	reserved
06	MDMA Read	Multiword DMA Read: Data is transferred from the external DMA bus to the internal buffer.
07	MDMA Write	Multiword DMA Write: Data is transferred from the internal buffer to the external DMA bus.

Fig 29. Handling PIO or GDMA read.

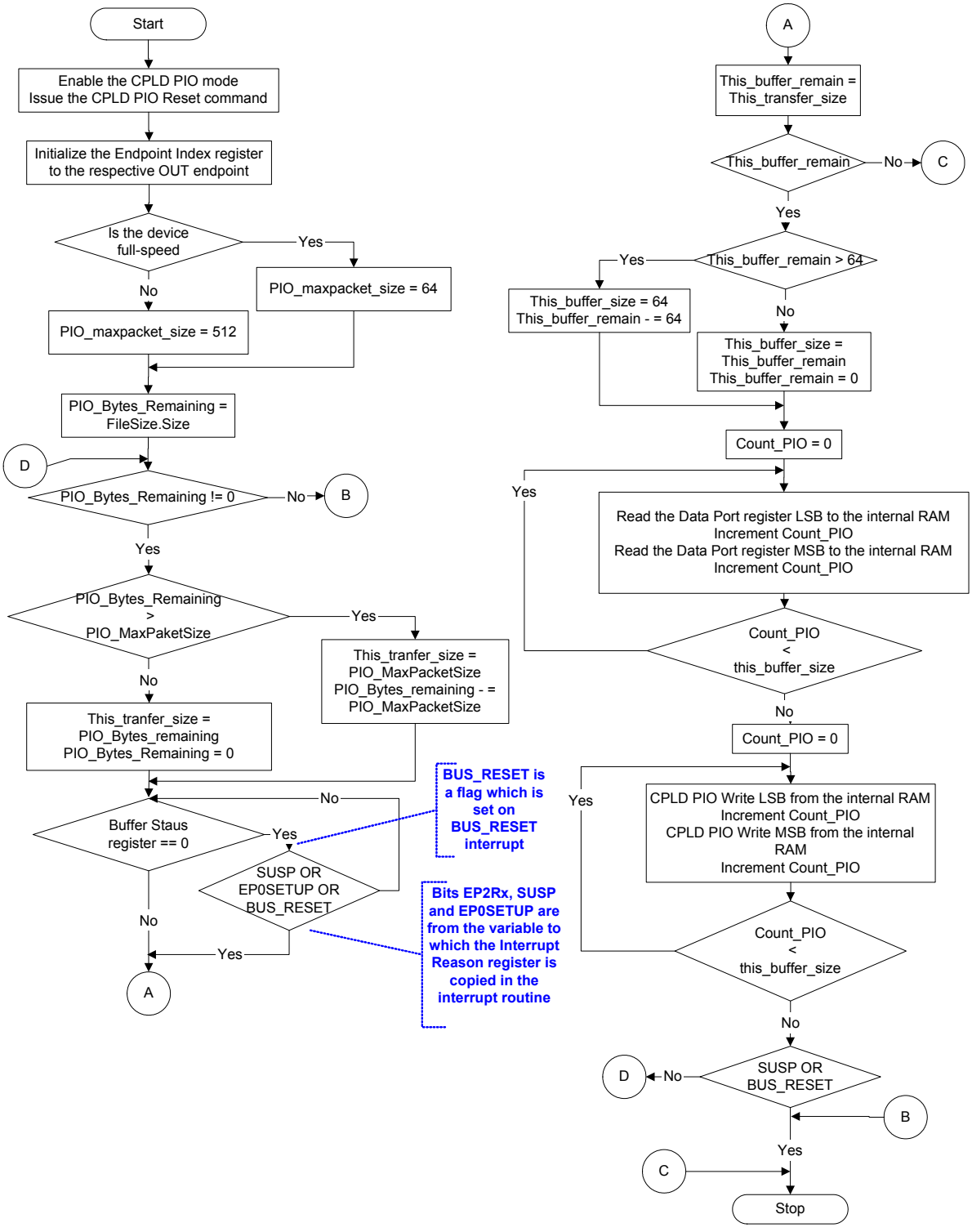


Fig 30. Handling PIO write.

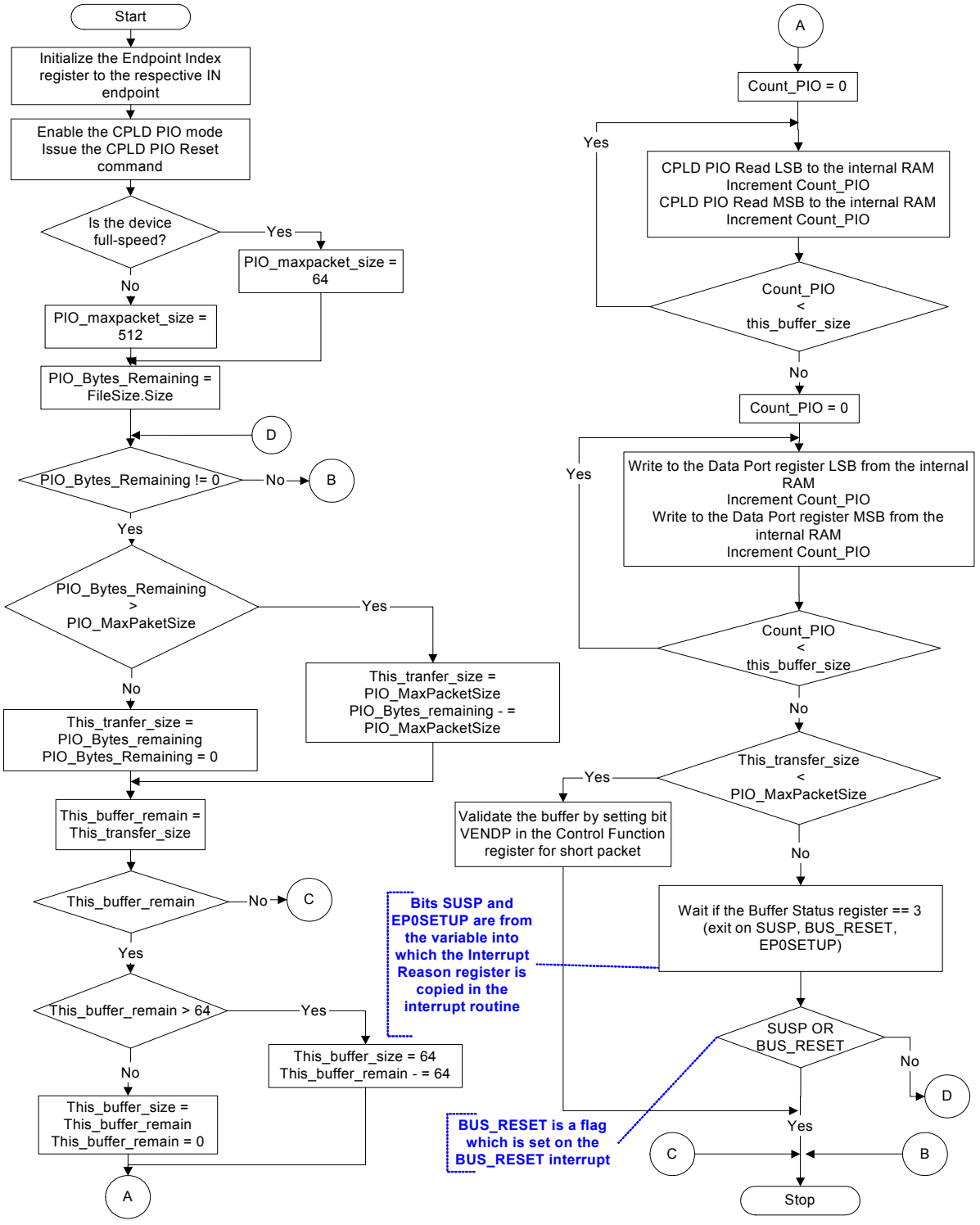


Fig 31. Handling PIO read.

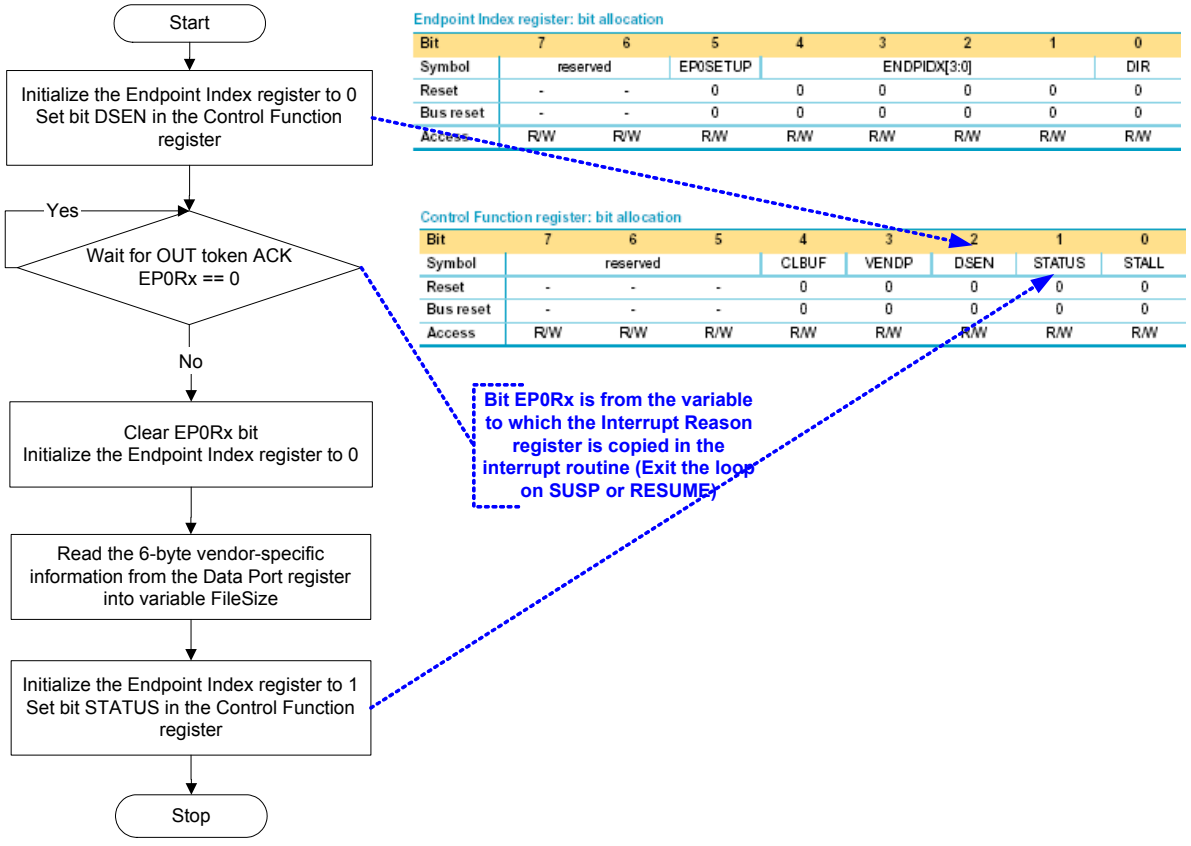


Fig 32. Handling GDMA Stop command.

6. Disclaimers

Life support — These products are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. Philips Semiconductors customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Philips Semiconductors for any damages resulting from such application.

Right to make changes — Philips Semiconductors reserves the right to make changes in the products - including circuits, standard cells, and/or software - described or contained herein in order to improve design and/or performance. When the product is in full production (status 'Production'), relevant changes will be communicated via a Customer Product/Process Change Notification (CPCN). Philips Semiconductors assumes no responsibility or liability for the use of any of these products, conveys no licence or title under any patent, copyright, or mask work right to these products, and makes no representations or warranties that these products are free from patent, copyright, or mask work right infringement, unless otherwise specified.

Application information — Applications that are described herein for any of these products are for illustrative purposes only. Philips Semiconductors make no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

7. Trademarks

SoftConnect — is a trademark of Koninklijke Philips Electronics N.V.

8. Contents

Date of release: 3 January 2005

Published in The Netherlands