M16C/6C Group

USB Sample Software

Introduction

This application note explains how to write the low-level software for Universal Serial Bus (USB) operation on the M16C/6C microcontroller.

The USB module is based on the USB Specification revision 2.0. It supports Full Speed operation.
There are seven endpoints: Control IN/OUT, Bulk IN × 2, Bulk OUT × 2 and Interrupt IN × 2.
Control, Bulk IN, Bulk OUT and Interrupt IN transfer modes are supported.
Each endpoint has its own FIFO buffer as specified below.

Table 1: Endpoint configuration

Endpoint	Transfer Type	Maximum Packet Size (bytes)	FIFO Buffer Capacity (bytes)	DMA Transfer
Endpoint 0	Setup	8	8	—
	Control IN	16	16	—
	Control OUT	16	16	—
Endpoint 1	Bulk OUT	64	64 X 2	Available
Endpoint 2	Bulk IN	64	64 X 2	Available
Endpoint 3	Interrupt IN	16	16	—
Endpoint 4	Bulk OUT	64	64 X 2	Available
Endpoint 5	Bulk IN	64	64 X 2	Available
Endpoint 6	Interrupt IN	16	16	—

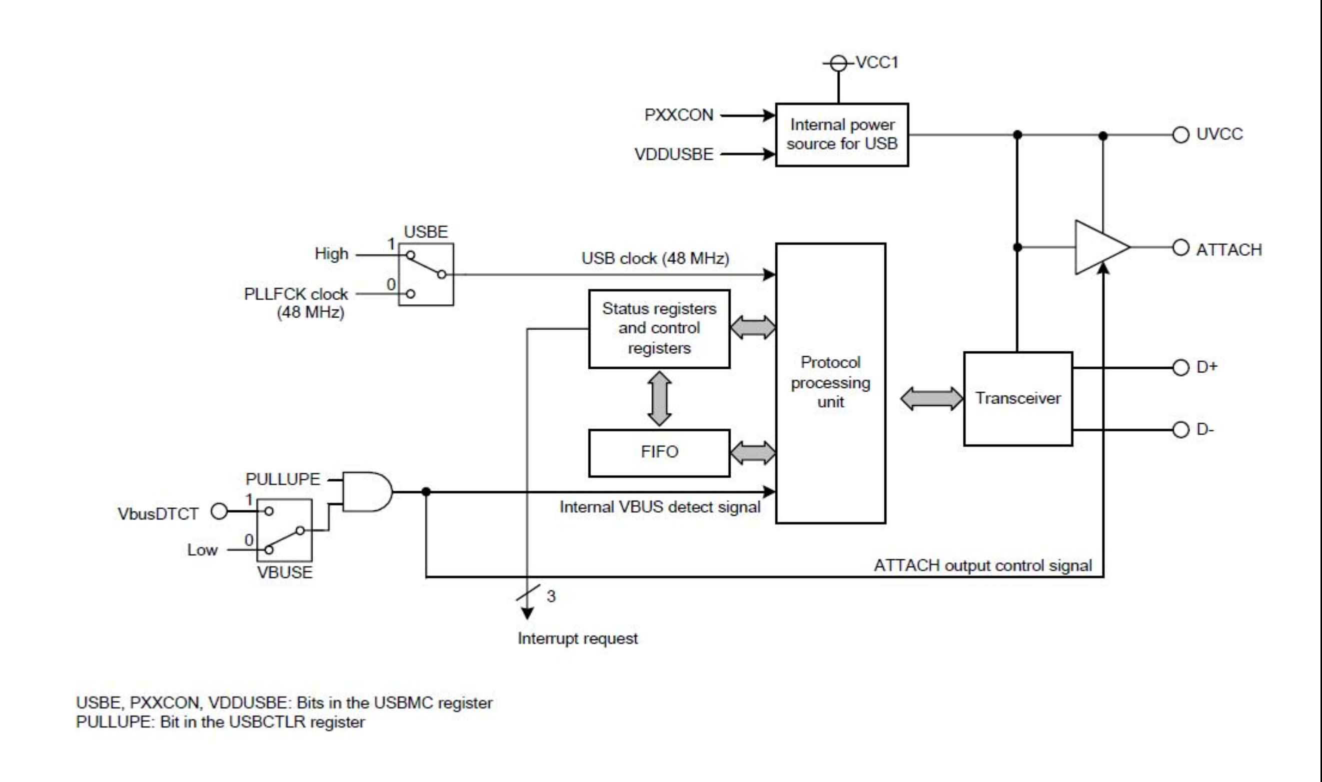
VCC1
PXXCON
VDDUSBE
Internal power source for USB
UVCC
ATTACH
USBE
High
PLLFCK clock (48 MHz)
USB clock (48 MHz)
Status registers and control registers
Protocol processing unit
Transceiver
D+
D-
FIFO
PULLUPE
VbusDTCT
Low
VBUSE
Internal VBUS detect signal
ATTACH output control signal
3
Interrupt request
USBE, PXXCON, VDDUSBE: Bits in the USBMC register
PULLUPE: Bit in the USBCTLR register

Figure 1. USB Block Diagram

Contents

1. Initialisation

This section shows how to initialise the M16C/6C so that it is ready to detect a USB cable connection and be enumerated by a host.

1.1 Pins

There are five I/O pins associated with the USB module. Configure these as specified in the following table.

Table 2: I/O pin assignments

Pin Name	Function	Configure as
ATTACH	Output used for D+ 1.5kΩ pull-up	Input pin. Set bit PD9_2 = 0.
VbusDTCT	Input the power supply signal from a host	Input pin. Set bit PD8_5 = 0.
D+	USB D+ input/output	Input pin. Set bit PD9_1 = 0.
D-	USB D- input/output	Input pin. Set bit PD9_0 = 0.
UVCC	Either input power supply for pins ATTACH, D+, and D- or if using internal USB power then 3.3V is output.	Dedicated pin, no configuration necessary.

The following code snippet configures the pins. Note that the Port 9 Direction Register is write protected by default.

```
prc2 = 1;   /* Enable writes to the PD9 register */
pd9 = 0xf8; /* Set ATTACH(P9_2), D+(P9_1) and D-(P9_0) pins as INPUT */
prc2 = 0;   /* Disable writes to the PD9 register */

pd8_5 = 0;  /* Set VbusDTCT pin as an INPUT */
```

1.2 Module Enable

The USB module must be enabled, by clearing the USBE bit in the USBMC register to 0, before any of the other USB registers can be accessed.

```
/* Enable USB module */
usbe_usbmc = 0;
```

1.3 Clocks

The USB module requires that the PLLFCK clock is configured to be 48 MHz. The following code sample does this assuming a 6 MHz main crystal frequency.

```
/* Set PLLFCK generation enabled */
plc06 = 1;
/* Set *8 and /3 therefore: */
/* PLL = 6*8*2/3 = 32MHz */
/* PLLFCK = 6*8 = 48MHz */
plcf = 0x04;

/* Enable PLL */
plc07 = 1;
```

See Figure 2. USB Initial Settings on page 5.

1.4 Power

If the voltage supplied to VCC1 is at least 4V then the required 3.3V D+/D- reference voltage can be produced internally by setting the VDDUSBE and PXXCON bits in the USBMC register. Note: This option requires the 125 KHz on-chip oscillator to be operating.

```
prc0 = 1;           /* Enable access to cm1 register    */
cm14 = 0;           /* Start 125 KHz oscillator         */
prc0 = 0;           /* Write protect cm1 registers      */

vddusbe_usbmc = 1;   /* 3.3V USB Internal power supplied */
pxxcon_usbmc = 1;    /* Enable VDDUSBE bit               */
```

If VCC1 is less than 4V the internal 3.3V power source cannot be used. A 3.3V supply must be connected to pin UVCC, with the VDDUSBE bit set to 0 and the PXXCON bit set to 1.

If powering the device from the USB VBus line (5V) then set the PWMD bit in the USBCTLR register to 1. The value of this bit is used by the M16C/6C when it automatically responds to the standard USB command 'GetStatus'.

Note that before the USBCTLR register can be accessed the USB module must be enabled by setting the USBE bit in the USBMC register to 0.

```
/* Bus powered mode */
pwmd_usbctlr = 1;
```

See Figure 2. USB Initial Settings on page 5.

1.5 Interrupt Vectors

There are three vectors relating to the USB module.

- Vector 54 USB Function 0
- Vector 55 USB Function 1
- Vector 56 USB Resume

Other than the Resume Interrupt, all other interrupts can be routed through either Vector USB Function 0 or 1 depending upon the setting of the USB Interrupt Select Registers (USBISR0-3). The decision as to which interrupt should use which vector is made purely on the grounds of increasing speed by reducing the number of status flags that need to be checked in a particular interrupt handler in order to ascertain what caused the interrupt.

The USB Resume interrupt only needs to be handled if the USB module is ever going to be actually disabled as part of a low-power mode suspended state. Otherwise a suspended state is exited through the SURSF interrupt which can be routed through either Vector USB Function 0 or 1.

Generally the USB interrupt handler needs to check the USB Interrupt Flag Registers (USBIFR0-3) for the interrupts that it has been set up to handle and act accordingly, including clearing the particular flag. The handling of particular interrupts shall be described in the section of this document relating to it. So for example the handling of the VBUS interrupt is described in Section 2. Cable Connection.

1.6 USB Module

The following flow diagram shows the procedure for USB initialisation.

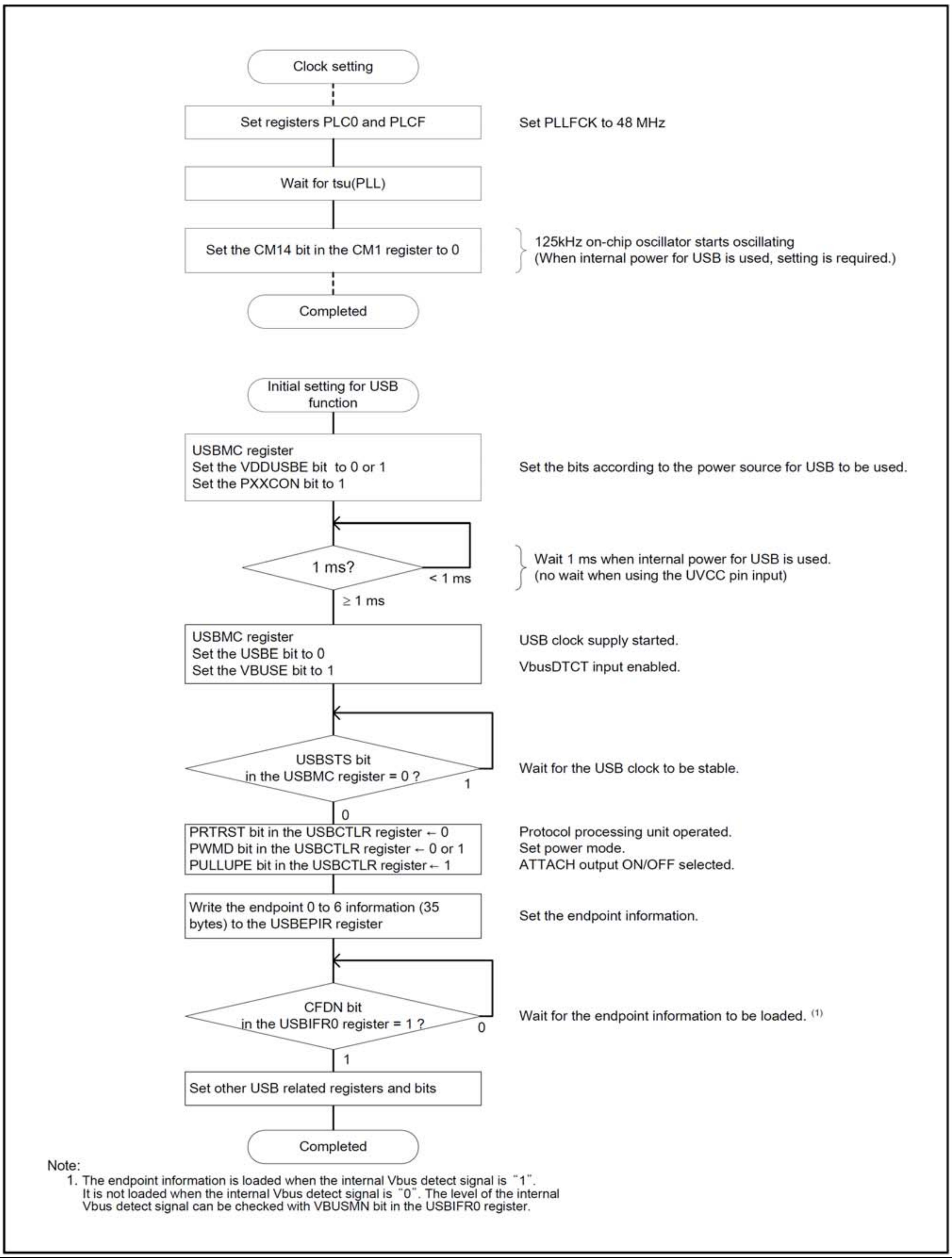
Clock setting
Set registers PLC0 and PLCF
Set PLLFCK to 48 MHz
Wait for tsu(PLL)
Set the CM14 bit in the CM1 register to 0
125kHz on-chip oscillator starts oscillating
(When internal power for USB is used, setting is required.)
Completed
Initial setting for USB function
USBMC register
Set the VDDUSBE bit to 0 or 1
Set the PXXCON bit to 1
Set the bits according to the power source for USB to be used.
1 ms?
< 1 ms
≥ 1 ms
Wait 1 ms when internal power for USB is used.
(no wait when using the UVCC pin input)
USBMC register
Set the USBE bit to 0
Set the VBUSE bit to 1
USB clock supply started.
VbusDTCT input enabled.
USBSTS bit in the USBMC register = 0 ?
1
0
Wait for the USB clock to be stable.
PRTRST bit in the USBCTLR register ← 0
PWMD bit in the USBCTLR register ← 0 or 1
PULLUPE bit in the USBCTLR register ← 1
Protocol processing unit operated.
Set power mode.
ATTACH output ON/OFF selected.
Write the endpoint 0 to 6 information (35 bytes) to the USBEPIR register
Set the endpoint information.
CFDN bit in the USBIFR0 register = 1 ?
0
1
Wait for the endpoint information to be loaded. (1)
Set other USB related registers and bits
Completed
Note:
1. The endpoint information is loaded when the internal Vbus detect signal is "1".
It is not loaded when the internal Vbus detect signal is "0". The level of the internal
Vbus detect signal can be checked with VBUSMN bit in the USBIFR0 register.

Figure 2. USB Initial Settings

A code sample of setting the Endpoint Information can be seen in Reference 9.1 USB Endpoint Information Loading on page 19.

In the above flow diagram, interrupts can be configured at the stage labelled “Set other USB related registers required”. The following code snippet shows the setting up of the interrupt priorities and the routing of all interrupts via interrupt 0 by the setting of the USB Interrupt Select Register:

```
/* Set USB interrupt priority */
usbint0ic = 0x03;
usbint1ic = 0x03;
usbrsmic = 0x03;

/* Use only USB Interrupt 0 */
usbisr0 = 0x00;
usbisr1 = 0x00;
usbisr2 = 0x00;
```

For the Cable Connection and Enumeration stages to operate as specified in the following sections the VBUSFE, BRSTE and SETUPTSE interrupts should be enabled. The handling of these interrupts will be explained in the following sections.

```
/* Enable Interrupts required for Connection and Enumeration */
brste_usbier0 = 1;
setuptse_usbier1 = 1;
vbusfe_usbier0 = 1;
```

2. Cable Connection

If the setup of the device has been completed as specified in the Initialisation section of this document then, when the USB cable is connected, the USB Bus Connection/Disconnection flag (VBUSF) will be set and a VBUSF interrupt generated. In the handler of this interrupt, the Internal VBUS Signal Monitor flag (VBUSMN) in the USB Interrupt Flag register 0 (USBIFR0) can be checked to see if this is a connection (VBUSMN = 1) or a disconnection (VBUSMN = 0) event. Clear the VBUSF flag and then wait for the USB Bus Reset Detect Flag (BRST) bit to cause a BRST interrupt.

The BRST interrupt should be handled by clearing the BRST flag and by clearing all Endpoint FIFOs by setting the relevant bits in the USBFCLR registers. Note that these registers are not readable so write to the register as a whole rather than doing a read-modify-write operation to access individual bits.

This code snippet clears all FIFO buffers on all endpoints:

```
usbfclr0 = 0x03; /* EP0oCLR and EP0iCLR */
usbfclr1 = 0x07; /* EP1CLR, EP2CLR & EP3CLR */
usbfclr2 = 0x07; /* EP4CLR, EP5CLR & EP6CLR */
```

If using the Endpoint Stall mechanism (See section 6. Stall) it is also a good idea to clear any possible stall conditions by setting the relevant bits in the USBEPSTL registers.

This code snippet clears all stall conditions on all endpoints:

```
usbepstl0 = 0x10; /* EP0 Stall Clear */
usbepstl1 = 0x70; /* EP 1, 2 & 3 Stall Clear */
usbepstl2 = 0x70; /* EP 4, 5 & 6 Stall Clear */
```

The following flowchart details this operation:-

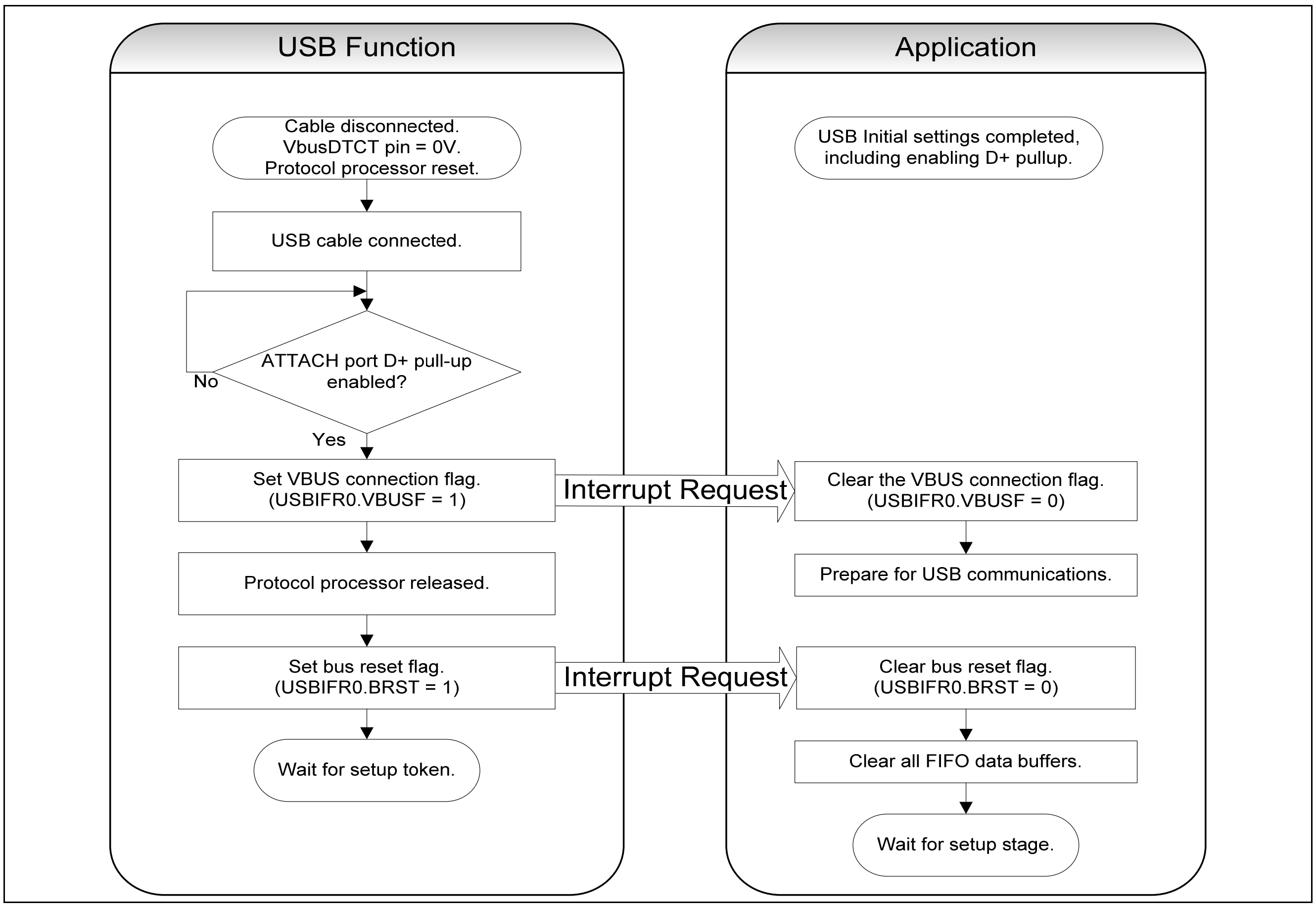
USB Function
Application
Cable disconnected. VbusDTCT pin = 0V. Protocol processor reset.
USB Initial settings completed, including enabling D+ pullup.
USB cable connected.
ATTACH port D+ pull-up enabled?
No
Yes
Set VBUS connection flag. (USBIFR0.VBUSF = 1)
Interrupt Request
Clear the VBUS connection flag. (USBIFR0.VBUSF = 0)
Protocol processor released.
Prepare for USB communications.
Set bus reset flag. (USBIFR0.BRST = 1)
Interrupt Request
Clear bus reset flag. (USBIFR0.BRST = 0)
Clear all FIFO data buffers.
Wait for setup token.
Wait for setup stage.

Figure 3. Cable Connection

3. Enumeration / Control Transfers

Following the USB Bus Connection/Disconnection Detect (VBUSF) and USB Bus Reset Detect (BRST) interrupts, the process of Enumeration will start with the reception of a Setup command.

Enumeration is the process of the host interrogating the device to discover what it is and what capabilities it has. The host does this by using Control Transfers. A Control transfer consists of the following stages:

- **Setup**. The host sends an 8-byte setup command of a fixed format.
- **Data**. Depending upon the command there may be a data stage in either direction IN or OUT.
- **Status**. This reports success or failure of the command.

3.1 Setup Stage

In the setup stage, command data from the host is analysed to determine the subsequent processing required such as if a data stage is required and what direction it is in.

When a Setup command is received the Setup Command Receive Complete flag (SETUPTS) will be set causing a SETUPTS interrupt. Clear the SETUPTS flag.

Because it is possible for the host to interrupt a previous control transfer by sending a new setup command, both the IN and OUT FIFOs on the control endpoint should be cleared. The 8-byte setup command should be read, one byte at a time, from the USP Endpoint 0 Status Data register (USBEPDR0S) and then the Endpoint 0S Read Complete Trigger bit (EP0SRDFN) set to 1.

The following code snippet performs this operation:

```
/* Clear the Interrupt Flag */
/* Set setups bit to 0. (Register must be accessed as 8bit)*/
usbifr1 = (unsigned char)~BIT_3;

/*Clear EP0i and EP0o FIFOs*/
/* Register is not readable - so don't use bit definitions.*/
usbfclr0 = (unsigned char)(BIT_0 | BIT_1); /* EP0oCLR = 1 and EP0iCLR = 1 */

/* Read 8 bytes of data from EP0 into temporary buffer */
for(index = 0; index < USB_SETUP_PACKET_SIZE; index++)
{
   SetupCmdBuffer[index] = (UINT8)usbepdr0s;
}

/*Set EP0 Status Read Complete*/
/*Set EP0SRDFN bit*/
usbtrg0 = BIT_2;
```

The setup commands that need to be supported depend upon the USB class being implemented but for basic enumeration only the standard Get Descriptor request needs to be handled as the M16C/6C automatically handles the following standard requests:

- Get Configuration
- Get Interface
- Get Status
- Set Address
- Set Feature
- Clear Feature

Note: If the Setup command is not supported or has not been understood the Control Endpoint (Endpoint 0) must be stalled. See Section 6. Stall.

The following flowchart details this setup operation:-

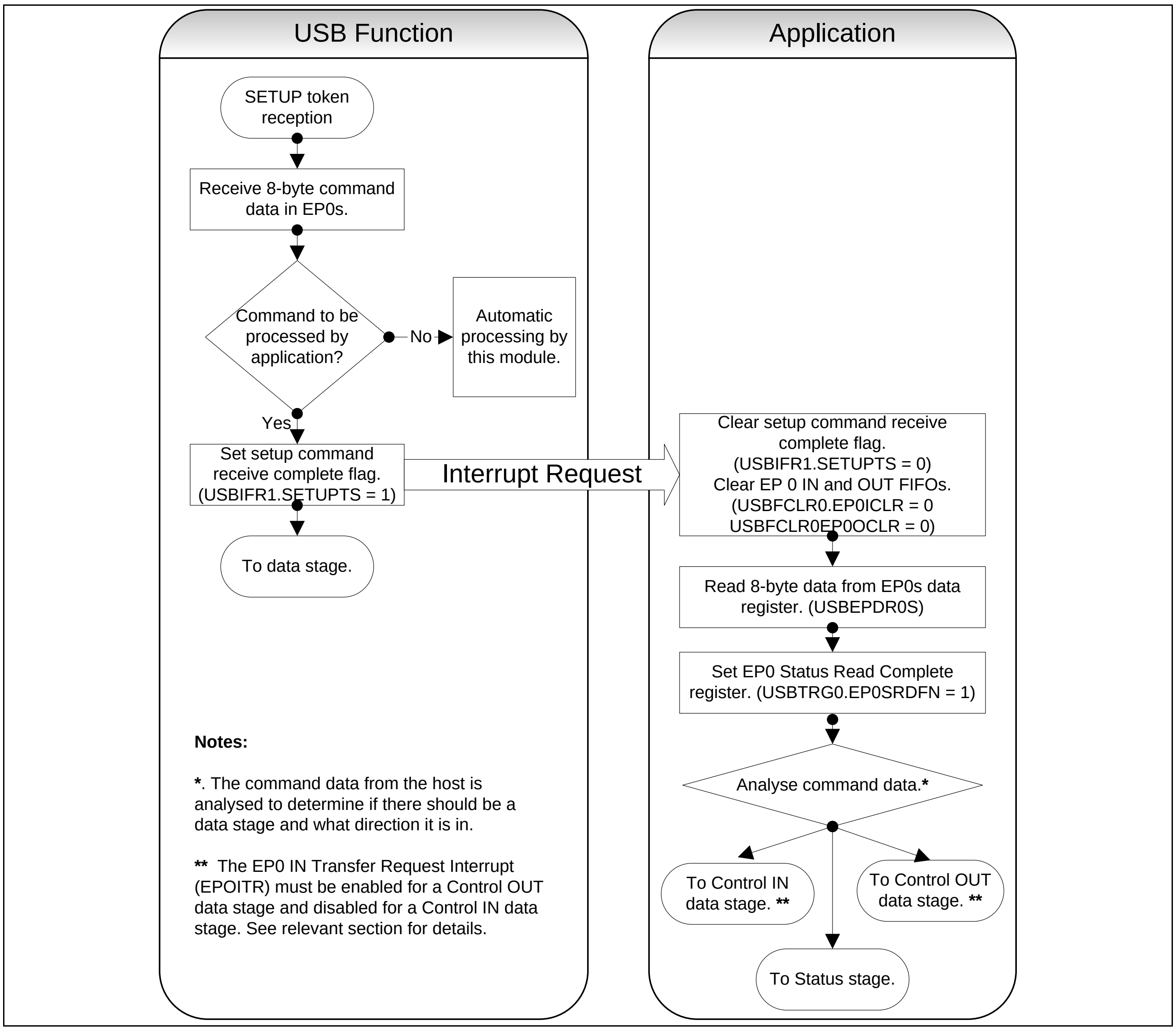
USB Function
Application
SETUP token reception
Receive 8-byte command data in EP0s.
Command to be processed by application?
No
Automatic processing by this module.
Yes
Set setup command receive complete flag. (USBIFR1.SETUPTS = 1)
Interrupt Request
To data stage.
Clear setup command receive complete flag. (USBIFR1.SETUPTS = 0) Clear EP 0 IN and OUT FIFOs. (USBFCLR0.EP0ICLR = 0 USBFCLR0EP0OCLR = 0)
Read 8-byte data from EP0s data register. (USBEPDR0S)
Set EP0 Status Read Complete register. (USBTRG0.EP0SRDFN = 1)
Analyse command data.*
To Control IN data stage. **
To Control OUT data stage. **
To Status stage.
Notes:
*. The command data from the host is analysed to determine if there should be a data stage and what direction it is in.
** The EP0 IN Transfer Request Interrupt (EPOITR) must be enabled for a Control OUT data stage and disabled for a Control IN data stage. See relevant section for details.

Figure 4. Setup Stage

3.2 Data Stage

The setup command includes a field (wLength) specifying the length of the data stage. There may be no data stage (wLength set to 0) in which case skip straight to the Status Stage. If there is a data stage then another field in the setup command (bmRequestType) says if the direction of the data flow is IN or OUT.

3.2.1 Control IN

A Control IN data stage is used to send data to the host on the control pipe in response to a setup command that requests it. Note that this is how descriptors will be passed to the host in response to the Get Descriptor standard request.

Enable interrupts Endpoint 0 IN Transmit Complete Flag (EP0iTS) and Endpoint 0 OUT Receive Complete Flag (EP0OTS). Ensure Endpoint 0 IN Transfer Request Flag (EP0iTR) is disabled.

Write the data a byte at a time to the USBEPDR0I register until all the data has been sent or until the maximum packet size of 16 bytes has been written. Set the Endpoint 0 IN Packet Enable Trigger bit (EP0IPKTE) to send the data and then wait for an EP0iTS or EP0oTS interrupt.

Handle an EP0iTS interrupt by sending up to another packets worth of data. If there is no more data to send then send a zero length packet by setting EP0IPKTE without first writing any data to USBEPDR0I.

An EP0oTS interrupt signals the end of the data stage as this is the host sending the status stage.
See Section 3.3, 'Status Stage'.

Note that some hosts do not wait to receive all the data that they have requested before moving on to the Status Stage. In this case abandon the remainder of the data stage.

The following flowchart details this operation:-

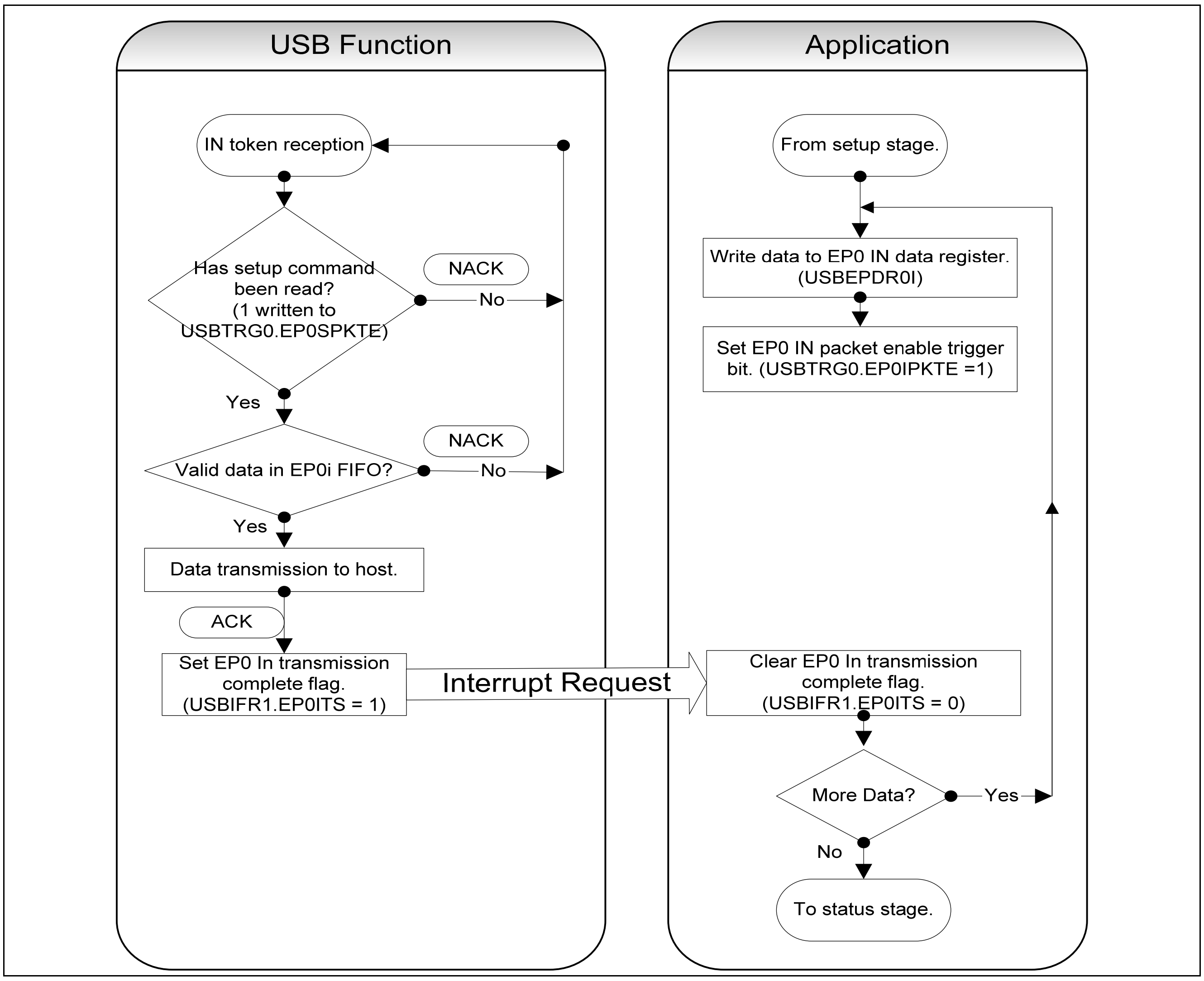
USB Function
Application
IN token reception
Has setup command been read? (1 written to USBTRG0.EP0SPKTE)
NACK
No
Yes
Valid data in EP0i FIFO?
NACK
No
Yes
Data transmission to host.
ACK
Set EP0 In transmission complete flag. (USBIFR1.EP0ITS = 1)
Interrupt Request
From setup stage.
Write data to EP0 IN data register. (USBEPDR0I)
Set EP0 IN packet enable trigger bit. (USBTRG0.EP0IPKTE =1)
Clear EP0 In transmission complete flag. (USBIFR1.EP0ITS = 0)
More Data?
Yes
No
To status stage.

Figure 5. Data Stage – Control IN.

3.2.2 Control OUT

A Control OUT data stage is used to receive data from the host on the control pipe following a setup command that requires it. Note that this is not required for basic enumeration of standard requests but there are some class requests that do require it such as the Communication Device Class (CDC/ACM) which uses the ‘SET LINE CODING’ request and the Human Interface Device (HID) class that uses a ‘SET REPORT’ request.

Enable EP0iTS, EP0oTS and EP0iTR interrupts.

When data is received from the host an EP0oTS interrupt will be generated. In the handler for this read the number of bytes (as indicated in the EPSZ0o register), a byte at a time, from the Endpoint 0 OUT Data register (EPDR0o). Clear the interrupt flag and set the read complete bit EP0oRDFN. This process repeats for each packet of data the host sends. When the host has sent all the data it intends to it will try and read the status stage. This will result in an EPOiTR interrupt being generated. See Section 3.3 ’Status Stage’.

The following flowchart details this operation:-

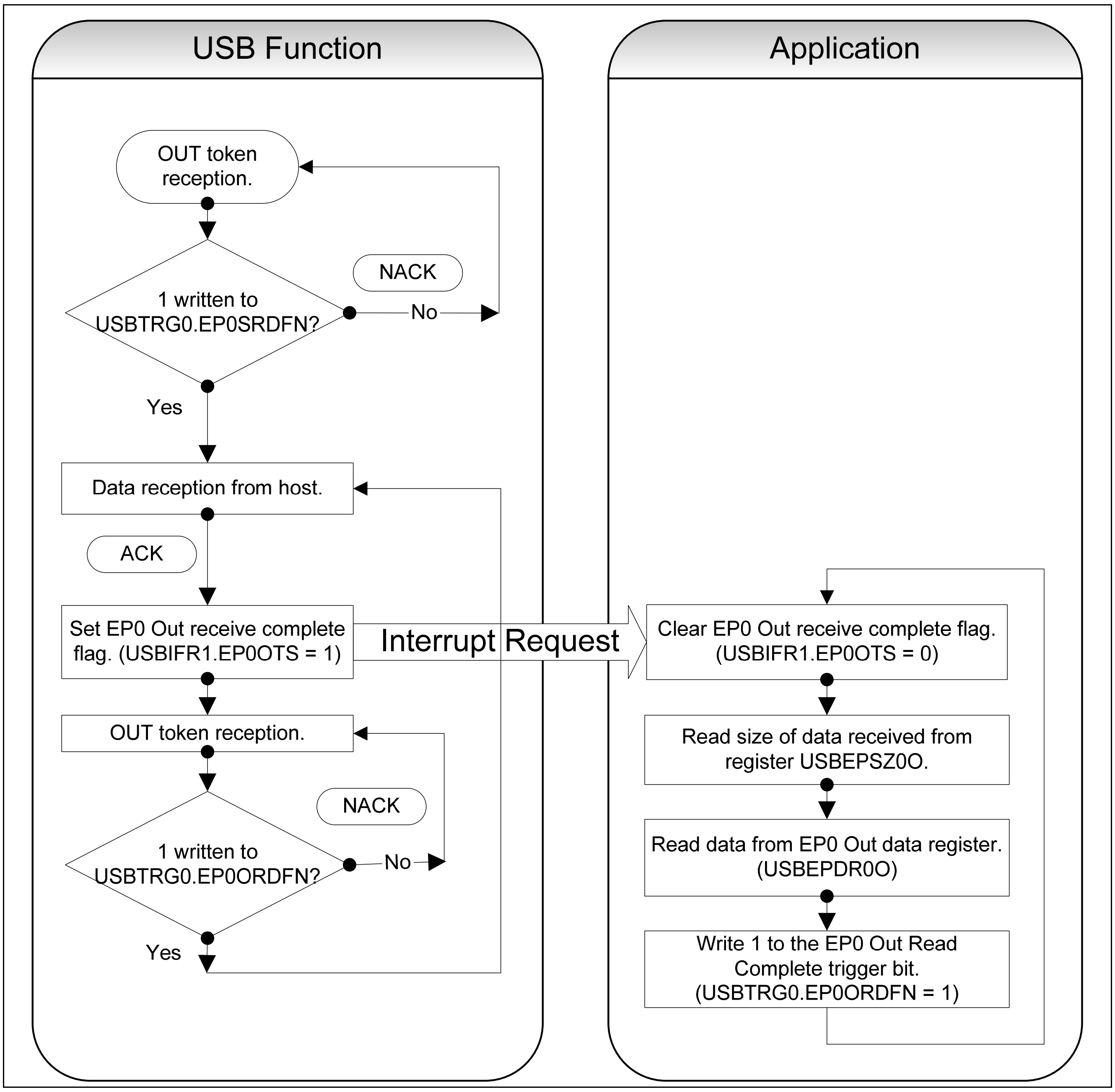
USB Function
Application
OUT token reception.
NACK
1 written to USBTRG0.EP0SRDFN?
No
Yes
Data reception from host.
ACK
Set EP0 Out receive complete flag. (USBIFR1.EP0OTS = 1)
Interrupt Request
Clear EP0 Out receive complete flag. (USBIFR1.EP0OTS = 0)
OUT token reception.
NACK
1 written to USBTRG0.EP0ORDFN?
No
Yes
Read size of data received from register USBEPSZ0O.
Read data from EP0 Out data register. (USBEPDR0O)
Write 1 to the EP0 Out Read Complete trigger bit. (USBTRG0.EP0ORDFN = 1)

Figure 6. Data Stage – Control OUT.

3.3 Status Stage

3.3.1 Sending

If there is not a data stage for this Control Transfer or a Data OUT stage has completed (EP0iTR interrupt received) then the device is the source of the status stage. Cause an ACK to be sent to the host by sending a zero length IN packet. Do this by setting EP0IPKTE bit without first writing any data to USBEPDR0I.

```
/* Transmit 0-byte data to host (ACK):- Packet Enable - Set EP0IPKTE bit*/
usbtrg0 = BIT_0;
```

Then handle the resulting EP0 IN Transmission Complete interrupt (EPOiTS) by clearing the interrupt flag.

The following flowchart details this operation:-

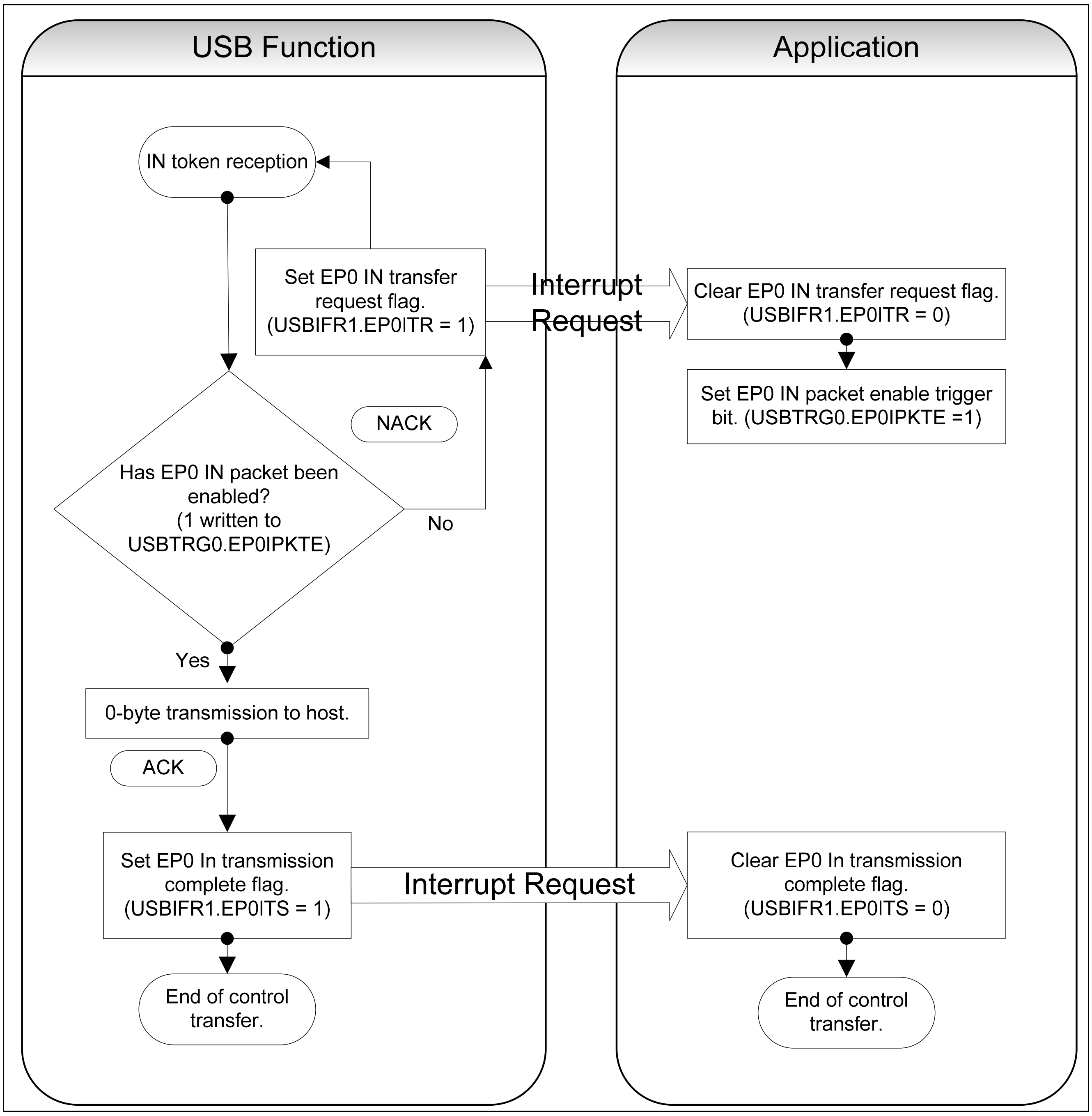
USB Function
Application
IN token reception
Set EP0 IN transfer request flag. (USBIFR1.EP0ITR = 1)
Interrupt Request
Clear EP0 IN transfer request flag. (USBIFR1.EP0ITR = 0)
Set EP0 IN packet enable trigger bit. (USBTRG0.EP0IPKTE =1)
NACK
Has EP0 IN packet been enabled? (1 written to USBTRG0.EP0IPKTE)
No
Yes
0-byte transmission to host.
ACK
Set EP0 In transmission complete flag. (USBIFR1.EP0ITS = 1)
Interrupt Request
Clear EP0 In transmission complete flag. (USBIFR1.EP0ITS = 0)
End of control transfer.
End of control transfer.

Figure 7. Status Stage (Control OUT)

3.3.2 Receiving

The host is the source of the status stage if there has been a Data IN stage. An EP0oTS interrupt will occur when the ACK is received from the host. Clear the EP0oTS interrupt flag and set the EP0 OUT Read complete trigger bit (EP0oRDFN).

```
/* Clear the Interrupt Flag */
/* Set EP0oTS bit to 0. (Register must be accessed as 8bit)*/
usbifr1 = (unsigned char)~BIT_2;

/* EP0o Read complete */
/*Set EP0ORDFN bit*/
usbtrg0 = BIT_1;
```

The following flowchart details this operation:-

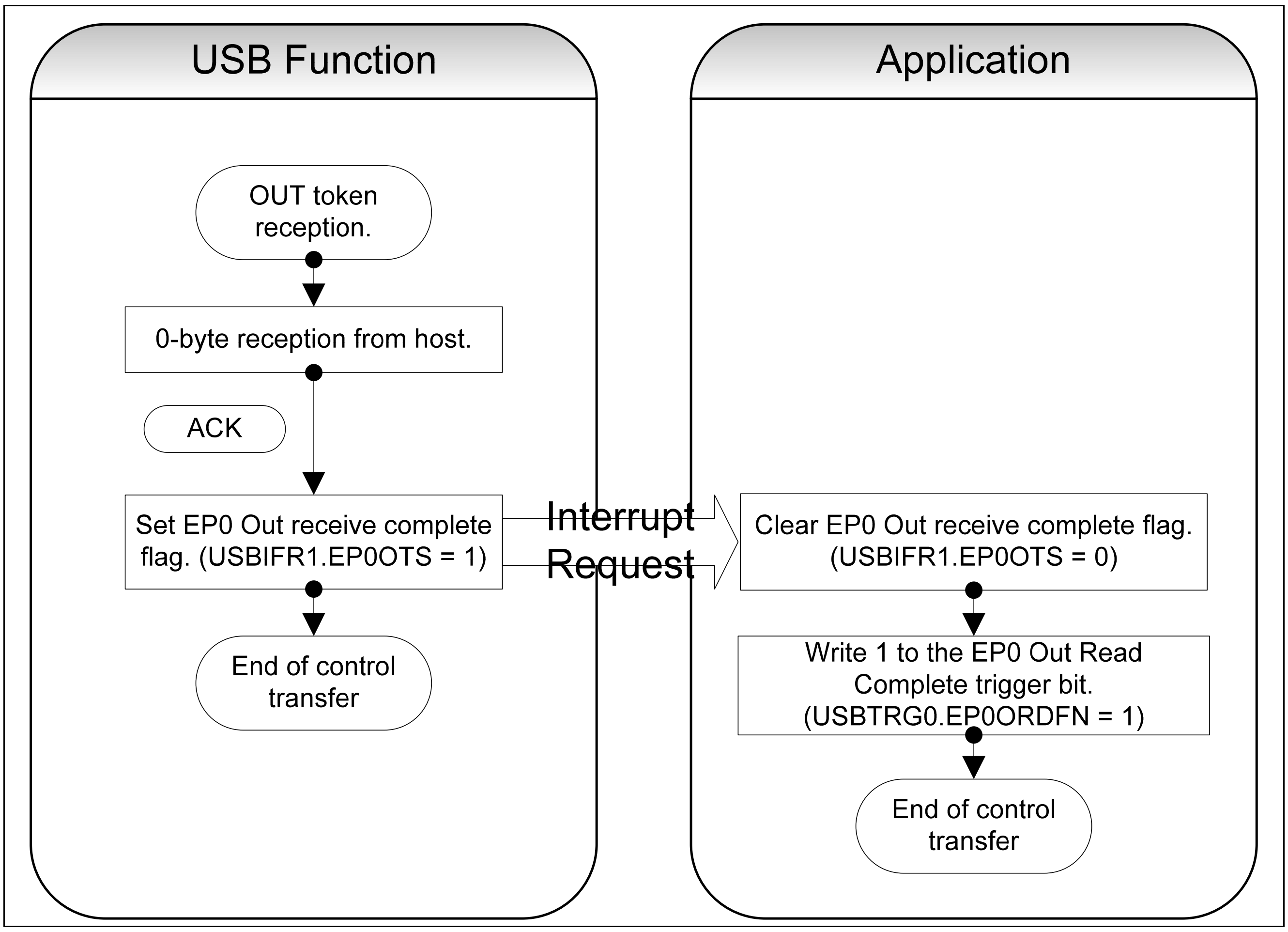
USB Function
Application
OUT token reception.
0-byte reception from host.
ACK
Set EP0 Out receive complete flag. (USBIFR1.EP0OTS = 1)
End of control transfer
Interrupt Request
Clear EP0 Out receive complete flag. (USBIFR1.EP0OTS = 0)
Write 1 to the EP0 Out Read Complete trigger bit. (USBTRG0.EP0ORDFN = 1)
End of control transfer

Figure 8. Status Stage (Control IN).

4. Bulk Transfer

The M16C/6C supports both Bulk IN and Bulk OUT.

4.1 Bulk IN

The M16C/6C has two identical Bulk IN endpoints, Endpoint 2 and Endpoint 5. The following description assumes that Endpoint 2 is being used but to change this to use Endpoint 5 it is simply a matter of using EP5 rather than EP2 related registers.

If you need to know when the host is requesting a Bulk IN transfer, then enable the Endpoint 2 Transfer Request (EP2TR) interrupt:

```
ep2tre_usbier2 = 1;
```

In the interrupt handler for this, or if not using the EP2TR interrupt then just at the point where you want to start a Bulk IN transfer, enable the Endpoint 2 FIFO Empty (EP2EMPTY) interrupt:

```
ep2emptye_usbier2 = 1;
```

In the EP2EMPTY interrupt handler write the data, one byte at a time, to the Endpoint 2 Data Register (USBEPDR2) until all the data has been sent or until the maximum packet size of 64 bytes has been written.
Set the Endpoint 2 Packet Enable Trigger bit (EP2PKTE) to send the data.

Repeat this handling of the EP2EMPTY interrupt until there is no more data to send. In some cases the host knows how much data the device will be sending but usually a transfer will need to be completed by ensuring that a non-complete packet of data has been sent. If the amount of data is equal to an integer number of full packets then in order to end the transfer on a non-complete packet it is necessary to send an empty packet by setting the EP2PKTE bit without first having written any data to USBEPDR2. The EP2EMPTY interrupt can then be disabled.

The following flowchart details this operation:-

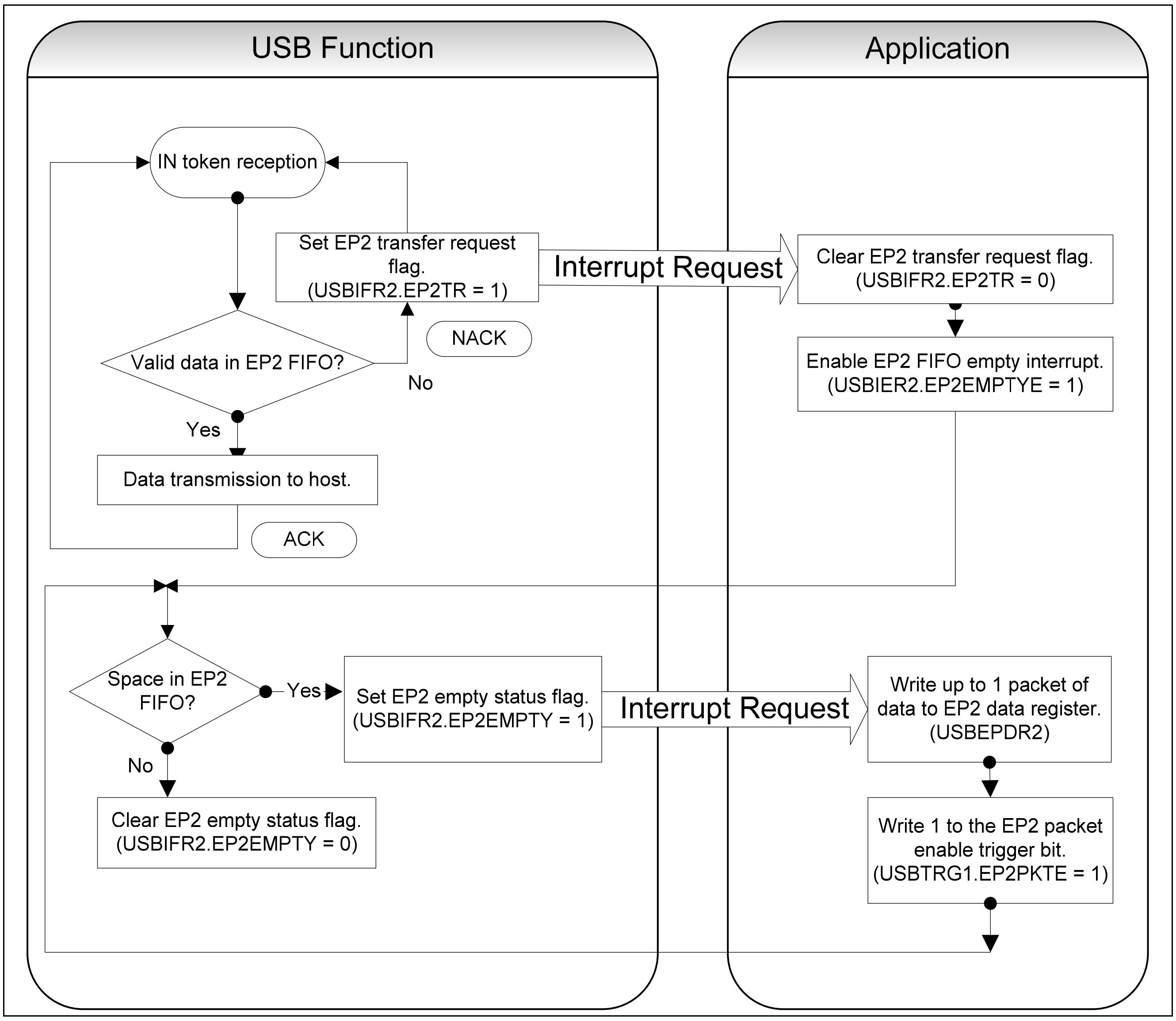
USB Function
Application
IN token reception
Set EP2 transfer request flag. (USBIFR2.EP2TR = 1)
Interrupt Request
Clear EP2 transfer request flag. (USBIFR2.EP2TR = 0)
NACK
Valid data in EP2 FIFO?
No
Yes
Enable EP2 FIFO empty interrupt. (USBIER2.EP2EMPTYE = 1)
Data transmission to host.
ACK
Space in EP2 FIFO?
Yes
Set EP2 empty status flag. (USBIFR2.EP2EMPTY = 1)
Interrupt Request
Write up to 1 packet of data to EP2 data register. (USBEPDR2)
No
Clear EP2 empty status flag. (USBIFR2.EP2EMPTY = 0)
Write 1 to the EP2 packet enable trigger bit. (USBTRG1.EP2PKTE = 1)

Figure 9. EP2 Bulk IN Transfer.

4.2 Bulk OUT

The M16C/6C has two identical Bulk OUT endpoints, EP1 and EP4. The following assumes EP1 is being used.

Enable the Endpoint 1 FIFO Full (EP1FULL) interrupt:

ep2tre_usbier2 = 1;

When an EP1FULL interrupt occurs it means that the host has sent some data. The number of bytes received can be read from the USB Endpoint 1 Receive Data Size register (USBEPSZ1). Read each byte, one byte at a time, from the USB Endpoint 1 Data register (USBEPDR1) and then set the read complete bit.

```
/*Read Complete*/
ep1rdfn_usbtrg1 = 1;
```

The following flowchart details this operation:-

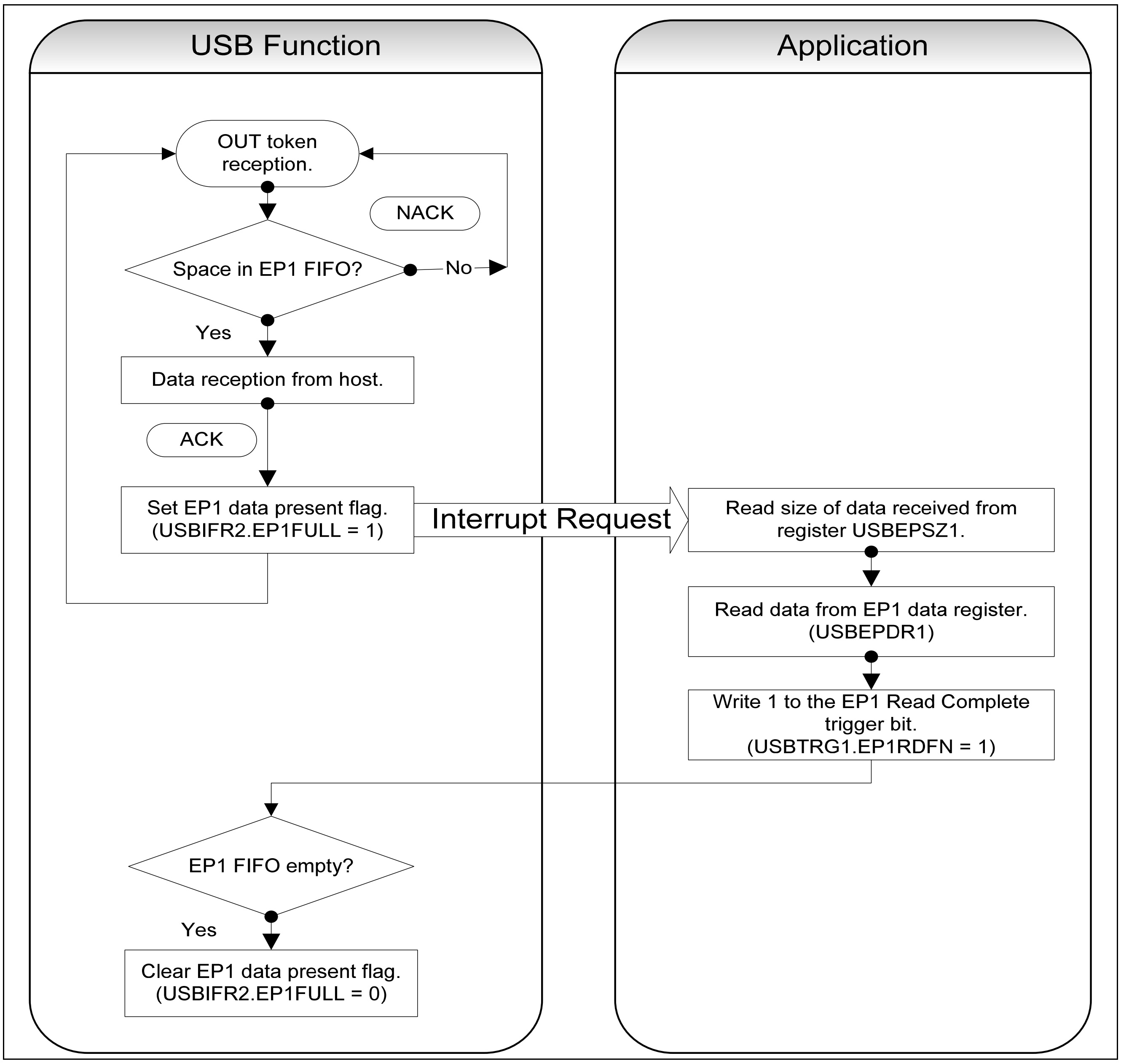
USB Function
Application
OUT token reception.
NACK
Space in EP1 FIFO?
No
Yes
Data reception from host.
ACK
Set EP1 data present flag. (USBIFR2.EP1FULL = 1)
Interrupt Request
Read size of data received from register USBEPSZ1.
Read data from EP1 data register. (USBEPDR1)
Write 1 to the EP1 Read Complete trigger bit. (USBTRG1.EP1RDFN = 1)
EP1 FIFO empty?
Yes
Clear EP1 data present flag. (USBIFR2.EP1FULL = 0)

Figure 10. EP1 Bulk OUT Transfer.

5. Interrupt Transfer

The M16C/6C supports two identical Interrupt IN endpoints, Endpoint 3 and Endpoint 6. The following assumes EP3 is being used.

To start an Interrupt IN transfer, send up to a packets worth of data to be sent to the host by writing the data, one byte at a time, to the Endpoint 3 Data register (USBEPDR3) until it has all been sent or until the maximum packet size of 16 bytes has been written. Set the Endpoint 3 Packet Enable Trigger bit (EP3PKTE) to send the data. If a full packet has been written then enable the Endpoint 3 Transmit Complete (EP3TS) interrupt. In the EP3TS interrupt handler clear the EP3TS interrupt flag and repeat this process to send up to another packets worth of data.

```
/*Clear the interrupt flag*/
/* Set EP3TS bit to 0. (Register must be accessed as 8bit)*/
usbifr2 = (unsigned char)~BIT_4;
```

Repeat this handling of the EP3TS interrupt until there is no more data to send. In some cases the host knows how much data the device will be sending but usually a transfer will need to be ended by ensuring that a non-complete packet of data has been sent. If the amount of data is equal to an integer number of full packets then in order to end the transfer on a non-complete packet it is necessary to send an empty packet by setting the EP3PKTE bit without first having written any data to USBEPDR3. The EP3TS interrupt can then be disabled.

The following flowchart details this operation:-

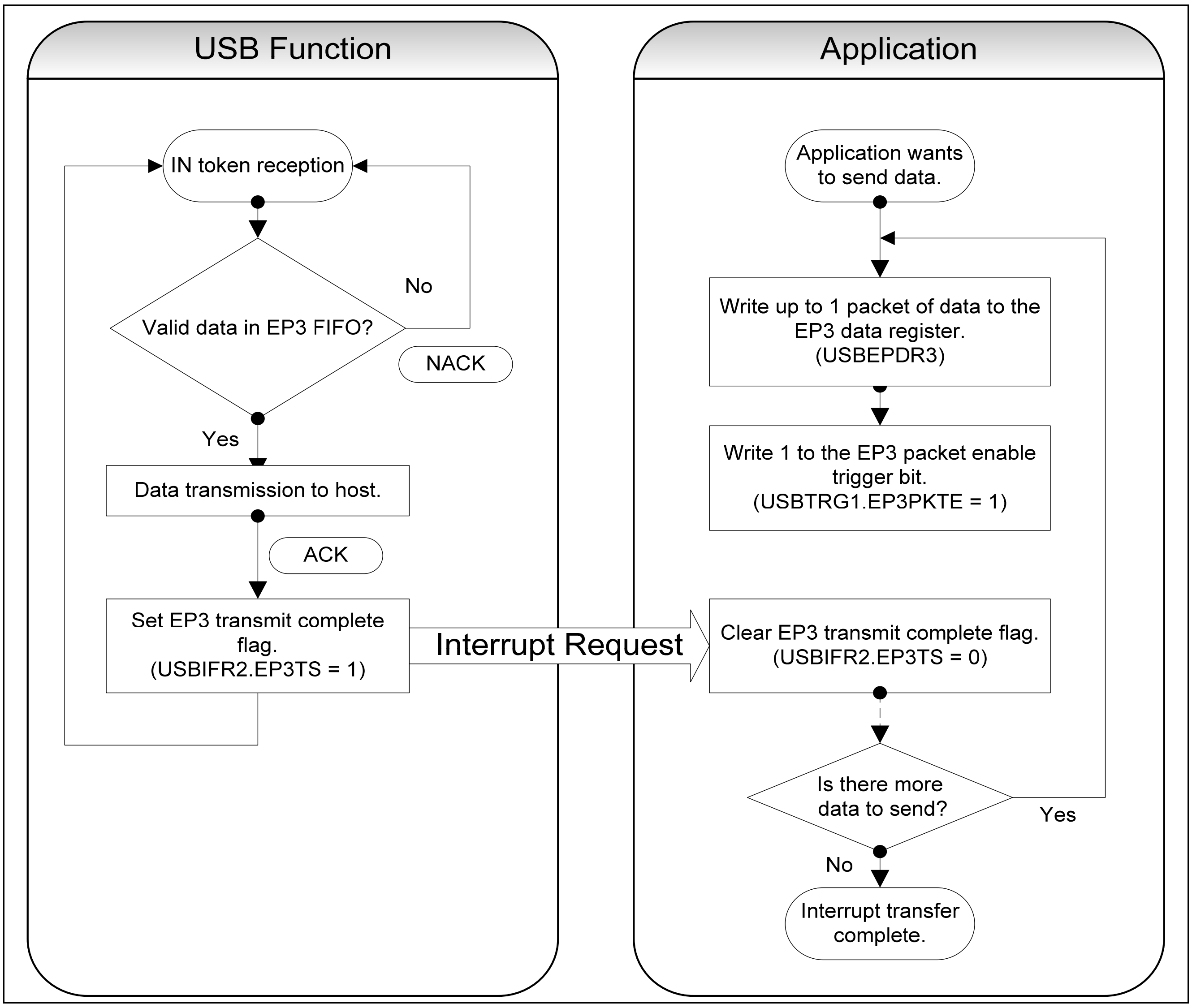
USB Function
Application
IN token reception
Valid data in EP3 FIFO?
No
NACK
Yes
Data transmission to host.
ACK
Set EP3 transmit complete flag. (USBIFR2.EP3TS = 1)
Interrupt Request
Application wants to send data.
Write up to 1 packet of data to the EP3 data register. (USBEPDR3)
Write 1 to the EP3 packet enable trigger bit. (USBTRG1.EP3PKTE = 1)
Clear EP3 transmit complete flag. (USBIFR2.EP3TS = 0)
Is there more data to send?
Yes
No
Interrupt transfer complete.

Figure 11. EP3 Interrupt Transfer.

6. Stall

A stall is a type of handshake packet that can be sent from a function to a host. This is used to indicate that a particular endpoint can not currently receive or transmit data or that a control pipe request is not supported. The USB Mass Storage class also uses it to end a transfer prematurely. A host can also command a function to enter a stalled state.

The M16C/6C automatically handles the 'Set Feature (ENDPOINT_HALT)' and 'Clear Feature (ENDPOINT_HALT)' commands for stall processing. However, if it is the function that needs to instigate the stall then this can be done by setting the endpoint's Stall Setting Bit in the USB Endpoint Stall registers.

```
/* Stall Endpoint 0, set EP0STLS bit*/
usbepstl0 = BIT_0;
```

Each endpoint also has a corresponding 'Stall Clear Bit' which can be set to clear the 'Stall Setting Bit'. Endpoint 0's stall is automatically cleared when the next Setup Control command is received. All endpoints can also independently be set up so that the endpoint's Stall Setting Bit is automatically cleared after a stall condition has been returned to the host. This is achieved by setting the endpoint's corresponding 'Automatic Stall Clear' bit.

```
/* Enable Automatic stall clear on EP1 */
ep1asce_usbstlsr1 = 1;
```

7. Suspend / Resume

NOTE: This document does not cover disabling the USB module in the suspended state or the operation of the remote wakeup feature. Please refer to the M16C/6C Hardware Manual for this.

The Suspend and Resume detect flag (SURSF) interrupt can be handled to know when the USB bus has been suspended and subsequently when it has been resumed. If not actually disabling the USB Module during the suspended state there is no need to handle the Standby Clear Resume Detect (SSRSME) interrupt. The device can instigate a wake up of the USB bus from the suspended state by executing a remote wakeup operation. If not using this feature then ensure that the Resume Request bit (RSME) is always set to 0.

8. Cable Disconnection

When the USB cable is disconnected the USB Bus Connection/Disconnection Flag (VBUSF) will be set and a VBUSF interrupt generated. In the handler of this interrupt the VBUSMN flag in register USBIFR0 can be checked to confirm that this is a disconnection (VBUSMN = 0) event. Clear the VBUSF flag.

9. Reference

9.1 USB Endpoint Information Loading

The USB Endpoint Information Register (USBEPIR) allows each endpoint to be configured for a specific USB Configuration, Interface and Alternate setting. The endpoint number as the host sees it can also be changed. This may be useful if the host application requires specific endpoints to support specific transfer types. The following code shows writing to the USBEPIR register in order to set up all endpoints for a device with a single USB Configuration and Interface with no Alternate setting. Each endpoint requires 5 byte writes to the USBEPIR register. If not using a certain endpoint then write all zeros to it. Once the configuration has been completed it can not be changed.

```
/* EP 0 */
usbepir = 0x00;
usbepir = 0x00;
usbepir = 0x20;
usbepir = 0x00;
usbepir = 0x00;

/* EP 1 */
usbepir = 0x14;
usbepir = 0x20;
usbepir = 0x80;
usbepir = 0x00;
usbepir = 0x01;

/* EP 2 */
usbepir = 0x24;
usbepir = 0x28;
usbepir = 0x80;
usbepir = 0x00;
usbepir = 0x02;

/* EP 3 */
usbepir = 0x34;
usbepir = 0x38;
usbepir = 0x20;
usbepir = 0x00;
usbepir = 0x03;

/* EP 4 */
usbepir = 0x44;
usbepir = 0x20;
usbepir = 0x80;
usbepir = 0x00;
usbepir = 0x04;

/* EP 5 */
usbepir = 0x54;
usbepir = 0x28;
usbepir = 0x80;
usbepir = 0x00;
usbepir = 0x05;

/* EP 6 */
usbepir = 0x64;
usbepir = 0x38;
usbepir = 0x20;
usbepir = 0x00;
usbepir = 0x06;
```

Website and Support

Renesas Technology Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/inquiry
csc@renesas.com