APPLICATION NOTE

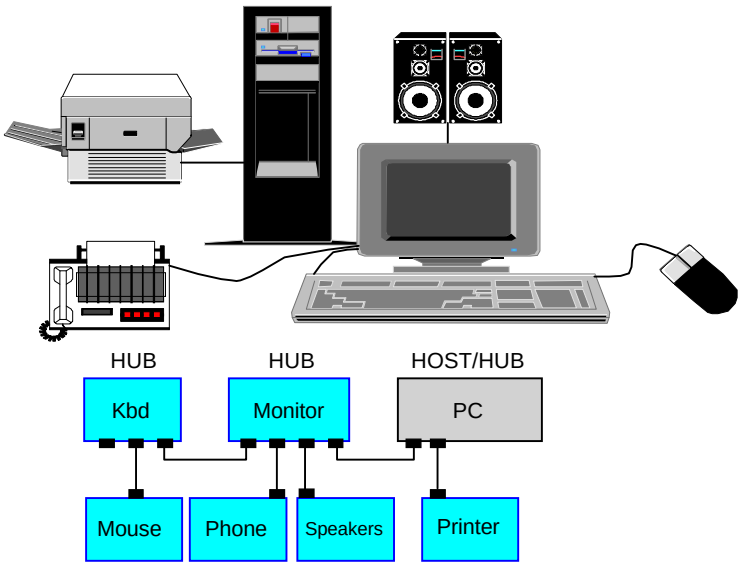
USING THE ST7 UNIVERSAL SERIAL BUS MICROCONTROLLER

by Microcontroller Division Applications

1 INTRODUCTION

1.1 WHAT IS USB

Figure 1. A USB topology example



The Universal Serial Bus is an industry standard that brings Plug-and-Play technology to the PC peripherals. The key features are:

- Ease of use: peripherals will be detected and configured automatically when physically attached, hot plugging allows adding and removing devices without powering down or rebooting, using a single connector for all devices.
- Built in power distribution for low power devices.
- Port expansion: enables up to 127 different PC peripherals to be plugged to a PC.

Table of Contents

1.2 SCOPE

The ST7 USB interface is a Universal Serial Bus peripheral that provides a means of connecting a PC peripheral serving as a function to a PC host. It supports low speed data transfers.

Objectives: this application note describes an example firmware for interaction with the USB interface hardware and support interactions between a USB device and a host system.

The USB function firmware is divided into three layers as shown in figure 2.

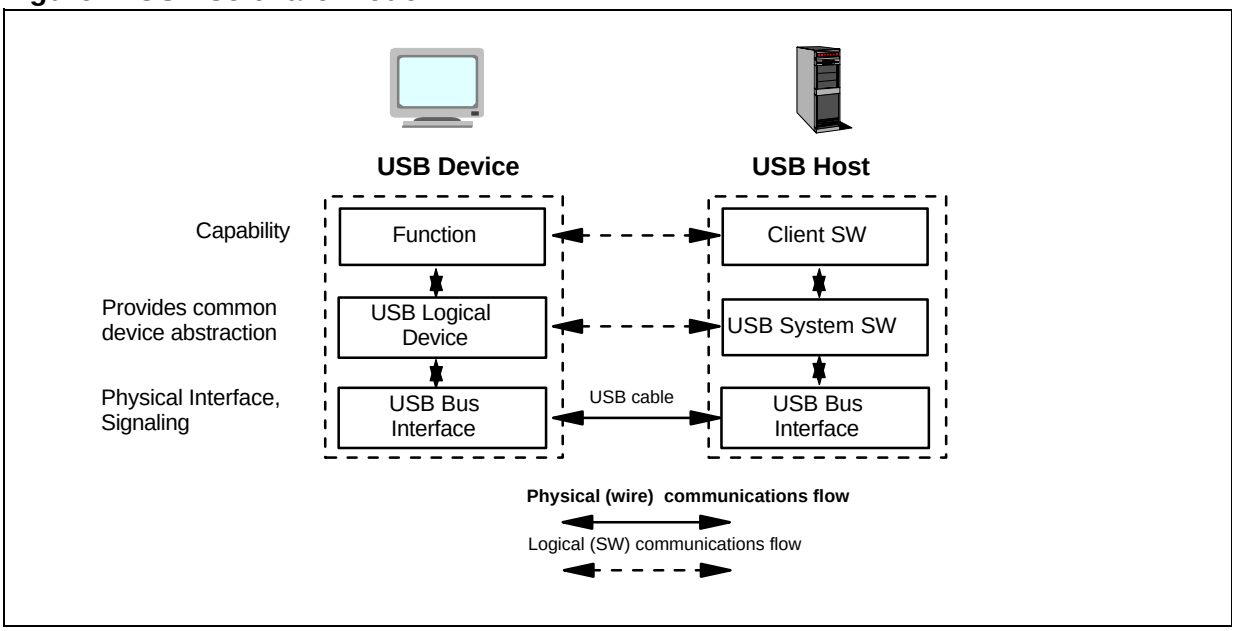
USB function layer: this firmware layer implements the functionality provided by the device.

USB logical device layer: this firmware layer implements all standard USB requests and low speed data transfers.

USB bus interface layer: this firmware layer is the interface between the USB logical device layer and the USB interface hardware.

The USB logical device is discussed in detailed in this application note.

Figure 2. USB software model



The following chapters provide flowcharts and description of the firmware routines needed to support the USB device operations:

- Device enumeration and configuration
- Data transfers
- Power management

2 DEVICE ENUMERATION AND CONFIGURATION

When a USB device is attached, the host issues a reset signal. When the reset signal is released, the device enters the unenumerated state.

2.1 USB RESET

The USB reset is independent from the chip reset. A USB reset signal resets the USB interface peripheral but not the ST7 core and other peripherals.

When a USB reset signal is detected on the bus, the RESET bit in the ISTR register is set and an USB interrupt is generated. All the USB interface registers are reset.

2.2 ENUMERATION

The host performs a bus enumeration to identify the attached device and to assign a unique address to it. The device responds to the requests sent by the host during the enumeration process on its default pipe (endpoint 0).

Enumeration steps:

1. Get device descriptor.

The host send a get device descriptor request. The device replies with its device descriptor to report its attributes (Device Class, maximum packet size for endpoint zero).

2. Set address

A USB device uses the default address after reset until the host assigns a unique address using the set address request. The firmware writes the device address assigned by the host in the DADDR register.

NOTE: The firmware must write the device address only after completion of the set address operation because the status stage that concludes the control transfer still uses the default address.

3. Get configuration

The host sends a get configuration. The device replies with its configuration descriptor, interface descriptor and endpoint descriptor. The configuration descriptor describes the number of interfaces provided by the configuration, the power source (Bus or Self powered) and the maximum power consumption of the USB device from the bus. The Interface descriptor describes the number of endpoints used by this interface. The Endpoint descriptor describes the transfer type supported and the bandwidth requirements.

Others Class-specific descriptors may be returned by the device depending on the function implemented.

4. Set Configuration

The host assigns a configuration value to the device based on the configuration information. The device is then in configured state and can draw the amount of power described in the configuration descriptor.

The device is now configured and ready to be used.

3 DATA TRANSFER

3.1 OVERVIEW

The ST7 USB interface can receive data on endpoint 0,1,2 and send data through endpoint 0,1,2 (endpoint 2 may not be available on all ST7 microcontrollers). The transfer types supported by the ST7 are:

1. Control transfer with endpoint 0 (SETUP, OUT and IN tokens)

2. Interrupt transaction with endpoint 1 (IN token)

Note: Transmission and reception on Endpoints 1 and 2 may be supported, depending on the microcontroller type and the conditional compilation options chosen (see CONDCOMP.H file).

3.1.1 DMA buffers

The received and sent data are stored in RAM buffers assigned to the DMA of the USB interface.

DMA buffers are a contiguous RAM area. The firmware must write the starting address of the DMA memory area in the DMAR register and in the DA7 and DA6 bits of the IDR register. The six least significant bits of the IDR register are managed by hardware. For detailed information on the mapping, see *USB interface* chapter of the ST7 data sheet.

3.1.2 CTR interrupt

When a correct transfer operation has been performed, a CTR interrupt (see CTR bit in the ISTR register) is generated. The following events generate a CTR interrupt:

1. ACK handshake returned by the host after a data has been sent in response to an IN token

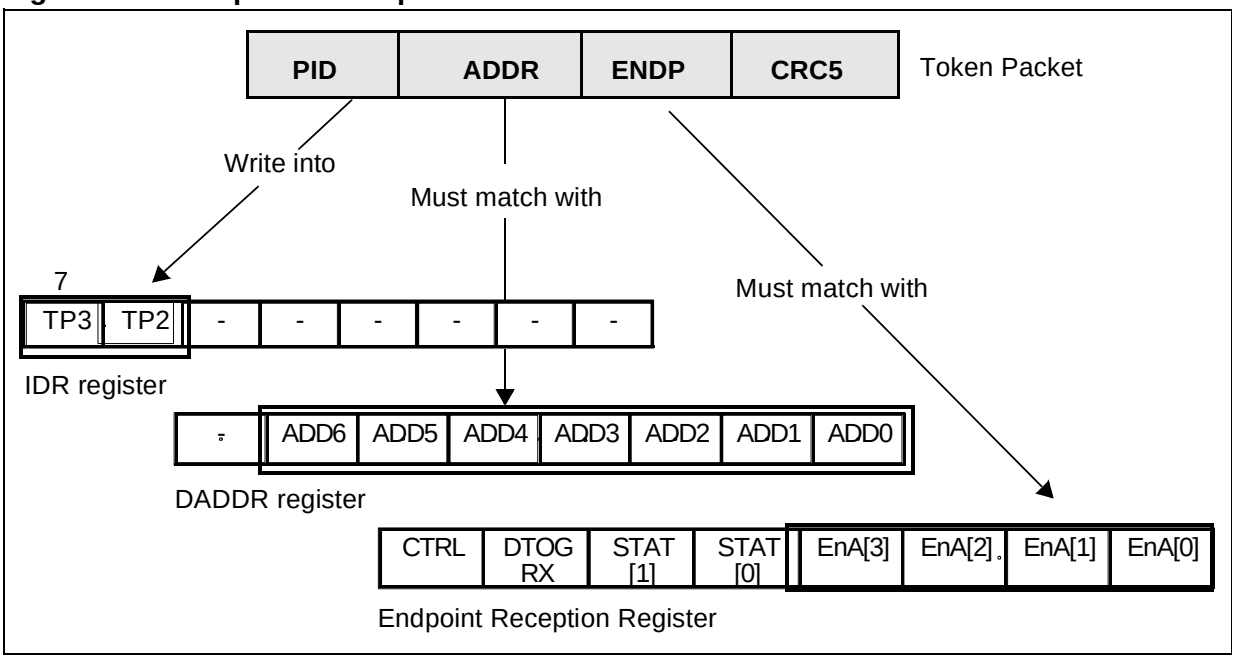
2. Valid SETUP or OUT token received followed by a data packet and ACK handshake returned to the host by hardware

Figure 3 describes the interaction between the received token packet and the hardware register of the ST7 USB interface. The token packet contains:

1. PID field specifying either IN, OUT or SETUP token

2. ADDR and ENDP fields identifying the endpoint that will receive the data packet following the token packet.

Figure 3. Token packet reception



The firmware must determine the data transfer direction and the endpoint number which has sent or received data by reading the IDR and PIDR registers.

The following table shows the data transfer direction corresponding to the each PID.

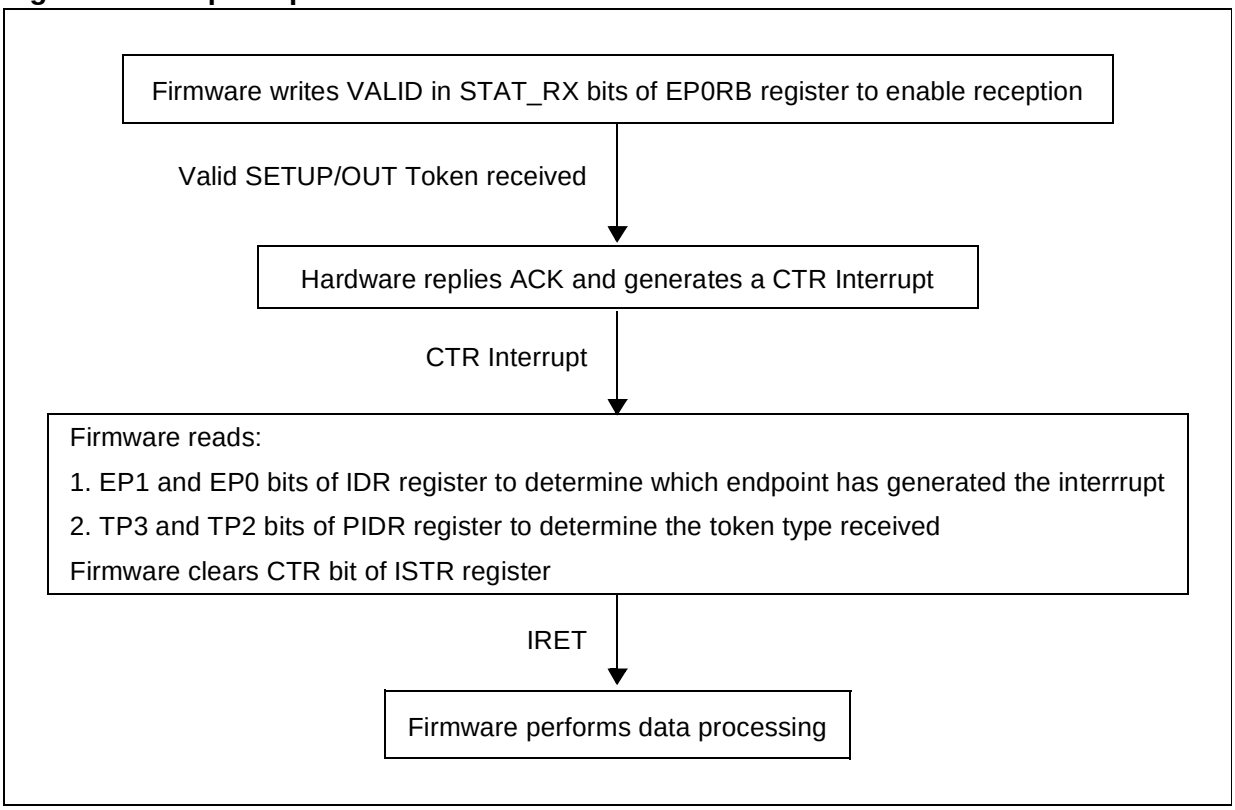
PID name	Data transfer direction
SETUP, OUT	from host to device
IN	from device to host

3.2 RECEIVE AND TRANSMIT PROCEDURES

3.2.1 Receive procedure on endpoint 0

Endpoint 0 receive data operation is explained in the figure 4.

Figure 4. Reception procedure



When the device receives a SETUP or an OUT PID, if the device address and the endpoint number contained in the SETUP or OUT packet match with the address written in DADDR register and one of the two endpoint numbers, the following operations are performed:

1. If the STAT_TX bits of the endpoint addressed are set to STALL, a STALL handshake is returned by hardware to the host.

2. If the STAT_TX bits of the endpoint addressed are set to NAK, a NAK handshake is returned by hardware to the host.

3. If the STAT_TX bits of the endpoint addressed are set to VALID, a DMA request on that endpoint is performed. Data are transferred in the DMA buffer. If the CRC is correct, an ACK handshake is returned by hardware to the host, DTOG_RX bit is toggled and the hardware disables the endpoint by setting the STAT_RX bits to 10 (NAK). The CTR bit in ISTR register is set, causing an interrupt if enabled.

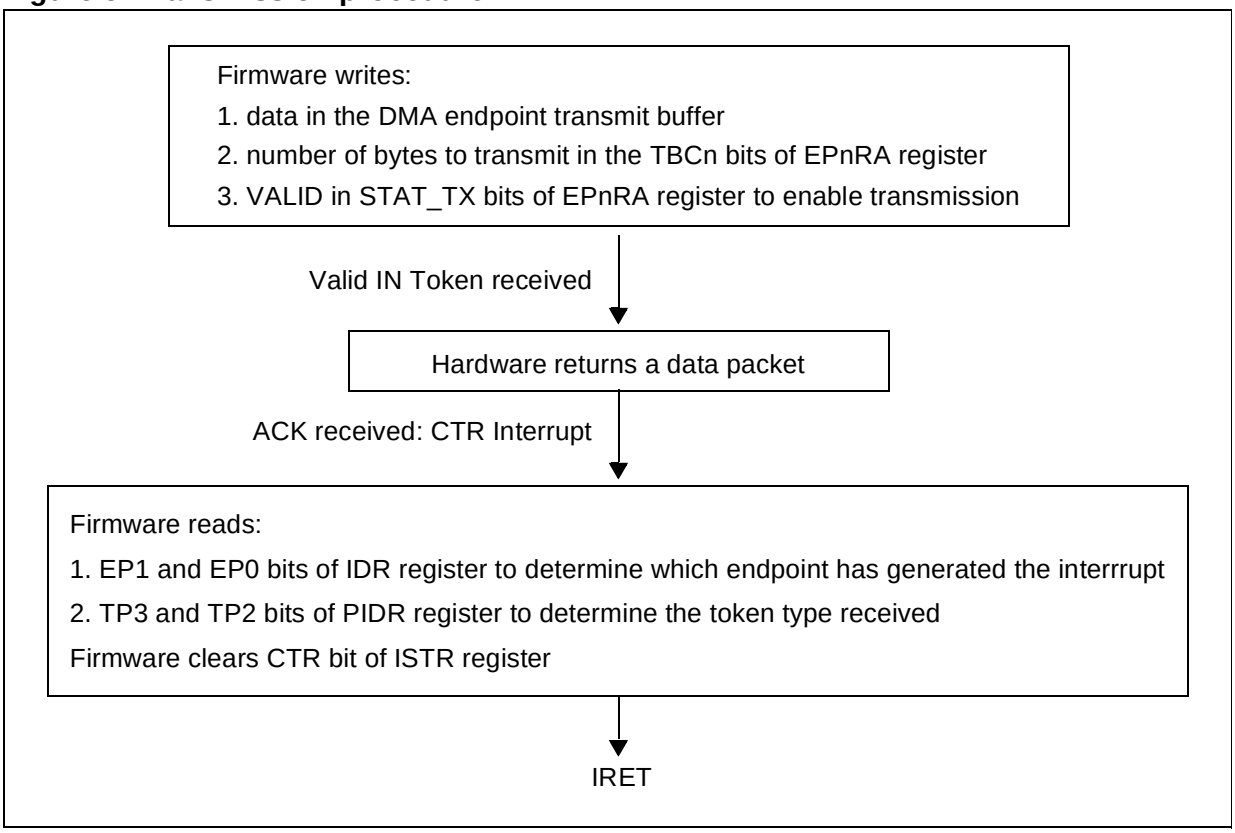
The following table summarizes the behavior of the USB interface when a SETUP/OUT token is received by endpoint 0.

STAT_RX1	STAT_RX0	Action performed by hardware when SETUP/OUT token received
0	0	No action. USB interface ignores the received token
0	1	STALL handshake returned
1	0	NAK handshake returned
1	1	1. Data received 2. STAT_RX0 reset 3. ACK handshake returned

3.2.2 Transmit procedure on endpoint 0 and endpoint 1

The transmit operation is explained in the figure 5.

Figure 5. Transmission procedure



When the device receives an IN PID, if the device address and the endpoint number contained in the IN packet match with the address written in DADDR register and one of the two endpoint numbers, the following operations are performed:

1. If the STAT_TX bits of the endpoint addressed are set to STALL, a STALL handshake is returned by hardware to the host.

2. If the STAT_TX bits of the endpoint addressed are set to NAK, a NAK handshake is returned by hardware to the host.

3. If the STAT_TX bits of the endpoint addressed are set to VALID, a DMA request on that endpoint is performed. A DATA0 or DATA1 data packet is sent according to the DTOG_TX bit in EPnRA register. After the last byte in the DMA buffer is sent, the CRC is sent by hardware. Upon receipt of ACK handshake from the host, DTOG_TX bit is toggled and the hardware disables the endpoint by setting the STAT_TX bits to 10 (NAK). The CTR bit in ISTR register is set, causing an interrupt if enabled.

The following table summarizes the behavior of the USB interface when a IN token is received by endpoint 0.

STAT_TX1	STAT_TX0	Action performed by hardware when IN token received
0	0	No action. USB interface ignores the received token
0	1	STALL handshake returned
1	0	NAK handshake returned
1	1	1. Data sent 2. STAT_TX0 reset

3.3 DATA TRANSFER WITH ENDPOINT 0

Endpoint 0 supports control transfers. There are two types of control transfers:

1. Control transfer with data phase

2. No-data control transfer.

As a consequence, a control transfer may have three transaction stages:

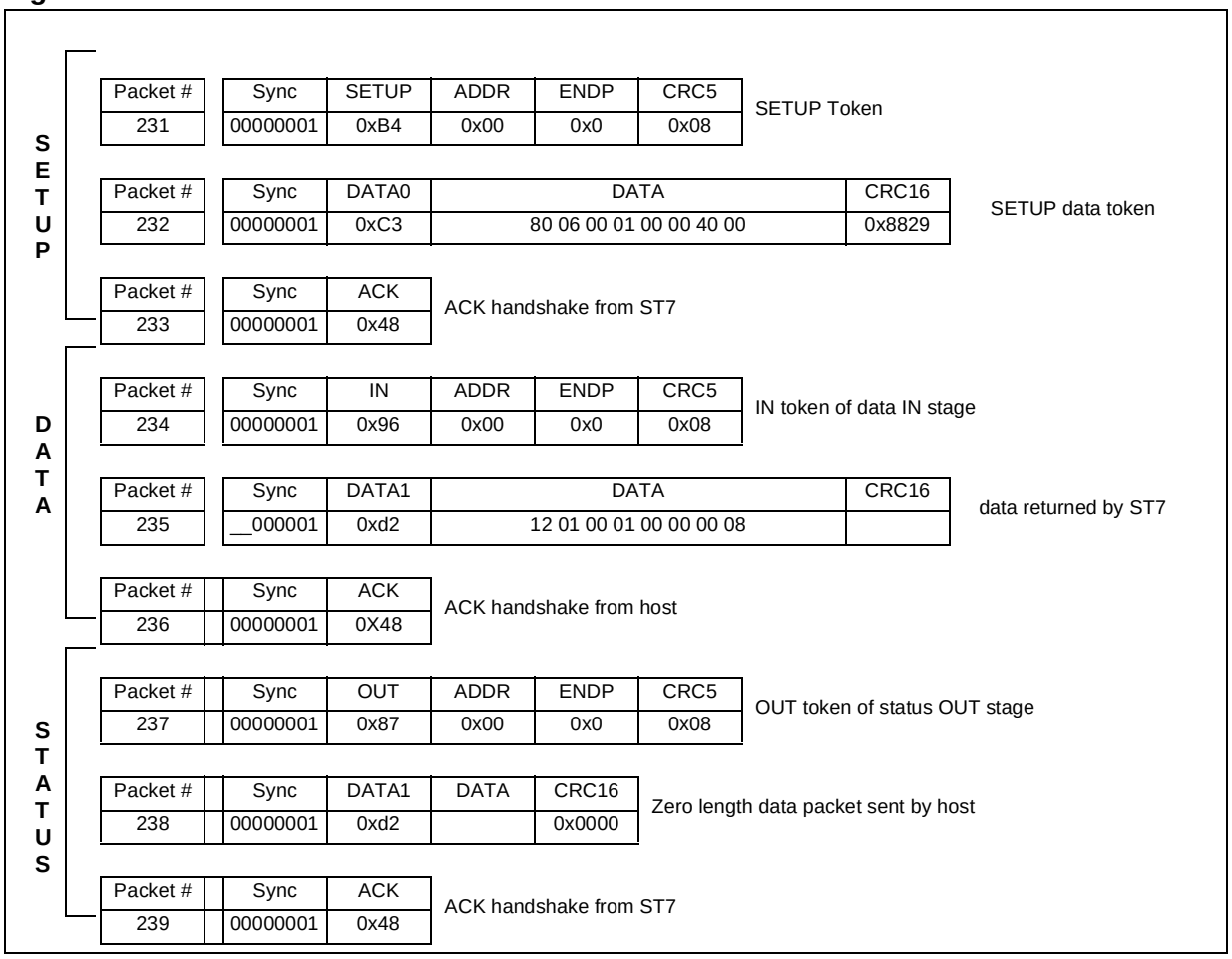
1. Setup Stage

2. Data Stage (not for no-data control transfer)

3. Status Stage

The figure 6 shows an example of control transfer with one data IN stage (only 8 data bytes are exchanged).

Figure 6. Control transfer



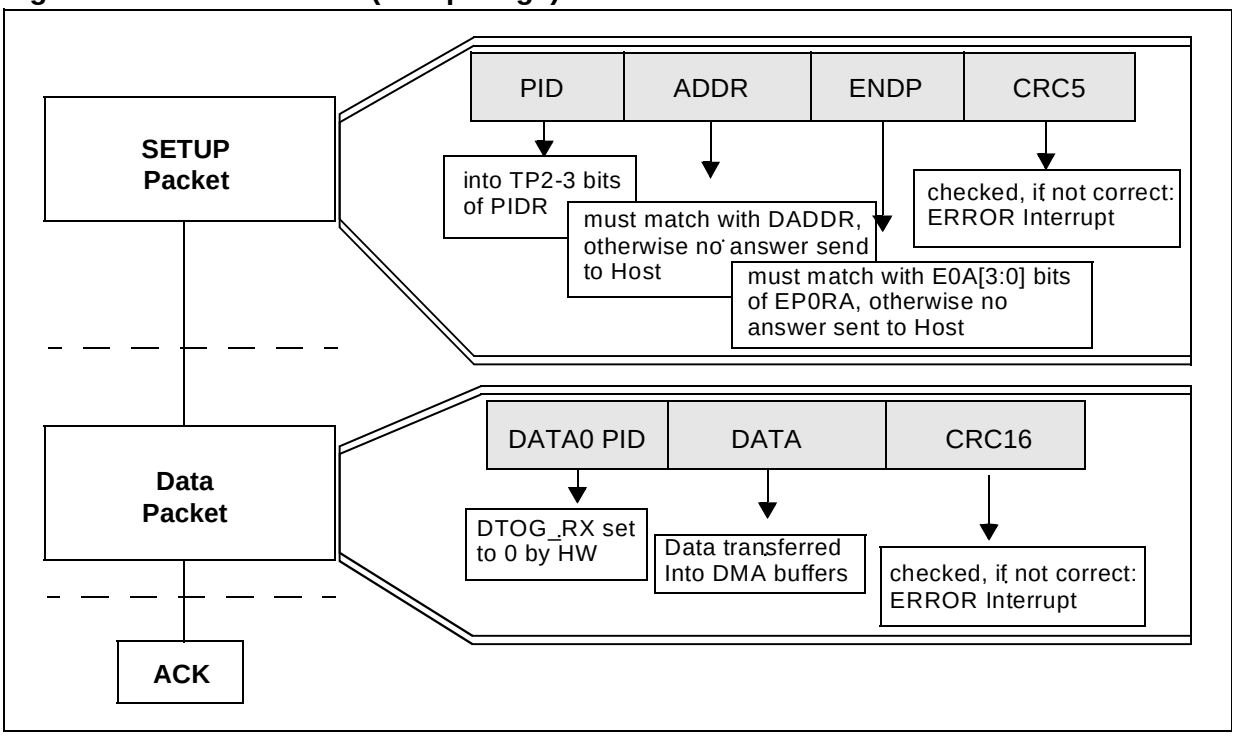
3.3.1 Setup stage

The host sends a setup token followed by a data field of 8 bytes. Endpoint 0 must always be able to accept a setup token. To support this requirement, the status bits STAT_RX0 and STAT_RX1 of the EP0RB register must be re-write to VALID after every data reception on endpoint 0.

The hardware interface ACKs the valid setup token and generates a Correct TRansfer (CTR) USB interrupt.

In the figure 7 endpoint number is 0 and the PID type is SETUP.

Figure 7. Control transfer (Setup stage)



The firmware must decode the 8 bytes received in the data packet following the SETUP token. The following table describes the 8 bytes of a setup data packet.

Offset	Field	Size	Description
0	bmRequestType	1	Used by firmware to determine the data xfer direction, the type of the request (standard, class, vendor) and the recipient of the request (device, interface, endpoint).
1	bRequest	1	Request code.
2	wValue	2	Varies according to the request.
4	wIndex	2	Varies according to the request.
6	wLength	2	If = 0, no-data control transfer. Used by firmware to know how many bytes have to be sent or received by the device.

Parsing these bytes, the firmware determines:

1. If there is a data phase looking at the wLength value

0 = No-data control transfer

2. The direction of the following data phase (if present) looking at bit 7 of the bmRequestType:

0 = Host to device (OUT data stage)

1 = Device to host (IN data stage)

If no data stage follows the setup stage, the firmware gets ready to perform the status stage.

3.3.2 Data stage

The data stage can be either from the host to the device or from the device to the host. It consists of one or more IN or OUT transactions. The amount of data to be sent during the data stage and its direction are specified during the setup stage. If, for instance, the amount of data to be transferred is 18 bytes, the data stage will have two data transfer of 8 bytes (maximum data payload) and a data transfer of 2 bytes.

3.3.3 Status stage

The status stage of a control transfer is the last operation in the sequence. It can be detected by the device when there is a change in direction of data flow from the previous stage.

The status stage reports to the host the outcome of previous setup and data stages of the transfer.

The status stage can use either IN or OUT tokens. For a control read (using IN token during data stage), the host send an OUT token followed by a zero length data packet and waits for the ACK handshake from the device. For a control write (using OUT token during data stage), the host send a IN token, the device returns a zero length data packet and the host returns an ACK handshake.

3.4 DATA TRANSFER WITH ENDPOINT 1

Endpoint 1 supports interrupt transaction.

3.4.1 Interrupt Transaction

When the function has some data to return to the host through the interrupt pipe, it must write this data in the DMA buffer and enable endpoint 1 in transmission by setting the EP1RA to VALID. The host polls endpoint 1 with a polling interval given in the endpoint descriptor by sending an IN token. The hardware interface replies with STALL, NAK or data.

4 POWER MANAGEMENT

The ST7 supports the following USB power management features:

• suspend/resume

• remote wake-up

4.1 SUSPEND/RESUME OPERATIONS

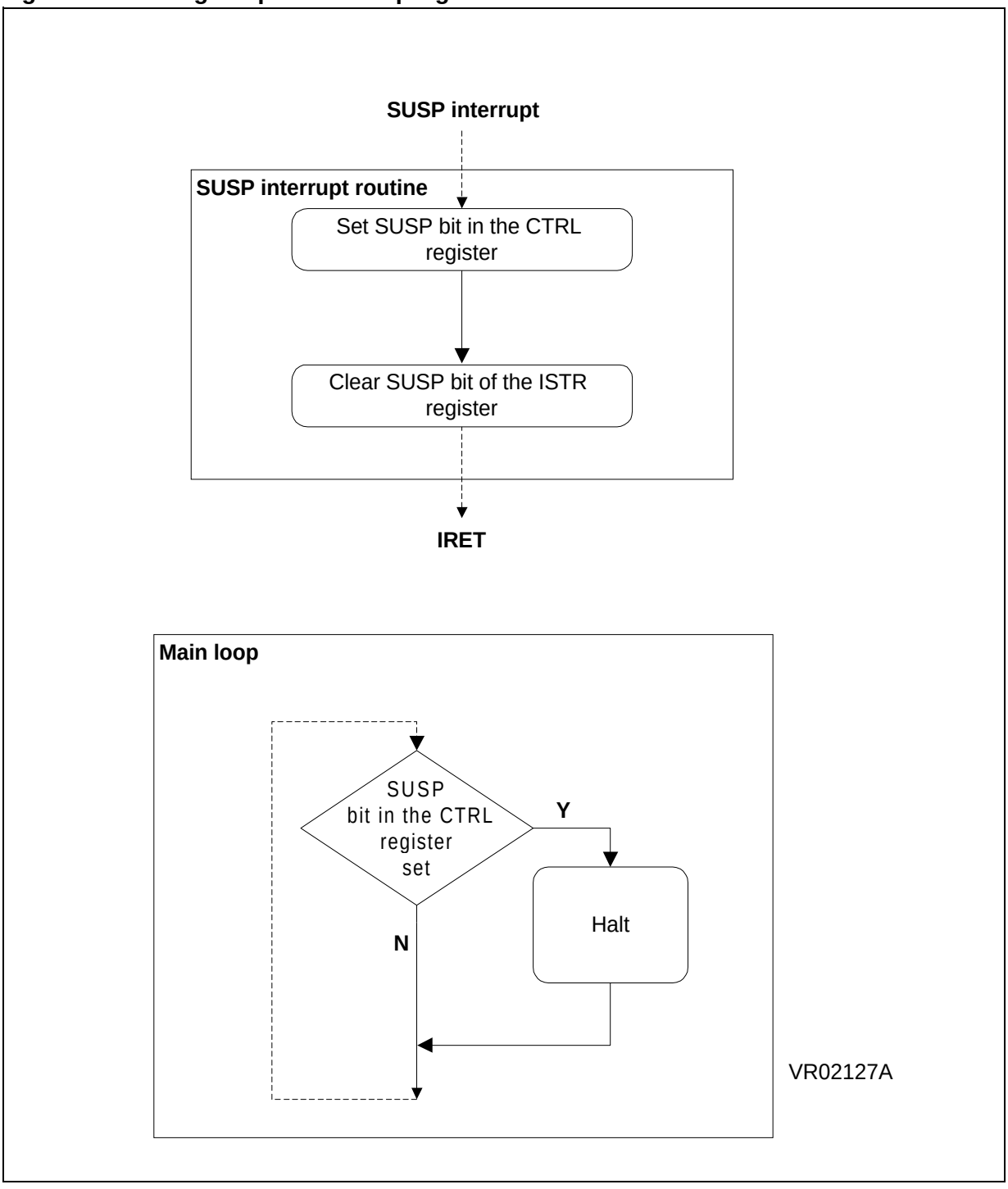
4.1.1 Entering suspend state

When no activity is seen on the bus for more than 3 ms, the ST7 enters suspend state. This functionality is implemented through the USB SUSP interrupt.

When this interrupt occurs the corresponding bmUsbIntFlag bit is set and the firmware must then set the SUSP bit in CTRL register to enter suspend state.

The firmware main loop checks if the SUSP bit is set to Halt the microcontroller in order to meet the power requirement of the USB specification. The whole application must not draw more than 500 µA from the USB bus in suspend state (only in bus powered applications).

Figure 8. Entering suspend mode program



4.1.2 Exiting suspend state

The microcontroller exits halt mode when one of the three following events occur.

1. Resume signal on the USB bus. The microcontroller exits halt mode and jumps to the USB end suspend interrupt routine.

2. USB reset on the USB bus. The microcontroller exits halt mode and jumps to the USB reset interrupt routine.

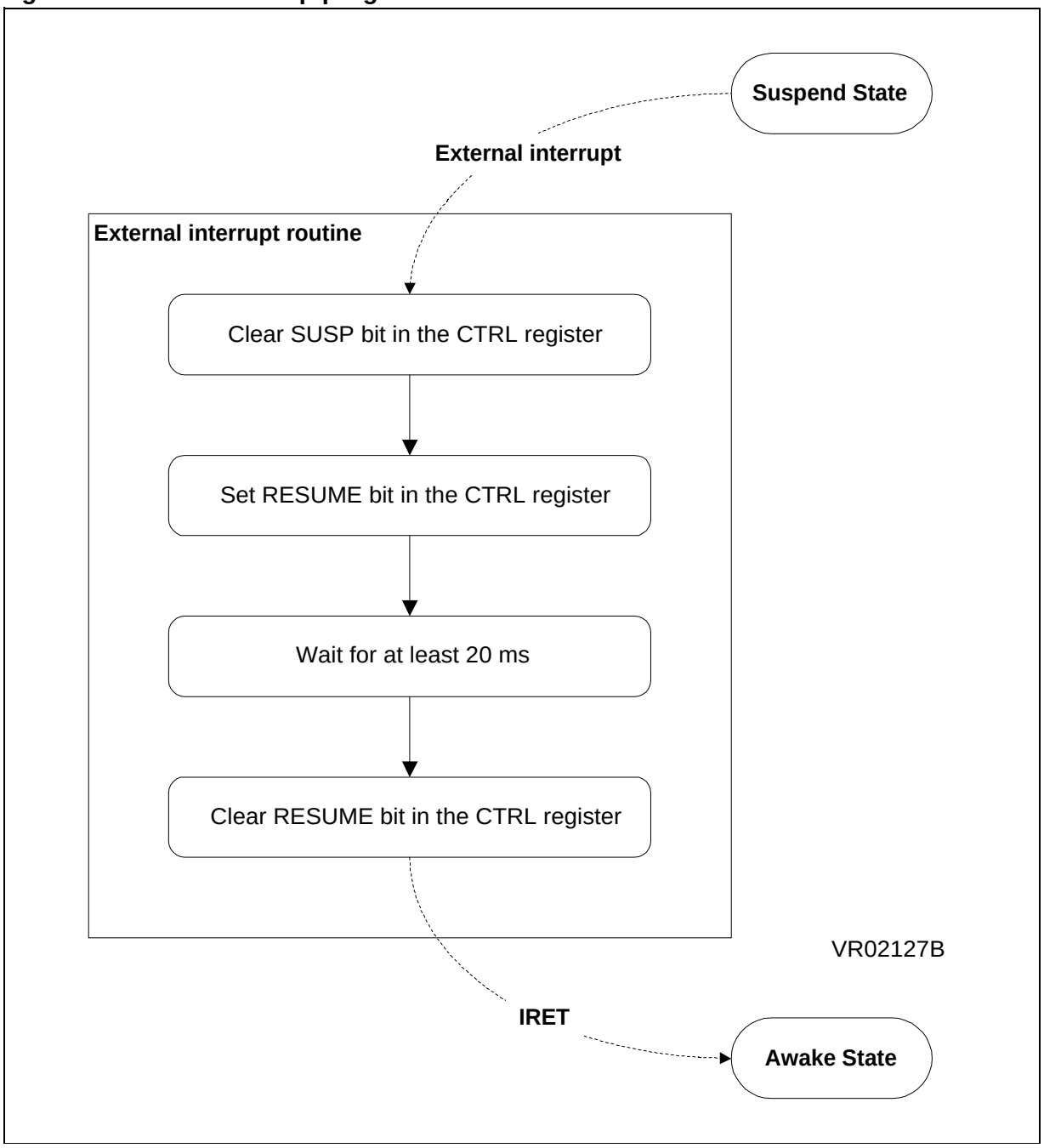
3. An external interrupt event is generated through an I/O port pin by the application. Then the microcontroller exits from Halt mode and jumps to the corresponding interrupt routine.

4.2 REMOTE WAKE-UP

When the ST7 is in suspend state and an enabled external interrupt occurs, it can initiate a resume signaling to wake up the system.

The function restarts the CPU clocks and executes the external interrupt routine. Within this external interrupt routine, the firmware clears the SUSP bit in the CTRL register and sets the RESUME bit in the CTLR register to force the resume signal on the USB bus. The firmware must keep the RESUME bit set for at least 20 ms.

Figure 9. Remote wake-up program



5 PROGRAM FLOW

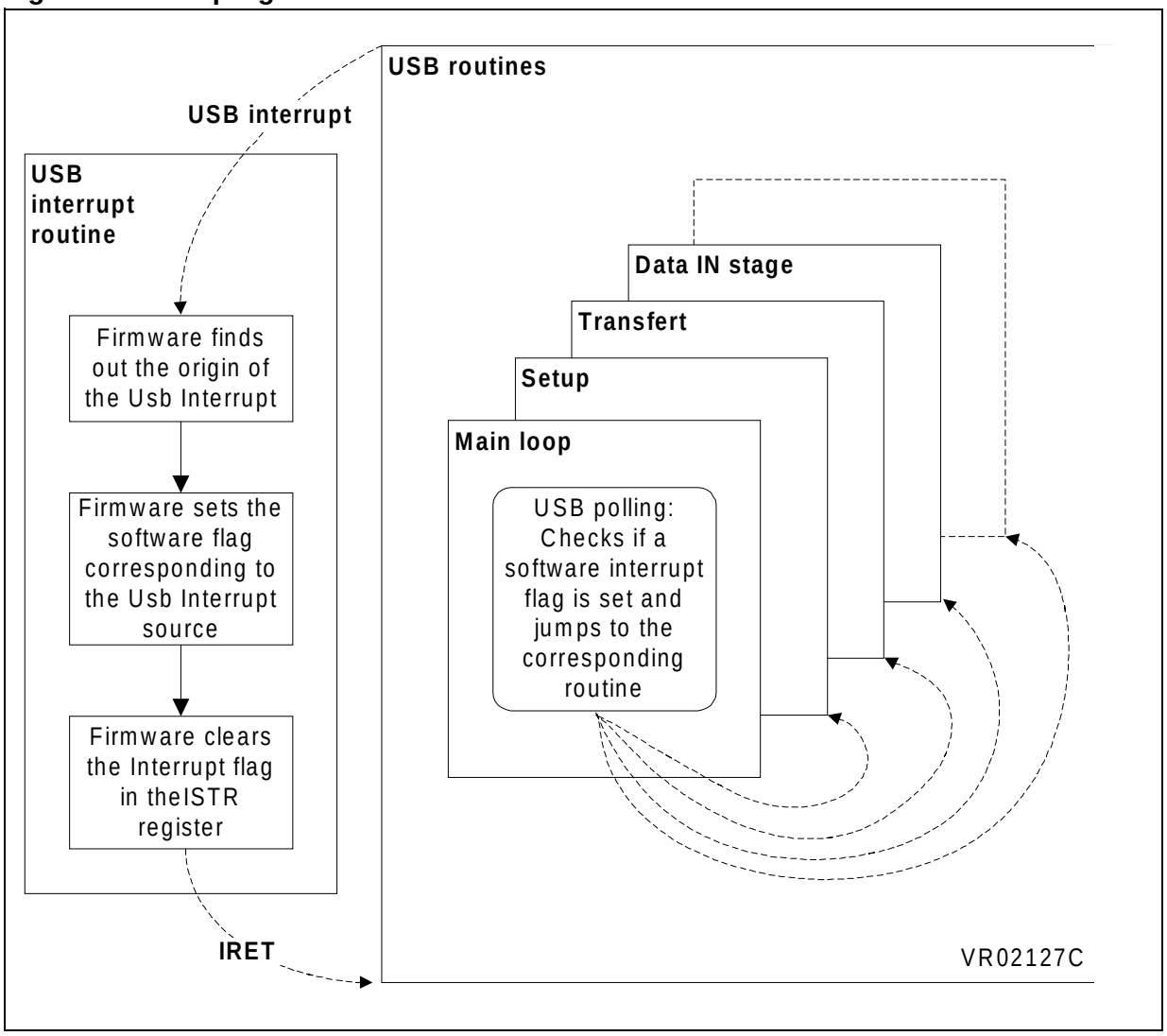
This chapter provides flowcharts of a low speed USB interface basic driver. These flowcharts describe the operations that the firmware must do to support data transfers used by standard USB requests.

5.1 PROGRAM ARCHITECTURE

The USB events are managed by interrupt. A single interrupt vector is used for the USB interrupt sources. The firmware must determine the interrupt origin by reading the ISTR register, set a bit in a software register and clear the interrupt flag.

The USB polling routine reads the software register to determine the USB interrupt source and jump to the corresponding interrupt routine.

Figure 10. USB program architecture



Before entering the main loop, the firmware must initialize the USB interface hardware.

5.2 USB INITIALIZATION

This routine initializes the hardware and resets the software registers to enable the USB interface. The initialization steps are:

1. Set the starting address of the endpoint DMA buffers

2. Enable endpoint 0 in reception

3. Write interrupt mask register

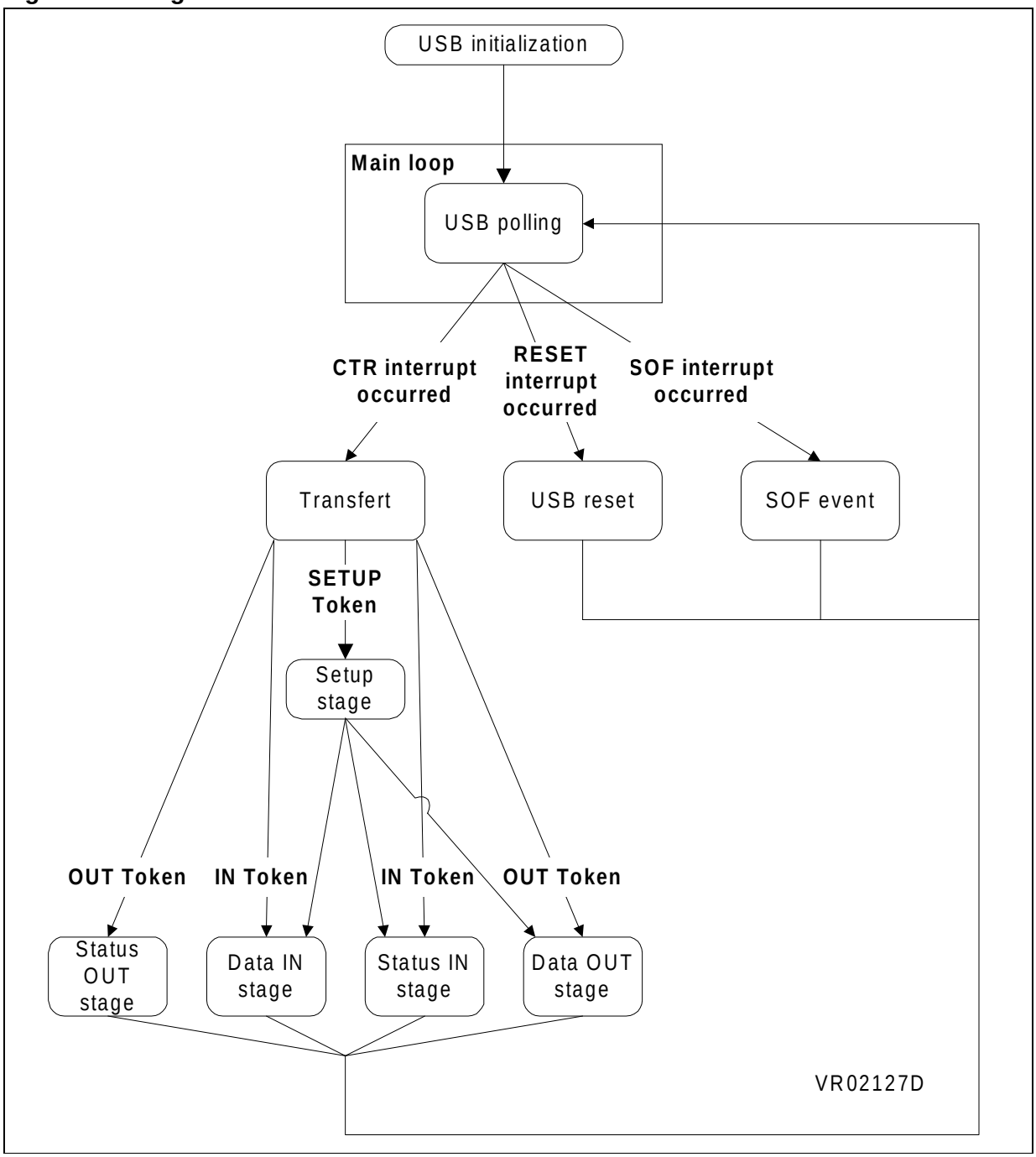
4. Power ON the internal 3.3V regulator to supply the external pull-up resistor used for detection by a hub.

5. Enable interrupts

5.3 USB POLLING

This routine is in the main loop and checks if a bit in the software interrupt register is set and jumps to the corresponding interrupt routine. The following figure describes the interaction between the USB polling routine in the main loop and the other USB routines.

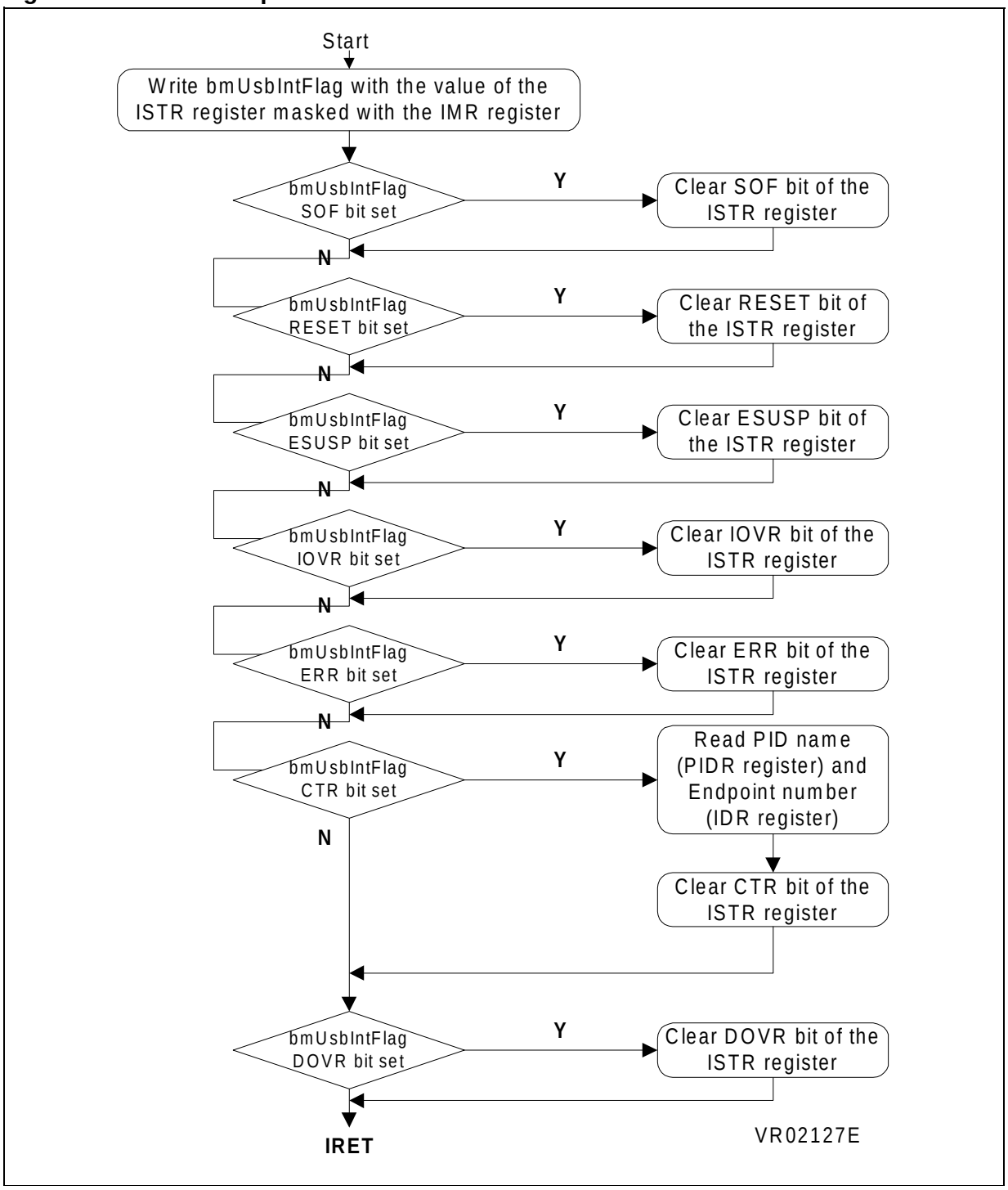
Figure 11. Program flow overview



5.4 USB INTERRUPT

The USB interrupt routine copies the ISTR register masked by the IMR register value to a software register and clears the bit of the ISTR register corresponding to the pending interrupt. The USB polling routine in the main loop checks if an interrupt software flag is set to execute the corresponding interrupt routine.

Figure 12. USB interrupt routine



5.5 TRANSFER

This routine is called by USB polling when the CTR (Correct TRansfer) bit in the ISTR register has been set. It determines which type of token has been received and by which endpoint number (zero or one) and jumps to the routine corresponding to the current USB event: setup stage, data IN stage, data OUT stage, status IN stage, status OUT stage on endpoint 0.

Figure 13. Transfer routine

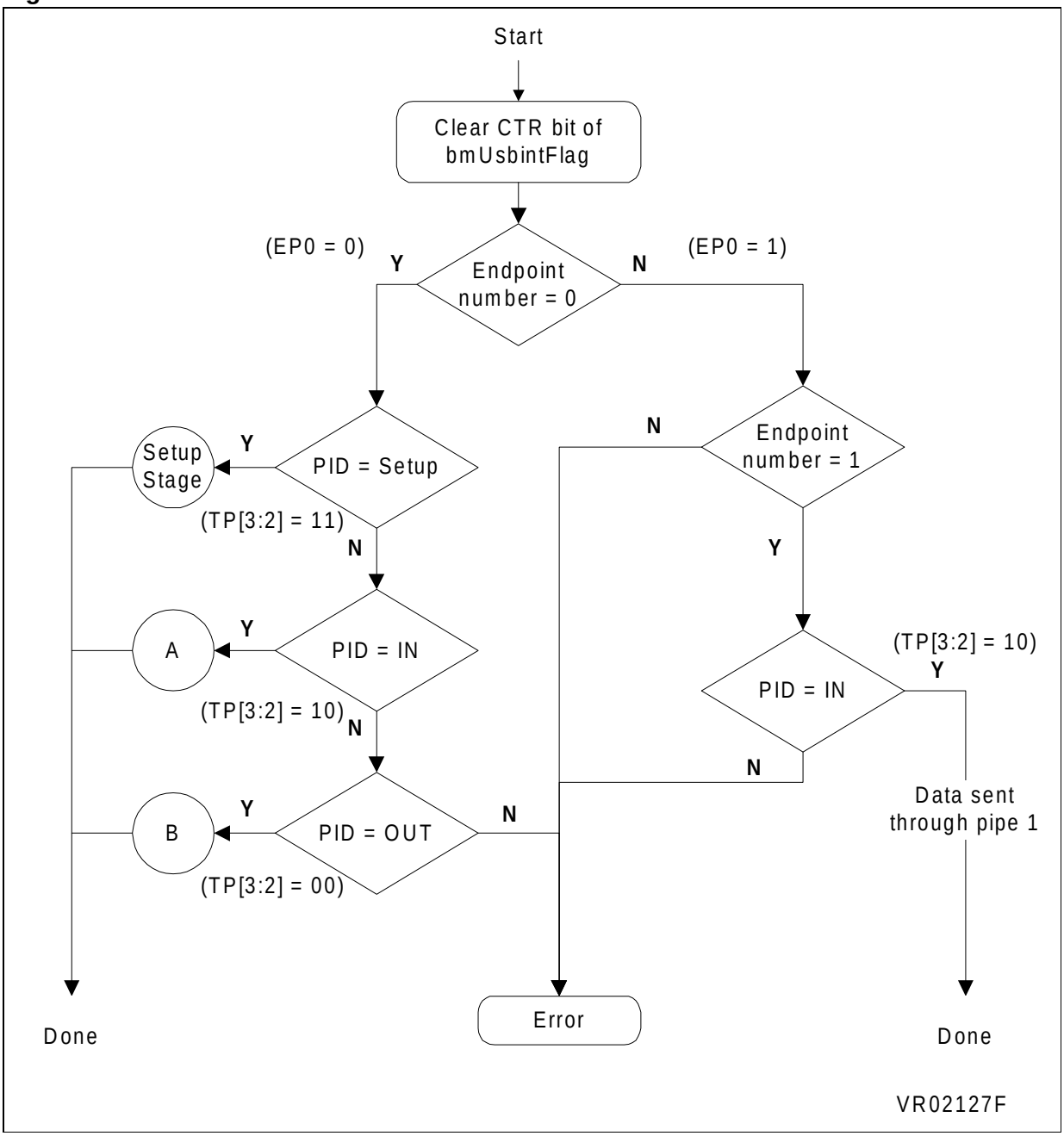


Figure 14. IN Token received by endpoint 0

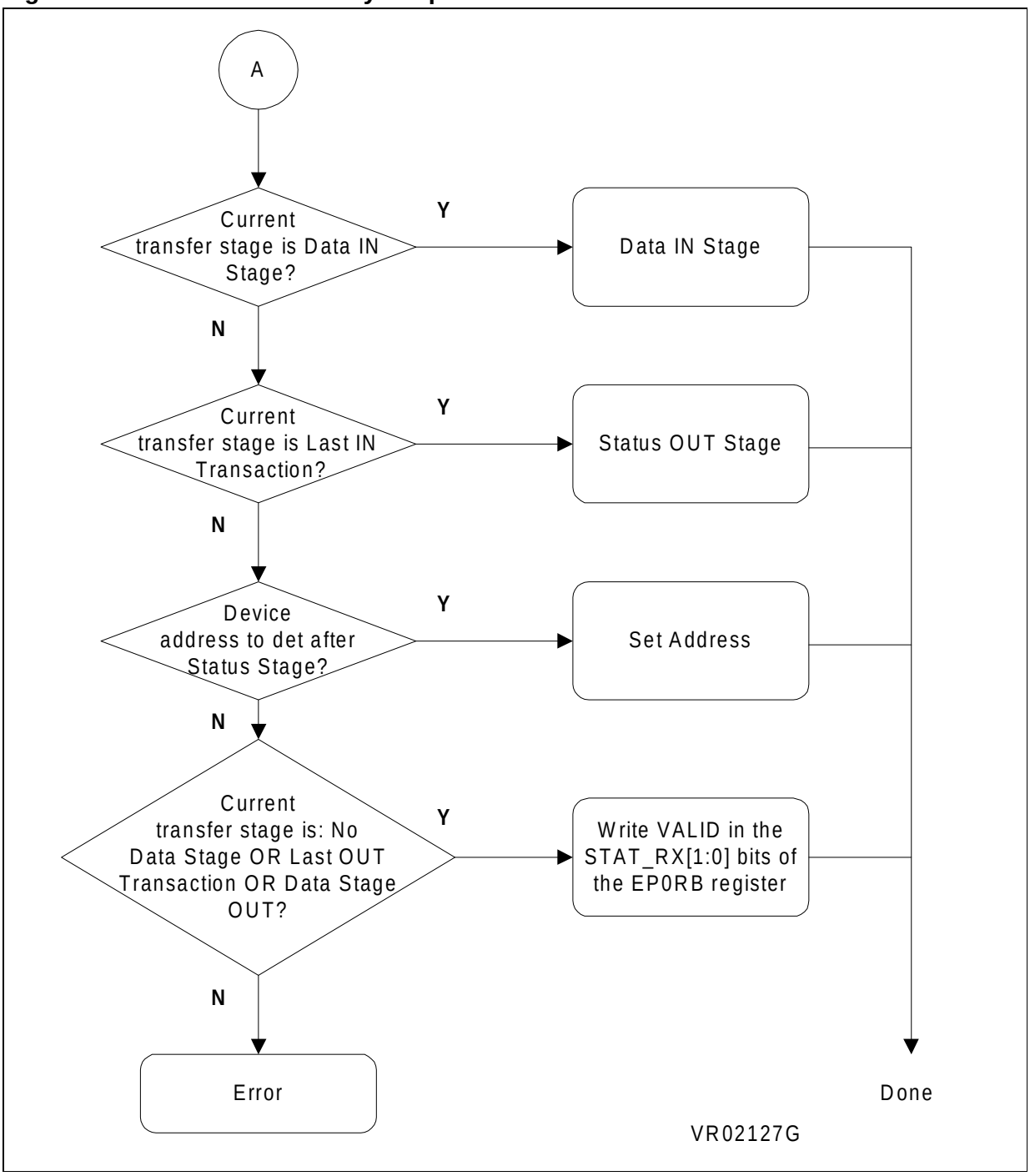
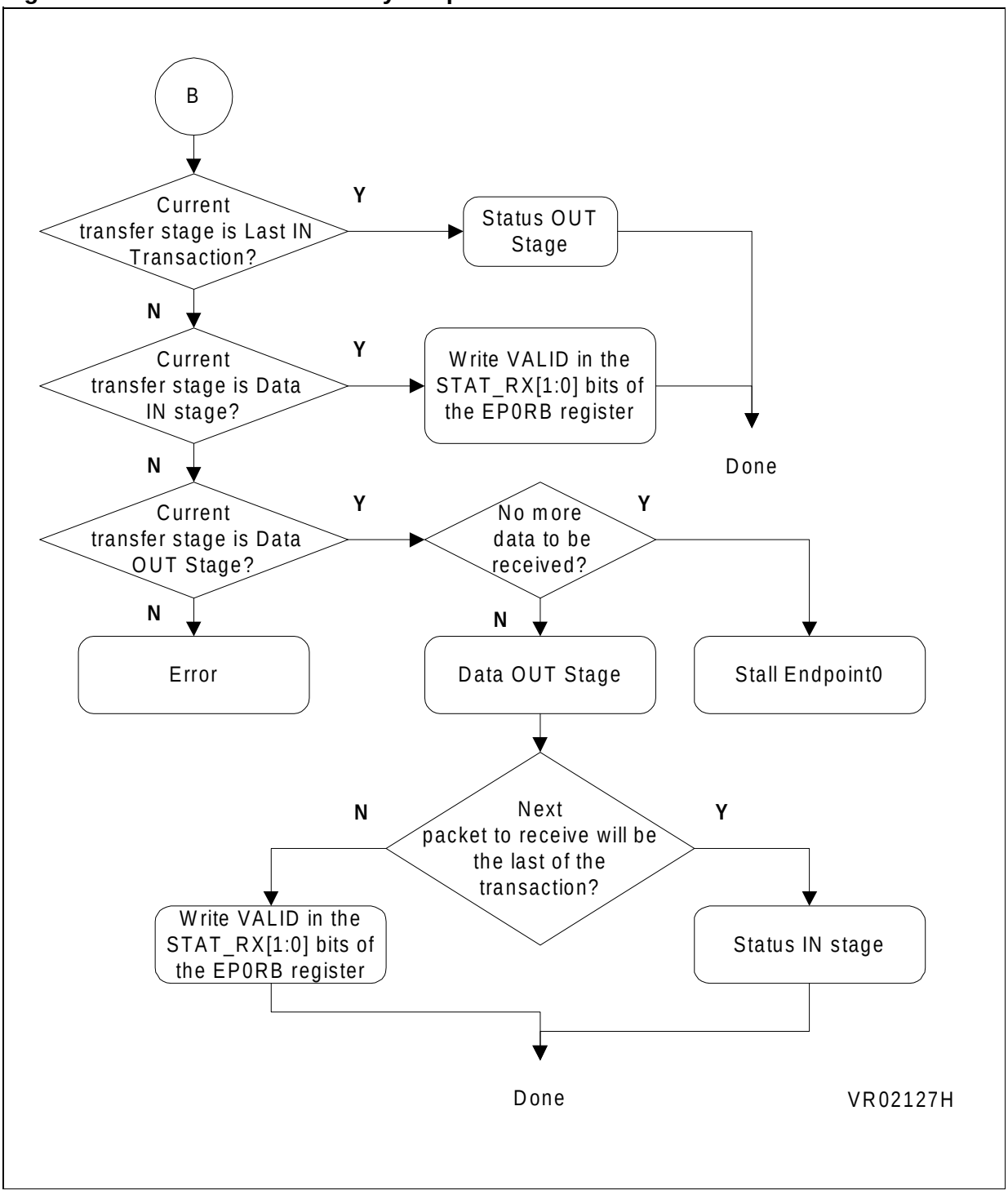


Figure 15. OUT Token received by endpoint 0



5.5.1 Setup stage

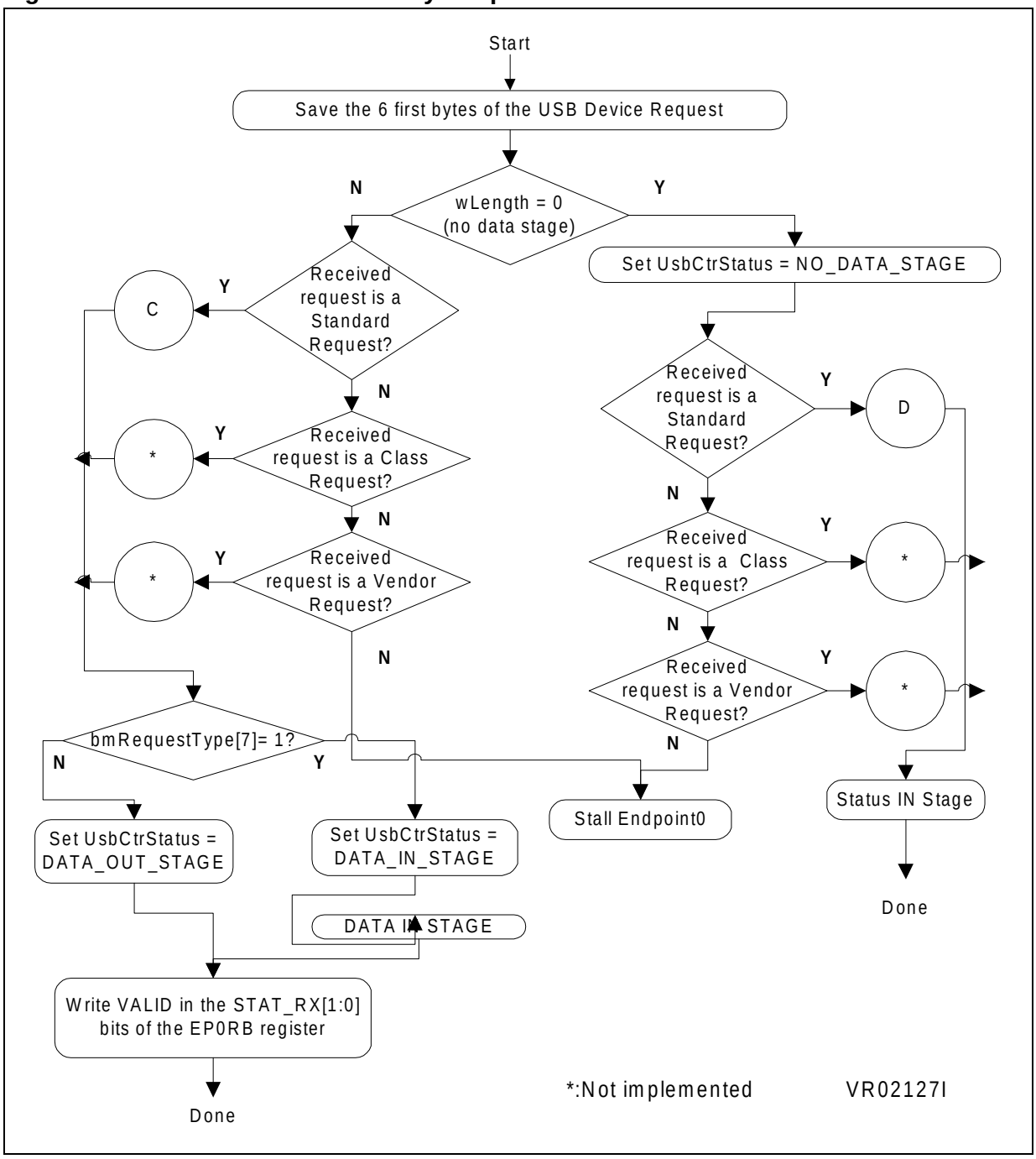
When a setup token is received, the firmware:

1. Saves the next six bytes after the setup token
2. Checks if the control transfer has a data stage or not
3. Decodes the request sent by the host

If the data transfer direction is from the device to the host, the firmware prepares data to be returned when the host sends an IN token, otherwise it sets the hardware ready to receive data from the host.

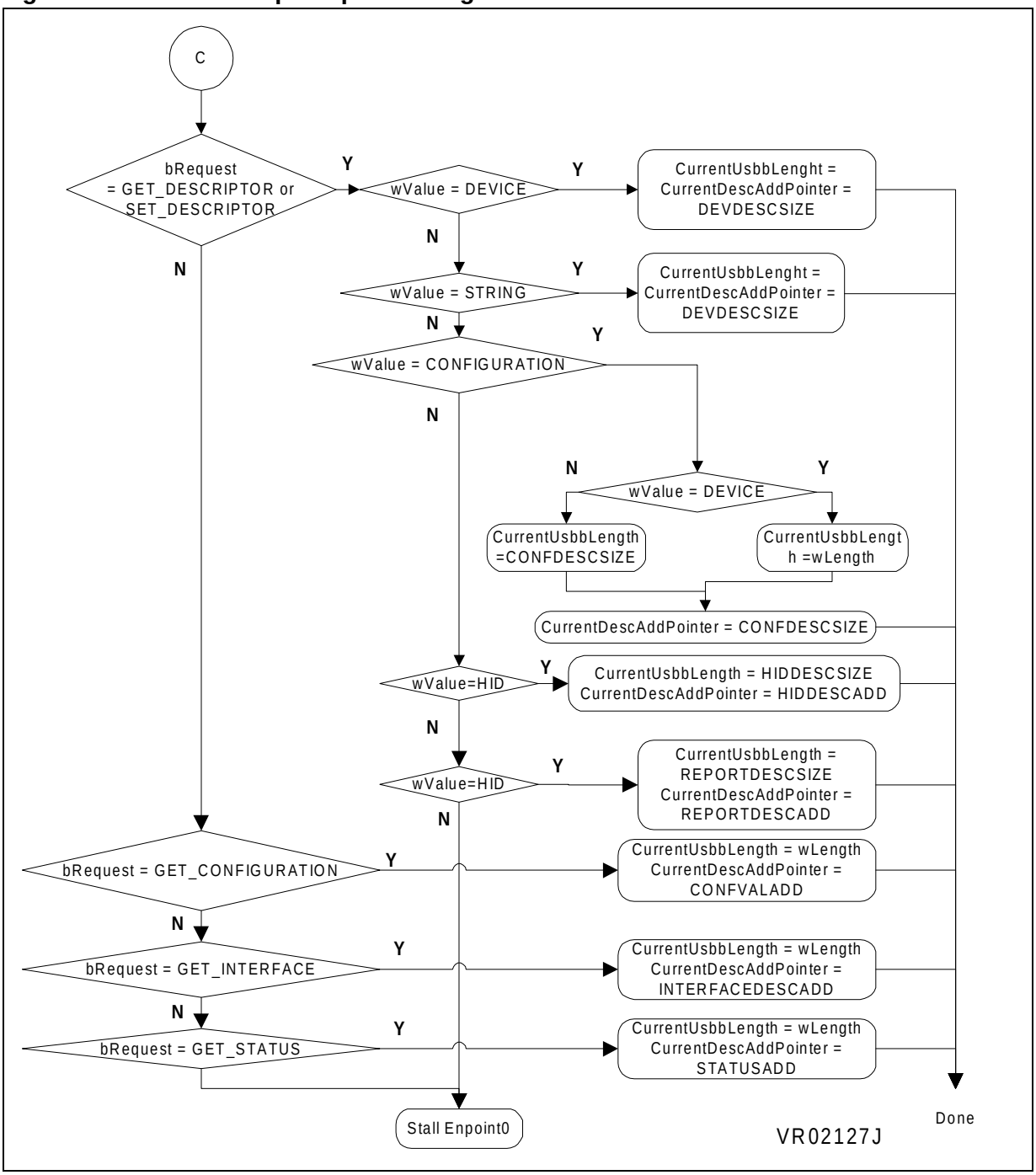
If there is no data transfer, the program jump to the status IN routine.

Figure 16. SETUP Token received by endpoint 0



See chapter 9 *USB Device Framework* of the USB specification for detailed information on bmRequestType and wLength.

Figure 17. Standard request processing with data transfer



See chapter 9 *USB Device Framework* of the USB specification for detailed information on bRequest, wValue and wLength.

When the GET_DESCRIPTOR, GET_CONFIGURATION requests are received the function must return the appropriate descriptor.

If a descriptor has a length longer than eight bytes, multiple data stages are needed. Two variables are used during the control transfer to return a descriptor stored in ROM to the PC host:

1. CurrentUsbbLength specifies the current total number of bytes to transmit.

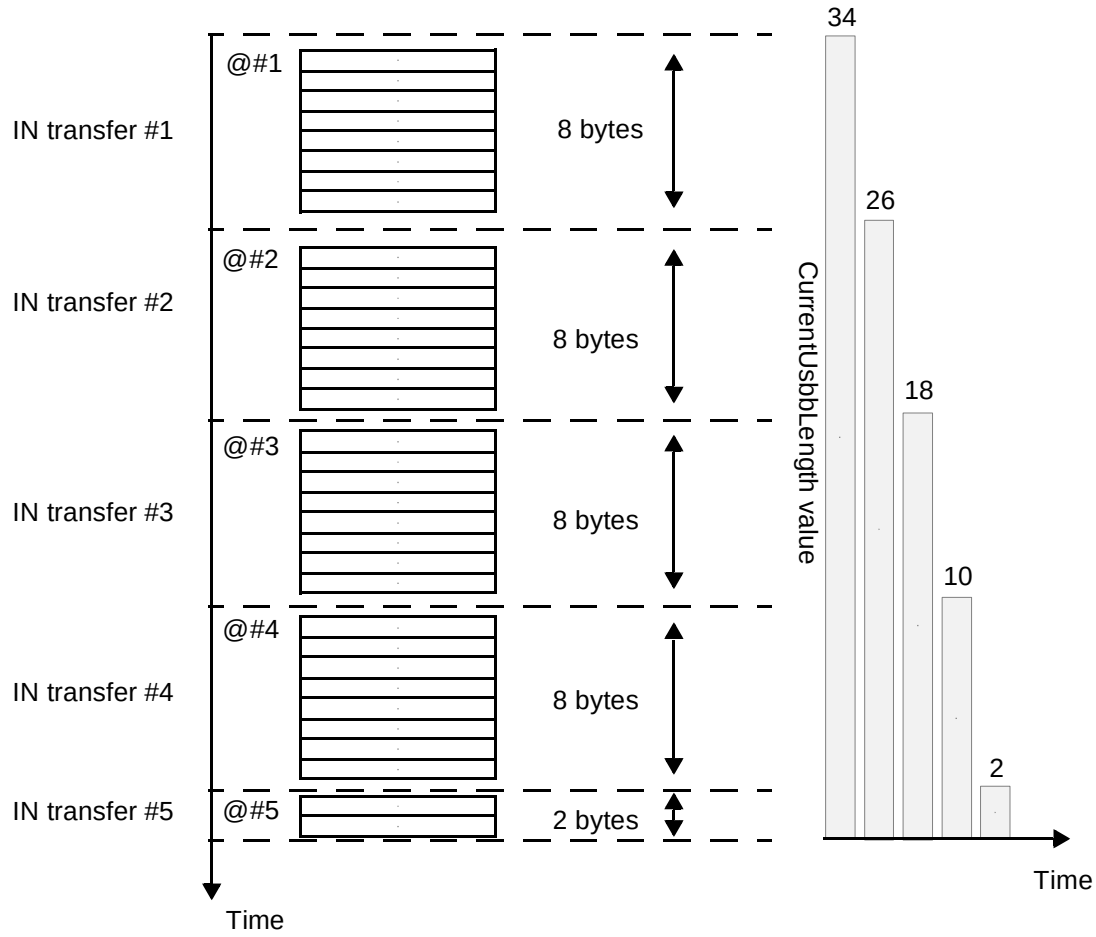
2. CurrentDescAddPointer specifies the current position of the descriptor pointer.

For example, if a descriptor is 34 bytes long, four data IN stages of 8 bytes and one of 2 bytes are needed to return the descriptor.

CurrentUsbbLength is initialized to 34 and CurrentDescAddPointer indicates the starting position in the array containing the descriptor before the first IN transfer.

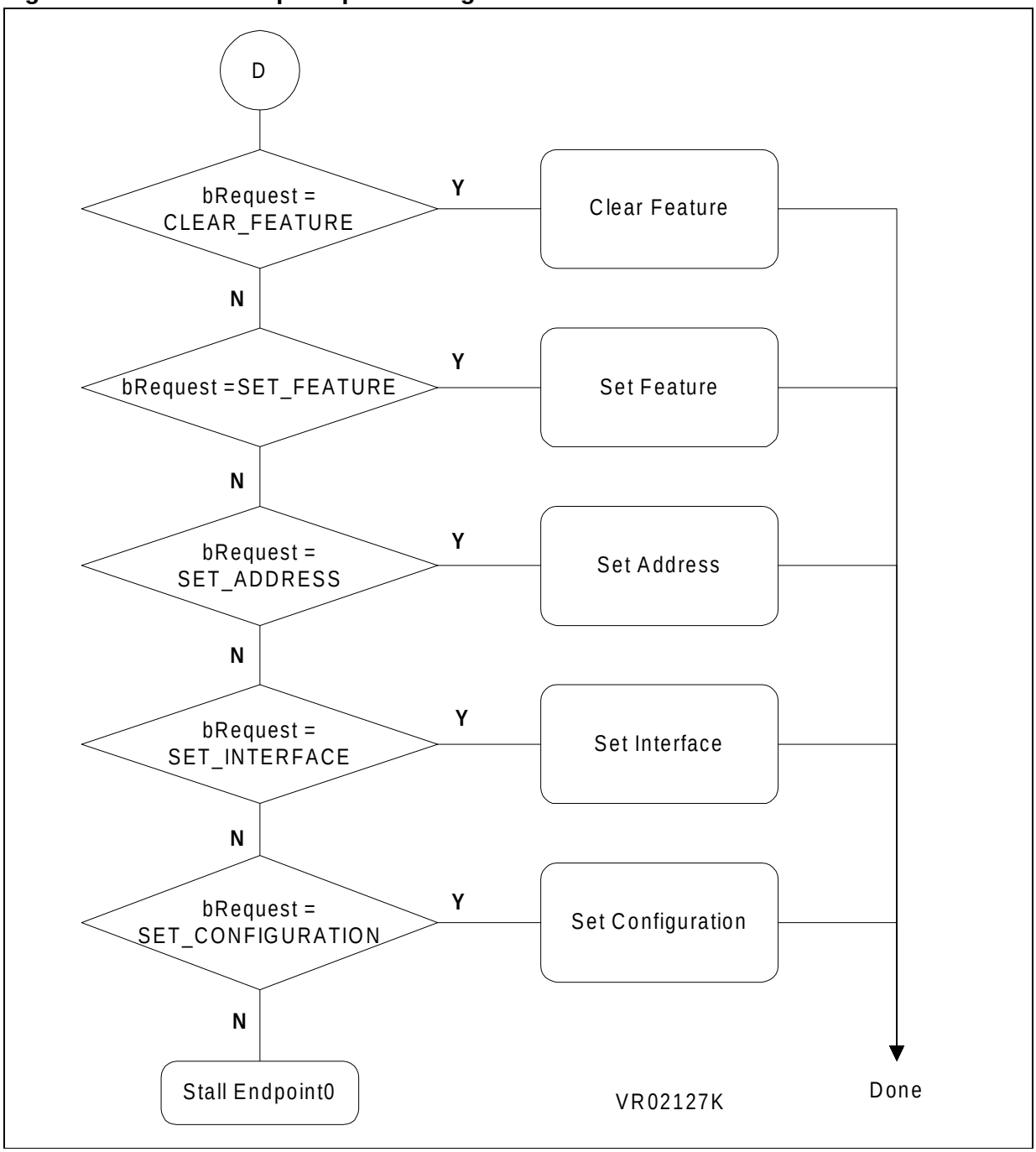
After every data IN stage, CurrentUsbbLength and CurrentDescAddPointer are decreased in order to indicate the current byte number remaining and the current position of the pointer in the descriptor array respectively.

Figure 18. Example of a 34 bytes long descriptor transfer



CurrentDescAdpointer will point to address @#1 for the first IN transfer, then to address @#2 for the second IN transfer till address @#5

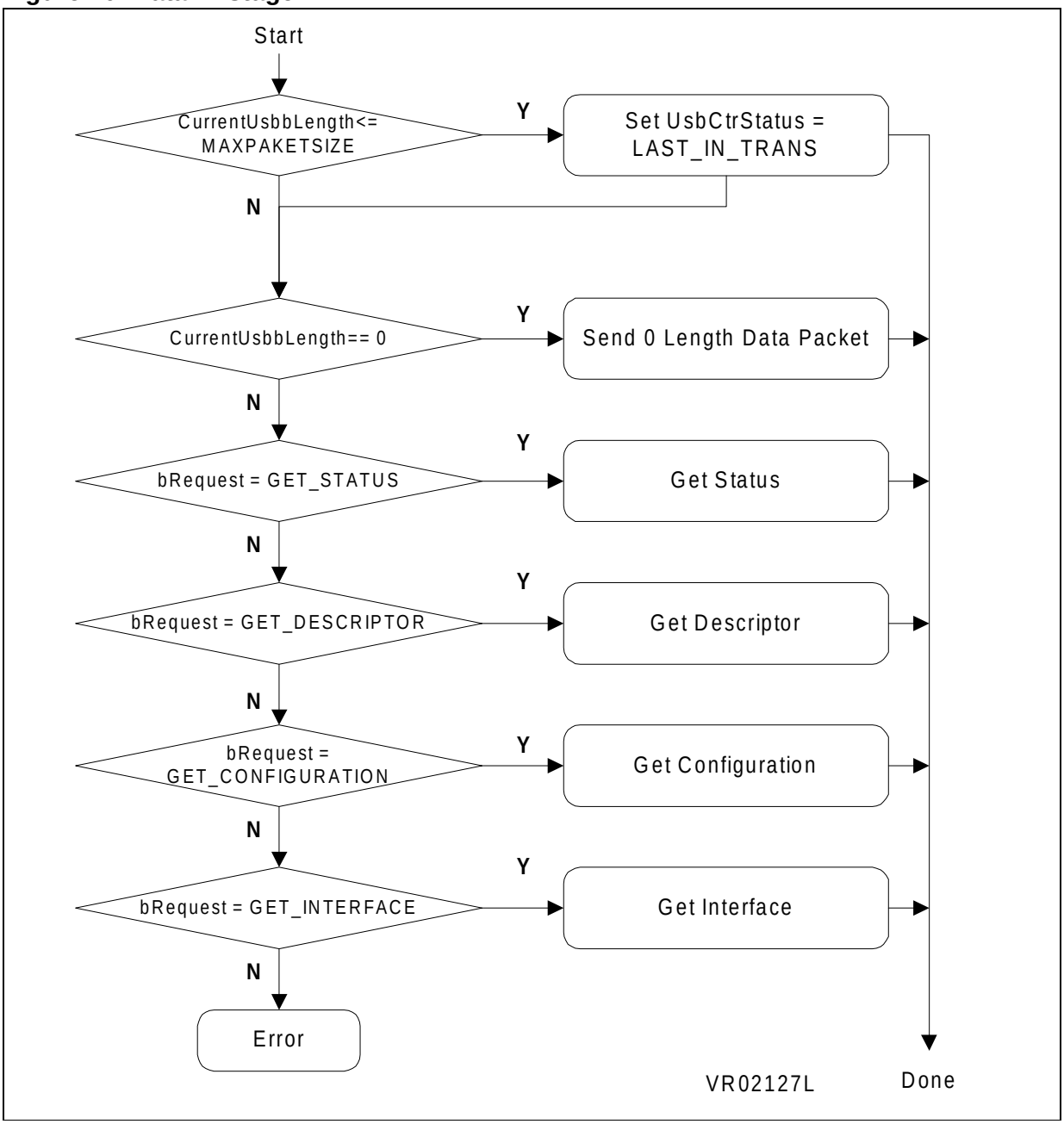
Figure 19. Standard request processing without data transfer



5.5.2 Data stages

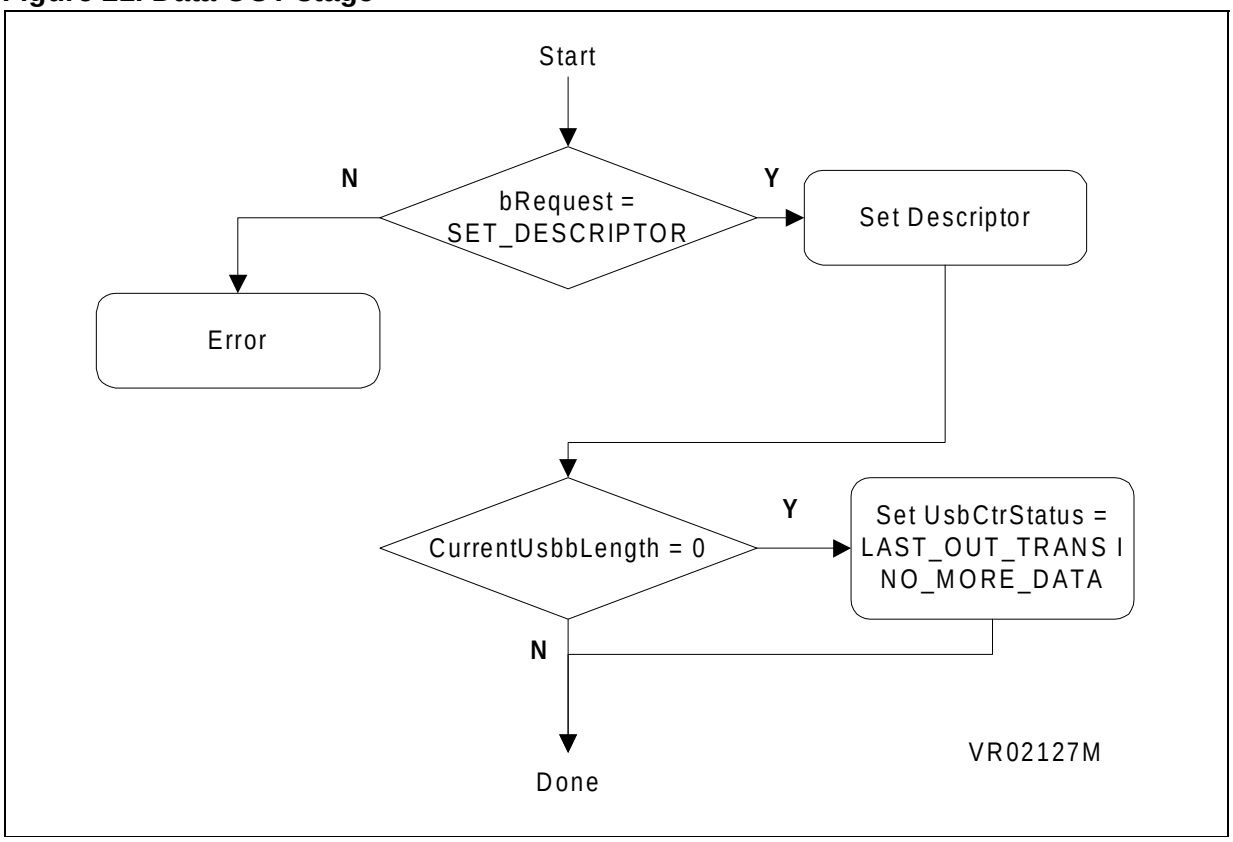
5.5.2.1 Data IN stage

Figure 20. Data IN stage



5.5.2.2 Data OUT stage

Figure 21. Data OUT stage

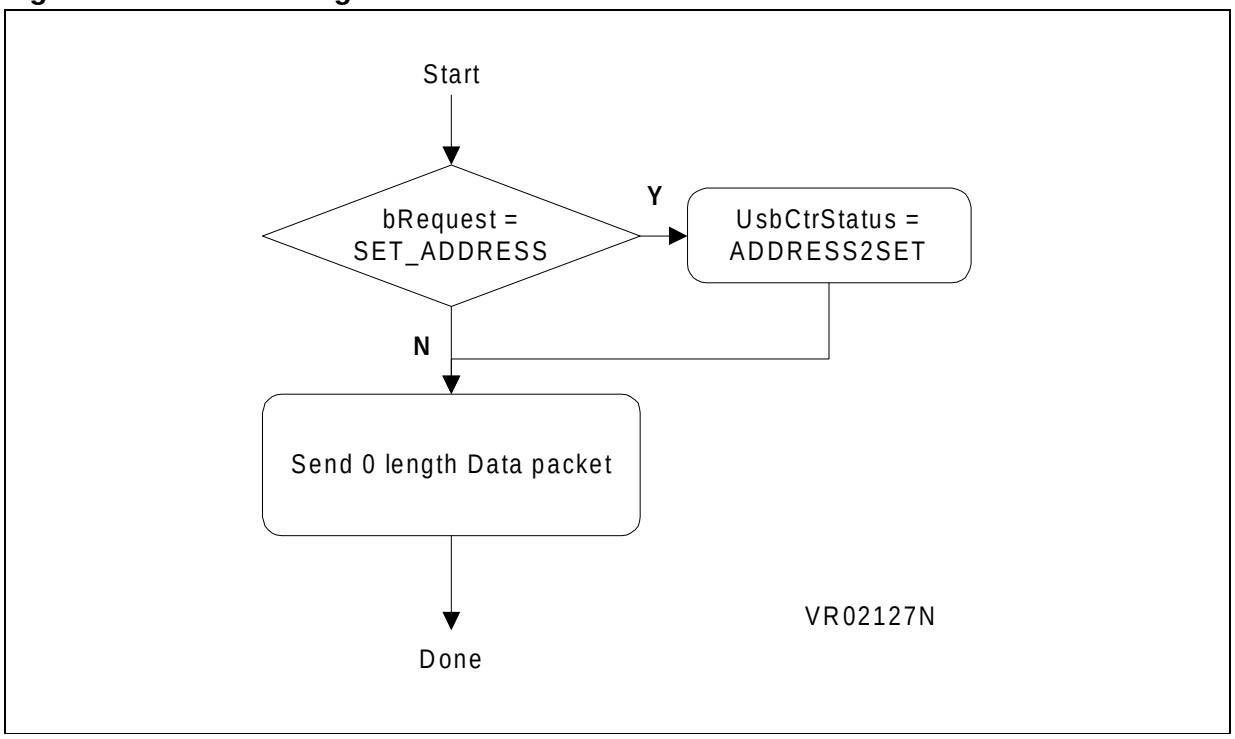


5.6 STATUS STAGES

5.6.1 Status IN stage

The function must return a zero length data packet in response to a IN token for the status IN stage. For a SET_ADDRESS request, the status IN stage routine is somewhat different. The firmware must wait for the completion of the SET_ADDRESS request processing to assign to the device the unique address sent by the host since the transfers for this request are addressed to the default address. A software flag is set if a SET_ADDRESS request is received. After completion of the control transfer, the firmware checks if this flag is set to write the unique address in the DADDR register (see figure *IN token received by endpoint 0*)

Figure 22. Status IN stage

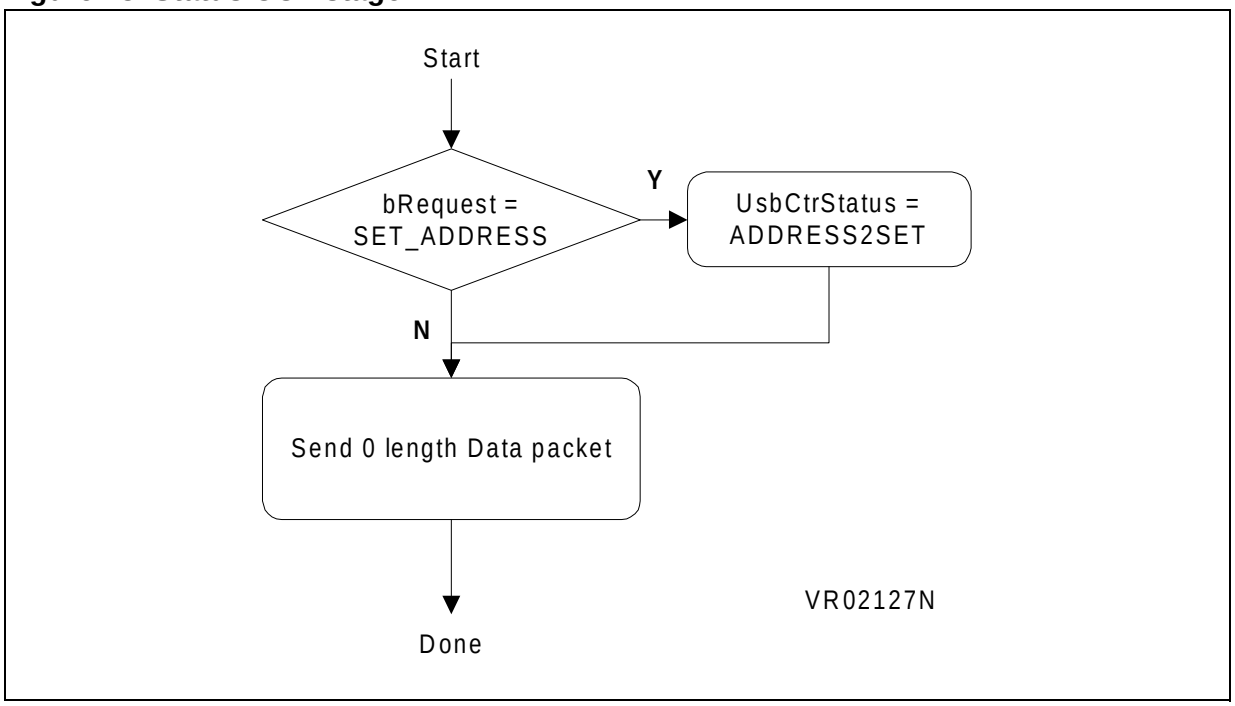


5.6.2 Status OUT stage

When the last data IN stage is performed, the firmware executes the status OUT stage routine. This routine sets the ST_OUT bit in the EP0RA register. When this bit is set, all non zero data transactions are STALLed by hardware.

When the OUT token of the status OUT stage is received, the status OUT stage routine is called and then clears the ST_OUT bit.

Figure 23. Status OUT stage



5.7 USB RESET

The USB reset routine is called when a USB reset signal has generated a USB reset interrupt. The firmware must then re-initialize the USB interface following this sequence:

1. Set the starting address of the endpoint DMA buffers

2. Enable endpoint 0 in reception

3. Write interrupt mask register

3. Power ON the internal 3.3V regulator to supply the external pull-up resistor used for detection by a hub.

5.8 START OF FRAME EVENT

This routine is called when a SOF indication on the bus has generated a USB SOF interrupt. See chapter 4.1.1 *Entering suspend state* for detailed information.

THE PRESENT NOTE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS WITH INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE TIME. AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM THE CONTENT OF SUCH A NOTE AND/OR THE USE MADE BY CUSTOMERS OF THE INFORMATION CONTAINED HEREIN IN CONNEXION WITH THEIR PRODUCTS.