

# USB9603/4 Full-Speed Firmware for USB Demo Board

National Semiconductor  
 Technical Bulletin  
 November 2002  
 Revision 1.0



## 1.0 Scope

This technical bulletin provides a short description of the National Semiconductor® USB9603/4 Firmware (FW), which runs on the USB demo board.

## 2.0 Firmware Description

The USB9603/4 FW runs on a CompactRISC 16B core. It runs from flash memory and uses on-chip RAM for its variables and stack implementation.

The USB9603/4 FW supports the following functions:

- Control packet transfer
- Isochronous packet transfer
- Bulk packet transfer
- Interrupt packet transfer
- Additional testing and debug features

## 3.0 USB9603/4 Software Architecture

### 3.1 DESCRIPTION

Figure 3-1 shows the main modules of the USB9603/4 FW and the various processes.

### 3.1.1 General

The USB9603/4 FW is a process that handles all USB tasks according to pre-defined priorities. All USB9603/4 FW communication with either the host or various hardware components is event driven.

Each interrupt is processed upon its reception. Some interrupt handlers set a flag to mark new tasks for processing by the main USB process. Others tasks are done directly from the interrupt handlers.

### 3.1.2 Interrupt Processor

The interrupt processor manager, *usb\_node\_handler*, is activated each time an event occurs. It parses the source of the interrupt source and initiates a process to handle the event according to the interrupt type.

The interrupts can be of the following types:

#### Receive

Receive interrupt indicates that the FIFO contains new data. Receive Handler validates the correctness of the data by making sure that there are no errors reported in the status registers.

In addition, for Bulk and Interrupt protocols, it makes sure that the toggle bit value is as expected.

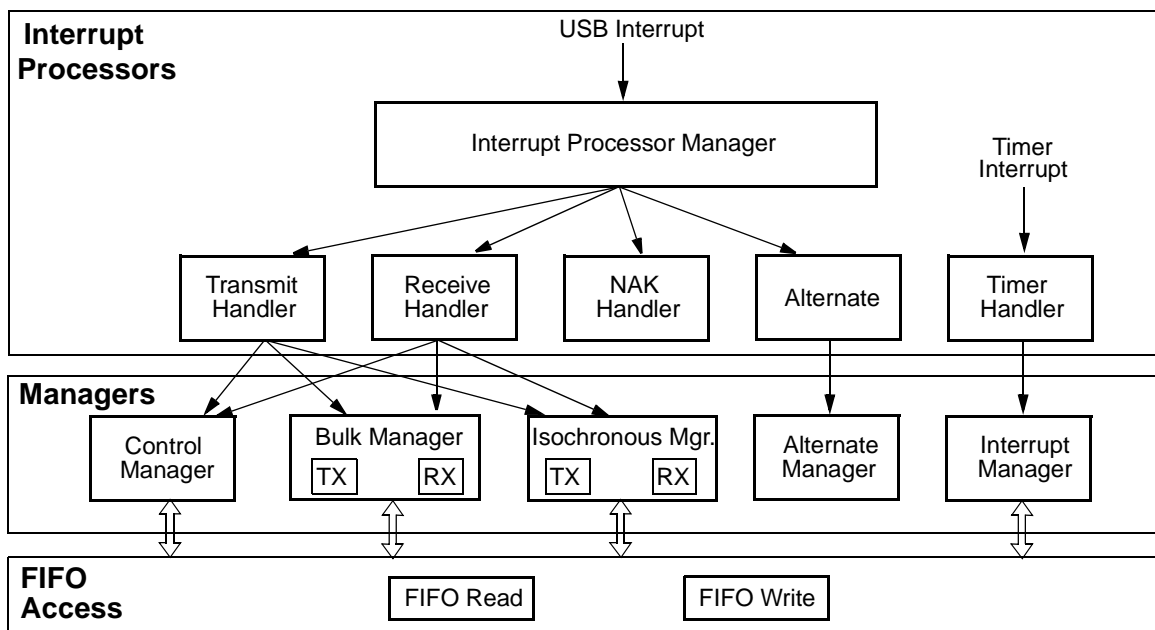


Figure 3-1. USB9603/4 Firmware Block Diagram

National Semiconductor is a registered trademark of National Semiconductor Corporation.  
 For a complete listing of National Semiconductor trademarks, please visit [www.national.com/trademarks](http://www.national.com/trademarks).

After data is validated, the Receive handler activates one of the USB receive processes, according to the FIFO, either by setting a flag for the main USB process or immediately in the event of Endpoint 0.

The *USB9604\_rx\_event\_handler* routine implements the Receive handler. It is called directly from the interrupt processor.

### Transmit

Transmit interrupt indicates that data was sent from any of the FIFOs. Transmit Interrupt validates that data was transmitted by the host by making sure that there are no errors reported in the status registers. Then it activates one of the USB Transmit processes according to the FIFO, either by setting a flag for the main USB process or immediately in the event of Endpoint 0.

The *USB9604\_tx\_event\_handler* routine implements the Transmit handler. It is called directly from the interrupt processor.

### NAK

A NAK interrupt is processed immediately from the interrupt processor handler. NAK is handled only for Endpoint 0. It re-enables the FIFO for an additional transaction.

### Alternate

An alternate interrupt is an interrupt from any of the hardware modules that is not USB data transaction related, for example, a DMA interrupt. The alternate interrupt handler identifies the source of the interrupt and calls the Alternate Manager.

The *USB9604\_alt\_event\_handler* routine processes the Alternate interrupt.

### 3.1.3 Main USB Process Activities

The main USB process runs in an endless loop and handles tasks generated by the interrupt handlers. The main USB process uses the following managers to handle all protocol tasks.

#### Control Manager

The Control Manager is activated by a Transmit/Receive interrupt, to/from Endpoint 0 FIFO.

For each Receive interrupt, the Control Manager reads a "Request" from the FIFO.

There are two types of requests:

- Standard USB requests: Used for getting status information from the USB node or for setting the node.
- Custom requests: For testing purposes; they either configure the USB node for the next test or read the results of the previous test.

Each request has a dedicated function to handle it. The names of the functions are defined in the `usb\usb.c` file.

A complete description of the supported requests can be found in Section 4.3.

The *USB\_device\_req\_handler* routine called by the *USB9604\_rx\_event\_handler* parses the request and calls

the request specific functions. In case of a Custom request it calls the appropriate function according to the test type and initializes the test.

Note: For Transmit interrupts, the Control Manager sends additional data to the Endpoint 0 FIFO (if it exists), according to the request currently being handled.

#### Bulk Manager

The Bulk Manager is in charge of bulk packet transfer and reception of packets and of analyzing the data received for testing purposes.

For each Receive interrupt, the Bulk Manager reads the data from the Bulk FIFO and processes it according to the type of the test. For one-direction tests, it compares the data with a pre-defined hexadecimal byte; for loopback tests, it stores the data in memory.

For each Transmit interrupt, the Bulk Manager writes the data to the Bulk FIFO according to the type of the test. For one-direction tests, it writes a pre-defined hexadecimal byte; for loopback tests, it loads stored data from memory.

The *BulkReceiveEvent* routine performs the Bulk receive data operations. The *BulkTransmitEvent* routine performs the Bulk transmit data operations.

#### Isochronous Manager

The Isochronous Manager is in charge of isochronous packet transfer and reception of packets and of analyzing the data received for testing purposes.

For each Receive interrupt, the Isochronous Manager reads the data from the Isochronous FIFO and processes it according to the type of the test. For one-direction tests, it compares the data with a pre-defined hexadecimal byte; for loopback tests, it stores the data in memory.

For each Transmit interrupt, the Isochronous Manager writes the data to the Isochronous FIFO according to the type of the test. For one-direction tests, it writes a pre-defined hexadecimal byte; for loopback tests, it loads stored data from memory.

The *IsoReceiveEvent* routine performs the Isochronous receive data operations; the *IsoTransmitEvent* routine performs the Isochronous transmit data operations.

#### Interrupt Manager

The Interrupt Manager is in charge of Interrupt packet transfer. The Interrupt Manager is activated by a periodic timer interrupt, and not by FIFO events as the other USB transfer managers. USB interrupt packet transmission is supposed to be generated by the USB device hardware. Since the USB9603/4 FW does not support a specific device, the trigger for an interrupt transaction is the timer and not a Device Hardware event.

The number of interrupts to generate and the time difference between each interrupt is configured prior to the test.

### 3.1.4 FIFO Access

#### Read FIFO

*USB\_fast\_read* is called whenever the FW knows exactly how many bytes of data to read from the FIFO. This function is used in some of the tests; however it is not recommended for general USB transactions. This routine accesses a specific FIFO according to the endpoint number and reads the exact number from the FIFO without checking how many new bytes of data are there.

*USB\_Receive\_Data* reads the new data from a FIFO. It receives the Endpoint number and reads all the new data from the FIFO.

**Write FIFO**

One function accesses the FIFOs for transmit. *USB\_Transmit\_Data* receives a data buffer to send and the size of the data to send. It copies the data to the FIFO and enables the transaction by calling *usbn9604\_tx\_enable*.

Note that *usbn9604\_tx\_enable* has a different implementation in Isochronous. It sets the IGN\_ISOMSK bit, and data is transmitted upon reception of the next IN token.

**4.0 USB9603/4 Firmware Interfaces**

The host and the USB9603/4 FW communicate via data that is sent or read from any of the USB FIFOs. Standard or Custom Requests, in Control Transfer, is the way to configure the FW or to read information for the USB node (for example, test results, transaction details or USB parameters).

**4.1 DESCRIPTORS**

USB9603/4 FW has one configuration of one interface divided into two alternate settings.

- Alternate setting 0 is the default USB setting; it contains all endpoints except Isochronous.
- Alternate setting 1 is the default USB9603/4 demo application setting; it contains all endpoints.

This partition was defined to support a USB standard request which defines the default setting without isochronous support. The demo application configures the device to use alternate setting 1.

**4.2 ENDPOINTS**

**Endpoint 1 - Bulk IN**

This endpoint is defined for both settings. It is used in one-direction Bulk Device-to-Host tests and in Bulk loopback tests.

**Endpoint 2 - Bulk OUT**

This endpoint is defined for both settings. It is used in one-direction Bulk Host-to-Device tests and in Bulk loopback tests.

**Endpoint 3 - Isochronous IN**

This endpoint is defined for Alternate Setting 1 only. It is used in one-direction Device-to-Host Isochronous tests and in Isochronous loopback tests.

**Endpoint 4 - Isochronous OUT**

This endpoint is defined for Alternate Setting 1 only. It is used in one-direction Isochronous Host-to-Device tests and in Isochronous loopback tests.

**Endpoint 5 - Interrupt**

This endpoint is defined for both settings. It is used interrupt test.

**4.3 REQUESTS**

**4.3.1 Standard Requests**

USB9603/4 FW supports all standard requests defined in the USB 1.1 specification. Refer to the specification for a detailed description of USB standard requests.

**4.3.2 Custom Requests**

Custom requests are specific requests for testing purposes. It supports test parameters configuration and test results fetching.

**4.3.2.1 BULK\_SEND\_IMM\_BACK**

This request configures the USB9603/4 FW to initiate a Bulk loopback test. Data is sent back to the host immediately.

Request Type: 0x43  
 bRequest: 0x10  
 wValue: Data Length LSB  
 wIndex: Data Length MSB  
 wLength: 0

**4.3.2.2 BULK\_SEND\_AFTER\_LAST\_BACK**

This request configures the USB9603/4 FW to initiate a Bulk loopback test. Data is sent back to the host after the last packet is received.

Request Type: 0x43  
 bRequest: 0x11  
 wValue: Data Length LSB  
 wIndex: Data Length MSB  
 wLength: 0

**4.3.2.3 BULK\_SEND\_DATA**

This request configures the USB9603/4 FW to initiate a one-direction Bulk test from the USB device to the host.

Request Type: 0x43  
 bRequest: 0x12  
 wValue: Data Length LSB  
 wIndex: Data Length MSB  
 wLength: Byte defined in wValue and wIndex is sent to the host the number of times defined in wLength

**4.3.2.4 BULK\_GET\_DATA**

This request configures the USBN9603/4 FW to initiate a one-direction Bulk test from the host to the USB device.

Request Type: 0x43  
 bRequest: 0x13  
 wValue: Data Length LSB  
 wIndex: Data Length MSB  
 wLength: Byte defined in wValue and wIndex is received from the host the number of times defined in wLength

**4.3.2.5 INTERRUPT\_GET\_INT**

This request configures the USBN9603/4 FW to initiate an Interrupt transfer test from the USB device to the host.

Request Type: 0x43  
 bRequest: 0x20  
 wValue: Number of interrupts to be generated  
 wIndex: Interval between interrupts  
 wLength: 0

**4.3.2.6 ISOCH\_SEND\_IMM\_BACK**

This request configures the USBN9603/4 FW to initiate a loopback Isochronous test. Data is sent back to the host immediately.

Request Type: 0x43  
 bRequest: 0x30  
 wValue: Data Length LSB  
 wIndex: Data Length MSB  
 wLength: 0

**4.3.2.7 ISOCH\_SEND\_DATA**

This request configures the USBN9603/4 FW to initiate a one-direction Bulk test from the USB device to the host.

Request Type: 0x43  
 bRequest: 0x31  
 wValue: Data Length LSB  
 wIndex: Data Length MSB  
 wLength: Byte defined in wValue and wIndex is sent to the host the number of times defined in wLength

**4.3.2.8 ISOCH\_GET\_DATA**

This request configures the USBN9603/4 FW to initiate a one-direction Isochronous test from the host to the USB device.

Request Type: 0x43  
 bRequest: 0x13  
 wValue: Data Length LSB  
 wIndex: Data Length MSB

wLength: Byte defined in wValue and wIndex is received from the host the number of times defined in wLength

**4.3.2.9 GET\_BULK\_FEEDBACK**

This request returns the results of the last BULK\_GET\_DATA request, if available. The USBN9603/4 FW returns the number of bytes received and the number of errors undetected by hardware.

Request Type: 0xC3  
 bRequest: 0x01  
 wValue: 0  
 wIndex: 0  
 wLength: 8 (Feedback packet length)

**4.3.2.10 GET\_ISO\_FEEDBACK**

This request returns the results of the last ISOCH\_GET\_DATA request, if available. The USBN9603/4 FW returns the number of bytes received and the number of errors undetected by hardware.

Request Type: 0xC3  
 bRequest: 0x02  
 wValue: 0  
 wIndex: 0  
 wLength: 8 (Feedback packet length)

**4.3.2.11 STOP\_REQUEST**

This request stops the current request.

Request Type: 0x43  
 bRequest: 0x03  
 wValue: 0  
 wIndex: 0  
 wLength: 0

**4.3.2.12 READ\_USB\_REG**

This request reads a selected USBN9603/4 register. The USBN9603/4 FW returns the register value.

Request Type: 0xc3  
 bRequest: 0x04  
 wValue: register address  
 wIndex: 0  
 wLength: 0

**4.3.2.13 WRITE\_USB\_REG**

This request writes to a selected USBN9603/4 register.

Request Type: 0xc3  
 bRequest: 0x05  
 wValue: register address  
 wIndex: new value of register  
 wLength: 0

## 5.0 DMA Operation

The firmware includes sample code for DMA operation. This code is located in: `usb\usbdrv.c`.

There are two main routines for DMA operation:

- `USB_init_dma` – This routine is called once when enabling a DMA usage for one of the Endpoints. The argument for this function is the Endpoint to use DMA.
- `USB_start_dma` – This routine is called at the start of transmit/receive using DMA, and whenever the DMA buffer is full or emptied, depending on the direction of the Endpoint. It enables the DMA and generates the next transaction. The argument is the number of packets to read or write.

Before enabling the DMA, it is most important to disable any interrupt from the Endpoint to avoid confusion between the DMA transactions and the Endpoint transactions.

The following macros are recommended:

- `DISABLE_TX_INTS(TX_FIFO[number of FIFO])` for transfer operations
- `DISABLE_RX_INTS(RX_FIFO[number of FIFO])` for receive operations

The interrupt handler is already compliant with DMA operation. It controls the interrupt processing and re-enables the DMA, if necessary. However, it does not handle data or configure the DMA controllers. To add these missing operations, you must update the following routine: `USBN9604_alt_event_handler(void)`.

## 6.0 Performance Enhancement Using Ping-Pong Buffers

Ping-Pong is a FIFO data buffering method.

Normally, FIFO data transmission or reception is set up using one endpoint for each pipe. However, allocating two same-direction endpoints to one pipe is an effective way to increase the data transfer rate because the second FIFO is filled while the contents of the first is being transmitted. This data buffering method is called ping-pong buffering.

In case a one-direction OUT Bulk test is initiated, the FW uses this method for performance enhancement.

The Request `BULK_GET_DATA` uses a Ping-Pong buffer.

The function `pingPong` is called after a receive interrupt handler sets a flag to the main USB process in an OUT bulk test. It replaces the standard USB Bulk Manager activity.

This function works only in Polling mode (i.e., no interrupt is used). However, generally, it is recommended to use interrupt driven mode.

## 7.0 Content Description

### 7.1 DIRECTORY TREE

```

FWSource-----> drv
                  |
                  -----> flash
                  |
                  -----> include
                  |
                  -----> io
                  |
                  -----> link
                  |
                  -----> Loopback
                  |
                  -----> start
                  |
                  -----> USB
                  |
                  -----> UART
    
```

### 7.2 DIRECTORY DESCRIPTION

#### drv

This directory includes files that initialize and control the CPU registers and peripherals.

#### flash

This directory includes the batch and output files required to download the FW. The `load_readme.txt` file describes the loading process.

#### include

This directory includes some general definitions in the header files.

#### io

This directory includes the files that access and control the DIP switches and the 7-segment display on the USB Demo board.

#### link

The linker definition files required for compilation of the FW, using the CompactRISC toolset.

#### Loopback

This directory contains files that handle testing features.

File Name	Description
<code>Command_api.h</code>	Custom request for all test features supported by the FW.
<code>parser.c</code>	Includes a main loop, waits for events of USB and handles them. Parses the custom requests.
<code>parser.h</code>	Definitions and enumerations of custom requests.

**start**

This directory includes assembler files that initialize CR16B registers.

**UART**

This directory includes all files for RS-232 interface for diagnostic purposes.

**USB**

This directory includes the various files required to initialize the USB and control USB transactions. These files are described in the following table:

File Name	Description
usb.h	Descriptor structure. Request structure. Enumeration of device class descriptor types and requests. General constants.
usb.c	USB initialization and configuration. Data definitions for all the descriptors. Standard request handlers (Control Transfer). Access to Control FIFO.
usbdrv.h	Macros for access to USB registers.
usbdrv.c	Handling of USB interrupts. Access to FIFOs. DMA support.
usb_regs.h	Definitions of all USB register addresses and fields.

## LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.



**National Semiconductor Corporation Americas**  
Email: [new.feedback@nsc.com](mailto:new.feedback@nsc.com)

**National Semiconductor Europe**  
Fax: +49 (0) 1 80 530 85 86  
Email: [europe.support@nsc.com](mailto:europe.support@nsc.com)  
Deutsch Tel: +49 (0) 69 9508 6208  
English Tel: +44 (0) 870 24 0 2171  
Français Tel: +33 (0) 1 41 91 87 90

**National Semiconductor Asia Pacific Customer Response Group**  
Fax: 65-250-4466  
Tel: 65-254-4466  
Email: [ap.support@nsc.com](mailto:ap.support@nsc.com)

**National Semiconductor Japan Ltd.**  
Fax: 81-3-5639-7507  
Tel: 81-3-5639-7560  
Email: [nsj.crc@jksmtp.nsc.com](mailto:nsj.crc@jksmtp.nsc.com)