National Semiconductor

The LM9831 and LM9832 USB Interface

Version 1.0
6/26/00

National Semiconductor

1 General Description

This document describes the host's view of the USB LM9831, the logic between the LM9831 and the USB interface and any other points specific to the use of the USB interface in LM9831 not described in the Configurable USB Device Adapter documentation.

The device implements one vendor specific USB interface, which consists of the control endpoint plus 3 other endpoints.

The last section is addendum documenting the changes between the LM9831 and the LM9832.

2 Endpoints

The USB interface supports 4 endpoints.

2.1 Control Endpoint

Endpoint number is 0, maximum packet size is 8 bytes, single buffering is used.

2.1.1 Requests

The default control endpoint supports the required standard requests.

Two additional vendor specific interface requests are available to read and write registers. Each request reads or writes a series of registers, beginning at the first register specified.

The WRITE_CONTROL request is patterned after the proposed imaging class specific interface request of the same name. For this request, bmRequestType = 0x41, bRequest = 0x00, wValue = address of first register to be written (0x0000 to 0x00ff), wIndex = 0x0000, and wLength = number of registers to be written (0x0000 to 0x00c0), and data is the data to be written.

The READ_CONTROL request is patterned after the proposed imaging class specific interface request of the same name. For this request, bmRequestType = 0xc1, bRequest = 0x00, wValue = address of first register to be read (0x0000 to 0x00ff), wIndex = 0x0000, and wLength = number of registers to be read (0x0000 to 0x00c0), and data is the data read.

2.1.2 Descriptors

The device descriptor programmed into the internal ROM specifies the following:
bLength = 0x12
bDescriptorType = 0x01 = DEVICE
bcdUSB[7:0] = 0x00 = 1.00
bcdUSB[15:8] = 0x01 = 1.00
bDeviceClass = 0x00 = Independent class interfaces
bDeviceSubClass = 0x00
bDeviceProtocol = 0x00
bMaxPacketSize0 = 0x08
idVendor[7:0] = 0x00 = National Semiconductor
idVendor[15:8] = 0x04 = National Semiconductor
idProduct[7:0] = 0x00
idProduct[15:8] = 0x10
bcdDevice[7:0] = 0x00 = 1.00
bcdDevice[15:8] = 0x01 = 1.00
iManufacturer = 0x01
iProduct = 0x02
iSerialNumber = 0x00 = not specified
bNumConfigurations = 0x01

The configuration descriptor programmed into the internal ROM specifies the following:
bLength = 0x09
bDescriptorType = 0x02 = CONFIGURATION
wTotalLength[7:0] = 0x27
wTotalLength[15:8] = 0x00

bNumInterfaces = 0x01
bConfigurationValue = 0x01
iConfiguration = 0x00 = not specified
bmAttributes = 0xa0 = {BusPowered,RemoteWakeup} if self_powered pin is low
= 0x60 = {SelfPowered,RemoteWakeup} if self_powered pin is high
MaxPower = 0xfa = 500ma if self_powered pin is low
= 0x01 = 2ma if self_powered pin is high

The interface descriptor programmed into the internal ROM specifies the following:
bLength = 0x09
bDescriptorType = 0x04 = INTERFACE
bInterfaceNumber = 0x00
bAlternateSetting = 0x00
bNumEndpoints = 0x03
bInterfaceClass = 0xff = Vendor
bInterfaceSubClass = 0x00
bInterfaceProtocol = 0xff = Vendor
iInterface = 0x00 = not specified

The endpoint 1 descriptor programmed into the internal ROM specifies the following:
bLength = 0x07
bDescriptorType = 0x05 = ENDPOINT
bEndpointAddress = 0x81 = {IN,1}
bmAttributes = 0x03 = Interrupt
wMaxPacketSize[7:0] = 0x01
wMaxPacketSize[15:8] = 0x00
bInterval = 0x10 = 16ms

The endpoint 2 descriptor programmed into the internal ROM specifies the following:
bLength = 0x07
bDescriptorType = 0x05 = ENDPOINT
bEndpointAddress = 0x82 = {IN,2}
bmAttributes = 0x02 = Bulk TBD: is this bit encoding right?
wMaxPacketSize[7:0] = 0x40
wMaxPacketSize[15:8] = 0x00
bInterval = 0x00

The endpoint 3 descriptor programmed into the internal ROM specifies the following:
bLength = 0x07
bDescriptorType = 0x05 = ENDPOINT
bEndpointAddress = 0x03 = {OUT,3}
bmAttributes = 0x02 = Bulk TBD: is this bit encoding right?
wMaxPacketSize[7:0] = 0x40
wMaxPacketSize[15:8] = 0x00
bInterval = 0x00

String descriptor 0 programmed into the internal ROM specifies the language IDs supported:
bLength = 0x04
bDescriptorType = 0x03 = STRING
LangID[7:0] = 0x09 = Primary Language = English
LangID[15:8] = 0x04 = Sub Language = US

String descriptor 1 programmed into the internal ROM specifies the manufacturer (in UNICODE):
bLength = 0x2e
bDescriptorType = 0x03 = STRING
string = “National Semiconductor”

String descriptor 2 programmed into the internal ROM specifies the product (in UNICODE):
bLength = 0x1e
bDescriptorType = 0x03 = STRING
string = “LM9831 Scanner”

2.2 Interrupt Endpoint

Endpoint number is 1, maximum packet size is 1 byte, buffer size is 8 bytes, single buffering is used.

Single data byte packets are returned on this endpoint. When set, each bit of the data byte indicates that the corresponding bit of the miscellaneous I/O status register (0x02) has changed since the register was last read. Such packets are only returned when the status bits change. When the host receives such a packet, it should then read the status register to determine the current values.

2.3 Bulk In Endpoint

Endpoint number is 2, maximum packet size is 64 bytes, double buffering is used, the buffer is shared with the bulk out endpoint.

Registers may be read via the bulk in endpoint by sending a sequence of command bytes on the bulk out endpoint, then reading the data bytes on the bulk in endpoint. There are 4 command bytes, which, in order, indicate the mode of the transfer, the starting address, and the number of bytes to be read (most significant byte, followed by least significant byte). The mode byte is bit mapped: bit 0 is set to indicate a register read operation; bit 1, if set, indicates that each byte will be read from the register at the next higher address that the previous byte (incrementing address mode), if clear, indicates that each byte will be read from the same register; all other bits are cleared.

Because register reads via this endpoint share the bulk out endpoint with register writes, such a write must complete before such a read is started, or vice versa. Since reads and writes via the control endpoint use different resources, reads or writes on the control endpoint may be performed simultaneously with reads or writes on the bulk endpoints.

2.4 Bulk Out Endpoint

Endpoint number is 3, maximum packet size is 64 bytes, double buffering is used, the buffer is shared with the bulk in endpoint.

Registers may be written via the bulk out endpoint by sending a sequence of command bytes, followed by the data bytes. There are 4 command bytes, which, in order, indicate the mode of the transfer, the starting address, and the number of bytes to be written (most significant byte, followed by least significant byte). The mode byte is bit mapped: bit 0 is cleared to indicate a register write; bit 1, if set, indicates that each byte will be written to the register at the next higher address that the previous byte (incrementing address mode), if clear, indicates that each byte will be written to the same register; all other bits are cleared.

Because register writes via this endpoint share the endpoint with register writes, such a read must complete before such a write is started, or vice versa. Since reads and writes via the control endpoint use different resources, reads or writes on the control endpoint may be performed simultaneously with reads or writes on the bulk endpoints.

3 Registers

3.1 Address space

The address space available for register access is 0x00 to 0xbf. 0xc0 to 0xff is not available for registers because the data transfer interface uses this range to address endpoint pipes.

3.2 Operational registers

LM9831's existing registers are mapped directly into the 0x00 to 0x7f address range.

Non-blocking flow control is implemented on accesses to register 0x00 (Pixel Data). If a read cannot be performed on this register because data is not available in the buffer, a retry status will be returned to the USB interface, thus enabling other endpoints to read/write other registers.

Accesses to all other registers use blocking flow control. An acknowledgement is not returned to the USB interface until the data has been accepted or provided by LM9831.

3.3 Diagnostic registers

The following registers are available for diagnostic and production test purposes.

0xb6, frame0_diag, r/w, resets to 0x00, controls short frame test mode used for speeding up testing of suspend and resume timing, 0x00 enables normal frame timer length of 36015 cycles of the 12MHz clock, 0x01 enables short frame timer length of 2223 cycles of the 12MHz clock.

0xb7, mac0_diag, r/o, provides visibility to various mac states

0xb8, mac1_diag, r/o, provides visibility to various mac states

0xb9, mac2_diag, r/o, provides visibility to various mac states

0xba, mac3_diag, r/o, provides visibility to various mac states

0xbb, phy0_diag, r/o, provides visibility to various phy states

0xbc, xcvr0_diag, r/o, receiver test register, bit 0 samples the state of the differential receiver, bit 1 samples the state of the D- single ended receiver, bit 2 samples the state of the D+ single ended receiver. When the self_powered strap input is high, these bits reflect the current state of the receivers. When the self powered strap input is low, these bits are latched. To test the receivers, with the self_powered strap input high, drive the D+ and D- inputs with the desired levels; take the self_powered pin low to latch the sample; do a USB transfer to read this register. There are two sets of receivers, differential receivers which are active in the operational states, and CMOS receivers which are active in suspend state. Be careful not to allow the USB interface to enter the suspend state (bus in J state for 3ms or longer) when testing the differential receivers. Note that the clock need not be running to sample the receiver states.

0xbd, xcvr1_diag, r/w, resets to 0x00, transmitter D+ test sequence, see description below

0xbe, xcvr2_diag, r/w, resets to 0x00, transmitter D- test sequence, see description below

0xbf, xcvr3_diag, r/w, resets to 0x00, transmitter enable test sequence, when any bit of xcvr3_diag is set, and the self_powered strap input is high, the transmitter test mode is entered. In this mode, an 8 bit pattern is sent by the D+ and D- transmitters. For the first bit, xcvr1_diag[0] sets the D+ logic level, xcvr2_diag[0] sets the D- logic level, and xcvr3_diag[0], if set, enables both transmitters. For the second through eigth bits of the test sequence, bits 1 to 7 of these registers are likewise used. The sequence will repeat as long as self_powered is high.

4 Power management

The USB bus state and device state control the bus power consumed by the device. Both states are available to the LM9831 logic. If the USB is suspended, the device will draw no more than 500uA from the USB (the pullup resistor consumes 200uA of this, nominally). If the USB is not suspended, but the device is not configured, the device will draw no more than 100mA from the USB. If the USB is not suspended and is configured, then the device may draw up to 500mA from the USB.

The miscellaneous I/O status register (0x02) forms the basis for initiating remote resume requests. If any bit of this register has changed since the last time the register was read, and the USB is suspended, a remote resume will be done. This logic is asynchronous to permit a remote resume to be initiated while the clock is stopped. Some of the logic may be shared with the interrupt endpoint logic. If the device remote wakeup enable feature is not enabled, the USB interface will not propagate the resume request onto the USB.

5 LM9832 Addendum

5.1 USB Interface ROM Changes from LM9831 to LM9832

There are two sets of changes. The first set adds support for vendor defined requests with the "device" as the recipient. The first ROM was coded to support vendor defined requests with only the "interface" as the recipient. Since then, Microsoft has released its Still Image interface (STI) for device drivers, which supports vendor specific requests with the "device" as the recipient, but not with the "interface" as the recipient. Furthermore, the STI does not even support the standard USB requests that enable and disable the device's remote wakeup feature. So, in order to support the READ_CONTROL, WRITE_CONTROL, SET_FEATURE(DEVICE_REMOTE_WAKEUP), and SET_FEATURE(DEVICE_REMOTE_WAKEUP) requests, vendor specific "device" versions of these requests were added to the ROM.

Incoming requests are interpreted by a progressive decision tree coded into the ROM. As each byte is read from the receive buffer, it is tested against several values. When a value matches, program execution branches to the code which further services that byte value. The new requests are supported by adding tests for the new byte values and adding corresponding code. Because only the first 2 bytes differ between the old and new READ_CONTROL and WRITE_CONTROL requests, this code merges back together, and the same code services byte 3 and onward of both the old and new requests.

The second set of changes consists of updates to the descriptors to indicate the new product, new revision of the USB specification, and use of vendor defined requests with the "device" as the recipient.

These changes were first coded into an external EEPROM and tested with several PC systems. The changes were then made to the uprog_sti.u file and compiled with the uasm script to produce a file for Verilog simulation and another file for ROM generation.

5.2 New Vendor-Specific Device Requests

bmReqTyp	bRequest	wValue	wIndex	wLength	Data	Description
40	04	00xx	0001	0001	yy	CLEAR_FEATURE(DEVICE_REMOTE_WAKEUP)
40	0C	00xx	0001	0001	yy	CLEAR_FEATURE(DEVICE_REMOTE_WAKEUP)
40	04	00xx	0003	0001	yy	SET_FEATURE(DEVICE_REMOTE_WAKEUP)
40	0C	00xx	0003	0001	yy	SET_FEATURE(DEVICE_REMOTE_WAKEUP)
40	04	offset	0000	length	data	WRITE_CONTROL(register_offset,data)
40	0C	offset	0000	length	data	WRITE_CONTROL(register_offset,data)
C0	04	offset	0000	length	data	READ_CONTROL(register_offset)
C0	0C	offset	0000	length	data	READ_CONTROL(register_offset)

xx = ignored (recommend setting to 01 to indicate DEVICE_REMOTE_WAKEUP feature selector)

yy = ignored, 1 byte

5.3 Changed Device Descriptor

----

bcdUSB: from: 0x0100 = "1.0" to: 0x0101 = "1.1"

bDeviceClass: from: 0x00 = Independent Class Interfaces to: 0xff = Vendor Specific

bDeviceProtocol: from: 0x00 to: 0xff = Vendor Specific

idProduct: from: 0x1000 to: 0x1001

5.4 Changed Product String Descriptor

----

from: "Merlin Scanner" to: "LM9832 42 Bit Scanner"

5.5 Additional Notes

With USB 1.1, bit 7 of the bmAttributes byte of the Configuration Descriptor is now a reserved bit, and should always be set to 1. However, this change did not make it into this ROM, which uses the USB 1.0 definition of this bit, which indicates BusPowered capability. Future revisions of the ROM should implement this change. Specifically, this involves making the following change in uprog.u:

```
from:    GetDescConfig1:  xmiti(0x60);           // bmAttributes = {SelfPowered,RemoteWakeup}
to:      GetDescConfig1:  xmiti(0xe0);           // bmAttributes = {SelfPowered,RemoteWakeup}
```