National Semiconductor

The Sight & Sound of Information

USB Interface Module for Applications

PCB Revision 1.1
Firmware Version 0207
Software Version 0.5

Reference Manual

CONTENTS

1 INTRODUCTION 4
1.1 OVERVIEW 4
1.2 SYSTEM LEVEL FEATURES 4
1.3 BLOCK DIAGRAM 4
2 INSTALLATION AND CONFIGURATION 5
2.1 INTRODUCTION 5
2.2 SYSTEM REQUIREMENTS 5
2.3 BOARD INSTALLATION AND POWER-UP 5
2.4 JUMPER DESCRIPTIONS 6
3 PROGRAMMING 6
3.1 INTRODUCTION 6
3.2 PROGRAMMING SETUP 6
3.3 PROGRAMMING CABLE 7
4 THEORY OF OPERATION 7
4.1 INTRODUCTION 7
4.2 USB INTERFACE MODULE 7
4.3 POWER AND CURRENT GAUGE 8
4.4 A/D CONVERTER 9
5 SOFTWARE 9
5.1 INTRODUCTION 9
5.2 COP8 FIRMWARE 9
5.2.1 FLASH MEMORY MAP 10
5.2.2 CONTROL CODES 11
5.2.3 SERIAL INTERFACES 13
I²C Compatible Interface 13
5.3 PC APPLICATION SOFTWARE 15
5.3.1 USING A DYNAMIC LINK LIBRARY 16
6 BOARD SPECIFICATIONS 22
6.1 INTRODUCTION 22
6.2 CONNECTORS 22
6.2.1 GPIO Connectors X1, X2 22
6.2.2 GPIO Connectors X3, X4 22
6.2.3 GPIO Connector TPB1 23
6.2.4 Programming Connector X5 23
Appendix A ELECTRICAL SCHEMATICS 25

1 INTRODUCTION

1.1 OVERVIEW

The USB Interface Module enables the user to control the application hardware through a PC's USB port and is a suitable replacement to a parallel (LPT) interface. An onboard microcontroller with flash memory enables the user to develop application specific functions or stand-alone systems. In addition, the USB port provides internal power that can be used by the application hardware.

1.2 SYSTEM LEVEL FEATURES

The USB Interface Module features:

- National's USBN9604 chip in a 28-pin SOIC package
- National's COP8-based microcontroller, COP8CBE9
- USB 1.1 / 2.0 compatible, slow mode
- Bus powered
- Hot plug in/out support
- 24 MHz clock
- Two 16-pin I/O connectors (double placed) to connect application board
- Optional 8-pin I/O connector
- Optional 7-pin connector for upgrade
- 16-channels 10-bit A/D converter
- Two 16-bit timers
- Two separated voltage sources for core and application board
- Three user selectable output voltages for application board
- National's current gauge LM3822 for current measurement

1.3 BLOCK DIAGRAM

Figure 1.1 shows a block diagram of the USB Interface Module and the connections between the PC and the application.

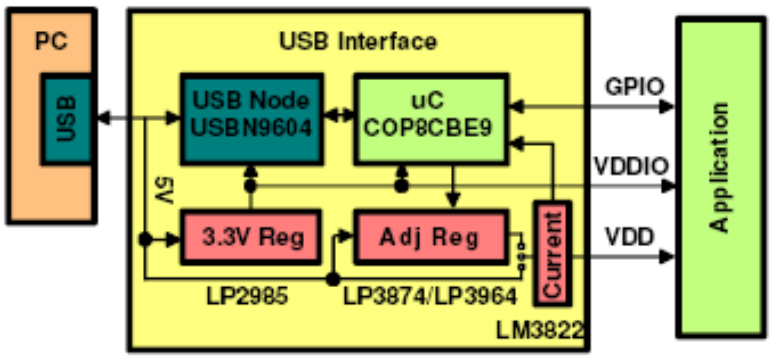


Figure 1.1 Block Diagram of the USB Interface Module and Environment

2 INSTALLATION AND CONFIGURATION

2.1 INTRODUCTION

This chapter describes the configuration of the USB Interface Module.

2.2 SYSTEM REQUIREMENTS

To use the USB Interface Module, the following is required:

- Host (PC) with USB connection
  - – 32 MB RAM (minimum)
  - – 2 MB available disk space
  - – Windows operating system (Win98/NT/Me/2000/XP)

For developers:

- Host (PC) with LPT+USB connection
- COP8 C-compiler, like CodeCraft or IAR
- COP8 ISP Flash Loader application, to reprogram the COP8CBE9
- Borland C/C++ or Microsoft Visual C++ or other high-level language compiler
- No Microsoft DDK needed

2.3 BOARD INSTALLATION AND POWER-UP

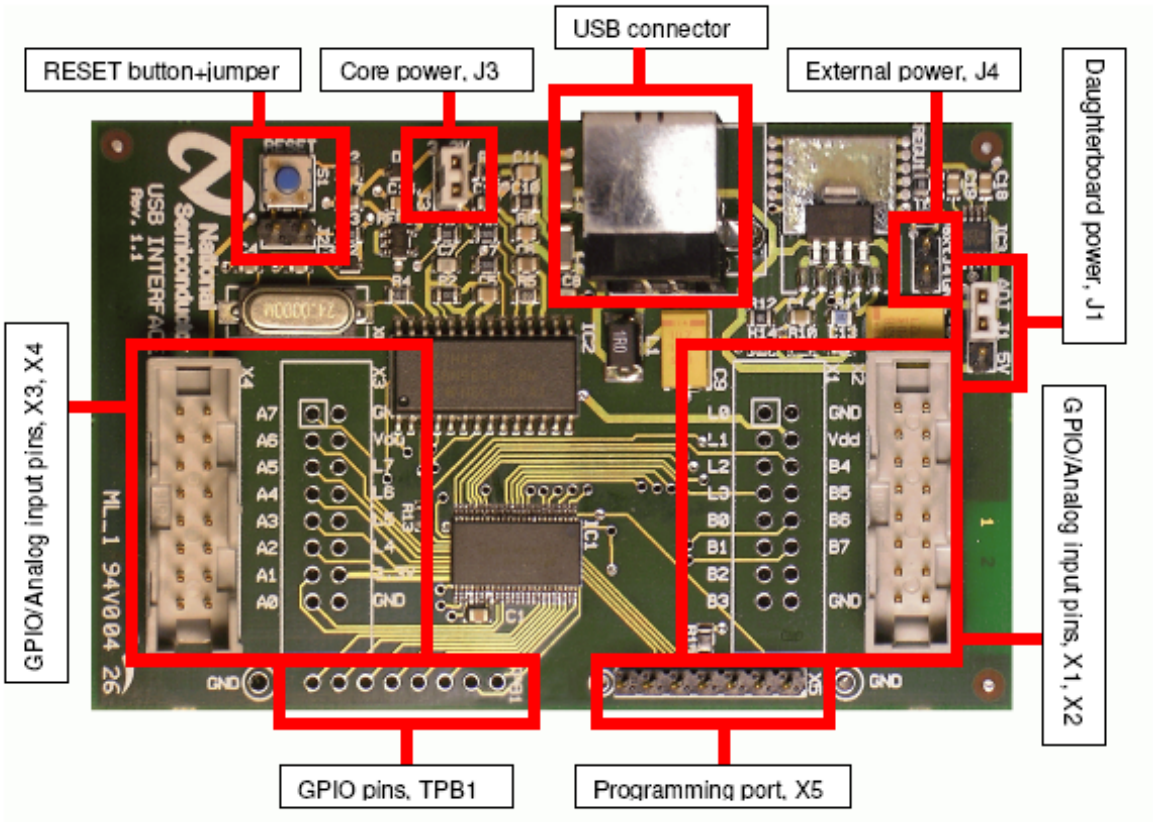


Figure 2.1 Jumper and Connector Locations of the USB Interface Module

2.4 JUMPER DESCRIPTIONS

The table below lists the on-board jumpers.

Table 2.1 Jumpers of the USB Interface Module

Jumper	Name	Setting	Description
J1	Daughterboard power	1-2	Provides adjustable voltage (3.0 / 3.9V) to daughterboard
		2-3	Provides 5V or External power to daughterboard
J2, S1	Reset	Short	Hardware reset to COP8CBE9 and USBN9604 chips
J3	Core power	Short	Provides 3.3V to COP8CBE9 and USBN9604 chips.
J4	External power	-	Stand-alone power input

3 PROGRAMMING

3.1 INTRODUCTION

This chapter describes how to program the flash memory of the COP8 microcontroller.

Note: The USB Interface Module is pre-programmed and can be used “as is” for evaluation purposes. Use this chapter only if you wish to change/reload the complete program code (firmware). For information on upgrading a current firmware through the USB port contact National Semiconductor.

3.2 PROGRAMMING SETUP

The USB Interface Module is programmed by loading the program code (firmware) into the flash memory of the COP8 controller through the parallel port. To (re)program the flash memory of the COP8, perform the steps below:

1. Connect programming cable to port X5 and parallel port of the PC.
2. Power the USB Interface Board by connecting a USB cable to board and the USB port of the PC.
3. Reset the COP8 microcontroller by pressing the RESET button. **Note:** If a microcontroller is in running mode, force it to In-System Programming mode (see datasheet of the COP8CBE9, Chapter: In-System Programming). To protect other circuitry before adding double voltage, remove resistor R13 temporarily.
4. Download the code using the ISP Flash Loader program and set COP8 into running mode. Always use the latest loader version available at National’s website.
5. Remove programming cable and press the RESET button.

3.3 PROGRAMMING CABLE

The pin configuration of the COP8 programming cable is shown in the following table:

Table 3.1 Pin Configuration of Programming Cable

USB Interface Board, port X5	Parallel port	Signal Name
4	10	SO
5	1	SCK
6	2	SI
7	18-25	Signal Ground

4 THEORY OF OPERATION

4.1 INTRODUCTION

This chapter describes the operation of the major components and features that comprise the USB Interface Module.

4.2 USB INTERFACE MODULE

The major components of the USB Interface Module are shown in Figure 4.1.

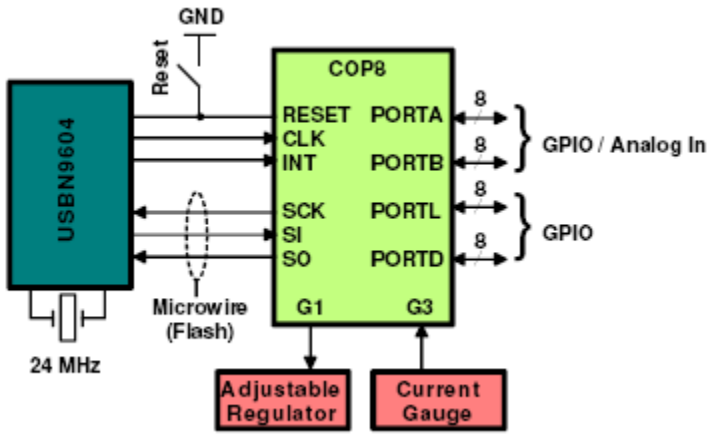


Figure 4.1 Major Components of the USB Interface Module

The USB Interface Module contains:

- COP8 microcontroller with built-in 256 byte RAM and 8Kb fFash EPROM
- USBN9604 high-speed USB node controller
- Adjustable regulator
- Current gauge for current measurement
- Microwire connection for COP8 internal flash programming
- 32 general purpose I/O pins

The main core of the USB Interface Module is a COP8 microcontroller. The COP8 interfaces to the USB controller and all the other devices and is used as the system controller. The COP8 internal flash memory and RAM are used to store the USBN9604 firmware and application specific code and to run it. System clock speed after startup is 9.6 MHz provided by USB9604 controller. Pre-programmed firmware allows manipulating all GPIO pins thought PC's USB port. Also low-level functions like flashing an EPROM through USB are supported.

4.3 POWER AND CURRENT GAUGE

The USB Interface Module needs at least 4V DC to create its internal power. This can be supplied by one of the following:

- 5V from the USB bus (i.e., bus powered)
- External DC adapter connected to J4 (i.e., self powered)

From the 5V DC source, the USB Interface Module creates two supplies (3.3V and adjustable voltage) with the help of two regulators as shown in Figure 4.2.

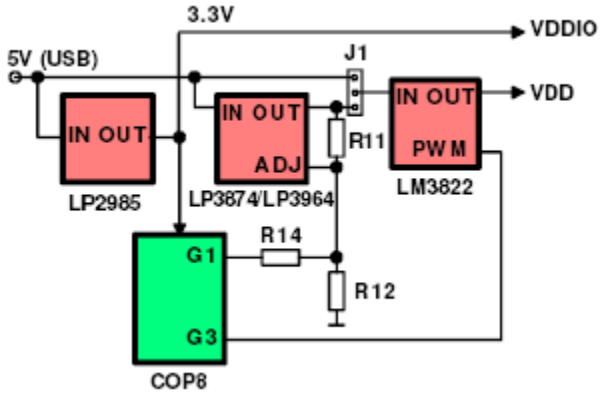


Figure 4.2 Power Management of the USB Interface Module

3.3V is for powering a COP8 and the USBN9604 controllers. There are three choices for adjustable voltages:) 3V, 3.9V, or 5V DC source (selectable by jumper J1). Adjustable voltages are defined by using resistors R11, R12, R14, and are user selectable using pin G1 of COP8. When G1 is programmed to low (ground), resistors R12 and R14 are connected in parallel. If G1 is in the Hi-Z state, only R12 affects the output voltage and R14 is disconnected. The adjustable voltages can be calculated using the following equation:

$$VDD = 1.216(1 + \frac{R11(R12 + R14)}{R12R14}$$

The USB Interface Module is generally a bus-powered device. It uses the 4.4V-5.0V max 0.5A bus-powered lines of the USB and creates the power supplies it needs. Even 5.5V DC source voltage can be used. The USB connection supports hot plug in/out.

Current consumption of the user application (daughterboard) can be measured using a current gauge chip. The sense range is between 0...1A. The PWM output of the chip is connected to the pin G3 of the COP8 microcontroller. The measuring cycle takes about 400ms.

4.4 A/D CONVERTER

The COP8 microcontroller has a built-in 10-bit analog-to-digital converter with 16 channel analog multiplexer. When the desired measurement voltage range is over VDDIO of COP8, simple resistor divider (R1, R2) can be added as shown in Figure 4.3. The filter capacitor C can be added to smooth the signal if necessary.

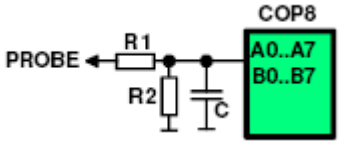


Figure 4.3 Analog Input Circuitry

The value of measured voltage can be calculated using the following equation:

$$Vprobe = \frac{VDDIO * ADC(R1 + R2}{1024 * R2}$$

5 SOFTWARE

5.1 INTRODUCTION

This chapter provides a short description of the firmware (FW), which runs on the USB Interface board and describes how to program and control the USB Interface from the PC. Use the COP8CBE9 datasheet for additional information on this process.

5.2 COP8 FIRMWARE

Most of the USB firmware code is imported from the COP8 USB Code Examples Library which is available at National's website. The USB Interface firmware runs on a COP8 core. It runs from flash memory and uses on-chip RAM for its variables. Current firmware supports the following functions and features:

USB related:

- Control packet transfer
- Human Interface Device (HID) class (USB joystick)

Application related:

- Internal Flash memory read/write/erase
- Internal RAM read/write
- Configuration/manipulation of I/O pins
- I²C/MICROWIRE/SPI interface transactions
- A/D conversion
- Internal timer programming

Basic structure of the firmware and relations between functions/services are shown in the following figure.

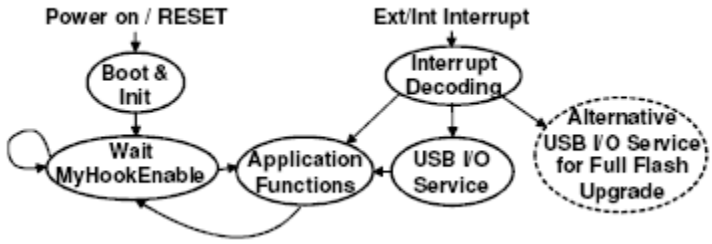


Figure 5.1 Structure and Relations of the Functions and Services

1. **Boot and Initialization**
   This is the code that initializes the microcontroller and all peripheral circuitry after power-on or RESET.
2. **Wait Loop**
   This is the code loop that keeps the microcontroller busy when there is nothing more pressing to do. These are the tasks that must be done regularly, but are not time critical and can be interrupted. From here it is possible to jump to application service when needed using flag MyHookEnable.
3. **Interrupt Decoding**
   This is the code that decodes the interrupt and decides what type of services are needed. The interrupt can initiate from the USB9604 controller, an external pin or an internal timer. These are the tasks that are time critical, and that must be serviced as soon as conditions warrant. These also tend to be the tasks that are asynchronous or that occur irregularly.
4. **USB I/O Service**
   This is the code that supports all USB related functions including control packet transfers and application command decoding. This contains most of the “driver” code is found.
5. **Application Functions**
   This is the code that is application specific. The user can add functions and modify the code.
6. **Alternative USB I/O Service**
   This is the code that should support the flashing process over the entire flash memory. This memory area can be used for making a copy of the main USB I/O service during an erase of a main block of the flash memory. That function is not fully implemented yet but is already supported by other services.

5.2.1 FLASH MEMORY MAP

Table 5.1 Flash Memory Map of the Firmware

Flash Memory Address (HEX)	Function
0000-0008	Boot
0009-00E8	Low level functions
00FF-01B0	Interrupt decoding
01E0-01FF	Interrupt table
0200-03FF	Constants
0400-04CA	Initialization
04CB-0C55	USB I/O service
0E00-167E	Application functions
1E00-1FFE	Alternative USB I/O service

5.2.2 CONTROL CODES

The Control Packet transfer is used to communicate between the USB Interface Module and the PC. The Standard Device Request GET_DESCRIPTOR is extended in current firmware and used to send application specific commands to the USB Interface Module using extended commands. The format of the extended packet of the Standard Device Request is shown in Table 5.2.

Table 5.2 Extended Packet of the Standard Device Request

Field name:	Request type	Request	CMD	VALUE	DATA	ADDR	ID	DATA1
Value:	0x80	GET_DESCRIPTOR	*Command*	*SubCmd*	*Data*	*Address*	*Id*	1

Format of the 4-byte return packet is shown in following figure:

Data1	*Data2*	*Data3*	*Data4*

Figure 5.2 Return Packet

Extended commands of the Standard Device Request are shown in Table 5.3:

Table 5.3 Extended Commands

Command	SubCmd	Symbol (defined in sw_hw.h)	Description
0x24	0x00	CMD_INIT	Sets regulator (pin G1) to its default value, 3.0V; disables internal timers (PWM function); configures some pins when *Data* = 0 (see Chapter 5.3.2)
	0x08	CMD_READRAM	Reads byte from internal RAM at *Address*
	0x10	CMD_WRITE	I²C/MICROWIRE/SPI write transaction, where the serial interface is given by two LSB bits of *Command*[1:0]: 0=I²C, 1=MICROWIREW, 2=SPI, 3=I²C2; address is given by variable *Address*; data is given by *Data*; I²C ID is given by *Id*. (Note 1)
	0x18	CMD_WRITERAM	Writes *Data* byte to internal RAM *Address*
	0x20	CMD_READ	I²C/MICROWIRE/SPI read transaction, where the serial interface is given by two LSB bits of *Command*[1:0]: 0=I²C, 1=MICROWIRE, 2=SPI, 3=I²C2; address is given by variable *Address*; I²C ID is given by *Id*. Data will be stored to *Data1*. (Note 1)
	0x28	CMD_MEASURECURRENT	Measures period of PWM cycle generated by current gauge chip. Returns 16-bit "high" pulse width value to *Data1* and *Data2*; "low" pulse width value to *Data3* and *Data4*. If measure fails, error code will be stored to *Data2*.

	0x30	CMD_MEASURE	A/D conversion from channel *Address*. Firmware turns selected pin into analog input if that is not configured before and restores old configuration after conversion.
	0x40	CMD_OUT	Writes *Data* byte to port A
	0x50	CMD_IN	Reads byte from port L
	0x60	CMD_SETREG	Turns G1 output to GND or Hi-Z state where output voltage is given by *Data* (1=3.0V, 0=3.9V)
	0x70	CMD_PWMMODE	Internal PWM-timer control (affects pin A5), where control byte is given by *Data*: 0=disable interrupt, 1=stop timer, 2=set duty cycle, 3=start timer. High-level pulse width is given by *Address*, low-level width by *Id*.
0x25	0x90	UPGRADECMD_STORE	Transfers 2 byte of data (value *Data* and *Address*) to internal RAM buffer
	0xA0	UPGRADECMD_FLASH	Flashes N-bytes from internal RAM buffer to flash memory. N is given by 3 LSB bit of *SubCmd*. 16-bit flash memory address is given by *Address* (MSBs) and *Data* (LSBs)
	0xC0	UPGRADECMD_ERASE	Erases 64-byte block in flash memory, where 16-bit flash memory address is given by *Address* (MSBs) and *Data* (LSBs)
	0xD0	UPGRADECMD_READ	Reads 2 bytes from flash memory, where 16-bit flash memory address is given by *Address* (MSBs) and *Data* (LSBs). Data bytes will be stored to *Data1* and *Data3*. Value *Id* must be zero.
	0xE0	UPGRADECMD_WRITE	Flashes *Id* byte to flash memory, where 16-bit flash memory address is given by *Address* (MSBs) and *Data* (LSBs)

Note1: There is an available feature option when MICROWIRE/SPI interface mode is selected. These features can be used by variable Id and are listed in Table 5.4.

Table 5.4 Features of the MICROWIRE/SPI Interface

Id (binary)	Feature
11xx xxxx	Inverts CS/SS line (pin A7)
1x1x xxxx	*Address* value will be 8-bit wide, no adding a read/write bit (LSB) by firmware
1xx1 xxxx	Swaps SI/SS pins (pins A0, A7)
1xxx 1xxx	Pulls SCL low at the end of R/W cycle (pin A6)

The error codes that firmware returns are listed in Table 5.5:

Table 5.5 Error Codes and Descriptions

Error Code	Symbol	Description
0x40 (bit 6)	ERROR_NOTINIT	This bit indicates the lack of initialization of the USB interface module. Last command has performed with default (BOOT) configuration. Requires initialization command.

0x00	ERROR_OK	Command performed successfully
0x01	ERROR_NA	Command performed but without error detection. For example MICROWIRE/SPI interfaces hasn't error detection feature.
0x82	ERROR_UNDEF	Undefined command
0x84	ERROR_USBIO	USB I/O error. Possible reasons: No hardware available or malfunctioning module or software problem
0x85	ERROR_NOACK0	No acknowledge signal from I²C slave device after ID has been sent
0x86	ERROR_NOACK1	No acknowledge signal from I²C slave device after Address has been sent
0x87	ERROR_NOACK2	No acknowledge signal from I²C slave device after Data has been sent

5.2.3 SERIAL INTERFACES

The firmware of the USB Interface Module supports three interface modes:

- I²C compatible interface (2 wire, serial)
- MICROWIRE interfaces (4 wire, serial)
- SPI interfaces (4 wire, serial)

The following table shows the pin configuration of the COP8 controller for all interface modes. Note that current firmware behaves as a MASTER of the serial interfaces.

Table 5.6 COP8 Pin Configurations

Interface	Pin Name	COP8 Pin	Comment
I²C compatible	SCL (clock) SDA (data in/out)	A6 (out/HiZ) A7 (out/HiZ), L6 (in)	Use pull up resistor for SCL. Use pull up resistor for SDA. Pin A7 must be connected to L6.
SPI / MICROWIRE	SCK (clock) SI (data in) SO (data out) SS (chip select)	A6 (out) L7 (in) A0 (out) A7 (out)	

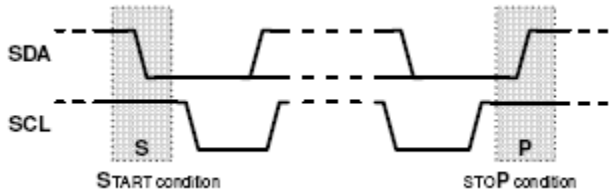
I²C Compatible Interface

I²C Signals

In I²C-compatible mode, the SCL pin is used for the I²C clock and the SDA pin is used for the I²C data. Both these signals need a pull-up resistor according to I²C specification. The values of the pull-up resistors are determined by the capacitance of the bus (typ. ~1.8k). See I²C specifications from Philips for further details. The period of the one clock pulse (SCL) is approximately 60µs.

I²C START and STOP Conditions

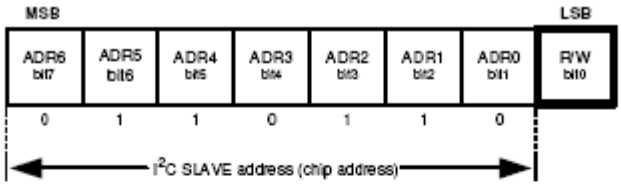
START and STOP bits classify the beginning and the end of the I²C session. START condition is defined as SDA signal transitioning from HIGH to LOW while SCL line is HIGH. STOP condition is defined as the SDA transitioning from LOW to HIGH while SCL is HIGH. The I²C master always generates START and STOP bits. The I²C bus is considered to be busy after START condition and free after STOP condition. During data transmission, the I²C master can generate repeated START conditions. The first START and the repeated START conditions are equivalent, function-wise.



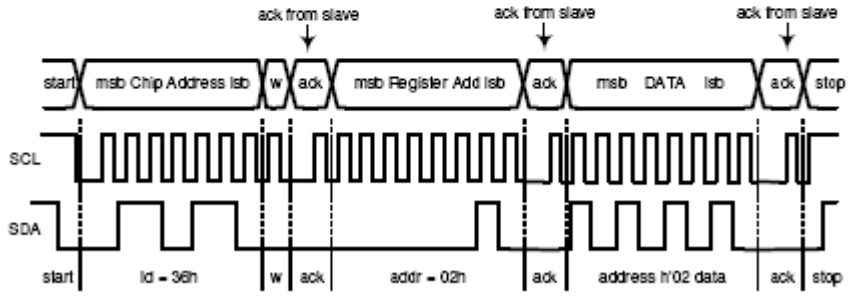
START and STOP Conditions

Transferring Data

Every byte put on the SDA line is eight bits long, with the most significant bit (MSB) being transferred first. Each byte of data has to be followed by an acknowledge bit. The transmitter (master) releases the SDA line (HIGH) during the acknowledge clock pulse. The receiver/slave must pull down the SDA line during the 9th clock pulse, signifying an acknowledge. A receiver which has been addressed must generate an acknowledge after each byte has been received. After the START condition, the I²C master sends a given chip address. This address is seven bits long followed by an eighth bit which is a data direction bit (R/W). Note that firmware skips to send a chip address in 2-cycle mode (selectable by user application). The second byte selects the register to which the data will be written. The third byte contains data to write to the selected register. When the MSB bit of the chip address is set to high, the firmware checks the level of the SCL line at 1st clock pulse of every cycle and waits a maximum 500ms until the receiver/slave releases an SCL line.

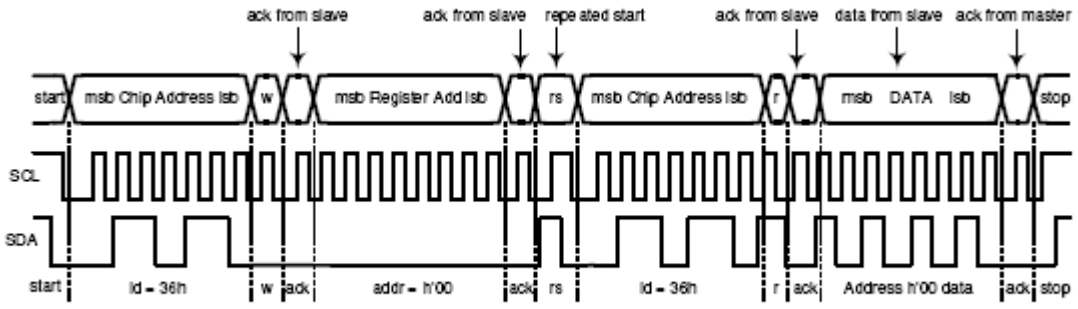


I²C Chip Address



I²C Write Cycle

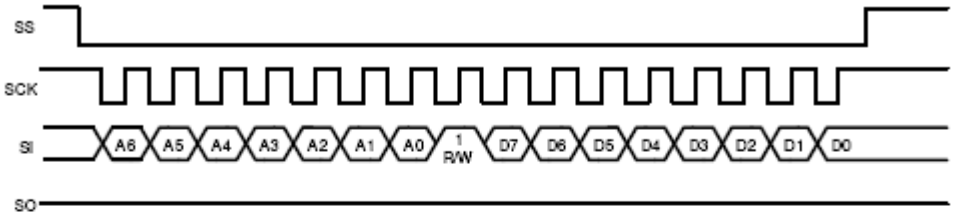
w = write (SDA = “0”)
r = read (SDA = “1”)
ack = acknowledge (SDA pulled down by either master or slave)
rs = repeated start
id= chip/device address
When a READ function is to be accomplished, a WRITE function must precede the READ function, as shown in the Read Cycle waveform.



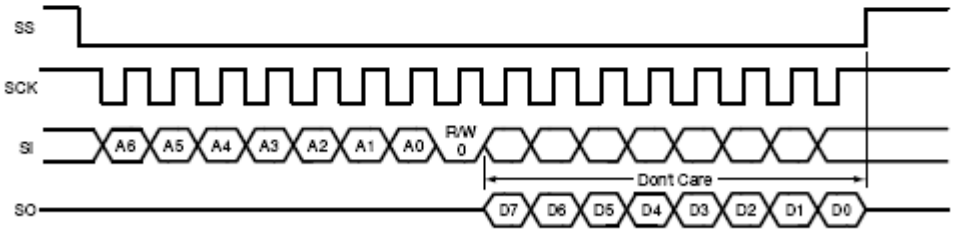
I²C Read Cycle

MICROWIRE / SPI Interface

The transmission consists of 16-bit Write and Read Cycles. One cycle consists of 7 Address bits, 1 Read/Write (R/W) bit and 8 Data bits. R/W bit high state defines a Write Cycle and low defines a Read Cycle. SO output is normally in the high-impedance state and it is active only when Data is sent out during a Read Cycle. A pull-up or pull-down resistor may be needed in the SO line if a floating logic signal can cause unintended current consumption in the input where SO is connected.
The Address and Data are transmitted MSB first. Data is clocked in on the rising edge of the SCK clock signal, while data is clocked out on the falling edge of SCK. Current firmware offers several features that can be used in MICROWIRE/SPI mode (see Table 5.4). the period of one clock pulse (SCK) is approximately 50µs.



SPI Write Cycle



SPI Read Cycle

5.3 PC APPLICATION SOFTWARE

The Windows operating system (OS) interprets USB Interface Module as a Human Interface Device (HID) and uses Microsoft's standard HID-driver (included in OS). The application code developer can access the USB Interface through the Dynamic Link Library (DLL) provided by National Semiconductor or by using standard Windows API functions. No special Windows DDK Development Kit is needed.

5.3.1 USING A DYNAMIC LINK LIBRARY

The precompiled DLL offers several functions, which can be used by application software. The most important functions and their usage are listed below. All used constants are defined in file *sw_fw.h*.

DllUSBDeviceDetect

The **DllUSBDeviceDetect** function detects, opens, and configures the USB Interface Module, if available. This is to be used once when the application starts.

UCHAR DllUSBDeviceDetect(UCHAR *Config***);**

Parameters
Config
Specifies the predefined configuration to the COP8 pins. The predefined configurations are shown in Table 5.6.

Return Value
If the function succeeds, the return value is ERROR_OK.
If the function fails, the return value is ERROR_USBIO.

DllUSBCloseDevice

The **DllUSBCloseDevice** closes the USB Interface connection. This is to be used once when the application exits.
VOID DllUSBCloseDevice(VOID);

DllReadFromDevice

The **DllReadFromDevice** function reads data byte from application hardware using I²C/MICROWIRE/SPI interface through the PC's USB/LPT Interface Module.

UCHAR DllReadFromDevice(WORD *wPort***, UCHAR** *uAddr***, UCHAR*** *uData***, UCHAR** *Interface***, UCHAR** *Id***);**

Parameters
wPort
Specifies a PC I/O port address, where the USB/LPT Interface module is connected.
The following values can be used:

Value	**Port**
0x2F8	USB port
0x278, 0x378, 0x3CB	Parallel ports (LPT)

uAddr
Specifies an address field of the I²C/MICROWIRE/SPI interface.
uData
Pointer to a variable that receives the data byte.

Interface

Specifies a serial interface for data transfer. Timing diagrams can be found in Section 5.2.3. The following values are defined:

Value	Interface
IF_I2C	I²C
IF_MICROWIRE	MICROWIRE
IF_SPI	SPI
IF_I2C2	I²C (2-cycle mode)

Id

Specifies an ID field of the I²C interface. When MICROWIRE/SPI interface is selected, that value specifies a feature option. See more about the available features in Table 5.4.

Return Value

If the function succeeds, the return value is dependent on the selected serial interface: ERROR_OK or ERROR_NA.
If the function fails, the return value can be ERROR_USBIO, ERROR_NOACK0, ERROR_NOACK1, or ERROR_NOACK2

See more about error codes in Table 5.5.

Remarks

This function performs initialization of the USB Interface Module when it is not initialized. If the module isn't initialized using the **DllUSBDeviceDetect** function, internal initialization procedure uses the default configuration (see Table 5.6).

DllWriteToDevice

The **DllWriteToDevice** function writes the data byte to application hardware using I²C/MICROWIRE/SPI interface through the PC's USB/LPT Interface Module.

UCHAR DllWriteToDevice(WORD *wPort***, UCHAR** *uAddr***, UCHAR** *uData***, UCHAR** *Interface***, UCHAR** *Id***);**

Parameters

wPort

Specifies a PC I/O port address, where the USB/LPT Interface Module is connected. The following values can be used:

Value	Port
0x2F8	USB port
0x278, 0x378, 0x3CB	Parallel ports (LPT)

uAddr

Specifies the address field of the I²C/MICROWIRE/SPI interface.

uData

Specifies the data field of the I²C/MICROWIRE/SPI interface.

Interface

Specifies a serial interface for data transfer. Timing diagrams can be found in Section 5.2.3. The following values are defined:

Value	Interface
0	I²C
1	MICROWIRE
2	SPI
3	I²C (2-cycle mode)

Id

Specifies the ID field of the I²C interface. When MICROWIRE/SPI interface is selected, that value specifies a feature option. See more about the available features in Table 5.4.

Return Value
If the function succeeds, the return value is dependent on the selected serial interface: ERROR_OK or ERROR_NA.
If the function fails, the return value can be ERROR_USBIO, ERROR_NOACK0, ERROR_NOACK1, or ERROR_NOACK2.

See more about error codes in from Table 5.5.

Remarks
This function performs initialization of the USB Interface Module when it is not initialized. If the module is not initialized using **DllUSBDeviceDetect** function, internal initialization procedure uses the default configuration (see Table 5.6).

DllUSBSendCmd_ReadFromPortAconfig (equivalent to ports B and L, change only letter)

The **DllUSBSendCmd_ReadFromPortAconfig** function reads the byte from the COP8 configuration register of port A.

UCHAR DllUSBSendCmd_ReadFromPortAconfig(UCHAR **uConfig***);*

Parameters
uConfig

Pointer to a variable that receives the configuration byte of port A.

Return Value
If the function succeeds, the return value is ERROR_OK.
If the function fails, the return value is ERROR_USBIO.
Remarks
This function performs initialization of the USB Interface Module when it is not initialized. If the module is not initialized using **DllUSBDeviceDetect** function, internal initialization procedure uses the default configuration (see Table 5.6).

DllUSBSendCmd_WriteToPortAconfig (equivalent to ports B and L, change only letter)

The **DllUSBSendCmd_WriteToPortAconfig** function writes the byte to the COP8 configuration register of port A.

UCHAR DllUSBSendCmd_WriteToPortAconfig(UCHAR **uConfig***);*

Parameters
uConfig

Specifies the configuration byte of port A.

Return Value
If the function succeeds, the return value is ERROR_OK.
If the function fails, the return value is ERROR_USBIO.

Remarks
This function performs initialization of the USB Interface Module when it is not initialized. If the module is not initialized using **DllUSBDeviceDetect** function, internal initialization procedure uses the default configuration (see Table 5.6).

DllUSBSendCmd_ReadFromPortA (equivalent to ports B and L, change only letter)

The **DllUSBSendCmd_ReadFromPortA** function reads the byte from the COP8 data register of port A.

UCHAR DllUSBSendCmd_ReadFromPortA (UCHAR **uData***);**

Parameters
uData
Pointer to a variable that receives the data byte of port A.

Return Value
If the function succeeds, the return value is ERROR_OK.
If the function fails, the return value is ERROR_USBIO.
Remarks
This function performs initialization of the USB Interface Module when it is not initialized. If the module is not initialized using **DllUSBDeviceDetect** function, internal initialization procedure uses the default configuration (see Table 5.6).

DllUSBSendCmd_WriteToPortA (equivalent to ports B and L, change only letter)

The **DllUSBSendCmd_WriteToPortA** function writes the byte to the COP8 data register of port A.

UCHAR DllUSBSendCmd_WriteToPortA (UCHAR **uData***);**

Parameters
uData
Specifies the data byte of port A.

Return Value
If the function succeeds, the return value is ERROR_OK.
If the function fails, the return value is ERROR_USBIO.
Remarks
This function performs initialization of the USB Interface Module when it is not initialized. If the module is not initialized using **DllUSBDeviceDetect** function, internal initialization procedure uses the default configuration (see Table 5.6).

DllUSBSendCmd_Measure

The **DllUSBSendCmd_Measure** function measures voltage from one of the selected analog channel (ports B or L of COP8).

UCHAR DllUSBSendCmd_Measure (UCHAR *Channel*, **WORD** **Result***);**

Parameters
Channel
Specifies a channel of the analog input. More information about analog channels can be found in the COP8CBE9 datasheet.

Result

Pointer to a variable that receives the measured value. Interpretation of the value is described in Section 4.4.

Return Value
If the function succeeds, the return value is ERROR_OK.
If the function fails, the return value is ERROR_USBIO.
Remarks
This function performs initialization of the USB Interface Module when it is not initialized. If the module is not initialized using **DllUSBDeviceDetect** function, internal initialization procedure uses the default configuration (see Table 5.6).

DllUSBSendCmd_MeasureCurrent

The **DllUSBSendCmd_MeasureCurrent** function measures the current of the application hardware (daughterboard), which flows through the current gauge chip.

UCHAR DllUSBSendCmd_MeasureCurrent(WORD **Result****);**

Parameters
Result

Pointer to a variable that receives the measured value in milliamps.

Return Value
If the function succeeds, the return value is ERROR_OK.
If the communication fails, the return value is ERROR_USBIO. If the function fails (missing PWM signal generated by current gauge chip), the return value is ERROR_UNDEF.

DllUSBSendCmd_GetRegVoltage

The **DllUSBSendCmd_GetRegVoltage** function gets the state of the adjustable voltage regulator.

UCHAR DllUSBSendCmd_GetRegVoltage(UCHAR **State****);**

Parameters
State

Pointer to a variable that receives the state value. When state the is 0, the regulator is adjusted to 3.9V; when the state is 1, the regulator is adjusted to 3.0V.

Return Value
If the function succeeds, the return value is ERROR_OK.
If the function fails, the return value is ERROR_USBIO.

DllUSBSendCmd_SetRegVoltage

The **DllUSBSendCmd_SetRegVoltage** function sets the state of the adjustable voltage regulator.

UCHAR DllUSBSendCmd_SetRegVoltage (UCHAR **State***);**

Parameters

State

Specifies a state value. When the s*tate* is 0, the regulator is adjusted to 3.9V; when the state is 1, the regulator is adjusted to 3.0V.

Return Value

If the function succeeds, the return value is ERROR_OK.
If the function fails, the return value is ERROR_USBIO.

DllUSBSendCmd_GetFlashVersion

The **DllUSBSendCmd_GetFlashVersion** function gets a firmware version from the USB Interface Module, which is stored into the flash memory of COP8.

UCHAR DllUSBSendCmd_GetFlashVersion (UCHAR **Version1***, UCHAR** **Version2***);**

Parameters

Version1

Pointer to a variable that receives the version number. The variable *Version1* shows the version number of the USB I/O service code.

Version2

Pointer to a variable that receives the version number. The variable *Version2* shows the version number of the application code.

Return Value

If the function succeeds, the return value is ERROR_OK.
If the function fails, the return value is ERROR_USBIO.

There are less important functions available, but they are not covered in this document.

6 BOARD SPECIFICATIONS

6.1 INTRODUCTION

This chapter presents the specification of the on-board connectors.

6.2 CONNECTORS

Table 6.1 lists all on-board connectors. The remainder of this section provides the related pin assignments for each connector.

Table 6.1 USB Interface Module Connectors

Connector No.	Connector Name
X1, X2	Pair of GPIO connectors 1 with VDD
X3, X4	Pair of GPIO connectors 2 with VDD and VDDIO
TPB1	GPIO connector
X5	Programming connector

6.2.1 GPIO Connectors X1, X2

Type: 16-pin, dual row (both connectors are equal)

Pin No.	Pin Name	Pin No.	Pin Name
1	GPIO pin L0	2	GND
3	GPIO pin L1	4	VDD
5	GPIO pin L2	6	GPIO pin B4 or analog channel CH12
7	GPIO pin L3	8	GPIO pin B5 or analog channel CH13
9	GPIO pin B0 or analog channel CH8	10	GPIO pin B6 or analog channel CH14
11	GPIO pin B1 or analog channel CH9	12	GPIO pin B7 or analog channel CH15
13	GPIO pin B2 or analog channel CH10	14	Not Used
15	GPIO pin B3 or analog channel CH11	16	GND

6.2.2 GPIO Connectors X3, X4

Type: 16-pin, dual row (both connectors are equal)

Pin No.	Pin Name	Pin No.	Pin Name
1	GPIO pin A7 or analog channel CH7	2	GND
3	GPIO pin A6 or analog channel CH6	4	VDD
5	GPIO pin A5 or analog channel CH5	6	GPIO pin L7
7	GPIO pin A4 or analog channel CH4	8	GPIO pin L6
9	GPIO pin A3 or analog channel CH3	10	GPIO pin L5
11	GPIO pin A2 or analog channel CH2	12	GPIO pin L4
13	GPIO pin A1 or analog channel CH1	14	VDDIO (3.3V)
15	GPIO pin A0 or analog channel CH0	16	GND

6.2.3 GPIO Connector TPB1

Type: 8-pin, single row

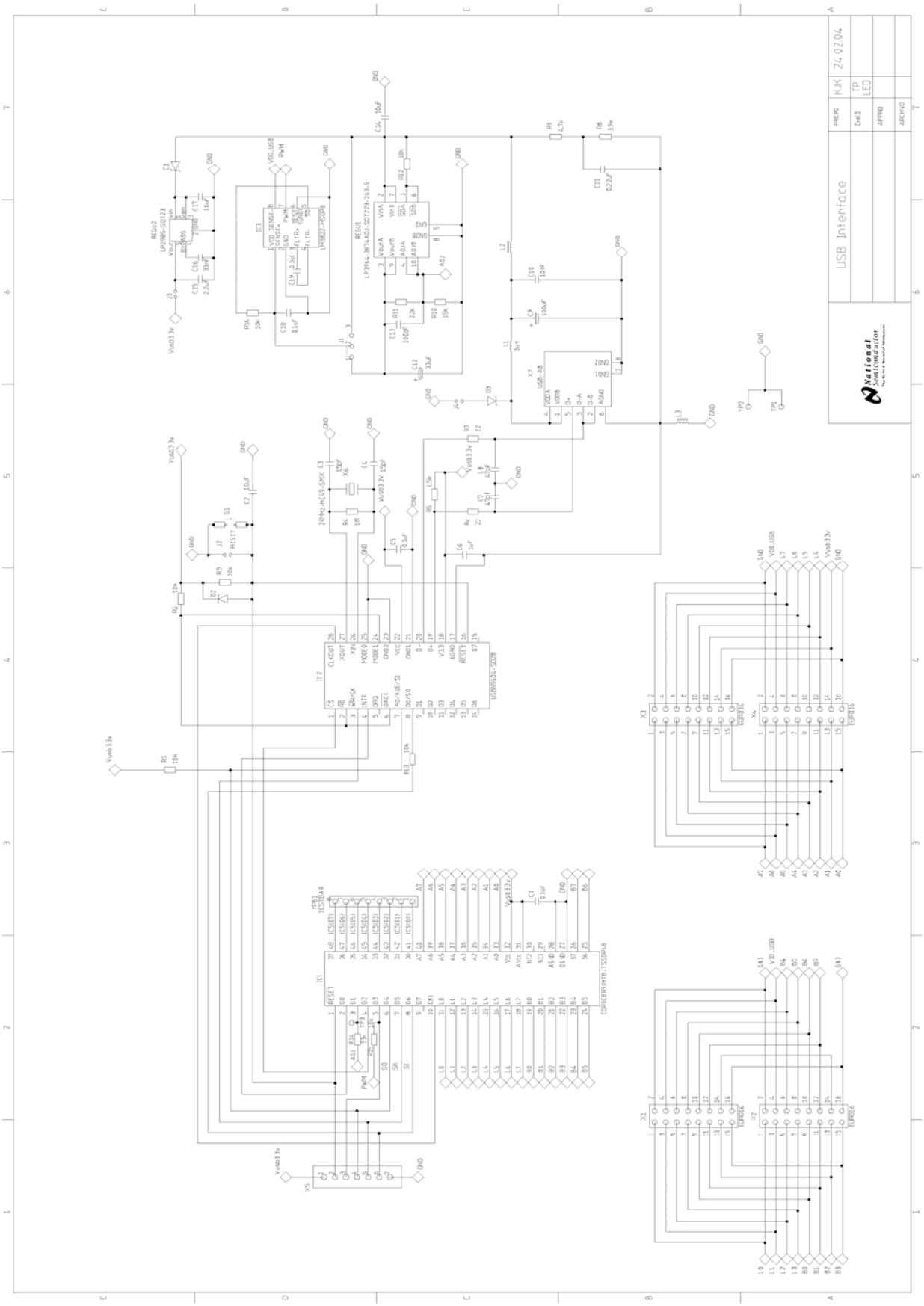
Pin No.	Pin Name
1	GPIO pin H7
2	GPIO pin H6
3	GPIO pin H5
4	GPIO pin H4
5	GPIO pin H3
6	GPIO pin H2
7	GPIO pin H1
8	GPIO pin H0

6.2.4 Programming Connector X5

Type: 7-pin, single row

Pin No.	Pin Name
1	VDDIO
2	$\overline{\text{RESET}}$
3	GPIO pin G3
4	SO
5	SCK
6	SI
7	GND

Appendix A ELECTRICAL SCHEMATIC



IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Mobile Processors	www.ti.com/omap		
Wireless Connectivity	www.ti.com/wirelessconnectivity		
	TI E2E Community Home Page		e2e.ti.com

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2012, Texas Instruments Incorporated