National Semiconductor

USBN9602 Interface Examples

By Jim Lyle, National Semiconductor

National Semiconductor's USBN9602 Universal Serial Bus Function Controller can be used in a wide variety of USB applications, and with a wide variety of microcontrollers. This white paper illustrates the latter using detailed examples. Specifically, it outlines the hardware and firmware required to connect the USBN9602 with COP8 (National Semiconductor), MC68HC11 (Motorola), and 80C188EB (Intel) microcontrollers[1].

Most of the firmware shown in this paper is drawn from a reference design for an HID Class[2] joystick. The complete source code for this firmware is available on National Semiconductor's web site[3]. While this joystick application is one specific example from one device class, the protocol mechanisms used are similar (or identical) to those required by most devices and most classes.

USB-Side Hardware Connections

The required USB-side hardware is shown in Figure 1. This circuitry should not change much from one application to the next, and much of it is primarily to reduce EMI emissions.

R1 is the pullup resistor which indicates to the host that this is a full-speed device. R3 and R4 are series termination resistors. C5 and C6 are high frequency bypass capacitors. L1 and L2 are ferrite beads on the USB power and ground connections. Bus power is available via the VUSB point if necessary. If not, L1 may be omitted.

The USBN9602 contains an internal 3.3V regulator sized to meet its own internal requirements and that of the pullup resistor. Do not use the internal regulator to power anything else. If desired, an

1. Strictly speaking, the 80C188EB is an integrated microprocessor (not a microcontroller) because it requires external memory. It is designed for embedded control applications, however, and so fits well with other true microcontrollers.
2. The USB Specification defines the basic framework for USB devices and the standard protocols they use. There are also device "Classes" which define additional protocol layers that reside above (and make use of) the standard protocols. Among these classes are Audio, Printer, Comm (Communications), and HID (Human Interface Device).
3. See http://www.national.com and search for USBN9602. From this devices home page, follow the link for "Design Tools'.

external regulator can be connected to the 3.3V pin. If you do this you must guarantee that the internal regulator remains disabled[1].

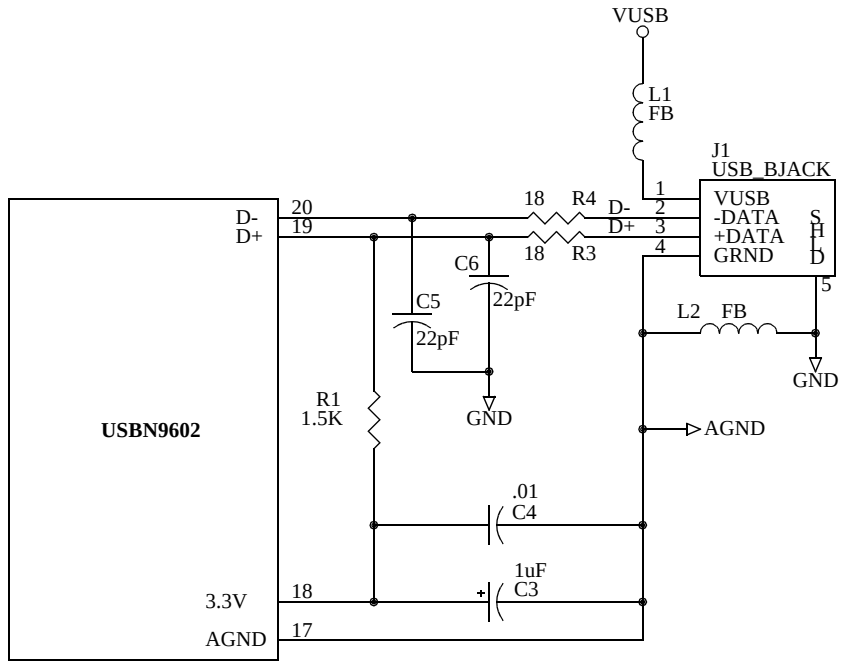

FIGURE 1. USB Side Connections to the USBN9602

Microcontroller-Side Specifics

The USBN9602 has a generic microcontroller interface designed to connect virtually any microcontroller or microprocessor with a minimum of additional circuitry. It provides both serial and parallel connections, and the latter may optionally take advantage of a DMA interface[2].

The following sections provide specific examples for the aforementioned list of microcontrollers.

COP8

National Semiconductor's COP8 microcontroller architecture defines a family of 8-bit controllers with a wide variety of features and functions. All members of the family have a MICROWIRE™ port, however. MICROWIRE is a four-wire serial interface used to connect external peripherals. It provides the means necessary to connect the USBN9602 in this case.

1. The internal regulator is disabled by default in the firmware discussed in this application. To enable it, the constant '*VREG_ON* must be defined at the top of the source code before compilation.
2. This requires a DMA controller in the target system.

The MICROWIRE connection between the COP8 microcontroller and the USBN9602 is shown in Figure 2. This is very simple, and only a few connections are necessary for the entire interface. **MODE0** and **MODE1** are both pulled high to select the MICROWIRE mode. The four MICROWIRE signals are labeled **/CS** (Chip Select), **SO** (Serial Output), **SK** (Serial Clock), and **SI** (Serial Input) and are wired straight across[1]. The **/RST** signal connects both the COP8 and the USBN9602, although the circuitry that drives it is not shown. Please see the COP8 Data Sheet for the appropriate reset circuit. The **DRQ** (DMA Request) and **/DACK** (DMA Acknowledgment) pins are respectively not connected and tied high. These are the proper connections to make when the DMA feature will not be used.

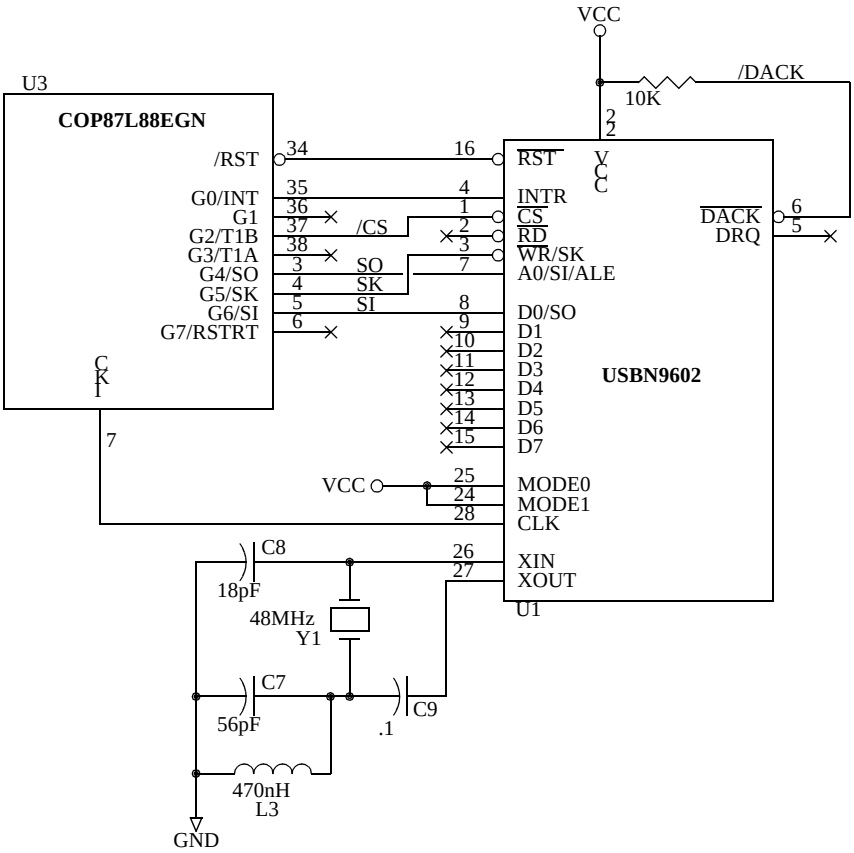

FIGURE 2. MICROWIRE Connection Between the COP8 and the USBN9602

Note that the connection between the USBN9602's **CLK** output pin and the COP8's **CKI** input is optional. The USBN9602 must have a 48 MHz third-overtone crystal for proper operation. This 48MHz internal clock is sub-divided by a firmware-selectable integer[2] and then driven onto the **CLK** pin. As shown here, this feature can eliminate the microcontroller's need for an additional crystal or oscillator.

The firmware required to communicate with the USBN9602 starts in Figure 3 with that used to configure the MICROWIRE port. This code enables the necessary output pins, selects the proper MICROWIRE mode and clock period, and then enables the MICROWIRE function.

1. The MICROWIRE signal names are referenced to the host (microcontroller). For example, the **SO** signal is the serial data output from the microcontroller. Therefore it must connect to the USBN9602's serial data input pin. Likewise, the **SI** signal must connect to the USBN9602's serial data output pin.
2. At reset, the divisor is preset to 12, producing a 4MHz clock on the **CLK** pin. The firmware can select a new divisor whenever necessary.

```
/*Configure the Microwire port                                    */
  PORTGC.SO  = 1;                    /*enable SO output          */
  PORTGC.SK  = 1;                    /*enable SK output          */
  PORTGC.SKSEL=0;                    /*selects norm. SK mode     */
  CNTRL.MSEL = 1;                    /*enable Microwire intf     */
  CNTRL.SL1  = 0;                    /*SK period = 2xTC          */
  CNTRL.SL0  = 0;                    /*SK period = 2xTC          */
```

FIGURE 3. COP8 MICROWIRE Configuration

Next, Figure 4 shows a series of macro definitions. These are used as building blocks for the following functions for two reasons. First, system specific details are better isolated here and make for more readable code further on. Second, the functions used to communicate with the USBN9602 are heavily used, and in-line code executes faster and is more efficient than nested function calls.

```
/* Send data out the microwire port **********************************/
#define MWOUT(dta) {                                                  \
            MWSR = dta;                    /*put in shft reg        */\
            PSW.BUSY = 1;                  /*start shifting         */\
            while (PSW.BUSY == 1);}        /*wait until done         */

/* Turn off all microwire chip selects *******************************/
#define MWCSOFF {USBCSOFF; EECSOFF; A2DCSOFF;}

/* send address and command out via the microwire port ***************/
#define MWADRCMD(adr,cmd) MWOUT((adr & 0x3F) | cmd)

/* Assert/De-assert the 9602 CS* signal ******************************/
#define USBCSENB SETBIT(PORTGC,BIT2);        /*enable the output bit  */
#define USBCSON {                                                     \
            MWCSOFF;                       /*de-assert chip sels   */\
            CLRBIT(PORTGD,BIT2);}          /*re-assert USB CS*      */
#define USBCSOFF SETBIT(PORTGD,BIT2);
```

FIGURE 4. COP8 Macros Used to Access the USBN9602

The *read_usb* and *write_usb* functions are shown in Figure 5. These are the actual functions that transfer data from and to the USBN9602, respectively, and they are similar to each other. First, the functions assert the USBN9602's chip select pin. Then they send the address and command (read or write) through the MICROWIRE port, and then receive or send the data the same way. Finally the functions de-assert the chip select pin, then *read_usb* returns the data.

```
/*********************************************************************/
/* This subroutine reads the USB register whose address is given.    */
/*********************************************************************/
byte read_usb(byte adr)
  {
        USBCSON;                            /*turn on CS              */
        MWADRCMD(adr,USBREAD);              /*send cmd and addr       */
        MWOUT(0);                           /*send dummy data         */
        USBCSOFF;                           /*turn off CS             */
        return(MWSR);                       /*return the result       */
  }

/*********************************************************************/
/* This subroutine writes the USB register whose address is given.   */
/*********************************************************************/
void write_usb(byte adr, byte dta)
  {
        USBCSON;                            /*turn on CS              */
        MWADRCMD(adr,USBWRITE);             /*send cmd and addr       */
        MWOUT(dta);                         /*send the data           */
        USBCSOFF;                           /*turn off CS             */
  }
```

FIGURE 5. COP8 *read_usb* and *write_usb* Routines

MC68HC11

Motorola's MC68HC11 microcontroller is a popular choice for embedded applications. It does not have a MICROWIRE interface, but it does have something very similar called an SPI (Serial Peripheral Interface). This has a number of features and modes, including one (Mode 0[1]) which is essentially MICROWIRE compatible. Therefore an SPI connection to the USBN9602 can be used with this microcontroller.

The SPI connection is shown in Figure 6, and indeed is so similar to the MICROWIRE case that only a few additional comments are necessary here. The SPI signals are named **MISO** (Master In, Slave Out), **MOSI** (Master Out, Slave In), **SCK** (Serial Clock), and **/SS** (Slave Select). The crystal or oscillator circuit is not shown for simplicity, but the same circuit shown in Figure 2 may be used.

Note that the **INTR** pin in this case is low-true (it was high-true in the COP8 example). The **INTR** output is programmable. Its polarity and type (totem-pole or open-collector) can be selected by writing the appropriate data pattern to the associated USBN9602 register[2].

The required SPI firmware is likewise very similar to the MICROWIRE examples. The only differences are those relating to minor differences in the hardware. An example of the SPI configuration code is shown in Figure 7, the hardware specific macros are in Figure 8, and the *read_usb* and *write_usb* functions are in Figure 9. Note that the latter functions are essentially identical to the MICROWIRE equivalents but for the macro prefixes.

1. Not to be confused with the USBN9602's **MODE0** pin.
2. The **INTR** output is tri-stated after reset.

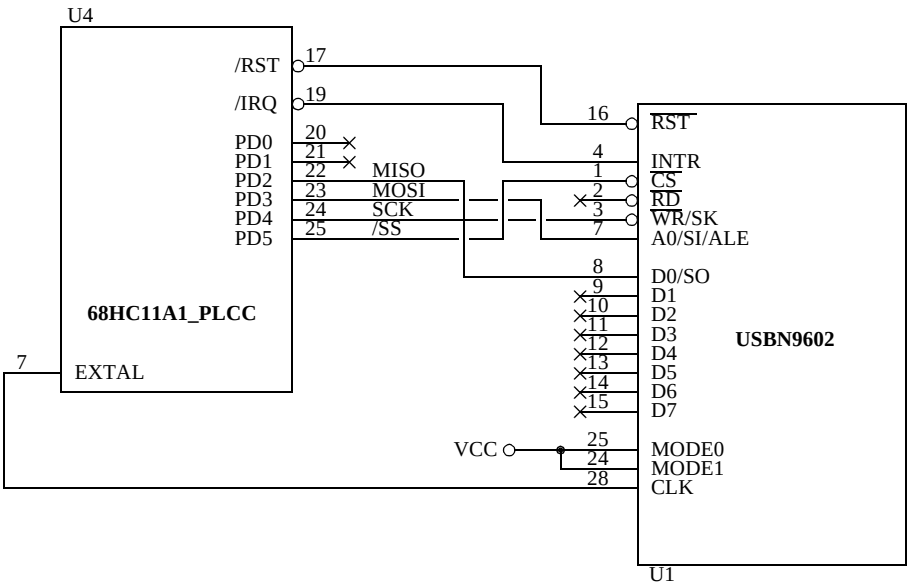

FIGURE 6. SPI Connection Between the MC68HC11 and the USBN9602

```
/*Enable the SPI port, putting it in Mode 0 (CPOL=0, CPHA=0),   */
/*Master mode, interrupts OFF, 1 MHz clock (with 8 MHz osc.),   */
/*Port D totem pole, bit 2 is input, all others outputs         */
/*Bit 5 is used as the CS* for the USBN9602                     */
  PORTD=0xFF;                          /*set to known value      */
  DDRD=0x3B;                           /*set PORTD[0..5] dir.    */
  SPCR=SPE+MSTR;                       /*enable, in master mode  */
  tmp=SPSR;                            /*flush the               */
  tmp=SPDR;                            /*  SPI port              */
```

FIGURE 7. MC68HC11 SPI Configuration

```
/* Send data out the SPI port ***************************************/
#define SPIOUT(dta) {                                               \
            SPDR = dta;                     /*put in shft reg      */\
            while (!TSTBIT(SPSR,SPIF));}    /*wait until done      */

/* Turn off all SPI chip selects ************************************/
#define SPICSOFF USBCSOFF

/* send address and command out via the SPI port ********************/
#define SPIADRCMD(adr,cmd) SPIOUT((adr & 0x3F) | cmd)

/* Assert/De-assert the 9602 CS* signal *****************************/
#define USBCSON {                                                   \
            SPICSOFF;                        /*de-assert chip sels */\
            CLRBIT(PORTD,BIT5);}             /*re-assert USB CS*    */
#define USBCSOFF SETBIT(PORTD,BIT5)
```

FIGURE 8. MC68HC11 Macros Used to Access the USBN9602

```
/*********************************************************************/
/* This subroutine reads the USB register whose address is given.    */
/*********************************************************************/
byte read_usb(byte adr)
  {
        USBCSON;                            /*turn on CS              */
        SPIADRCMD(adr,USBREAD);             /*send cmd and addr       */
        SPIOUT(0);                          /*send dummy data         */
        USBCSOFF;                           /*turn off CS             */
        return(SPDR);                       /*return the result       */
  }

/*********************************************************************/
/* This subroutine writes the USB register whose address is given.   */
/*********************************************************************/
void write_usb(byte adr, byte dta)
  {
        USBCSON;                            /*turn on CS              */
        SPIADRCMD(adr,USBWRITE);            /*send cmd and addr       */
        SPIOUT(dta);                        /*send the data           */
        USBCSOFF;                           /*turn off CS             */
  }
```

FIGURE 9. MC68HC11 *read_usb* and *write_usb* Routines

80C188EB

The COP8 and MC68HC11 use the USBN9602's serial (MICROWIRE) interface mode. The USBN9602 also supports two parallel modes: Multiplexed and Non-Multiplexed. The former is for microprocessors that multiplex their address and data buses, and the latter is for microprocessors and controllers that do not. The parallel modes require more wires, but provide higher bandwidth.

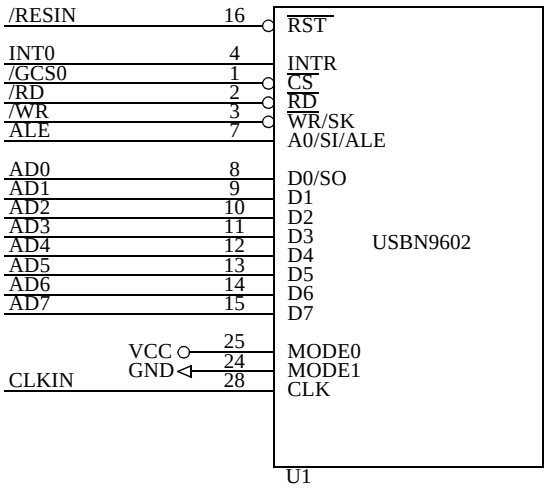

FIGURE 10. Parallel Connection (Multiplexed) Between the 80C188EB and the USBN9602

The 80C188EB microprocessor does indeed have a multiplexed address and data bus. This processor can be connected to the USBN9602 using the Multiplexed Mode as shown in Figure 10. The signals on the left correspond to 80C188EB pin names. Note in particular that **MODE0** is pulled high and **MODE1** is pulled low, selecting the Multiplexed Mode. Note also that the '188's **ALE** signal (Address Latch Enable) signal is used to indicate that a valid address is on **AD[0..7]**.

The 80C188EB single-board computer used to test this application has an address latch built into it, and indeed does not even bring the **ALE** signal out to the expansion connector used to connect the USBN9602. Therefore in this case it was possible (and necessary) to use the Non-Multiplexed Mode, as shown in Figure 11. Note that both **MODE0** and **MODE1** are pulled low, and that what was the **ALE** signal has been replaced by the latched (de-multiplexed and buffered) **BA[0]** bit. Otherwise the connections remain the same.

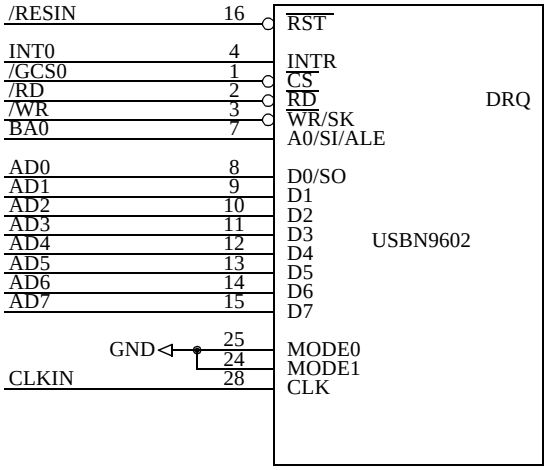

FIGURE 11. Parallel Connection (Non-Multiplexed) Between the 80C188EB and the USBN9602

```
/*********************************************************************/
/* This subroutine reads the USB register whose address is given.    */
/*********************************************************************/
byte read_usb(byte adr)
  {
        outp((IO_9602+1),adr);              /*write to address reg    */
        return(inp(IO_9602));               /*return the reg data     */
  }

/*********************************************************************/
/* This subroutine writes the USB register whose address is given.   */
/*********************************************************************/
void write_usb(byte adr, byte dta)
  {
        outp((IO_9602+1),adr);              /*write to address reg    */
        outp((IO_9602),dta);                /*write the reg data      */
  }
```

FIGURE 12. 80C188EB *read_usb* and *write_usb* Routines (Non-Multiplexed Mode)

The firmware required for these parallel modes is generally much simpler than for the serial interfaces because more of the job is done in hardware by the microcontroller's bus interface. The only configuration that is necessary is to initialize the chip select logic used (in this case **/GCS0**). There are no macros defined because hardware does all of those operations. Therefore the *read_usb* and *write_usb* functions simplify as shown in Figure 12. Here, *IO_9602* is defined as the base address of the USBN9602. In the Non-Multiplexed Mode the register address must first be written to the address register (at location *IO_9602+1*), then the register can be read from or written to (at location *IO_9602*).

The equivalent Multiplexed Mode functions are even simpler. These are shown in Figure 13. Note that here the functions have reduce to a single statement. In this case it would probably make more sense to replace these functions with macros for even greater efficiency.

```
/*********************************************************************/
/* This subroutine reads the USB register whose address is given.    */
/*********************************************************************/
byte read_usb(byte adr)
  {
        return(inp(IO_9602+adr));           /*return the reg data     */
  }

/*********************************************************************/
/* This subroutine writes the USB register whose address is given.   */
/*********************************************************************/
void write_usb(byte adr, byte dta)
  {
        outp((IO_9602+adr),dta);            /*write the reg data      */
  }
```

FIGURE 13. 80C188EB *read_usb* and *write_usb* Routines (Multiplexed Mode)

Summary

The USBN9602 is designed to be easily connected with many different kinds of microcontrollers. This paper has shown specifically how to do it with COP8, MC68HC11, and 80C188EB devices, but the hardware and firmware discussed also provide a more general overview of the essential concepts and constructs. Information presented here can be applied even for other target microcontrollers.

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 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
Tel: 1-800-272-9959
Fax: 1-800-737-7018
Email: support@nsc.com

National Semiconductor Europe
Fax: (+49) 0-180-530 85 86
Email: europe.support@nsc.com
Deutsch Tel: (+49) 0-180-530 85 85
English Tel: (+49) 0-180-532 78 32

National Semiconductor Asia Pacific Customer Response Group
Tel: 65-254-4466
Fax: 65-250-4466
Email: sea.support@nsc.com

National Semiconductor Japan Ltd.
Tel: 81-3-5620-6175
Fax: 81-3-5620-6179