AND8267/D

Controlling the NCP5602 with the I2C Software

Prepared by: Michael Bairanzade
ON Semiconductor

ON Semiconductor®

http://onsemi.com

APPLICATION NOTE

Abstract: The NCP5602 double LED driver includes the capability to remotely control either the normal backlight or the icon function. Moreover, when running the back light mode, the chip has a sixteen steps output current adjustment function to accurately set up the LED brightness. These functions are controlled by a standard I2C protocol as briefly depicted in the NCP5602 data sheet. This Application Note gives a software example to control all the functions embedded into the chip from a basic eight bit micro controller. The assembler code can be ported to other MCU, assuming the basic instructions are available in such MCU.

I2C PROTOCOL: BASIC CONCEPT

The I2C, an acronym for Interface Interchange Chip, has been developed by Philips semiconductors more than 25 years ago. The concept is built on a serial communication, using one clock line to synchronize the data flow, the second line being a bi−directional open drain dedicated to the data content. The two lines are identified as:

– SCL → Serial Clock Line
– SDA → Serial Data Line

The data are based on a byte format with a rate up to 400 kHz for the standard I2C, and up to 3 MHz for the new High Speed protocol. On top of the format, the main advantage of the I2C is the capability to share a common clock and data lines bus with several peripheral devices. This is achieved by using the first byte, when a new transaction takes place, as the address of the physical device one wants to access. In order to make sure no collision occur on the SDA line, the system use only one master at a time, the other peripherals being in the slave mode, ready to read the I2C data bus.

When no data transfer takes place, both lines are at High level with no clock activity. To start a data exchange, the Master forces the SDA low while SCL = High: this is the START pulse and all the peripheral shall be ready to receive the next byte following that bit.

As already mentioned, the first byte carries the physical address of the selected peripheral sharing the same I2C bus. The address is built with the bits[7:1], MSB first, the LSB bit being used to identify the type of communication: see Figure 1. Moreover, in order to avoid data collision, the physical address must be registered by the I2C committee, making sure no device, sharing a common bus, could have identical address.

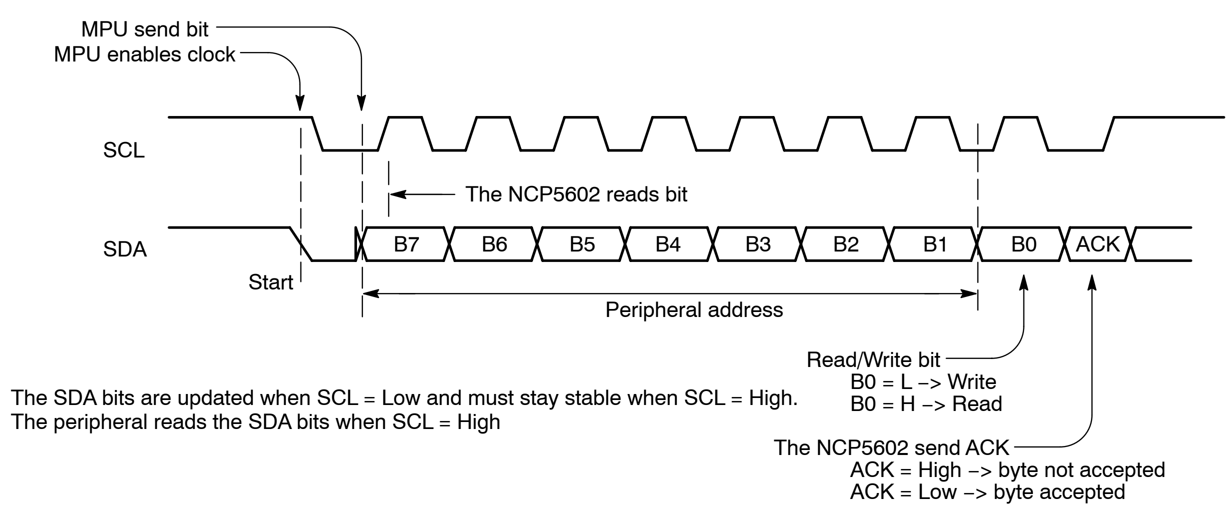

Figure 1. Basic I2C Address Byte Format

© Semiconductor Components Industries, LLC, 2006

All the timings are fully defined by the I2C specifications and any system dealing with the protocol must fulfill these specifications (see I2C document, version 2)

Of course, the concept would be useless without the capability to send one or several data byte following the selected address, but such a feature is a part of the I2C protocol.

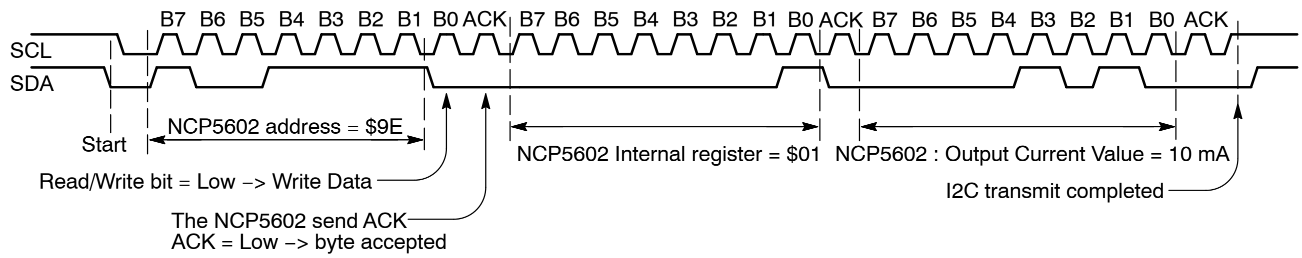

Figure 2. Basic I2C NCP5602 Byte Format Example

To communicate with the NCP5602, the MCU must send three consecutive bytes:

– First byte → physical address = $9E
– Second byte → internal register access = $01
– Third byte → output current value: $00 to $20

The clock and data signals do not need to have an accurate 50% duty cycle, but care must be observed to send the right number of clock pulses to the NCP5602: incorrect count yields lost of synchronization and the chip no longer acknowledges the new programming data. In particular, the software must take into account the 9^{th} bit requested to support the ACK feed back from the NCP5602.

On the other hand, since the SDA line is used to transmit the data in both direction, the I/O pin of the MCU must be configured to either an Output (when sending data on the SDA) or an Input when waiting for the acknowledge.

The basic waveforms given Figure 2 illustrate the digital contain of SCL / SDA to program a 10 mA output current.

The transmission is completed when a STOP bit is send by the MCU: this is achieved by rising SCL to High, keeping SDA = Low, then rising SDA while SCL is at the High level: see Figure 2.

SOFTWARE SECTION

Note: the software developed in this Application Note can be freely re−used to support customer engineering purposes. However, the software is delivered "as is" and ON Semiconductor assumes no responsibility in the event of no function in a final system: see the legal note attached at the end of this document.

The primary purpose of this routine was the support of the white LED drivers developed by ON Semiconductor. A simple but efficient low cost micro controller has been selected in the 8 bits machines family existing in the Freescale portfolio. On the other hand, since the same routine is intended to be use in more complex circuit, it has been decided to use the assembly tools instead of the C or C+ protocol.

The selected MC9S08QG8 micro controller packaged in a QFN16, brings the basic resources necessary to develop the routine:

– Flash memory 8 k
– RAM 512 octets
– I/O pins 12
– Clock embedded oscillator
– RS232 port embedded SCI

Although an I2C block exist in the MS9S08QG8, a specific routine has been created to make possible an easy transfer to another controller.

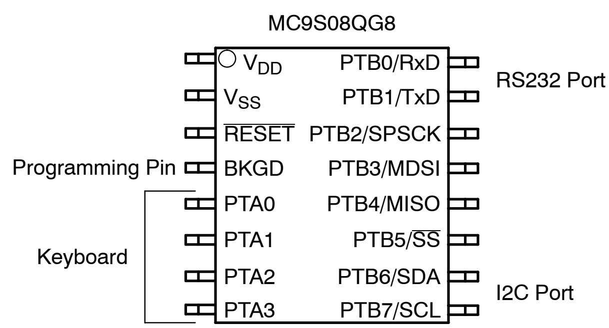

Figure 3. Micro Controller Block Diagram

The interrupt driven keyboard facility, included in the MCU, makes possible an easy test routine implementation to evaluate both the software and the hardware used to support the NCP5602 development. We will use all four bits of the PORTA as depicted in the schematic diagram.

The RS232 port is very useful to communicate with an external PC, making possible, but not mandatory, a remote control of the NCP5602.

The I2C port will use the two pins already identified in the MCU block diagram, although the integrated routines will not be used and replaced by the specific code.

KEYBOARD SUPPORT

The keyboard is connected to the four PORTA bits[3:0], the interrupt function being activated by the KBIPE register. With four external push buttons, the system is capable to control all the functions built in the NCP5602 white LED driver. A digital timer filters out most of the bounces generated when pushing the buttons. Once a key is identified, the associated sub routine is called and the appropriate function takes place. The software clear the keyboard interrupt upon return to the idle state. The basic flowchart is provided Figure 4.

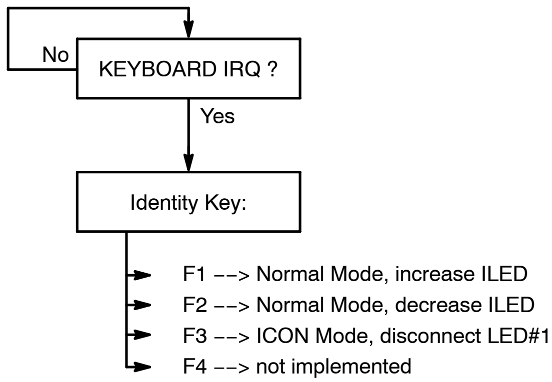

Figure 4. Keyboard Flowchart

The sub routines, associated with each keys, can carry any type of functions requested by the designer.

I2C PROCEDURE

The sub routine is called by the software when either the keyboard has been activated, or if a command is detected on the RS232 port. Depending upon the requested command, the sub routine send the appropriate message to the I2C port. As already depicted, three bytes will be transferred:

- First byte → physical address = $9E (cannot be changed)
- Second byte → internal register access = $01 (cannot be changed)
- Third byte → output current value: $00 to $20

The software evaluates the Acknowledge bit returned by the NCP5602 for each byte and either keeps going the procedure is ACK = Low, or set up an error and exit the sub routine if ACK = High.

The transmission speed is 200 kHz with the basic clock, and the three bytes are downloaded in 135 μs. However, the NCP5602 is not I2C low speed limited and a 400 kHz SCL clock can be used without any risk of data lost during the transmission.

The basic flowchart of the program is provided Figure 5.

The schematic NCP5602 demo board is given Figure 6 and shall be controlled by sending the two I2C signals to the SDA and SCL pins. The software, depicted in this document, is embedded into the hardware given Figure 7. At this point, the end user can either re−use both the hardware and the software, or port the sources into a different MCU controller.

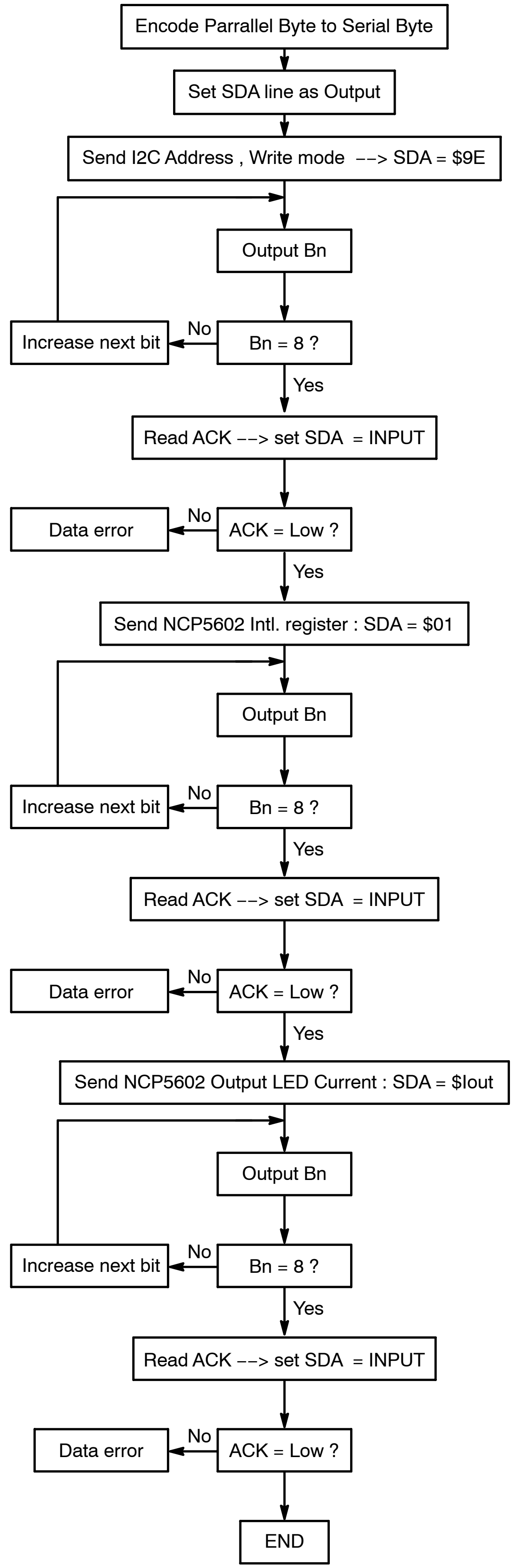

Figure 5. I2C Basic Flowchart

```
************************************************
* EQU TABLE *
************************************************
F1              equ         !0          ;keyboard 1
F2              equ         !1          ;keyboard 2
F3              equ         !2          ;keyboard 3
F4              equ         !3          ;keyboard 4
LED             equ         !4          ;system LED
SDA             equ         !6          ;I2C SDA signal
SCL             equ         !7          ;I2C SCL signal
ICON            equ         !5          ;ICON demo board

**************************************************************
* setup program origin *
**************************************************************
                org         RAMstart
temp            rmb         1
rs232           rmb         1           ;save RS232 content
gSDA            rmb         1           ;contains I2C data
ChipAdr         rmb         1           ;contains slave I2C address
ChipReg         rmb         1           ;contains NCP5602 internal register address
gLED            rmb         1           ;contains the output LED current 1mA/step
gICON           rmb         1           ;control ICON mode

**************************************************************
* Init_KEY – Turns on the Keyboard PORTA *
*                                       *
**************************************************************
Init_KEY:
                bclr        !1,KBISC    ;disable keyboard IRQ
                mov         #$00,KBIES  ;detect falling edge
                lda         #$0F
                sta         PTAPE       ;enable PORTA Pull Up resistors
                mov         #$0F,KBIPE  ;enable PORTA bits[0..3] as keyboard
                mov         #$04,KBISC  ;clear previous flag
                bset        !1,KBISC    ;enable keyboard IRQ
                rts
**************************************************************
* KEYBOARD_ISR – SupportKeyboard Interrupt Service Routine. *
*                                       *
**************************************************************
KeyBoard_isr:
                lda         PTAD        ;get Keybd
                sta         temp        ;store keyboard
                bsr         keyFilter   ;digital filter
                lda         PTAD        ;read PORTA  again
                cmpa        temp        ;check is same contain
                bne         exitKey     ;if not, this is a bounce –> do not proceed
                coma                    ;invert byte
                and         #$0F        ;extract low nibble
                cmpa        #$01        ;check if F1
                bne         testF2
                jsr         KeyF1
                bra         exitKey
                jsr         sendI2C
testF2          cmpa        #$02        ;check if F2
```

```
                bne         testF3
                jsr         KeyF2
                bra         exitKey
testF3          cmpa        #$04            ;check if F3
                bne         testF4
                jsr         KeyF3
                bra         exitKey
testF4          cmpa        #$08            ;check if F4
                bne         exitKey
                jsr         KeyF4
                bra         exitKey
exitKey         bset        !2,KBISC        ;clear previous flag
                rti                         ;return from Interrupt
;*** ON Semiconductor / Toulouse / France    ***
;** Michael Bairanzade                       ***
;** MC9S08QG8 – TSSOP16                      ***
;** File: NCP5602_I2C_CNTL.ASM               ***

;**************************************************************************
;Revision :     Original    = 00 September 2005  ***
;               update      = 1.0 December 2005
;**************************************************************************
   ;send the NCP5602 I2C address = $9E

SendI2C:
                bset        SDA,PTBDD       ;set PORTB Bit6  as Output
                lda         ChipAdr         ;get the I2C byte to send
                jsr         sendStart       ;send the Chip Address
                lda         PORTD
                lsra                        ;extract SDA
                lsra                        ;extract SDA
                lsra                        ;extract SDA
                bcc         sChipReg        ;SDA = Low –> acknowledge OK
                jmp         ACKerror        ;SDA = High –> no acknowledge, error

   ;send the NCP5602 register address = $01

sChipReg        bclr        SDA,PTBD        ;force I/O = L
                bset        SDA,PTBDD       ;set PORTB Bit6  as Output to write
                                            ;to the NCP5602
                bclr        SCL,PTBD        ;set CLOCK = L
                lda         ChipReg         ;send the internal register address
                jsr         sendByte
                lda         PORTD
                lsra        ;extract bit2
                lsra        ;extract bit2
                lsra        ;extract bit2
                bcc         sDATA           ;SDA = Low –> acknowledge OK
                jmp         ACKerror        ;SDA = High –> no acknowledge, error

   ;send the NCP5602 data byte

sDATA           bclr        SDA,PTBD        ;set PORTB Bit6  as Output to write to NCP5608
                bset        SDA,PTBDD
                bclr        SCL,PTBD        ;set CLOCK = L
                lda         gSDA
```

```
            jsr     sendByte
            bclr    SDA,PTBDD       ;set PORTD Bit2  as Input to read
                                    ;the ACKNOWLEDGE
            lda     PORTD
            lsra                    ;extract bit2
            lsra                    ;extract bit2
            lsra                    ;extract bit2
            bcc     exitI2C         ;SDA = Low -> acknowledge OK
            jmp     ACKerror        ;SDA = High -> no acknowledge, error

exitI2C     bclr    SDA,PTBD
            bset    SDA,PTBDD
            bclr    SCL,PTBD
            jsr     delay5
            bset    SCL,PTBD
            jsr     delay5
            bset    SDA,PTBD
            rts                     ;I2C transmission completed
;
;
;
ACKerror    ldhx    #mes2           ;get I2C error message
            jsr     sci_string_out  ;send message to RS232
            rts
;

sendStart   pshx                    ;start the I2C link
            bset    SCL,PTBD        ;preset Clock = H
            ldx     #$08            ;preset I2C clock bit count
            bclr    SDA,PTBD        ;force DATA = L to Start the frame
            nop
nextBit     bclr    SCL,PTBD        ;CLOCK = LOW
            lsla                    ;rotate 8 bits into Carry
            bcs     sendHbit
            bclr    SDA,PTBD        ;SDA = L
            bra     posCLK
sendHbit    bset    SDA,PTBD        ;SDA = H
posCLK      bset    SCL,PTBD        ;CLOCK = High
            decx                    ;decrement clock count
            bne     nextBit
            bclr    SCL,PTBD        ;CLOCK = L
            bclr    SDA,PTBDD       ;set PORTD Bit2  as Input to read
                                    ;the ACKNOWLEDGE
            jsr     delay5          ;wait to make sure the signal
                                    ;is stable
            bset    SCL,PTBD        ;CLOCK = H
            jsr     delay5
            pulx
            rts
;
;
sendByte    pshx                    ;send one byte to the I2C port
            bset    SCL,PTBD        ;preset Clock = H
            ldx     #$07            ;preset I2C clock bit count
            lsla
            nop
```

```
nextBitD    bclr    SCL,PTBD     ;CLOCK = LOW
            lsla                 ;rotate 8 bits into Carry
            bcs     sendHbitD
            bclr    SDA,PTBD     ;SDA = L
            bra     posCLKD
sendHbitD   bset    SDA,PTBD     ;SDA = H
posCLKD     bset    SCL,PTBD     ;CLOCK = High
            decx                 ;decrement clock count
            bne     nextBitD
            bclr    SCL,PTBD     ;CLOCK = L
            bclr    SDA,PTBDD    ;set I/O as Input
            jsr     delay5
            bset    SCL,PTBD     ;CLOCK = H
            jsr     delay5
            pulx
            rts
;
;
mes2                db           "I2C error : no ACK return@"
;
```

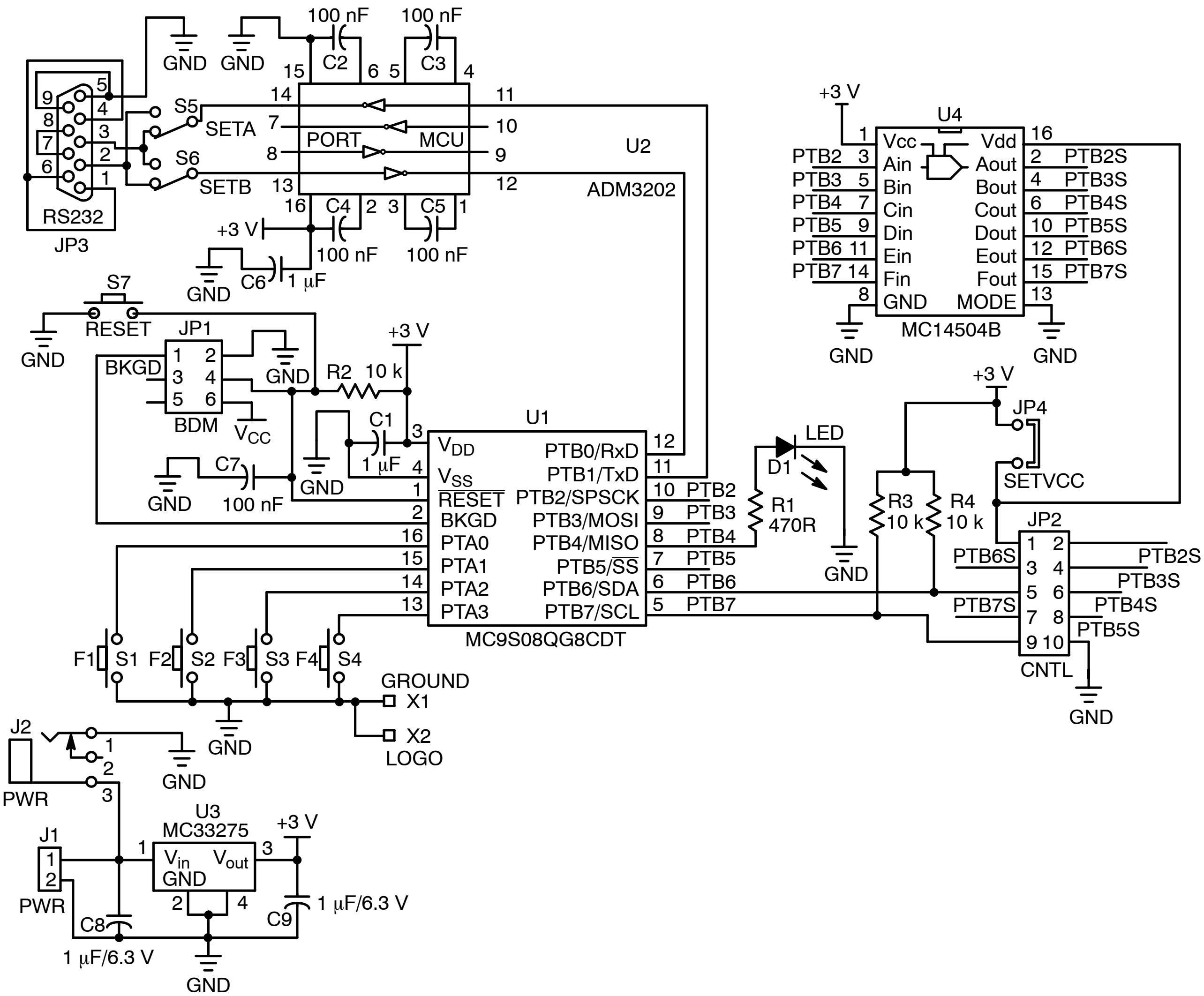

Figure 6. Remote Control Demo Board

The voltage regulator U3 has been implemented to accommodate large input supply voltage, the level shifter –U4– being useful to drive peripherals powered from supply higher than the 3.3 V maximum operating voltage of the MC9S08QG8 micro controller. These two extra chips – U3 & U4 – are not mandatory in a final application and depend solely upon the power supply/MCU voltage range.

The IDC−6 connector is used to program the memory flash of the MC9S08QG8 by means of the P&E hardware. On the other hand, the serial port, built with U2 and JP3, is not necessary to control the NCP5602, but is a powerful tool to monitor the operation during the debug.

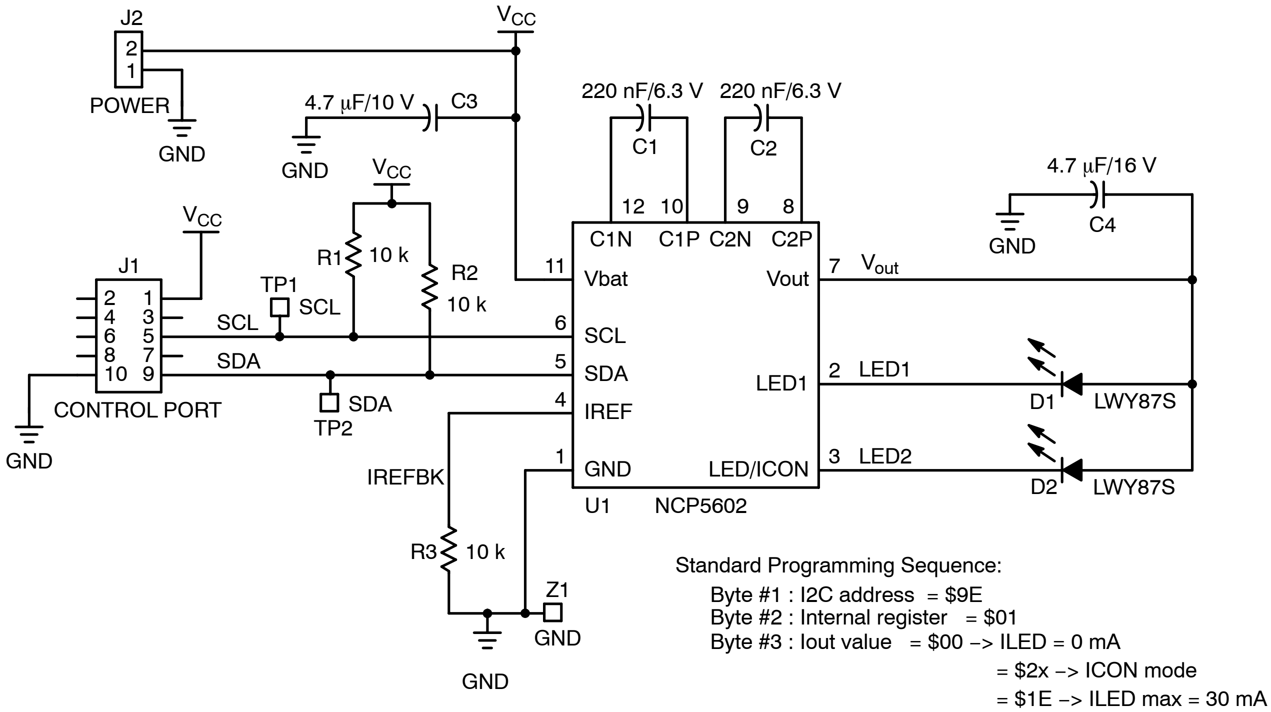

Figure 7. NCP5602 Demo Board Schematic Diagram

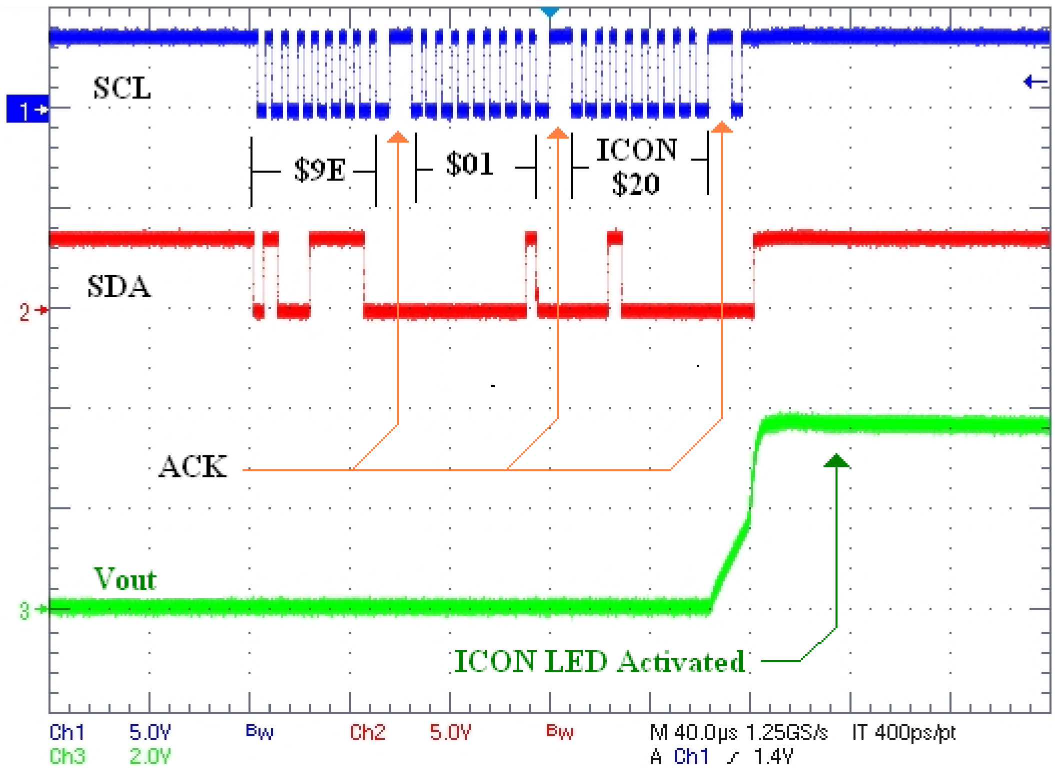

Figure 8. ICON Activation

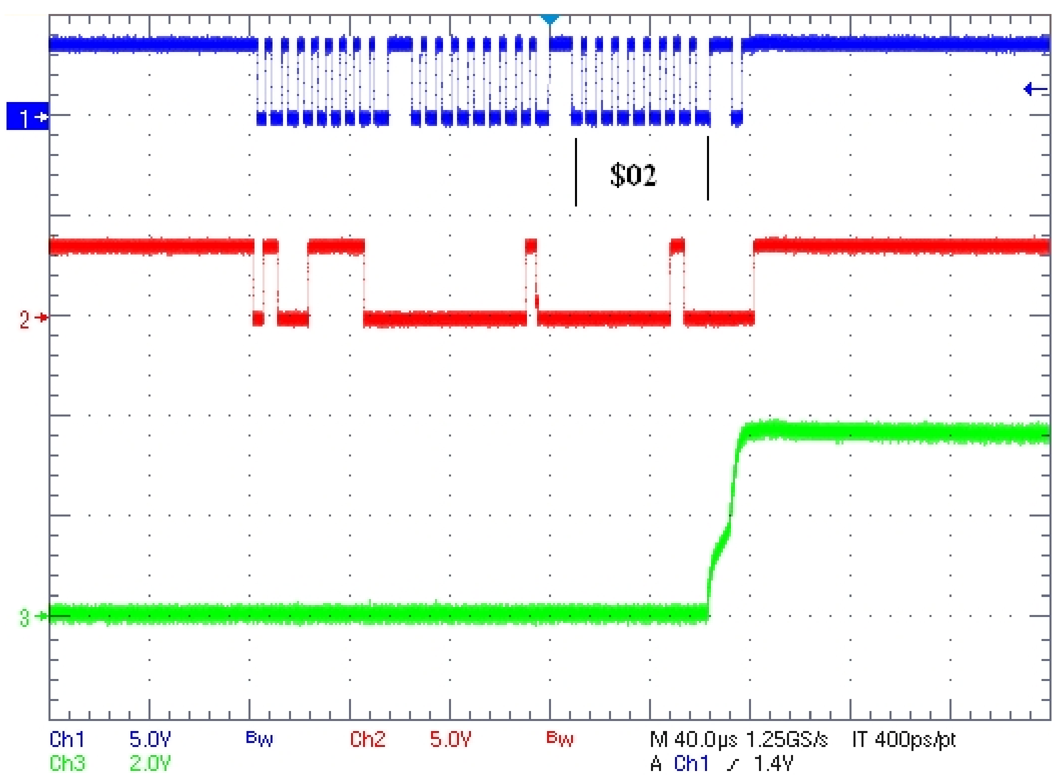

Figure 9. Activate the Back Light : two LED, 2mA per LED

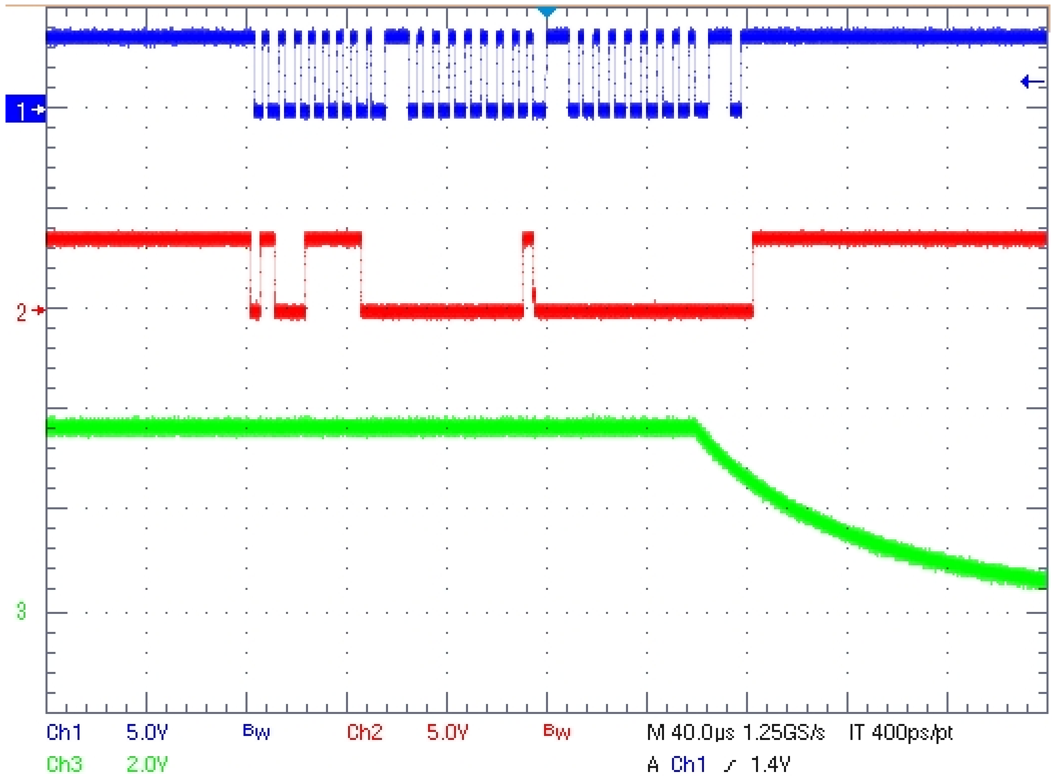

Figure 10. Switch Off Back Light and ICON

ON Semiconductor and ON are registered trademarks of Semiconductor Components Industries, LLC (SCILLC). SCILLC reserves the right to make changes without further notice to any products herein. SCILLC makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does SCILLC assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation special, consequential or incidental damages. “Typical” parameters which may be provided in SCILLC data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals” must be validated for each customer application by customer’s technical experts. SCILLC does not convey any license under its patent rights nor the rights of others. SCILLC products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the SCILLC product could create a situation where personal injury or death may occur. Should Buyer purchase or use SCILLC products for any such unintended or unauthorized application, Buyer shall indemnify and hold SCILLC and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that SCILLC was negligent regarding the design or manufacture of the part. SCILLC is an Equal Opportunity/Affirmative Action Employer. This literature is subject to all applicable copyright laws and is not for resale in any manner.

PUBLICATION ORDERING INFORMATION

LITERATURE FULFILLMENT:
Literature Distribution Center for ON Semiconductor
P.O. Box 5163, Denver, Colorado 80217 USA
Phone: 303−675−2175 or 800−344−3860 Toll Free USA/Canada
Fax: 303−675−2176 or 800−344−3867 Toll Free USA/Canada
Email: orderlit@onsemi.com

N. American Technical Support: 800−282−9855 Toll Free USA/Canada
Europe, Middle East and Africa Technical Support:
Phone: 421 33 790 2910
Japan Customer Focus Center
Phone: 81−3−5773−3850

ON Semiconductor Website: **www.onsemi.com**

Order Literature: http://www.onsemi.com/orderlit

For additional information, please contact your local Sales Representative