SYNACORP TRADING & SERVICES
No.9, 1st Floor, Lorong 1/SS2, Bandar Tasek Mutiara, 14120 Simpang Ampat, S.Prai (S), Penang
Tel : +604.502.1726 Hunting Line : 012.403.3474 Fax : +604.502.1726
(Website) http://www.synacorp.my (Email) sales@synacorp.com.my

TSL2561 Luminosity Sensor
Application Example

TSL2561 is designed particularly for display panels (LCD, OLED, etc.) with the purpose of extending battery life and providing optimum viewing in diverse lighting conditions. Display panel backlighting, which can account for up to 30 to 40 percent of total platform power, can be automatically managed. TSL2561 is also ideal for controlling keyboard illumination to manage exposure control in digital cameras. The TSL2561 is ideal in notebook/tablet PCs, LCD monitors, flat-panel televisions, cell phones, and digital cameras and other applications include street light control, security lighting, sunlight harvesting, machine vision and automotive instrumentation clusters.

APPLICATION INFORMATION: SOFTWARE

Basic Operation

After applying V_{DD}, the device will initially be in the power-down state. To operate the device, issue a command to access the CONTROL register followed by the data value 03h to power up the device. At this point, both ADC channels will begin a conversion at the default integration time of 400 ms. After 400 ms, the conversion results will be available in the DATA0 and DATA1 registers. Use the following pseudo code to read the data registers:

```
// Read ADC Channels Using Read Word Protocol - RECOMMENDED
       Address = 0x39                          //Slave addr – also 0x29 or 0x49

       //Address the Ch0 lower data register and configure for Read Word
       Command = 0xAC                          //Set Command bit and Word bit

       //Reads two bytes from sequential registers 0x0C and 0x0D
       //Results are returned in DataLow and DataHigh variables
       ReadWord (Address, Command, DataLow, DataHigh)
       Channel0 = 256 * DataHigh + DataLow

       //Address the Ch1 lower data register and configure for Read Word
       Command = 0xAE                          //Set bit fields 7 and 5

       //Reads two bytes from sequential registers 0x0E and 0x0F
       //Results are returned in DataLow and DataHigh variables
       ReadWord (Address, Command, DataLow, DataHigh)
       Channel1 = 256 * DataHigh + DataLow     //Shift DataHigh to upper byte

// Read ADC Channels Using Read Byte Protocol
       Address = 0x39                          //Slave addr - also 0x29 or 0x49
       Command = 0x8C                          //Address the Ch0 lower data register
       ReadByte (Address, Command, DataLow)    //Result returned in DataLow
       Command = 0x8D                          //Address the Ch0 upper data register
       ReadByte (Address, Command, DataHigh)   //Result returned in DataHigh
       Channel0 = 256 * DataHigh + DataLow     //Shift DataHigh to upper byte


       Command = 0x8E                          //Address the Ch1 lower data register
       ReadByte (Address, Command, DataLow)    //Result returned in DataLow
       Command = 0x8F                          //Address the Ch1 upper data register
       ReadByte (Address, Command, DataHigh)   //Result returned in DataHigh
```

```
Channel1 = 256 * DataHigh + DataLow     //Shift DataHigh to upper byte
```

Configuring the Timing Register

The command, timing, and control registers are initialized to default values on power up. Setting these registers to the desired values would be part of a normal initialization or setup procedure. In addition, to maximize the performance of the device under various conditions, the integration time and gain may be changed often during operation. The following pseudo code illustrates a procedure for setting up the timing register for various options:

```
// Set up Timing Register
       //Low Gain (1x), integration time of 402ms (default value)
       Address = 0x39
       Command = 0x81
       Data = 0x02
       WriteByte(Address, Command, Data)

       //Low Gain (1x), integration time of 101ms
       Data = 0x01
       WriteByte(Address, Command, Data)

       //Low Gain (1x), integration time of 13.7ms
       Data = 0x00
       WriteByte(Address, Command, Data)

       //High Gain (16x), integration time of 101ms
       Data = 0x11
       WriteByte(Address, Command, Data)

//Read data registers (see Basic Operation example)

//Perform Manual Integration
       //Set up for manual integration with Gain of 1x
       Data = 0x03
       //Set manual integration mode - device stops converting
       WriteByte(Address, Command, Data)

       //Begin integration period
       Data = 0x0B
       WriteByte(Address, Command, Data)

       //Integrate for 50ms
       Sleep (50)                        //Wait for 50ms

       //Stop integrating
       Data = 0x03
       WriteByte(Address, Command, Data)


//Read data registers (see Basic Operation example)
```

Interrupts

The interrupt feature of the TSL256x device simplifies and improves system efficiency by eliminating the need to poll the sensor for a light intensity value. Interrupt styles are determined by the INTR field in the Interrupt Register. The interrupt feature may be disabled by writing a field value of 00h to the Interrupt Control Register so that polling can be performed.

The versatility of the interrupt feature provides many options for interrupt configuration and usage. The primary purpose of the interrupt function is to provide a meaningful change in light intensity. However, it also be used as an end-of-conversion signal. The concept of a *meaningful change* can be defined by the user both in terms of light intensity and time, or persistence, of that change in intensity. The TSL256x device implements two 16-bit-wide interrupt threshold registers that allow the user to define a threshold above and below the current light level. An interrupt will then be generated when the value of a conversion exceeds either of these limits. For simplicity of programming, the threshold comparison is accomplished only with Channel 0. This simplifies calculation of thresholds that are based, for example, on a percent of the current light level. It is adequate to use only one channel when calculating light intensity differences since, for a given light source, the channel 0 and channel 1 values are linearly proportional to each other and thus both values scale linearly with light intensity.

To further control when an interrupt occurs, the TSL256x device provides an interrupt persistence feature. This feature allows the user to specify a number of conversion cycles for which a light intensity exceeding either interrupt threshold must persist before actually generating an interrupt. This can be used to prevent transient changes in light intensity from generating an unwanted interrupt. With a value of 1, an interrupt occurs immediately whenever either threshold is exceeded. With values of *N*, where *N* can range from 2 to 15, *N* consecutive conversions must result in values outside the interrupt window for an interrupt to be generated. For example, if *N* is equal to 10 and the integration time is 402 ms, then an interrupt will not be generated unless the light level persists for more than 4 seconds outside the threshold.

Two different interrupt styles are available: Level and SMBus Alert. The difference between these two interrupts styles is how they are cleared. Both result in the interrupt line going active low and remaining low until the interrupt is cleared. A level style interrupt is cleared by setting the CLEAR bit (bit 6) in the COMMAND register. The SMBus Alert style interrupt is cleared by an Alert Response as described in the Interrupt Control Register section and SMBus specification.

To configure the interrupt as an end-of-conversion signal, the interrupt PERSIST field is set to 0. Either Level or SMBus Alert style can be used. An interrupt will be generated upon completion of each conversion. The interrupt threshold registers are ignored. The following example illustrates the configuration of a level interrupt:

```
// Set up end-of-conversion interrupt, Level style
      Address = 0x39                          //Slave addr also 0x29 or 0x49
      Command = 0x86                          //Address Interrupt Register
      Data = 0x10                             //Level style, every ADC cycle
      WriteByte(Address, Command, Data)
```

The following example pseudo code illustrates the configuration of an SMB Alert style interrupt when the light intensity changes 20% from the current value, and persists for 3 conversion cycles:

```
// Read current light level
      Address = 0x39                           //Slave addr also 0x29 or 0x49
      Command = 0xAC                           //Set Command bit and Word bit
      ReadWord (Address, Command, DataLow, DataHigh)
      Channel0 = (256 * DataHigh) + DataLow
```

```
//Calculate upper and lower thresholds
T_Upper = Channel0 + (0.2 * Channel0)
T_Lower = Channel0 – (0.2 * Channel0)

//Write the lower threshold register
Command = 0xA2                          //Addr lower threshold reg, set Wor Bit
WriteWord (Address, Command, T_Lower.LoByte, T_Lower.HiByte)

//Write the upper threshold register
Command = 0xA4                          //Addr upper threshold reg,set Word bit
WriteWord (Address, Command, T_Upper.LoByte, T_Upper.HiByte)

//Enable interrupt
Command = 0x86                          //Address interrupt register
Data = 0x23                             //SMBAlert style, PERSIST = 3
WriteByte(Address, Command, Data)
```

In order to generate an interrupt on demand during system test or debug, a test mode (INTR = 11) can be used. The following example illustrates how to generate an interrupt on demand:

```
// Generate an interrupt
       Address = 0x39                          //Slave addr also 0x29 or 0x49
       Command = 0x86                          //Address Interrupt register
       Data = 0x30                             //Test interrupt
       WriteByte(Address, Command, Data)


       //Interrupt line should now be low
```

Calculating Lux

The TSL256x is intended for use in ambient light detection applications such as display backlight control, where adjustments are made to display brightness or contrast based on the brightness of the ambient light, as perceived by the human eye. Conventional silicon detectors respond strongly to infrared light, which the human eye does not see. This can lead to significant error when the infrared content of the ambient light is high, such as with incandescent lighting, due to the difference between the silicon detector response and the brightness perceived by the human eye.

This problem is overcome in the TSL256x through the use of two photodiodes. One of the photodiodes (channel 0) is sensitive to both visible and infrared light, while the second photodiode (channel 1) is sensitive primarily to infrared light. An integrating ADC converts the photodiode currents to digital outputs. Channel 1 digital output is used to compensate for the effect of the infrared component of light on the channel 0 digital outputs. The ADC digital outputs from the two channels are used in a formula to obtain a value that approximates the human eye response in the commonly used Illuminance unit of Lux:

CS Package

For 0 < CH1/CH0 _ 0.52	Lux = 0.0315 _ CH0 − 0.0593 _ CH0 _ ((CH1/CH0)1.4)
For 0.52 < CH1/CH0 _ 0.65	Lux = 0.0229 _ CH0 − 0.0291 _ CH1
For 0.65 < CH1/CH0 _ 0.80	Lux = 0.0157 _ CH0 − 0.0180 _ CH1
For 0.80 < CH1/CH0 _ 1.30	Lux = 0.00338 _ CH0 − 0.00260 _ CH1
For CH1/CH0 > 1.30	Lux = 0

T, FN, and CL Package

For 0 < CH1/CH0 _ 0.50	Lux = 0.0304 _ CH0 − 0.062 _ CH0 _ ((CH1/CH0)1.4)

For 0.50 < CH1/CH0 _ 0.61	Lux = 0.0224 _ CH0 − 0.031 _ CH1
For 0.61 < CH1/CH0 _ 0.80	Lux = 0.0128 _ CH0 − 0.0153 _ CH1
For 0.80 < CH1/CH0 _ 1.30	Lux = 0.00146 _ CH0 − 0.00112 _ CH1
For CH1/CH0 > 1.30	Lux = 0

The formulas shown above were obtained by optical testing with fluorescent and incandescent light sources, and apply only to open-air applications. Optical apertures (e.g. light pipes) will affect the incident light on the device.

Simplified Lux Calculation

Below is the argument and return value including source code (shown on following page) for calculating lux. The source code is intended for embedded and/or microcontroller applications. Two individual code sets are provided, one for the T, FN, and CL packages, and one for the CS package. All floating point arithmetic operations have been eliminated since embedded controllers and microcontrollers generally do not support these types of operations. Since floating point has been removed, scaling must be performed prior to calculating illuminance if the integration time is not 402 ms and/or if the gain is not 16_ as denoted in the source code on the following pages. This sequence scales first to mitigate rounding errors induced by decimal math.

```
extern unsigned int CalculateLux(unsigned int iGain, unsigned int tInt, unsigned int
ch0, unsigned int ch1, int iType)


//****************************************************************************
//
// Copyright _ 2004-2005 TAOS, Inc.
//
// THIS CODE AND INFORMATION IS PROVIDED ”AS IS” WITHOUT WARRANTY OF ANY
// KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
// IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR
// PURPOSE.
//
//     Module Name:
//             lux.cpp
//
//****************************************************************************

#define LUX_SCALE 14       // scale by 2^14
#define RATIO_SCALE 9      // scale ratio by 2^9

//−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
// Integration time scaling factors
//−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

#define CH_SCALE           10      // scale channel values by 2^10
#define CHSCALE_TINT0      0x7517 // 322/11 * 2^CH_SCALE
#define CHSCALE_TINT1      0x0fe7 // 322/81 * 2^CH_SCALE

//−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
// T, FN, and CL Package coefficients
//−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
// For Ch1/Ch0=0.00 to 0.50
//            Lux/Ch0=0.0304−0.062*((Ch1/Ch0)^1.4)
//            piecewise approximation
//                   For Ch1/Ch0=0.00 to 0.125:
//                          Lux/Ch0=0.0304−0.0272*(Ch1/Ch0)
//
//                   For Ch1/Ch0=0.125 to 0.250:
//                          Lux/Ch0=0.0325−0.0440*(Ch1/Ch0)
```

```
//
//                    For Ch1/Ch0=0.250 to 0.375:
//                          Lux/Ch0=0.0351-0.0544*(Ch1/Ch0)
//
//                    For Ch1/Ch0=0.375 to 0.50:
//                          Lux/Ch0=0.0381-0.0624*(Ch1/Ch0)
//
// For Ch1/Ch0=0.50 to 0.61:
//             Lux/Ch0=0.0224-0.031*(Ch1/Ch0)
//
// For Ch1/Ch0=0.61 to 0.80:
//             Lux/Ch0=0.0128-0.0153*(Ch1/Ch0)
//
// For Ch1/Ch0=0.80 to 1.30:
//             Lux/Ch0=0.00146-0.00112*(Ch1/Ch0)
//
// For Ch1/Ch0>1.3:
//              Lux/Ch0=0
//----------------------------------------------------
#define K1T 0x0040         // 0.125 * 2^RATIO_SCALE
#define B1T 0x01f2         // 0.0304 * 2^LUX_SCALE
#define M1T 0x01be         // 0.0272 * 2^LUX_SCALE


#define K2T 0x0080         // 0.250 * 2^RATIO_SCALE
#define B2T 0x0214         // 0.0325 * 2^LUX_SCALE
#define M2T 0x02d1         // 0.0440 * 2^LUX_SCALE

#define K3T 0x00c0         // 0.375 * 2^RATIO_SCALE
#define B3T 0x023f         // 0.0351 * 2^LUX_SCALE
#define M3T 0x037b         // 0.0544 * 2^LUX_SCALE

#define K4T 0x0100         // 0.50 * 2^RATIO_SCALE
#define B4T 0x0270         // 0.0381 * 2^LUX_SCALE
#define M4T 0x03fe         // 0.0624 * 2^LUX_SCALE
#define K5T 0x0138         // 0.61 * 2^RATIO_SCALE
#define B5T 0x016f         // 0.0224 * 2^LUX_SCALE
#define M5T 0x01fc         // 0.0310 * 2^LUX_SCALE

#define K6T 0x019a         // 0.80 * 2^RATIO_SCALE
#define B6T 0x00d2         // 0.0128 * 2^LUX_SCALE
#define M6T 0x00fb         // 0.0153 * 2^LUX_SCALE

#define K7T 0x029a         // 1.3 * 2^RATIO_SCALE
#define B7T 0x0018         // 0.00146 * 2^LUX_SCALE
#define M7T 0x0012         // 0.00112 * 2^LUX_SCALE

#define K8T 0x029a         // 1.3 * 2^RATIO_SCALE
#define B8T 0x0000         // 0.000 * 2^LUX_SCALE
#define M8T 0x0000         // 0.000 * 2^LUX_SCALE

//----------------------------------------------------
// CS package coefficients
//----------------------------------------------------
// For 0 <= Ch1/Ch0 <= 0.52
//             Lux/Ch0 = 0.0315-0.0593*((Ch1/Ch0)^1.4)
//             piecewise approximation
//                    For 0 <= Ch1/Ch0 <= 0.13
//                          Lux/Ch0 = 0.0315-0.0262*(Ch1/Ch0)
//                    For 0.13 <= Ch1/Ch0 <= 0.26
//                          Lux/Ch0 = 0.0337-0.0430*(Ch1/Ch0)
//                    For 0.26 <= Ch1/Ch0 <= 0.39
//                          Lux/Ch0 = 0.0363-0.0529*(Ch1/Ch0)
//                    For 0.39 <= Ch1/Ch0 <= 0.52
//                          Lux/Ch0 = 0.0392-0.0605*(Ch1/Ch0)
// For 0.52 < Ch1/Ch0 <= 0.65
//      Lux/Ch0 = 0.0229-0.0291*(Ch1/Ch0)
// For 0.65 < Ch1/Ch0 <= 0.80
```

```
//      Lux/Ch0 = 0.00157-0.00180*(Ch1/Ch0)
// For 0.80 < Ch1/Ch0 <= 1.30
//     Lux/Ch0 = 0.00338-0.00260*(Ch1/Ch0)
// For Ch1/Ch0 > 1.30
//     Lux = 0
//----------------------------------------------------
#define K1C 0x0043        // 0.130 * 2^RATIO_SCALE
#define B1C 0x0204        // 0.0315 * 2^LUX_SCALE
#define M1C 0x01ad        // 0.0262 * 2^LUX_SCALE

#define K2C 0x0085        // 0.260 * 2^RATIO_SCALE
#define B2C 0x0228        // 0.0337 * 2^LUX_SCALE
#define M2C 0x02c1        // 0.0430 * 2^LUX_SCALE

#define K3C 0x00c8        // 0.390 * 2^RATIO_SCALE
#define B3C 0x0253        // 0.0363 * 2^LUX_SCALE
#define M3C 0x0363        // 0.0529 * 2^LUX_SCALE

#define K4C 0x010a        // 0.520 * 2^RATIO_SCALE
#define B4C 0x0282        // 0.0392 * 2^LUX_SCALE
#define M4C 0x03df        // 0.0605 * 2^LUX_SCALE

#define K5C 0x014d        // 0.65 * 2^RATIO_SCALE
#define B5C 0x0177        // 0.0229 * 2^LUX_SCALE
#define M5C 0x01dd        // 0.0291 * 2^LUX_SCALE

#define K6C 0x019a        // 0.80 * 2^RATIO_SCALE
#define B6C 0x0101        // 0.0157 * 2^LUX_SCALE
#define M6C 0x0127        // 0.0180 * 2^LUX_SCALE

#define K7C 0x029a        // 1.3 * 2^RATIO_SCALE
#define B7C 0x0037        // 0.00338 * 2^LUX_SCALE
#define M7C 0x002b        // 0.00260 * 2^LUX_SCALE
#define K8C 0x029a        // 1.3 * 2^RATIO_SCALE
#define B8C 0x0000        // 0.000 * 2^LUX_SCALE
#define M8C 0x0000        // 0.000 * 2^LUX_SCALE

// lux equation approximation without floating point calculations
//////////////////////////////////////////////////////////////////////////////
// Routine: unsigned int CalculateLux(unsigned int ch0, unsigned int ch0, int iType)
//
// Description: Calculate the approximate illuminance (lux) given the raw
//              channel values of the TSL2560. The equation if implemented
//              as a piece-wise linear approximation.
//
// Arguments: unsigned int iGain - gain, where 0:1X, 1:16X
//            unsigned int tInt - integration time, where 0:13.7mS, 1:100mS, 2:402mS,
//             3:Manual
//            unsigned int ch0 - raw channel value from channel 0 of TSL2560
//            unsigned int ch1 - raw channel value from channel 1 of TSL2560
//            unsigned int iType - package type (T or CS)
//
// Return: unsigned int - the approximate illuminance (lux)
//
//////////////////////////////////////////////////////////////////////////////
unsigned int CalculateLux(unsigned int iGain, unsigned int tInt, unsigned int ch0,
      unsigned int ch1, int iType)
{

      //------------------------------------------------------------------------
      // first, scale the channel values depending on the gain and integration time
      // 16X, 402mS is nominal.
      // scale if integration time is NOT 402 msec
      unsigned long chScale;
      unsigned long channel1;
      unsigned long channel0;
      switch (tInt)
      {
```

```
        case 0:      // 13.7 msec
               chScale = CHSCALE_TINT0;
               break;
        case 1: // 101 msec
               chScale = CHSCALE_TINT1;
               break;
        default: // assume no scaling
               chScale = (1 << CH_SCALE);
               break;
}

// scale if gain is NOT 16X
if (!iGain) chScale = chScale << 4;     // scale 1X to 16X

// scale the channel values
channel0 = (ch0 * chScale) >> CH_SCALE;
channel1 = (ch1 * chScale) >> CH_SCALE;
//-------------------------------------------------------------------------

// find the ratio of the channel values (Channel1/Channel0)
// protect against divide by zero
unsigned long ratio1 = 0;
if (channel0 != 0) ratio1 = (channel1 << (RATIO_SCALE+1)) / channel0;

// round the ratio value
unsigned long ratio = (ratio1 + 1) >> 1;

// is ratio <= eachBreak ?
unsigned int b, m;
switch (iType)
{
        case 0: // T, FN and CL package
               if ((ratio >= 0) && (ratio <= K1T))
                      {b=B1T; m=M1T;}
               else if (ratio <= K2T)
                      {b=B2T; m=M2T;}
               else if (ratio <= K3T)
                      {b=B3T; m=M3T;}
               else if (ratio <= K4T)
                      {b=B4T; m=M4T;}
               else if (ratio <= K5T)
                      {b=B5T; m=M5T;}
               else if (ratio <= K6T)
                      {b=B6T; m=M6T;}
               else if (ratio <= K7T)
                      {b=B7T; m=M7T;}
               else if (ratio > K8T)
                      {b=B8T; m=M8T;}
               break;

        case 1:// CS package
               if ((ratio >= 0) && (ratio <= K1C))
                      {b=B1C; m=M1C;}
               else if (ratio <= K2C)
                      {b=B2C; m=M2C;}
               else if (ratio <= K3C)
                      {b=B3C; m=M3C;}
               else if (ratio <= K4C)
                      {b=B4C; m=M4C;}
               else if (ratio <= K5C)
                      {b=B5C; m=M5C;}
               else if (ratio <= K6C)
                      {b=B6C; m=M6C;}
               else if (ratio <= K7C)
                      {b=B7C; m=M7C;}
               else if (ratio > K8C)
                      {b=B8C; m=M8C;}
               break;
}
```

```
    unsigned long temp;
    temp = ((channel0 * b) - (channel1 * m));

    // do not allow negative lux value
    if (temp < 0) temp = 0;

    // round lsb (2^(LUX_SCALE-1))
    temp += (1 << (LUX_SCALE-1));

    // strip off fractional portion
    unsigned long lux = temp >> LUX_SCALE;

    return(lux);

}
```

APPLICATION INFORMATION: HARDWARE

Power Supply Decoupling and Application Hardware Circuit

The power supply lines must be decoupled with a 0.1 μF capacitor placed as close to the device package as possible (Figure 18). The bypass capacitor should have low effective series resistance (ESR) and low effective series inductance (ESI), such as the common ceramic types, which provide a low impedance path to ground at high frequencies to handle transient currents caused by internal logic switching.

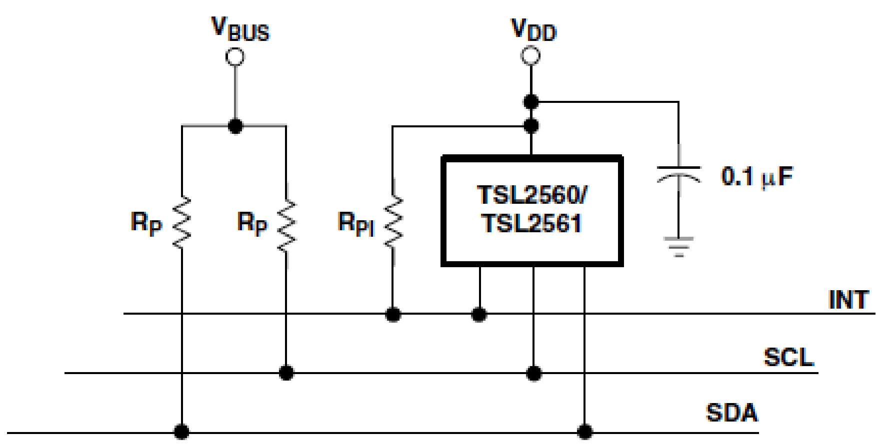

Figure 18. Bus Pull-Up Resistors

Pull-up resistors (Rp) maintain the SDAH and SCLH lines at a *high* level when the bus is free and ensure the signals are pulled up from a low to a high level within the required rise time. For a complete description of the SMBus maximum and minimum Rp values, please review the SMBus Specification at http://www.smbus.org/specs. For a complete description of I2C maximum and minimum Rp values, please review the I2C Specification at http://www.semiconductors.philips.com.

Pull-up resistors (RPI) is also required for the interrupt (INT), which functions as a wired-AND signal in a similar fashion to the SCL and SDA lines. A typical impedance value between 10 kΩ and 100 kΩ can be used. Please note that while Figure 18 shows INT being pulled up to VDD, the interrupt can optionally be pulled up to VBUS.