DESIGNER'S NOTEBOOK

TAOS TEXAS ADVANCED OPTOELECTRONIC SOLUTIONS

TSL2771 I²C Communications and Proximity Algorithm

by Joe Smith, Todd Bishop, Kerry Glover and Florencio Villasenor

August 2011

Abstract

TAOS Light-to-Digital sensors can perform object detection and gesture control by using the proximity (with or without integrated Ambient light detection) flexible I2C configurable sensors. The following designer notebook discusses proximity configuration variables with a TAOS TSL2771X digital light sensor family.

Introduction

TAOS provides a product line of light-to-digital products that interface to a microprocessor utilizing an I²C interface. This document describes the interface protocol and provides several examples of how to initialize a part and read data based on that protocol. For this design note, the TSL2771 will be utilized however the concepts apply to many of the other devices as well.

When power is first applied to TAOS digital light sensors, it will power up to a default configuration. To start taking a light reading for proximity, the device must be communicated to using I2C commands. This initialization is required whenever power is removed from the device, or the sensor settings need to be modified due to changing object detection targets or other dynamic system requirements

I²C Protocol

Interface and control of the device is through an I²C serial interface, using standard (<100Kbits/s) or fast (400Kbits/s) mode, to a set of registers that provide access to device control functions and retrieve output data. The device supports a slave address of either 0x29 or 0x39 hex using 7 bit addressing protocol. For actual I²C programming, the address should be shifted to the left one bit and the R/W bit inserted at the Least Significant Bit (LSB) position; zero for write, one for read. For example, for an address of 0x39, the proper address byte for a write would be 0x72 and for a read would be 0x73.

Slave Address = (0x39 << 1) + r/w = 0111 001X = 0x72 for Write, 0x73 for Read

The I²C standard provides for three types of bus transactions: write, read and a combined protocol. During a write operation, the master issues a START operation followed by the first byte which is the Slave Address with the LSB being a zero for write. The slave acknowledges by setting the ACK bit. The master then sends a command byte followed by one or more data bytes. The slave acknowledges receipt of each byte by setting the ACK bit. When the master is done sending data, it issues a STOP operation.

During a read operation, the master issues a START operation, followed by the Slave Address, with the LSB a one for read. The slave acknowledges by setting the ACK bit. The slave then sends data bytes, which the master acknowledges. When the master receives that last byte that it needs, it set a NACK and issues a STOP condition. In the combined protocol, a write operation is performed, without

a STOP, immediately followed by a restart and a read command. For a complete description of I²C protocols, please review the I²C Specification at: http://www.semiconductors.philips.com

The write protocol is to write to the slave device with a command byte followed by a single data byte.

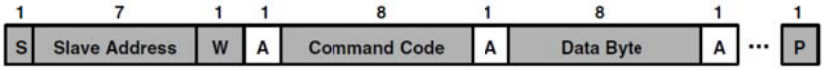


Figure 1: I²C Write Protocol

The read protocol can use separate write then read or the combined format to write the command code and immediately read the data from the device. Data from the internal ADC operation are typically 16 bit values accessed with a two byte read operation.

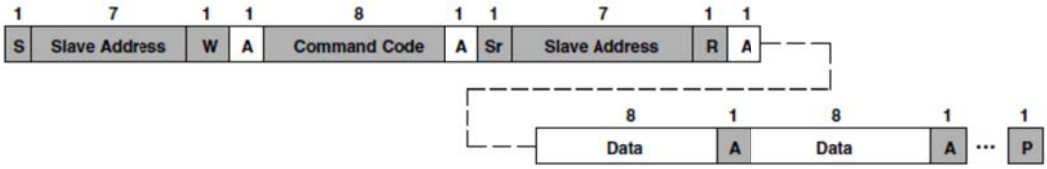


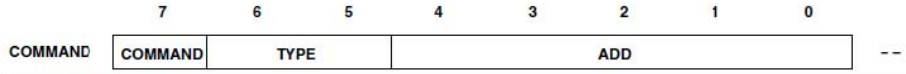
Figure 2: I²C Read Protocol – Combined Format

Command Code

Accessing data in the TAOS sensors requires first setting an address and then accessing the data. Therefore, there is almost always a write followed by a read. In rare cases, the write operation will not be utilized, in which case the read will utilize the address from the last write operation. Such cases would include polling to know when a particular status bit was set.

The first byte written to the device after the slave address will be a command byte. If the operation is to write to a data register, then multiple bytes can be written with the second byte going to the data register. For read commands, there is typically a write operation following by a read from the appropriate register. The combined protocol is ideal for this type of operation.

To specify that the first byte is a command byte, the MSB is set. The lower 7 bits are used to either specify the 5 bit address to be accessed or to specify a special function such as an interrupt reset. The following is a description of the command register of the TSL2771.



FIELD	BITS	DESCRIPTION
COMMAND	7	Select Command Register. Must write as 1 when addressing COMMAND register.
TYPE	6:5	Selects type of transaction to follow in subsequent data transfers:
		FIELD VALUE: INTEGRATION TIME
		00: Repeated byte protocol transaction
		01: Auto-increment protocol transaction
		10: Reserved — Do not use
		11: Special function — See description below
		Transaction type 00 will repeatedly read the same register with each data access. Transaction type 01 will provide an auto-increment function to read successive register bytes.
ADD	4:0	Address register/special function register. Depending on the transaction type, see above, this field either specifies a special function command or selects the specific control-status-register for following write and read transactions:
		FIELD VALUE: READ VALUE
		00000: Normal — no action
		00101: Proximity interrupt clear
		00110: ALS interrupt clear
		00111: Proximity and ALS interrupt clear
		other: Reserved — Do not write
		ALS/Proximity Interrupt Clear. Clears any pending ALS/Proximity interrupt. This special function is self clearing.

Figure 3: Command Register

There are two protocols that are used to access registers. The repeated byte protocol is used to access a single byte only. Multiple bytes read at that address will read the same register multiple times. The repeated byte protocol is only used in reading a status register where the same register needs to be read repeatedly. The auto-increment protocol increments the address after each byte is read. This allows for reading of 1, 2 or up to the full set of 32 registers. This is the typical protocol that is used. In the auto-increment protocol, the value of 0xA0 added to the 5 bit register address.

TSL2771 Register Set

The following is a summary of the register set for the TSL2771. The first sixteen registers are read/write control register which are initialized for a particular application. The next two register are the ID and Status registers. The last 6 register are read only registers for data from the device.

ADDRESS	RESISTER NAME	R/W	REGISTER FUNCTION	RESET VALUE
---	COMMAND	W	Specifies register address	0x00
0x00	ENABLE	R/W	Enables states and interrupts	0x00
0x01	ATIME	R/W	ALS ADC time	0xFF
0x02	PTIME	R/W	Proximity ADC time	0xFF
0x03	WTIME	R/W	Wait time	0xFF
0x04	AILTL	R/W	ALS interrupt low threshold low byte	0x00
0x05	AILTH	R/W	ALS interrupt low threshold high byte	0x00
0x06	AIHTL	R/W	ALS interrupt high threshold low byte	0x00
0x07	AIHTH	R/W	ALS interrupt high threshold high byte	0x00
0x08	PILTL	R/W	Proximity interrupt low threshold low byte	0x00
0x09	PILTH	R/W	Proximity interrupt low threshold high byte	0x00
0x0A	PIHTL	R/W	Proximity interrupt high threshold low byte	0x00
0x0B	PIHTH	R/W	Proximity interrupt high threshold high byte	0x00
0x0C	PERS	R/W	Interrupt persistence filters	0x00
0x0D	CONFIG	R/W	Configuration	0x00
0x0E	PPCOUNT	R/W	Proximity pulse count	0x00
0x0F	CONTROL	R/W	Gain control register	0x00
0x12	ID	R	Device ID	ID
0x13	STATUS	R	Device status	0x00
0x14	CDATA	R	Clear ADC low data register	0x00
0x15	CDATAH	R	Clear ADC high data register	0x00
0x16	IRDATA	R	IR ADC low data register	0x00
0x17	IRDATAH	R	IR ADC high data register	0x00
0x18	PDATA	R	Proximity ADC low data register	0x00
0x19	PDATAH	R	Proximity ADC high data register	0x00

Figure 4: Register Address

	7	6	5	4	3	2	1	0	
ENABLE	Reserved		PIEN	AIEN	WEN	PEN	AEN	PON	Address 0x00

FIELD	BITS	DESCRIPTION
Reserved	7:6	Reserved. Write as 0.
PIEN	5	Proximity interrupt mask. When asserted, permits proximity interrupts to be generated.
AIEN	4	ALS interrupt mask. When asserted, permits ALS interrupts to be generated.
WEN	3	Wait Enable. This bit activates the wait feature. Writing a 1 activates the wait timer. Writing a 0 disables the wait timer.
PEN	2	Proximity enable. This bit activates the proximity function. Writing a 1 enables proximity. Writing a 0 disables proximity.
AEN	1	ALS Enable. This bit actives the two channel ADC. Writing a 1 activates the ALS. Writing a 0 disables the ALS.
PON [1, 2]	0	Power ON. This bit activates the internal oscillator to permit the timers and ADC channels to operate. Writing a 1 activates the oscillator. Writing a 0 disables the oscillator.

NOTES: 1. See Power Management section for more information.
2. A minimum interval of 2.72 ms must pass after PON is asserted before either a proximity or ALS can be initiated. This required time is enforced by the hardware in cases where the firmware does not provide it.

Figure 5: Enable Register

Proximity Algorithm

TAOS Proximity sensors use an external light source (generally an infrared LED emitter) to emit light, which is then viewed by the integrated light detector to measure the amount of reflected light when an object is in the light path. The amount of light detected from a reflected surface can then be used to determine an object's proximity to the sensor or motion.

There are many parameters in an application system other than the sensor that effect proximity detection. Some of these are the LED power output and beam angle, the surface texture and color to be detected, the mechanical placement of the sensor and emitter, and any other objects in the light path that reduce, scatter, or change the direction of the light.

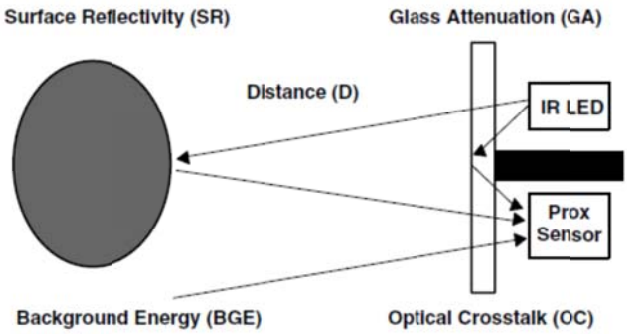


Figure 6: Proximity Detection

The TSL2771X has several proximity variables to allow for very good flexibility for different target distance detection applications.

The proximity register controlled variables are:

(PDRIVE) LED drive current (12,25,50,100mA). Choose a higher LED drive current for longer distance detection. The LED drive current is controlled by a regulated current sink on the LDR pin. This feature eliminates the need to use a current limiting resistor to control LED current. The LED drive current can be configured for 12.5 mA,25 mA, 50 mA, or 100 mA. For higher LED drive requirements, an external P type transistor can be used to control the LED current.

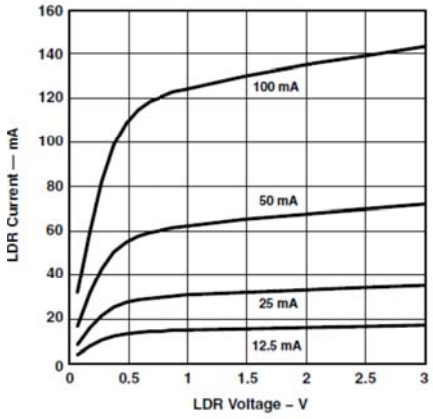


Figure 7: Typical LDR Current vs Voltage

(PPCOUNT) LED light output pulses. Choose 1,2,4,8 pulses setting generally. The more pulses, the greater the distance, but also there is a larger offset (larger starting number).

The number of LED pulses can be programmed to any value between 1 and 255 pulses as needed. Increasing the number of LED pulses at a given current will increase the sensor sensitivity. Sensitivity grows by the square root of the number of pulses. Each pulse has a 16-µs period.

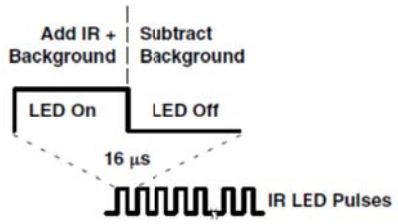


Figure 10: Proximity IR LED Waveform

(PDIODE) The light sensor diode channel used to detect Proximity.

(Ch0 vis+IR), Ch1 (mainly IR), or CH0+1.

Choose CH1 for best visible light rejection.

Choose CH0 or CH0+1 for greater distance.

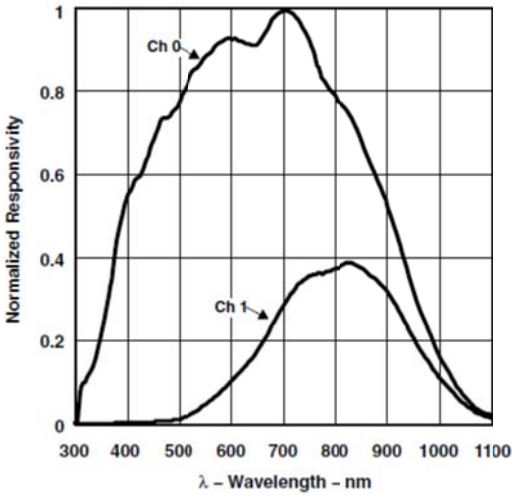


Figure 11: Spectral Responsivity

(PTIME) Proximity integration time is used to set the Full Scale number in the proximity register.

Default 2.7ms is 10 bit = 1023 maximum full scale count.

This can be increased to produce a larger number, but not recommended. The proximity integration time (PTIME) is the period of time that the internal ADC converts the analog signal to a digital count. It is recommend that this be set to a minimum of PTIME = 0xFF or 2.72 ms.

(PDATA) Proximity Data output 16 bit register. At the end of the integration cycle, the results are latched into the proximity data (PDATA) register.

The combination of LED power and number of pulses can be used to control the distance at which the sensor can detect proximity. An example of the distances covered with settings such that each curve covers 2× the distance. Counts up to 64 pulses provide a 16× range.

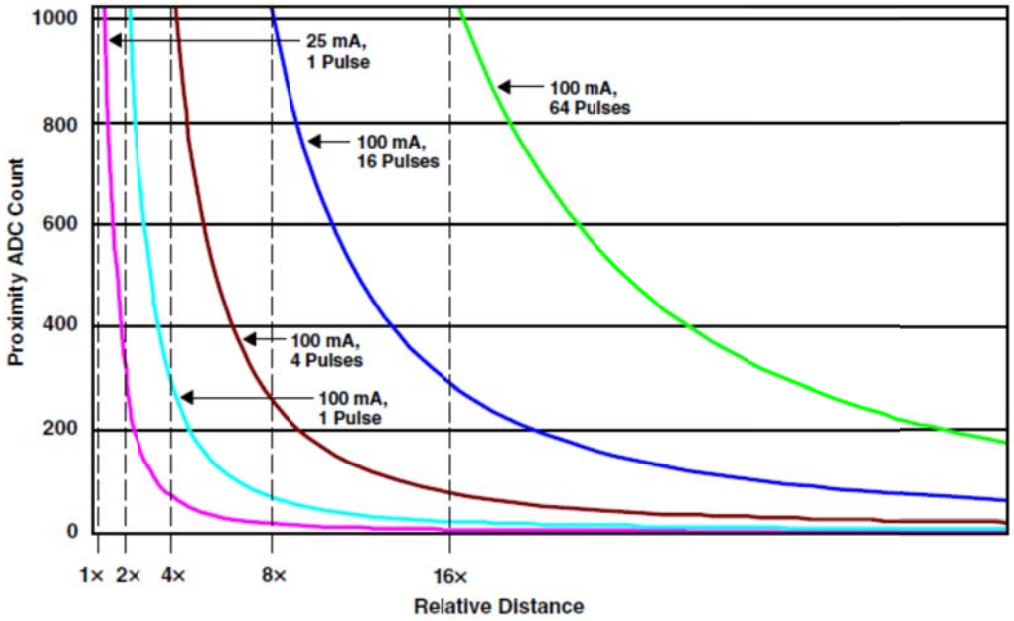


Figure 12: Proximity ADC Count vs Relative Distance

The following is a sample of Pseudo code for configuring TAOS Light-to-Digital Proximity and Ambient light sensor for the TSL2771X. There are many device options possible. Please refer to the device datasheet for a complete list of register setting options.

A recommended starting configuration for proximity is:

(PDRIVE) = 100mA drive
(PPCOUNT) = 4 pulse
(PDIODE) = CH1
(PTIME) = 2.7ms

Example of Initialize Using I²C Pseudo-code

The following shows the actual data sequence as written to the I²C port. The following symbols are used to represent the different I²C conditions:

S: I²C Start
SR: I²C Restart
P: I²C Stop

Refer to TSL2771X/2571X Datasheet for register description details. The following is a register set summary. The following is an example of an initialization sequence:

```
S_0x72_0xA1_0xDB_P   //Set ATIME (Reg 0x01) to 100ms
S_0x72_0xA2_0xFF_P   //Set PTIME (Reg 0x02) to 2.7ms
S_0x72_0xAE_0x04_P   //Set Prox pulses (Reg 0x0E)to 4
S_0x72_0xAF_0x20_P   //LED drive to 100mA, prox diode to CH1, AGain to 1x
S_0x72_0xAC_0x11_P   //Set Persistence (Prox and ALS) to 1 (optional)
S_0x72_0xA0_0x3F_P   //Enables Prox and ALS Interrupt,
                     //Enables Wait, Prox and ALS; Powers Up device
```

The following is an example of a read data sequence after an interval timer:

```
S_0x72_0xB4_SR_0x73_0xXX_0xXX_P //Sets Reg address for C0DATA (0x14)
                           //Reads two bytes before setting the stop bit
                           //C0DATA = (256 * C0DATAH + C0DATAL)

S_0x72_0xB6_SR_0x73_0xXX_0xXX_P //Sets Reg address for C1DATA (0x16)
                           //Reads two bytes before setting the stop bit
                           //C1DATA = (256 * C1DATAH + C1DATAL)

S_0x72_0xB8_SR_0x73_0xXX_0xXX_P //Sets Reg address for PDATA (0x18)
                           //Reads two bytes before setting the stop bit
                           //PDATA = (256 * PDATAH + PDATAL)
```

The following is an example of a command to stop the device operation (set to low power mode):

```
 S_0x72_0xA0_0x00_P        //Clears the Enable register (Reg 0)
```

The following is an example of a command to clear any interrupts:

```
 S_0x72_0xE7_P             //Clear All interrupts, CMD E0 special function
```

FAQ (Frequently Asked Questions)

Q: Why is it recommended for PTIME = 0xFF in the data sheet.

A: At a PTIME of 0xFF, the resultant ADC conversion is 10 bits, however, the actual noise free resolution of the ADC conversion is less that 10 bits. Increasing PTIME increases the digitized resolution of both the signal and noise, keeping SNR constant. It is more beneficial to keep PTIME low in order to maximize the number of proximity cycles that can be obtained in a given amount of time.

Q: Is there any way to decrease the amount of noise in the proximity data

A: The proximity signal has Gaussian noise which can be reduced by averaging multiple proximity samples. The noise will be reduced the square root of the number of samples averaged.

Q: Please provide clarification about AIEN, PIEN bit in the Enable register. When AIEN and PIEN are disabled, are the PINT and AINT bits in the status register are still generated. How can the device be forced to disable interrupts?

A: There are two types of interrupts: Software, via an I²C register and via a hardware pin. AINT and PINT register bits reflect whether an interrupt condition has occurred and are independent from the AIEN and PIEN interrupt enable bits. AIEN and PIEN control the gating of whether the AINT and PINT will appear on the external hardware interrupt pin.

Q: When AVALID bit goes to clear is that a result of PON=0? Is there another way to clear AVALID bit?

A: The AVALID bit indicates when the internal state machine has finished one cycle and there is valid data available in the internal registers. The bit is only cleared when PON=0.

Q: Is there another way other than testing AVALID to determine the end of an integration cycle?

A: The interrupt bits AINT and PINT can be used to indicate the end of an integration cycle if the persistence field is set to"every ADC cycle". Clearing the interrupt then waiting for the interrupt to be asserted can be used to determine the end of each integration cycle.

Q: Are the ALS and Proximity data synchronized? If so, how do we get the ALS and Prox data for the same machine cycle?

Example: Set the register below to keep the data at last conversion cycle:

- WEN, PEN, AEN, PON =1
- Read Channel Light data = Valid Data
- After confirm AVALID = 1, then PON=0
- Read Channel Light data = zero after PON=0.

Why isn't the last data read still in Channel light registers?

A: Registers reset to 0 after power down (PON=0). Data should remain in the data registers until they are overwritten with new data or reset at powerdown. In order to have data in both sets of data registers, read data register after both Prox and ALS integration cycles and before powering down the device.

Q: The 3:2 bit in control register is reserved. What will happen if I write to reserved bits?

A: Setting the reserved bits could adversely affect the performance of the device.

Q: Command register I²C type is described as follows in datasheet.
Type 00 is same as I²C read protocol - combined Format.
Type 01 is same as I²C read protocol.

A: Either type could be used in either protocol. The choice between type 00 and type 01 is determined by whether or not you want the address register to increment when reading multiple registers. As an example, you might want to read the two ALS data registers with type 01. In this case, you would reference register 15 and the two bytes that you would receive would be from registers 15 and 16. Conversely, if you wanted to poll a single register continuously waiting for a bit to be set, you could use type 00. In this instance, each byte received would be from the same register. Usually type 00 is used when reading a single register such as a status bit in a register, and type 01 is used when reading adjacent registers.

The choice in which protocol to use depends upon whether or not the program pointer is already pointing to the register that you want to read. In many cases the combined protocol is used to read registers because it allows you to first write a command to the slave to say which register you would like to read. In some cases, this can be avoided and the read protocol can be used. For more information, see section 3.10 (pages 13-15) of the I²C spec (http://www.nxp.com/documents/user_manual/UM10204.pdf).

Q: Continuous read of PDATA, sometimes has PDATA=0. Why do these reading occur?

A: This could be due to reading noise on the device. This would be the case if 0 were in the normal distribution of readings. If this is the case, multiple prox readings averaged together would help improve SNR.

Conclusion

Following the guidelines of this document will assist in successfully communicating with TAOS digital light sensors. Please visit www.taosinc.com for more information on all the families of TAOS light sensing products.