INTELLIGENT OPTO SENSOR Number 4

DESIGNER'S NOTEBOOK

TAOS TEXAS ADVANCED OPTOELECTRONIC SOLUTIONS™

Using the TSL3301 with a Microcontroller

Controlling the TSL3301 in the Absence of a Continuous Clock

Contributed by Phil Pilgrim, Bueno Systems, Inc.

June 21, 2001
Revision A

Introduction

The TAOS TSL3301 linear optical sensor array with analog-to-digital converter simplifies acquisition of 102-pixel, single-dimension images using simple 8-bit microcontrollers. All communication with the device occurs using three digital (TTL-compatible) lines: SDIN, the serial data input to the sensor; SDOUT, the serial data output from the sensor; and SCLK, the serial data clock. The protocol is isosynchronous. That is to say, even though a synchronous clock is used at the bit level to clock data into and out of the device, the data itself contains additional byte synchronization cues in the form of start and stop bits. Each 8-bit data packet is preceded by a single, 0-level start bit and is followed by a single 1-level stop bit. Therefore each byte transfer requires 10 clocks.

The bytes sent to the TSL3301 include both commands and data. A command may tell the device to start or stop integration. Data may include values being written into the sensor's gain or offset registers. Because SCLK is also the device's sole state machine clock, *once a command is written, additional clocks may be needed by the device to carry out the command.*

For this reason, the TSL3301 is most conveniently deployed in systems that can provide a continuously running clock stream. For example, a USART running in isosynchronous mode would provide such a resource. In such a system, one merely has to send bytes to the USART and retrieve them. The USART provides a continuous clock stream upon which the serialized bytes "hitch a ride", thus insuring ample post-command clocking where necessary.

But USARTs are a luxury in many applications, particularly when simple 8-bit microcontrollers are deployed. In these situations, it is more common for serial I/O to be done by "bit banging", i.e. toggling and reading I/O pins directly under program control. Needless to say, providing a continuous clock under these circumstances at anywhere near the TSL3301's top performance level (10MHz) would be nearly impossible.

The obvious solution, and the topic of this application note, is to provide only the clocking needed for each function, letting SCLK sit idle at all other times. We will only consider half-duplex operation here, *i.e.* non-overlapped reading and writing. Though it is possible to increase performance dramatically by breaching this restriction, it can get quite messy and is beyond the scope of this document.

Please note: The techniques outlined here reflect a particular implementation, namely the TSL3301EVM. They will not be optimum for every application but rather illustrate a method that “works” in a particular instance. *The TSL3301 datasheet should always be used as the reference of highest authority when writing your own drivers for this chip.*

Throughout this document the following conventions are used. “0xHH” is used to represent a byte value in hexadecimal form, where the last two characters “HH” represent the two character hexadecimal value. The notation “Byte.0” represents one bit of a byte, names by “Byte”, with the bit position indicated by the digit following the decimal point.

Controlling the TSL3301 can be broken up into eight basic steps:

1. Reset.
2. Device setup (or re-setup).
3. Start integration.
4. Wait for the desired integration time.
5. Stop integration.
6. Start pixel readout.
7. Read 102 pixels.
8. Go back to step 3 (or 2).

Reset

Resetting the TSL3301 is accomplished by sending a <break> condition (30 clocks with SDIN held low), followed by 10 idle clocks with SDIN high, followed by a reset command and a write to the MODE register.

1. Clear SCLK.
2. Clear SDIN.
3. Pulse SCLK (i.e. high, then low) 30 times. ;<break>
4. Set SDIN.
5. Pulse SCLK 10 times. ;Synchronize start bit detection.
6. ***SEND***(0x1B) ;RESET command.
7. Pulse SCLK 5 times.
8. ***SEND***(0x5F) ;Write to mode register command.
9. ***SEND***(0x00) ;Clear mode register.

SEND is a subroutine defined as follows and is used throughout this document.

SEND(*Byte*):

```
1.  Clear SDIN.                     ;Start bit.
2.  Pulse SCLK once.
3.  Count = 8.
4.  SDIN = Byte.0                   ;8 data bits, LSB first.
5.  Pulse SCLK.
6.  Shift Byte right one.
7.  Decrement Count.
8.  Jump to 4 if Count > 0.
9.  Set SDIN                        ;Stop bit.
10. Pulse SCLK once.
11. Return.
```

Please note that SEND always returns with SDIN set and SCLK cleared.

Device Setup

Setting up the TSL3301, once it is reset and the mode register has been cleared, involves writing to the gain and offset registers. In this example, we will set the gain to 5 and the offset to –1:

```
1.  SEND(0x40)                      ;Left offset
2.  SEND(0x81)                      ;-1 in sign/magnitude
3.  SEND(0x41)                      ;Left gain
4.  SEND(0x05)                      ;5
5.  SEND(0x42)                      ;Middle offset
6.  SEND(0x81)                      ;-1 in sign/magnitude
7.  SEND(0x43)                      ;Middle gain
8.  SEND(0x05)                      ;5
9.  SEND(0x44)                      ;Right offset
10. SEND(0x81)                      ;-1 in sign/magnitude
11. SEND(0x45)                      ;Right gain
12. SEND(0x05)                      ;5
```

Start Integration

Starting integration simply involves sending the start integrate command to the sensor, followed by some extra clocks:

```
1.  SEND(0x08)              ;Start integration.
2.  Pulse SCLK 22 times.
```

Wait for the Desired Integration Time

Most microcontrollers have a timer of some sort – usually a free-running 8- or 16-bit count-up or count-down affair. An easy way to get the correct integration time is to set this timer immediately upon starting integration to a value which will cause the timer to roll to zero when the desired time has elapsed. This can be used to cause an interrupt, or you can just poll the interrupt request flag with the interrupt disabled. The actual technique will vary depending on which microcontroller you use. If you use interrupts, how to respond to the interrupt is yet another issue, which is taken up later in this paper.

Stop Integration

Stopping integration, like starting, involves sending a command and some extra clocks:

```
1.  SEND(0x10)
2.  Pulse SCLK 5 times.
```

Start Pixel Readout

To initiate readout from the TSL3301, send the Start Read command and wait for a start bit on SDOUT:

```
1.  SEND(0x02)
2.  Exit if SDOUT is low.
3.  Pulse SCLK once.
4.  Jump back to 2.
```

Read 102 Pixels

```
1.  Pixels = 102
2.  PixelValue = RCV()
3.  Decrement Pixels.
4.  Jump back to 2 if Pixels > 0.
```

This uses the subroutine ***RCV***, defined as follows:

RCV:

```
1.  Pulse SCLK.                 ;Get past start bit.
2.  Count = 8.                  ;Initialize bit count.
3.  Value = 0.                  ;Initialize return value.
4.  Shift Value right by 1.     ;Data comes out LSB first.
5.  Value.7 = SDOUT.
6.  Pulse SCLK
7.  Decrement Count.
8.  Jump back to 4 if Count > 0.
9.  Pulse SCLK                  ;Into next start bit.
10. Return Value.
```

Note that you do not need to check for the start bit of each pixel – just the first one. After that, it's safe to assume they're where they're supposed to be.

Timing Issues

Aspects we have ignored in the prior algorithm clips are the timing requirements of both the sensor and the microprocessor. Both need to be addressed by the programmer. For those using very fast processors (*e.g.* the Ubicom SX series), it is possible to generate or expect signals that violate the TSL3301's timing specs. You may have to sprinkle NOPs in your code to compensate. Also, be sure you are familiar with your microcontroller's pipelining, especially when it comes to I/O. Changes to a controller's output pin resulting from one instruction may occur *after* an input pin is read by the *next* instruction. As a consequence, you may need extra delay between pulsing SCLK and reading SDOUT. In tight timing situations, you can even set up the next SCLK pulse by raising SCLK, *then* reading SDOUT. When in doubt, use a scope!

Another timing issue regards integration time consistency. To get the smoothest, most even scans, all frames need to be integrated for as close to the same time as possible. As alluded to before, an on-chip timer in the microcontroller commonly handles this. In the TSL3301 EVM, a timer interrupt service routine executes 344827 times per second and handles the state machine associated with triggers, delays, and integration time. But reads and writes to the sensor are performed in the foreground. The interaction between foreground and background is carried out with flags. What this means, though, is that the foreground routine must sit and poll the flags for the proper times to start and stop integration. In many applications, this would be grossly inefficient.

A possible solution would be to let the timer interrupt routine start and stop integration directly. You have to be careful, though, that this cannot happen during foreground reads or writes to the sensor. A standard tactic would be to disable interrupts during foreground operations. But to do so might compromise the timing, particularly where very short exposures are concerned. It would be better to use a flag to signal who has control of the sensor. When the foreground program needs a new frame, it would set an "integration request" flag. This would signal the interrupt service routine to start a new integration. As long as this flag was set, the foreground program would know not to access the sensor; but it would also be free to do other things. At the end of the integration period, the background routine would stop integration and clear the request flag. This would signal the foreground that an exposure was ready to be read out and that it could resume control of the sensor.

A final word on timing involves efficiency. Reading 102 pixels requires more than 1020 clock pulses. You may need to get this done as quickly as possible. The TSL3301 allows up to a 10MHz clock rate, which works out to 102µs for a full readout. But even a Ubicom SX processor running at 50MIPs can't sustain this rate with the looping construct we used in ***RCV***. The trick is to unroll the loop, eliminating the bit counter and conditional branches:

RCV:

```
1. Pulse SCLK.              ;Get past start bit.
2. Value.0 = SDOUT
3. Pulse SCLK
4. Value.1 = SDOUT
```

```
5.  Pulse SCLK
6.  Value.2 = SDOUT
7.  Pulse SCLK
8.  Value.3 = SDOUT
9.  Pulse SCLK
10. Value.4 = SDOUT
11. Pulse SCLK
12. Value.5 = SDOUT
13. Pulse SCLK
14. Value.6 = SDOUT
15. Pulse SCLK
16. Value.7 = SDOUT
17. Pulse SCLK
18. Pulse SCLK                      ;Into next start bit.
19. Return Value.
```