The TCS230 Color Sensor & MD_TCS230 Arduino Library

Contents

Sensor Package and Pinouts

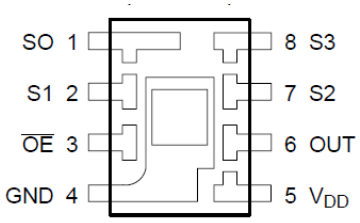

Table 1 - Terminal Functions

Name	Pin	I/O	Description
GND	4		Power Supply Ground. All Voltages are referenced to this ground
V_{DD}	5		Supply Voltage (2.7-5.5V)
/OE	3	I	Enable f_O (active low). When OE is high the OUT becomes high impedance, allowing multiple sensors to share the same OUT line
OUT	6	O	Output frequency f_O
S0, S1	1, 2	I	Output frequency scale selection inputs (see 0 0)
S2, S3	7, 8	I	Photodiode (color filter) selection inputs (see 0 0)

Functional Description

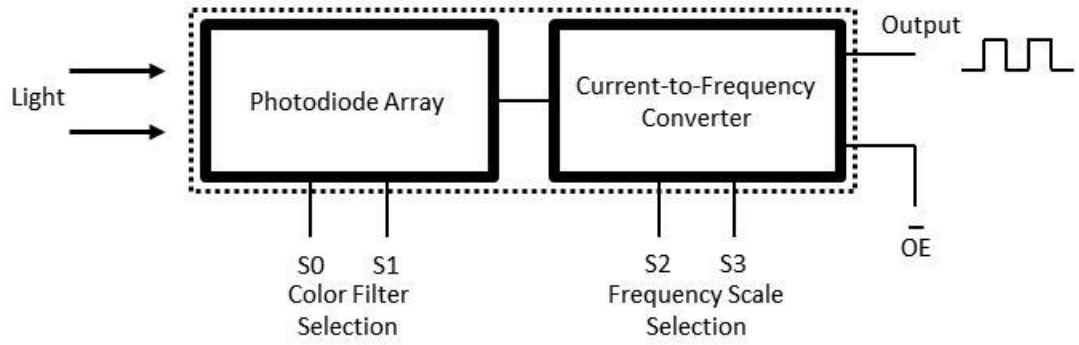

Figure 1 - Functional Block Diagram

- Digital inputs and digital output allow direct interface to a microcontroller or other logic circuitry.
- Output enable (/OE) places the output in the high-impedance state for multiple-unit sharing of a microcontroller input line.
- The output is a square wave (50% duty cycle) with frequency (f_O) directly proportional to light intensity (irradiance).
- The full-scale output frequency can be scaled by one of three preset values and off, via two control input pins (S0, S1 0).
- Four types of photodiodes - Red, Green, Blue and Clear (no filter) - are pin-selectable (S2, S3 0) to read the individual components of the color detected.

Table 2 - Scale Selection Inputs

S0	S1	Output Frequency Scaling (f_O)
L	L	Power Down
L	H	2%
H	L	20%
H	H	100%

Table 3 - Color Filter Selection

S2	S3	Photodiode (Color) Selection
L	L	Red
L	H	Blue
H	L	Clear (no filter)
H	H	Green

Using Breakout Boards

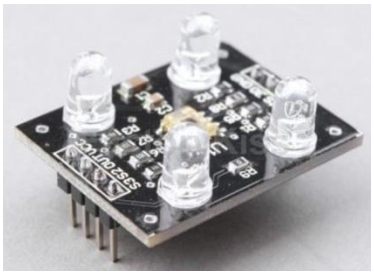

Figure 2 - Sensor Breakout Board

The sensor can be purchased mounted on any number of inexpensive breakout boards, similar in design to that shown in 0. These boards extend the package connections to header pins, and integrate LEDs for illumination of the target object.

Many of these boards include a design 'feature' connecting OE directly to GND. Test with a multimeter by measuring the resistance between OE and GND – zero ohms reading indicates a direct connection.

This configuration will create problems if

- The OE line is driven HIGH by external circuitry – this creates a short circuit between GND and the output of the driving circuit!
- Multiple sensors need to be connected to the same OUT, as the OUT lines cannot be switched into high impedance mode.

Restoring normal function to the OE input is possible by cutting a track on the PCB. This is found between resistors R5 and R8, highlighted by the circle in 0. This may be different for other boards, so it is wise to follow the connections to make sure the correct track is cut.

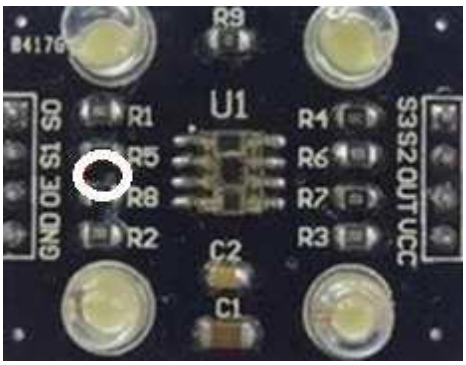

Figure 3 - OE Modification

Figure 4 - Sensor Shroud

Another modification to make the device readings consistent and repeatable is to shroud the sensor. This can be made from black card, wrapped and taped to the breakout board. The shroud eliminates stray light and ensures all the reflected light goes back to the sensor - use black so that the light reflected back into the sensor does not include color components originating from the shroud. My setup is shown in 0 above.

Interpreting Sensor Data

The output is a square wave (50% duty cycle) with frequency (f_O) directly proportional to light intensity:

$$f_O = f_D + (\mathrm{Re})(\mathrm{Ee})$$

where f_O is the output frequency; f_D is the output frequency for dark condition (when Ee = 0); Re is the device responsivity for a given wavelength of light in kHz/(mW/cm^2); Ee is the incident irradiance in mW/cm^2.

f_D is an output frequency resulting from leakage currents. As shown in the equation above, this frequency represents a light-independent term in the total output frequency f_O. At very low light levels (dark colors), this dark frequency can be a significant portion of f_O. The dark frequency is also temperature dependent.

As f_O is directly proportional to frequency, it is possible to map between the frequency and RGB color value (0-255 for each of R, G and B) using linear interpolation.

Two points on the RGB line are well determined – pure Black (RGB 0, 0, 0) and pure White (255, 255, 255). The values returned by the sensor can be read using easily obtainable color swatches:

- A black color card gives us the dark condition constant f_D. This is the origin (zero value) for the RGB straight line conversion.
- A white color card gives us the extreme RGB point f_W, also known as white balance. Knowing f_D, this value can be used to scale all intermediate frequencies to a corresponding RGB value.

The proportional relationship is expressed by the standard straight line equation $y = mx + b$ where

- y is the reading obtained (in our case f_O)
- x is the normalised RGB value
- b is the value of y when x is 0 (in our case f_D)
- m is the slope, or proportionality constant, of the line (in our case $[f_W\text{-}f_D]/255$).

The resulting equation is

$$f_O = f_D + \frac{x \cdot (f_W - f_D)}{255}$$

or, rearranging to give us the desired RGB value

$$x = \frac{255 \cdot (f_O - f_D)}{(f_W - F_D)}$$

Implementing a System

Measuring Frequency

The sensor's output is a square wave with 50% duty cycle (on/off equal amounts) with frequency directly proportional to light intensity, so accurately measuring the frequency is a fundamental requirement of any implementation.

To measure frequency we need to measure the number of cycles in a fixed time period. The number of cycles in 1 second is measured in Hz. As long as the system is able to sample cycle transitions at a rate at least twice the expected frequency, we can be assured of an accurate reading. This rate should be easily achievable with an Arduino based implementation, even at 100% frequency scale.

Output scaling can be used to increase the resolution for a given clock rate or to maximize resolution as the light input changes.

The MD_TCS230 library uses a timer interrupt for the gate interval. If another interrupt is running, or interrupts are disabled by the main program, response to the timer could be delayed. That will lengthen the gate interval, perhaps leading to counting more cycles. The next gate interval will be shorter (if normal interrupt response occurs), so a corresponding decrease will occur in the next measurement. This mechanism is implemented in the *FreqCount* library.

The longer the time period allowed for reading a value, the higher the accuracy. The library is set to use 1000ms, giving a value in Hz, as the basis of measurement. A divisor is applied to the time and the resulting count is multiplied by the divisor. Shorter sampling times are possible when the color differences between samples are more distinct. Higher resolution is obtained using a lower time divisor. There is no additional benefit counting cycles beyond 1000ms using this method.

Sensor Calibration

To calibrate the sensors, we need to measure the raw data that is returned with black and white test cards and then use that data to scale subsequent readings to an RGB value.

It is not necessary to calibrate the sensor every time it is used. Calibration can be done separately from the application, and the constants determined during testing applied to the sensor from the code running the final application. Care should be taken to calibrate the sensor in a position that closely approximates final conditions.

References

TAOS Inc, "TCS230 Programmable Color Light-to-Frequency Converter", TAOS046, February 2003.

Charles Poynton, "Sensing Color with the TAOS TCS230", TAOS Inc, 17 May 2005.

Paul J Stoffregen, "FreqCount Library", http://www.pjrc.com/teensy/td_libs_FreqCount.html,accessed 18 February 2013.

MD_TCS230 Library Reference

Initialising

MD_TCS230(uint8_t s2, uint8_t s3);
MD_TCS230(uint8_t s2, uint8_t s3, uint8_t oe);
MD_TCS230(uint8_t s2, uint8_t s3, uint8_t s0, uint8_t s1);
MD_TCS230(uint8_t s2, uint8_t s3, uint8_t s0, uint8_t s1, uint8_t oe);

Various forms of the constructor for this class. The parameters are all the Arduino pin numbers used to control the nominated pins on the TCS230. The form used will depend on the configuration of the hardware connections to the Arduino. At a minimum S2 and S3 are required for color filter selection.

begin()

Initialise the object. The object needs to initialise itself using Arduino libraries and so cannot be completely initialised in the constructor. Should be done once for every object created.

Reading Data

read()

Start a read cycle from the Sensor. The function will coordinate enabling the sensor, selecting the right filters in sequence and calling the FreqCount library to obtain a full color reading.

available()

Determine when a sensor read has completed. The method will return *true* when a value is ready for processing, at which time a call to *getRGB()* or *getRaw()* will retrieve the data for the calling program.

getRGB(colorData *rgb)

Get the last read sensor data in RGB format into the structure **rgb*. The elements of RGB are retrieved using *value[TCS230_RGB_R]*, *value[TCS230_RGB_G]* and *value[TCS230_RGB_B]* for the Red, Green and Blue elements, respectively.

getRaw(sensorData *d)

Get the last read sensor data as raw counts into the structure **d*. The elements may be retrieved using *value[TCS230_RGB_R]*, *value[TCS230_RGB_G]* and *value[TCS230_RGB_B]* for the Red, Green and Blue elements, respectively.

This function is most useful in obtaining calibration data that to be passed back to the *setDarkCal()* and *setWhiteCal()* methods. To improve accuracy in the calibration, the data may be averaged over several readings before setting the calibration data.

setSampling(uint8_t t)

Set the sampling period divisor. The library is set for a maximum of 1000ms for a sample (divisor = 1). Higher divisors create shorter sample times (eg, 5 = 200ms, 10 = 100ms, 100 = 10ms). Shorter samples are less accurate and only recommended when the range of colors to be detected are highly separated on the RGB color chart. The 'right' value for the divisor is dependent on the application. The default value is 10 (100ms sample).

readSingle()

Initiate a blocking read of from the sensor. The method returns the raw data read from the sensor as a *uint32_t*, normalised to a value in Hz, once the read cycle is completed. How long this takes is determined by the sampling period. The calling program is responsible for enabling the device and setting the appropriate color filter.

setDarkCal(sensorData *d)

Set the dark calibration data. The information is retrieved using a call to *getRaw()*.

setWhiteCal(sensorData *d)

Set the white balance calibration data. The information is retrieved using a call to *getRaw*().

Miscellaneous

setFilter(uint8_t f)

Set the photodiode filter to one of *TCS230_RGB_R*, *TCS230_RGB_G*, *TCS230_RGB_B* or *TCS230_RGB_X* to filter Red, Green, Blue or leave unfiltered. The read method automatically invokes all the filters in sequence using this method.

setFrequency(uint8_t f)

Set the sensor frequency prescaler to *TCS230_FREQ_HI*, *TCS230_FREQ_MID*, *TCS230_FREQ_LO* or *TCS230_FREQ_OFF* (100%, 20%, 2%, off respectively) using the pins defined for S0 and S1. If none of the pins are defined, no action is taken. The default frequency initialised by the library is *TCS230_FREQ_HI*.

setEnable(bool t)

Enable (*true*) or disable (*false*) sensor output by (in order of preference):

- If the OE pin is defined, setting the OE pin to the appropriate HIGH/LOW conditions. This is preferred mode as the high impedance mode can be used for OUT signal isolation.
- If the SO and S1 pins are defined, setting the output frequency to the value set by the *setFrequency()* method or to *TCS230_FREQ_OFF*.
- No action is taken if none of OE, S0 and S1 is defined.