Arduino Gyroscope Driver

Generated by Doxygen 1.8.9.1

Tue Aug 18 2015 22:52:00

ii CONTENTS

Contents

1	Hier	Hierarchical Index 1					
	1.1	Class H	lierarchy	1			
2	Clas	s Index		1			
	2.1	Class L	ist	1			
3	File	Index		1			
	3.1	File List	t	1			
4	Clas	s Docun	nentation	2			
	4.1	ColorRe	ecognition Class Reference	2			
		4.1.1	Detailed Description	2			
		4.1.2	Member Function Documentation	2			
	4.2	ColorRe	ecognitionTCS230 Class Reference	3			
		4.2.1	Detailed Description	5			
		4.2.2	Member Enumeration Documentation	5			
		4.2.3	Constructor & Destructor Documentation	5			
		4.2.4	Member Function Documentation	5			
		4.2.5	Member Data Documentation	7			
	4.3	ecognitionTCS230PI Class Reference	8				
		4.3.1	Detailed Description	9			
		4.3.2	Member Enumeration Documentation	9			
		4.3.3	Constructor & Destructor Documentation	9			
		4.3.4	Member Function Documentation	9			
		4.3.5	Member Data Documentation	11			
5 File Documentation		Docume	entation	11			
		ColorRe	ecognition.cpp File Reference	11			
		5.1.1	Macro Definition Documentation	12			
	5.2	ColorRe	ecognition.cpp	12			
	5.3	ColorRe	ecognition.h File Reference	13			
	5.4	.4 ColorRecognition.h					
	5.5	ecognitionTCS230.cpp File Reference	14				
		5.5.1	Macro Definition Documentation	14			
	5.6	ColorRe	ecognitionTCS230.cpp	14			
	5.7	ColorRe	ecognitionTCS230.h File Reference	16			
		5.7.1	Macro Definition Documentation	16			
	5.8	ColorRe	ecognitionTCS230.h	18			
	5.9	ColorRecognitionTCS230PI.cpp File Reference					
			Macro Definition Documentation	19			

1 Hierarchical Index

5.10 ColorRecognitionTCS230PI.cpp	19
5.11 ColorRecognitionTCS230PI.h File Reference	20
5.11.1 Macro Definition Documentation	21
5.12 ColorRecognitionTCS230PI.h	22
5.13 simple_read.c File Reference	23
5.13.1 Function Documentation	23
5.14 simple_read.c	24
Index	25
1 Hierarchical Index	
1.1 Class Hierarchy	
This inheritance list is sorted roughly, but not completely, alphabetically:	
ColorRecognition	2
ColorRecognitionTCS230	3
ColorRecognitionTCS230PI	8
2 Class Index	
2.1 Class List	
Here are the classes, structs, unions and interfaces with brief descriptions:	
ColorRecognition Arduino - Color Recognition Sensor	2
ColorRecognitionTCS230	3
ColorRecognitionTCS230PI	8
3 File Index	
3.1 File List	
Here is a list of all files with brief descriptions:	
ColorRecognition.cpp	11
ColorRecognition.h	13
ColorRecognitionTCS230.cpp	14
ColorRecognitionTCS230.h	16
ColorRecognitionTCS230Pl.cpp	18

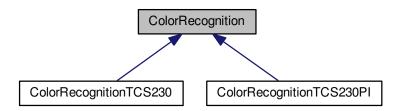
ColorRecognitionTCS230Pl.h	20
simple read.c	23

4 Class Documentation

4.1 ColorRecognition Class Reference

#include <ColorRecognition.h>

Inheritance diagram for ColorRecognition:



Public Member Functions

- virtual unsigned char getRed ()=0
- virtual unsigned char getGreen ()=0
- virtual unsigned char getBlue ()=0
- virtual bool fillRGB (unsigned char buf[3])=0

4.1.1 Detailed Description

Arduino - Color Recognition Sensor.

ColorRecognition.h

The abstract class for the color recognition sensors.

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 14 of file ColorRecognition.h.

4.1.2 Member Function Documentation

4.1.2.1 virtual bool ColorRecognition::fillRGB (unsigned char buf[3]) [pure virtual]

Returns the blue color intensity.

The blue color intensity.

 $Implemented\ in\ Color Recognition TCS 230,\ and\ Color Recognition TCS 230 PI.$

4.1.2.2 virtual unsigned char ColorRecognition::getBlue() [pure virtual]

Returns the blue color intensity.

The blue color intensity.

Implemented in ColorRecognitionTCS230, and ColorRecognitionTCS230PI.

4.1.2.3 virtual unsigned char ColorRecognition::getGreen() [pure virtual]

Returns the green color intensity.

The green color intensity.

Implemented in ColorRecognitionTCS230, and ColorRecognitionTCS230PI.

4.1.2.4 virtual unsigned char ColorRecognition::getRed() [pure virtual]

Returns the red color intensity.

The red color intensity.

Implemented in ColorRecognitionTCS230, and ColorRecognitionTCS230PI.

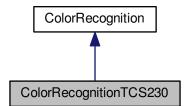
The documentation for this class was generated from the following file:

· ColorRecognition.h

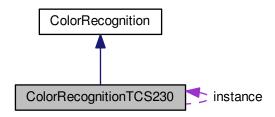
4.2 ColorRecognitionTCS230 Class Reference

#include <ColorRecognitionTCS230.h>

Inheritance diagram for ColorRecognitionTCS230:



Collaboration diagram for ColorRecognitionTCS230:



Public Types

enum Filter { RED_FILTER, GREEN_FILTER, BLUE_FILTER, CLEAR_FILTER }

Public Member Functions

- void initialize (unsigned char outPin, unsigned char s2Pin, unsigned char s3Pin)
- void adjustWhiteBalance ()
- unsigned char getRed ()
- unsigned char getGreen ()
- unsigned char getBlue ()
- bool fillRGB (unsigned char buf[3])

Static Public Member Functions

- static ColorRecognitionTCS230 * getInstance ()
- static void setFilter (Filter filter)

Public Attributes

· Filter currentFilter

Private Member Functions

• ColorRecognitionTCS230 ()

Static Private Member Functions

- static void externalInterruptHandler ()
- static void timerInterruptHandler ()

Private Attributes

- unsigned char s2Pin
- unsigned char s3Pin
- · unsigned char outPin
- · int count
- int lastFrequencies [3]
- int whiteBalanceFrequencies [3]

Static Private Attributes

• static ColorRecognitionTCS230 instance

4.2.1 Detailed Description

Definition at line 96 of file ColorRecognitionTCS230.h.

4.2.2 Member Enumeration Documentation

4.2.2.1 enum ColorRecognitionTCS230::Filter

Filter color enumeration.

Enumerator

RED_FILTER
GREEN_FILTER
BLUE_FILTER
CLEAR_FILTER

Definition at line 141 of file ColorRecognitionTCS230.h.

4.2.3 Constructor & Destructor Documentation

4.2.3.1 ColorRecognitionTCS230::ColorRecognitionTCS230() [inline], [private]

Private constructor.

Definition at line 227 of file ColorRecognitionTCS230.h.

4.2.4 Member Function Documentation

4.2.4.1 void ColorRecognitionTCS230::adjustWhiteBalance ()

Store the current read as the maximum frequency for each color.

It tells what is considered white.

Definition at line 31 of file ColorRecognitionTCS230.cpp.

4.2.4.2 void ColorRecognitionTCS230::externalInterruptHandler() [static], [private]

Device output interruption handler.

Definition at line 38 of file ColorRecognitionTCS230.cpp.

4.2.4.3 bool ColorRecognitionTCS230::fillRGB (unsigned char buf[3]) [virtual]

Returns the blue color intensity.

The blue color intensity.

Implements ColorRecognition.

Definition at line 85 of file ColorRecognitionTCS230.cpp.

4.2.4.4 unsigned char ColorRecognitionTCS230::getBlue() [virtual]

Returns the blue color intensity.

The blue color intensity.

Implements ColorRecognition.

Definition at line 78 of file ColorRecognitionTCS230.cpp.

4.2.4.5 unsigned char ColorRecognitionTCS230::getGreen() [virtual]

Returns the green color intensity.

The green color intensity.

Implements ColorRecognition.

Definition at line 71 of file ColorRecognitionTCS230.cpp.

4.2.4.6 static ColorRecognitionTCS230* ColorRecognitionTCS230::getInstance() [inline], [static]

Singleton.

Gets the instance of the driver.

Returns

Definition at line 155 of file ColorRecognitionTCS230.h.

4.2.4.7 unsigned char ColorRecognitionTCS230::getRed() [virtual]

Returns the red color intensity.

The red color intensity.

Implements ColorRecognition.

Definition at line 64 of file ColorRecognitionTCS230.cpp.

4.2.4.8 void ColorRecognitionTCS230::initialize (unsigned char outPin, unsigned char s2Pin, unsigned char s3Pin)

Initializes the IO and timers.

Parameters

outPin	The out pin. (NOTE: It must be the 2 or 3 pin to support external interrupts).
s2Pin	The s2 pin.
s3Pin	The s3 pin.

Returns

Definition at line 18 of file ColorRecognitionTCS230.cpp.

4.2.4.9 void ColorRecognitionTCS230::setFilter (Filter filter) [static]

Sets the s2 and s3 pins according of the color passed as filter.

```
S2 S3 PHOTODIODE TYPE
L L Red
L H Blue
H L Clear (no filter)
H H Green
```

Parameters

```
filter The next filter.
```

Definition at line 92 of file ColorRecognitionTCS230.cpp.

```
4.2.4.10 void ColorRecognitionTCS230::timerInterruptHandler( ) [static], [private]
```

TimerOne interrupt handler.

Definition at line 42 of file ColorRecognitionTCS230.cpp.

4.2.5 Member Data Documentation

```
4.2.5.1 int ColorRecognitionTCS230::count [private]
```

Holds the number of interrupts of the current filter.

Definition at line 119 of file ColorRecognitionTCS230.h.

4.2.5.2 Filter ColorRecognitionTCS230::currentFilter

Current filter.

Definition at line 148 of file ColorRecognitionTCS230.h.

4.2.5.3 ColorRecognitionTCS230 ColorRecognitionTCS230::instance [static], [private]

Singleton.

The instance.

Definition at line 134 of file ColorRecognitionTCS230.h.

4.2.5.4 int ColorRecognitionTCS230::lastFrequencies[3] [private]

Holds the last count for each filter.

Definition at line 124 of file ColorRecognitionTCS230.h.

4.2.5.5 unsigned char ColorRecognitionTCS230::outPin [private]

The out pin.

NOTE: It must be the 2 or 3 pin to support external interrupts.

Definition at line 114 of file ColorRecognitionTCS230.h.

4.2.5.6 unsigned char ColorRecognitionTCS230::s2Pin [private]

The s2 pin.

Definition at line 102 of file ColorRecognitionTCS230.h.

4.2.5.7 unsigned char ColorRecognitionTCS230::s3Pin [private]

The s3 pin.

Definition at line 107 of file ColorRecognitionTCS230.h.

4.2.5.8 int ColorRecognitionTCS230::whiteBalanceFrequencies[3] [private]

Holds the maximum frequencies.

Definition at line 129 of file ColorRecognitionTCS230.h.

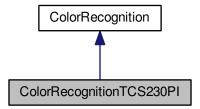
The documentation for this class was generated from the following files:

- ColorRecognitionTCS230.h
- ColorRecognitionTCS230.cpp

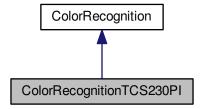
4.3 ColorRecognitionTCS230PI Class Reference

#include <ColorRecognitionTCS230PI.h>

Inheritance diagram for ColorRecognitionTCS230PI:



Collaboration diagram for ColorRecognitionTCS230PI:



Public Types

enum Filter { RED_FILTER, GREEN_FILTER, BLUE_FILTER, CLEAR_FILTER }

Public Member Functions

- ColorRecognitionTCS230PI (unsigned char outPin, unsigned char s2Pin, unsigned char s3Pin)
- void adjustWhiteBalance ()
- void adjustBlackBalance ()
- unsigned char getRed ()
- unsigned char getGreen ()
- unsigned char getBlue ()
- bool fillRGB (unsigned char buf[3])
- long getFrequency (unsigned int samples)
- void setFilter (Filter filter)

Private Attributes

- unsigned char s2Pin
- unsigned char s3Pin
- unsigned char outPin
- long minFrequency [3]
- long maxFrequency [3]

4.3.1 Detailed Description

Definition at line 81 of file ColorRecognitionTCS230Pl.h.

- 4.3.2 Member Enumeration Documentation
- 4.3.2.1 enum ColorRecognitionTCS230PI::Filter

Filter color enumeration.

Enumerator

RED_FILTER
GREEN_FILTER
BLUE_FILTER
CLEAR_FILTER

Definition at line 114 of file ColorRecognitionTCS230Pl.h.

- 4.3.3 Constructor & Destructor Documentation
- 4.3.3.1 ColorRecognitionTCS230PI::ColorRecognitionTCS230PI (unsigned char *outPin*, unsigned char *s2Pin*, unsigned char *s3Pin*)

Private constructor.

Definition at line 16 of file ColorRecognitionTCS230PI.cpp.

- 4.3.4 Member Function Documentation
- 4.3.4.1 void ColorRecognitionTCS230PI::adjustBlackBalance ()

Store the current read as the maximum frequency for each color.

It tells what is considered white.

Definition at line 37 of file ColorRecognitionTCS230Pl.cpp.

4.3.4.2 void ColorRecognitionTCS230PI::adjustWhiteBalance ()

Store the current read as the minimum frequency for each color.

It tells what is considered black.

Definition at line 30 of file ColorRecognitionTCS230PI.cpp.

4.3.4.3 bool ColorRecognitionTCS230PI::fillRGB (unsigned char buf[3]) [virtual]

Returns the blue color intensity.

The blue color intensity.

Implements ColorRecognition.

Definition at line 68 of file ColorRecognitionTCS230PI.cpp.

4.3.4.4 unsigned char ColorRecognitionTCS230PI::getBlue() [virtual]

Returns the blue color intensity.

The blue color intensity.

Implements ColorRecognition.

Definition at line 60 of file ColorRecognitionTCS230PI.cpp.

4.3.4.5 long ColorRecognitionTCS230PI::getFrequency (unsigned int samples)

Gets the frequency from the out pin.

NOTE: It uses pulseln, collects some samples and calculate the frequency.

The out pin generates a square wave, we sum the times between the raise edge and divide by the number of samples.

```
1 2 3
---- ---- ----- -----
```

Returns

The pin frequency.

Definition at line 87 of file ColorRecognitionTCS230PI.cpp.

4.3.4.6 unsigned char ColorRecognitionTCS230PI::getGreen() [virtual]

Returns the green color intensity.

The green color intensity.

Implements ColorRecognition.

Definition at line 52 of file ColorRecognitionTCS230PI.cpp.

4.3.4.7 unsigned char ColorRecognitionTCS230Pl::getRed() [virtual]

Returns the red color intensity.

The red color intensity.

Implements ColorRecognition.

Definition at line 44 of file ColorRecognitionTCS230PI.cpp.

5 File Documentation 11

4.3.4.8 void ColorRecognitionTCS230PI::setFilter (Filter filter)

Sets the s2 and s3 pins according of the color passed as filter.

```
S2 S3 PHOTODIODE TYPE
L L Red
L H Blue
H L Clear (no filter)
H H Green
```

Parameters

```
filter The next filter.
```

Definition at line 75 of file ColorRecognitionTCS230Pl.cpp.

4.3.5 Member Data Documentation

```
4.3.5.1 long ColorRecognitionTCS230Pl::maxFrequency[3] [private]
```

The maximum frequency.

Definition at line 107 of file ColorRecognitionTCS230Pl.h.

4.3.5.2 long ColorRecognitionTCS230PI::minFrequency[3] [private]

The minimum frequency.

Definition at line 102 of file ColorRecognitionTCS230Pl.h.

4.3.5.3 unsigned char ColorRecognitionTCS230Pl::outPin [private]

The out pin.

Definition at line 97 of file ColorRecognitionTCS230Pl.h.

4.3.5.4 unsigned char ColorRecognitionTCS230Pl::s2Pin [private]

The s2 pin.

Definition at line 87 of file ColorRecognitionTCS230Pl.h.

4.3.5.5 unsigned char ColorRecognitionTCS230PI::s3Pin [private]

The s3 pin.

Definition at line 92 of file ColorRecognitionTCS230Pl.h.

The documentation for this class was generated from the following files:

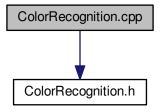
- ColorRecognitionTCS230PI.h
- ColorRecognitionTCS230PI.cpp

5 File Documentation

5.1 ColorRecognition.cpp File Reference

```
#include "ColorRecognition.h"
```

Include dependency graph for ColorRecognition.cpp:



Macros

```
    #define __ARDUINO_DRIVER_COLOR_RECOGNITION_CPP__ 1
```

5.1.1 Macro Definition Documentation

5.1.1.1 #define __ARDUINO_DRIVER_COLOR_RECOGNITION_CPP__ 1

Arduino - Color Recognition Sensor.

ColorRecognition.h

The abstract class for the color recognition sensors.

Author

Dalmir da Silva dalmirdasilva@gmail.com

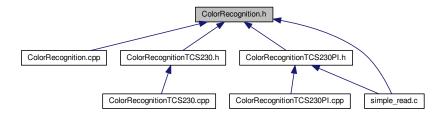
Definition at line 12 of file ColorRecognition.cpp.

5.2 ColorRecognition.cpp

```
00001
00011 #ifndef __ARDUINO_DRIVER_COLOR_RECOGNITION_CPP__
00012 #define __ARDUINO_DRIVER_COLOR_RECOGNITION_CPP__ 1
00013
00014 #include "ColorRecognition.h"
00015
00016 #endif /* __ARDUINO_DRIVER_COLOR_RECOGNITION_CPP__ */
```

5.3 ColorRecognition.h File Reference

This graph shows which files directly or indirectly include this file:



Classes

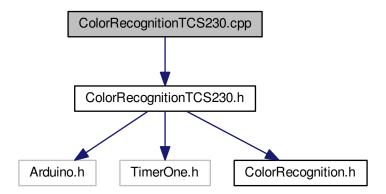
• class ColorRecognition

5.4 ColorRecognition.h

```
00001
00011 #ifndef __ARDUINO_DRIVER_COLOR_RECOGNITION_H_
00012 #define __ARDUINO_DRIVER_COLOR_RECOGNITION_H__ 1
00013
00014 class ColorRecognition {
00015 public:
00016
00022
         virtual unsigned char getRed() = 0;
00023
00029
         virtual unsigned char getGreen() = 0;
00030
00036
         virtual unsigned char getBlue() = 0;
00037
00043
          virtual bool fillRGB(unsigned char buf[3]) = 0;
00044 };
00045
00046 #endif /* __ARDUINO_DRIVER_COLOR_RECOGNITION_H__ */
```

5.5 ColorRecognitionTCS230.cpp File Reference

#include "ColorRecognitionTCS230.h"
Include dependency graph for ColorRecognitionTCS230.cpp:



Macros

#define __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230_CPP__ 1

5.5.1 Macro Definition Documentation

5.5.1.1 #define __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230_CPP__ 1

Arduino - Color Recognition Sensor.

ColorRecognitionTCS230.h

The abstract class for the Color Recognition TCS230 sensor.

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 12 of file ColorRecognitionTCS230.cpp.

5.6 ColorRecognitionTCS230.cpp

```
00001
00011 #ifndef __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230_CPP_
00012 #define __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230_CPP_ 1
00013
00014 #include "ColorRecognitionTCS230.h"
00015
00016 ColorRecognitionTCS230 ColorRecognitionTCS230::instance
00017
00018 void ColorRecognitionTCS230::initialize(unsigned char outPin, unsigned
       char s2Pin, unsigned char s3Pin) {
          this->s2Pin;
this->s2Pin = s2Pin;
this->s3Pin = s3Pin;
this->outPin = outPin;
this->currentFilter = CLEAR_FILTER;
00019
00020
00021
00022
00023
           pinMode(s2Pin, OUTPUT);
```

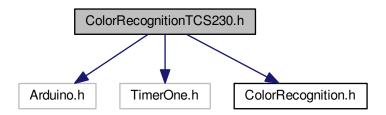
```
00024
          pinMode(s3Pin, OUTPUT);
00025
          pinMode(outPin, INPUT);
00026
          Timer1.initialize();
00027
          Timer1.attachInterrupt(ColorRecognitionTCS230::timerInterruptHandler
     );
00028
          attachInterrupt((outPin - 2),
      ColorRecognitionTCS230::externalInterruptHandler, RISING);
00029 }
00030
00031 void ColorRecognitionTCS230::adjustWhiteBalance() {
00032
          delay(4000);
          instance.whiteBalanceFrequencies[0] =
00033
      instance.lastFrequencies[0];
00034
          instance.whiteBalanceFrequencies[1] =
     instance.lastFrequencies[1];
00035
          instance.whiteBalanceFrequencies[2] =
     instance.lastFrequencies[2];
00036 }
00037
00038 void ColorRecognitionTCS230::externalInterruptHandler() {
00039
         instance.count++;
00040 }
00041
00042 void ColorRecognitionTCS230::timerInterruptHandler() {
00043
         switch (instance.currentFilter) {
          case CLEAR_FILTER:
00044
00045
              setFilter(RED_FILTER);
00046
             break;
00047
          case RED_FILTER:
00048
             instance.lastFrequencies[0] = instance.
     count:
00049
              setFilter(GREEN_FILTER);
00050
00051
          case GREEN_FILTER:
             instance.lastFrequencies[1] = instance.
00052
     count;
00053
              setFilter(BLUE FILTER);
00054
             break;
00055
         case BLUE_FILTER:
00056
             instance.lastFrequencies[2] = instance.
     count;
00057
              setFilter(RED_FILTER);
00058
              break:
00059
00060
          instance.count = 0;
00061
          Timer1.setPeriod(1000000);
00062 }
00063
00064 unsigned char ColorRecognitionTCS230::getRed() {
        if (lastFrequencies[0] > whiteBalanceFrequencies[0]) {
00065
00066
              return 255;
00067
00068
          return (unsigned char) map(lastFrequencies[0], 0,
     whiteBalanceFrequencies[0], 0, 255);
00069 }
00070
00071 unsigned char ColorRecognitionTCS230::getGreen() {
00072
         if (lastFrequencies[1] > whiteBalanceFrequencies[1]) {
00073
              return 255;
00074
          }
00075
          return (unsigned char) map(lastFrequencies[1], 0,
     whiteBalanceFrequencies[1], 0, 255);
00076 }
00077
00078 unsigned char ColorRecognitionTCS230::getBlue() {
00079
         if (lastFrequencies[2] > whiteBalanceFrequencies[2]) {
              return 255:
08000
00081
          return (unsigned char) map(lastFrequencies[2], 0,
00082
      whiteBalanceFrequencies[2], 0, 255);
00083 }
00084
00085 bool ColorRecognitionTCS230::fillRGB(unsigned char buf[3]) {
         buf[0] = getRed();
buf[1] = getGreen();
00086
00087
          buf[2] = getBlue();
00088
00089
          return true;
00090 }
00091
00092 void ColorRecognitionTCS230::setFilter(Filter filter) {
         unsigned char s2 = LOW, s3 = LOW;
instance.currentFilter = filter;
00093
00094
00095
          if (filter == CLEAR_FILTER || filter == GREEN_FILTER) {
00096
              s2 = HIGH;
00097
          if (filter == BLUE_FILTER || filter == GREEN_FILTER) {
00098
00099
              s3 = HIGH;
```

```
00100    }
00101    digitalWrite(instance.s2Pin, s2);
00102    digitalWrite(instance.s3Pin, s3);
00103 }
00104    00105 #endif /* _ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230_CPP__ */
```

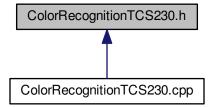
5.7 ColorRecognitionTCS230.h File Reference

```
#include <Arduino.h>
#include <TimerOne.h>
#include <ColorRecognition.h>
```

Include dependency graph for ColorRecognitionTCS230.h:



This graph shows which files directly or indirectly include this file:



Classes

• class ColorRecognitionTCS230

Macros

#define MAX_FRQUENCY_IN_HZ 1000

5.7.1 Macro Definition Documentation

5.7.1.1 #define MAX_FRQUENCY_IN_HZ 1000

Arduino - Color Recognition Sensor.

ColorRecognitionTCS230.h

The abstract class for the Color Recognition TCS230 sensor.

Author

Dalmir da Silva dalmirdasilva@gmail.com In this driver we are assuming the S0 pin is LOW and S1 pin is HIGH. With output frequency at 2%. It saves arduino pins also.

```
S0 S1 OUTPUT FREQUENCY
L L Power down
L H 2%
H L 20%
H H 100%
```

Also we are assuming the OE pin is LOW, this pin controls the device activation. If OE is LOW the device is enable.

Output frequency scaling:

Output-frequency scaling is controlled by two logic inputs, S0 and S1. The internal light-to-frequency converter generates a fixed-pulsewidth pulse train. Scaling is accomplished by internally connecting the pulse-train output of the converter to a series of frequency dividers. Divided outputs are 50%-duty cycle square waves with relative frequency values of 100%, 20%, and 2%. Because division of the output frequency is accomplished by counting pulses of the principal internal frequency, the final-output period represents an average of the multiple periods of the principle frequency.

The output-scaling counter registers are cleared upon the next pulse of the principal frequency after any transition of the S0, S1, S2, S3, and OE lines. The output goes high upon the next subsequent pulse of the principal frequency, beginning a new valid period. This minimizes the time delay between a change on the input lines and the resulting new output period. The response time to an input programming change or to an irradiance step change is one period of new frequency plus 1 μ S. The scaled output changes both the full–scale frequency and the dark frequency by the selected scale factor. The frequency-scaling function allows the output range to be optimized for a variety of measurement techniques. The scaled-down outputs may be used where only a slower frequency counter is available, such as low-cost microcontroller, or where period measurement techniques are used.

Measuring the frequency:

The choice of interface and measurement technique depends on the desired resolution and data acquisition rate. For maximum data-acquisition rate, period-measurement techniques are used. Output data can be collected at a rate of twice the output frequency or one data point every microsecond for full-scale output. Period measurement requires the use of a fast reference clock with available resolution directly related to reference clock rate. Output scaling can be used to increase the resolution for a given clock rate or to maximize resolution as the light input changes. Period measurement is used to measure rapidly varying light levels or to make a very fast measurement of a constant light source. Maximum resolution and accuracy may be obtained using frequency-measurement, pulse-accumulation, or integration techniques. Frequency measurements provide the added benefit of averaging out random- or high-frequency variations (jitter) resulting from noise in the light signal. Resolution is limited mainly by available counter registers and allowable measurement time. Frequency measurement is well suited for slowly varying or constant light levels and for reading average light levels over short periods of time. Integration (the accumulation of pulses over a very long period of time) can be used to measure exposure, the amount of light present in an area over a given time period. When used with a BASIC Stamp, the TCS230's output frequency can be read using the Stamp's statement, as shown in the example code on the front side of this sheet. In this example, and were both pulled "high", enabling the TCS230's fastest output rate. However, this rate can be as much as 600KHz or more at maximum light intensity

MAX: 600KHz (I don't belive) we are usin: 2% of such frequence we are usin: 1 second between the interrupts.

```
(600 * 2 / 100) * 1000 it is too much.
```

Definition at line 94 of file ColorRecognitionTCS230.h.

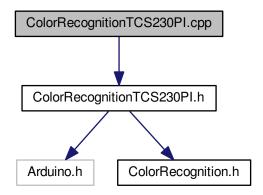
5.8 ColorRecognitionTCS230.h

```
00001
00011 #ifndef __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230_H_
00012 #define __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230_H__ 1
00013
00014 #include <Arduino.h>
00015 #include <TimerOne.h>
00016 #include <ColorRecognition.h>
00017
00094 #define MAX_FRQUENCY_IN_HZ 1000
00095
00096 class ColorRecognitionTCS230 : public ColorRecognition {
00097 private:
00098
00102
          unsigned char s2Pin;
00103
00107
          unsigned char s3Pin;
00108
00114
          unsigned char outPin;
00115
00119
          int count;
00120
00124
          int lastFrequencies[3];
00125
00129
          int whiteBalanceFrequencies[3]:
00130
00134
          static ColorRecognitionTCS230 instance;
00135
00136 public:
00137
00141
          enum Filter {
00142
              RED_FILTER, GREEN_FILTER, BLUE_FILTER,
     CLEAR_FILTER
00143
          };
00144
00148
          Filter currentFilter:
00149
          static ColorRecognitionTCS230* getInstance() {
00155
00156
              return &ColorRecognitionTCS230::instance;
00157
00158
          void initialize (unsigned char outPin, unsigned char s2Pin,
00169
00170
                  unsigned char s3Pin);
00171
00177
          void adjustWhiteBalance();
00178
00184
          unsigned char getRed();
00185
00191
          unsigned char getGreen();
00192
00198
          unsigned char getBlue();
00199
00205
          bool fillRGB(unsigned char buf[3]);
00206
00220
          static void setFilter (Filter filter);
00221
00222 private:
00223
00227
          ColorRecognitionTCS230() {
00228
              whiteBalanceFrequencies[0] = MAX_FRQUENCY_IN_HZ;
              whiteBalanceFrequencies[1] = MAX_FRQUENCY_IN_HZ;
whiteBalanceFrequencies[2] = MAX_FRQUENCY_IN_HZ;
00229
00230
00231
          }
00232
00236
          static void externalInterruptHandler();
00237
00241
          static void timerInterruptHandler();
00242 };
00244 #endif /* __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230_H__ */
```

5.9 ColorRecognitionTCS230Pl.cpp File Reference

#include "ColorRecognitionTCS230PI.h"

Include dependency graph for ColorRecognitionTCS230PI.cpp:



Macros

#define __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230PI_CPP__ 1

5.9.1 Macro Definition Documentation

```
5.9.1.1 #define ARDUINO DRIVER COLOR RECOGNITION TCS230PI CPP 1
```

Arduino - Color Recognition Sensor.

ColorRecognitionTCS230PI.h

The abstract class for the Color Recognition TCS230 sensor.

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 12 of file ColorRecognitionTCS230PI.cpp.

5.10 ColorRecognitionTCS230Pl.cpp

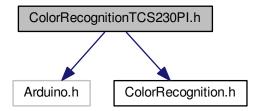
```
00001
00011 #ifndef __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230PI_CPP_
00012 #define __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230PI_CPP__ 1
00013
00014 #include "ColorRecognitionTCS230PI.h"
00015
00016 ColorRecognitionTCS230PI::ColorRecognitionTCS230PI(
      unsigned char outPin,
00017
                unsigned char s2Pin, unsigned char s3Pin) {
           this->s2Pin = s2Pin;
this->s3Pin = s3Pin;
00018
00019
           this->outPin = outPin;
00020
           pinMode(s2Pin, OUTPUT);
00021
           pinMode(s3Pin, OUTPUT);
00022
00023
           pinMode(outPin, INPUT);
00024
           for (unsigned char i = 0; i < 3; i++) {</pre>
                minFrequency[i] = 0;
maxFrequency[i] = 1000;
00025
00026
00027
            }
00028 }
00029
```

```
00030 void ColorRecognitionTCS230PI::adjustWhiteBalance() {
        for (unsigned char i = 0; i < 3; i++) {</pre>
00032
              setFilter((Filter) i);
00033
              maxFrequency[i] = getFrequency(255);
00034
00035 }
00037 void ColorRecognitionTCS230PI::adjustBlackBalance() {
00038
       for (unsigned char i = 0; i < 3; i++) {
00039
              setFilter((Filter) i);
00040
              minFrequency[i] = getFrequency(255);
00041
         }
00042 }
00043
00044 unsigned char ColorRecognitionTCS230PI::getRed() {
       setFilter(RED_FILTER);
00045
          if (lastFrequencies[0] > whiteBalanceFrequencies[0]) {
00046
00047
              return 255;
00049
          return (unsigned char) map(getFrequency(SAMPLES),
     minFrequency[0], maxFrequency[0], 0, 255);
00050 }
00051
00052 unsigned char ColorRecognitionTCS230PI::getGreen() {
00053
          setFilter(GREEN_FILTER);
          if (lastFrequencies[1] > whiteBalanceFrequencies[1]) {
00054
00055
              return 255;
00056
00057
          return (unsigned char) map(getFrequency(SAMPLES),
     minFrequency[1], maxFrequency[1], 0, 255);
00058 }
00059
00060 unsigned char ColorRecognitionTCS230PI::getBlue() {
00061
       setFilter(BLUE_FILTER);
00062
          if (lastFrequencies[2] > whiteBalanceFrequencies[2]) {
00063
              return 255;
00064
00065
          return (unsigned char) map(getFrequency(SAMPLES),
     minFrequency[2], maxFrequency[2], 0, 255);
00066 }
00067
00068 bool ColorRecognitionTCS230PI::fillRGB(unsigned char buf[3]) {
         buf[0] = getRed();
buf[1] = getGreen();
00069
00070
00071
          buf[2] = getBlue();
00072
          return true;
00073 }
00074
00075 void ColorRecognitionTCS230PI::setFilter(
     Filter filter) {
00076
         unsigned char s2 = LOW, s3 = LOW;
00077
          if (filter == CLEAR_FILTER || filter == GREEN_FILTER) {
00078
              s2 = HIGH;
00079
          if (filter == BLUE_FILTER || filter == GREEN_FILTER) {
00080
00081
              s3 = HIGH;
00083
          digitalWrite(s2Pin, s2);
00084
          digitalWrite(s3Pin, s3);
00085 }
00086
00087 long ColorRecognitionTCS230PI::getFrequency(unsigned int samples) {
          long frequency = 0;
for (unsigned int i = 0; i < samples; i++)</pre>
00088
00089
00090
              frequency += 500000 / pulseIn(outPin, HIGH, 250000);
00091
00092
          return frequency / samples;
00093 }
00094
00095 #endif /* __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230PI_CPP__ */
```

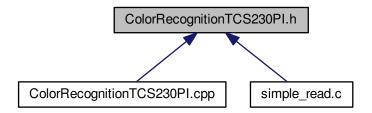
5.11 ColorRecognitionTCS230Pl.h File Reference

```
#include <Arduino.h>
#include <ColorRecognition.h>
```

Include dependency graph for ColorRecognitionTCS230PI.h:



This graph shows which files directly or indirectly include this file:



Classes

• class ColorRecognitionTCS230PI

Macros

• #define SAMPLES 32

5.11.1 Macro Definition Documentation

5.11.1.1 #define SAMPLES 32

Arduino - Color Recognition Sensor.

ColorRecognitionTCS230PI.h

The abstract class for the Color Recognition TCS230 sensor.

Author

Dalmir da Silva dalmirdasilva@gmail.com In this driver we are assuming the S0 pin is LOW and S1 pin is HIGH. With output frequency at 2%. It saves arduino pins also.

```
S0 S1 OUTPUT FREQUENCY
L L Power down
L H 2%
H L 20%
H H 100%
```

Also we are assuming the OE pin is LOW, this pin controls the device activation. If OE is LOW the device is enable.

Output frequency scaling:

Output-frequency scaling is controlled by two logic inputs, S0 and S1. The internal light-to-frequency converter generates a fixed-pulsewidth pulse train. Scaling is accomplished by internally connecting the pulse-train output of the converter to a series of frequency dividers. Divided outputs are 50%-duty cycle square waves with relative frequency values of 100%, 20%, and 2%. Because division of the output frequency is accomplished by counting pulses of the principal internal frequency, the final-output period represents an average of the multiple periods of the principle frequency.

The output-scaling counter registers are cleared upon the next pulse of the principal frequency after any transition of the S0, S1, S2, S3, and OE lines. The output goes high upon the next subsequent pulse of the principal frequency, beginning a new valid period. This minimizes the time delay between a change on the input lines and the resulting new output period. The response time to an input programming change or to an irradiance step change is one period of new frequency plus 1 μ S. The scaled output changes both the full–scale frequency and the dark frequency by the selected scale factor. The frequency-scaling function allows the output range to be optimized for a variety of measurement techniques. The scaled-down outputs may be used where only a slower frequency counter is available, such as low-cost microcontroller, or where period measurement techniques are used.

Measuring the frequency:

The choice of interface and measurement technique depends on the desired resolution and data acquisition rate. For maximum data-acquisition rate, period-measurement techniques are used. Output data can be collected at a rate of twice the output frequency or one data point every microsecond for full-scale output. Period measurement requires the use of a fast reference clock with available resolution directly related to reference clock rate. Output scaling can be used to increase the resolution for a given clock rate or to maximize resolution as the light input changes. Period measurement is used to measure rapidly varying light levels or to make a very fast measurement of a constant light source. Maximum resolution and accuracy may be obtained using frequency-measurement, pulse-accumulation, or integration techniques. Frequency measurements provide the added benefit of averaging out random- or high-frequency variations (jitter) resulting from noise in the light signal. Resolution is limited mainly by available counter registers and allowable measurement time. Frequency measurement is well suited for slowly varying or constant light levels and for reading average light levels over short periods of time. Integration (the accumulation of pulses over a very long period of time) can be used to measure exposure, the amount of light present in an area over a given time period.

Definition at line 79 of file ColorRecognitionTCS230Pl.h.

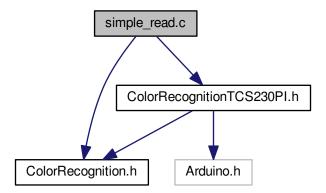
5.12 ColorRecognitionTCS230Pl.h

```
00001
00011 #ifndef __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230_PI_H_
00012 #define __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230_PI_H_
00013
00014 #include <Arduino.h>
00015 #include <ColorRecognition.h>
00016
00079 #define SAMPLES
08000
00081 class ColorRecognitionTCS230PI : public ColorRecognition {
00082 private:
00083
00087
          unsigned char s2Pin:
00088
          unsigned char s3Pin;
00093
00097
          unsigned char outPin;
00098
          long minFrequency[3];
00103
00107
          long maxFrequency[3];
00108
```

```
00109 public:
00110
00114
          enum Filter {
00115
             RED_FILTER, GREEN_FILTER, BLUE_FILTER,
     CLEAR_FILTER
00116
         };
00117
00121
          ColorRecognitionTCS230PI(unsigned char outPin, unsigned char s2Pin,
00122
                 unsigned char s3Pin);
00123
00129
         void adjustWhiteBalance();
00130
00136
          void adjustBlackBalance();
00137
00143
          unsigned char getRed();
00144
          unsigned char getGreen();
00150
00151
00157
          unsigned char getBlue();
00158
00164
          bool fillRGB(unsigned char buf[3]);
00165
00184
          long getFrequency(unsigned int samples);
00185
00199
          void setFilter(Filter filter);
00200
00201 };
00202
00203 #endif /* __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230_PI_H__ */
```

5.13 simple_read.c File Reference

```
#include <ColorRecognition.h>
#include <ColorRecognitionTCS230PI.h>
Include dependency graph for simple_read.c:
```



Functions

- void setup ()
- void loop ()

5.13.1 Function Documentation

5.13.1.1 void loop ()

Definition at line 33 of file simple_read.c.

```
5.13.1.2 void setup ( )
```

Definition at line 4 of file simple_read.c.

5.14 simple_read.c

```
00001 #include <ColorRecognition.h>
00002 #include <ColorRecognitionTCS230PI.h>
00003
00004 void setup() {
00005
00006
        Serial.begin(9600);
00007
        ColorRecognitionTCS230PI tcs230(2, 3, 4);
80000
00009
        Serial.println("Adjust white color, show something white to the sensor and press y."); while (!Serial.available() && Serial.read() !='y');
00010
00011
00012
         Serial.read();
00013
         Serial.println("Adjusting...");
00014
         tcs230.adjustWhiteBalance();
00015
00016
        Serial.println("Adjust black color, show something black to the sensor and press y."); while (!Serial.available() && Serial.read() != 'y');
00017
00018
         Serial.read();
00019
         Serial.println("Adjusting...");
00020
         tcs230.adjustBlackBalance();
00021
         while (1) {
00022
00023
          Serial.print("Read: ");
           Serial.println(tcs230.getRed());
00025
           Serial.print("Green: ");
00026
           Serial.println(tcs230.getGreen());
00027
           Serial.print("Blue ");
           Serial.println(tcs230.getBlue());
00028
00029
           delay(3000);
00030
00031 }
00032
00033 void loop() {
00034 }
```

Index

ARDUINO_DRIVER_COLOR_RECOGNITION_CP	whiteBalanceFrequencies, 8
P	ColorRecognitionTCS230.cpp, 14
ColorRecognition.cpp, 12	ARDUINO_DRIVER_COLOR_RECOGNITIO↔
ARDUINO_DRIVER_COLOR_RECOGNITION_TC	N_TCS230_CPP, 14
S230PI_CPP	ColorRecognitionTCS230.h, 16, 18
ColorRecognitionTCS230PI.cpp, 19	MAX_FRQUENCY_IN_HZ, 16
ARDUINO_DRIVER_COLOR_RECOGNITION_TC	ColorRecognitionTCS230PI, 8
S230 CPP	adjustBlackBalance, 9
ColorRecognitionTCS230.cpp, 14	adjustWhiteBalance, 9
	BLUE_FILTER, 9
adjustBlackBalance	CLEAR FILTER, 9
ColorRecognitionTCS230PI, 9	ColorRecognitionTCS230PI, 9
adjustWhiteBalance	fillRGB, 10
ColorRecognitionTCS230, 5	Filter, 9
ColorRecognitionTCS230PI, 9	GREEN_FILTER, 9
	getBlue, 10
BLUE_FILTER	getFrequency, 10
ColorRecognitionTCS230, 5	getGreen, 10
ColorRecognitionTCS230PI, 9	getRed, 10
	maxFrequency, 11
CLEAR_FILTER	minFrequency, 11
ColorRecognitionTCS230, 5	outPin, 11
ColorRecognitionTCS230PI, 9	RED_FILTER, 9
ColorRecognition, 2	s2Pin, 11
fillRGB, 2	s3Pin, 11
getBlue, 2	
getGreen, 3	setFilter, 10
getRed, 3	ColorRecognitionTCS230Pl.cpp, 18, 19
ColorRecognition.cpp, 11, 12	ARDUINO_DRIVER_COLOR_RECOGNITIO
ARDUINO_DRIVER_COLOR_RECOGNITIO	N_TCS230PI_CPP, 19
N_CPP, 12	ColorRecognitionTCS230Pl.h, 20, 22
ColorRecognition.h, 13	SAMPLES, 21
ColorRecognitionTCS230, 3	Color Paga annikian TOCOCO 7
adjustWhiteBalance, 5	ColorRecognitionTCS230, 7
BLUE FILTER, 5	currentFilter
CLEAR_FILTER, 5	ColorRecognitionTCS230, 7
ColorRecognitionTCS230, 5	avtarnallatarrunti landlar
count, 7	externalInterruptHandler
currentFilter, 7	ColorRecognitionTCS230, 5
externalInterruptHandler, 5	fillRGB
fillRGB, 5	ColorRecognition, 2
Filter, 5	ColorRecognitionTCS230, 5
GREEN_FILTER, 5	,
getBlue, 6	ColorRecognitionTCS230PI, 10
getGreen, 6	Filter
getInstance, 6	ColorRecognitionTCS230, 5
getRed, 6	ColorRecognitionTCS230PI, 9
initialize, 6	GREEN FILTER
instance, 7	ColorRecognitionTCS230, 5
lastFrequencies, 7	ColorRecognitionTCS230PI, 9
outPin, 7	getBlue
RED_FILTER, 5	ColorRecognition, 2
s2Pin, 7	ColorRecognitionTCS230, 6
s3Pin, 7	ColorRecognitionTCS230PI, 10
setFilter, 6	getFrequency
timerInterruptHandler, 7	ColorRecognitionTCS230PI, 10

26 INDEX

```
getGreen
    ColorRecognition, 3
    ColorRecognitionTCS230, 6
    ColorRecognitionTCS230PI, 10
getInstance
    ColorRecognitionTCS230, 6
getRed
    ColorRecognition, 3
    ColorRecognitionTCS230, 6
    ColorRecognitionTCS230PI, 10
initialize
    ColorRecognitionTCS230, 6
instance
    ColorRecognitionTCS230, 7
lastFrequencies
    ColorRecognitionTCS230, 7
loop
    simple_read.c, 23
MAX FRQUENCY IN HZ
    ColorRecognitionTCS230.h, 16
maxFrequency
    ColorRecognitionTCS230PI, 11
minFrequency
    ColorRecognitionTCS230PI, 11
outPin
    ColorRecognitionTCS230, 7
    ColorRecognitionTCS230PI, 11
RED_FILTER
    ColorRecognitionTCS230, 5
    ColorRecognitionTCS230PI, 9
s2Pin
    ColorRecognitionTCS230, 7
    ColorRecognitionTCS230PI, 11
s3Pin
     ColorRecognitionTCS230, 7
    ColorRecognitionTCS230PI, 11
SAMPLES
    ColorRecognitionTCS230PI.h, 21
setFilter
    ColorRecognitionTCS230, 6
    ColorRecognitionTCS230PI, 10
setup
    simple_read.c, 23
simple_read.c, 23, 24
    loop, 23
    setup, 23
timerInterruptHandler
    ColorRecognitionTCS230, 7
whiteBalanceFrequencies
    ColorRecognitionTCS230, 8
```