Arduino Gyroscope Driver

Generated by Doxygen 1.8.9.1

Tue Aug 18 2015 22:52:00

Contents

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

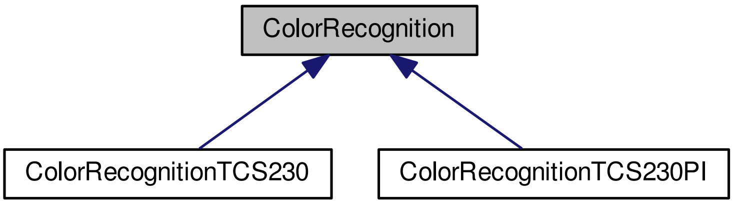
ColorRecognitionTCS230PI.cpp 18

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 ColorRecognitionTCS230, and ColorRecognitionTCS230PI.

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.

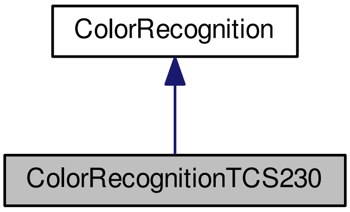
The documentation for this class was generated from the following file:

- ColorRecognition.h

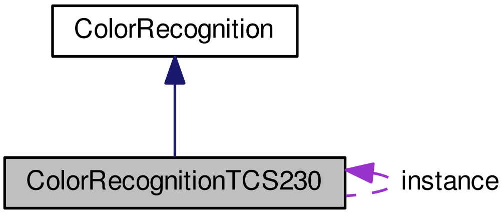
4.2 ColorRecognitionTCS230 Class Reference

```
#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.

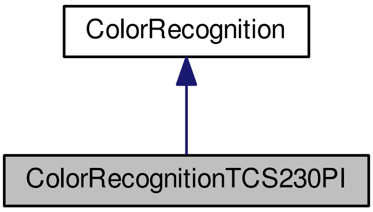
The documentation for this class was generated from the following files:

- ColorRecognitionTCS230.h
- ColorRecognitionTCS230.cpp

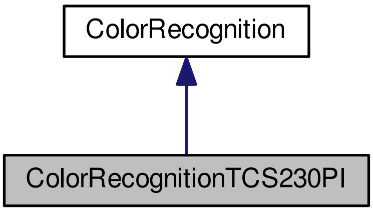
4.3 ColorRecognitionTCS230PI Class Reference

```
#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 ColorRecognitionTCS230PI.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 ColorRecognitionTCS230PI.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 ColorRecognitionTCS230PI.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 pulseIn, 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 ColorRecognitionTCS230PI::getRed ( ) [virtual]

Returns the red color intensity.

The red color intensity.

Implements ColorRecognition.

Definition at line 44 of file ColorRecognitionTCS230PI.cpp.

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 ColorRecognitionTCS230PI.cpp.

4.3.5 Member Data Documentation

4.3.5.1 long ColorRecognitionTCS230PI::maxFrequency[3] [private]

The maximum frequency.

Definition at line 107 of file ColorRecognitionTCS230PI.h.

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

The minimum frequency.

Definition at line 102 of file ColorRecognitionTCS230PI.h.

4.3.5.3 unsigned char ColorRecognitionTCS230PI::outPin [private]

The out pin.

Definition at line 97 of file ColorRecognitionTCS230PI.h.

4.3.5.4 unsigned char ColorRecognitionTCS230PI::s2Pin [private]

The s2 pin.

Definition at line 87 of file ColorRecognitionTCS230PI.h.

4.3.5.5 unsigned char ColorRecognitionTCS230PI::s3Pin [private]

The s3 pin.

Definition at line 92 of file ColorRecognitionTCS230PI.h.

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

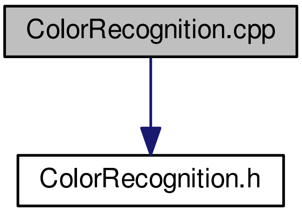
- ColorRecognitionTCS230PI.h
- ColorRecognitionTCS230PI.cpp

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`

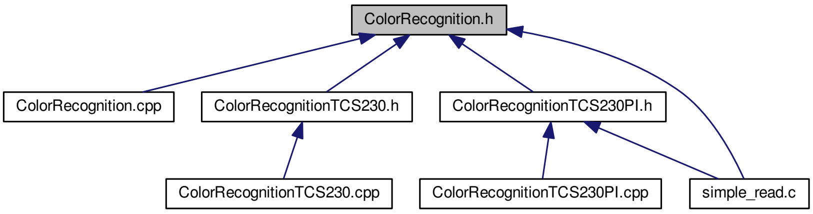
Definition at line 12 of file ColorRecognition.cpp.

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:


ColorRecognition.h
ColorRecognition.cpp
ColorRecognitionTCS230.h
ColorRecognitionTCS230PI.h
ColorRecognitionTCS230.cpp
ColorRecognitionTCS230PI.cpp
simple_read.c

Classes

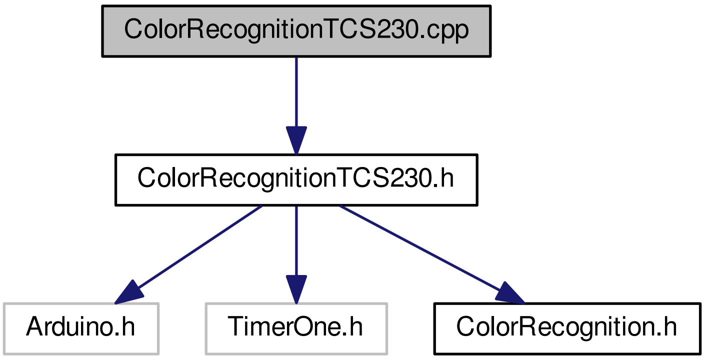
- class ColorRecognition

5.4 ColorRecognition.h

```

#ifndef __ARDUINO_DRIVER_COLOR_RECOGNITION_H__
#define __ARDUINO_DRIVER_COLOR_RECOGNITION_H__ 1

class ColorRecognition {
public:

    virtual unsigned char getRed() = 0;

    virtual unsigned char getGreen() = 0;

    virtual unsigned char getBlue() = 0;

    virtual bool fillRGB(unsigned char buf[3]) = 0;
};

#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

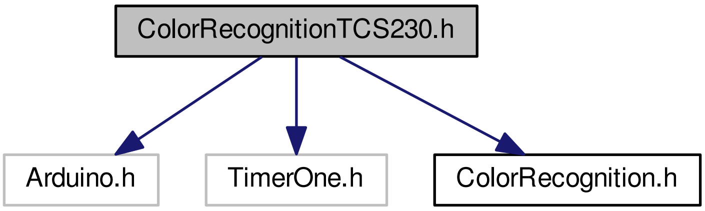
```
00001
00011 #ifndef __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230_CPP__
00012 #define __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230_CPP__ 1
00013
00014 #include "ColorRecognitionTCS230.h"
00015
00016 ColorRecognitionTCS230 ColorRecognitionTCS230::instance
      ;
00017
00018 void ColorRecognitionTCS230::initialize(unsigned char outPin, unsigned
      char s2Pin, unsigned char s3Pin) {
00019     this->s2Pin = s2Pin;
00020     this->s3Pin = s3Pin;
00021     this->outPin = outPin;
00022     this->currentFilter = CLEAR_FILTER;
00023     pinMode(s2Pin, OUTPUT);
```

```
    pinMode(s3Pin, OUTPUT);
    pinMode(outPin, INPUT);
    Timer1.initialize();
    Timer1.attachInterrupt(ColorRecognitionTCS230::timerInterruptHandler);
    attachInterrupt((outPin - 2), ColorRecognitionTCS230::externalInterruptHandler, RISING);
}

void ColorRecognitionTCS230::adjustWhiteBalance() {
    delay(4000);
    instance.whiteBalanceFrequencies[0] = instance.lastFrequencies[0];
    instance.whiteBalanceFrequencies[1] = instance.lastFrequencies[1];
    instance.whiteBalanceFrequencies[2] = instance.lastFrequencies[2];
}

void ColorRecognitionTCS230::externalInterruptHandler() {
    instance.count++;
}

void ColorRecognitionTCS230::timerInterruptHandler() {
    switch (instance.currentFilter) {
    case CLEAR_FILTER:
        setFilter(RED_FILTER);
        break;
    case RED_FILTER:
        instance.lastFrequencies[0] = instance.count;
        setFilter(GREEN_FILTER);
        break;
    case GREEN_FILTER:
        instance.lastFrequencies[1] = instance.count;
        setFilter(BLUE_FILTER);
        break;
    case BLUE_FILTER:
        instance.lastFrequencies[2] = instance.count;
        setFilter(RED_FILTER);
        break;
    }
    instance.count = 0;
    Timer1.setPeriod(1000000);
}

unsigned char ColorRecognitionTCS230::getRed() {
    if (lastFrequencies[0] > whiteBalanceFrequencies[0]) {
        return 255;
    }
    return (unsigned char) map(lastFrequencies[0], 0, whiteBalanceFrequencies[0], 0, 255);
}

unsigned char ColorRecognitionTCS230::getGreen() {
    if (lastFrequencies[1] > whiteBalanceFrequencies[1]) {
        return 255;
    }
    return (unsigned char) map(lastFrequencies[1], 0, whiteBalanceFrequencies[1], 0, 255);
}

unsigned char ColorRecognitionTCS230::getBlue() {
    if (lastFrequencies[2] > whiteBalanceFrequencies[2]) {
        return 255;
    }
    return (unsigned char) map(lastFrequencies[2], 0, whiteBalanceFrequencies[2], 0, 255);
}

bool ColorRecognitionTCS230::fillRGB(unsigned char buf[3]) {
    buf[0] = getRed();
    buf[1] = getGreen();
    buf[2] = getBlue();
    return true;
}

void ColorRecognitionTCS230::setFilter(Filter filter) {
    unsigned char s2 = LOW, s3 = LOW;
    instance.currentFilter = filter;
    if (filter == CLEAR_FILTER || filter == GREEN_FILTER) {
        s2 = HIGH;
    }
    if (filter == BLUE_FILTER || filter == GREEN_FILTER) {
        s3 = HIGH;
    }
    digitalWrite(instance.s2Pin, s2);
    digitalWrite(instance.s3Pin, s3);
}

#endif /* __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230_CPP__ */
```

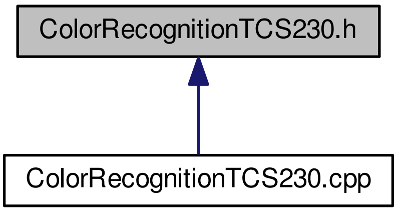
5.7 ColorRecognitionTCS230.h File Reference

```
#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 µS. 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.

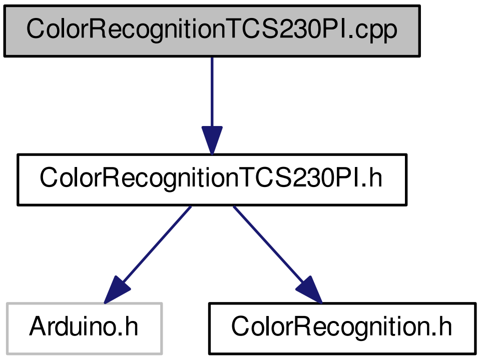
5.8 ColorRecognitionTCS230.h

```
#ifndef __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230_H__
#define __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230_H__ 1

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

#define MAX_FRQUENCY_IN_HZ 1000

class ColorRecognitionTCS230 : public ColorRecognition {
private:

    unsigned char s2Pin;

    unsigned char s3Pin;

    unsigned char outPin;

    int count;

    int lastFrequencies[3];

    int whiteBalanceFrequencies[3];

    static ColorRecognitionTCS230 instance;

public:

    enum Filter {
        RED_FILTER, GREEN_FILTER, BLUE_FILTER, CLEAR_FILTER
    };

    Filter currentFilter;

    static ColorRecognitionTCS230* getInstance() {
        return &ColorRecognitionTCS230::instance;
    }

    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 void setFilter(Filter filter);

private:

    ColorRecognitionTCS230() {
        whiteBalanceFrequencies[0] = MAX_FRQUENCY_IN_HZ;
        whiteBalanceFrequencies[1] = MAX_FRQUENCY_IN_HZ;
        whiteBalanceFrequencies[2] = MAX_FRQUENCY_IN_HZ;
    }

    static void externalInterruptHandler();

    static void timerInterruptHandler();
};

#endif /* __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230_H__ */
```

5.9 ColorRecognitionTCS230PI.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 ColorRecognitionTCS230PI.cpp

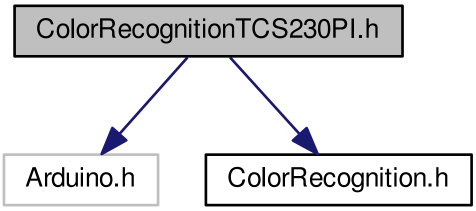
```
00001
00011 #ifndef __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230PI_CPP__
00012 #define __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230PI_CPP__ 1
00013
00014 #include "ColorRecognitionTCS230PI.h"
00015
00016 ColorRecognitionTCS230PI::ColorRecognitionTCS230PI(
      unsigned char outPin,
00017         unsigned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

```
void ColorRecognitionTCS230PI::adjustWhiteBalance() {
    for (unsigned char i = 0; i < 3; i++) {
        setFilter((Filter) i);
        maxFrequency[i] = getFrequency(255);
    }
}

void ColorRecognitionTCS230PI::adjustBlackBalance() {
    for (unsigned char i = 0; i < 3; i++) {
        setFilter((Filter) i);
        minFrequency[i] = getFrequency(255);
    }
}

unsigned char ColorRecognitionTCS230PI::getRed() {
    setFilter(RED_FILTER);
    if (lastFrequencies[0] > whiteBalanceFrequencies[0]) {
        return 255;
    }
    return (unsigned char) map(getFrequency(SAMPLES), minFrequency[0], maxFrequency[0], 0, 255);
}

unsigned char ColorRecognitionTCS230PI::getGreen() {
    setFilter(GREEN_FILTER);
    if (lastFrequencies[1] > whiteBalanceFrequencies[1]) {
        return 255;
    }
    return (unsigned char) map(getFrequency(SAMPLES), minFrequency[1], maxFrequency[1], 0, 255);
}

unsigned char ColorRecognitionTCS230PI::getBlue() {
    setFilter(BLUE_FILTER);
    if (lastFrequencies[2] > whiteBalanceFrequencies[2]) {
        return 255;
    }
    return (unsigned char) map(getFrequency(SAMPLES), minFrequency[2], maxFrequency[2], 0, 255);
}

bool ColorRecognitionTCS230PI::fillRGB(unsigned char buf[3]) {
    buf[0] = getRed();
    buf[1] = getGreen();
    buf[2] = getBlue();
    return true;
}

void ColorRecognitionTCS230PI::setFilter(Filter filter) {
    unsigned char s2 = LOW, s3 = LOW;
    if (filter == CLEAR_FILTER || filter == GREEN_FILTER) {
        s2 = HIGH;
    }
    if (filter == BLUE_FILTER || filter == GREEN_FILTER) {
        s3 = HIGH;
    }
    digitalWrite(s2Pin, s2);
    digitalWrite(s3Pin, s3);
}

long ColorRecognitionTCS230PI::getFrequency(unsigned int samples) {
    long frequency = 0;
    for (unsigned int i = 0; i < samples; i++) {
        frequency += 500000 / pulseIn(outPin, HIGH, 250000);
    }
    return frequency / samples;
}

#endif /* __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230PI_CPP__ */
```

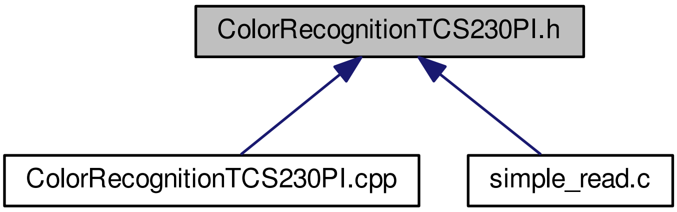
5.11 ColorRecognitionTCS230PI.h File Reference

```
#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 µS. 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 ColorRecognitionTCS230PI.h.

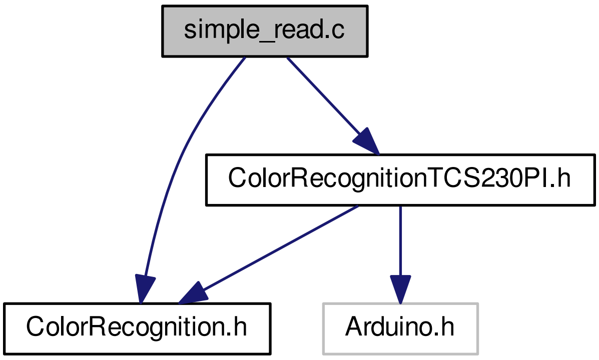
5.12 ColorRecognitionTCS230PI.h

```
00001
00011 #ifndef __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230_PI_H__
00012 #define __ARDUINO_DRIVER_COLOR_RECOGNITION_TCS230_PI_H__ 1
00013
00014 #include <Arduino.h>
00015 #include <ColorRecognition.h>
00016
00079 #define SAMPLES   32
00080
00081 class ColorRecognitionTCS230PI : public ColorRecognition {
00082 private:
00083
00087     unsigned char s2Pin;
00088
00092     unsigned char s3Pin;
00093
00097     unsigned char outPin;
00098
00102     long minFrequency[3];
00103
00107     long maxFrequency[3];
00108
public:

    enum Filter {
        RED_FILTER, GREEN_FILTER, BLUE_FILTER,
CLEAR_FILTER
    };

    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);

};

#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
00008   ColorRecognitionTCS230PI tcs230(2, 3, 4);
00009
00010   Serial.println("Adjust white color, show something white to the sensor and press y.");
00011   while(!Serial.available() && Serial.read() != ’y’);
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.");
00017   while(!Serial.available() && Serial.read() != ’y’);
00018   Serial.read();
00019   Serial.println("Adjusting...");
00020   tcs230.adjustBlackBalance();
00021
00022   while (1) {
00023     Serial.print("Read: ");
00024     Serial.println(tcs230.getRed());
00025     Serial.print("Green: ");
00026     Serial.println(tcs230.getGreen());
00027     Serial.print("Blue ");
00028     Serial.println(tcs230.getBlue());
00029     delay(3000);
00030   }
00031 }
00032
00033 void loop() {
00034 }
```

Index