MÔN: HỆ THỐNG NHÚNG

TÀI LIỆU HƯỚNG DẪN CHO DRIVER CẢM BIẾN TCS34725

GVHD: TS. Bùi Hà Đức

Tên: Nguyễn Trung Nhân - 211460333

Trần Trương Huy Hoàng - 21146389

1. Overview of sensors TCS34725

The TCS34725 sensor is a type of color sensor used in industrial and consumer applications to detect and measure the color of light based on the three primary colors: red, green, and blue. Below are the specifications of the sensor.

1.1 Structure and Operating Principle

Sensor Structure: The TCS34725 consists of an array of photodiodes and an RGB (Red, Green, Blue) and Clear color filter.

Operating Principle: When light enters the sensor, it is separated into different color components by the color filter. Corresponding photodiodes measure the intensity of each color component (red, green, blue, and clear). The signals collected from the photodiodes are converted into digital values through an Analog-to-Digital Converter (ADC).

1.2 Specifications and applications

1.2.1 Technical Specifications

Resolution: 16-bit

Dynamic Range: High, with low signal-to noise ratio

Spectral Range: 400 nm to 700 nm

Interface: I2C (address 0x29)

Power Supply: Operates at a voltage range of 2.7V to 3.6V

1.2.2 Applications

Consumer Electronics: Screen brightness adjustment and white balance calibration.

Industrial Automation: Product color inspection, sorting, and quality control.

Agriculture: Measuring the color index of crops to assess growth conditions

Healthcare: Analyzing the color of medical samples

1.3 Advantages and Disadvantages

1.3.1 Advantages

High sensitivity and accuracy
Compact size, easy to integrate into systems.
I2C protocol allows easy communication with microcontrollers.

1.3.2 Disadvantages

Susceptible to ambient light interference if not properly shielded.

2. Requirements when connecting Raspberry to sensor using I2C protocol

A mandatory requirement when connecting the Raspberry Pi to the sensor via the I2C protocol is to use the I2C-1 interface (SDA: pin 3, SCL: pin 5).

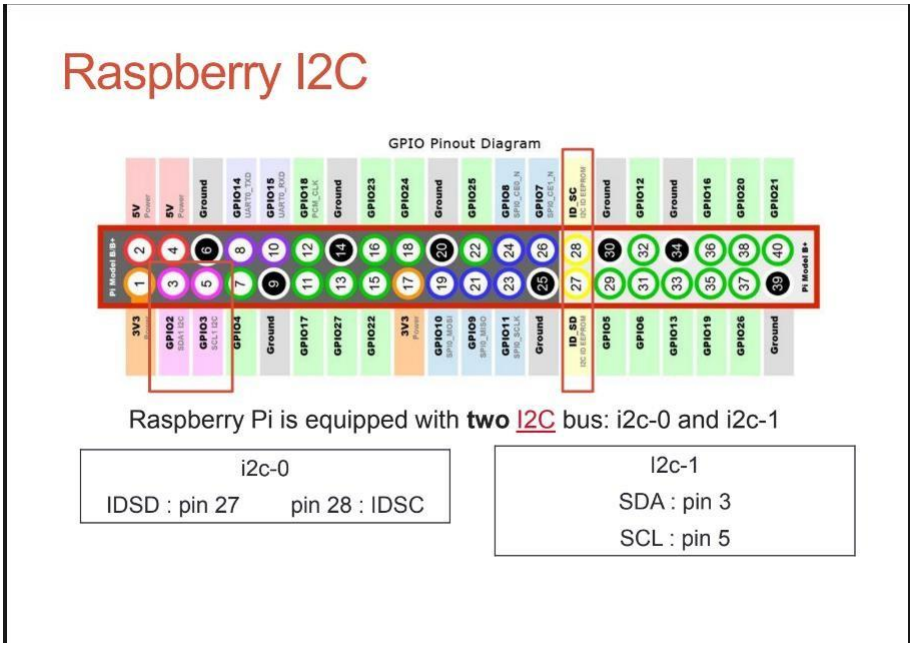

Figure 1: Raspberry Pi Pinout Diagram

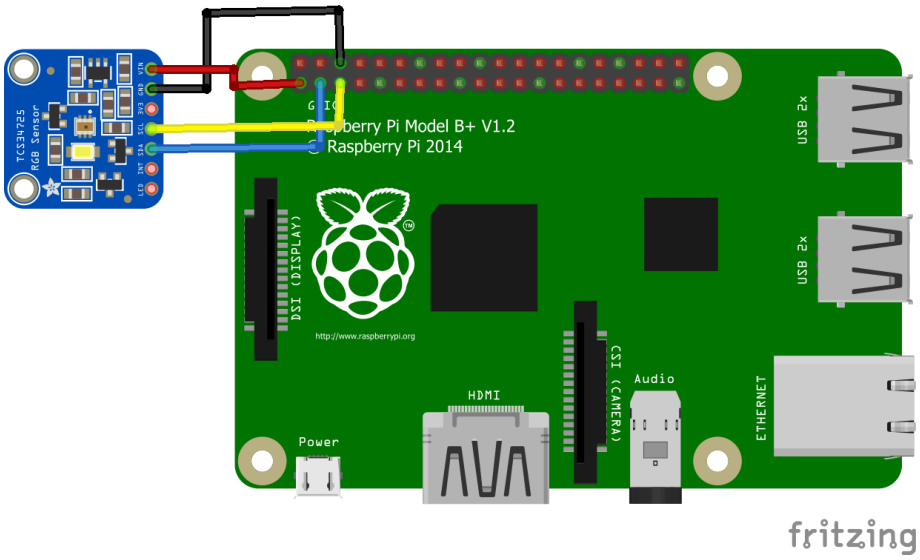

Figure 2: Sensor Connection Diagram with Raspberry Pi

The I2C address of the sensor can be found in the datasheet or by using the following commands:

sudo i2cdetect -y 1 hoặc *gpio i2cdetect*

3. Install Driver for RaspberryPi to use TCS34725 sensor

To install the "tcs34725_driver" for the Raspberry Pi, follow these steps:

Step 1: Navigate to the "/boot" directory of the Raspberry Pi.

Step 2: Convert the Raspberry Pi device tree blob (.dtb) file to a device tree source (.dts) file using the following command:

"dtc -I dts -O dtb -o your_liscence_raspPi.dtb your_liscence_raspPi.dts"

Note: At this step, it is important to select the correct Raspberry Pi model to ensure compatibility with the corresponding .dtb file.
Example: Using raspberry Pi 3 with the following command:
"dtc -I dtb -O dts -o bcm2710-rpi-3-b.dts bcm2710-rpi-3-b.dtb"
Note: Replace bcm2710-rpi-3-b.dtb with the appropriate .dtb file for your specific Raspberry Pi model.

Step 3: To proceed with Step 3, after opening the .dts file created in Step 2, you need to search for the section where the I2C-1 is defined. Once found, add the necessary code snippet to enable the I2C-1 interface:

tcs34725@29{

compatible = "taos, tcs34725";

reg = <0x29>;}

Step 4: After editing the file, save the changes and recompile the .dts file back into .dtb.

Finally, reboot the Raspberry Pi for the changes to take effect

"dtc -I dts -O dtb -o your_liscence_raspPi.dtb your_liscence_raspPi.dts"

Example: Using raspberry Pi 3 with the following command:

"dtc -I dts -O dtb -o bcm2710-rpi-3-b.dtb bcm2710-rpi-3-b.dts"

"sduo reboot"

Step 5: Makefile and install Driver into system

- Ensure that the "driver" file and the Makefile are in the same folder. Then run the make command.

“make”

- Once the make command completes successfully, the file driver.ko will be generated.
- The next step is to install the driver into the Raspberry Pi using the following command:” sudo insmod driver.ko”. You can check the installation status using the dmesg command “dmesg”.
- Similarly, if you want to remove the installed driver, use the following command: “sudo rmmod driver”.

4. Implementing I2C device drivers

4.1 Extra client data

Each client structure has a special data field that can point to any structure at all. You should use this to keep device-specific data.

```
/* store the value */
void i2c_set_clientdata(struct i2c_client *client, void *data);

/* retrieve the value */
void *i2c_get_clientdata(const struct i2c_client *client);
```

4.2 Accessing the client

Let's say we have a valid client structure. At some time, we will need to gather information from the client, or write new information to the client.

I have found it useful to define foo_read and foo_write functions for this. For some cases, it will be easier to call the I2C functions directly, but many chips have some kind of register-value idea that can easily be encapsulated.

The below functions are simple examples, and should not be copied literally:

```
int foo_read_value(struct i2c_client *client, u8 reg)
{
    if (reg < 0x10) /* byte-sized register */
        return i2c_smbus_read_byte_data(client, reg);
    else        /* word-sized register */
        return i2c_smbus_read_word_data(client, reg);
}

int foo_write_value(struct i2c_client *client, u8 reg, u16 value)
{
    if (reg == 0x10)       /* Impossible to write - driver error! */
        return -EINVAL;
    else if (reg < 0x10)   /* byte-sized register */
        return i2c_smbus_write_byte_data(client, reg, value);
    else                   /* word-sized register */
        return i2c_smbus_write_word_data(client, reg, value);
}
```

4.3 Devices/Driver Binding

I2C device drivers using this binding model work just like any other kind of driver in Linux: they provide a probe() method to bind to those devices, and a remove() method to unbind.

```
"static int foo_probe(struct i2c_client *client);
static void foo_remove(struct i2c_client *client);"
```

Remember that the i2c_driver does not create those client handles. The handle may be used during foo_probe(). If foo_probe() reports success (zero not a negative status code) it may

save the handle and use it until foo_remove() returns. That binding model is used by most Linux drivers.
The probe function is called when an entry in the id_table name field matches the device's name. If the probe function needs that entry, it can retrieve it using
*"const struct i2c_device_id *id = i2c_match_id(foo_idtable, client);"*

5. Installation and Usage of the TCS34725_lib Library for Reading the Sensor TCS34725

5.1 Install library TCS34725_lib

To complement the driver, the team has developed a dedicated library for the TCS34725 sensor. The "tcs34725_lib" library provides users with functions to configure the sensor and retrieve values from the four color channels: clear, red, green, and blue. To install and use the library, follow these steps:

Step 1: Make sure that your "code_lib.c" and "tcs34725_library.h" files are in the same folder.

Step 2: Proceed with installing the library by following these steps:

- Compile the file into position-independent code:

"gcc -c -fPIC code_lib.c -o code.o"

- Create the shared library:

"gcc -shared -o libtcs34725_library.so code.o"

- Copy file *tcs34725_library.h* to *usr/include* and *tcs34725_library.so* to *usr/lib*

- Link your program with the library

"gcc file.c -L -ltcs34725_library -o newfile"

By completing the above steps, you have successfully installed the "tcs34725_lib" library on the Raspberry Pi. Details about the library are provided in section 4.2.

6. How to use this library TCS34725_lib

The TCS34725 sensor has multiple registers, each serving a different purpose. Configuring the parameters of each register depends on the user's specific needs. The library provides functions that allow users to easily interact with the sensor's registers.

6.1.1 Turn on Sensor

turn_On_Sensor();

Function: Enable the sensor and establish I2C communication.
Note: This is the first function that must be called to work with the sensor.

6.1.2 ENABLE Register (0x00)

Init_Enable(value);

Function: This register is used to activate the sensor and its features, including powering the sensor, enabling the RGB light source, and enabling interrupts.

The value parameter refers to the data written to the ENABLE register. Users can refer to the following table for specific values::

7	6	5	4	3	2	1	0
Reserved			AIEN	WEN	Reserved	AEN	PON

Fields	Bits	Description
Reserved	7:5	Reserved. Write as 0.
AIEN	4	RGBC interrupt enable. When asserted, permits RGBC interrupts to be generated.
WEN	3	Wait enable. This bit activates the wait feature. Writing a 1 activates the wait timer. Writing a 0 disables the wait timer.
Reserved	2	Reserved. Write as 0.
AEN	1	RGBC enable. This bit actives the two-channel ADC. Writing a 1 activates the RGBC. Writing a 0 disables the RGBC.
PON[1], [2]	0	Power ON. This bit activates the internal oscillator to permit the timers and ADC channels to operate. Writing a 1 activates the oscillator. Writing a 0 disables the oscillator.

Figure 3: ENABLE Register

6.1.3 ATIME Register (0x01)

Init_Atime(value);

Function: Sets the integration time, which is the duration the sensor accumulates (measures) light before outputting a result. The value is written to the register, and users can refer to the table for valid settings:

The library provides predefined value options for easy configuration.

value	Thời gian
atime_2phay4ms	2,4 ms
atime_24ms	24 ms
atime_101ms	101 ms
atime_154ms	154 ms
atime_700ms	700 ms

Bảng 1: Giá trị truyền xuống thanh ghi ATIME

6.1.4 WTIME Register (0x03)

Init_Wtiem(value);

Function: Sets the wait time between measurements, helping to reduce power consumption when continuous operation is not needed. The value is written to the register, and users can refer to the table for valid settings:

The library provides predefined value options for this register as well.

value	Thời gian
wtime_2phay4ms	2,4 ms
wtime_204ms	204 ms
wtime_614ms	614 ms

Bảng 2: Giá trị truyền xuống thanh ghi WTIME

6.1.5 AILT and AIHT Registers (0x04 - 0x07)

Init_AiltL();

Init_Ailth();

Init_Aihtl();

Init_Aihth();

Function: Set the low and high thresholds for ambient light interrupts. If the measured light intensity falls outside these thresholds, the sensor will generate an interrupt signal.

6.1.6 PERS Register (0x0C)

Init_Pers(value);

Function: Sets the persistence threshold to determine how many consecutive times a condition must be met before an interrupt is triggered. The value is written to the register.

The library provides predefined value options for this register.

value	Giá trị
pers_every	Toàn bộ RGBC
pers_1	1 clear
pers_2	2 clear
pers_3	3 clear
pers_5	5 clear
pers_10	10 clear
pers_15	15 clear
pers_20	20 clear
pers_25	25 clear
pers_30	30 clear
pers_35	35 clear
pers_40	40 clear
pers_45	45 clear
pers_50	50 clear
pers_55	55 clear
pers_60	60 clear

Bảng 3: Value passed down to PERS register

6.1.7 CONFIGURATION Register (0x0D)

Init_Config(value);

Function: Contains additional configuration settings, such as low-power wait mode and fast wait mode. The value is written to the register, and users can refer to the provided table.

The library provides predefined value options for easy configuration

value	Trạng thái
config_on	ON
config_off	OFF

Bảng 4: Value passed down to CONFIGURATON register

6.1.8 Thanh ghi CONTROL (0x0F)

Init_Control(value);

Function: Adjusts the sensor's gain to change the measurement sensitivity. Users can refer to the table for available gain settings.

The library provides predefined value options for this register

value	chế độ
gain_x1	x1
gain_x4	X4
gain_x16	x16
gain_x60	X60

Bảng 5: Value passed down to CONTROL register

6.1.9 Thanh ghi Clear, Red, Green, Blue

Read_CLEAR_data(); Read_RED_data(); Read_GREEN_data(); Read_BLUE_data();

Function: four functions in turn with four functions.

- Returns the measured light intensity that passes through the IR filter..
- Returns the red light intensity measurement..
- Returns the green light intensity measurement.
- Returns the blue light intensity measurement.