PiStorms in MaRTe OS for RPI.

Lego Mindstorms' sensors and actuators handlers for Raspberry Pi and MaRTE OS

AUTHOR

Carlos Ayerbe González, Mario Aldea Rivas, Héctor Pérez Tijero

Version V.1

02/06/2017

Introduction and Estructure

Authors:

Carlos Ayerbe González, Mario Aldea Rivas, Héctor Pérez Tijero

Introduction

The main objective of the project is to provide an environment for the development of Lego mindstorms robots that use the computer Raspberry Pi and MaRTE OS as the operating system. The main contribution of the project will be the implementation of Lego Mindstorms driver handlers for Raspberry Pi and MaRTE OS.

Components of the development environment:

Operating system MaRTE OS for Raspberry Pi.
Cross-Development Environment Linux â†’ Raspberry Pi / MaRTE OS.
Lego Mindstorm / Raspberry Pi Adapter
EV3 sensor drivers.
EV3 actuator drivers(motors).
Other drivers (camera, ...)

Cross-Development Environment Linux â†’ Raspberry Pi / MaRTE OS.

The cross-development environment consists of:

1. Compilers GCC and GNAT in their cross-version to use as an x86 / Linux host and target an ARM1176JZF-S processor (Raspberry Pi processor).
2. Gdb debugger.

Lego Mindstorm / Raspberry Pi Adapter

There are commercial "bricks" that allow to connect the sensors and motors of Lego Mindstorm with the Raspberry Pi. The two most important adapters that have been located: To do this project we work with PiStorms adapter.

1. PiStorms `PiStorms Link`.
2. BrickPi `BrickPi Link`.

EV3 sensor drivers

Lego Mindstorms version EV3 sensors that have been developed are:

1. Ultrasonic Sensor (It can measure de distance).

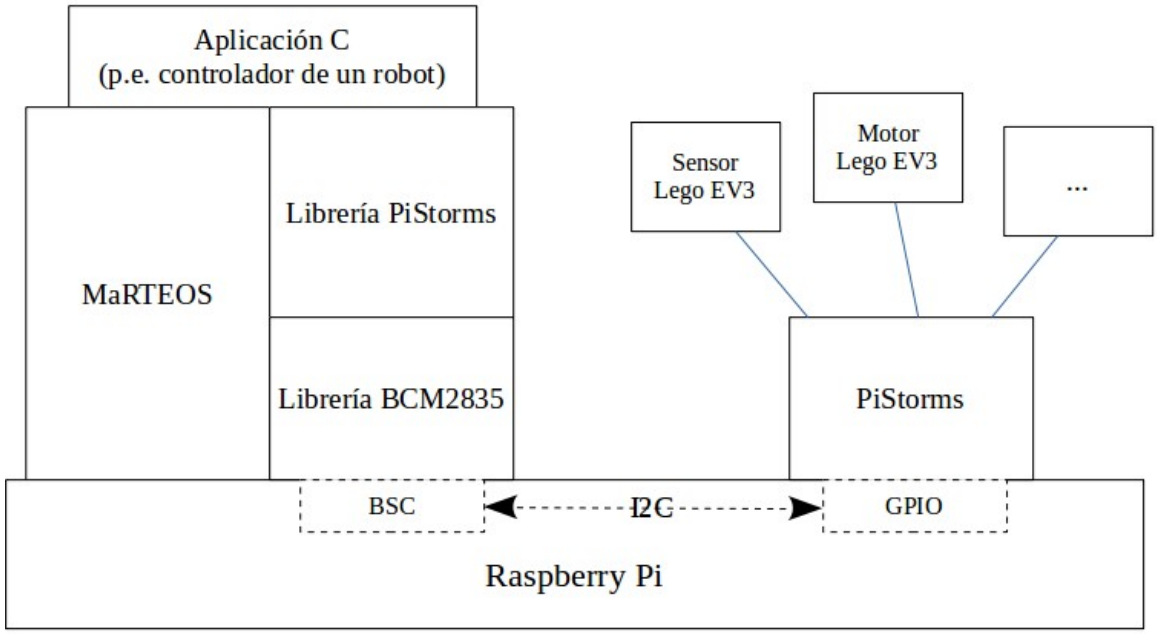
2. Gyroscope Sensort (Measurement of tilt angle and speed of rotation).
3. Color Sensor (It can measure de intensity of light and determinate colors).
4. Touch Sensor (Detects if it is touched or not and the number of touches).

EV3 actuator drivers(motors).

The Lego Mindstorms EV3 includes 2 engine types [9]. Both engines include a relative rotary encoder and it should be the driver of the operating system that acts as a tachometer by measuring the speed of rotation of the motor based on the reading of said rotary encoder. Here is a link with more information `Motor Link`

Estructure

The following image shows the estructure of the aplication.



Module Index

Modules

Here is a list of all modules:

File Index

File List

Here is a list of all documented files with brief descriptions:

Module Documentation

I2C access

Functions
int **bcm2835_i2c_begin** (void)
void **bcm2835_i2c_end** (void)
void **bcm2835_i2c_setSlaveAddress** (uint8_t addr)
void **bcm2835_i2c_setClockDivider** (uint16_t divider)
void **bcm2835_i2c_set_baudrate** (uint32_t baudrate)
uint8_t **bcm2835_i2c_write** (const char *buf, uint32_t len)
uint8_t **bcm2835_i2c_read** (char *buf, uint32_t len)
uint8_t **bcm2835_i2c_read_register_rs** (char *regaddr, char *buf, uint32_t len)
uint8_t **bcm2835_i2c_write_read_rs** (char *cmds, uint32_t cmds_len, char *buf, uint32_t buf_len)

Detailed Description

These functions let you use I2C (The Broadcom Serial Control bus with the Philips I2C bus/interface version 2.1 January 2000.) to interface with an external I2C device.

Function Documentation

int bcm2835_i2c_begin (void)

Start I2C operations. Forces RPi I2C pins P1-03 (SDA) and P1-05 (SCL) to alternate function ALT0, which enables those pins for I2C interface. You should call **bcm2835_i2c_end()** when all I2C functions are complete to return the pins to their default functions

Returns:

1 if successful, 0 otherwise (perhaps because you are not running as root)

See Also:

bcm2835_i2c_end()

Definition at line 745 of file bcm2835.c.

void bcm2835_i2c_end (void)

End I2C operations. I2C pins P1-03 (SDA) and P1-05 (SCL) are returned to their default INPUT behaviour.

Definition at line 776 of file bcm2835.c.

uint8_t bcm2835_i2c_read (char * *buf*, uint32_t *len*)

Transfers any number of bytes from the currently selected I2C slave. (as previously set by

See Also:

bcm2835_i2c_setSlaveAddress)

Parameters:

in	*buf*	Buffer of bytes to receive.

in	*len*	Number of bytes in the buf buffer, and the number of bytes to received.

Returns:

reason see **bcm2835I2CReasonCodes**

Definition at line 900 of file bcm2835.c.

uint8_t bcm2835_i2c_read_register_rs (char * *regaddr*, char * *buf*, uint32_t *len*)

Allows reading from I2C slaves that require a repeated start (without any prior stop) to read after the required slave register has been set. For example, the popular MPL3115A2 pressure and temperature sensor. Note that your device must support or require this mode. If your device does not require this mode then the standard combined:

See Also:

bcm2835_i2c_write
bcm2835_i2c_read are a better choice. Will read from the slave previously set by **bcm2835_i2c_setSlaveAddress**

Parameters:

in	*regaddr*	Buffer containing the slave register you wish to read from.
in	*buf*	Buffer of bytes to receive.
in	*len*	Number of bytes in the buf buffer, and the number of bytes to received.

Returns:

reason see **bcm2835I2CReasonCodes**

Definition at line 975 of file bcm2835.c.

void bcm2835_i2c_set_baudrate (uint32_t *baudrate*)

Sets the I2C clock divider by converting the baudrate parameter to the equivalent I2C clock divider. (see

See Also:

bcm2835_i2c_setClockDivider) For the I2C standard 100khz you would set baudrate to 100000 The use of baudrate corresponds to its use in the I2C kernel device driver. (Of course, bcm2835 has nothing to do with the kernel driver)

Definition at line 820 of file bcm2835.c.

void bcm2835_i2c_setClockDivider (uint16_t *divider*)

Sets the I2C clock divider and therefore the I2C clock speed.

Parameters:

in	*divider*	The desired I2C clock divider, one of BCM2835_I2C_CLOCK_DIVIDER_*, see **bcm2835I2CClockDivider**

Definition at line 804 of file bcm2835.c.

void bcm2835_i2c_setSlaveAddress (uint8_t *addr*)

Sets the I2C slave address.

Parameters:

in	*addr*	The I2C slave address.

Definition at line 789 of file bcm2835.c.

uint8_t bcm2835_i2c_write (const char * *buf*, uint32_t *len*)

Transfers any number of bytes to the currently selected I2C slave. (as previously set by

See Also:

bcm2835_i2c_setSlaveAddress)

Parameters:

in	*buf*	Buffer of bytes to send.
in	*len*	Number of bytes in the buf buffer, and the number of bytes to send.

Returns:

reason see **bcm2835I2CReasonCodes**

Definition at line 829 of file bcm2835.c.

uint8_t bcm2835_i2c_write_read_rs (char * *cmds*, uint32_t *cmds_len*, char * *buf*, uint32_t *buf_len*)

Allows sending an arbitrary number of bytes to I2C slaves before issuing a repeated start (with no prior stop) and reading a response. Necessary for devices that require such behavior, such as the MLX90620. Will write to and read from the slave previously set by

See Also:

bcm2835_i2c_setSlaveAddress

Parameters:

in	*cmds*	Buffer containing the bytes to send before the repeated start condition.
in	*cmds_len*	Number of bytes to send from cmds buffer
in	*buf*	Buffer of bytes to receive.
in	*buf_len*	Number of bytes to receive in the buf buffer.

Returns:

reason see **bcm2835I2CReasonCodes**

Definition at line 1066 of file bcm2835.c.

Initialize and set up de bank

Functions

int **pistorms_init** (void)

Initialize the bcm2835 library and start I2C operations.

int **pistorms_close** (void)

Close the bcm2835 library and end I2C operations .

int **_set_active_bank** (int connector_id)

Sets the I2C slave address.

char * **pistorms_get_device_id** (int connector_id)

Obtain the ID of the sensor.

Detailed Description

These functions let you use bcm2835 library, i2c commands, select the correct bank and obtain the sensors id to future uses.

Function Documentation

int _set_active_bank (int *connector_id*)

Sets the I2C slave address.

Parameters:

connector_id	Bank and Port to plug the sensor.

Returns:

the value of active_bank, -1 if is incorrect.

Sets the I2C slave address (BANK_A or BANK_B). But if the current slave address is the same than the connector_id, the function doesn't set again the slave address because it is not neccessary.

Definition at line 68 of file marte_pistorms.c.

int pistorms_close (void)

Close the bcm2835 library and end I2C operations .

Returns:

return 1 if successful else 0.

Close the bcm2835 calling the function **bcm2835_close()** and end I2C operations calling **bcm2835_i2c_end()**, if there is some error the function is going to return 0 , if not the function is going to return 1.

Definition at line 50 of file marte_pistorms.c.

char* pistorms_get_device_id (int *connector_id*)

Obtain the ID of the sensor.

Parameters:

connector_id	Bank and Port to plug the sensor.

Returns:

the ID of the sensor if it is plugged in,if not error.

Obtains the ID of the Sensor which is plugged in the correct port of the Pistorms. For a correct result, it is neccesary to indicate the bank and port that the sensor is plugged in. If the sensor is connected in other port or bank, the function returns an error.

Definition at line 109 of file marte_pistorms.c.

int pistorms_init (void)

Initialize the bcm2835 library and start I2C operations.

Returns:

return 1 if successful else 0.

Initialize the bcm2835 calling the function **bcm2835_init()** and start I2C operations calling **bcm2835_i2c_begin()**, if there is some error the function is going to return 0 , if not the function is going to return 1.

< Initialize bcm2835 and I2C

Definition at line 33 of file marte_pistorms.c.

Main features of the Pistorms Brick

Functions

int **pistorms_brick_led_On** (int bank_id, int red, int green, int blue)
Writes to the specified RGB LED.

int **pistorms_brick_led_Off** (int bank_id)
Turn off the led.

char * **pistorms_brick_get_firmware_version** (int bank_id)
Returns the PiStorms firmware version.

char * **pistorms_brick_get_vendor_id** (int bank_id)
Returns the PiStorms vendor ID.

char * **pistorms_brick_get_device_id** (int bank_id)
Returns the PiStorms device ID.

int **pistorms_brick_get_battery_voltage** (void)
Obtains the input battery voltage.

int **pistorms_brick_get_key_press_value** (void)
Check if any button GO is pressed.

int **pistorms_brick_get_key_press_count** (void)
Obatins the GO button press count.

void **pistorms_brick_reset_key_press_count** (void)
Resets the GO button press count.

char * **pistorms_brick_touch_screen_X_asis** (void)
Obtain the value of the X asis.

char * **pistorms_brick_touch_screen_Y_asis** (void)
Obtain the value of the Y asis.

int **pistorms_brick_screen_is_touched** (void)
Detects if touchsreen is Touched.

Detailed Description

These functions let you use the brick, its touch screen, button GO, leds, battery..

Function Documentation

int pistorms_brick_get_battery_voltage (void)

Obtains the input battery voltage.

Returns:

input battery voltage.

Returns the input battery voltage.

Definition at line 200 of file marte_pistorms_brick.c.

char* pistorms_brick_get_device_id (int *bank_id*)

Returns the PiStorms device ID.

Parameters:

bank_id	the Bank (is the same devide ID to BANK_A and BANK_B).

Returns:

the PiStorms device ID.

Returns the PiStorms device ID.

Definition at line 185 of file marte_pistorms_brick.c.

char* pistorms_brick_get_firmware_version (int *bank_id*)

Returns the PiStorms firmware version.

Parameters:

bank_id	the Bank (is the same firmware version to BANK_A and BANK_B).

Returns:

the PiStorms firmware version.

Returns the PiStorms firmware version of the Bank selected.

Definition at line 154 of file marte_pistorms_brick.c.

int pistorms_brick_get_key_press_count (void)

Obatins the GO button press count.

Returns:

GO button press count.

Returns the number of times that GO button is pressed.

Definition at line 232 of file marte_pistorms_brick.c.

int pistorms_brick_get_key_press_value (void)

Check if any button GO is pressed.

Returns:

1 if it is pressed or 0 if not.

Definition at line 216 of file marte_pistorms_brick.c.

char* pistorms_brick_get_vendor_id (int *bank_id*)

Returns the PiStorms vendor ID.

Parameters:

bank_id	the Bank (is the same vendor ID to BANK_A and BANK_B).

Returns:

the PiStorms vendor ID.

Returns the PiStorms vendor ID.

Definition at line 170 of file marte_pistorms_brick.c.

int pistorms_brick_led_Off (int *bank_id*)

Turn off the led.

Parameters:

bank_id	the Bank to turn off its led.

Returns:

Pistorms codes see **marte_pistorms.h** in the section of PistormsCodes

Turn off the led of the specified Bank.

It is necessary to generate a short time after doing the write to wait for the change in the mode. This time is generate by th efunction nanosleep().

```
struct timespec tim;
tim.tv_sec = 0;
tim.tv_nsec = 10000000;
```

Definition at line 117 of file marte_pistorms_brick.c.

int pistorms_brick_led_On (int *bank_id*, int *red*, int *green*, int *blue*)

Writes to the specified RGB LED.

Parameters:

bank_id	the Bank to turn on its led.

Returns:

Pistorms codes see **marte_pistorms.h** in the section of PistormsCodes

Turn on the led of the specified Bank. Writes to the specified RGB LED.

It is necessary to generate a short time after doing the write to wait for the change in the mode. This time is generate by th efunction nanosleep().

```
struct timespec tim;
tim.tv_sec = 0;
tim.tv_nsec = 10000000;
```

Definition at line 76 of file marte_pistorms_brick.c.

void pistorms_brick_reset_key_press_count (void)

Resets the GO button press count.

Change the count of GO button to 0.

Definition at line 247 of file marte_pistorms_brick.c.

int pistorms_brick_screen_is_touched (void)

Detects if touchsreen is Touched.

Returns:

0 if screen is not touched or 1 if it is touched.

Detects if touchsreen is Touched.

Definition at line 288 of file marte_pistorms_brick.c.

char* pistorms_brick_touch_screen_X_asis (void)

Obtain the value of the X asis.

Returns:

the value of the X asis.

If the touch screen is touched, this function is going to return the value of the X asis

Definition at line 260 of file marte_pistorms_brick.c.

char* pistorms_brick_touch_screen_Y_asis (void)

Obtain the value of the Y asis.

Returns:

the value of the Y asis.

If the touch screen is touched, this function is going to return the value of the Y asis.

Definition at line 274 of file marte_pistorms_brick.c.

Functions of the motors

Functions

int **pistorms_motor_go** (int connector_id, char go)
Turn on the motor with an indicated configuration.

long **pistorms_motor_get_pos** (int connector_id)
Obtains the current encoder position of the motor.

int **pistorms_motor_set_pos** (int connector_id, long pos)
Set up the encoder of the motor with a new position.

int **pistorms_motor_reset_pos** (int connector_id)
Resets the encoder position of the specified motor.

int **pistorms_motor_reset_all_parameters** (int bank_id)
Reset all Encoder values and motor parameters.

int **pistorms_motor_set_speed** (int connector_id, int speed)
Run the motor at a set speed for an unlimited duration.

int **pistorms_motor_set_secs** (int connector_id, int time)
Run motor in time mode.

int **pistorms_motor_float** (int connector_id)
Stop the motor smoothly with float.

int **pistorms_motor_float_sync** (int bank_id)
Stop both the motors of said bank at the same time motors are stopped smoothly with float.

int **pistorms_motor_brake** (int connector_id)
Stop the motor abruptly with brake.

int **pistorms_motor_brake_sync** (int bank_id)
Stop both the motors of said bank at the same time motors are stopped abruptly with a brake.

Detailed Description

These functions let you use the large and medium motors, control the speed, the time of running in seconds, the position encoder...

Function Documentation

int pistorms_motor_brake (int *connector_id*)

Stop the motor abruptly with brake.

Parameters:

connector_id	Bank and Port to plug the motor.

Stop the motor abruptly with brake.

Definition at line 365 of file marte_pistorms_motors.c.

int pistorms_motor_brake_sync (int *bank_id*)

Stop both the motors of said bank at the same time motors are stopped abruptly with a brake.

Parameters:

bank_id	the Bank to brake the motors.

Motors are stopped with a brake. You can call this function on any motor of that bank and it will work on both motors of that bank.

Definition at line 392 of file marte_pistorms_motors.c.

int pistorms_motor_float (int *connector_id*)

Stop the motor smoothly with float.

Parameters:

connector_id	Bank and Port to plug the motor.

Stop the motor smoothly with float. This function only goes with Large Motor, if it is used for Medium Motor it will stop as a brake.

Definition at line 312 of file marte_pistorms_motors.c.

int pistorms_motor_float_sync (int *bank_id*)

Stop both the motors of said bank at the same time motors are stopped smoothly with float.

Parameters:

bank_id	the Bank to float the motors.

Motors are stopped smoothly with float You can call this function on any motor of that bank and it will work on both motors of that bank. This function only goes with Large Motor, if it is used for Medium Motor it will stop as a brake.

long pistorms_motor_get_pos (int *connector_id*)

Obtains the current encoder position of the motor.

Parameters:

connector_id	Bank and Port to plug the motor.

Returns:

the current encoder position of the motor.

Returns the current encoder position of the motor. When the motor rotates clockwise, the position will increase. Likewise, rotating counter-clockwise causes the position to decrease.

Definition at line 132 of file marte_pistorms_motors.c.

int pistorms_motor_go (int *connector_id*, char *go*)

Turn on the motor with an indicated configuration.

Parameters:

connector_id	Bank and Port to plug the motor.
go	indicate the mode that the motor is going to run.

This function power up the motor with a specified configuration. It is necessary say which parameters you have changed: For example if you want to run the motor for 5 seconds with 50 of speed you have to do:

```
1° Call the function pistorms_motor_run_secs(int connector_id, int time)
2° Call the function  pistorms_motor_set_speed(int connector_id,int speed)
3° Call this function pistorms_motor_go(int connector_id, SPEED_GO | TIME_GO)

The param char go have different values :
```

SPEED_GO RAMP_SPEED CHANGE_BASED_ON_ENCODER ENCODER_GO BRAKE_FLOAT_MOVEMENT ENCODER_ACTIVE_FEEDBACK TIME_GO MOTOR_GO

Definition at line 100 of file marte_pistorms_motors.c.

int pistorms_motor_reset_all_parameters (int *bank_id*)

Reset all Encoder values and motor parameters.

Parameters:

bank_id	Bank with the motors.

Reset all Encoder values and motor parameters. (This does not reset the PID parameters).

Definition at line 228 of file marte_pistorms_motors.c.

int pistorms_motor_reset_pos (int *connector_id*)

Resets the encoder position of the specified motor.

Parameters:

connector_id	Bank and Port to plug the motor.

The encoder of the motor is going to be 0. When the motor rotates clockwise, the position will increase. Likewise, rotating counter-clockwise causes the position to decrease.

Definition at line 198 of file marte_pistorms_motors.c.

int pistorms_motor_set_pos (int *connector_id*, long *pos*)

Set up the encoder of the motor with a new position.

Parameters:

connector_id	Bank and Port to plug the motor.
long	The position that the encoder of the motor is going to achieve.

The encoder of the motor is going to have a new value.

Definition at line 161 of file marte_pistorms_motors.c.

int pistorms_motor_set_secs (int *connector_id*, int *time*)

Run motor in time mode.

Parameters:

connector_id	Bank and Port to plug the motor.
time	Seconds that the motor is going to run.

Run the motor for a specific time in seconds. To run the motor in time mode, it is neccesary to specified the speed to with the function **pistorms_motor_set_speed**

int pistorms_motor_set_speed (int *connector_id*, int *speed*)

Run the motor at a set speed for an unlimited duration.

Parameters:

connector_id	Bank and Port to plug the motor.
speed	The speed at which to turn the motor.

Run the motor at a set speed for an unlimited duration. The value of speed is from 1 to 100. The max power is 100. It is like the percentage of the motor's power.

Definition at line 252 of file marte_pistorms_motors.c.

Functions for use the Color EV3 Sensor

Functions

int **pistorms_sensor_color_configure** (int connector_id)

Detects if the Color Sensor is connect correctly.

int **pistorms_color_set_mode** (int connector_id, int mode)

Configure the mode of the Color Sensor.

int **pistorms_color_read_light** (int connector_id, int mode)

Color sensor can measure the intensity of light that enters the small window on the face of the sensor.

int **pistorms_color_measure** (int connector_id)

Color sensor recognizes seven colors.

Detailed Description

These functions let you use the color EV3 Sensor

Function Documentation

int pistorms_color_measure (int *connector_id*)

Color sensor recognizes seven colors.

Parameters:

connector_id	Bank and Port to plug the sensor.

Returns:

black (1), blue (2), green (3), yellow (4), red (5), white (6), and brown (7) —plus No Color (0).

In measure color mode, this sensor can differentiate some colors.

Definition at line 82 of file marte_pistorms_sensor_color.c.

int pistorms_color_read_light (int *connector_id*, int *mode*)

Color sensor can measure the intensity of light that enters the small window on the face of the sensor.

Parameters:

connector_id	Bank and Port to plug the sensor.
mode	it can be "REFLECTED" or "AMBIENT".

Returns:

the intensity of light. 0 (very dark) to 100 (very light).

(Reflected Light Intensity) The Color Sensor measures the intensity of light reflected back from a red light–emitting lamp. The sensor uses a scale of 0 (very dark) to 100 (very light). This means your robot might be programmed to move around on a white surface until a black line is detected, or to interpret a color-coded identification card.

(Ambient Light Intensity) The Color Sensor measures the strength of light that enters the window from its environment,such as sunlight or the beam of a flashlight. The sensor uses a scale of 0 (very dark) to 100 (very light). This means your robot might be programmed to set off an alarm when the sun rises in the morning, or stop action if the lights go out.

Definition at line 101 of file marte_pistorms_sensor_color.c.

int pistorms_color_set_mode (int *connector_id*, int *mode*)

Configure the mode of the Color Sensor.

Parameters:

connector_id	Bank and Port to plug the sensor.
mode	it can be "REFLECTED_LIGHT", "AMBIENT_LIGHT" or "MEASURE_COLOR".

Returns:

-1 if there is an error, or 1 if the sensor is correct. See **marte_pistorms.h** in the section of PistormsCodes

Set up the mode of the sensor, it can be REFLECTED_LIGHT, AMBIENT_LIGHT or MEASURE_COLOR.

Definition at line 54 of file marte_pistorms_sensor_color.c.

int pistorms_sensor_color_configure (int *connector_id*)

Detects if the Color Sensor is connect correctly.

Parameters:

connector_id	Bank and Port to plug the sensor.

Returns:

-1 if there is an error, or 1 if the sensor is correct. See **marte_pistorms.h** in the section of PistormsCodes

If the Color sensor is connect, the function is going to return 1 if it is connect correctly or -1 if it isn't.

Definition at line 30 of file marte_pistorms_sensor_color.c.

Functions for use the Gyro EV3 Sensor

Functions

int **pistorms_sensor_gyro_configure** (int connector_id)

Detects if the Gyro Sensor is connect correctly.

int **pistorms_gyro_set_mode** (int connector_id, int mode)

Configure the mode of the Gyro Sensor.

short **pistorms_gyro_read** (int connector_id, int mode)

Read data of the Gyro Sensor depends on the mode.

Detailed Description

These functions let you use the gyro EV3 Sensor

Function Documentation

short pistorms_gyro_read (int *connector_id*, int *mode*)

Read data of the Gyro Sensor depends on the mode.

Parameters:

connector_id	Bank and Port to plug the sensor.
mode	it could be "ANGLE" or "RATE".

Returns:

the angle in degrees if ANGLE or the rotational speed in degrees per second if RATE.

Mode Angle: You can use this rotation angle to detect, for example, how far your robot has turned. This feature means you are able to program turns (on the axis the Gyro Sensor is measuring) with an accuracy of +/- 3 degrees for a 90-degree turn.

Mode Rate: If you rotate the Gyro Sensor in the direction of the arrows on the case of the sensor, the sensor can detect the rate of rotation in degrees per second. (The sensor can measure a maximum rate of spin of 440 degrees per second).

Definition at line 74 of file marte_pistorms_sensor_gyro.c.

int pistorms_gyro_set_mode (int *connector_id*, int *mode*)

Configure the mode of the Gyro Sensor.

Parameters:

connector_id	Bank and Port to plug the sensor.
mode	it can be "ANGLE" or "RATE".

Returns:

-1 if there is an error, or 1 if the sensor is correct. See **marte_pistorms.h** in the section of PistormsCodes

Set up the mode of the sensor, it can be ANGLE or RATE.

Definition at line 50 of file marte_pistorms_sensor_gyro.c.

int pistorms_sensor_gyro_configure (int *connector_id*)

Detects if the Gyro Sensor is connect correctly.

Parameters:

connector_id	Bank and Port to plug the sensor.

Returns:

-1 if there is an error, or 1 if the sensor is correct. See **marte_pistorms.h** in the section of PistormsCodes

If the Gyro sensor is connect, the function is going to return 1 if it is connect correctly or -1 if it isn't.

Definition at line 27 of file marte_pistorms_sensor_gyro.c.

Functions for use the Touch EV3 Sensor

Functions

int **pistorms_sensor_configure_touch** (int connector_id)
Detects if the Touch Sensor is connect correctly.

int **pistorms_is_touched** (int connector_id)
check if the sensor is touched.

int **pistorms_num_touches** (int connector_id)
Count how many times the sensor was touched.

int **pistorms_reset_touches** (int connector_id)
Reset the count.

Detailed Description

These functions let you use the touch EV3 Sensor

Function Documentation

int pistorms_is_touched (int *connector_id*)

check if the sensor is touched.

Parameters:

connector_id	Bank and Port to plug the sensor.

Returns:

true if it is touched.

Check if the sensor is touched, if it is pressed the function returns True(1), if not, it returns False(0).

Definition at line 47 of file marte_pistorms_sensor_touch.c.

int pistorms_num_touches (int *connector_id*)

Count how many times the sensor was touched.

Parameters:

connector_id	Bank and Port to plug the sensor.

Returns:

count of touches since last reset (or power on).

With PiStorms it is possible to count how many times the sensor was touched. This count is maintained since the PiStorms was powered on. You can reset this count with See **marte_pistorms.h** in the section of PistormsCodes

Definition at line 61 of file marte_pistorms_sensor_touch.c.

int pistorms_reset_touches (int *connector_id*)

Reset the count.

Parameters:

connector_id	Bank and Port to plug the sensor.

Returns:

Pistorms codes, See **marte_pistorms.h** in the section of PistormsCodes

With this function, is possible reset the count of touches.

Definition at line 74 of file marte_pistorms_sensor_touch.c.

int pistorms_sensor_configure_touch (int *connector_id*)

Detects if the Touch Sensor is connect correctly.

Parameters:

connector_id	Bank and Port to plug the sensor.

Returns:

-1 if there is an error, or 1 if the sensor is correct. See **marte_pistorms.h** in the section of PistormsCodes

If the Touch sensor is connect, the function is going to return 1 if it is connect correctly or -1 if it isn't.

Functions for use the Ultrasonic EV3 Sensor

Functions

int **pistorms_sensor_ultrasonic_configure** (int connector_id)

Detects if the Ultrasonic Sensor is connect correctly.

int **pistorms_ultrasonic_set_mode** (int connector_id, int mode)

Configure the mode of the Ultrasonic Sensor.

float **pistorms_ultrasonic_read_distance** (int connector_id, int mode)

Ultrasonic sensor can measure the distance to an object in front of it.

int **pistorms_ultrasonic_presence** (int connector_id)

Ultrasonic sensor can detect another Ultrasonic Sensor operating nearby.

Detailed Description

These functions let you use the ultrasonic EV3 Sensor

Function Documentation

int pistorms_sensor_ultrasonic_configure (int *connector_id*)

Detects if the Ultrasonic Sensor is connect correctly.

Parameters:

connector_id	Bank and Port to plug the sensor.

Returns:

-1 if there is an error, or 1 if the sensor is correct. See **marte_pistorms.h** in the section of PistormsCodes

If the Ultrasonic sensor is connect, the function is going to return 1 if it is connect correctly or -1 if it isn't.

Definition at line 30 of file marte_pistorms_sensor_ultrasonic.c.

int pistorms_ultrasonic_presence (int *connector_id*)

Ultrasonic sensor can detect another Ultrasonic Sensor operating nearby.

Parameters:

connector_id	Bank and Port to plug the sensor.

Returns:

1 if there is another Ultrasonic sensor, or 0 if not.

In Presence Mode, this sensor can detect another Ultrasonic Sensor operating nearby. When listening for presence, the sensor detects sound signals but does not send them.

Definition at line 82 of file marte_pistorms_sensor_ultrasonic.c.

float pistorms_ultrasonic_read_distance (int *connector_id*, int *mode*)

Ultrasonic sensor can measure the distance to an object in front of it.

Parameters:

connector_id	Bank and Port to plug the sensor.
mode	it can be "CENTIMETERS" or "INCHES".

Returns:

distance to an object in centimeters or inches.

(Centimeters) The detectable distance is between 3 and 250 centimeters (with an accuracy of +/- 1 centimeters). A value of 255 centimeters means the sensor is not able to detect any object in front of it.

(Inches) The measurable distance is between 1 and 99 inches (with an accuracy of +/- 0.394 inches). A value of 100 inches means the sensor is not able to detect any object in front of it.

Definition at line 101 of file marte_pistorms_sensor_ultrasonic.c.

int pistorms_ultrasonic_set_mode (int *connector_id*, int *mode*)

Configure the mode of the Ultrasonic Sensor.

Parameters:

connector_id	Bank and Port to plug the sensor.
mode	it can be "CENTIMETERS", "INCHES" or PRESENCE.

Returns:

-1 if there is an error, or 1 if the sensor is correct. See **marte_pistorms.h** in the section of PistormsCodes

Set up the mode of the sensor, it can be CENTIMETERS, INCHES or PRESENCE.

Definition at line 54 of file marte_pistorms_sensor_ultrasonic.c.

Main features for sensors in general

Functions

int **pistorms_port_set_type_sensor** (int connector_id, int type)

Determine sensor type on the specified port.

int **pistorms_sensor_get_mode** (int connector_id)

Obtain the mode that the EV3 Sensor is running.

int **pistorms_sensor_set_mode** (int connector_id, int mode)

Set the mode for the EV3 Sensor.

char * **pistorms_sensor_read** (int connector_id)

Read the data of EV3 sensors.

Detailed Description

These functions let you use sensors, modify the type of connector, change the mode of the sensor, read the sensor data...

Function Documentation

int pistorms_port_set_type_sensor (int *connector_id*, int *type*)

Determine sensor type on the specified port.

Parameters:

connector_id	Bank and Port to plug the Touch sensor.
type	the type of the sensor which is plugged.

Returns:

Pistorms codes, see **marte_pistorms.h** in the section of PistormsCodes

Configure the type mode on the specified port. Default Type Modes in Pistorms is EV3 ---> value 19. So it is neccesary to change this value to use other types of sensors.

It is necessary to generate a short time after doing the write to wait for the change in the mode. This time is generate by th efunction nanosleep().

```
struct timespec tim;
tim.tv_sec = 0;
tim.tv_nsec = 10000000;
```

< Value of Type mode

Definition at line 28 of file marte_pistorms_sensors.c.

int pistorms_sensor_get_mode (int *connector_id*)

Obtain the mode that the EV3 Sensor is running.

Parameters:

connector_id	Bank and Port to plug the EV3 sensor.

Returns:

value depends on the mode of the sensor(value --> 0,1 or 2) or -1 if there is an error, see **marte_pistorms.h** in the section of PistormsCodes

Gets the mode of EV3 Sensor. Default mode is 0 for all EV3 Sensors. Gyro Sensor -- |-->value = 0 -> ANGLE |-->value = 1 -> RATE Color Sensor -- |-->value = 0 -> REFLECTED_LIGHT |--

>value = 1 -> AMBIENT_LIGHT |-->value = 2 -> MEASURE_COLOR Infrared Sensor -- |-->value = 0 -> PROXIMITY |-->value = 1 -> BEACON |-->value = 2 -> REMOTE Ultrasonic Sensor -- |-->value = 0 -> PROXIMITY_CENTIMETERS |-->value = 1 -> PROXIMITY_INCHES |-->value = 2 -> PRESENCE

< Read number of bytes of the requerided register

Definition at line 63 of file marte_pistorms_sensors.c.

char* pistorms_sensor_read (int *connector_id*)

Read the data of EV3 sensors.

Parameters:

connector_id	Bank and Port to plug the EV3 sensor.

Returns:

the buffer with the data which the sensor is reading.

Gets the data that the sensor is reading in real time. The function is going to return the buffer with the information of the sensor is reading.

Definition at line 124 of file marte_pistorms_sensors.c.

int pistorms_sensor_set_mode (int *connector_id*, int *mode*)

Set the mode for the EV3 Sensor.

Parameters:

connector_id	Bank and Port to plug the EV3 sensor.
mode	Mode of the Sensor.

Returns:

Pistorms codes, see **marte_pistorms.h** in the section of PistormsCodes

Gets the mode of EV3 Sensor. Default mode is 0 for all EV3 Sensors. Gyro Sensor -- |-->value = 0 -> ANGLE |-->value = 1 -> RATE Color Sensor -- |-->value = 0 -> REFLECTED_LIGHT |-->value = 1 -> AMBIENT_LIGHT |-->value = 2 -> MEASURE_COLOR Infrared Sensor -- |-->value = 0 -> PROXIMITY |-->value = 1 -> BEACON |-->value = 2 -> REMOTE Ultrasonic Sensor -- |-->value = 0 -> PROXIMITY_CENTIMETERS |-->value = 1 -> PROXIMITY_INCHES |-->value = 2 -> PRESENCE

It is necessary to generate a short time after doing the write to wait for the change in the mode. This time is generate by th efunction nanosleep().

```
struct timespec tim;
tim.tv_sec = 0;
tim.tv_nsec = 10000000;
```

Definition at line 92 of file marte_pistorms_sensors.c.

File Documentation

bcm2835.c File Reference

C library for Broadcom BCM 2835 as used in Raspberry Pi.

```
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <sys/types.h>
#include "bcm2835.h"
```

Macros

#define **BCK2835_LIBRARY_BUILD**
#define **MAP_FAILED** (void *) -1

Functions

uint32_t * **bcm2835_regbase** (uint8_t regbase)
void **bcm2835_set_debug** (uint8_t d)
unsigned int **bcm2835_version** (void)
uint32_t **bcm2835_peri_read** (volatile uint32_t *paddr)
uint32_t **bcm2835_peri_read_nb** (volatile uint32_t *paddr)
void **bcm2835_peri_write** (volatile uint32_t *paddr, uint32_t value)
void **bcm2835_peri_write_nb** (volatile uint32_t *paddr, uint32_t value)
void **bcm2835_peri_set_bits** (volatile uint32_t *paddr, uint32_t value, uint32_t mask)
void **bcm2835_gpio_fsel** (uint8_t pin, uint8_t mode)
void **bcm2835_gpio_set** (uint8_t pin)
void **bcm2835_gpio_clr** (uint8_t pin)
void **bcm2835_gpio_set_multi** (uint32_t mask)
void **bcm2835_gpio_clr_multi** (uint32_t mask)
uint8_t **bcm2835_gpio_lev** (uint8_t pin)
uint8_t **bcm2835_gpio_eds** (uint8_t pin)
uint32_t **bcm2835_gpio_eds_multi** (uint32_t mask)
void **bcm2835_gpio_set_eds** (uint8_t pin)
void **bcm2835_gpio_set_eds_multi** (uint32_t mask)
void **bcm2835_gpio_ren** (uint8_t pin)
void **bcm2835_gpio_clr_ren** (uint8_t pin)
void **bcm2835_gpio_fen** (uint8_t pin)
void **bcm2835_gpio_clr_fen** (uint8_t pin)
void **bcm2835_gpio_hen** (uint8_t pin)
void **bcm2835_gpio_clr_hen** (uint8_t pin)
void **bcm2835_gpio_len** (uint8_t pin)
void **bcm2835_gpio_clr_len** (uint8_t pin)
void **bcm2835_gpio_aren** (uint8_t pin)
void **bcm2835_gpio_clr_aren** (uint8_t pin)
void **bcm2835_gpio_afen** (uint8_t pin)
void **bcm2835_gpio_clr_afen** (uint8_t pin)
void **bcm2835_gpio_pud** (uint8_t pud)

void **bcm2835_gpio_pudclk** (uint8_t pin, uint8_t on)
uint32_t **bcm2835_gpio_pad** (uint8_t group)
void **bcm2835_gpio_set_pad** (uint8_t group, uint32_t control)
void **bcm2835_delay** (unsigned int millis)
void **bcm2835_delayMicroseconds** (uint64_t micros)
void **bcm2835_gpio_write** (uint8_t pin, uint8_t on)
void **bcm2835_gpio_write_multi** (uint32_t mask, uint8_t on)
void **bcm2835_gpio_write_mask** (uint32_t value, uint32_t mask)
void **bcm2835_gpio_set_pud** (uint8_t pin, uint8_t pud)
int **bcm2835_spi_begin** (void)
void **bcm2835_spi_end** (void)
void **bcm2835_spi_setBitOrder** (uint8_t __attribute__((unused)) order)
void **bcm2835_spi_setClockDivider** (uint16_t divider)
void **bcm2835_spi_setDataMode** (uint8_t mode)
uint8_t **bcm2835_spi_transfer** (uint8_t value)
void **bcm2835_spi_transfernb** (char *tbuf, char *rbuf, uint32_t len)
void **bcm2835_spi_writenb** (char *tbuf, uint32_t len)
void **bcm2835_spi_transfern** (char *buf, uint32_t len)
void **bcm2835_spi_chipSelect** (uint8_t cs)
void **bcm2835_spi_setChipSelectPolarity** (uint8_t cs, uint8_t active)
int **bcm2835_i2c_begin** (void)
void **bcm2835_i2c_end** (void)
void **bcm2835_i2c_setSlaveAddress** (uint8_t addr)
void **bcm2835_i2c_setClockDivider** (uint16_t divider)
void **bcm2835_i2c_set_baudrate** (uint32_t baudrate)
uint8_t **bcm2835_i2c_write** (const char *buf, uint32_t len)
uint8_t **bcm2835_i2c_read** (char *buf, uint32_t len)
uint8_t **bcm2835_i2c_read_register_rs** (char *regaddr, char *buf, uint32_t len)
uint8_t **bcm2835_i2c_write_read_rs** (char *cmds, uint32_t cmds_len, char *buf, uint32_t buf_len)
uint64_t **bcm2835_st_read** (void)
void **bcm2835_st_delay** (uint64_t offset_micros, uint64_t micros)
void **bcm2835_pwm_set_clock** (uint32_t divisor)
void **bcm2835_pwm_set_mode** (uint8_t channel, uint8_t markspace, uint8_t enabled)
void **bcm2835_pwm_set_range** (uint8_t channel, uint32_t range)
void **bcm2835_pwm_set_data** (uint8_t channel, uint32_t data)
int **bcm2835_init** (void)

Variables

uint32_t * **bcm2835_peripherals_base** = (uint32_t *)**BCM2835_PERI_BASE**
uint32_t **bcm2835_peripherals_size** = **BCM2835_PERI_SIZE**
uint32_t * **bcm2835_peripherals** = (uint32_t *)MAP_FAILED
volatile uint32_t * **bcm2835_gpio** = (uint32_t *)MAP_FAILED
volatile uint32_t * **bcm2835_pwm** = (uint32_t *)MAP_FAILED
volatile uint32_t * **bcm2835_clk** = (uint32_t *)MAP_FAILED
volatile uint32_t * **bcm2835_pads** = (uint32_t *)MAP_FAILED
volatile uint32_t * **bcm2835_spi0** = (uint32_t *)MAP_FAILED
volatile uint32_t * **bcm2835_bsc0** = (uint32_t *)MAP_FAILED
volatile uint32_t * **bcm2835_bsc1** = (uint32_t *)MAP_FAILED
volatile uint32_t * **bcm2835_st** = (uint32_t *)MAP_FAILED

Detailed Description

C library for Broadcom BCM 2835 as used in Raspberry Pi.

Author:
Mike McCauley

Date:
31 Mar 2015

Version:
1.23

Definition in file **bcm2835.c**.

Function Documentation

void bcm2835_delay (unsigned int *millis*)

Delays for the specified number of milliseconds. Uses nanosleep(), and therefore does not use CPU until the time is up. However, you are at the mercy of nanosleep(). From the manual for nanosleep(): If the interval specified in req is not an exact multiple of the granularity underlying clock (see time(7)), then the interval will be rounded up to the next multiple. Furthermore, after the sleep completes, there may still be a delay before the CPU becomes free to once again execute the calling thread.

Parameters:

in	*millis*	Delay in milliseconds

Definition at line 449 of file bcm2835.c.

void bcm2835_delayMicroseconds (uint64_t *micros*)

Delays for the specified number of microseconds. Uses a combination of nanosleep() and a busy wait loop on the BCM2835 system timers, However, you are at the mercy of nanosleep(). From the manual for nanosleep(): If the interval specified in req is not an exact multiple of the granularity underlying clock (see time(7)), then the interval will be rounded up to the next multiple. Furthermore, after the sleep completes, there may still be a delay before the CPU becomes free to once again execute the calling thread. For times less than about 450 microseconds, uses a busy wait on the System Timer. It is reported that a delay of 0 microseconds on RaspberryPi will in fact result in a delay of about 80 microseconds. Your mileage may vary.

Parameters:

in	*micros*	Delay in microseconds

Definition at line 459 of file bcm2835.c.

void bcm2835_gpio_afen (uint8_t *pin*)

Enable Asynchronous Falling Edge Detect Enable for the specified pin. When a falling edge is detected, sets the appropriate pin in Event Detect Status. Asynchronous means the incoming signal is not sampled by the system clock. As such falling edges of very short duration can be detected.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

Definition at line 393 of file bcm2835.c.

void bcm2835_gpio_aren (uint8_t *pin*)

Enable Asynchronous Rising Edge Detect Enable for the specified pin. When a rising edge is detected, sets the appropriate pin in Event Detect Status. Asynchronous means the incoming signal is not sampled by the system clock. As such rising edges of very short duration can be detected.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

Definition at line 377 of file bcm2835.c.

void bcm2835_gpio_clr (uint8_t *pin*)

Sets the specified pin output to LOW.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

See Also:

bcm2835_gpio_write()

Definition at line 249 of file bcm2835.c.

void bcm2835_gpio_clr_afen (uint8_t *pin*)

Disable Asynchronous Falling Edge Detect Enable for the specified pin.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

Definition at line 400 of file bcm2835.c.

void bcm2835_gpio_clr_aren (uint8_t *pin*)

Disable Asynchronous Rising Edge Detect Enable for the specified pin.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

Definition at line 384 of file bcm2835.c.

void bcm2835_gpio_clr_fen (uint8_t *pin*)

Disable Falling Edge Detect Enable for the specified pin.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

Definition at line 336 of file bcm2835.c.

void bcm2835_gpio_clr_hen (uint8_t *pin*)

Disable High Detect Enable for the specified pin.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

Definition at line 352 of file bcm2835.c.

void bcm2835_gpio_clr_len (uint8_t *pin*)

Disable Low Detect Enable for the specified pin.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

Definition at line 368 of file bcm2835.c.

void bcm2835_gpio_clr_multi (uint32_t *mask*)

Sets any of the first 32 GPIO output pins specified in the mask to LOW.

Parameters:

in	*mask*	Mask of pins to affect. Use eg: (1 << RPI_GPIO_P1_03) | (1 << RPI_GPIO_P1_05)

See Also:

bcm2835_gpio_write_multi()

Definition at line 264 of file bcm2835.c.

void bcm2835_gpio_clr_ren (uint8_t *pin*)

Disable Rising Edge Detect Enable for the specified pin.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

Definition at line 320 of file bcm2835.c.

uint8_t bcm2835_gpio_eds (uint8_t *pin*)

Event Detect Status. Tests whether the specified pin has detected a level or edge as requested by **bcm2835_gpio_ren()**, **bcm2835_gpio_fen()**, **bcm2835_gpio_hen()**, **bcm2835_gpio_len()**, **bcm2835_gpio_aren()**, **bcm2835_gpio_afen()**. Clear the flag for a given pin by calling bcm2835_gpio_set_eds(pin);

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

Returns:

HIGH if the event detect status for the given pin is true.

Definition at line 282 of file bcm2835.c.

uint32_t bcm2835_gpio_eds_multi (uint32_t *mask*)

Same as **bcm2835_gpio_eds()** but checks if any of the pins specified in the mask have detected a level or edge.

Parameters:

in	*mask*	Mask of pins to check. Use eg: (1 << RPI_GPIO_P1_03) | (1 << RPI_GPIO_P1_05)

Returns:

Mask of pins HIGH if the event detect status for the given pin is true.

Definition at line 290 of file bcm2835.c.

void bcm2835_gpio_fen (uint8_t *pin*)

Enable Falling Edge Detect Enable for the specified pin. When a falling edge is detected, sets the appropriate pin in Event Detect Status. The GPRENn registers use synchronous edge detection. This means the input signal is sampled using the system clock and then it is looking for a ?100? pattern on the sampled signal. This has the effect of suppressing glitches.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

Definition at line 329 of file bcm2835.c.

void bcm2835_gpio_fsel (uint8_t *pin*, uint8_t *mode*)

GPIO register access These functions allow you to control the GPIO interface. You can set the function of each GPIO pin, read the input state and set the output state.

Sets the Function Select register for the given pin, which configures the pin as Input, Output or one of the 6 alternate functions.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.
in	*mode*	Mode to set the pin to, one of BCM2835_GPIO_FSEL_* from **bcm2835FunctionSelect**

Definition at line 230 of file bcm2835.c.

void bcm2835_gpio_hen (uint8_t *pin*)

Enable High Detect Enable for the specified pin. When a HIGH level is detected on the pin, sets the appropriate pin in Event Detect Status.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

Definition at line 345 of file bcm2835.c.

void bcm2835_gpio_len (uint8_t *pin*)

Enable Low Detect Enable for the specified pin. When a LOW level is detected on the pin, sets the appropriate pin in Event Detect Status.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

Definition at line 361 of file bcm2835.c.

uint8_t bcm2835_gpio_lev (uint8_t *pin*)

Reads the current level on the specified pin and returns either HIGH or LOW. Works whether or not the pin is an input or an output.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

Returns:

the current level either HIGH or LOW

Definition at line 271 of file bcm2835.c.

uint32_t bcm2835_gpio_pad (uint8_t *group*)

Reads and returns the Pad Control for the given GPIO group.

Parameters:

in	*group*	The GPIO pad group number, one of BCM2835_PAD_GROUP_GPIO_*

Returns:

Mask of bits from BCM2835_PAD_* from **bcm2835PadGroup**

Definition at line 426 of file bcm2835.c.

void bcm2835_gpio_pud (uint8_t *pud*)

Sets the Pull-up/down register for the given pin. This is used with **bcm2835_gpio_pudclk()** to set the Pull-up/down resistor for the given pin. However, it is usually more convenient to use **bcm2835_gpio_set_pud()**.

Parameters:

in	*pud*	The desired Pull-up/down mode. One of BCM2835_GPIO_PUD_* from bcm2835PUDControl

See Also:

bcm2835_gpio_set_pud()

Definition at line 409 of file bcm2835.c.

void bcm2835_gpio_pudclk (uint8_t *pin*, uint8_t *on*)

Clocks the Pull-up/down value set earlier by **bcm2835_gpio_pud()** into the pin.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.
in	*on*	HIGH to clock the value from **bcm2835_gpio_pud()** into the pin. LOW to remove the clock.

See Also:

bcm2835_gpio_set_pud()

Definition at line 418 of file bcm2835.c.

void bcm2835_gpio_ren (uint8_t *pin*)

Enable Rising Edge Detect Enable for the specified pin. When a rising edge is detected, sets the appropriate pin in Event Detect Status. The GPRENn registers use synchronous edge detection. This means the input signal is sampled using the system clock and then it is looking for a ?011? pattern on the sampled signal. This has the effect of suppressing glitches.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

Definition at line 313 of file bcm2835.c.

void bcm2835_gpio_set (uint8_t *pin*)

Sets the specified pin output to HIGH.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

See Also:

bcm2835_gpio_write()

Definition at line 241 of file bcm2835.c.

void bcm2835_gpio_set_eds (uint8_t *pin*)

Sets the Event Detect Status register for a given pin to 1, which has the effect of clearing the flag. Use this afer seeing an Event Detect Status on the pin.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

Definition at line 298 of file bcm2835.c.

void bcm2835_gpio_set_eds_multi (uint32_t *mask*)

Same as **bcm2835_gpio_set_eds()** but clears the flag for any pin which is set in the mask.

Parameters:

in	*mask*	Mask of pins to clear. Use eg: (1 << RPI_GPIO_P1_03) | (1 << RPI_GPIO_P1_05)

Definition at line 306 of file bcm2835.c.

void bcm2835_gpio_set_multi (uint32_t *mask*)

Sets any of the first 32 GPIO output pins specified in the mask to HIGH.

Parameters:

in	*mask*	Mask of pins to affect. Use eg: (1 << RPI_GPIO_P1_03) | (1 << RPI_GPIO_P1_05)

See Also:

bcm2835_gpio_write_multi()

Definition at line 257 of file bcm2835.c.

void bcm2835_gpio_set_pad (uint8_t *group*, uint32_t *control*)

Sets the Pad Control for the given GPIO group.

Parameters:

in	*group*	The GPIO pad group number, one of BCM2835_PAD_GROUP_GPIO_*
in	*control*	Mask of bits from BCM2835_PAD_* from **bcm2835PadGroup**. Note that it is not necessary to include BCM2835_PAD_PASSWRD in the mask as this is automatically included.

Definition at line 438 of file bcm2835.c.

void bcm2835_gpio_set_pud (uint8_t *pin*, uint8_t *pud*)

Sets the Pull-up/down mode for the specified pin. This is more convenient than clocking the mode in with **bcm2835_gpio_pud()** and **bcm2835_gpio_pudclk()**.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.
in	*pud*	The desired Pull-up/down mode. One of BCM2835_GPIO_PUD_* from bcm2835PUDControl

Definition at line 533 of file bcm2835.c.

void bcm2835_gpio_write (uint8_t *pin*, uint8_t *on*)

Sets the output state of the specified pin

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.
in	*on*	HIGH sets the output to HIGH and LOW to LOW.

Definition at line 491 of file bcm2835.c.

void bcm2835_gpio_write_mask (uint32_t *value*, uint32_t *mask*)

Sets the first 32 GPIO output pins specified in the mask to the value given by value

Parameters:

in	*value*	values required for each bit masked in by mask, eg: (1 << RPI_GPIO_P1_03) | (1 << RPI_GPIO_P1_05)
in	*mask*	Mask of pins to affect. Use eg: (1 << RPI_GPIO_P1_03) | (1 << RPI_GPIO_P1_05)

Definition at line 509 of file bcm2835.c.

void bcm2835_gpio_write_multi (uint32_t *mask*, uint8_t *on*)

Sets any of the first 32 GPIO output pins specified in the mask to the state given by on

Parameters:

in	*mask*	Mask of pins to affect. Use eg: (1 << RPI_GPIO_P1_03) | (1 << RPI_GPIO_P1_05)
in	*on*	HIGH sets the output to HIGH and LOW to LOW.

Definition at line 500 of file bcm2835.c.

int bcm2835_init (void)

Library initialisation and management These functions allow you to intialise and control the bcm2835 library

Initialise the library by opening /dev/mem (if you are root) or /dev/gpiomem (if you are not) and getting pointers to the internal memory for BCM 2835 device registers. You must call this (successfully) before calling any other functions in this library (except bcm2835_set_debug). If **bcm2835_init()** fails by returning 0, calling any other function may result in crashes or other failures. If **bcm2835_init()** succeeds but you are not running as root, then only gpio operations are permitted, and calling any other functions may result in crashes or other failures. . Prints messages to stderr in case of errors.

Returns:

1 if successful else 0

Definition at line 1282 of file bcm2835.c.

uint32_t bcm2835_peri_read (volatile uint32_t * *paddr*)

Reads 32 bit value from a peripheral address WITH a memory barrier before and after each read. This is safe, but slow. The MB before protects this read from any in-flight reads that didn't use a MB. The MB after protects subsequent reads from another peripheral.

Parameters:

in	*paddr*	Physical address to read from. See BCM2835_GPIO_BASE etc.

Returns:

the value read from the 32 bit register

See Also:

Physical Addresses

Definition at line 131 of file bcm2835.c.

uint32_t bcm2835_peri_read_nb (volatile uint32_t * *paddr*)

Reads 32 bit value from a peripheral address WITHOUT the read barriers You should only use this when: o your code has previously called **bcm2835_peri_read()** for a register within the same peripheral, and no read or write to another peripheral has occurred since. o your code has called bcm2835_memory_barrier() since the last access to ANOTHER peripheral.

Parameters:

in	*paddr*	Physical address to read from. See BCM2835_GPIO_BASE etc.

Returns:

the value read from the 32 bit register

See Also:

Physical Addresses

Definition at line 154 of file bcm2835.c.

void bcm2835_peri_set_bits (volatile uint32_t * *paddr*, uint32_t *value*, uint32_t *mask*)

Alters a number of bits in a 32 peripheral regsiter. It reads the current valu and then alters the bits defines as 1 in mask, according to the bit value in value. All other bits that are 0 in the mask are unaffected. Use this to alter a subset of the bits in a register. Memory barriers are used. Note that this is not atomic; an interrupt routine can cause unexpected results.

Parameters:

in	*paddr*	Physical address to read from. See BCM2835_GPIO_BASE etc.
in	*value*	The 32 bit value to write, masked in by mask.
in	*mask*	Bitmask that defines the bits that will be altered in the register.

See Also:

Physical Addresses

Definition at line 201 of file bcm2835.c.

void bcm2835_peri_write (volatile uint32_t * *paddr*, uint32_t *value*)

Writes 32 bit value from a peripheral address WITH a memory barrier before and after each write This is safe, but slow. The MB before ensures that any in-flight write to another peripheral completes before this write is issued. The MB after ensures that subsequent reads and writes to another peripheral will see the effect of this write.

This is a tricky optimization; if you aren't sure, use the barrier version.

Parameters:

in	*paddr*	Physical address to read from. See BCM2835_GPIO_BASE etc.
in	*value*	The 32 bit value to write

See Also:

Physical Addresses

Definition at line 170 of file bcm2835.c.

void bcm2835_peri_write_nb (volatile uint32_t * *paddr*, uint32_t *value*)

Writes 32 bit value from a peripheral address without the write barrier You should only use this when: o your code has previously called **bcm2835_peri_write()** for a register within the same peripheral, and no other peripheral access has occurred since. o your code has called bcm2835_memory_barrier() since the last access to ANOTHER peripheral.

This is a tricky optimization; if you aren't sure, use the barrier version.

Parameters:

in	*paddr*	Physical address to read from. See BCM2835_GPIO_BASE etc.
in	*value*	The 32 bit value to write

See Also:

Physical Addresses

Definition at line 185 of file bcm2835.c.

void bcm2835_pwm_set_clock (uint32_t *divisor*)

Pulse Width Modulation Allows control of 2 independent PWM channels. A limited subset of GPIO pins can be connected to one of these 2 channels, allowing PWM control of GPIO pins. You have to set the desired pin into a particular Alt Fun to PWM output. See the PWM documentation on the Main Page.

Sets the PWM clock divisor, to control the basic PWM pulse widths.

Parameters:

in	*divisor*	Divides the basic 19.2MHz PWM clock. You can use one of the common values BCM2835_PWM_CLOCK_DIVIDER_* in **bcm2835PWMClockDivider**

Definition at line 1208 of file bcm2835.c.

void bcm2835_pwm_set_data (uint8_t *channel*, uint32_t *data*)

Sets the PWM pulse ratio to emit to DATA/RANGE, where RANGE is set by **bcm2835_pwm_set_range()**.

Parameters:

in	*channel*	The PWM channel. 0 or 1.
in	*data*	Controls the PWM output ratio as a fraction of the range. Can vary from 0 to RANGE.

Definition at line 1270 of file bcm2835.c.

void bcm2835_pwm_set_mode (uint8_t *channel*, uint8_t *markspace*, uint8_t *enabled*)

Sets the mode of the given PWM channel, allowing you to control the PWM mode and enable/disable that channel

Parameters:

in	*channel*	The PWM channel. 0 or 1.
in	*markspace*	Set true if you want Mark-Space mode. 0 for Balanced mode.
in	*enabled*	Set true to enable this channel and produce PWM pulses.

Definition at line 1225 of file bcm2835.c.

void bcm2835_pwm_set_range (uint8_t *channel*, uint32_t *range*)

Sets the maximum range of the PWM output. The data value can vary between 0 and this range to control PWM output

Parameters:

in	*channel*	The PWM channel. 0 or 1.
in	*range*	The maximum value permitted for DATA.

Definition at line 1260 of file bcm2835.c.

uint32_t* bcm2835_regbase (uint8_t *regbase*)

Low level register access These functions provide low level register access, and should not generally need to be used

Gets the base of a register

Parameters:

in	*regbase*	You can use one of the common values BCM2835_REGBASE_* in **bcm2835RegisterBase**

Returns:

the register base

See Also:

Physical Addresses

Definition at line 94 of file bcm2835.c.

void bcm2835_set_debug (uint8_t *debug*)

Sets the debug level of the library. A value of 1 prevents mapping to /dev/mem, and makes the library print out what it would do, rather than accessing the GPIO registers. A value of 0, the default, causes normal operation. Call this before calling **bcm2835_init()**;

Parameters:

in	*debug*	The new debug level. 1 means debug

Definition at line 118 of file bcm2835.c.

int bcm2835_spi_begin (void)

SPI access These functions let you use SPI0 (Serial Peripheral Interface) to interface with an external SPI device.

Start SPI operations. Forces RPi SPI0 pins P1-19 (MOSI), P1-21 (MISO), P1-23 (CLK), P1-24 (CE0) and P1-26 (CE1) to alternate function ALT0, which enables those pins for SPI interface. You should call **bcm2835_spi_end()** when all SPI funcitons are complete to return the pins to their default functions.

See Also:

bcm2835_spi_end()

Returns:

1 if successful, 0 otherwise (perhaps because you are not running as root)

Definition at line 543 of file bcm2835.c.

void bcm2835_spi_chipSelect (uint8_t *cs*)

Sets the chip select pin(s) When an **bcm2835_spi_transfer()** is made, the selected pin(s) will be asserted during the transfer.

Parameters:

in	*cs*	Specifies the CS pins(s) that are used to activate the desired slave. One of BCM2835_SPI_CS*, see **bcm2835SPIChipSelect**

Definition at line 730 of file bcm2835.c.

void bcm2835_spi_end (void)

End SPI operations. SPI0 pins P1-19 (MOSI), P1-21 (MISO), P1-23 (CLK), P1-24 (CE0) and P1-26 (CE1) are returned to their default INPUT behaviour.

Definition at line 566 of file bcm2835.c.

void bcm2835_spi_setChipSelectPolarity (uint8_t *cs*, uint8_t *active*)

Sets the chip select pin polarity for a given pin When an **bcm2835_spi_transfer()** occurs, the currently selected chip select pin(s) will be asserted to the value given by active. When transfers are not happening, the chip select pin(s) return to the complement (inactive) value.

Parameters:

in	*cs*	The chip select pin to affect
in	*active*	Whether the chip select pin is to be active HIGH

Definition at line 737 of file bcm2835.c.

void bcm2835_spi_setClockDivider (uint16_t *divider*)

Sets the SPI clock divider and therefore the SPI clock speed.

Parameters:

in	*divider*	The desired SPI clock divider, one of BCM2835_SPI_CLOCK_DIVIDER_*, see **bcm2835SPIClockDivider**

Definition at line 586 of file bcm2835.c.

void bcm2835_spi_setDataMode (uint8_t *mode*)

Sets the SPI data mode Sets the clock polariy and phase

Parameters:

in	*mode*	The desired data mode, one of BCM2835_SPI_MODE*, see **bcm2835SPIMode**

Definition at line 592 of file bcm2835.c.

uint8_t bcm2835_spi_transfer (uint8_t *value*)

Transfers one byte to and from the currently selected SPI slave. Asserts the currently selected CS pins (as previously set by bcm2835_spi_chipSelect) during the transfer. Clocks the 8 bit value out on MOSI, and simultaneously clocks in data from MISO. Returns the read data byte from the slave. Uses polled transfer as per section 10.6.1 of the BCM 2835 ARM Peripherls manual

Parameters:

in	*value*	The 8 bit data byte to write to MOSI

Returns:

The 8 bit byte simultaneously read from MISO

See Also:

bcm2835_spi_transfern()

Definition at line 600 of file bcm2835.c.

void bcm2835_spi_transfern (char * *buf*, uint32_t *len*)

Transfers any number of bytes to and from the currently selected SPI slave using bcm2835_spi_transfernb. The returned data from the slave replaces the transmitted data in the buffer.

Parameters:

in,out	*buf*	Buffer of bytes to send. Received bytes will replace the contents
in	*len*	Number of bytes int eh buffer, and the number of bytes to send/received

See Also:

bcm2835_spi_transfer()

Definition at line 725 of file bcm2835.c.

void bcm2835_spi_transfernb (char * *tbuf*, char * *rbuf*, uint32_t *len*)

Transfers any number of bytes to and from the currently selected SPI slave. Asserts the currently selected CS pins (as previously set by bcm2835_spi_chipSelect) during the transfer. Clocks the len 8 bit bytes out on MOSI, and simultaneously clocks in data from MISO. The data read read from the slave is placed into rbuf. rbuf must be at least len bytes long Uses polled transfer as per section 10.6.1 of the BCM 2835 ARM Peripherls manual

Parameters:

in	*tbuf*	Buffer of bytes to send.
out	*rbuf*	Received bytes will by put in this buffer
in	*len*	Number of bytes in the tbuf buffer, and the number of bytes to

		send/received

See Also:

bcm2835_spi_transfer()

Definition at line 637 of file bcm2835.c.

void bcm2835_spi_writenb (char * *buf*, uint32_t *len*)

Transfers any number of bytes to the currently selected SPI slave. Asserts the currently selected CS pins (as previously set by bcm2835_spi_chipSelect) during the transfer.

Parameters:

in	*buf*	Buffer of bytes to send.
in	*len*	Number of bytes in the tbuf buffer, and the number of bytes to send

Definition at line 680 of file bcm2835.c.

void bcm2835_st_delay (uint64_t *offset_micros*, uint64_t *micros*)

Delays for the specified number of microseconds with offset.

Parameters:

in	*offset_micros*	Offset in microseconds
in	*micros*	Delay in microseconds

Definition at line 1198 of file bcm2835.c.

uint64_t bcm2835_st_read (void)

System Timer access Allows access to and delays using the System Timer Counter.

Read the System Timer Counter register.

Returns:

the value read from the System Timer Counter Lower 32 bits register

Definition at line 1168 of file bcm2835.c.

unsigned int bcm2835_version (void)

Returns the version number of the library, same as BCM2835_VERSION

Returns:

the current library version number

Definition at line 123 of file bcm2835.c.

Variable Documentation

volatile uint32_t* bcm2835_bsc0 = (uint32_t *)MAP_FAILED

Base of the BSC0 registers. Available after bcm2835_init has been called (as root)

Definition at line 74 of file bcm2835.c.

volatile uint32_t* bcm2835_bsc1 = (uint32_t *)MAP_FAILED

Base of the BSC1 registers. Available after bcm2835_init has been called (as root)

Definition at line 75 of file bcm2835.c.

volatile uint32_t* bcm2835_clk = (uint32_t *)MAP_FAILED

Base of the CLK registers. Available after bcm2835_init has been called (as root)

Definition at line 71 of file bcm2835.c.

volatile uint32_t* bcm2835_gpio = (uint32_t *)MAP_FAILED

Base of the GPIO registers. Available after bcm2835_init has been called

Definition at line 69 of file bcm2835.c.

volatile uint32_t* bcm2835_pads = (uint32_t *)MAP_FAILED

Base of the PADS registers. Available after bcm2835_init has been called (as root)

Definition at line 72 of file bcm2835.c.

uint32_t* bcm2835_peripherals = (uint32_t *)MAP_FAILED

Virtual memory address of the mapped peripherals block

Definition at line 65 of file bcm2835.c.

uint32_t* bcm2835_peripherals_base = (uint32_t *)BCM2835_PERI_BASE

Physical address and size of the peripherals block May be overridden on RPi2

Definition at line 60 of file bcm2835.c.

uint32_t bcm2835_peripherals_size = BCM2835_PERI_SIZE

Size of the peripherals block to be mapped

Definition at line 61 of file bcm2835.c.

volatile uint32_t* bcm2835_pwm = (uint32_t *)MAP_FAILED

Base of the PWM registers. Available after bcm2835_init has been called (as root)

Definition at line 70 of file bcm2835.c.

volatile uint32_t* bcm2835_spi0 = (uint32_t *)MAP_FAILED

Base of the SPI0 registers. Available after bcm2835_init has been called (as root)

Definition at line 73 of file bcm2835.c.

volatile uint32_t* bcm2835_st = (uint32_t *)MAP_FAILED

Base of the ST (System Timer) registers. Available after bcm2835_init has been called (as root)

Definition at line 76 of file bcm2835.c.

bcm2835.h File Reference

C library for Broadcom BCM 2835 as used in Raspberry Pi.

```
#include <stdint.h>
```

Macros

#define **BCM2835_VERSION** 10050 /* Version 1.50 */
#define **BCM2835_PWM_CONTROL** 0
#define **BCM2835_PWM_STATUS** 1
#define **BCM2835_PWM_DMAC** 2
#define **BCM2835_PWM0_RANGE** 4
#define **BCM2835_PWM0_DATA** 5
#define **BCM2835_PWM_FIF1** 6
#define **BCM2835_PWM1_RANGE** 8
#define **BCM2835_PWM1_DATA** 9
#define **BCM2835_PWMCLK_CNTL** 40
#define **BCM2835_PWMCLK_DIV** 41
#define **BCM2835_PWM_PASSWRD** (0x5A << 24)
#define **BCM2835_PWM1_MS_MODE** 0x8000
#define **BCM2835_PWM1_USEFIFO** 0x2000
#define **BCM2835_PWM1_REVPOLAR** 0x1000
#define **BCM2835_PWM1_OFFSTATE** 0x0800
#define **BCM2835_PWM1_REPEATFF** 0x0400
#define **BCM2835_PWM1_SERIAL** 0x0200
#define **BCM2835_PWM1_ENABLE** 0x0100
#define **BCM2835_PWM0_MS_MODE** 0x0080
#define **BCM2835_PWM_CLEAR_FIFO** 0x0040
#define **BCM2835_PWM0_USEFIFO** 0x0020
#define **BCM2835_PWM0_REVPOLAR** 0x0010
#define **BCM2835_PWM0_OFFSTATE** 0x0008
#define **BCM2835_PWM0_REPEATFF** 0x0004
#define **BCM2835_PWM0_SERIAL** 0x0002
#define **BCM2835_PWM0_ENABLE** 0x0001
#define **delay**(x) **bcm2835_delay**(x)
#define **delayMicroseconds**(x) **bcm2835_delayMicroseconds**(x)

Enumerations

enum **bcm2835PWMClockDivider** { **BCM2835_PWM_CLOCK_DIVIDER_2048** = 2048, **BCM2835_PWM_CLOCK_DIVIDER_1024** = 1024, **BCM2835_PWM_CLOCK_DIVIDER_512** = 512, **BCM2835_PWM_CLOCK_DIVIDER_256** = 256, **BCM2835_PWM_CLOCK_DIVIDER_128** = 128, **BCM2835_PWM_CLOCK_DIVIDER_64** = 64, **BCM2835_PWM_CLOCK_DIVIDER_32** = 32, **BCM2835_PWM_CLOCK_DIVIDER_16** = 16, **BCM2835_PWM_CLOCK_DIVIDER_8** = 8, **BCM2835_PWM_CLOCK_DIVIDER_4** = 4, **BCM2835_PWM_CLOCK_DIVIDER_2** = 2, **BCM2835_PWM_CLOCK_DIVIDER_1** = 1 }

bcm2835PWMClockDivider Specifies the divider used to generate the PWM clock from the system clock. Figures below give the divider, clock period and clock frequency. Clock divided is based on nominal PWM base clock rate of 19.2MHz The frequencies shown for each divider have been confirmed by measurement Functions

int **bcm2835_i2c_begin** (void)
void **bcm2835_i2c_end** (void)

void **bcm2835_i2c_setSlaveAddress** (uint8_t addr)
void **bcm2835_i2c_setClockDivider** (uint16_t divider)
void **bcm2835_i2c_set_baudrate** (uint32_t baudrate)
uint8_t **bcm2835_i2c_write** (const char *buf, uint32_t len)
uint8_t **bcm2835_i2c_read** (char *buf, uint32_t len)
uint8_t **bcm2835_i2c_read_register_rs** (char *regaddr, char *buf, uint32_t len)
uint8_t **bcm2835_i2c_write_read_rs** (char *cmds, uint32_t cmds_len, char *buf, uint32_t buf_len)

int **bcm2835_init** (void)
int **bcm2835_close** (void)
void **bcm2835_set_debug** (uint8_t debug)
unsigned int **bcm2835_version** (void)

uint32_t * **bcm2835_regbase** (uint8_t regbase)
uint32_t **bcm2835_peri_read** (volatile uint32_t *paddr)
uint32_t **bcm2835_peri_read_nb** (volatile uint32_t *paddr)
void **bcm2835_peri_write** (volatile uint32_t *paddr, uint32_t value)
void **bcm2835_peri_write_nb** (volatile uint32_t *paddr, uint32_t value)
void **bcm2835_peri_set_bits** (volatile uint32_t *paddr, uint32_t value, uint32_t mask)

void **bcm2835_gpio_fsel** (uint8_t pin, uint8_t mode)
void **bcm2835_gpio_set** (uint8_t pin)
void **bcm2835_gpio_clr** (uint8_t pin)
void **bcm2835_gpio_set_multi** (uint32_t mask)
void **bcm2835_gpio_clr_multi** (uint32_t mask)
uint8_t **bcm2835_gpio_lev** (uint8_t pin)
uint8_t **bcm2835_gpio_eds** (uint8_t pin)
uint32_t **bcm2835_gpio_eds_multi** (uint32_t mask)
void **bcm2835_gpio_set_eds** (uint8_t pin)
void **bcm2835_gpio_set_eds_multi** (uint32_t mask)
void **bcm2835_gpio_ren** (uint8_t pin)
void **bcm2835_gpio_clr_ren** (uint8_t pin)
void **bcm2835_gpio_fen** (uint8_t pin)
void **bcm2835_gpio_clr_fen** (uint8_t pin)
void **bcm2835_gpio_hen** (uint8_t pin)
void **bcm2835_gpio_clr_hen** (uint8_t pin)
void **bcm2835_gpio_len** (uint8_t pin)
void **bcm2835_gpio_clr_len** (uint8_t pin)
void **bcm2835_gpio_aren** (uint8_t pin)
void **bcm2835_gpio_clr_aren** (uint8_t pin)
void **bcm2835_gpio_afen** (uint8_t pin)
void **bcm2835_gpio_clr_afen** (uint8_t pin)
void **bcm2835_gpio_pud** (uint8_t pud)
void **bcm2835_gpio_pudclk** (uint8_t pin, uint8_t on)
uint32_t **bcm2835_gpio_pad** (uint8_t group)
void **bcm2835_gpio_set_pad** (uint8_t group, uint32_t control)
void **bcm2835_delay** (unsigned int millis)
void **bcm2835_delayMicroseconds** (uint64_t micros)
void **bcm2835_gpio_write** (uint8_t pin, uint8_t on)
void **bcm2835_gpio_write_multi** (uint32_t mask, uint8_t on)
void **bcm2835_gpio_write_mask** (uint32_t value, uint32_t mask)
void **bcm2835_gpio_set_pud** (uint8_t pin, uint8_t pud)

int **bcm2835_spi_begin** (void)
void **bcm2835_spi_end** (void)
void **bcm2835_spi_setBitOrder** (uint8_t order)
void **bcm2835_spi_setClockDivider** (uint16_t divider)
void **bcm2835_spi_setDataMode** (uint8_t mode)
void **bcm2835_spi_chipSelect** (uint8_t cs)
void **bcm2835_spi_setChipSelectPolarity** (uint8_t cs, uint8_t active)
uint8_t **bcm2835_spi_transfer** (uint8_t value)
void **bcm2835_spi_transfernb** (char *tbuf, char *rbuf, uint32_t len)
void **bcm2835_spi_transfern** (char *buf, uint32_t len)
void **bcm2835_spi_writenb** (char *buf, uint32_t len)

uint64_t **bcm2835_st_read** (void)
void **bcm2835_st_delay** (uint64_t offset_micros, uint64_t micros)

void **bcm2835_pwm_set_clock** (uint32_t divisor)
void **bcm2835_pwm_set_mode** (uint8_t channel, uint8_t markspace, uint8_t enabled)
void **bcm2835_pwm_set_range** (uint8_t channel, uint32_t range)
void **bcm2835_pwm_set_data** (uint8_t channel, uint32_t data)
#define **HIGH** 0x1
#define **LOW** 0x0
#define **BCM2835_CORE_CLK_HZ** 250000000
#define **BMC2835_RPI2_DT_FILENAME** "/proc/device-tree/soc/ranges"
#define **BMC2835_RPI2_DT_PERI_BASE_ADDRESS_OFFSET** 4
#define **BMC2835_RPI2_DT_PERI_SIZE_OFFSET** 8
#define **BCM2835_PERI_BASE** 0x20000000
#define **BCM2835_PERI_SIZE** 0x01000000
#define **BCM2835_ST_BASE** 0x3000
#define **BCM2835_GPIO_PADS** 0x100000
#define **BCM2835_CLOCK_BASE** 0x101000
#define **BCM2835_GPIO_BASE** 0x200000
#define **BCM2835_SPI0_BASE** 0x204000
#define **BCM2835_BSC0_BASE** 0x205000
#define **BCM2835_GPIO_PWM** 0x20C000
#define **BCM2835_BSC1_BASE** 0x804000
#define **BCM2835_PAGE_SIZE** (4*1024)
#define **BCM2835_BLOCK_SIZE** (4*1024)
#define **BCM2835_GPFSEL0** 0x0000
#define **BCM2835_GPFSEL1** 0x0004
#define **BCM2835_GPFSEL2** 0x0008
#define **BCM2835_GPFSEL3** 0x000c
#define **BCM2835_GPFSEL4** 0x0010
#define **BCM2835_GPFSEL5** 0x0014
#define **BCM2835_GPSET0** 0x001c
#define **BCM2835_GPSET1** 0x0020
#define **BCM2835_GPCLR0** 0x0028
#define **BCM2835_GPCLR1** 0x002c
#define **BCM2835_GPLEV0** 0x0034
#define **BCM2835_GPLEV1** 0x0038
#define **BCM2835_GPEDS0** 0x0040
#define **BCM2835_GPEDS1** 0x0044
#define **BCM2835_GPREN0** 0x004c
#define **BCM2835_GPREN1** 0x0050

#define **BCM2835_GPFEN0** 0x0058
#define **BCM2835_GPFEN1** 0x005c
#define **BCM2835_GPHEN0** 0x0064
#define **BCM2835_GPHEN1** 0x0068
#define **BCM2835_GPLEN0** 0x0070
#define **BCM2835_GPLEN1** 0x0074
#define **BCM2835_GPAREN0** 0x007c
#define **BCM2835_GPAREN1** 0x0080
#define **BCM2835_GPAFEN0** 0x0088
#define **BCM2835_GPAFEN1** 0x008c
#define **BCM2835_GPPUD** 0x0094
#define **BCM2835_GPPUDCLK0** 0x0098
#define **BCM2835_GPPUDCLK1** 0x009c
#define **BCM2835_PADS_GPIO_0_27** 0x002c
#define **BCM2835_PADS_GPIO_28_45** 0x0030
#define **BCM2835_PADS_GPIO_46_53** 0x0034
#define **BCM2835_PAD_PASSWRD** (0x5A << 24)
#define **BCM2835_PAD_SLEW_RATE_UNLIMITED** 0x10
#define **BCM2835_PAD_HYSTERESIS_ENABLED** 0x08
#define **BCM2835_PAD_DRIVE_2mA** 0x00
#define **BCM2835_PAD_DRIVE_4mA** 0x01
#define **BCM2835_PAD_DRIVE_6mA** 0x02
#define **BCM2835_PAD_DRIVE_8mA** 0x03
#define **BCM2835_PAD_DRIVE_10mA** 0x04
#define **BCM2835_PAD_DRIVE_12mA** 0x05
#define **BCM2835_PAD_DRIVE_14mA** 0x06
#define **BCM2835_PAD_DRIVE_16mA** 0x07
#define **BCM2835_SPI0_CS** 0x0000
#define **BCM2835_SPI0_FIFO** 0x0004
#define **BCM2835_SPI0_CLK** 0x0008
#define **BCM2835_SPI0_DLEN** 0x000c
#define **BCM2835_SPI0_LTOH** 0x0010
#define **BCM2835_SPI0_DC** 0x0014
#define **BCM2835_SPI0_CS_LEN_LONG** 0x02000000
#define **BCM2835_SPI0_CS_DMA_LEN** 0x01000000
#define **BCM2835_SPI0_CS_CSPOL2** 0x00800000
#define **BCM2835_SPI0_CS_CSPOL1** 0x00400000
#define **BCM2835_SPI0_CS_CSPOL0** 0x00200000
#define **BCM2835_SPI0_CS_RXF** 0x00100000
#define **BCM2835_SPI0_CS_RXR** 0x00080000
#define **BCM2835_SPI0_CS_TXD** 0x00040000
#define **BCM2835_SPI0_CS_RXD** 0x00020000
#define **BCM2835_SPI0_CS_DONE** 0x00010000
#define **BCM2835_SPI0_CS_TE_EN** 0x00008000
#define **BCM2835_SPI0_CS_LMONO** 0x00004000
#define **BCM2835_SPI0_CS_LEN** 0x00002000
#define **BCM2835_SPI0_CS_REN** 0x00001000
#define **BCM2835_SPI0_CS_ADCS** 0x00000800
#define **BCM2835_SPI0_CS_INTR** 0x00000400
#define **BCM2835_SPI0_CS_INTD** 0x00000200
#define **BCM2835_SPI0_CS_DMAEN** 0x00000100
#define **BCM2835_SPI0_CS_TA** 0x00000080
#define **BCM2835_SPI0_CS_CSPOL** 0x00000040
#define **BCM2835_SPI0_CS_CLEAR** 0x00000030
#define **BCM2835_SPI0_CS_CLEAR_RX** 0x00000020
#define **BCM2835_SPI0_CS_CLEAR_TX** 0x00000010

#define **BCM2835_SPI0_CS_CPOL** 0x00000008
#define **BCM2835_SPI0_CS_CPHA** 0x00000004
#define **BCM2835_SPI0_CS_CS** 0x00000003
#define **BCM2835_BSC_C** 0x0000
#define **BCM2835_BSC_S** 0x0004
#define **BCM2835_BSC_DLEN** 0x0008
#define **BCM2835_BSC_A** 0x000c
#define **BCM2835_BSC_FIFO** 0x0010
#define **BCM2835_BSC_DIV** 0x0014
#define **BCM2835_BSC_DEL** 0x0018
#define **BCM2835_BSC_CLKT** 0x001c
#define **BCM2835_BSC_C_I2CEN** 0x00008000
#define **BCM2835_BSC_C_INTR** 0x00000400
#define **BCM2835_BSC_C_INTT** 0x00000200
#define **BCM2835_BSC_C_INTD** 0x00000100
#define **BCM2835_BSC_C_ST** 0x00000080
#define **BCM2835_BSC_C_CLEAR_1** 0x00000020
#define **BCM2835_BSC_C_CLEAR_2** 0x00000010
#define **BCM2835_BSC_C_READ** 0x00000001
#define **BCM2835_BSC_S_CLKT** 0x00000200
#define **BCM2835_BSC_S_ERR** 0x00000100
#define **BCM2835_BSC_S_RXF** 0x00000080
#define **BCM2835_BSC_S_TXE** 0x00000040
#define **BCM2835_BSC_S_RXD** 0x00000020
#define **BCM2835_BSC_S_TXD** 0x00000010
#define **BCM2835_BSC_S_RXR** 0x00000008
#define **BCM2835_BSC_S_TXW** 0x00000004
#define **BCM2835_BSC_S_DONE** 0x00000002
#define **BCM2835_BSC_S_TA** 0x00000001
#define **BCM2835_BSC_FIFO_SIZE** 16
#define **BCM2835_ST_CS** 0x0000
#define **BCM2835_ST_CLO** 0x0004
#define **BCM2835_ST_CHI** 0x0008

enum **bcm2835RegisterBase** { **BCM2835_REGBASE_ST** = 1, **BCM2835_REGBASE_GPIO** = 2,
BCM2835_REGBASE_PWM = 3, **BCM2835_REGBASE_CLK** = 4,
BCM2835_REGBASE_PADS = 5, **BCM2835_REGBASE_SPI0** = 6,
BCM2835_REGBASE_BSC0 = 7, **BCM2835_REGBASE_BSC1** = 8 }

bcm2835RegisterBase Register bases for ***bcm2835_regbase()*** enum **bcm2835FunctionSelect** { **BCM2835_GPIO_FSEL_INPT** = 0x00, **BCM2835_GPIO_FSEL_OUTP** = 0x01, **BCM2835_GPIO_FSEL_ALT0** = 0x04, **BCM2835_GPIO_FSEL_ALT1** = 0x05, **BCM2835_GPIO_FSEL_ALT2** = 0x06, **BCM2835_GPIO_FSEL_ALT3** = 0x07, **BCM2835_GPIO_FSEL_ALT4** = 0x03, **BCM2835_GPIO_FSEL_ALT5** = 0x02, **BCM2835_GPIO_FSEL_MASK** = 0x07 }

bcm2835PortFunction Port function select modes for ***bcm2835_gpio_fsel()*** enum **bcm2835PUDControl** { **BCM2835_GPIO_PUD_OFF** = 0x00, **BCM2835_GPIO_PUD_DOWN** = 0x01, **BCM2835_GPIO_PUD_UP** = 0x02 }

bcm2835PUDControl Pullup/Pulldown defines for ***bcm2835_gpio_pud()*** enum **bcm2835PadGroup** { **BCM2835_PAD_GROUP_GPIO_0_27** = 0, **BCM2835_PAD_GROUP_GPIO_28_45** = 1, **BCM2835_PAD_GROUP_GPIO_46_53** = 2 }

bcm2835PadGroup Pad group specification for ***bcm2835_gpio_pad()*** enum **RPiGPIOPin** { **RPI_GPIO_P1_03** = 0, **RPI_GPIO_P1_05** = 1, **RPI_GPIO_P1_07** = 4, **RPI_GPIO_P1_08** = 14, **RPI_GPIO_P1_10** = 15, **RPI_GPIO_P1_11** = 17, **RPI_GPIO_P1_12** = 18, **RPI_GPIO_P1_13** = 21, **RPI_GPIO_P1_15** = 22, **RPI_GPIO_P1_16** = 23, **RPI_GPIO_P1_18** = 24, **RPI_GPIO_P1_19** = 10, **RPI_GPIO_P1_21** = 9, **RPI_GPIO_P1_22** = 25, **RPI_GPIO_P1_23** = 11, **RPI_GPIO_P1_24** = 8, **RPI_GPIO_P1_26** = 7, **RPI_V2_GPIO_P1_03** = 2, **RPI_V2_GPIO_P1_05** = 3, **RPI_V2_GPIO_P1_07** = 4, **RPI_V2_GPIO_P1_08** = 14, **RPI_V2_GPIO_P1_10** = 15, **RPI_V2_GPIO_P1_11** = 17, **RPI_V2_GPIO_P1_12** = 18, **RPI_V2_GPIO_P1_13** = 27, **RPI_V2_GPIO_P1_15** = 22, **RPI_V2_GPIO_P1_16** = 23, **RPI_V2_GPIO_P1_18** = 24, **RPI_V2_GPIO_P1_19** = 10, **RPI_V2_GPIO_P1_21** = 9, **RPI_V2_GPIO_P1_22** = 25, **RPI_V2_GPIO_P1_23** = 11, **RPI_V2_GPIO_P1_24** = 8, **RPI_V2_GPIO_P1_26** = 7, **RPI_V2_GPIO_P1_29** = 5, **RPI_V2_GPIO_P1_31** = 6, **RPI_V2_GPIO_P1_32** = 12, **RPI_V2_GPIO_P1_33** = 13, **RPI_V2_GPIO_P1_35** = 19, **RPI_V2_GPIO_P1_36** = 16, **RPI_V2_GPIO_P1_37** = 26, **RPI_V2_GPIO_P1_38** = 20, **RPI_V2_GPIO_P1_40** = 21, **RPI_V2_GPIO_P5_03** = 28, **RPI_V2_GPIO_P5_04** = 29, **RPI_V2_GPIO_P5_05** = 30, **RPI_V2_GPIO_P5_06** = 31, **RPI_BPLUS_GPIO_J8_03** = 2, **RPI_BPLUS_GPIO_J8_05** = 3, **RPI_BPLUS_GPIO_J8_07** = 4, **RPI_BPLUS_GPIO_J8_08** = 14, **RPI_BPLUS_GPIO_J8_10** = 15, **RPI_BPLUS_GPIO_J8_11** = 17, **RPI_BPLUS_GPIO_J8_12** = 18, **RPI_BPLUS_GPIO_J8_13** = 27, **RPI_BPLUS_GPIO_J8_15** = 22, **RPI_BPLUS_GPIO_J8_16** = 23, **RPI_BPLUS_GPIO_J8_18** = 24, **RPI_BPLUS_GPIO_J8_19** = 10, **RPI_BPLUS_GPIO_J8_21** = 9, **RPI_BPLUS_GPIO_J8_22** = 25, **RPI_BPLUS_GPIO_J8_23** = 11, **RPI_BPLUS_GPIO_J8_24** = 8, **RPI_BPLUS_GPIO_J8_26** = 7, **RPI_BPLUS_GPIO_J8_29** = 5, **RPI_BPLUS_GPIO_J8_31** = 6, **RPI_BPLUS_GPIO_J8_32** = 12, **RPI_BPLUS_GPIO_J8_33** = 13, **RPI_BPLUS_GPIO_J8_35** = 19, **RPI_BPLUS_GPIO_J8_36** = 16, **RPI_BPLUS_GPIO_J8_37** = 26, **RPI_BPLUS_GPIO_J8_38** = 20, **RPI_BPLUS_GPIO_J8_40** = 21 }

GPIO Pin Numbers. enum **bcm2835SPIBitOrder** { **BCM2835_SPI_BIT_ORDER_LSBFIRST** = 0, **BCM2835_SPI_BIT_ORDER_MSBFIRST** = 1 }

bcm2835SPIBitOrder SPI Bit order Specifies the SPI data bit ordering for ***bcm2835_spi_setBitOrder()*** enum **bcm2835SPIMode** { **BCM2835_SPI_MODE0** = 0, **BCM2835_SPI_MODE1** = 1, **BCM2835_SPI_MODE2** = 2, **BCM2835_SPI_MODE3** = 3 }

SPI Data mode Specify the SPI data mode to be passed to ***bcm2835_spi_setDataMode()*** enum **bcm2835SPIChipSelect** { **BCM2835_SPI_CS0** = 0, **BCM2835_SPI_CS1** = 1, **BCM2835_SPI_CS2** = 2, **BCM2835_SPI_CS_NONE** = 3 }

bcm2835SPIChipSelect Specify the SPI chip select pin(s) enum **bcm2835SPIClockDivider** { **BCM2835_SPI_CLOCK_DIVIDER_65536** = 0,

BCM2835_SPI_CLOCK_DIVIDER_32768 = 32768,
BCM2835_SPI_CLOCK_DIVIDER_16384 = 16384,
BCM2835_SPI_CLOCK_DIVIDER_8192 = 8192,
BCM2835_SPI_CLOCK_DIVIDER_4096 = 4096,
BCM2835_SPI_CLOCK_DIVIDER_2048 = 2048,
BCM2835_SPI_CLOCK_DIVIDER_1024 = 1024,
BCM2835_SPI_CLOCK_DIVIDER_512 = 512,
BCM2835_SPI_CLOCK_DIVIDER_256 = 256,
BCM2835_SPI_CLOCK_DIVIDER_128 = 128, **BCM2835_SPI_CLOCK_DIVIDER_64** = 64, **BCM2835_SPI_CLOCK_DIVIDER_32** = 32, **BCM2835_SPI_CLOCK_DIVIDER_16** = 16, **BCM2835_SPI_CLOCK_DIVIDER_8** = 8, **BCM2835_SPI_CLOCK_DIVIDER_4** = 4, **BCM2835_SPI_CLOCK_DIVIDER_2** = 2, **BCM2835_SPI_CLOCK_DIVIDER_1** = 1 }

bcm2835SPIClockDivider Specifies the divider used to generate the SPI clock from the system clock. Figures below give the divider, clock period and clock frequency. Clock divided is based on nominal base clock rate of 250MHz It is reported that (contrary to the documentation) any even divider may used. The frequencies shown for each divider have been confirmed by measurement. enum **bcm2835I2CClockDivider** {
BCM2835_I2C_CLOCK_DIVIDER_2500 = 2500,
BCM2835_I2C_CLOCK_DIVIDER_626 = 626,
BCM2835_I2C_CLOCK_DIVIDER_150 = 150,
BCM2835_I2C_CLOCK_DIVIDER_148 = 148 }

bcm2835I2CClockDivider Specifies the divider used to generate the I2C clock from the system clock. Clock divided is based on nominal base clock rate of 250MHz enum **bcm2835I2CReasonCodes** { **BCM2835_I2C_REASON_OK** = 0x00,
BCM2835_I2C_REASON_ERROR_NACK = 0x01,
BCM2835_I2C_REASON_ERROR_CLKT = 0x02,
BCM2835_I2C_REASON_ERROR_DATA = 0x04 }

bcm2835I2CReasonCodes Specifies the reason codes for the bcm2835_i2c_write and bcm2835_i2c_read functions. uint32_t * **bcm2835_peripherals_base**

uint32_t **bcm2835_peripherals_size**
uint32_t * **bcm2835_peripherals**
volatile uint32_t * **bcm2835_st**
volatile uint32_t * **bcm2835_gpio**
volatile uint32_t * **bcm2835_pwm**
volatile uint32_t * **bcm2835_clk**
volatile uint32_t * **bcm2835_pads**
volatile uint32_t * **bcm2835_spi0**
volatile uint32_t * **bcm2835_bsc0**
volatile uint32_t * **bcm2835_bsc1**

Detailed Description

C library for Broadcom BCM 2835 as used in Raspberry Pi.

Author:
Mike McCauley

Date:
31 Mar 2015

Version:
1.20

Definition in file **bcm2835.h**.

Macro Definition Documentation

#define BCM2835_BLOCK_SIZE (4*1024)

Size of memory block on RPi

Definition at line 601 of file bcm2835.h.

#define BCM2835_BSC0_BASE 0x205000

Base Address of the BSC0 registers

Definition at line 527 of file bcm2835.h.

#define BCM2835_BSC1_BASE 0x804000

Base Address of the BSC1 registers

Definition at line 531 of file bcm2835.h.

#define BCM2835_BSC_A 0x000c

BSC Master Slave Address

Definition at line 897 of file bcm2835.h.

#define BCM2835_BSC_C 0x0000

BSC Master Control

Definition at line 894 of file bcm2835.h.

#define BCM2835_BSC_C_CLEAR_1 0x00000020

Clear FIFO Clear

Definition at line 909 of file bcm2835.h.

#define BCM2835_BSC_C_CLEAR_2 0x00000010

Clear FIFO Clear

Definition at line 910 of file bcm2835.h.

#define BCM2835_BSC_C_I2CEN 0x00008000

I2C Enable, 0 = disabled, 1 = enabled

Definition at line 904 of file bcm2835.h.

#define BCM2835_BSC_C_INTD 0x00000100

Interrupt on DONE

Definition at line 907 of file bcm2835.h.

#define BCM2835_BSC_C_INTR 0x00000400

Interrupt on RX

Definition at line 905 of file bcm2835.h.

#define BCM2835_BSC_C_INTT 0x00000200

Interrupt on TX

Definition at line 906 of file bcm2835.h.

#define BCM2835_BSC_C_READ 0x00000001

Read transfer

Definition at line 911 of file bcm2835.h.

#define BCM2835_BSC_C_ST 0x00000080

Start transfer, 1 = Start a new transfer

Definition at line 908 of file bcm2835.h.

#define BCM2835_BSC_CLKT 0x001c

BSC Master Clock Stretch Timeout

Definition at line 901 of file bcm2835.h.

#define BCM2835_BSC_DEL 0x0018

BSC Master Data Delay

Definition at line 900 of file bcm2835.h.

#define BCM2835_BSC_DIV 0x0014

BSC Master Clock Divider

Definition at line 899 of file bcm2835.h.

#define BCM2835_BSC_DLEN 0x0008

BSC Master Data Length

Definition at line 896 of file bcm2835.h.

#define BCM2835_BSC_FIFO 0x0010

BSC Master Data FIFO

Definition at line 898 of file bcm2835.h.

#define BCM2835_BSC_FIFO_SIZE 16

BSC FIFO size

Definition at line 925 of file bcm2835.h.

#define BCM2835_BSC_S 0x0004

BSC Master Status

Definition at line 895 of file bcm2835.h.

#define BCM2835_BSC_S_CLKT 0x00000200

Clock stretch timeout

Definition at line 914 of file bcm2835.h.

#define BCM2835_BSC_S_DONE 0x00000002

Transfer DONE

Definition at line 922 of file bcm2835.h.

#define BCM2835_BSC_S_ERR 0x00000100

ACK error

Definition at line 915 of file bcm2835.h.

#define BCM2835_BSC_S_RXD 0x00000020

RXD FIFO contains data

Definition at line 918 of file bcm2835.h.

#define BCM2835_BSC_S_RXF 0x00000080

RXF FIFO full, 0 = FIFO is not full, 1 = FIFO is full

Definition at line 916 of file bcm2835.h.

#define BCM2835_BSC_S_RXR 0x00000008

RXR FIFO needs reading (full)

Definition at line 920 of file bcm2835.h.

#define BCM2835_BSC_S_TA 0x00000001

Transfer Active

Definition at line 923 of file bcm2835.h.

#define BCM2835_BSC_S_TXD 0x00000010

TXD FIFO can accept data

Definition at line 919 of file bcm2835.h.

#define BCM2835_BSC_S_TXE 0x00000040

TXE FIFO full, 0 = FIFO is not full, 1 = FIFO is full

Definition at line 917 of file bcm2835.h.

#define BCM2835_BSC_S_TXW 0x00000004

TXW FIFO needs writing (full)

Definition at line 921 of file bcm2835.h.

#define BCM2835_CLOCK_BASE 0x101000

Base Address of the Clock/timer registers

Definition at line 521 of file bcm2835.h.

#define BCM2835_CORE_CLK_HZ 250000000

Speed of the core clock core_clk 250 MHz

Definition at line 494 of file bcm2835.h.

#define BCM2835_GPAFEN0 0x0088

GPIO Pin Async. Falling Edge Detect 0

Definition at line 635 of file bcm2835.h.

#define BCM2835_GPAFEN1 0x008c

GPIO Pin Async. Falling Edge Detect 1

Definition at line 636 of file bcm2835.h.

#define BCM2835_GPAREN0 0x007c

GPIO Pin Async. Rising Edge Detect 0

Definition at line 633 of file bcm2835.h.

#define BCM2835_GPAREN1 0x0080

GPIO Pin Async. Rising Edge Detect 1

Definition at line 634 of file bcm2835.h.

#define BCM2835_GPCLR0 0x0028

GPIO Pin Output Clear 0

Definition at line 619 of file bcm2835.h.

#define BCM2835_GPCLR1 0x002c

GPIO Pin Output Clear 1

Definition at line 620 of file bcm2835.h.

#define BCM2835_GPEDS0 0x0040

GPIO Pin Event Detect Status 0

Definition at line 623 of file bcm2835.h.

#define BCM2835_GPEDS1 0x0044

GPIO Pin Event Detect Status 1

Definition at line 624 of file bcm2835.h.

#define BCM2835_GPFEN0 0x0058

GPIO Pin Falling Edge Detect Enable 0

Definition at line 627 of file bcm2835.h.

#define BCM2835_GPFEN1 0x005c

GPIO Pin Falling Edge Detect Enable 1

Definition at line 628 of file bcm2835.h.

#define BCM2835_GPFSEL0 0x0000

GPIO register offsets from BCM2835_GPIO_BASE. Offsets into the GPIO Peripheral block in bytes per 6.1 Register View GPIO Function Select 0

Definition at line 611 of file bcm2835.h.

#define BCM2835_GPFSEL1 0x0004

GPIO Function Select 1

Definition at line 612 of file bcm2835.h.

#define BCM2835_GPFSEL2 0x0008

GPIO Function Select 2

Definition at line 613 of file bcm2835.h.

#define BCM2835_GPFSEL3 0x000c

GPIO Function Select 3

Definition at line 614 of file bcm2835.h.

#define BCM2835_GPFSEL4 0x0010

GPIO Function Select 4

Definition at line 615 of file bcm2835.h.

#define BCM2835_GPFSEL5 0x0014

GPIO Function Select 5

Definition at line 616 of file bcm2835.h.

#define BCM2835_GPHEN0 0x0064

GPIO Pin High Detect Enable 0

Definition at line 629 of file bcm2835.h.

#define BCM2835_GPHEN1 0x0068

GPIO Pin High Detect Enable 1

Definition at line 630 of file bcm2835.h.

#define BCM2835_GPIO_BASE 0x200000

Base Address of the GPIO registers

Definition at line 523 of file bcm2835.h.

#define BCM2835_GPIO_PADS 0x100000

Base Address of the Pads registers

Definition at line 519 of file bcm2835.h.

#define BCM2835_GPIO_PWM 0x20C000

Base Address of the PWM registers

Definition at line 529 of file bcm2835.h.

#define BCM2835_GPLEN0 0x0070

GPIO Pin Low Detect Enable 0

Definition at line 631 of file bcm2835.h.

#define BCM2835_GPLEN1 0x0074

GPIO Pin Low Detect Enable 1

Definition at line 632 of file bcm2835.h.

#define BCM2835_GPLEV0 0x0034

GPIO Pin Level 0

Definition at line 621 of file bcm2835.h.

#define BCM2835_GPLEV1 0x0038

GPIO Pin Level 1

Definition at line 622 of file bcm2835.h.

#define BCM2835_GPPUD 0x0094

GPIO Pin Pull-up/down Enable

Definition at line 637 of file bcm2835.h.

#define BCM2835_GPPUDCLK0 0x0098

GPIO Pin Pull-up/down Enable Clock 0

Definition at line 638 of file bcm2835.h.

#define BCM2835_GPPUDCLK1 0x009c

GPIO Pin Pull-up/down Enable Clock 1

Definition at line 639 of file bcm2835.h.

#define BCM2835_GPREN0 0x004c

GPIO Pin Rising Edge Detect Enable 0

Definition at line 625 of file bcm2835.h.

#define BCM2835_GPREN1 0x0050

GPIO Pin Rising Edge Detect Enable 1

Definition at line 626 of file bcm2835.h.

#define BCM2835_GPSET0 0x001c

GPIO Pin Output Set 0

Definition at line 617 of file bcm2835.h.

#define BCM2835_GPSET1 0x0020

GPIO Pin Output Set 1

Definition at line 618 of file bcm2835.h.

#define BCM2835_PAD_DRIVE_10mA 0x04

10mA drive current

Definition at line 680 of file bcm2835.h.

#define BCM2835_PAD_DRIVE_12mA 0x05

12mA drive current

Definition at line 681 of file bcm2835.h.

#define BCM2835_PAD_DRIVE_14mA 0x06

14mA drive current

Definition at line 682 of file bcm2835.h.

#define BCM2835_PAD_DRIVE_16mA 0x07

16mA drive current

Definition at line 683 of file bcm2835.h.

#define BCM2835_PAD_DRIVE_2mA 0x00

2mA drive current

Definition at line 676 of file bcm2835.h.

#define BCM2835_PAD_DRIVE_4mA 0x01

4mA drive current

Definition at line 677 of file bcm2835.h.

#define BCM2835_PAD_DRIVE_6mA 0x02

6mA drive current

Definition at line 678 of file bcm2835.h.

#define BCM2835_PAD_DRIVE_8mA 0x03

8mA drive current

Definition at line 679 of file bcm2835.h.

#define BCM2835_PAD_HYSTERESIS_ENABLED 0x08

Hysteresis enabled

Definition at line 675 of file bcm2835.h.

#define BCM2835_PAD_PASSWRD (0x5A << 24)

Pad Control masks Password to enable setting pad mask

Definition at line 673 of file bcm2835.h.

#define BCM2835_PAD_SLEW_RATE_UNLIMITED 0x10

Slew rate unlimited

Definition at line 674 of file bcm2835.h.

#define BCM2835_PADS_GPIO_0_27 0x002c

Pad control register offsets from BCM2835_GPIO_PADS Pad control register for pads 0 to 27

Definition at line 668 of file bcm2835.h.

#define BCM2835_PADS_GPIO_28_45 0x0030

Pad control register for pads 28 to 45

Definition at line 669 of file bcm2835.h.

#define BCM2835_PADS_GPIO_46_53 0x0034

Pad control register for pads 46 to 53

Definition at line 670 of file bcm2835.h.

#define BCM2835_PAGE_SIZE (4*1024)

Size of memory page on RPi

Definition at line 599 of file bcm2835.h.

#define BCM2835_PERI_BASE 0x20000000

Physical addresses for various peripheral register sets Base Physical Address of the BCM 2835 peripheral registers Note this is different for the RPi2 BCM2836, where this is derived from /proc/device-tree/soc/ranges If /proc/device-tree/soc/ranges exists on a RPi 1 OS, it would be expected to contain the following numbers:

Peripherals block base address on RPi 1

Definition at line 510 of file bcm2835.h.

#define BCM2835_PERI_SIZE 0x01000000

Size of the peripherals block on RPi 1

Definition at line 512 of file bcm2835.h.

#define BCM2835_PWM0_ENABLE 0x0001

Channel Enable

Definition at line 998 of file bcm2835.h.

#define BCM2835_PWM0_MS_MODE 0x0080

Run in Mark/Space mode

Definition at line 991 of file bcm2835.h.

#define BCM2835_PWM0_OFFSTATE 0x0008

Ouput Off state

Definition at line 995 of file bcm2835.h.

#define BCM2835_PWM0_REPEATFF 0x0004

Repeat last value if FIFO empty

Definition at line 996 of file bcm2835.h.

#define BCM2835_PWM0_REVPOLAR 0x0010

Reverse polarity

Definition at line 994 of file bcm2835.h.

#define BCM2835_PWM0_SERIAL 0x0002

Run in serial mode

Definition at line 997 of file bcm2835.h.

#define BCM2835_PWM0_USEFIFO 0x0020

Data from FIFO

Definition at line 993 of file bcm2835.h.

#define BCM2835_PWM1_ENABLE 0x0100

Channel Enable

Definition at line 989 of file bcm2835.h.

#define BCM2835_PWM1_MS_MODE 0x8000

Run in Mark/Space mode

Definition at line 983 of file bcm2835.h.

#define BCM2835_PWM1_OFFSTATE 0x0800

Ouput Off state

Definition at line 986 of file bcm2835.h.

#define BCM2835_PWM1_REPEATFF 0x0400

Repeat last value if FIFO empty

Definition at line 987 of file bcm2835.h.

#define BCM2835_PWM1_REVPOLAR 0x1000

Reverse polarity

Definition at line 985 of file bcm2835.h.

#define BCM2835_PWM1_SERIAL 0x0200

Run in serial mode

Definition at line 988 of file bcm2835.h.

#define BCM2835_PWM1_USEFIFO 0x2000

Data from FIFO

Definition at line 984 of file bcm2835.h.

#define BCM2835_PWM_CLEAR_FIFO 0x0040

Clear FIFO

Definition at line 992 of file bcm2835.h.

#define BCM2835_PWM_PASSWRD (0x5A << 24)

Password to enable setting PWM clock

Definition at line 981 of file bcm2835.h.

#define BCM2835_SPI0_BASE 0x204000

Base Address of the SPI0 registers

Definition at line 525 of file bcm2835.h.

#define BCM2835_SPI0_CLK 0x0008

SPI Master Clock Divider

Definition at line 798 of file bcm2835.h.

#define BCM2835_SPI0_CS 0x0000

SPI Master Control and Status

Definition at line 796 of file bcm2835.h.

#define BCM2835_SPI0_CS_ADCS 0x00000800

ADCS Automatically Deassert Chip Select

Definition at line 818 of file bcm2835.h.

#define BCM2835_SPI0_CS_CLEAR 0x00000030

Clear FIFO Clear RX and TX

Definition at line 824 of file bcm2835.h.

#define BCM2835_SPI0_CS_CLEAR_RX 0x00000020

Clear FIFO Clear RX

Definition at line 825 of file bcm2835.h.

#define BCM2835_SPI0_CS_CLEAR_TX 0x00000010

Clear FIFO Clear TX

Definition at line 826 of file bcm2835.h.

#define BCM2835_SPI0_CS_CPHA 0x00000004

Clock Phase

Definition at line 828 of file bcm2835.h.

#define BCM2835_SPI0_CS_CPOL 0x00000008

Clock Polarity

Definition at line 827 of file bcm2835.h.

#define BCM2835_SPI0_CS_CS 0x00000003

Chip Select

Definition at line 829 of file bcm2835.h.

#define BCM2835_SPI0_CS_CSPOL 0x00000040

Chip Select Polarity

Definition at line 823 of file bcm2835.h.

#define BCM2835_SPI0_CS_CSPOL0 0x00200000

Chip Select 0 Polarity

Definition at line 808 of file bcm2835.h.

#define BCM2835_SPI0_CS_CSPOL1 0x00400000

Chip Select 1 Polarity

Definition at line 807 of file bcm2835.h.

#define BCM2835_SPI0_CS_CSPOL2 0x00800000

Chip Select 2 Polarity

Definition at line 806 of file bcm2835.h.

#define BCM2835_SPI0_CS_DMA_LEN 0x01000000

Enable DMA mode in Lossi mode

Definition at line 805 of file bcm2835.h.

#define BCM2835_SPI0_CS_DMAEN 0x00000100

DMAEN DMA Enable

Definition at line 821 of file bcm2835.h.

#define BCM2835_SPI0_CS_DONE 0x00010000

Done transfer Done

Definition at line 813 of file bcm2835.h.

#define BCM2835_SPI0_CS_INTD 0x00000200

INTD Interrupt on Done

Definition at line 820 of file bcm2835.h.

#define BCM2835_SPI0_CS_INTR 0x00000400

INTR Interrupt on RXR

Definition at line 819 of file bcm2835.h.

#define BCM2835_SPI0_CS_LEN 0x00002000

LEN LoSSI enable

Definition at line 816 of file bcm2835.h.

#define BCM2835_SPI0_CS_LEN_LONG 0x02000000

Enable Long data word in Lossi mode if DMA_LEN is set

Definition at line 804 of file bcm2835.h.

#define BCM2835_SPI0_CS_LMONO 0x00004000

Unused

Definition at line 815 of file bcm2835.h.

#define BCM2835_SPI0_CS_REN 0x00001000

REN Read Enable

Definition at line 817 of file bcm2835.h.

#define BCM2835_SPI0_CS_RXD 0x00020000

RXD RX FIFO contains Data

Definition at line 812 of file bcm2835.h.

#define BCM2835_SPI0_CS_RXF 0x00100000

RXF - RX FIFO Full

Definition at line 809 of file bcm2835.h.

#define BCM2835_SPI0_CS_RXR 0x00080000

RXR RX FIFO needs Reading (full)

Definition at line 810 of file bcm2835.h.

#define BCM2835_SPI0_CS_TA 0x00000080

Transfer Active

Definition at line 822 of file bcm2835.h.

#define BCM2835_SPI0_CS_TE_EN 0x00008000

Unused

Definition at line 814 of file bcm2835.h.

#define BCM2835_SPI0_CS_TXD 0x00040000

TXD TX FIFO can accept Data

Definition at line 811 of file bcm2835.h.

#define BCM2835_SPI0_DC 0x0014

SPI DMA DREQ Controls

Definition at line 801 of file bcm2835.h.

#define BCM2835_SPI0_DLEN 0x000c

SPI Master Data Length

Definition at line 799 of file bcm2835.h.

#define BCM2835_SPI0_FIFO 0x0004

SPI Master TX and RX FIFOs

Definition at line 797 of file bcm2835.h.

#define BCM2835_SPI0_LTOH 0x0010

SPI LOSSI mode TOH

Definition at line 800 of file bcm2835.h.

#define BCM2835_ST_BASE 0x3000

Offsets for the bases of various peripherals within the peripherals block / Base Address of the System Timer registers

Definition at line 517 of file bcm2835.h.

#define BCM2835_ST_CHI 0x0008

System Timer Counter Upper 32 bits

Definition at line 963 of file bcm2835.h.

#define BCM2835_ST_CLO 0x0004

System Timer Counter Lower 32 bits

Definition at line 962 of file bcm2835.h.

#define BCM2835_ST_CS 0x0000

System Timer Control/Status

Definition at line 961 of file bcm2835.h.

#define BCM2835_VERSION 10050 /* Version 1.50 */

C library for Broadcom BCM 2835 as used in Raspberry Pi

This is a C library for Raspberry Pi (RPi). It provides access to GPIO and other IO functions on the Broadcom BCM 2835 chip, allowing access to the GPIO pins on the 26 pin IDE plug on the RPi board so you can control and interface with various external devices.

It provides functions for reading digital inputs and setting digital outputs, using SPI and I2C, and for accessing the system timers. Pin event detection is supported by polling (interrupts are not supported).

It is C++ compatible, and installs as a header file and non-shared library on any Linux-based distro (but clearly is no use except on Raspberry Pi or another board with BCM 2835).

The version of the package that this documentation refers to can be downloaded from `http://www.airspayce.com/mikem/bcm2835/bcm2835-1.50.tar.gz` You can find the latest version at `http://www.airspayce.com/mikem/bcm2835`

Several example programs are provided.

Based on data in `http://elinux.org/RPi_Low-level_peripherals` and `http://www.raspberrypi.org/wp-content/uploads/2012/02/BCM2835-ARM-Peripherals.pdf` and `http://www.scribd.com/doc/101830961/GPIO-Pads-Control2`

You can also find online help and discussion at `http://groups.google.com/group/bcm2835` Please use that group for all questions and discussions on this topic. Do not contact the author directly, unless it is to discuss commercial licensing. Before asking a question or reporting a bug, please read `http://www.catb.org/esr/faqs/smart-questions.html`

Tested on debian6-19-04-2012, 2012-07-15-wheezy-raspbian, 2013-07-26-wheezy-raspbian and Occidentalisv01, 2016-02-09 Raspbian Jessie. CAUTION: it has been observed that when detect enables such as **bcm2835_gpio_len()** are used and the pin is pulled LOW it can cause temporary hangs on 2012-07-15-wheezy-raspbian, 2013-07-26-wheezy-raspbian and Occidentalisv01. Reason for this is not yet determined, but we suspect that an interrupt handler is hitting a hard loop on those OSs. If you must use **bcm2835_gpio_len()** and friends, make sure you disable the pins with **bcm2835_gpio_clr_len()** and friends after use.

Running as root

Prior to the release of Raspbian Jessie in Feb 2016, access to any peripheral device via /dev/mem on the RPi required the process to run as root. Raspbian Jessie permits non-root users to access the GPIO peripheral (only) via /dev/gpiomem, and this library supports that limited mode of operation.

If the library runs with effective UID of 0 (ie root), then **bcm2835_init()** will attempt to open /dev/mem, and, if successful, it will permit use of all peripherals and library functions.

If the library runs with any other effective UID (ie not root), then **bcm2835_init()** will attempt to open /dev/gpiomem, and, if successful, will only permit GPIO operations. In particular, **bcm2835_spi_begin()** and **bcm2835_i2c_begin()** will return false and all other non-gpio operations may fail silently or crash.

Installation

This library consists of a single non-shared library and header file, which will be installed in the usual places by make install

```
# download the latest version of the library, say bcm2835-1.xx.tar.gz, then:
tar zxvf bcm2835-1.xx.tar.gz
cd bcm2835-1.xx
./configure
make
sudo make check
sudo make install
```

Physical Addresses

The functions **bcm2835_peri_read()**, **bcm2835_peri_write()** and **bcm2835_peri_set_bits()** are low level peripheral register access functions. They are designed to use physical addresses as described in section 1.2.3 ARM physical addresses of the BCM2835 ARM Peripherals manual. Physical addresses range from 0x20000000 to 0x20FFFFFF for peripherals. The bus addresses for peripherals are set up to map onto the peripheral bus address range starting at 0x7E000000. Thus a peripheral advertised in the manual at bus address 0x7Ennnnnn is available at physical address 0x20nnnnnn.

On RPI 2, the peripheral addresses are different and the bcm2835 library gets them from reading /proc/device-tree/soc/ranges. This is only availble with recent versions of the kernel on RPI 2.

After initialisation, the base address of the various peripheral registers are available with the following externals: bcm2835_gpio bcm2835_pwm bcm2835_clk bcm2835_pads bcm2835_spio0 bcm2835_st bcm2835_bsc0 bcm2835_bsc1

Raspberry Pi 2 (RPI2)

For this library to work correctly on RPI2, you MUST have the device tree support enabled in the kernel. You should also ensure you are using the latest version of Linux. The library has been tested on RPI2 with 2015-02-16-raspbian-wheezy and ArchLinuxARM-rpi-2 as of 2015-03-29.

When device tree suport is enabled, the file /proc/device-tree/soc/ranges will appear in the file system, and the bcm2835 module relies on its presence to correctly run on RPI2 (it is optional for RPI1). Without device tree support enabled and the presence of this file, it will not work on RPI2.

To enable device tree support:

```
sudo raspi-config
 under Advanced Options - enable Device Tree
 Reboot.
```

Pin Numbering

The GPIO pin numbering as used by RPi is different to and inconsistent with the underlying BCM 2835 chip pin numbering. `http://elinux.org/RPi_BCM2835_GPIOs`

RPi has a 26 pin IDE header that provides access to some of the GPIO pins on the BCM 2835, as well as power and ground pins. Not all GPIO pins on the BCM 2835 are available on the IDE header.

RPi Version 2 also has a P5 connector with 4 GPIO pins, 5V, 3.3V and Gnd.

The functions in this library are designed to be passed the BCM 2835 GPIO pin number and *not* the RPi pin number. There are symbolic definitions for each of the available pins that you should use for convenience. See **RPiGPIOPin**.

SPI Pins

The bcm2835_spi_* functions allow you to control the BCM 2835 SPI0 interface, allowing you to send and received data by SPI (Serial Peripheral Interface). For more information about SPI, see `http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus`

When **bcm2835_spi_begin()** is called it changes the bahaviour of the SPI interface pins from their default GPIO behaviour in order to support SPI. While SPI is in use, you will not be able to control the state of the SPI pins through the usual bcm2835_spi_gpio_write(). When **bcm2835_spi_end()** is called, the SPI pins will all revert to inputs, and can then be configured and controled with the usual bcm2835_gpio_* calls.

The Raspberry Pi GPIO pins used for SPI are:

1 P1-19 (MOSI)
2 P1-21 (MISO)
3 P1-23 (CLK)
4 P1-24 (CE0)
5 P1-26 (CE1)

I2C Pins

The bcm2835_i2c_* functions allow you to control the BCM 2835 BSC interface, allowing you to send and received data by I2C ("eye-squared cee"; generically referred to as "two-wire interface") . For more information about I?C, see `http://en.wikipedia.org/wiki/I%C2%B2C`

The Raspberry Pi V2 GPIO pins used for I2C are:

6 P1-03 (SDA)
7 P1-05 (SLC)

PWM

The BCM2835 supports hardware PWM on a limited subset of GPIO pins. This bcm2835 library provides functions for configuring and controlling PWM output on these pins.

The BCM2835 contains 2 independent PWM channels (0 and 1), each of which be connnected to a limited subset of GPIO pins. The following GPIO pins may be connected to the following PWM channels (from section 9.5):

GPIO PIN	RPi pin	PWM Channel	ALT FUN
12		0	0
13		1	0
18	1-12	0	5
19		1	5
40		0	0
41		1	0
45		1	0
52		0	1
53		1	1

In order for a GPIO pin to emit output from its PWM channel, it must be set to the Alt Function given above. Note carefully that current versions of the Raspberry Pi only expose one of these pins (GPIO 18 = RPi Pin 1-12) on the IO headers, and therefore this is the only IO pin on the RPi that can be used for PWM. Further it must be set to ALT FUN 5 to get PWM output.

Both PWM channels are driven by the same PWM clock, whose clock dvider can be varied using **bcm2835_pwm_set_clock()**. Each channel can be separately enabled with **bcm2835_pwm_set_mode()**. The average output of the PWM channel is determined by the ratio of DATA/RANGE for that channel. Use **bcm2835_pwm_set_range()** to set the range and **bcm2835_pwm_set_data()** to set the data in that ratio

Each PWM channel can run in either Balanced or Mark-Space mode. In Balanced mode, the hardware sends a combination of clock pulses that results in an overall DATA pulses per RANGE pulses. In Mark-Space mode, the hardware sets the output HIGH for DATA clock pulses wide, followed by LOW for RANGE-DATA clock pulses.

The PWM clock can be set to control the PWM pulse widths. The PWM clock is derived from a 19.2MHz clock. You can set any divider, but some common ones are provided by the BCM2835_PWM_CLOCK_DIVIDER_* values of **bcm2835PWMClockDivider**.

For example, say you wanted to drive a DC motor with PWM at about 1kHz, and control the speed in 1/1024 increments from 0/1024 (stopped) through to 1024/1024 (full on). In that case you might set the clock divider to be 16, and the RANGE to 1024. The pulse repetition frequency will be 1.2MHz/1024 = 1171.875Hz.

SPI

In order for bcm2835 library SPI to work, you may need to disable the SPI kernel module using:

```
sudo raspi-config
 under Advanced Options - enable Device Tree
 under Advanced Options - disable SPI
 Reboot.
```

Real Time performance constraints

The bcm2835 is a library for user programs (i.e. they run in 'userland'). Such programs are not part of the kernel and are usually subject to paging and swapping by the kernel while it does other things besides running your program. This means that you should not expect to get real-time performance or real-time timing constraints from such programs. In particular, there is no guarantee that the **bcm2835_delay()** and **bcm2835_delayMicroseconds()** will return after exactly the time requested. In fact, depending on other activity on the host, IO etc, you might get significantly longer delay times than the one you asked for. So please dont expect to get exactly the time delay you request.

Arjan reports that you can prevent swapping on Linux with the following code fragment:

```
struct sched_param sp;
memset(&sp, 0, sizeof(sp));
sp.sched_priority = sched_get_priority_max(SCHED_FIFO);
sched_setscheduler(0, SCHED_FIFO, &sp);
mlockall(MCL_CURRENT | MCL_FUTURE);
```

Bindings to other languages

mikem has made Perl bindings available at CPAN: `http://search.cpan.org/~mikem/Device-BCM2835-1.9/lib/Device/BCM2835.pm` Matthew Baker has kindly made Python bindings available at: https: github.com/mubeta06/py-libbcm2835 Gary Marks has created a Serial Peripheral Interface (SPI) command-line utility for Raspberry Pi, based on the bcm2835 library. The utility, spincl, is licensed under Open Source GNU GPLv3 by iP Solutions (`http://ipsolutionscorp.com`), as a free download with source included: `http://ipsolutionscorp.com/raspberry-pi-spi-utility/`

Open Source Licensing GPL V2

This is the appropriate option if you want to share the source code of your application with everyone you distribute it to, and you also want to give them the right to share who uses it. If you wish to use this software under Open Source Licensing, you must contribute all your source code to the open source community in accordance with the GPL Version 2 when your application is distributed. See `http://www.gnu.org/copyleft/gpl.html` and COPYING

Acknowledgements

Some of this code has been inspired by Dom and Gert. The I2C code has been inspired by Alan Barr.

Revision History

Version:

1.0 Initial release

1.1 Minor bug fixes
1.2 Added support for SPI
1.3 Added **bcm2835_spi_transfern()**
1.4 Fixed a problem that prevented SPI CE1 being used. Reported by David Robinson.
1.5 Added **bcm2835_close()** to deinit the library. Suggested by C?sar Ortiz
1.6 Document testing on 2012-07-15-wheezy-raspbian and Occidentalisv01 Functions **bcm2835_gpio_ren()**, **bcm2835_gpio_fen()**, **bcm2835_gpio_hen() bcm2835_gpio_len()**, **bcm2835_gpio_aren()** and **bcm2835_gpio_afen()** now changes only the pin specified. Other pins that were already previously enabled stay enabled. Added **bcm2835_gpio_clr_ren()**, **bcm2835_gpio_clr_fen()**, **bcm2835_gpio_clr_hen() bcm2835_gpio_clr_len()**, **bcm2835_gpio_clr_aren()**, **bcm2835_gpio_clr_afen()** to clear the enable for individual pins, suggested by Andreas Sundstrom.
1.7 Added bcm2835_spi_transfernb to support different buffers for read and write.
1.8 Improvements to read barrier, as suggested by maddin.
1.9 Improvements contributed by mikew: I noticed that it was mallocing memory for the mmaps on /dev/mem. It's not necessary to do that, you can just mmap the file directly, so I've removed the mallocs (and frees). I've also modified delayMicroseconds() to use nanosleep() for long waits, and a busy wait on a high resolution timer for the rest. This is because I've found that calling nanosleep() takes at least 100-200 us. You need to link using '-lrt' using this version. I've added some unsigned casts to the debug prints to silence compiler warnings I was getting, fixed some typos, and changed the value of BCM2835_PAD_HYSTERESIS_ENABLED to 0x08 as per Gert van Loo's doc at `http://www.scribd.com/doc/101830961/GPIO-Pads-Control2` Also added a define for the passwrd value that Gert says is needed to change pad control settings.
1.10 Changed the names of the delay functions to **bcm2835_delay()** and **bcm2835_delayMicroseconds()** to prevent collisions with wiringPi. Macros to map delay()-> **bcm2835_delay()** and Macros to map delayMicroseconds()-> **bcm2835_delayMicroseconds()**, which can be disabled by defining BCM2835_NO_DELAY_COMPATIBILITY
1.11 Fixed incorrect link to download file
1.12 New GPIO pin definitions for RPi version 2 (which has a different GPIO mapping)
1.13 New GPIO pin definitions for RPi version 2 plug P5 Hardware base pointers are now available (after initialisation) externally as bcm2835_gpio bcm2835_pwm bcm2835_clk bcm2835_pads bcm2835_spi0.
1.14 Now compiles even if CLOCK_MONOTONIC_RAW is not available, uses CLOCK_MONOTONIC instead. Fixed errors in documentation of SPI divider frequencies based on 250MHz clock. Reported by Ben Simpson.
1.15 Added **bcm2835_close()** to end of examples as suggested by Mark Wolfe.
1.16 Added bcm2835_gpio_set_multi, bcm2835_gpio_clr_multi and bcm2835_gpio_write_multi to allow a mask of pins to be set all at once. Requested by Sebastian Loncar.
1.17 Added bcm2835_gpio_write_mask. Requested by Sebastian Loncar.
1.18 Added bcm2835_i2c_* functions. Changes to bcm2835_delayMicroseconds: now uses the RPi system timer counter, instead of clock_gettime, for improved accuracy. No need to link with -lrt now. Contributed by Arjan van Vught.
1.19 Removed inlines added by previous patch since they don't seem to work everywhere. Reported by olly.
1.20 Patch from Mark Dootson to close /dev/mem after access to the peripherals has been granted.
1.21 delayMicroseconds is now not susceptible to 32 bit timer overruns. Patch courtesy Jeremy Mortis.
1.22 Fixed incorrect definition of BCM2835_GPFEN0 which broke the ability to set falling edge events. Reported by Mark Dootson.
1.23 Added bcm2835_i2c_set_baudrate and bcm2835_i2c_read_register_rs. Improvements to bcm2835_i2c_read and bcm2835_i2c_write functions to fix ocasional reads not completing. Patched by Mark Dootson.
1.24 Mark Dootson p[atched a problem with his previously submitted code under high load from other processes.
1.25 Updated author and distribution location details to airspayce.com

1.26 Added missing unmapmem for pads in bcm2835_close to prevent a memory leak. Reported by Hartmut Henkel.
1.27 **bcm2835_gpio_set_pad()** no longer needs BCM2835_PAD_PASSWRD: it is now automatically included. Added suport for PWM mode with bcm2835_pwm_* functions.
1.28 Fixed a problem where **bcm2835_spi_writenb()** would have problems with transfers of more than 64 bytes dues to read buffer filling. Patched by Peter Würtz.
1.29 Further fix to SPI from Peter Würtz.
1.30 10 microsecond delays from bcm2835_spi_transfer and bcm2835_spi_transfern for significant performance improvements, Patch by Alan Watson.
1.31 Fix a GCC warning about dummy variable, patched by Alan Watson. Thanks.
1.32 Added option I2C_V1 definition to compile for version 1 RPi. By default I2C code is generated for the V2 RPi which has SDA1 and SCL1 connected. Contributed by Malcolm Wiles based on work by Arvi Govindaraj.
1.33 Added command line utilities i2c and gpio to examples. Contributed by Shahrooz Shahparnia.
1.34 Added **bcm2835_i2c_write_read_rs()** which writes an arbitrary number of bytes, sends a repeat start, and reads from the device. Contributed by Eduardo Steinhorst.
1.35 Fix build errors when compiled under Qt. Also performance improvements with SPI transfers. Contributed b Udo Klaas.
1.36 Make automake's test runner detect that we're skipping tests when not root, the second one makes us skip the test when using fakeroot (as used when building Debian packages). Contributed by Guido Günther.
1.37 Moved confiure.in to configure.ac as receommnded by autoreconf.
Improvements to bcm2835_st_read to account for possible timer overflow, contributed by 'Ed'.
Added definitions for Raspberry Pi B+ J8 header GPIO pins.
.38 Added bcm2835_regbase for the benefit of C# wrappers, patch by Frank Hommers
.39 Beta version of RPi2 compatibility. Not tested here on RPi2 hardware. Testers please confirm correct operation on RPi2.
Unneccessary 'volatile' qualifiers removed from all variables and signatures.
Removed unsupportable PWM dividers, based on a report from Christophe Cecillon.
Minor improvements to spi.c example.
.40 Correct operation on RPi2 has been confirmed.
Fixed a number of compiler errors and warnings that occur when **bcm2835.h** is included in code compiled with -Wall -Woverflow -Wstrict-overflow -Wshadow -Wextra -pedantic. Reported by tlhackque.
Fixed a problem where calling bcm2835_delayMicroseconds loops forever when debug is set. Reported by tlhackque.
Reinstated use of volatile in 2 functions where there was a danger of lost reads or writes. Reported by tlhackque.
.41 Added BCM2835_VERSION macro and new function **bcm2835_version()**; Requested by tlhackque.
Improvements to peripheral memory barriers as suggested by tlhackque.
Reinstated some necessary volatile declarations as requested by tlhackque.
.42 Further improvements to memory barriers with the patient assistance and patches of tlhackque.
.43 Fixed problems with compiling barriers on RPI 2 with Arch Linux and gcc 4.9.2. Reported and patched by Lars Christensen.
Testing on RPI 2, with ArchLinuxARM-rpi-2-latest and 2015-02-16-raspbian-wheezy.
.44 Added documention about the need for device tree to be enabled on RPI2.
Improvements to detection of availablity of DMB instruction based on value of __ARM_ARCH macro.
.45 Fixed an error in the pad group offsets that would prevent **bcm2835_gpio_set_pad()** and **bcm2835_gpio_pad()** working correctly with non-0 pad groups. Reported by Guido.
1.46 2015-09-18 Added symbolic definitions for remaining pins on 40 pin GPIO header on RPi 2.
.47 2015-11-18 Fixed possibly incorrect reads in bcm2835_i2c_read_register_rs, patch from Eckhardt Ulrich.

.48 2015-12-08 Added patch from Eckhardt Ulrich that fixed problems that could cause hanging with bcm2835_i2c_read_register_rs and others.
1.49 2016-01-05 Added patch from Jonathan Perkin with new functions **bcm2835_gpio_eds_multi()** and **bcm2835_gpio_set_eds_multi()**.
1.50 2016-02-28 Added support for running as non-root, permitting access to GPIO only. Functions **bcm2835_spi_begin()** and **bcm2835_i2c_begin()** will now return 0 if not running as root (which prevents access to the SPI and I2C peripherals, amongst others). Testing on Raspbian Jessie.

Author:

Mike McCauley (`mikem@airspayce.com`) DO NOT CONTACT THE AUTHOR DIRECTLY: USE THE LISTS

Definition at line 472 of file bcm2835.h.

#define BMC2835_RPI2_DT_FILENAME "/proc/device-tree/soc/ranges"

On RPi2 with BCM2836, and all recent OSs, the base of the peripherals is read from a /proc file

Definition at line 497 of file bcm2835.h.

#define BMC2835_RPI2_DT_PERI_BASE_ADDRESS_OFFSET 4

Offset into BMC2835_RPI2_DT_FILENAME for the peripherals base address

Definition at line 499 of file bcm2835.h.

#define BMC2835_RPI2_DT_PERI_SIZE_OFFSET 8

Offset into BMC2835_RPI2_DT_FILENAME for the peripherals size address

Definition at line 501 of file bcm2835.h.

#define HIGH 0x1

Constants for passing to and from library functions The values here are designed to be passed to various functions in the bcm2835 library.

This means pin HIGH, true, 3.3volts on a pin.

Definition at line 489 of file bcm2835.h.

#define LOW 0x0

This means pin LOW, false, 0volts on a pin.

Definition at line 491 of file bcm2835.h.

Enumeration Type Documentation

enum bcm2835FunctionSelect

bcm2835PortFunction Port function select modes for **bcm2835_gpio_fsel()**

Enumerator

BCM2835_GPIO_FSEL_INPT Input 0b000

BCM2835_GPIO_FSEL_OUTP Output 0b001

BCM2835_GPIO_FSEL_ALT0 Alternate function 0 0b100

BCM2835_GPIO_FSEL_ALT1 Alternate function 1 0b101

BCM2835_GPIO_FSEL_ALT2 Alternate function 2 0b110,

BCM2835_GPIO_FSEL_ALT3 Alternate function 3 0b111

BCM2835_GPIO_FSEL_ALT4 Alternate function 4 0b011

BCM2835_GPIO_FSEL_ALT5 Alternate function 5 0b010

BCM2835_GPIO_FSEL_MASK Function select bits mask 0b111

Definition at line 644 of file bcm2835.h.

enum bcm2835I2CClockDivider

bcm2835I2CClockDivider Specifies the divider used to generate the I2C clock from the system clock. Clock divided is based on nominal base clock rate of 250MHz

Enumerator

BCM2835_I2C_CLOCK_DIVIDER_2500 2500 = 10us = 100 kHz

BCM2835_I2C_CLOCK_DIVIDER_626 622 = 2.504us = 399.3610 kHz

BCM2835_I2C_CLOCK_DIVIDER_150 150 = 60ns = 1.666 MHz (default at reset)

BCM2835_I2C_CLOCK_DIVIDER_148 148 = 59ns = 1.689 MHz

Definition at line 931 of file bcm2835.h.

enum bcm2835I2CReasonCodes

bcm2835I2CReasonCodes Specifies the reason codes for the bcm2835_i2c_write and bcm2835_i2c_read functions.

Enumerator

BCM2835_I2C_REASON_OK Success

BCM2835_I2C_REASON_ERROR_NACK Received a NACK

BCM2835_I2C_REASON_ERROR_CLKT Received Clock Stretch Timeout

BCM2835_I2C_REASON_ERROR_DATA Not all data is sent / received

Definition at line 942 of file bcm2835.h.

enum bcm2835PadGroup

bcm2835PadGroup Pad group specification for **bcm2835_gpio_pad()**

Enumerator

BCM2835_PAD_GROUP_GPIO_0_27 Pad group for GPIO pads 0 to 27

BCM2835_PAD_GROUP_GPIO_28_45 Pad group for GPIO pads 28 to 45

BCM2835_PAD_GROUP_GPIO_46_53 Pad group for GPIO pads 46 to 53

Definition at line 688 of file bcm2835.h.

enum bcm2835PUDControl

bcm2835PUDControl Pullup/Pulldown defines for **bcm2835_gpio_pud()**

Enumerator

BCM2835_GPIO_PUD_OFF Off ? disable pull-up/down 0b00

BCM2835_GPIO_PUD_DOWN Enable Pull Down control 0b01

BCM2835_GPIO_PUD_UP Enable Pull Up control 0b10

Definition at line 660 of file bcm2835.h.

enum bcm2835PWMClockDivider

bcm2835PWMClockDivider Specifies the divider used to generate the PWM clock from the system clock. Figures below give the divider, clock period and clock frequency. Clock divided is based on nominal PWM base clock rate of 19.2MHz The frequencies shown for each divider have been confirmed by measurement

Enumerator

BCM2835_PWM_CLOCK_DIVIDER_2048 2048 = 9.375kHz

BCM2835_PWM_CLOCK_DIVIDER_1024 1024 = 18.75kHz

BCM2835_PWM_CLOCK_DIVIDER_512 512 = 37.5kHz

BCM2835_PWM_CLOCK_DIVIDER_256 256 = 75kHz

BCM2835_PWM_CLOCK_DIVIDER_128 128 = 150kHz

BCM2835_PWM_CLOCK_DIVIDER_64 64 = 300kHz

BCM2835_PWM_CLOCK_DIVIDER_32 32 = 600.0kHz

BCM2835_PWM_CLOCK_DIVIDER_16 16 = 1.2MHz

BCM2835_PWM_CLOCK_DIVIDER_8 8 = 2.4MHz

BCM2835_PWM_CLOCK_DIVIDER_4 4 = 4.8MHz

BCM2835_PWM_CLOCK_DIVIDER_2 2 = 9.6MHz, fastest you can get

BCM2835_PWM_CLOCK_DIVIDER_1 1 = 4.6875kHz, same as divider 4096

Definition at line 1006 of file bcm2835.h.

enum bcm2835RegisterBase

bcm2835RegisterBase Register bases for **bcm2835_regbase()**

Enumerator

BCM2835_REGBASE_ST Base of the ST (System Timer) registers.

BCM2835_REGBASE_GPIO Base of the GPIO registers.

BCM2835_REGBASE_PWM Base of the PWM registers.

BCM2835_REGBASE_CLK Base of the CLK registers.

BCM2835_REGBASE_PADS Base of the PADS registers.

BCM2835_REGBASE_SPI0 Base of the SPI0 registers.

BCM2835_REGBASE_BSC0 Base of the BSC0 registers.

BCM2835_REGBASE_BSC1 Base of the BSC1 registers.

Definition at line 586 of file bcm2835.h.

enum bcm2835SPIBitOrder

bcm2835SPIBitOrder SPI Bit order Specifies the SPI data bit ordering for **bcm2835_spi_setBitOrder()**

Enumerator

BCM2835_SPI_BIT_ORDER_LSBFIRST LSB First

BCM2835_SPI_BIT_ORDER_MSBFIRST MSB First

Definition at line 834 of file bcm2835.h.

enum bcm2835SPIChipSelect

bcm2835SPIChipSelect Specify the SPI chip select pin(s)

Enumerator

BCM2835_SPI_CS0 Chip Select 0

BCM2835_SPI_CS1 Chip Select 1

BCM2835_SPI_CS2 Chip Select 2 (ie pins CS1 and CS2 are asserted)

BCM2835_SPI_CS_NONE No CS, control it yourself

Definition at line 854 of file bcm2835.h.

enum bcm2835SPIClockDivider

bcm2835SPIClockDivider Specifies the divider used to generate the SPI clock from the system clock. Figures below give the divider, clock period and clock frequency. Clock divided is based on nominal base clock rate of 250MHz It is reported that (contrary to the documentation) any even divider may used. The frequencies shown for each divider have been confirmed by measurement.

Enumerator

BCM2835_SPI_CLOCK_DIVIDER_65536 65536 = 262.144us = 3.814697260kHz

BCM2835_SPI_CLOCK_DIVIDER_32768 32768 = 131.072us = 7.629394531kHz

BCM2835_SPI_CLOCK_DIVIDER_16384 16384 = 65.536us = 15.25878906kHz

BCM2835_SPI_CLOCK_DIVIDER_8192 8192 = 32.768us = 30/51757813kHz

BCM2835_SPI_CLOCK_DIVIDER_4096 4096 = 16.384us = 61.03515625kHz

BCM2835_SPI_CLOCK_DIVIDER_2048 2048 = 8.192us = 122.0703125kHz

BCM2835_SPI_CLOCK_DIVIDER_1024 1024 = 4.096us = 244.140625kHz

BCM2835_SPI_CLOCK_DIVIDER_512 512 = 2.048us = 488.28125kHz

BCM2835_SPI_CLOCK_DIVIDER_256 256 = 1.024us = 976.5625kHz

BCM2835_SPI_CLOCK_DIVIDER_128 128 = 512ns = = 1.953125MHz

BCM2835_SPI_CLOCK_DIVIDER_64 64 = 256ns = 3.90625MHz

BCM2835_SPI_CLOCK_DIVIDER_32 32 = 128ns = 7.8125MHz

BCM2835_SPI_CLOCK_DIVIDER_16 16 = 64ns = 15.625MHz

BCM2835_SPI_CLOCK_DIVIDER_8 8 = 32ns = 31.25MHz

BCM2835_SPI_CLOCK_DIVIDER_4 4 = 16ns = 62.5MHz

BCM2835_SPI_CLOCK_DIVIDER_2 2 = 8ns = 125MHz, fastest you can get

BCM2835_SPI_CLOCK_DIVIDER_1 1 = 262.144us = 3.814697260kHz, same as 0/65536

Definition at line 869 of file bcm2835.h.

enum bcm2835SPIMode

SPI Data mode Specify the SPI data mode to be passed to **bcm2835_spi_setDataMode()**

Enumerator

BCM2835_SPI_MODE0 CPOL = 0, CPHA = 0

BCM2835_SPI_MODE1 CPOL = 0, CPHA = 1

BCM2835_SPI_MODE2 CPOL = 1, CPHA = 0

BCM2835_SPI_MODE3 CPOL = 1, CPHA = 1

Definition at line 843 of file bcm2835.h.

enum RPiGPIOPin

GPIO Pin Numbers.

Here we define Raspberry Pin GPIO pins on P1 in terms of the underlying BCM GPIO pin numbers. These can be passed as a pin number to any function requiring a pin. Not all pins on the RPi 26 bin IDE plug are connected to GPIO pins and some can adopt an alternate function. RPi version 2 has some slightly different pinouts, and these are values RPI_V2_*. RPi B+ has yet differnet pinouts and these are defined in RPI_BPLUS_*. At bootup, pins 8 and 10 are set to UART0_TXD, UART0_RXD (ie the alt0 function) respectively When SPI0 is in use (ie after **bcm2835_spi_begin()**), SPI0 pins are dedicated to SPI and cant be controlled independently. If

you are using the RPi Compute Module, just use the GPIO number: there is no need to use one of these symbolic names

Enumerator

RPI_GPIO_P1_03 Version 1, Pin P1-03

RPI_GPIO_P1_05 Version 1, Pin P1-05

RPI_GPIO_P1_07 Version 1, Pin P1-07

RPI_GPIO_P1_08 Version 1, Pin P1-08, defaults to alt function 0 UART0_TXD

RPI_GPIO_P1_10 Version 1, Pin P1-10, defaults to alt function 0 UART0_RXD

RPI_GPIO_P1_11 Version 1, Pin P1-11

RPI_GPIO_P1_12 Version 1, Pin P1-12, can be PWM channel 0 in ALT FUN 5

RPI_GPIO_P1_13 Version 1, Pin P1-13

RPI_GPIO_P1_15 Version 1, Pin P1-15

RPI_GPIO_P1_16 Version 1, Pin P1-16

RPI_GPIO_P1_18 Version 1, Pin P1-18

RPI_GPIO_P1_19 Version 1, Pin P1-19, MOSI when SPI0 in use

RPI_GPIO_P1_21 Version 1, Pin P1-21, MISO when SPI0 in use

RPI_GPIO_P1_22 Version 1, Pin P1-22

RPI_GPIO_P1_23 Version 1, Pin P1-23, CLK when SPI0 in use

RPI_GPIO_P1_24 Version 1, Pin P1-24, CE0 when SPI0 in use

RPI_GPIO_P1_26 Version 1, Pin P1-26, CE1 when SPI0 in use

RPI_V2_GPIO_P1_03 Version 2, Pin P1-03

RPI_V2_GPIO_P1_05 Version 2, Pin P1-05

RPI_V2_GPIO_P1_07 Version 2, Pin P1-07

RPI_V2_GPIO_P1_08 Version 2, Pin P1-08, defaults to alt function 0 UART0_TXD

RPI_V2_GPIO_P1_10 Version 2, Pin P1-10, defaults to alt function 0 UART0_RXD

RPI_V2_GPIO_P1_11 Version 2, Pin P1-11

RPI_V2_GPIO_P1_12 Version 2, Pin P1-12, can be PWM channel 0 in ALT FUN 5

RPI_V2_GPIO_P1_13 Version 2, Pin P1-13

RPI_V2_GPIO_P1_15 Version 2, Pin P1-15

RPI_V2_GPIO_P1_16 Version 2, Pin P1-16

RPI_V2_GPIO_P1_18 Version 2, Pin P1-18

RPI_V2_GPIO_P1_19 Version 2, Pin P1-19, MOSI when SPI0 in use

RPI_V2_GPIO_P1_21 Version 2, Pin P1-21, MISO when SPI0 in use

RPI_V2_GPIO_P1_22 Version 2, Pin P1-22

RPI_V2_GPIO_P1_23 Version 2, Pin P1-23, CLK when SPI0 in use

RPI_V2_GPIO_P1_24 Version 2, Pin P1-24, CE0 when SPI0 in use

RPI_V2_GPIO_P1_26 Version 2, Pin P1-26, CE1 when SPI0 in use

RPI_V2_GPIO_P1_29 Version 2, Pin P1-29

RPI_V2_GPIO_P1_31 Version 2, Pin P1-31

RPI_V2_GPIO_P1_32 Version 2, Pin P1-32

RPI_V2_GPIO_P1_33 Version 2, Pin P1-33

RPI_V2_GPIO_P1_35 Version 2, Pin P1-35

RPI_V2_GPIO_P1_36 Version 2, Pin P1-36

RPI_V2_GPIO_P1_37 Version 2, Pin P1-37

RPI_V2_GPIO_P1_38 Version 2, Pin P1-38

RPI_V2_GPIO_P1_40 Version 2, Pin P1-40

RPI_V2_GPIO_P5_03 Version 2, Pin P5-03

RPI_V2_GPIO_P5_04 Version 2, Pin P5-04

RPI_V2_GPIO_P5_05 Version 2, Pin P5-05

RPI_V2_GPIO_P5_06 Version 2, Pin P5-06

RPI_BPLUS_GPIO_J8_03 B+, Pin J8-03

RPI_BPLUS_GPIO_J8_05 B+, Pin J8-05

RPI_BPLUS_GPIO_J8_07 B+, Pin J8-07

RPI_BPLUS_GPIO_J8_08 B+, Pin J8-08, defaults to alt function 0 UART0_TXD

RPI_BPLUS_GPIO_J8_10 B+, Pin J8-10, defaults to alt function 0 UART0_RXD

RPI_BPLUS_GPIO_J8_11 B+, Pin J8-11

RPI_BPLUS_GPIO_J8_12 B+, Pin J8-12, can be PWM channel 0 in ALT FUN 5

RPI_BPLUS_GPIO_J8_13 B+, Pin J8-13

RPI_BPLUS_GPIO_J8_15 B+, Pin J8-15

RPI_BPLUS_GPIO_J8_16 B+, Pin J8-16

RPI_BPLUS_GPIO_J8_18 B+, Pin J8-18

RPI_BPLUS_GPIO_J8_19 B+, Pin J8-19, MOSI when SPI0 in use

RPI_BPLUS_GPIO_J8_21 B+, Pin J8-21, MISO when SPI0 in use

RPI_BPLUS_GPIO_J8_22 B+, Pin J8-22

RPI_BPLUS_GPIO_J8_23 B+, Pin J8-23, CLK when SPI0 in use

RPI_BPLUS_GPIO_J8_24 B+, Pin J8-24, CE0 when SPI0 in use

RPI_BPLUS_GPIO_J8_26 B+, Pin J8-26, CE1 when SPI0 in use

RPI_BPLUS_GPIO_J8_29 B+, Pin J8-29,

RPI_BPLUS_GPIO_J8_31 B+, Pin J8-31,

RPI_BPLUS_GPIO_J8_32 B+, Pin J8-32,

RPI_BPLUS_GPIO_J8_33 B+, Pin J8-33,

RPI_BPLUS_GPIO_J8_35 B+, Pin J8-35,

RPI_BPLUS_GPIO_J8_36 B+, Pin J8-36,

RPI_BPLUS_GPIO_J8_37 B+, Pin J8-37,

RPI_BPLUS_GPIO_J8_38 B+, Pin J8-38,

RPI_BPLUS_GPIO_J8_40 B+, Pin J8-40,

Definition at line 709 of file bcm2835.h.

Function Documentation

int bcm2835_close (void)

Close the library, deallocating any allocated memory and closing /dev/mem

Returns:

1 if successful else 0

void bcm2835_delay (unsigned int *millis*)

Delays for the specified number of milliseconds. Uses nanosleep(), and therefore does not use CPU until the time is up. However, you are at the mercy of nanosleep(). From the manual for nanosleep(): If the interval specified in req is not an exact multiple of the granularity underlying clock (see time(7)), then the interval will be rounded up to the next multiple. Furthermore, after the sleep completes, there may still be a delay before the CPU becomes free to once again execute the calling thread.

Parameters:

in	*millis*	Delay in milliseconds

Definition at line 449 of file bcm2835.c.

void bcm2835_delayMicroseconds (uint64_t *micros*)

Delays for the specified number of microseconds. Uses a combination of nanosleep() and a busy wait loop on the BCM2835 system timers, However, you are at the mercy of nanosleep(). From the manual for nanosleep(): If the interval specified in req is not an exact multiple of the granularity underlying clock (see time(7)), then the interval will be rounded up to the next multiple. Furthermore, after the sleep completes, there may still be a delay before the CPU becomes free to once again execute the calling thread. For times less than about 450 microseconds, uses a busy wait on the System Timer. It is reported that a delay of 0 microseconds on RaspberryPi will in fact result in a delay of about 80 microseconds. Your mileage may vary.

Parameters:

in	*micros*	Delay in microseconds

Definition at line 459 of file bcm2835.c.

void bcm2835_gpio_afen (uint8_t *pin*)

Enable Asynchronous Falling Edge Detect Enable for the specified pin. When a falling edge is detected, sets the appropriate pin in Event Detect Status. Asynchronous means the incoming signal is not sampled by the system clock. As such falling edges of very short duration can be detected.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

Definition at line 393 of file bcm2835.c.

void bcm2835_gpio_aren (uint8_t *pin*)

Enable Asynchronous Rising Edge Detect Enable for the specified pin. When a rising edge is detected, sets the appropriate pin in Event Detect Status. Asynchronous means the incoming signal is not sampled by the system clock. As such rising edges of very short duration can be detected.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

Definition at line 377 of file bcm2835.c.

void bcm2835_gpio_clr (uint8_t *pin*)

Sets the specified pin output to LOW.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

See Also:

bcm2835_gpio_write()

Definition at line 249 of file bcm2835.c.

void bcm2835_gpio_clr_afen (uint8_t *pin*)

Disable Asynchronous Falling Edge Detect Enable for the specified pin.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

Definition at line 400 of file bcm2835.c.

void bcm2835_gpio_clr_aren (uint8_t *pin*)

Disable Asynchronous Rising Edge Detect Enable for the specified pin.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

Definition at line 384 of file bcm2835.c.

void bcm2835_gpio_clr_fen (uint8_t *pin*)

Disable Falling Edge Detect Enable for the specified pin.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

Definition at line 336 of file bcm2835.c.

void bcm2835_gpio_clr_hen (uint8_t *pin*)

Disable High Detect Enable for the specified pin.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

Definition at line 352 of file bcm2835.c.

void bcm2835_gpio_clr_len (uint8_t *pin*)

Disable Low Detect Enable for the specified pin.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

Definition at line 368 of file bcm2835.c.

void bcm2835_gpio_clr_multi (uint32_t *mask*)

Sets any of the first 32 GPIO output pins specified in the mask to LOW.

Parameters:

in	*mask*	Mask of pins to affect. Use eg: (1 << RPI_GPIO_P1_03) | (1 << RPI_GPIO_P1_05)

See Also:

bcm2835_gpio_write_multi()

Definition at line 264 of file bcm2835.c.

void bcm2835_gpio_clr_ren (uint8_t *pin*)

Disable Rising Edge Detect Enable for the specified pin.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

Definition at line 320 of file bcm2835.c.

uint8_t bcm2835_gpio_eds (uint8_t *pin*)

Event Detect Status. Tests whether the specified pin has detected a level or edge as requested by **bcm2835_gpio_ren()**, **bcm2835_gpio_fen()**, **bcm2835_gpio_hen()**, **bcm2835_gpio_len()**, **bcm2835_gpio_aren()**, **bcm2835_gpio_afen()**. Clear the flag for a given pin by calling bcm2835_gpio_set_eds(pin);

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

Returns:

HIGH if the event detect status for the given pin is true.

Definition at line 282 of file bcm2835.c.

uint32_t bcm2835_gpio_eds_multi (uint32_t *mask*)

Same as **bcm2835_gpio_eds()** but checks if any of the pins specified in the mask have detected a level or edge.

Parameters:

in	*mask*	Mask of pins to check. Use eg: (1 << RPI_GPIO_P1_03) | (1 << RPI_GPIO_P1_05)

Returns:

Mask of pins HIGH if the event detect status for the given pin is true.

Definition at line 290 of file bcm2835.c.

void bcm2835_gpio_fen (uint8_t *pin*)

Enable Falling Edge Detect Enable for the specified pin. When a falling edge is detected, sets the appropriate pin in Event Detect Status. The GPRENn registers use synchronous edge detection. This means the input signal is sampled using the system clock and then it is looking for a ?100? pattern on the sampled signal. This has the effect of suppressing glitches.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

Definition at line 329 of file bcm2835.c.

void bcm2835_gpio_fsel (uint8_t *pin*, uint8_t *mode*)

GPIO register access These functions allow you to control the GPIO interface. You can set the function of each GPIO pin, read the input state and set the output state.

Sets the Function Select register for the given pin, which configures the pin as Input, Output or one of the 6 alternate functions.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.
in	*mode*	Mode to set the pin to, one of BCM2835_GPIO_FSEL_* from **bcm2835FunctionSelect**

Definition at line 230 of file bcm2835.c.

void bcm2835_gpio_hen (uint8_t *pin*)

Enable High Detect Enable for the specified pin. When a HIGH level is detected on the pin, sets the appropriate pin in Event Detect Status.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

Definition at line 345 of file bcm2835.c.

void bcm2835_gpio_len (uint8_t *pin*)

Enable Low Detect Enable for the specified pin. When a LOW level is detected on the pin, sets the appropriate pin in Event Detect Status.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

Definition at line 361 of file bcm2835.c.

uint8_t bcm2835_gpio_lev (uint8_t *pin*)

Reads the current level on the specified pin and returns either HIGH or LOW. Works whether or not the pin is an input or an output.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

Returns:

the current level either HIGH or LOW

Definition at line 271 of file bcm2835.c.

uint32_t bcm2835_gpio_pad (uint8_t *group*)

Reads and returns the Pad Control for the given GPIO group.

Parameters:

in	*group*	The GPIO pad group number, one of BCM2835_PAD_GROUP_GPIO_*

Returns:

Mask of bits from BCM2835_PAD_* from **bcm2835PadGroup**

Definition at line 426 of file bcm2835.c.

void bcm2835_gpio_pud (uint8_t *pud*)

Sets the Pull-up/down register for the given pin. This is used with **bcm2835_gpio_pudclk()** to set the Pull-up/down resistor for the given pin. However, it is usually more convenient to use **bcm2835_gpio_set_pud()**.

Parameters:

in	*pud*	The desired Pull-up/down mode. One of BCM2835_GPIO_PUD_* from bcm2835PUDControl

See Also:

bcm2835_gpio_set_pud()

Definition at line 409 of file bcm2835.c.

void bcm2835_gpio_pudclk (uint8_t *pin*, uint8_t *on*)

Clocks the Pull-up/down value set earlier by **bcm2835_gpio_pud()** into the pin.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.
in	*on*	HIGH to clock the value from **bcm2835_gpio_pud()** into the pin. LOW to remove the clock.

See Also:

bcm2835_gpio_set_pud()

Definition at line 418 of file bcm2835.c.

void bcm2835_gpio_ren (uint8_t *pin*)

Enable Rising Edge Detect Enable for the specified pin. When a rising edge is detected, sets the appropriate pin in Event Detect Status. The GPRENn registers use synchronous edge detection. This means the input signal is sampled using the system clock and then it is looking for a ?011? pattern on the sampled signal. This has the effect of suppressing glitches.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

Definition at line 313 of file bcm2835.c.

void bcm2835_gpio_set (uint8_t *pin*)

Sets the specified pin output to HIGH.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

See Also:

bcm2835_gpio_write()

Definition at line 241 of file bcm2835.c.

void bcm2835_gpio_set_eds (uint8_t *pin*)

Sets the Event Detect Status register for a given pin to 1, which has the effect of clearing the flag. Use this afer seeing an Event Detect Status on the pin.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.

Definition at line 298 of file bcm2835.c.

void bcm2835_gpio_set_eds_multi (uint32_t *mask*)

Same as **bcm2835_gpio_set_eds()** but clears the flag for any pin which is set in the mask.

Parameters:

in	*mask*	Mask of pins to clear. Use eg: (1 << RPI_GPIO_P1_03) | (1 << RPI_GPIO_P1_05)

Definition at line 306 of file bcm2835.c.

void bcm2835_gpio_set_multi (uint32_t *mask*)

Sets any of the first 32 GPIO output pins specified in the mask to HIGH.

Parameters:

in	*mask*	Mask of pins to affect. Use eg: (1 << RPI_GPIO_P1_03) | (1 << RPI_GPIO_P1_05)

See Also:

bcm2835_gpio_write_multi()

Definition at line 257 of file bcm2835.c.

void bcm2835_gpio_set_pad (uint8_t *group*, uint32_t *control*)

Sets the Pad Control for the given GPIO group.

Parameters:

in	*group*	The GPIO pad group number, one of BCM2835_PAD_GROUP_GPIO_*
in	*control*	Mask of bits from BCM2835_PAD_* from **bcm2835PadGroup**. Note that it is not necessary to include BCM2835_PAD_PASSWRD in the mask as this is automatically included.

Definition at line 438 of file bcm2835.c.

void bcm2835_gpio_set_pud (uint8_t *pin*, uint8_t *pud*)

Sets the Pull-up/down mode for the specified pin. This is more convenient than clocking the mode in with **bcm2835_gpio_pud()** and **bcm2835_gpio_pudclk()**.

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.
in	*pud*	The desired Pull-up/down mode. One of BCM2835_GPIO_PUD_* from bcm2835PUDControl

Definition at line 533 of file bcm2835.c.

void bcm2835_gpio_write (uint8_t *pin*, uint8_t *on*)

Sets the output state of the specified pin

Parameters:

in	*pin*	GPIO number, or one of RPI_GPIO_P1_* from **RPiGPIOPin**.
in	*on*	HIGH sets the output to HIGH and LOW to LOW.

Definition at line 491 of file bcm2835.c.

void bcm2835_gpio_write_mask (uint32_t *value*, uint32_t *mask*)

Sets the first 32 GPIO output pins specified in the mask to the value given by value

Parameters:

in	*value*	values required for each bit masked in by mask, eg: (1 << RPI_GPIO_P1_03) | (1 << RPI_GPIO_P1_05)
in	*mask*	Mask of pins to affect. Use eg: (1 << RPI_GPIO_P1_03) | (1 << RPI_GPIO_P1_05)

Definition at line 509 of file bcm2835.c.

void bcm2835_gpio_write_multi (uint32_t *mask*, uint8_t *on*)

Sets any of the first 32 GPIO output pins specified in the mask to the state given by on

Parameters:

in	*mask*	Mask of pins to affect. Use eg: (1 << RPI_GPIO_P1_03) | (1 << RPI_GPIO_P1_05)
in	*on*	HIGH sets the output to HIGH and LOW to LOW.

Definition at line 500 of file bcm2835.c.

int bcm2835_init (void)

Library initialisation and management These functions allow you to intialise and control the bcm2835 library

Initialise the library by opening /dev/mem (if you are root) or /dev/gpiomem (if you are not) and getting pointers to the internal memory for BCM 2835 device registers. You must call this (successfully) before calling any other functions in this library (except bcm2835_set_debug). If **bcm2835_init()** fails by returning 0, calling any other function may result in crashes or other failures. If **bcm2835_init()** succeeds but you are not running as root, then only gpio operations are permitted, and calling any other functions may result in crashes or other failures. . Prints messages to stderr in case of errors.

Returns:

1 if successful else 0

Definition at line 1282 of file bcm2835.c.

uint32_t bcm2835_peri_read (volatile uint32_t * *paddr*)

Reads 32 bit value from a peripheral address WITH a memory barrier before and after each read. This is safe, but slow. The MB before protects this read from any in-flight reads that didn't use a MB. The MB after protects subsequent reads from another peripheral.

Parameters:

in	*paddr*	Physical address to read from. See BCM2835_GPIO_BASE etc.

Returns:

the value read from the 32 bit register

See Also:

Physical Addresses

Definition at line 131 of file bcm2835.c.

uint32_t bcm2835_peri_read_nb (volatile uint32_t * *paddr*)

Reads 32 bit value from a peripheral address WITHOUT the read barriers You should only use this when: o your code has previously called **bcm2835_peri_read()** for a register within the same peripheral, and no read or write to another peripheral has occurred since. o your code has called bcm2835_memory_barrier() since the last access to ANOTHER peripheral.

Parameters:

in	*paddr*	Physical address to read from. See BCM2835_GPIO_BASE etc.

Returns:

the value read from the 32 bit register

See Also:

Physical Addresses

Definition at line 154 of file bcm2835.c.

void bcm2835_peri_set_bits (volatile uint32_t * *paddr*, uint32_t *value*, uint32_t *mask*)

Alters a number of bits in a 32 peripheral regsiter. It reads the current valu and then alters the bits defines as 1 in mask, according to the bit value in value. All other bits that are 0 in the mask are unaffected. Use this to alter a subset of the bits in a register. Memory barriers are used. Note that this is not atomic; an interrupt routine can cause unexpected results.

Parameters:

in	*paddr*	Physical address to read from. See BCM2835_GPIO_BASE etc.
in	*value*	The 32 bit value to write, masked in by mask.
in	*mask*	Bitmask that defines the bits that will be altered in the register.

See Also:

Physical Addresses

Definition at line 201 of file bcm2835.c.

void bcm2835_peri_write (volatile uint32_t * *paddr*, uint32_t *value*)

Writes 32 bit value from a peripheral address WITH a memory barrier before and after each write This is safe, but slow. The MB before ensures that any in-flight write to another peripheral completes before this write is issued. The MB after ensures that subsequent reads and writes to another peripheral will see the effect of this write.

This is a tricky optimization; if you aren't sure, use the barrier version.

Parameters:

in	*paddr*	Physical address to read from. See BCM2835_GPIO_BASE etc.
in	*value*	The 32 bit value to write

See Also:

Physical Addresses

Definition at line 170 of file bcm2835.c.

void bcm2835_peri_write_nb (volatile uint32_t * *paddr*, uint32_t *value*)

Writes 32 bit value from a peripheral address without the write barrier You should only use this when: o your code has previously called **bcm2835_peri_write()** for a register within the same peripheral, and no other peripheral access has occurred since. o your code has called bcm2835_memory_barrier() since the last access to ANOTHER peripheral.

This is a tricky optimization; if you aren't sure, use the barrier version.

Parameters:

in	*paddr*	Physical address to read from. See BCM2835_GPIO_BASE etc.
in	*value*	The 32 bit value to write

See Also:

Physical Addresses

Definition at line 185 of file bcm2835.c.

void bcm2835_pwm_set_clock (uint32_t *divisor*)

Pulse Width Modulation Allows control of 2 independent PWM channels. A limited subset of GPIO pins can be connected to one of these 2 channels, allowing PWM control of GPIO pins. You have to set the desired pin into a particular Alt Fun to PWM output. See the PWM documentation on the Main Page.

Sets the PWM clock divisor, to control the basic PWM pulse widths.

Parameters:

in	*divisor*	Divides the basic 19.2MHz PWM clock. You can use one of the common values BCM2835_PWM_CLOCK_DIVIDER_* in

		bcm2835PWMClockDivider

Definition at line 1208 of file bcm2835.c.

void bcm2835_pwm_set_data (uint8_t *channel*, uint32_t *data*)

Sets the PWM pulse ratio to emit to DATA/RANGE, where RANGE is set by **bcm2835_pwm_set_range()**.

Parameters:

in	*channel*	The PWM channel. 0 or 1.
in	*data*	Controls the PWM output ratio as a fraction of the range. Can vary from 0 to RANGE.

Definition at line 1270 of file bcm2835.c.

void bcm2835_pwm_set_mode (uint8_t *channel*, uint8_t *markspace*, uint8_t *enabled*)

Sets the mode of the given PWM channel, allowing you to control the PWM mode and enable/disable that channel

Parameters:

in	*channel*	The PWM channel. 0 or 1.
in	*markspace*	Set true if you want Mark-Space mode. 0 for Balanced mode.
in	*enabled*	Set true to enable this channel and produce PWM pulses.

Definition at line 1225 of file bcm2835.c.

void bcm2835_pwm_set_range (uint8_t *channel*, uint32_t *range*)

Sets the maximum range of the PWM output. The data value can vary between 0 and this range to control PWM output

Parameters:

in	*channel*	The PWM channel. 0 or 1.
in	*range*	The maximum value permitted for DATA.

Definition at line 1260 of file bcm2835.c.

uint32_t* bcm2835_regbase (uint8_t *regbase*)

Low level register access These functions provide low level register access, and should not generally need to be used

Gets the base of a register

Parameters:

in	*regbase*	You can use one of the common values BCM2835_REGBASE_* in **bcm2835RegisterBase**

Returns:

the register base

See Also:

Physical Addresses

Definition at line 94 of file bcm2835.c.

void bcm2835_set_debug (uint8_t *debug*)

Sets the debug level of the library. A value of 1 prevents mapping to /dev/mem, and makes the library print out what it would do, rather than accessing the GPIO registers. A value of 0, the default, causes normal operation. Call this before calling **bcm2835_init()**;

Parameters:

in	*debug*	The new debug level. 1 means debug

Definition at line 118 of file bcm2835.c.

int bcm2835_spi_begin (void)

SPI access These functions let you use SPI0 (Serial Peripheral Interface) to interface with an external SPI device.

Start SPI operations. Forces RPi SPI0 pins P1-19 (MOSI), P1-21 (MISO), P1-23 (CLK), P1-24 (CE0) and P1-26 (CE1) to alternate function ALT0, which enables those pins for SPI interface. You should call **bcm2835_spi_end()** when all SPI funcitons are complete to return the pins to their default functions.

See Also:

bcm2835_spi_end()

Returns:

1 if successful, 0 otherwise (perhaps because you are not running as root)

Definition at line 543 of file bcm2835.c.

void bcm2835_spi_chipSelect (uint8_t *cs*)

Sets the chip select pin(s) When an **bcm2835_spi_transfer()** is made, the selected pin(s) will be asserted during the transfer.

Parameters:

in	*cs*	Specifies the CS pins(s) that are used to activate the desired slave. One of BCM2835_SPI_CS*, see **bcm2835SPIChipSelect**

Definition at line 730 of file bcm2835.c.

void bcm2835_spi_end (void)

End SPI operations. SPI0 pins P1-19 (MOSI), P1-21 (MISO), P1-23 (CLK), P1-24 (CE0) and P1-26 (CE1) are returned to their default INPUT behaviour.

Definition at line 566 of file bcm2835.c.

void bcm2835_spi_setBitOrder (uint8_t *order*)

Sets the SPI bit order NOTE: has no effect. Not supported by SPI0. Defaults to

Parameters:

in	*order*	The desired bit order, one of BCM2835_SPI_BIT_ORDER_*, see **bcm2835SPIBitOrder**

void bcm2835_spi_setChipSelectPolarity (uint8_t *cs*, uint8_t *active*)

Sets the chip select pin polarity for a given pin When an **bcm2835_spi_transfer()** occurs, the currently selected chip select pin(s) will be asserted to the value given by active. When transfers are not happening, the chip select pin(s) return to the complement (inactive) value.

Parameters:

in	*cs*	The chip select pin to affect
in	*active*	Whether the chip select pin is to be active HIGH

Definition at line 737 of file bcm2835.c.

void bcm2835_spi_setClockDivider (uint16_t *divider*)

Sets the SPI clock divider and therefore the SPI clock speed.

Parameters:

in	*divider*	The desired SPI clock divider, one of BCM2835_SPI_CLOCK_DIVIDER_*, see **bcm2835SPIClockDivider**

Definition at line 586 of file bcm2835.c.

void bcm2835_spi_setDataMode (uint8_t *mode*)

Sets the SPI data mode Sets the clock polariy and phase

Parameters:

in	*mode*	The desired data mode, one of BCM2835_SPI_MODE*, see **bcm2835SPIMode**

Definition at line 592 of file bcm2835.c.

uint8_t bcm2835_spi_transfer (uint8_t *value*)

Transfers one byte to and from the currently selected SPI slave. Asserts the currently selected CS pins (as previously set by bcm2835_spi_chipSelect) during the transfer. Clocks the 8 bit value out on MOSI, and simultaneously clocks in data from MISO. Returns the read data byte from the slave. Uses polled transfer as per section 10.6.1 of the BCM 2835 ARM Peripherls manual

Parameters:

in	*value*	The 8 bit data byte to write to MOSI

Returns:

The 8 bit byte simultaneously read from MISO

See Also:

bcm2835_spi_transfern()

Definition at line 600 of file bcm2835.c.

void bcm2835_spi_transfern (char * *buf*, uint32_t *len*)

Transfers any number of bytes to and from the currently selected SPI slave using bcm2835_spi_transfernb. The returned data from the slave replaces the transmitted data in the buffer.

Parameters:

in,out	*buf*	Buffer of bytes to send. Received bytes will replace the contents
in	*len*	Number of bytes int eh buffer, and the number of bytes to send/received

See Also:

bcm2835_spi_transfer()

Definition at line 725 of file bcm2835.c.

void bcm2835_spi_transfernb (char * *tbuf*, char * *rbuf*, uint32_t *len*)

Transfers any number of bytes to and from the currently selected SPI slave. Asserts the currently selected CS pins (as previously set by bcm2835_spi_chipSelect) during the transfer. Clocks the len 8 bit bytes out on MOSI, and simultaneously clocks in data from MISO. The data read read from the slave is placed into rbuf. rbuf must be at least len bytes long Uses polled transfer as per section 10.6.1 of the BCM 2835 ARM Peripherls manual

Parameters:

in	*tbuf*	Buffer of bytes to send.
out	*rbuf*	Received bytes will by put in this buffer
in	*len*	Number of bytes in the tbuf buffer, and the number of bytes to send/received

See Also:

bcm2835_spi_transfer()

Definition at line 637 of file bcm2835.c.

void bcm2835_spi_writenb (char * *buf*, uint32_t *len*)

Transfers any number of bytes to the currently selected SPI slave. Asserts the currently selected CS pins (as previously set by bcm2835_spi_chipSelect) during the transfer.

Parameters:

in	*buf*	Buffer of bytes to send.
in	*len*	Number of bytes in the tbuf buffer, and the number of bytes to send

Definition at line 680 of file bcm2835.c.

void bcm2835_st_delay (uint64_t *offset_micros*, uint64_t *micros*)

Delays for the specified number of microseconds with offset.

Parameters:

in	*offset_micros*	Offset in microseconds
in	*micros*	Delay in microseconds

Definition at line 1198 of file bcm2835.c.

uint64_t bcm2835_st_read (void)

System Timer access Allows access to and delays using the System Timer Counter.

Read the System Timer Counter register.

Returns:

the value read from the System Timer Counter Lower 32 bits register

Definition at line 1168 of file bcm2835.c.

unsigned int bcm2835_version (void)

Returns the version number of the library, same as BCM2835_VERSION

Returns:

the current library version number

Definition at line 123 of file bcm2835.c.

Variable Documentation

volatile uint32_t* bcm2835_bsc0

Base of the BSC0 registers. Available after bcm2835_init has been called (as root)

Definition at line 74 of file bcm2835.c.

volatile uint32_t* bcm2835_bsc1

Base of the BSC1 registers. Available after bcm2835_init has been called (as root)

Definition at line 75 of file bcm2835.c.

volatile uint32_t* bcm2835_clk

Base of the CLK registers. Available after bcm2835_init has been called (as root)

Definition at line 71 of file bcm2835.c.

volatile uint32_t* bcm2835_gpio

Base of the GPIO registers. Available after bcm2835_init has been called

Definition at line 69 of file bcm2835.c.

volatile uint32_t* bcm2835_pads

Base of the PADS registers. Available after bcm2835_init has been called (as root)

Definition at line 72 of file bcm2835.c.

uint32_t* bcm2835_peripherals

Virtual memory address of the mapped peripherals block

Definition at line 65 of file bcm2835.c.

uint32_t* bcm2835_peripherals_base

Physical address and size of the peripherals block May be overridden on RPi2

Definition at line 60 of file bcm2835.c.

uint32_t bcm2835_peripherals_size

Size of the peripherals block to be mapped

Definition at line 61 of file bcm2835.c.

volatile uint32_t* bcm2835_pwm

Base of the PWM registers. Available after bcm2835_init has been called (as root)

Definition at line 70 of file bcm2835.c.

volatile uint32_t* bcm2835_spi0

Base of the SPI0 registers. Available after bcm2835_init has been called (as root)

Definition at line 73 of file bcm2835.c.

volatile uint32_t* bcm2835_st

Base of the ST (System Timer) registers. Available after bcm2835_init has been called (as root)

Definition at line 76 of file bcm2835.c.

main_car.c File Reference

Short example that control a robot car.

```
#include "marte_pistorms_sensor_touch.h"
#include "marte_pistorms_sensor_ultrasonic.h"
#include "marte_pistorms_brick.h"
#include "marte_pistorms_motors.h"
#include "marte_pistorms.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
```

Macros

#define **ULTRASONIC_ADDR** BANK_B_PORT_1
#define **TOUCH_ADDR** BANK_B_PORT_2
#define **LED_A** **BANK_A**
#define **LED_B** BANK_B
#define **MOTOR_1** BANK_B_PORT_1
#define **MOTOR_2** BANK_B_PORT_2
#define **MOTORS_BANK_B** BANK_B

Functions

int **main** (int argc, char **argv)

Detailed Description

Short example that control a robot car.

Author:

Carlos Ayerbe González

Date:

17 1 Apr 2017

Version:

1.0

This is a main program that has been designed to give control a car, when the car is going to touch some object it stops, go back and turn to avoid the object.

Definition in file **main_car.c**.

mainpage.h File Reference

Definition of class Template.

Detailed Description

Definition of class Template.

Definition in file **mainpage.h**.

marte_pistorms.c File Reference

Drivers for sensors and motors from Pistorms + Raspberry PI model B.

```
#include "bcm2835.h"
#include "marte_pistorms.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include "marte_pistorms_internal.h"
```

Macros

#define **PORT_1_SENSOR_ID** 0x71
#define **PORT_2_SENSOR_ID** 0xA5

Functions

int **pistorms_init** (void)
Initialize the bcm2835 library and start I2C operations.

int **pistorms_close** (void)
Close the bcm2835 library and end I2C operations .

int **_set_active_bank** (int connector_id)
Sets the I2C slave address.

char * **pistorms_get_device_id** (int connector_id)
Obtain the ID of the sensor.

Variables

char **bufSensorID** [16] = {0}

Detailed Description

Drivers for sensors and motors from Pistorms + Raspberry PI model B.

Author:
Carlos Ayerbe González

Date:
17 Jan 2017

Version:
1.0

Definition in file **marte_pistorms.c**.

Macro Definition Documentation

#define PORT_1_SENSOR_ID 0x71

Register for Sensor_1 ID and Sensor_2 ID

Definition at line 21 of file marte_pistorms.c.

marte_pistorms.h File Reference

Drivers for sensors and motors from Pistorms + Raspberry PI model B.

Macros

#define **BANK_A** 1
#define **BANK_B** 2
#define **BANK_A_ADDR** 0x1a
#define **BANK_B_ADDR** 0x1b
#define **BANK_A_PORT_1** 1
#define **BANK_A_PORT_2** 2
#define **BANK_B_PORT_1** 3
#define **BANK_B_PORT_2** 4
#define **PISTORMS_REASON_OK** 1
PistormsCodes Pistorms codes for the result of some functions.

#define **PISTORMS_ERROR_WRONG_CONNECTOR_ID** -1
#define **PISTORMS_INITIALIZATION_OK** 1
#define **PISTORMS_ERROR_INITIALIZATION** 0
#define **PISTORMS_CLOSE_OK** 1
#define **PISTORMS_ERROR_CLOSE** 0
#define **PISTORMS_ERROR_SENSOR_ID** 0
#define **PISTORMS_ERROR_SENSOR_MODE** 0
#define **PISTORMS_ERROR_BAD_CONNECTOR** "ERROR"
#define **PISTORMS_ERROR_NOT_INITIALIZED** "ERROR"

Functions

int **pistorms_init** (void)
Initialize the bcm2835 library and start I2C operations.

int **pistorms_close** (void)
Close the bcm2835 library and end I2C operations .

int **_set_active_bank** (int connector_id)
Sets the I2C slave address.

char * **pistorms_get_device_id** (int connector_id)
Obtain the ID of the sensor.

Detailed Description

Drivers for sensors and motors from Pistorms + Raspberry PI model B.

Author:
Carlos Ayerbe González

Date:
17 Jan 2017

Version:
1.0

This is a C library for Raspberry Pi (RPi). It provides access to GPIO and other IO functions on the Broadcom BCM 2835 chip. It provides functions for initilize and close I2C connection, also it provides to obtain the id of the sensors.

Definition in file **marte_pistorms.h**.

Macro Definition Documentation

#define BANK_A 1

Addreses of Pistorms' Banks

Definition at line 22 of file marte_pistorms.h.

#define PISTORMS_CLOSE_OK 1

Close success

Definition at line 40 of file marte_pistorms.h.

#define PISTORMS_ERROR_BAD_CONNECTOR "ERROR"

Incorrect Port or Bank

Definition at line 44 of file marte_pistorms.h.

#define PISTORMS_ERROR_CLOSE 0

Can not close

Definition at line 41 of file marte_pistorms.h.

#define PISTORMS_ERROR_INITIALIZATION 0

Can not be initialize

Definition at line 39 of file marte_pistorms.h.

#define PISTORMS_ERROR_NOT_INITIALIZED "ERROR"

Can not be initialize

Definition at line 45 of file marte_pistorms.h.

#define PISTORMS_ERROR_SENSOR_ID 0

Wrong sensor ID

Definition at line 42 of file marte_pistorms.h.

#define PISTORMS_ERROR_SENSOR_MODE 0

Wrong sensor Mode

Definition at line 43 of file marte_pistorms.h.

#define PISTORMS_ERROR_WRONG_CONNECTOR_ID -1

Incorrect Port or Bank

Definition at line 37 of file marte_pistorms.h.

#define PISTORMS_INITIALIZATION_OK 1

Initialization success

Definition at line 38 of file marte_pistorms.h.

#define PISTORMS_REASON_OK 1

PistormsCodes Pistorms codes for the result of some functions.

Success

Definition at line 36 of file marte_pistorms.h.

marte_pistorms_brick.c File Reference

Driver for control the Touch of EV3 Sensor.

```
#include "marte_pistorms.h"
#include "marte_pistorms_brick.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <time.h>
#include "marte_pistorms_internal.h"
```

Functions

int **_set_active_bank_brick** (int bank_id)
Sets the I2C slave address.

int **pistorms_brick_led_On** (int bank_id, int red, int green, int blue)
Writes to the specified RGB LED.

int **pistorms_brick_led_Off** (int bank_id)
Turn off the led.

char * **pistorms_brick_get_firmware_version** (int bank_id)
Returns the PiStorms firmware version.

char * **pistorms_brick_get_vendor_id** (int bank_id)
Returns the PiStorms vendor ID.

char * **pistorms_brick_get_device_id** (int bank_id)
Returns the PiStorms device ID.

int **pistorms_brick_get_battery_voltage** (void)
Obtains the input battery voltage.

int **pistorms_brick_get_key_press_value** (void)
Check if any button GO is pressed.

int **pistorms_brick_get_key_press_count** (void)
Obatins the GO button press count.

void **pistorms_brick_reset_key_press_count** (void)
Resets the GO button press count.

char * **pistorms_brick_touch_screen_X_asis** (void)
Obtain the value of the X asis.

char * **pistorms_brick_touch_screen_Y_asis** (void)
Obtain the value of the Y asis.

int **pistorms_brick_screen_is_touched** (void)
Detects if touchsreen is Touched.

Variables

char **bufData** [32] = {0}

char **bufAsisX** [10] = {0}
char **bufAsisY** [10] = {0}

Detailed Description

Driver for control the Touch of EV3 Sensor.

Author:

Carlos Ayerbe González

Date:

9 Feb 2017

Version:

1.0

Definition in file **marte_pistorms_brick.c**.

Function Documentation

int _set_active_bank_brick (int *bank_id*)

Sets the I2C slave address.

Parameters:

bank_id	Bank to configure.

Returns:

the value of active_bank.

Sets the I2C slave address (BANK_A or BANKB_B). But if the current slave address is the same than the connector_id, the function doesn't set again the slave address because it is not neccessary.

Definition at line 39 of file marte_pistorms_brick.c.

marte_pistorms_brick.h File Reference

Driver for control the brick of Pistorms.

Macros

#define **PISTORMS_FIRMWARE_VERSION** 0x00
#define **PISTORMS_VENDOR_ID** 0x08
#define **PISTORMS_DEVICE_ID** 0x10
#define **PISTORMS_LED_RED_VALUE** 0xD7
#define **PISTORMS_LED_GREEN_VALUE** 0xD8
#define **PISTORMS_LED_BLUE_VALUE** 0xD9
#define **PISTORMS_INPUT_BUTTON_VALUE** 0xDA
#define **PISTORMS_INPUT_BUTTON_COUNT** 0xDB
#define **PISTORMS_INPUT_TOUCH_SCREEN_X** 0xE3
#define **PISTORMS_INPUT_TOUCH_SCREEN_Y** 0xE5
#define **PISTORMS_INPUT_BATTERY_VOLTAGE** 0x6E

Functions

int **pistorms_brick_led_On** (int bank_id, int red, int green, int blue)
Writes to the specified RGB LED.

int **pistorms_brick_led_Off** (int bank_id)
Turn off the led.

char * **pistorms_brick_get_firmware_version** (int bank_id)
Returns the PiStorms firmware version.

char * **pistorms_brick_get_vendor_id** (int bank_id)
Returns the PiStorms vendor ID.

char * **pistorms_brick_get_device_id** (int bank_id)
Returns the PiStorms device ID.

int **pistorms_brick_get_battery_voltage** (void)
Obtains the input battery voltage.

int **pistorms_brick_get_key_press_value** (void)
Check if any button GO is pressed.

int **pistorms_brick_get_key_press_count** (void)
Obatins the GO button press count.

void **pistorms_brick_reset_key_press_count** (void)
Resets the GO button press count.

char * **pistorms_brick_touch_screen_X_asis** (void)
Obtain the value of the X asis.

char * **pistorms_brick_touch_screen_Y_asis** (void)
Obtain the value of the Y asis.

int **pistorms_brick_screen_is_touched** (void)
Detects if touchsreen is Touched.

Detailed Description

Driver for control the brick of Pistorms.

Author:

Carlos Ayerbe González

Date:

9 Feb 2017

Version:

1.0

This is a C library for Raspberry Pi (RPi). It provides control over brick of Pistorms. This sensor gives us the control of the leds, GO button , battery voltage and screen.

Definition in file **marte_pistorms_brick.h**.

marte_pistorms_internal.h File Reference

Library to add a debugger into the code.

Macros

#define **printf_dbg**(...)

Detailed Description

Library to add a debugger into the code.

Author:

Carlos Ayerbe González

Date:

8 Mar 2017

Version:

1.0

This file adds a debugger to the aplication, if you add the library and uncomment the constant "#define DBG" the aplication is going to run the debugger.

Definition in file **marte_pistorms_internal.h**.

marte_pistorms_motors.c File Reference

Drivers for motors from Pistorms + Raspberry PI model B.

```
#include "marte_pistorms.h"
#include "marte_pistorms_motors.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include "marte_pistorms_internal.h"
```

Macros

#define **PISTORMS_MOTOR_COMMANDS** 0x41
#define **PISTORMS_MOTOR_1_ENCODER_TARGET** 0x42
#define **PISTORMS_MOTOR_2_ENCODER_TARGET** 0x4A
#define **PISTORMS_MOTOR_1_SPEED** 0x46
#define **PISTORMS_MOTOR_2_SPEED** 0x4E
#define **PISTORMS_MOTOR_1_TIME** 0x47
#define **PISTORMS_MOTOR_2_TIME** 0x4F
#define **PISTORMS_MOTOR_1_COMMAND_REGISTER_A** 0x49
#define **PISTORMS_MOTOR_2_COMMAND_REGISTER_A** 0x51
#define **PISTORMS_MOTOR_1_ENCODER_POSITION** 0x52
#define **PISTORMS_MOTOR_2_ENCODER_POSITION** 0x56
#define **PISTORMS_MOTOR_1_FLOAT** 0x61
#define **PISTORMS_MOTOR_2_FLOAT** 0x62
#define **PISTORMS_MOTORS_SYNC_FLOAT** 0x63
#define **PISTORMS_MOTOR_1_BRAKE** 0x41
#define **PISTORMS_MOTOR_2_BRAKE** 0x42
#define **PISTORMS_MOTORS_SYNC_BRAKE** 0x43
#define **PISTORMS_MOTORS_PARAMETERS_RESET** 0x52
#define **PISTORMS_MOTOR_1_ENCODER_RESET** 0x72
#define **PISTORMS_MOTOR_2_ENCODER_RESET** 0x73

Functions

int **_set_sync_bank** (int bank_id)
Sets the I2C slave address.

int **pistorms_motor_go** (int connector_id, char go)
Turn on the motor with an indicated configuration.

long **pistorms_motor_get_pos** (int connector_id)
Obtains the current encoder position of the motor.

int **pistorms_motor_set_pos** (int connector_id, long pos)
Set up the encoder of the motor with a new position.

int **pistorms_motor_reset_pos** (int connector_id)
Resets the encoder position of the specified motor.

int **pistorms_motor_reset_all_parameters** (int bank_id)
Reset all Encoder values and motor parameters.

int **pistorms_motor_set_speed** (int connector_id, int speed)

Run the motor at a set speed for an unlimited duration.

int **pistorms_motor_set_running_time** (int connector_id, int seconds)
int **pistorms_motor_float** (int connector_id)

Stop the motor smoothly with float.

int **pistorms_motor_float_sync** (bank_id)
int **pistorms_motor_brake** (int connector_id)

Stop the motor abruptly with brake.

int **pistorms_motor_brake_sync** (int bank_id)

Stop both the motors of said bank at the same time motors are stopped abruptly with a brake.

Variables

char **motor_data** [32] = {0}
int **data**

Detailed Description

Drivers for motors from Pistorms + Raspberry PI model B.

Author:

Carlos Ayerbe González

Date:

16 Feb 2017

Version:

1.0

Definition in file **marte_pistorms_motors.c**.

Function Documentation

int _set_sync_bank (int *bank_id*)

Sets the I2C slave address.

Parameters:

connector_id	Bank to plug the motors.

Returns:

the value of active_bank, -1 if is incorrect.

Sets the I2C slave address (BANK_A or BANKB_B). But if the current slave address is the same than the connector_id, the function doesn't set again the slave address because it is not neccessary.

Definition at line 63 of file marte_pistorms_motors.c.

marte_pistorms_motors.h File Reference

Drivers for motors from Pistorms + Raspberry PI model B.

Macros

#define **SPEED_GO** 0x01
#define **RAMP_SPEED** 0x02
#define **CHANGE_BASED_ON_ENCODER** 0x04
#define **ENCODER_GO** 0x08
#define **BRAKE_FLOAT_MOVEMENT** 0x10
#define **ENCODER_ACTIVE_FEEDBACK** 0x20
#define **TIME_GO** 0x40
#define **MOTOR_GO** 0x80

Functions

int **pistorms_motor_go** (int connector_id, char go)
Turn on the motor with an indicated configuration.

long **pistorms_motor_get_pos** (int connector_id)
Obtains the current encoder position of the motor.

int **pistorms_motor_set_pos** (int connector_id, long pos)
Set up the encoder of the motor with a new position.

int **pistorms_motor_reset_pos** (int connector_id)
Resets the encoder position of the specified motor.

int **pistorms_motor_reset_all_parameters** (int bank_id)
Reset all Encoder values and motor parameters.

int **pistorms_motor_set_speed** (int connector_id, int speed)
Run the motor at a set speed for an unlimited duration.

int **pistorms_motor_set_secs** (int connector_id, int time)
Run motor in time mode.

int **pistorms_motor_float** (int connector_id)
Stop the motor smoothly with float.

int **pistorms_motor_float_sync** (int bank_id)
Stop both the motors of said bank at the same time motors are stopped smoothly with float.

int **pistorms_motor_brake** (int connector_id)
Stop the motor abruptly with brake.

int **pistorms_motor_brake_sync** (int bank_id)
Stop both the motors of said bank at the same time motors are stopped abruptly with a brake.

Detailed Description

Drivers for motors from Pistorms + Raspberry PI model B.

Author:
Carlos Ayerbe González

Date:
16 Feb 2017

Version:
1.0

This is a C library for Raspberry Pi (RPi). It provides control and functions for using motors in Pistorms brick.

Definition in file **marte_pistorms_motors.h**.

marte_pistorms_sensor_color.c File Reference

Driver for control the Color of EV3 Sensor.

```
#include "marte_pistorms.h"
#include "marte_pistorms_sensors.h"
#include "marte_pistorms_sensor_color.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include "marte_pistorms_internal.h"
```

Functions

int **pistorms_sensor_color_configure** (int connector_id)
Detects if the Color Sensor is connect correctly.

int **pistorms_color_set_mode** (int connector_id, int mode)
Configure the mode of the Color Sensor.

int **pistorms_color_measure** (int connector_id)
Color sensor recognizes seven colors.

int **pistorms_color_read_light** (int connector_id, int mode)
Color sensor can measure the intensity of light that enters the small window on the face of the sensor.

Variables

char * **read_data**

Detailed Description

Driver for control the Color of EV3 Sensor.

Author:
Carlos Ayerbe González

Date:
8 Feb 2017

Version:
1.0

Definition in file **marte_pistorms_sensor_color.c**.

marte_pistorms_sensor_color.h File Reference

Driver for control the Color of EV3 Sensor.

Macros

#define **COLOR_SENSOR_ID** "COL-REFLECT"
#define **REFLECTED_LIGHT** 0
#define **AMBIENT_LIGHT** 1
#define **MEASURE_COLOR** 2
#define **REFLECTED** 0
#define **AMBIENT** 1

Functions

int **pistorms_sensor_color_configure** (int connector_id)
Detects if the Color Sensor is connect correctly.

int **pistorms_color_set_mode** (int connector_id, int mode)
Configure the mode of the Color Sensor.

int **pistorms_color_read_light** (int connector_id, int mode)
Color sensor can measure the intensity of light that enters the small window on the face of the sensor.

int **pistorms_color_measure** (int connector_id)
Color sensor recognizes seven colors.

Detailed Description

Driver for control the Color of EV3 Sensor.

Author:
Carlos Ayerbe González

Date:
28 Mar 2017

Version:
1.0

This is a C library for Raspberry Pi (RPi). It provides control over color sensor. This sensor is a digital sensor that can detect the color or intensity of light that enters the small window on the face of the sensor. This sensor can be used in three different modes: Color Mode, Reflected Light Intensity Mode, and Ambient Light Intensity Mode.

Definition in file **marte_pistorms_sensor_color.h**.

marte_pistorms_sensor_gyro.c File Reference

Driver for control the Gyro of EV3 Sensor.

```
#include "marte_pistorms.h"
#include "marte_pistorms_sensors.h"
#include "marte_pistorms_sensor_gyro.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include "marte_pistorms_internal.h"
```

Functions

int **pistorms_sensor_gyro_configure** (int connector_id)
Detects if the Gyro Sensor is connect correctly.

int **pistorms_gyro_set_mode** (int connector_id, int mode)
Configure the mode of the Gyro Sensor.

short **pistorms_gyro_read** (int connector_id, int mode)
Read data of the Gyro Sensor depends on the mode.

Variables

char * **gyro_data**

Detailed Description

Driver for control the Gyro of EV3 Sensor.

Author:
Carlos Ayerbe González

Date:
8 Feb 2017

Version:
1.0

Definition in file **marte_pistorms_sensor_gyro.c**.

marte_pistorms_sensor_gyro.h File Reference

Driver for control the Gyro of EV3 Sensor.

Macros

#define **GYRO_SENSOR_ID** "GYRO-RATE"
#define **ANGLE** 0
#define **RATE** 1

Functions

int **pistorms_sensor_gyro_configure** (int connector_id)
Detects if the Gyro Sensor is connect correctly.

int **pistorms_gyro_set_mode** (int connector_id, int mode)
Configure the mode of the Gyro Sensor.

short **pistorms_gyro_read** (int connector_id, int mode)
Read data of the Gyro Sensor depends on the mode.

Detailed Description

Driver for control the Gyro of EV3 Sensor.

Author:
Carlos Ayerbe González

Date:
8 Feb 2017

Version:
1.0

This is a C library for Raspberry Pi (RPi). It provides control over gyro sensor. This sensor gives us the rotational motion on a single axis and the total rotation angle in degrees.

Definition in file **marte_pistorms_sensor_gyro.h**.

marte_pistorms_sensor_touch.c File Reference

Driver for control the Touch of EV3 Sensor.

```
#include "marte_pistorms.h"
#include "marte_pistorms_sensor_touch.h"
#include "marte_pistorms_sensors.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include "marte_pistorms_internal.h"
```

Functions

int **pistorms_sensor_touch_configure** (int connector_id)
int **pistorms_is_touched** (int connector_id)
check if the sensor is touched.

int **pistorms_num_touches** (int connector_id)
Count how many times the sensor was touched.

int **pistorms_reset_touches** (int connector_id)
Reset the count.

Detailed Description

Driver for control the Touch of EV3 Sensor.

Author:
Carlos Ayerbe González

Date:
6 Feb 2017

Version:
1.0

Definition in file **marte_pistorms_sensor_touch.c**.

marte_pistorms_sensor_touch.h File Reference

Driver for control the Touch of EV3 Sensor.

Macros

#define **TOUCH_SENSOR_ID** "Touch"

Functions

int **pistorms_sensor_configure_touch** (int connector_id)
Detects if the Touch Sensor is connect correctly.

int **pistorms_is_touched** (int connector_id)
check if the sensor is touched.

int **pistorms_num_touches** (int connector_id)
Count how many times the sensor was touched.

int **pistorms_reset_touches** (int connector_id)
Reset the count.

Detailed Description

Driver for control the Touch of EV3 Sensor.

Author:
Carlos Ayerbe González

Date:
6 Feb 2017

Version:
1.0

This is a C library for Raspberry Pi (RPi). It provides control over touch sensor. This sensor gives us if the sensor is touched or not, and the number of times it has been touched .

Definition in file **marte_pistorms_sensor_touch.h**.

marte_pistorms_sensor_ultrasonic.c File Reference

Driver for control the Ultrasonic of EV3 Sensor.

```
#include "marte_pistorms.h"
#include "marte_pistorms_sensors.h"
#include "marte_pistorms_sensor_ultrasonic.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include "marte_pistorms_internal.h"
```

Functions

int **pistorms_sensor_ultrasonic_configure** (int connector_id)
Detects if the Ultrasonic Sensor is connect correctly.

int **pistorms_ultrasonic_set_mode** (int connector_id, int mode)
Configure the mode of the Ultrasonic Sensor.

int **pistorms_ultrasonic_presence** (int connector_id)
Ultrasonic sensor can detect another Ultrasonic Sensor operating nearby.

float **pistorms_ultrasonic_read_distance** (int connector_id, int mode)
Ultrasonic sensor can measure the distance to an object in front of it.

Variables

char * **read_data**

Detailed Description

Driver for control the Ultrasonic of EV3 Sensor.

Author:
Carlos Ayerbe González

Date:
8 Feb 2017

Version:
1.0

Definition in file **marte_pistorms_sensor_ultrasonic.c**.

marte_pistorms_sensor_ultrasonic.h File Reference

Driver for control the Ultrasonic of EV3 Sensor.

Macros

#define **ULTRASONIC_SENSOR_ID** "US-DIST-CM"
#define **PROXIMITY_CENTIMETERS** 0
#define **PROXIMITY_INCHES** 1
#define **PRESENCE** 2
#define **CENTIMETERS** 0
#define **INCHES** 1

Functions

int **pistorms_sensor_ultrasonic_configure** (int connector_id)
Detects if the Ultrasonic Sensor is connect correctly.

int **pistorms_ultrasonic_set_mode** (int connector_id, int mode)
Configure the mode of the Ultrasonic Sensor.

float **pistorms_ultrasonic_read_distance** (int connector_id, int mode)
Ultrasonic sensor can measure the distance to an object in front of it.

int **pistorms_ultrasonic_presence** (int connector_id)
Ultrasonic sensor can detect another Ultrasonic Sensor operating nearby.

Detailed Description

Driver for control the Ultrasonic of EV3 Sensor.

Author:
Carlos Ayerbe González

Date:
8 Feb 2017

Version:
1.0

This is a C library for Raspberry Pi (RPi). It provides control over ultrasonic sensor. This sensor gives us the distance to an object in front of it. It does this by sending out high-frequency sound waves and measuring how long it takes the sound to reflect back to the sensor.(It could be in centimeters or in inches). Also this sensor can detect another Ultrasonic Sensor operating nearby. When listening for presence, the sensor detects sound signals but does not send them.

Definition in file **marte_pistorms_sensor_ultrasonic.h**.

marte_pistorms_sensors.c File Reference

Drivers for sensors from Pistorms + Raspberry PI model B.

```
#include "marte_pistorms.h"
#include "marte_pistorms_sensors.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <time.h>
#include "marte_pistorms_internal.h"
```

Functions

int **pistorms_port_set_type_sensor** (int connector_id, int type)
Determine sensor type on the specified port.

int **pistorms_sensor_get_mode** (int connector_id)
Obtain the mode that the EV3 Sensor is running.

int **pistorms_sensor_set_mode** (int connector_id, int mode)
Set the mode for the EV3 Sensor.

char * **pistorms_sensor_read** (int connector_id)
Read the data of EV3 sensors.

Variables

char **sensor_data** [32] = {0}
int **data**

Detailed Description

Drivers for sensors from Pistorms + Raspberry PI model B.

Author:
Carlos Ayerbe González

Date:
6 Feb 2017

Version:
1.0

Definition in file **marte_pistorms_sensors.c**.

marte_pistorms_sensors.h File Reference

Drivers for sensors from Pistorms + Raspberry PI model B.

Macros

#define **PORT_TYPE_1** 0x6F
#define **PORT_TYPE_2** 0xA3
#define **PORT_1_READY** 0x70
#define **PORT_1_SENSOR_ID** 0x71
#define **PORT_1_MODE** 0x81
#define **PORT_1_DATA** 0x83
#define **PORT_1_DATA_RESET** 0x84
#define **PORT_2_READY** 0xA4
#define **PORT_2_SENSOR_ID** 0xA5
#define **PORT_2_MODE** 0xB5
#define **PORT_2_DATA** 0xB7
#define **PORT_2_DATA_RESET** 0xB8
#define **TOUCH_TYPE** 18
#define **EV3_TYPE** 19

Functions

int **pistorms_port_set_type_sensor** (int connector_id, int type)
Determine sensor type on the specified port.

int **pistorms_sensor_get_mode** (int connector_id)
Obtain the mode that the EV3 Sensor is running.

int **pistorms_sensor_set_mode** (int connector_id, int mode)
Set the mode for the EV3 Sensor.

char * **pistorms_sensor_read** (int connector_id)
Read the data of EV3 sensors.

Detailed Description

Drivers for sensors from Pistorms + Raspberry PI model B.

Author:
Carlos Ayerbe González

Date:
6 Feb 2017

Version:
1.0

This is a C library for Raspberry Pi (RPi). It provides control and functions for using sensors in Pistorms brick.

Definition in file **marte_pistorms_sensors.h**.

Macro Definition Documentation

#define PORT_1_READY 0x70

Registers for EV3 Sensors

Definition at line 22 of file marte_pistorms_sensors.h.

#define PORT_TYPE_1 0x6F

I2C Registers for Sensor Modes

Definition at line 18 of file marte_pistorms_sensors.h.

Example Documentation

main_car.c

It is a car with two large motors , one touch sensor and one ultrasonic sensor, the car starts and when the ultrasonic sensor detects an object , the two motors stop, and the car go back and then it turns to avoid the object. See **main_car.c**