Pushing Performance

People | Power | Partnership

HARTING MICA® GPIO Library Documentation

1. Edition 2017, 10/01

This library documentation explains how to use the GPIO library and demonstrates the calls to configure the GPIOs.

Contents

1 Getting started

1.1 Prerequisites

For using the MICA® GPIO library, you need at least a container with a minimal Debian installation providing apt package management. You can find one on the HARTING MICA® website following *Technical Resources* → *Link to the MICA development containers* → *Certified* → *Latest* → *Debian*.

1. Download the container and follow the instructions for installation.
2. Start the Debian Container. Then, open its Web UI to get the instructions for its setup.
3. Install the SSH Server as root:
   `apt install openssh-server`
4. Make sure that an scp client is installed on your computer (e.g. WinSCP for Windows™).

1.2 Installation of the MICA® GPIO library

1. Copy the package **libmica-gpio.deb** into the /tmp directoty of your debian container, e.g. via WinSCP.
2. Install the package **libmica-gpio.deb** as root:
   `dpkg -i /tmp/libmica-gpio.deb`
3. If required, fix missing dependecies and repeat step 2. Make sure **libhidapi-libusb0** will be installed while resolving the dependencies, if it was not installed before.
   `apt -f install`

1.3 Installation of optional software

The given code examples at the end of this document are written in C / Python. Before you can run the examples you have to install the languages:

- If you want to use **c** with **gcc** (see chapter 2.3 for a code example):
  `apt-get update && apt-get install libc6-dev`
  `apt-get install gcc`
- If you want to use **python** (see chapter 2.4 for a code example):
  `apt-get install python3`

2 Interface of the library

The GPIOs can only be accessed by one container at a time. Make sure that all other containers using the GPIOs on your MICA® (e.g. the GPIO container) are stopped before using the library.

2.1 Types

Type	Description/Value
MICA_GPIO_DIRECTION	direction of a pin
enum	0 — INPUT 1 — OUTPUT
MICA_GPIO_STATE	state of a pin
enum	0 — LOW 1 — HIGH

2.2 Functions

enum MICA_GPIO_DIRECTION mica_gpio_get_direction(unsigned char id);
Gets the direction of a pin.

Parameters		
unsigned char	id	integer value between 1 and 8 (id of pin 1 to pin 8)

Return Value	
0 — INPUT	if direction was set to INPUT before
1 — OUTPUT	if direction was set to OUTPUT before
-1	if direction is not set or parameter is out of range

void mica_gpio_set_direction(unsigned char id, enum MICA_GPIO_DIRECTION direction);
Sets the direction of a pin.

Parameters		
unsigned char	id	integer value between 1 and 8 (id of pin 1 to pin 8)
enum MICA_GPIO_DIRECTION	direction	INPUT, OUTPUT

unsigned char mica_gpio_get_enable(unsigned char id);
Gets the enable state of a pin.

Parameters		
unsigned char	id	integer value between 1 and 8 (id of pin 1 to pin 8)

Return Value	
TRUE	if pin is enabled
FALSE	otherwise (OUTPUT pins are never enabled)

void mica_gpio_set_enable(unsigned char id, unsigned char enable);
Sets the enable state of an INPUT pin.
When the enable state is set to TRUE, a callback can be set to handle state changes.

Parameters		
unsigned char	id	integer value between 1 and 8 (id of pin 1 to pin 8)
unsigned char	enable	TRUE, FALSE

enum MICA_GPIO_STATE mica_gpio_get_state(unsigned char id);
Gets the state of a pin.

Parameters

unsigned char	id	integer value between 1 and 8 (id of pin 1 to pin 8)

Return Value

0 — LOW	if pin is off (INPUT pins are always off)
1 — HIGH	if pin is on
-1	if parameter is out of range

void mica_gpio_set_state(unsigned char id, enum MICA_GPIO_STATE state);
Sets the state of a pin.

Parameters

unsigned char	id	integer value between 1 and 8 (id of pin 1 to pin 8)
enum MICA_GPIO_STATE	state	LOW, HIGH

void *mica_gpio_set_callback(mica_gpio_callback callback, void *data);
Sets a callback for pins with direction:INPUT and enable:TRUE.

Parameters

mica_gpio_callback	callback	function that will be called when the state of an INPUT pin changes
void	*data	user defined data which will be passed back when the callback function is called; as the callback function will be called from a seperate thread, make sure that the pointer *data is still valid

typedef void (*mica_gpio_callback)(int id, enum MICA_GPIO_STATE state, void *data);
Defines the type of function used for the callback.

Parameters

int	id	integer value between 1 and 8 (id of pin 1 to pin 8)
enum	MICA_GPIO_STATE state	LOW, HIGH
void	*data	user defined data which was passed when setting the callback

2.3 Example in C

To run the example code, copy it to your debian container.

You can also download the code main.c directly from this file.

```
#include <stdio.h>
#include <unistd.h>

#include "mica_gpio.h"

#define FALSE 0
#define TRUE 1

void cb(int id, enum MICA_GPIO_STATE state, void *data) {
                printf("Input: %d state: %d changed (data: %s)\n", id, state, (char *) data);
                fflush(stdout);
}

int main(int argc, char* argv[]) {
                char *data = "user data";
                mica_gpio_set_callback(cb, data);

                mica_gpio_set_direction(1, INPUT);
                mica_gpio_set_enable(1, TRUE);
                sleep(5);
                mica_gpio_set_enable(1, FALSE);
                int state = mica_gpio_get_state(1);
                printf("State %d\n", state);

                mica_gpio_set_direction(1, OUTPUT);
                mica_gpio_set_state(1, HIGH);
                sleep(1);
                mica_gpio_set_state(1, LOW);

                mica_gpio_set_callback(NULL, NULL);
}
```

Compile the code:

```
gcc -l mica-gpio -o main main.c
```

Run the code:

```
root@Debian-mica40:/home/c# ./main
INFO: Initializing hardware ...
INFO: Initialization finished...
Input: 0 state: -1 changed (data: user data)
State 0
Input: -1 state: -1 changed (data: user data)
```

2.4 Example in Python

To run the example code, copy it to your debian container and make sure that the file is executable.

You can also download the code main.py directly from this file.

```
from enum import Enum
from time import sleep
from ctypes import CDLL, CFUNCTYPE, c_int, c_void_p, c_char_p

class Direction(Enum):
  INPUT=0
  OUTPUT=1

class State(Enum):
  LOW = 0
  HIGH= 1

lib = CDLL("libmica-gpio.so")
CMPFUNC = CFUNCTYPE(c_void_p, c_int, c_int, c_char_p)

def cb(id, state, data):
  print("Input: %d state: %d changed (data: '%s')\n" % (id, state > -1 and State(state) or state, data))

data = "user data"
lib.mica_gpio_set_callback(CMPFUNC(cb), data)

lib.mica_gpio_set_direction(1, Direction.INPUT.value)
lib.mica_gpio_set_enable(1, True)
sleep(5)
lib.mica_gpio_set_enable(1, False)
state = lib.mica_gpio_get_state(1)
print("State %s" % State(state).name)

lib.mica_gpio_set_direction(1, Direction.OUTPUT.value)
lib.mica_gpio_set_state(1, State.HIGH.value)
sleep(1)
lib.mica_gpio_set_state(1, State.LOW.value)
```

Run the code:

```
root@Debian-mica40:/home/python# python3 main.py
INFO: Initializing hardware ...
INFO: Initialization finished...
Input: 0 state: -1 changed (data: 'b'8\xdb\xbf'')

State LOW
```