REFMAN: bulk_led.c

Bernardo Fávero Andreeti, Eduardo Luzzi e José Augusto Comiotto Rottini

2.0
08/2014

Índice dos Arquivos

Lista de Arquivos

Esta é a lista de todos os arquivos e suas respectivas descrições:

Arquivos

Referência do Arquivo bulk_led_2.0.c

```
#include <stdio.h>
#include <sys/types.h>
#include <libusb-1.0/libusb.h>
```

Definições e Macros

- #define **DEV_ENDPOINT** 0x01
  DEV_ENDPOINT MCP2210 EndPoint.
- #define **HOST_ENDPOINT** 0x81
  HOST_ENDPOINT Computer EndPoint.
- #define **DEV_VID** 1240
  DEV_VID 0x81 User configurable.
- #define **DEV_PID** 222
  DEV_PID 0x81 Values for MCP2210.

Funções

- int **transfer_data** (libusb_device_handle *handle, unsigned char *data)
  transfer_data() *This function calls the bulk transfer available on libusb.*
- int **main** (void)

Variáveis

- **else**
- **return**

Definições e macros

#define DEV_ENDPOINT 0x01

DEV_ENDPOINT MCP2210 EndPoint.

#define DEV_PID 222

DEV_PID 0x81 Values for MCP2210.

#define DEV_VID 1240

DEV_VID 0x81 User configurable.

#define HOST_ENDPOINT 0x81

HOST_ENDPOINT Computer EndPoint.

Funções

int main (void)

libusb_init() Initialize library session.

libusb_set_debug() Set log message verbosity.

libusb_get_device_list() Get list of devices connected.

libusb_get_device_descriptor() Get device descriptor.

libusb_open_device_with_vid_pid() Try to get a handle to MCP2210 using corresponding VID and PID.

libusb_free_device_list() Releases the device.

libusb_claim_interface() Claim interface to MCP2210.

libusb_release_interface() Release the claimed interface.

libusb_close() Closes the library.

libusb_exit() Exit context.

```
78 {
79     libusb_device **list, *found = NULL;
80     libusb_device_handle *handle = NULL;
81     libusb_context *ctx = NULL;
82 
83     int r;
84     ssize_t cnt, i, n;
85 
86     unsigned char SetCS[64], SetSpiS[64], TxSpi[64];
87     unsigned char *SetChipSettings = SetCS, *SetSpiSettings = SetSpiS, *TransferSpiData
= TxSpi;
88 
89         /* SET CHIP SETTINGS POWER-UP DEFAULT */
90     SetChipSettings[0] = 0x60; // Set NVRAM Parameters Comand Code
91     SetChipSettings[1] = 0x20; // Set Chip Settings
92     SetChipSettings[2] = 0x00;
93     SetChipSettings[3] = 0x00;
94     for(n=4;n<13;n++)
95     {
96         SetChipSettings[n] = 0x01; // All GP's as Chip Select
97     }
98     SetChipSettings[13] = 0xFF; // GPIO Value
99     SetChipSettings[14] = 0xFF;
100     SetChipSettings[15] = 0xFF; // GPIO Direction
101     SetChipSettings[16] = 0xFF;
102     SetChipSettings[17] = 0x01; // Wake-up Disabled, No Interrupt Counting, SPI Bus is
Released Between Transfer
103     SetChipSettings[18] = 0x00; // Chip Settings not protected
104     for(n=19;n<64;n++)
105     {
106         SetChipSettings[n] = 0x00; // Reserved
107     }
108         /* SET SPI POWER-UP TRANSFER SETTINGS */
109     SetSpiSettings[0] = 0x60; // Set NVRAM Parameters Comand Code
110     SetSpiSettings[1] = 0x10; // Set SPI Transfer Settings
111     SetSpiSettings[2] = 0x00;
112     SetSpiSettings[3] = 0x00;
113     SetSpiSettings[4] = 0x80; // 4 Bytes to configure Bit Rate
114     SetSpiSettings[5] = 0x8D;
115     SetSpiSettings[6] = 0x5B;
116     SetSpiSettings[7] = 0x00; // 6.000.000 bps = 005B8D80 hex
117     SetSpiSettings[8] = 0xFF; // Idle Chip Select Value
```

```
118     SetSpiSettings[9] = 0xFF;
119     SetSpiSettings[10] = 0xEF; // Active Chip Select Value
120     SetSpiSettings[11] = 0xFF;
121     SetSpiSettings[12] = 0x00; // Chip Select to Data Delay (low byte)
122     SetSpiSettings[13] = 0x00; // Chip Select to Data Delay (high byte)
123     SetSpiSettings[14] = 0x00; // Last Data Byte to CS (low byte)
124     SetSpiSettings[15] = 0x00; // Last Data Byte to CS (high byte)
125     SetSpiSettings[16] = 0x00; // Delay Between Subsequent Data Bytes (low byte)
126     SetSpiSettings[17] = 0x00; // Delay Between Subsequent Data Bytes (high byte)
127     SetSpiSettings[18] = 0x03; // Bytes to Transfer per SPI Transaction (low byte)
128     SetSpiSettings[19] = 0x00; // Bytes to Transfer per SPI Transaction (high byte)
129     SetSpiSettings[20] = 0x00; // SPI mode 0
130     for(n=21;n<64;n++)
131     {
132         SetSpiSettings[n] = 0x00; // Reserved
133     }
134         /* TRANSFER SPI DATA */
135     TransferSpiData[0] = 0x42; // Transfer SPI Data Command Code
136     TransferSpiData[1] = 0x03; // Number of bytes to be transferred
137     TransferSpiData[2] = 0x00;
138     TransferSpiData[3] = 0x00; // Reserved
139     TransferSpiData[4] = 0x40; // SPI data to be sent
140     TransferSpiData[5] = 0x00;
141     TransferSpiData[6] = 0x00;
142     for(n=7;n<64;n++)
143     {
144         TransferSpiData[n] = 0xFF;
145     }
150     r = libusb_init(&ctx);
151     if (r < 0)
152         return r;
153
158     libusb_set_debug(ctx, 3);
159
164     cnt = libusb_get_device_list(ctx, &list);
165     if (cnt < 0)
166         return (int) cnt;
167
168     for (i = 0; i < cnt; i++)
169     {
170         libusb_device *device = list[i];
171
172         struct libusb_device_descriptor desc;
173
178         libusb_get_device_descriptor(device, &desc);
179
180         if (desc.idVendor == DEV_VID && desc.idProduct == DEV_PID)
181         {
182             found = device;
183             printf("Found MCP2210 connected to the system!\n");
184             break;
185         }
186     }
187     if (found)
188     {
193         handle = libusb open device with vid pid(ctx, DEV VID, DEV PID);
194         if(handle == NULL)
195             printf("Error opening device!\n\t-ERROR CODE: %d\n",r);
196         else
197             printf("Device opened.\n\n");
198     }
199     else
200     {
201         printf("Device not found, exiting...\n");
206         libusb_free_device_list(list, 1);
207         return 1;
208     }
209
210     libusb free device list(list, 1); // releases the device
211
```

```
212     if(libusb_kernel_driver_active(handle,0) == 1) // find out if kernel driver is
attached
213     {
214         printf("\tKernel Driver Active, Detaching...\n");
215         if(libusb detach kernel driver(handle,0) == 0)
216             printf("\t\t->Kernel Driver Detached!\n");
217     }
222     r = libusb_claim_interface(handle,0); // claim interface 0 to MCP2210
223     if(r<0)
224     {
225         printf("Could not claim interface, exiting...\n");
226         libusb_close (handle);
227         libusb_exit(ctx);
228         return 1;
229     }
230     printf("\t->Claimed interface!\n");
231         // First Step: Write command to Configure all GP's as CS
232     r = transfer_data(handle, SetChipSettings);
233     if(r == 1)
234     {
235         libusb close (handle);
236         libusb_exit(ctx);
237         return 1;
238     }
239         // Second Step: Set SPI settings and select MCP23S08 (GP4=0)
240     r = transfer_data(handle, SetSpiSettings);
241     if(r == 1)
242     {
243         libusb_close (handle);
244         libusb_exit(ctx);
245         return 1;
246     }
247         // Third Step: Send commands and data to MCP23S08
248     while (1)
249     {
250         r = transfer_data(handle, TransferSpiData);
251         if(r == 2)
252             break; // SPI data transfer completed
253
254         else if(r == 1)
255         {
256             libusb_close (handle);
257             libusb_exit(ctx);
258             return 1;
259         }
260     }
261     TransferSpiData[4] = 0x40; TransferSpiData[5] = 0x0A; TransferSpiData[6] = 0xFF;//
SPI data to be sent
262
263     while(1)
264     {
265         r = transfer_data(handle, TransferSpiData);
266         if(r == 2)
267             break; // SPI data transfer completed
268
269         else if(r == 1)
270         {
271             libusb_close (handle);
272             libusb_exit(ctx);
273             return 1;
274         }
275     }
276     printf("\t\t->Data Sent\n");
281     r = libusb_release_interface(handle, 0);
282     if(r!=0)
283     {
284             printf("Cannot Release Interface\n");
285         libusb close (handle);
286         libusb exit(ctx);
287             return 1;
288     }
```

```
289     printf("Released Interface\n");
294     libusb_close(handle);
299     libusb_exit(ctx);
300     return 0;
301 }
```

int transfer_data (libusb_device_handle * *handle*, unsigned char * *data*)

transfer_data() This function calls the bulk transfer available on libusb.

libusb_bulk_transfer() Send data to MCP2210.

libusb_bulk_transfer() Receives device response.

```
43 {
44     int byte_count, rslt;
45     unsigned char ReceivedData[64];
46     unsigned char *Response = ReceivedData;
47
52     rslt = libusb_bulk_transfer(handle, DEV_ENDPOINT, data, 64, &byte_count, 0);
57         rslt = libusb_bulk_transfer(handle, HOST_ENDPOINT, Response, 64, &byte_count,
0);
58         if(rslt == 0 && byte_count == 64) // if successfully received all bytes
59         {
60             if(ReceivedData[0]==0x42 && ReceivedData[1]==0x00 && ReceivedData[3]==0x10)
61                 return 2;
62         }
63         else
64         {
65             printf("Reading Error! ERROR CODE = %d\n", rslt);
66             return 1;
67         }
68     }
```
