REFMAN: bulk_temp.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:

bulk_temp_2.0.c 3

Arquivos

Referência do Arquivo bulk_temp_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)

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.

```
{
    libusb_device **list, *found = NULL;
    libusb_device_handle *handle = NULL;
    libusb_context *ctx = NULL;

    int r;
    ssize_t cnt, i, n, c=0;

    unsigned char SetCS[64], SetSpiS[64], TxSpi[64];
    unsigned char *SetChipSettings = SetCS, *SetSpiSettings = SetSpiS, *TransferSpiData = TxSpi;

        /* SET CHIP SETTINGS POWER-UP DEFAULT */
    SetChipSettings[0] = 0x60; // Set NVRAM Parameters Comand Code
    SetChipSettings[1] = 0x20; // Set Chip Settings
    SetChipSettings[2] = 0x00;
    SetChipSettings[3] = 0x00;
    for(n=4;n<13;n++)
    {
        SetChipSettings[n] = 0x01; // All GP's as Chip Select
    }
    SetChipSettings[13] = 0xFF; // GPIO Value
    SetChipSettings[14] = 0xFF;
    SetChipSettings[15] = 0xFF; // GPIO Direction
    SetChipSettings[16] = 0xFF;
    SetChipSettings[17] = 0x01; // Wake-up Disabled, No Interrupt Counting, SPI Bus is Released Between Transfer
    SetChipSettings[18] = 0x00; // Chip Settings not protected
    for(n=19;n<64;n++)
    {
        SetChipSettings[n] = 0x00; // Reserved
    }
        /* SET SPI POWER-UP TRANSFER SETTINGS */
    SetSpiSettings[0] = 0x60; // Set NVRAM Parameters Comand Code
    SetSpiSettings[1] = 0x10; // Set SPI Transfer Settings
    SetSpiSettings[2] = 0x00;
    SetSpiSettings[3] = 0x00;
    SetSpiSettings[4] = 0x80; // 4 Bytes to configure Bit Rate
    SetSpiSettings[5] = 0x8D;
    SetSpiSettings[6] = 0x5B;
    SetSpiSettings[7] = 0x00; // 6.000.000 bps = 005B8D80 hex
    SetSpiSettings[8] = 0xFF; // Idle Chip Select Value
    SetSpiSettings[9] = 0xFF;
    SetSpiSettings[10] = 0x7F; // Active Chip Select Value, GP7 = 0
    SetSpiSettings[11] = 0xFF;
    SetSpiSettings[12] = 0x00; // Chip Select to Data Delay (low byte)
    SetSpiSettings[13] = 0x00; // Chip Select to Data Delay (high byte)
    SetSpiSettings[14] = 0x00; // Last Data Byte to CS (low byte)
    SetSpiSettings[15] = 0x00; // Last Data Byte to CS (high byte)
    SetSpiSettings[16] = 0x00; // Delay Between Subsequent Data Bytes (low byte)
    SetSpiSettings[17] = 0x00; // Delay Between Subsequent Data Bytes (high byte)
    SetSpiSettings[18] = 0x02; // Bytes to Transfer per SPI Transaction (low byte)
    SetSpiSettings[19] = 0x00; // Bytes to Transfer per SPI Transaction (high byte)
    SetSpiSettings[20] = 0x00; // SPI mode 0
    for(n=21;n<64;n++)
    {
        SetSpiSettings[n] = 0x00; // Reserved
    }
        /* TRANSFER SPI DATA */
    TransferSpiData[0] = 0x42; // Transfer SPI Data Command Code
    TransferSpiData[1] = 0x02; // Number of bytes to be transferred
    TransferSpiData[2] = 0x00;
    TransferSpiData[3] = 0x00; // Reserved
    TransferSpiData[4] = 0x00; // SPI data to be sent
    TransferSpiData[5] = 0x00;
    TransferSpiData[6] = 0xFF;
    for(n=7;n<64;n++)
    {
        TransferSpiData[n] = 0xFF;
    }
    r = libusb_init(&ctx); // initialize library session
    if (r < 0)
        return r;
    libusb_set_debug(ctx, 3);
    cnt = libusb_get_device_list(ctx, &list); // get list of devices connected
    if (cnt < 0)
        return (int) cnt;

    for (i = 0; i < cnt; i++)
    {
        libusb_device *device = list[i];

        struct libusb_device_descriptor desc;
        libusb_get_device_descriptor(device, &desc); // get device descriptor

        if (desc.idVendor == DEV_VID && desc.idProduct == DEV_PID)
        {
            found = device;
            printf("Found MCP2210 connected to the system!\n");
            break;
        }
    }
    if (found)
    {
        handle = libusb_open_device_with_vid_pid(ctx, DEV_VID, DEV_PID); // try to get a handle to MCP2210 using corresponding VID and PID
        if(handle == NULL)
            printf("Error opening device!\n\t-ERROR CODE: %d\n",r);
        else
            printf("Device opened.\n\n");
    }
    else
    {
        printf("Device not found, exiting...\n");
        libusb_free_device_list(list, 1);
        return 1;
    }

    libusb_free_device_list(list, 1); // releases the device

    if(libusb_kernel_driver_active(handle,0) == 1) // find out if kernel driver is attached
    {
        printf("\tKernel Driver Active, Detaching...\n");
        if(libusb_detach_kernel_driver(handle,0) == 0)
```

```
            printf("\t\t->Kernel Driver Detached!\n");
    }
    r = libusb_claim_interface(handle,0); // claim interface 0 to MCP2210
    if(r<0)
    {
        printf("Could not claim interface, exiting...\n");
        libusb_close (handle);
        libusb_exit(ctx);
        return 1;
    }
    printf("\t->Claimed interface!\n");
        // First Step: Write command to Configure all GP's as CS
    r = transfer_data(handle, SetChipSettings);
    if(r == 1)
    {
        libusb_close (handle);
        libusb_exit(ctx);
        return 1;
    }
        // Second Step: Set SPI settings and select TC77 (GP7=0)
    r = transfer_data(handle, SetSpiSettings);
    if(r == 1)
    {
        libusb_close (handle);
        libusb_exit(ctx);
        return 1;
    }
    // Third Step: Temperature read
    while(1)
    {
        r = transfer_data(handle, TransferSpiData);
        sleep(1);
        if(r == 1)
        {
            libusb_close (handle);
            libusb_exit(ctx);
            return 1;
        }
    }
    r = libusb_release_interface(handle, 0); //release the claimed interface
    if(r!=0)
    {
            printf("Cannot Release Interface\n");
        libusb_close (handle);
        libusb_exit(ctx);
            return 1;
    }
    printf("Released Interface\n");
    libusb_close(handle); // closes the library
    libusb_exit(ctx); // exit context
    return 0;
}
```

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.

```
{
    int byte_count, rslt, sign, temp;
    double tempC;
    unsigned char ReceivedData[64];
    unsigned char *Response = ReceivedData;

    rslt = libusb_bulk_transfer(handle, DEV_ENDPOINT, data, 64, &byte_count, 0);
```

```c
    if(rslt == 0 && byte_count == 64)
    {
        rslt = libusb_bulk_transfer(handle, HOST_ENDPOINT, Response, 64, &byte_count, 0);
        if(rslt == 0 && byte_count == 64) // successfully received all bytes
        {
            if(ReceivedData[0]==0x42 && ReceivedData[1]==0x00 && ReceivedData[2] == 0x02 && ReceivedData[3]==0x10) // condition for a new temperature read
            {
                sign = ReceivedData[4] & 0x80;

                if(sign == 0)
                    temp = (ReceivedData[4] << 8 | ReceivedData[5]) >> 3;
                else
                    temp = (((ReceivedData[4] & 0x7F) << 8 | ReceivedData[5]) >> 3) - 4096;
                tempC = temp;
                tempC = tempC * 0.0625; // conversion to celsius

                printf("-> Temperature = %.2f Celsius\n", tempC);
            }
            return 2;
        }
        else
        {
            printf("Reading Error! ERROR CODE = %d\n", rslt);
            return 1;
        }
    }
    else
    {
        printf("Writing Error!\n");
        return 1;
    }
    return 0;
}
```