一、编译动态链接库
编译动态链接库之前
1.安装 libusb-1.0
$sudo apt-get install libusb-1.0-0-dev
2.安装 g++编译器
$sudo apt-get install g++
二、编译动态链接库并配置到系统中
1.编译库文件
进入目录
$make -f Make-lib-linux
2.配置路径
复制动态链接库到系统动态库目录:
$sudo cp libslabhidtouart.so.1.0 /usr/lib/
创建.so 链接:
$sudo ln -sf /usr/lib/libslabhidtouart.so.1.0 /usr/lib/libslabhidtouart.so
创建.so.1 链接:
$sudo ln -sf /usr/lib/libslabhidtouart.so.1.0 /usr/lib/libslabhidtouart.so.1
到此可以在 pc 机上编写自己的 pci 通信程序，直接使用 pc 对 pci 通信进行调试
3.配置交叉编译器的 libusb-1.0.so.0.1.0 动态库
到开发板上找到 libusb-1.0.so.0.1.0 库文件
#find /usr/lib -name libusb-1.0*
执行后看到三个文件，其中一个库文件，两个链接文件，拷贝库文件到 pc，并创建格式相同的链接文件
#mount -o nolock 192.168.1.100:/挂载目录 /mnt

$sudo cp libusb-1.0.so.0.1.0 /usr/lib
$sudo ln -sf libusb-1.0.so.0.1.0 libusb-1.0.so
$sudo ln -sf libusb-1.0.so.0.1.0 libusb-1.0.so.0
到此交叉编译器编译时可以正常链接到 libusb-1.0.so.0.1.0 库
4.配置交叉编译器的 libslabhidtouart.so.1.0 动态库
修改 Makefile-lib-linux 脚本，将编译器改为交叉编译器，后执行
$make -f Make-linux-lib
拷贝 libslabhidtouart.so.1.0 动态库到交叉编译器的指定目录
$sudo cp libslabhidtouart.so.1.0 /develop-usr/gcc-linaro-arm-linux-gnueabihf-4.7-2013.04-20130415_linux/arm-linux-gnueabihf/lib/
$cd /develop-usr/gcc-linaro-arm-linux-gnueabihf-4.7-2013.04-20130415_linux/arm-linux-gnueabihf/lib/
$sudo ln -sf libusb-1.0.so.0.1.0 libusb-1.0.so
$sudo ln -sf libusb-1.0.so.0.1.0 libusb-1.0.so.0
到此交叉编译器编译时可以正常链接到 libusb-1.0.so.0.1.0 库

三、接口 API
/***/

pc 通信 API，已经全部封装到了 OperateCP2110.h OperateCP2110.cpp 中，使用两个文件的同时，需要包含同目录下的 include 文件夹下面的头文件
1.打开 pci 设备
BOOL OpenDevice()
执行成功返回逻辑 1，失败返回逻辑 0
2.关闭 pci 设备
BOOL CloseDevice()
执行成功返回逻辑 1，失败返回逻辑 0
3.将主机与 pci 的字节流同步，保证主机与 pci 之间的写入、读出是同步的

BOOL SyncDataStream()

执行成功返回逻辑 1，失败返回逻辑 0

4.向 0-31 号寄存器写入数据，其中 0-31 号寄存器可读可写，32-52 号寄存器只写，其中 6、9、10、15、19、25、29、30、31、34-47 号寄存器未被使用

BOOL Write1_31Register(int reg,int data);

参数 int reg 为可以使用的寄存器编号，参数 int data 是要写入的值，其范围参考相关文档

执行成功返回逻辑 1，失败返回逻辑 0

5.Ramp 指针归零

BOOL SetRampPointer()

执行成功返回逻辑 1，失败返回逻辑 0

6.向 Ramp 表写入数据，没写一次，表的数据位置指针自增 1

BOOL WriteRampTable(int value)

执行成功返回逻辑 1，失败返回逻辑 0

7.将当前运行环境的参数保存到 flash

BOOL Sava2Flash()

执行成功返回逻辑 1，失败返回逻辑 0

8.加载 flash 中保存的参数到寄存器

BOOL RestoreRegsitersFromFlash()

执行成功返回逻辑 1，失败返回逻辑 0

9.加载 flash 中的数据表

BOOL RestoreTableFromFlash()

执行成功返回逻辑 1，失败返回逻辑 0

10.读取 0-31 号寄存器的值及其校验和

BOOL ReadRegisters(BYTE* &registerDataArray)

参数 BYTE* &registerDataArray 为至少 65 个字节长度的数组首地址，前 64 字节为 32 个 16 位寄存器的值，MSB 在前，最后两个字节为校验和

执行成功返回逻辑 1，失败返回逻辑 0

11.将运行环境的系统参数恢复为默认值

BOOL RestoreDefaults()

执行成功返回逻辑 1，失败返回逻辑 0

/**/

四、测试程序

进入测试程序文件夹，修改编译器为目标平台编译器

$make

$file testfile

若编译器为交叉编译器，返回的字符串中包含 ARM 字符，得到正确的板级运行程序

若编译器为 pc 机编译器，返回的字符串中包含 x86 字符，得到正确的 pc 机运行程序