用 tslib 为 MiniGUI 提供触摸屏的 IAL 引擎

（陈云川 ybc2084@163.com UESTC,CD 2007-04-16）

摘要：本文给出了一种通过tslib为MiniGUI提供触摸屏的IAL引擎的方法。测试情况表明，该方法能够有效工作。同时，由于tslib作为高层接口所具有的抽象能力，这种方法也应该能够在多种其它平台上工作。

关键字：MiniGUI、触摸屏、IAL引擎、tslib、ARM-Linux

1 引言

GAL（Graphic Abstract Layer，图形抽象层）和 IAL（Input Abstract Layer，输入抽象层）是 MiniGUI 的两个基础设施，MiniGUI 的高度可移植性在很大程度上也是由于这两个接口提供了独立于硬件的抽象能力。但是 GAL 和 IAL 并不是生来就存在的，它们也需要通过底层代码来实现，通常把实现这两个接口的底层代码称之为“图形引擎”和“输入引擎”。GAL 和 IAL 与 MiniGUI 的关系，就好比驱动程序与操作系统的关系一样。那么，为什么本文只提 IAL 引擎的实现而不提 GAL 引擎的实现呢？原因是这样的：Linux 系统提供了一种基础设施——FrameBuffer，通过这个设施，Linux 下的图形输出有了统一的接口。FrameBuffer 既在桌面系统中存在，也大量的存在于基于 Linux 的嵌入式系统中。由于 MiniGUI 可以通过 FrameBuffer 获得统一的图形引擎接口，因此其适应性大大提高，也就很少需要再为 MiniGUI 编写 GAL 驱动层了。但是对于输入而言，由于 Linux 没有提供统一的接口，因此常常不得不为 MiniGUI 编写 IAL 驱动层。

在参考文献[1]中，对如何开发定制的输入引擎做了详细的说明。但作者不打算沿袭其套路，而是采用另外一种方法来为MiniGUI构建输入引擎——事实表明，这种做法是有效的，并且在某种程度上更具通用性。

2 实现过程

为了便于叙述，以下在提到源代码路径时，都表示是相对于/home/cyc/minigui/而言的。比如：libminigui-1.3.3/src/ial/ial.c 就等价于/home/cyc/minigui/libminigui-1.3.3/src/ial/ial.c。

2.1 MiniGUI 的 IAL 接口定义

首先来看看 MiniGUI 中的 IAL 接口是如何定义的（见 libminigui-1.3.3/src/include/ial.h）：

```
typedef struct tagINPUT
{
    char*     id;
```

```
        // Initialization and termination
        BOOL (*init_input) (struct tagINPUT *input, const char* mdev, const char* mtype);
        void (*term_input) (void);

        // Mouse operations
        int    (*update_mouse) (void);
        void (*get_mouse_xy) (int* x, int* y);
        void (*set_mouse_xy) (int x, int y);
        int    (*get_mouse_button) (void);
        void (*set_mouse_range) (int minx, int miny, int maxx, int maxy);
        void (*suspend_mouse) (void);
        int (*resume_mouse) (void);

        // Keyboard operations
        int    (*update_keyboard) (void);
        const char* (*get_keyboard_state) (void);
        void (*suspend_keyboard) (void);
        int (*resume_keyboard) (void);
        void (*set_leds) (unsigned int leds);

        // Event
#ifdef _LITE_VERSION
        int (*wait_event) (int which, int maxfd, fd_set *in, fd_set *out, fd_set *except,
                        struct timeval *timeout);
#else
        int (*wait_event) (int which, fd_set *in, fd_set *out, fd_set *except,
                        struct timeval *timeout);
#endif

        char mdev [MAX_PATH + 1];
}INPUT;

extern INPUT* cur_input;
```

MiniGUI 通过一个 INPUT 结构体来表示一个输入引擎。下面简要介绍一下后面将会用到的 INPUT 结构体的各个成员。

- id　　输入引擎的名称；
- init_input　　输入引擎的初始化函数；
- term_input　　输入引擎的终止函数；
- update_mouse　　更新鼠标位置；
- get_mouse_button　　获取鼠标按键值；

- wait_event 等待输入事件到来，可以是键盘，也可以是鼠标。

指针 cur_input 是一个全局变量，用来指向 MiniGUI 所用的输入引擎。所有获取键盘消息或者鼠标按键消息的操作都是通过该指针来完成的。另外，在 INPUT 结构体中定义的实际上都是一些函数指针，在 MiniGUI 启动的时候，MiniGUI 将会自动根据配置文件 MiniGUI.cfg 中的配置信息来决定应该将哪些函数的入口地址赋给这些指针。也就是说，将函数指针指向具体函数的工作是在 init_input()函数中完成的。

由于本章只用到触摸屏输入，而不会用到键盘，因此，只需为鼠标相关的函数指针赋值即可。通常的做法都是在 init_input()中打开鼠标设备（比如/dev/mouse），然后为 INPUT 结构中的某些函数指针赋予相应的函数入口地址。此后，当 MiniGUI 要获取鼠标或者键盘消息时，实际上是通过 INPUT 结构体中的函数指针调用相应的函数。

2.2 tslib 的 API 接口

作者的想法是通过 tslib 来构建 MiniGUI 的输入引擎。tslib 是一个用于触摸屏设备的函数库。通过这样一个函数库，可以将编程者从繁琐的数据处理中解脱出来。为什么会出现这样的情况呢？因为触摸屏的坐标和液晶显示屏之间的坐标并不是一一对应的，所以，要让从触摸屏上得到的坐标正确转换为液晶显示屏上的坐标，需要经过一个转换过程。除此之外，tslib 还以插件的形式提供了一些附加的功能，比如去除点击触摸屏时的抖动等。

下面就先来看看 tslib 都提供了哪些接口。首先，tslib 中定义了两个结构体：

```
struct tsdev {
	int fd;
	struct tslib_module_info *list;
};
struct ts_sample {
	int	x;
	int	y;
	unsigned int	pressure;
	struct timeval	tv;
};
```

struct tsdev 表示的是触摸屏设备，其中，fd 是打开的触摸屏设备的文件描述符，list 是一个链表，在这个链表中依次存放着指向 tslib 的插件的指针。

struct ts_sample 用于存放按键消息，x 和 y 表示按键的坐标位置，注意这两个坐标都是以液晶屏幕的坐标系为基准的，pressure 表示的是按键的轻重程度，tv 是一个 struct timeval 类型的变量，表示按键发生的时间。

在基于 tslib 的程序中，所要用到的数据结构就只有上面这样两个。下面再看看 tslib 的函数，这里只列举后面将会用到的几个函数：

```
struct tsdev *ts_open(const char *dev_name, int nonblock);
int ts_config(struct tsdev *);
int ts_close(struct tsdev *);
```

```
int ts_fd(struct tsdev *);
int ts_read(struct tsdev *, struct ts_sample *, int);
```

ts_open()函数打开触摸屏设备，第一个参数 dev_name 是要打开的触摸屏设备的文件名，第二个参数 nonblock 指明以何种方式读写触摸屏设备，如果 nonblock 非 0，则以非阻塞方式访问，如果 nonblock 为 0，则以阻塞方式访问。如果 ts_open()打开触摸屏设备成功，则返回一个 struct tsdev 类型的指针，否则返回 NULL。

ts_config()函数的作用是读取触摸屏配置文件，并决定是否加载相关的插件。在作者所用的平台上，触摸屏配置文件为/etc/ts.conf，其内容如下：

```
module variance xlimit=50 ylimit=50 pthreshold=1
module dejitter xdelta=10 ydelta=10 pthreshold=1
module linear
```

上述内容表明 tslib 要加载三个插件模块，其作用分别是限定点击力度的方差、去除点击抖动、将触摸屏上的点击坐标转换成液晶屏幕上的坐标。variance 模块会采集四个采样点并计算其方差，只有当这四个采样点在 x 轴和 y 轴方向上的方差都小于或等于模块参数 xlimit 和 ylimit 限定的范围时，这几个采样点才会被接受，否则 variance 模块将重新采集四个采样点计算。直到有符合要求的采样点才会将其递交给应用程序。dejitter 模块的作用是去抖动，其参数的含义用当前的采样点的坐标与前一个采样点的坐标求差值，只有当差值小于或者等于 xdelta 和 ydelta 限定的范围时，采样点的坐标值才有效。

如果正确加载了插值模块，ts_config()返回 0，否则返回-1。

ts_close()的作用是释放触摸屏设备以及相关的资源。

ts_fd()返回打开的触摸屏设备的文件描述符。

ts_read()的作用是从触摸屏设备中读取采样点的坐标。第一个参数指向一个已经打开的触摸屏设备，第二个参数是一个 struct ts_sample 指针，从触摸屏设备读取到的值将填充到该指针指向的空间中，第三个参数指定了要读取多少个采样点。

有两个文件是与 tslib 密切相关的，这两个文件都位于/etc/目录下：ts.conf 指出了 MiniGUI 应该加载哪些插件模块；另一个文件是 pointercal，在这个文件中包含了触摸屏的校准数据（calibration），正是由于该文件的存在，tslib 才能正确地在触摸屏坐标和液晶屏幕的坐标之间进行转换。在 tslib 的源代码中包含了工具 ts_calibrate，如果没有 pointercal 文件，那么可以用这个工具来产生 pointercal 文件中的校准数据。

2.3 交叉编译 tslib

在看完了 tslib 提供的编程接口之后，下面要做的事情就是为 ARM-Linux 编译 tslib，这并不困难。首先，从网上下载 tslib.tar.bz2，下载之后解压并解包。本章中，假定 tslib.tar.bz2 和解包得到的 tslib/目录都位于/home/cyc/minigui/目录下，依次执行下面的命令：

```
[root@cyc tslib]# ./configure --host=arm-linux --prefix=/home/cyc/minigui/tslib/arm-linux-build/
```

```
[root@cyc tslib]# make
[root@cyc tslib]# make install
```

tslib 的交叉编译就完成了，编译生成的文件被安装到了当前目录下的 arm-linux-build/中，如下所示：

```
[root@cyc tslib]# cd arm-linux-build/
[root@cyc arm-linux-build]# ls
bin  etc  include  lib  share
[root@cyc arm-linux-build]# ls bin/
ts_calibrate  ts_print  ts_test
[root@cyc arm-linux-build]# ls etc/
ts.conf  ts.conf~
[root@cyc arm-linux-build]# ls include/
tslib.h
[root@cyc arm-linux-build]# ls lib/
libts-0.0.so.0  libts-0.0.so.0.1.0  libts.a  libts.la  libts.so
```

几个必须的文件是：ts.conf，应该把它复制到目标板的/etc/目录下；libts-0.0.so.0、libts-0.0.so.0.1.0、libts.so，这三个文件应该被复制到目标板的/usr/local/lib/目录下。

如果需要对触摸屏进行校准，那么还需要把 ts_calibrate 也拷贝到目标板上，具体位置可以任选，拷贝到目标板上之后将文件属性改为可执行即可。ts_print 和 ts_test 可以用来对生成的 tslib 进行测试，读者可以根据自己的需要决定是否将其拷贝到目标板上。

2.4 改写 IAL 引擎

完成了对 tslib 的交叉编译之后，下一步的事情就是改写 MiniGUI 的 IAL 引擎。MiniGUI 自带的 IAL 输入引擎中，有一个叫做 SMDK2410。为了尽可能简单，作者决定在其基础上稍作修改，使之符合我们的要求即可。修改后得到的文件全貌如下（libminigui-1.3.3/src/ial/2410.c）：

```
25 #include <stdio.h>
26 #include <stdlib.h>
27 #include <string.h>
28 #include <unistd.h>
29 #include <fcntl.h>
30
31 #include "common.h"
32 #include "tslib.h"
33
34 #ifdef _SMDK2410_IAL
35
36 #include <sys/ioctl.h>
```

```
37 #include <sys/poll.h>
38 #include <sys/types.h>
39 #include <sys/stat.h>
40 #include <linux/kd.h>
41
42 #include "ial.h"
43 #include "2410.h"
44
45 //#ifndef  _DEBUG
46 //#define  _DEBUG                    // for debugging
47 //#endif
48
49 #ifdef     _DEBUG
50 #undef     _DEBUG                    // for release
51 #endif
52
53 /* for storing data reading from /dev/input/event1 */
54 typedef struct {
55         unsigned short pressure;
56         unsigned short x;
57         unsigned short y;
58         unsigned short pad;
59 } TS_EVENT;
60
61 static unsigned char state [NR_KEYS];
62 static int mousex = 0;
63 static int mousey = 0;
64 static TS_EVENT ts_event;
65 static struct tsdev *ts;
66
67 /************************   Low Level Input Operations **********************/
68 /*
69  * Mouse operations -- Event
70  */
71 static int mouse_update(void)
72 {
73         return 1;
74 }
75
76 static void mouse_getxy(int *x, int* y)
77 {
78         if (mousex < 0) mousex = 0;
79         if (mousey < 0) mousey = 0;
80         if (mousex > 639) mousex = 639;
```

```
 81         if (mousey > 479) mousey = 479;
 82
 83 #ifdef _DEBUG
 84         printf ("mousex = %d, mousey = %d\n", mousex, mousey);
 85 #endif
 86
 87         *x = mousex;
 88         *y = mousey;
 89 }
 90
 91 static int mouse_getbutton(void)
 92 {
 93         return ts_event.pressure;
 94 }
 95
 96 #ifdef _LITE_VERSION
 97 static int wait_event (int which, int maxfd, fd_set *in, fd_set *out, fd_set *except,
 98                         struct timeval *timeout)
 99 #else
100 static int wait_event (int which, fd_set *in, fd_set *out, fd_set *except,
101                         struct timeval *timeout)
102 #endif
103 {
104         struct ts_sample sample;
105         int    ret = 0;
106         int fd;
107         fd_set rfds;
108         int e;
109
110         if (!in) {
111                 in = &rfds;
112                 FD_ZERO (in);
113         }
114
115  fd = ts_fd(ts);
116
117         if ((which & IAL_MOUSEEVENT) && fd >= 0) {
118                 FD_SET (fd, in);
119 #ifdef _LITE_VERSION
120                 if (fd > maxfd) maxfd = fd;
121 #endif
122         }
123 #ifdef _LITE_VERSION
124         e = select (maxfd + 1, in, out, except, timeout) ;
```

```
125 #else
126         e = select (FD_SETSIZE, in, out, except, timeout) ;
127 #endif
128
129         if (e > 0) {
130
131             // input events is coming
132             if (fd > 0 && FD_ISSET (fd, in)) {
133                 FD_CLR (fd, in);
134                 ts_event.x=0;
135                 ts_event.y=0;
136
137                 ret = ts_read(ts, &sample, 1);
138                 if (ret < 0) {
139                     perror("ts_read()");
140                     exit(-1);
141                 }
142
143                 ts_event.x = sample.x;
144                 ts_event.y = sample.y;
145                 ts_event.pressure = (sample.pressure > 0 ? 4:0);
146
147                 if (ts_event.pressure > 0 &&
148                     (ts_event.x >= 0 && ts_event.x <= 639) &&
149                     (ts_event.y >= 0 && ts_event.y <= 479)) {
150                     mousex = ts_event.x;
151                     mousey = ts_event.y;
152                 }
153
154 #ifdef _DEBUG
155                 if (ts_event.pressure > 0) {
156                         printf ("mouse down: ts_event.x = %d, ts_event.y = %d,
ts_event.pressure = %d\n",
157                             ts_event.x, ts_event.y, ts_event.pressure);
158                 }
159 #endif
160                 ret |= IAL_MOUSEEVENT;
161
162                 return (ret);
163             }
164
165         }
166         else if (e < 0) {
167             return -1;
```

```
168         }
169
170         return (ret);
171 }
172
173 BOOL Init2410Input (INPUT* input, const char* mdev, const char* mtype)
174 {
175         char *ts_device = NULL;
176
177         if ((ts_device = getenv("TSLIB_TSDEVICE")) != NULL) {
178
179                 // open touch screen event device in blocking mode
180                 ts = ts_open(ts_device, 0);
181         } else {
182 #ifdef USE_INPUT_API
183                 ts = ts_open("/dev/input/event0", 0);
184 #else
185                 ts = ts_open("/dev/touchscreen/ucb1x00", 0);
186 #endif
187         }
188
189         if (!ts) {
190                 perror("ts_open()");
191                 exit(-1);
192         }
193
194         if (ts_config(ts)) {
195                 perror("ts_config()");
196                 exit(-1);
197         }
198
199         input->update_mouse = mouse_update;
200         input->get_mouse_xy = mouse_getxy;
201         input->set_mouse_xy = NULL;
202         input->get_mouse_button = mouse_getbutton;
203         input->set_mouse_range = NULL;
204
205         input->wait_event = wait_event;
206         mousex = 0;
207         mousey = 0;
208         ts_event.x = ts_event.y = ts_event.pressure = 0;
209
210         return TRUE;
211 }
```

```
212
213 void Term2410Input(void)
214 {
215         if (ts)
216                 ts_close(ts);
217 }
218
219 #endif /* _SMDK2410_IAL */
```

2.5 程序简要说明

简单说明一下上述程序。第 32 行包含了头文件 tslib.h，这是必须的，因为要用到 tslib 的编程接口。第 45～51 行的宏定义是用来控制是否输出调试信息的，作者在最初修改此输入引擎的过程中开启了_DEBUG 宏，在完成之后则关闭了该宏，避免输出一大堆用于调试目的的信息。

第 54～59 行的 TS_EVENT 结构是 SMDK2410 输入引擎原来定义的，为了使得代码改动幅度最小化，作者决定将其保留。第 61 行定义的数组 state 用来保存键盘按键的键值，由于作者所用的评估板没有接键盘，因此实际上 state 数组未被使用到。第 62～63 行的 mousex 和 mousey 用于保存触摸屏点击处的坐标。第 65 行定义了一个触摸屏设备指针 ts，该指针的作用就如同采用系统调用进行 IO 时的文件描述符一样。

第 71～74 行的 mouse_update()函数不做任何事情，总是返回 1，表示触摸屏的光标位置总是能够自动更新。实际上，触摸屏上光标的自动更新应该是由触摸屏驱动完成的，故此处不需做任何实质性的动作。

第 76～89 行的 mouse_getxy()函数返回点击事件发生处的坐标，返回的坐标值通过参数指针返回。注意到如果坐标值超出了允许的范围之内，将对其进行“饱和”处理。对于作者所用的评估板，其触摸屏的允许范围是 640x480。

第 91～94 行的 mouse_getbutton()函数返回点击动作的轻重（以压力表示）。

整个输入引擎中最重要的部分是第 96～171 行的 wait_event()函数，在这个函数中，输入引擎将不断地从外部设备获取发生的输入事件。但是，如果采用通常的办法直接用 read()来读取数据显然不行，为什么呢？因为如果 open()打开的文件描述符是阻塞型的，那么当发出 read()调用的时候还没有数据到来 read()系统调用将会被阻塞。这将导致什么问题呢？实际上，这样做导致的后果是严重的，因为，阻塞的系统调用将导致整个程序挂起，直到有输入事件到来，在这段时间之内，程序将无法作出响应。为了解决这个问题，MiniGUI 中的输入引擎部分采用了 select()系统调用来实现这样一个功能：仅当外部事件发生时，才发出 read()调用读取数据。

select()系统调用的原型如下：

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
```

```
int select(int n, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);
```

select()系统调用引入了描述符集合的概念，描述符集合用 fd_set 表示，代表着一组文件描述符。select()系统调用本质上也是阻塞的，但是与 read()和 write()等系统调用的永久阻塞性质不同，select()系统调用的阻塞时间是可以控制的，其最后一个参数 timeout 就是用来指定调用的超时时间。select()系统调用能够同时监视三个描述符集合，这三个描述符集合分别是：可读文件描述符集合 readfds、可写文件描述符集合 writefds、异常描述符集合 exceptfds，当在这三个描述符集合上有相应的事件发生时，比如，在可读文件描述符集合 readfds 中的某个文件描述符上有数据可读时，select()系统调用将返回，返回值为描述符集合上的描述符总数。如果是因为超时时间到达而导致的返回，select()系统调用将返回 0；如果是因为错误而导致的返回，select()系统调用将返回-1 并设置 errno 为相应的错误代码。另外，需要特别说明一下 select()系统调用的第一个参数 n。这个参数应该设置为三个描述符集合上最大的秒数子加 1，嗯，这个参数的确很奇怪，但它就是这样工作的。

为了操纵文件描述符集合 fd_set，大多数系统中都定义了下面这样四个宏定义：

```
FD_CLR(int fd, fd_set *set);
FD_ISSET(int fd, fd_set *set);
FD_SET(int fd, fd_set *set);
FD_ZERO(fd_set *set);
```

FD_ZERO()将清空一个描述符集合，也就是说其中不含任何描述符；FD_SET()将把一个描述符 fd 添加到描述符集合 set 中去；FD_CLR()将从某个描述符集合中删除一个描述符。FD_ISSET()将测试描述符集合 set 中是否包含了某个描述符 fd，这个宏常常在 select()返回之后用来判断是哪个描述符可读或者可写。

现在回到本文的主题。第 110～113 行判断是否指定了可读文件描述符集合，如果没有指定，就指定一个，并将其清空。然后在第 115 行通过 tslib 函数 ts_fd()得到触摸屏设备对应的文件描述符。第 117～122 行，如果发生的是鼠标事件（触摸屏事件可以被看作是鼠标事件）并且触摸屏设备对应的文件描述符有效，就将触摸屏设备对应的文件描述符设置到可读描述符集合中。第 123～127 行，调用 select()。第 132 行，如果 select()调用既没有超时也没有出错，而是正常返回，那么就检查触摸屏设备对应的文件描述符是否在可读描述符集合中，如果是，就从触摸屏设备对应的文件描述符中读取数据（第 137 行）。第 147～152 行，将得到的点击事件的坐标值保存到两个全局变量中。其它一些琐碎的处理这里就不多说了。

第 173～211 行的 Init2410Input()函数是该输入引擎的初始化函数。tslib 默认是通过环境变量 TSLIB_TSDEVICE 来确定触摸屏设备，因此，第 177 行首先检查是否定义了环境变量 TSLIB_TSDEVICE，如果定义了，就用其所指定的设备名。在作者所用的平台上，环境变量 TSLIB_TSDEVICE 定义为/dev/input/event1。打开设备之后，第 194～194 行对设备进行配置，实际上就是决定是否为其加载 tslib 的插件模块。最后，第 199～208 行将前面定义的几个函数赋给输入引擎的相应函数指针，并设置某些全局变量的初始值。

第 213～217 行的 Term2410Input()函数所完成的功能只有一个：关闭触摸屏设备。

到此，就为 MiniGUI 添加了适合于触摸屏设备的输入引擎。由于 tslib 适用于多种触摸屏设备，因此上述代码应该可以在多种触摸屏设备上工作。

2.6 重新交叉编译 MiniGUI

由于修改了 MiniGUI 的源代码，因此还需要重新编译 MiniGUI。因为用到了 tslib 库，所以必须在编译的时候告诉 MiniGUI 到哪里去找到 tslib 相关的头文件和共享库文件。具体做法如下所示：

```
[root@cyc root]# cd /home/cyc/minigui/libminigui-1.3.3
[root@cyc libminigui-1.3.3]# ./configure --enable-lite –enable-debug --prefix=\
> /home/cyc/minigui/libminigui-1.3.3/arm-linux-build/ \
> --host=arm-linux --target=arm-linux --enable-smdk2410ial \
> CFLAGS="-I/home/cyc/minigui/tslib/arm-linux-build/include \
> -L/home/cyc/minigui/tslib/arm-linux-build/lib -lts"
[root@cyc libminigui-1.3.3]# make
[root@cyc libminigui-1.3.3]# make install
```

这里说一下为什么要指定 CFLAGS 标志。其实，通过指定这个标志，告诉编译器应该到哪里去找 tslib 有关的头文件和共享文件，-lts 则告诉链接器最后生成的 MiniGUI 的共享库文件最后要和 ts 库（ts 是 touchscreen 的缩写）链接。

另外这里也开启了–enable-debug 标志，这样做的原因是为了能够对添加的输入引擎代码部分进行调试。

最后，由于是在 SMDK2410 输入引擎的基础上进行修改而得到的适合作者所用平台的输入引擎，因此需要将 MiniGUI.cfg 文件中的 ial_engine 一项修改为：

```
ial_engine=SMDK2410
```

这样，当MiniGUI启动的时候，MiniGUI就会试图启动SMDK2410 输入引擎，但实际上启动的却是被作者修改后的输入引擎。简言之，这里作者干了一件“偷梁换柱”的勾当。虽然这种处理方式不是太好，却是属于比较简单和容易理解的一种实现方式。在作者所用的评估平台上，MiniGUI.cfg中几个重要的项的设置情况如下所示。有关如何修改MiniGUI.cfg的详细说明请参见[2]。

```
[system]
# GAL engine
gal_engine=fbcon

# IAL engine
ial_engine=SMDK2410

mdev=/dev/input/event1
mtype=IMPS2

[fbcon]
```


```
defaultmode=640x480-16bpp
```

3 测试与结论

本文详细说明了如何通过 tslib 来为 MiniGUI 提供 IAL 引擎。实际的测试结果表明，这种做法是有效的。同时，由于 tslib 本身所具有的抽象性，这种方法应该也能够在其它平台上工作。由于条件所限，作者未能在其它平台上进行测试。

4 参考文献

[1] 北京飞漫软件技术有限公司。MiniGUI 编程指南 for MiniGUI Ver 1.3.x。2003 年 10 月

[2] 北京飞漫软件技术有限公司。MiniGUI 用户手册 for MiniGUI Ver 1.3.x。2003 年 10 月

[3] 北京飞漫软件技术有限公司。MiniGUI API Reference Documentation for MiniGUI Ver 1.3.x。2003 年 10 月