BitCode

UART 键盘接口

Arduino 驱动库

使用说明书

索引

索引.....2
综述.....3
驱动库的安装.....4
硬件连接.....5
驱动库的使用.....6
建立实例.....6
初始化 setup().....6
基本应用：单按键.....6
进阶应用：长按键.....7
进阶应用：组合键.....8
函数参考手册.....9
设置函数.....9
setDetectMode() 设置键盘检测模式.....9
setLongpressCount() 设置长按键计数值.....9
setCallback() 设置回调函数.....9
defCombinedKey() 定义组合键.....9
组合键组成定义数组.....10
组合键列表数组.....10
defLongpressKey() 定义长按键.....11
长按键定义数组.....11
长按键列表数组.....11
输入函数.....12
checkChanges() 更新键盘状态.....12
longpressTick() 长按键计数.....12
查询函数.....12
isKeyChanged() 查询是否有新按键.....12
getKeyValue() 获取键值.....13
使用方法示例.....14
例 1. 普通单键.....14
例 2. 长按键.....14
例 3. 长按键+组合键.....15

综述

BC6xxx 系列和 BC759x 系列芯片提供统一的 UART 单线键盘接口，本驱动库可适用于以下芯片：

BC6301 —— 30 键键盘接口

BC6040 —— 40 键键盘接口

BC6561 —— 56 键键盘接口

BC6088 —— 88 键键盘接口

BC7595 —— 48 段 LED(6 位数码管)+48 键键盘接口

BC7591 —— 256 段 LED(32 位数码管)+96 键键盘接口

本驱动库适用于所有的 Arduino 器件，可用于硬件串口和软件串口。

使用本驱动库，除了可以很方便地处理普通的单按键事件，仅需增加几行代码，就可以实现如组合键和长按键这样的复杂键盘功能。

UART 单线键盘接口，每次传送一个 1 字节代表一个键盘事件，使用值 0-0x7F 代表按键键值，最高位作为按键释放标志，即 0-0x7F 表示按键按下，0x80-0xFF 表示按键释放，因此理论上最大的键盘数量为 128 键，实际芯片按键数量最多的是 BC7591 支持 96 键。本驱动库利用未使用的键值空间，让用户可以把组合键和长按键定义为自定义的键值，这样在用户程序中，处理组合键、长按键就变得和处理普通按键一样简单，仅需要根据不同键值进行程序分支，将所有复杂的转换和判断封装在了驱动库内。

键盘接口的使用非常简单，仅需 3 个步骤：

1. 在 loop()中，呼叫 checkChanges()函数。

2. 查询 isKeyChanged()

3. 如果 isKeyChanged()返回结果为 true, 用 getKeyValue()获取键值

有关 UART 键盘接口的详细资料，请查阅 BC6xxx 或者 BC759x 系列芯片的数据手册。

驱动库的安装

驱动库的安装非常简单，从 Arduino IDE 菜单中，选择

“项目(Skech) --> 加载库(Include Library) --> 添加.ZIP 库...(Add .ZIP Library...) ”

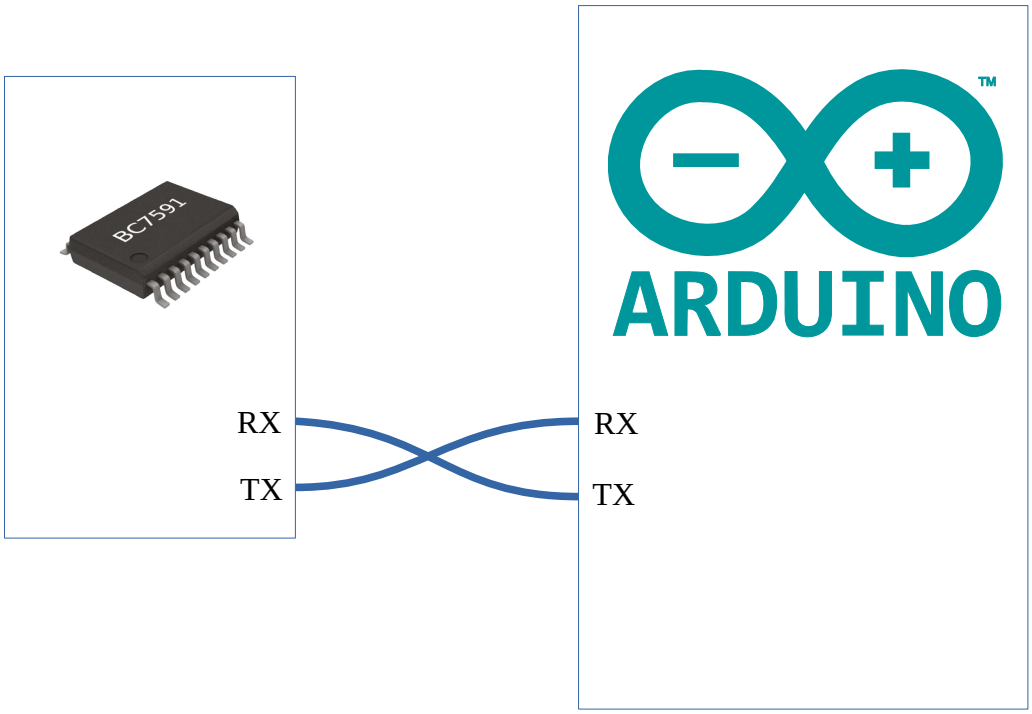
选择 BC_key_scan.zip 文件，即可完成安装。

或者直接将下载的 BC_key_scan.zip 文件解压后，手动将 BC_key_scan 目录拷贝到 Arduino 目录中的 libraries 子目录中，也可达到同样效果。

安装完成后，即可在“项目(Skech) --> 加载库(Include Library)” 菜单中，看到 BC_key_scan 库，表示已经可以使用了。

硬件连接

BC6xxx 和 BC759x 系列芯片均采用串口通讯，通讯使用芯片上 2 个引脚 TX 和 RX，其中 TX 为芯片的键盘接口的输出，BC6xxx 芯片没有 RX 引脚，仅在 BC759x 芯片上有，为 LED 显示部分使用。连接时，芯片的 TX/RX 引脚与 Arduino 的串口 TX/RX 引脚交叉对接：



驱动库的使用

建立实例

BC6xxx 和 BC759x 系列芯片的接口为串口，因此本驱动库的使用，也必须依赖于 Arduino 的串口。串口可以是 Arduino 的硬件串口(如 Serial, Serial1, Serial2 等)，也可以是软件串口(Software Serial)。

对于像 Arduino UNO 这样的型号，只有一个硬件串口，且该串口已经用于和电脑的通讯，因此适于使用软件串口，否则会和 IDE 发生冲突（除非程序下载到硬件使用外部编程器不占用 Arduino 的串口）。

使用前，除了加载驱动库，还必须建立驱动库的实例，如:

```
BcKeyScan          Keypad(Serial1);
```

其中 BcKeyScan 是本驱动的名称，Keypad 是用户给实例起的名字，可以是任意内容，只要符合 Arduino 变量命名规则即可。括号内是所使用的串口，如果是硬件串口，应该是 Serial, 或者是 Serial1, Serial2 等。

如果使用软件串口，需要 2 个额外的步骤，第一步需要加载软件串口库 SoftwareSerial，该库是 Arduino 的标准库之一。第二步需要在建立本驱动的实例之前，现建立一个软件串口的实例:

```
SoftwareSerial     swSerial(2, 3);
```

其中 swSerial 是用户给实例所起的名称，可以是符合 Arduino 变量名命名规则的任何内容。括号内 2, 3 是软件串口所使用的 RX 和 TX 引脚。详情请参阅 Arduino 软件串口库的资料。

建立软件串口的实例后，才可建立本驱动的实例:

```
BcKeyScan          Keypad(swSerial);
```

初始化 setup()

本驱动库本身并不需要做任何初始化即可使用，不过串口必须经过正确的初始化，才能正确连接 BC6xxx 和 BC759x 芯片。串口必须初始化为 9600 波特率。

```
Serial1.begin(9600);     // 使用硬件串口时
```

或者

```
swSerial.begin(9600);   // 使用软件串口时
```

虽然驱动库无需初始化即可使用，不过如果用户需要使用组合键或者长按键这样的功能，会需要一些额外的步骤，也有设置需要在 setup()中完成。详情请参阅后文。

基本应用：单按键

建立了键盘接口实例后，仅需 3 条语句，即可实现基本的键盘功能:

1. 在 loop()中，呼叫 checkChanges() 函数，检查键盘的状态。

2. 呼叫 isKeyChanged() 函数，该函数返回一个 bool 值，为 true 或者是 false。如果返回值为 true，标明有新的按键事件，如果为 false，标明没有需要处理的按键。

3. 如果上一步返回值是 true, 则呼叫 getKeyValue() 函数，该函数返回值即为所按下按键的键值。

知道是哪个键按下了，接下来只需要根据需要转去相应的处理就行了。在默认状态下，驱动库只报告按键按下的事件，而忽略按键的释放。不过 BC6xxx 和 BC759x 芯片都具有区别检测按键的按下和释放的能力，如果用户希望检测按键的释放，比如让按键按下和释放执行不同的动作，可以通过一个函数通知驱动库检测按键的释放:

```
setDetectMode(1);          // 设置驱动库同时检测按键的释放
```

检测模式目前有 2 种，0 为默认模式，不检测按键释放，1 为同时检测按键的按下和释放。按键检测模式可以随时改变，如果在整个应用过程中不需要改变键盘检测模式，可以把这个设置操作放到 setup() 函数中。

进阶应用：长按键

驱动库将长按键赋予和普通单按键一样的键值，这样用户可以像处理普通单按键一样处理长按键。长按键的键值由用户自行定义，定义的键值应该选择不与普通按键相冲突的值。UART 键盘接口协议的键值范围为 0-127, 目前 BC6xxx 和 BC759x 芯片最多的键数为 96 键，芯片天然的键值均从 0 开始顺序排列，用户自定义的键值可以使用所用芯片未使用到的键值。

使用长按键首先需要定义哪些键需要检测长按键以及用户为各个长按键所分配的键值，并通知驱动库。定义长按键通过“长按键定义”和“长按键列表”两个数组完成。第一个数组，定义长按键的自定义键值，数组的数据类型为 unsigned char, 由 2 个元素组成，第一个是需要检测长按键的按键的键值，第二个是用户分配给代表这个按键长按的键值。实际上这个键值定义的数据可以有很多个，需要检测多少个键的长按键，就定义多少个这样的数组。举例如下:

```
const unsigned char Lp1[2] = {5, 120};
const unsigned char Lp2[2] = {20, 121};
```

上面的定义表示，定义 5 号键的长按键的键值为 120； 20 号键的长按键的键值为 121.

定义了长按键的键值后，我们需要另外一个数组，把这些定义通知驱动库。这第二种数组，实际上内容就是上面那些自定义键值数组的列表:

```
const unsigned char* LPList[2] = {Lp1, Lp2};
```

有了这第二个数组，就可以利用函数 defLongpressKey()告知驱动库开始检测这些键的长按键。

```
defLongpressKey(LPList, 2); // 第二个参数 2 为数组 LPList[]中元素的数量
```

这个函数，一般可以放在 setup() 中呼叫。经过这几步操作，现在驱动库已经可以检测长按键，不过有一个问题并没有涉及，就是长按键究竟按多长时间计算。

驱动库本身内并没有计时程序，因此长按键的时间并不是按时间计算的，而是通过一个内部的计数器，通过计数的数值来间接计算时间。用户需要周期性地呼叫函数 longpressTick(), 每呼叫一次，内部的计数器就加一，当计数值达到了预设的门限值，就认为是达到了长按键的时间。用户需要确定呼叫 longpressTick()函数的时间间隔，然后根据需要的长按键时间，计算出计数的上限值。

计数的上限值的默认值是 200, 不过用户可以通过 setLongpressCount()函数改变它，上限值的取值范围是 1-65534。

一般 setLongpressCount()会放在 setup()中，而 longpressTick()函数，则在 loop()中呼叫。

请参阅后文中的实际使用的例子。

进阶应用：组合键

组合键的使用和长按键非常类似，也需要“组合键定义”和“组合键列表”两个数组来告知驱动库如何检测组合键，只不过“组合键定义”数组的格式，和长按键有所不同。

组合键定义数组的数据类型也是 unsigned char, 不过组合键定义数组的长度由组合键的组成键数来决定，长度可变。比如下面两个定义:

```
const unsigned char Cb1[] = {2, 122, 0, 7};
const unsigned char Cb2[] = {3, 123, 11, 15, 20};
```

组合键定义数组的格式，第一个元素为该组合键组成元素的数量，即键数，第二个值为用户自定义的代表该组合键的键值，后面跟随不定长度的键值列表，代表组成该组合键的各键的键值。上面两个例子中，Cb1 由 2 个键组成，自定义键值 122, 由 0 号和 7 号键组成。而 Cb2 由 3 个键组成，自定义键值 123, 由 11, 15,和 20 号键组成。

组合键最多由 7 个键组成。

组合键列表数组格式和长按键列表一样:

```
const unsigned char* CBList[] = {Cb1, Cb2};
```

有了组合键列表数组后，呼叫 defCombinedKey()函数，这个函数和 defLongpressKey()函数用法一致:

defCombinedKey(CBList, 2);

这个函数同样一般放在 setup()中呼叫。

组合键定义后，不需要额外的操作，即可工作。当定义的组合键发生时，比如上例中当 0 号和 7 号键同时按下时，isKeyChanged()将返回 true，而 getKeyValue()将返回 122.

函数参考手册

设置函数

setDetectMode() 设置键盘检测模式

函数使用格式:

```
setDetectMode(Mode);
```

此函数控制函数库的键盘检测模式。输入参数 Mode 为 0 时，本驱动库只报告按键的按下(导通)事件，而忽略按键的释放；Mode 为 1 时，驱动库将同时检测按键的按下和释放，一个键按下和释放的过程中，会两次令 is_key_changed() 查询的结果不为 0. 这里的按键，也包括用户自定义的组合键。

驱动库默认的工作状态为 Mode=0, 不检测按键释放。

setLongpressCount() 设置长按键计数值

函数使用格式:

```
setLongpressCount(CountLimit);
```

驱动库的长按键检测的时间，通过对 longpressTick() 函数被呼叫的次数进行记次来实现。对设置了检测长按键的按键，如果在按键保持该状态的时间内，longpressTick()被呼叫的次数超过了这里设置的 CountLimit 值，则驱动库就会报告一个长按键事件。

长按键计数值的默认值为 2000.

setCallback() 设置回调函数

函数使用格式:

```
set_callback(*pCallbackFunc);
```

此函数的输入参数为一具有 unsigned char 输入参数类型，返回类型为 void 的函数指针。当设置了此回调函数后，每当有新按键产生，都会自动呼叫此函数，而 isKeyChanged() 查询，将不再能查询到新按键事件。回调函数被呼叫时，将以新按键的键值作为参数，用户可以在回调函数内完成按键的处理。传递的键值，也包括用户自定义的组合键和长按键的键值。

回调函数为用户提供了除“ isKeyChanged() 查询 —— get_key_value() 获取键值”方式外的另外一种按键处理方式。回调函数方式在每次 checkChanges() 被呼叫时自动检查是否有新的按键事件，如果有，则会自动转去呼叫回调函数，可以在 serialEvent()中呼叫 checkChanges()，这样按键一发生立刻就自动转去处理。此函数属于进阶用法，一般应用可以不必使用。

defCombinedKey() 定义组合键

函数使用格式:

```
defCombinedKey(pCBKeyList, CBKeyCount);
```

如果需要使用组合键，必须通过此函数将组合键的定义告知驱动库。本驱动库对组合键的工作方式，是由用户定义特定的键盘组合，并赋予一个特殊的键值，然后驱动库会监视键盘上这个特定的组合，当组合出现时，就会像对待普通按键一样，报告一个新按键事件。

组合键的定义，通过 2 个(种)数组完成:

组合键组成定义数组

定义每一个组合键由哪些键组成。数组的数据类型为 unsigned char，内容为以下格式: {count, defined_value, key1, key2, ... keyn} 其中，count 为该组合键所含的键数量，由 2 个键组成，则为 2, 由 3 个键组成，则为 3. 一个组合键，最多由 7 个按键组成。defined_value 是用户自定义的赋给该组合键的特殊键值，当组合键发生时，驱动库将报告一个新按键事件，而键值即为此 defined_value. 为避免和单独按键的键值冲突，此自定义键值应避开键盘芯片已经使用的键值，一般可在 97-126 的范围内选择。key1, key2 ... keyn, 为组成组合键的各个键的原始键值。一个实际定义的例子:

```
const unsigned char cb1[] = {2, 125, 3, 19};
const unsigned char cb2[] = {3, 124, 4, 7, 10};
```

上面定义标明，名为 cb1 的组合键由 2 个键组成，分别为 3 号键和 19 号键，赋予的特殊键值为 125；名为 cb2 的组合键由 3 个键组成，分别为 4 号键，7 号键和 10 号键，赋予的特殊键值为 124.

此数组并不直接使用于 defCombinedKey() 函数的呼叫，但是是下面组合键列表数组的成员，因此是必须的。

组合键列表数组

数组的类型为 unsigned char* , 其成员为组合键组合定义数组的列表。总共有多少个组合键，该数组就有多少个成员。以上面的组合键组成定义为例，有 cb1 和 cb2 两个组合键，组合键列表数组的定义为:

```
const unsigned char* cb_list[] = {cb1, cb2};
```

具有上面 2 个数组的定义后，就具备了呼叫本函数的全部参数条件，仍以上面的数据为例，呼叫本函数的方式为:

```
defCombinedKey( cb_list, 2);
```

输入参数:

`pCBKeyList` ： 指向组合键列表数组的指针

`CBKeyCount` ： 组合键的数量

组合键只应在 `setup()` 中定义一次，不应重复定义。

defLongpressKey() 定义长按键

函数使用格式:

```
defLongpressKey(pLPKeyList, LPKeyCount);
```

定义长按键和定义组合键非常类似。函数的输入参数为两个，分别是:

`pLPKeyList`：指向长按键列表数组的指针

`LPKeyCount`：长按键的数量

传递长按键定义所需的 2 个数组:

长按键定义数组

定义长按键为哪个键，以及代表该键长按的用户自定义键值。数组的类型为 unsigned char，长度固定为 2 个字节，格式为 {key_value, defined_value}. 其中，key_value 为待检测长按键的按键的原始键值，这个键值，也可以是用户定义的组合键的键值，如果设置成组合键的键值，就可以完成对组合键的长按键检测。defined_value 为用户定义的代表该键长按的键值。自定义键值的选择和组合键自定义键值一样，应该选择使用不会与其它键冲突的键值。

每一个需检测长按的键，对应一个长按键定义数组，如果系统需要 2 个长按键，则需要定义 2 个长按键定义数组。如下例:

```
const unsigned char lp1[] = {4, 122};
const unsigned char lp2[] = {124, 121};
```

在上面的定义中，定义了两个长按键，第一个是 4 号键，是键盘上的独立物理按键，自定义键值 122,即该键长按时，驱动库会报告一个键值为 122 的新按键。第二个键待检测的是一个用户自定义的组合键，键值为 124, 长按键的自定义键值为 121. 其含义是如果用户自定义的键值为 124 的组合键长按了一定时间，驱动库会报告一个键值为 121 的新按键。如果沿用上面的例子，就是当物理 4 号，7 号和 10 号按键同时按下保持一定时间，就会产生一个键值为 121 的新按键事件。

长按键列表数组

数组类型为 unsigned char*，其内容为长按键定义数组的指针。数组元素的个数，就是系统中长按键的个数。使用上面的例子，则产生的列表数组为:

```
const unsigned char* lp_list[] = {lp1, lp2};
```

具有了长按键的定义数组后，就可以呼叫本函数通知驱动库所需要检测的长按键:

```
def_longpress_key(lp_list, 2);
```

上面函数的意义为：使用长按键列表 lp_list 中的长按键，总数量为 2 个。

定义长按键时，有一个特殊键值，专门用于代表无按键操作。如果将原始键值定义为 255(0xFF)，则代表对“无按键”的检测。如果最后一次按键释放后经过了设定的长按键时间内没有任何键盘动作，则驱动库就会报告一个键盘事件，键值为此处设定的自定义键值。检测“无按键”的定义数组例子:

```
const unsigned char no_key[] = {255, 255};
const unsigned char* lp_list[] = {lp1, lp2 ... no_key};
```

上面例子中，对“无按键”检测的自定义键值也设为了 255, 开启了长按键检测后，如果最后一次按键被释放后经过设定的长按键时间期间没有任何键盘操作，驱动库就会报告一个键值是 255 的键盘事件。

对“无按键”的检测，并不需要使能按键释放检测模式。

输入函数

checkChanges() 更新键盘状态

函数使用格式:

```
checkChanges();
```

这个函数用来检查是否有新的按键。这个函数被呼叫后，如果满足了新按键的条件，驱动库就会报告一个新键盘事件，isKeyChanged()就会返回 true，或者如果设置成了使用回调函数的方式，在本函数返回前，回调函数就会被呼叫。

longpressTick() 长按键计数

函数使用格式:

```
longpressTick();
```

长按键的检测，依赖于此函数。此函数应被定期呼叫。每次这个函数被呼叫，驱动库内部的计数器就加一，而每次有新的键盘活动，该计数器则被清零。如果该计数器达到了设定的上限(通过 setLongpressCount()函数设置)，就会检查此前最后一个按键是否为待检测的长按键之一，如果是，就会报告一个长按键事件。

如果呼叫此函数的时间间隔固定，则长按键的时间，就是呼叫时间间隔×设定的计数上限。

此函数一般可以 loop()中呼叫，如果用户需要临时停止对长按键的检测，只需要停止呼叫此函数。

查询函数

isKeyChanged() 查询是否有新按键

函数使用格式:

```
isKeyChanged();
```

用户程序通过此函数获得键盘的状态信息。程序返回值为 bool 类型，当返回 false（0）时，表示没有新的按键变化，返回 true（1）时表示有新按键。新按键是指在当前工作模式下可被检测的按键变化，例如如果设置为不检测按键释放，则任何按键释放的事件不会影响此查询的结果。新按键同时包括组合键和长按键事件。

对于组合键，当组合成员中的每一个键按下时，都会如同单独的按键一样报告一个单键的新按键事件，但组合键中的最后一个键按下时，因为已经满足了组合键的条件，故只会报告组合键的新按键事件，而不会再单独产生该个体按键的按键事件。

当有两个键同时按下时，两个按键事件会连续产生，但如果在第一个按键事件产生后程序未能及时用getKeyValue()获取键值，该事件将被第二个键盘事件替代。如果第一个按键是重要按键需要捕捉，建议将其和第二个键设置为组合键。

getKeyValue() 获取键值

函数使用格式:

```
getKeyValue();
```

用户通过此函数获得键盘事件的键值。此函数可以在任何时候呼叫。此函数被呼叫后，如果此前有尚未获取的新按键，即 isKeyChanged()函数返回 true，呼叫后 isKeyChanged()则会恢复为返回 false.

驱动库提供了键值锁存的功能，只要在下一个新按键到来之前，都可以通过本函数读取到最近一次的键值，因此减低了对用户程序实时性的要求。当设置为不检测按键释放时，两次连续按键的时间间隔，通常最小为几百毫秒，用户程序可以有充分的时间处理当前的任务，待空闲时再对键盘做出反应。如果设置为按键按下和释放同时检测，则按键按下和释放都会产生键盘事件，这个时间间隔可能会缩短为几十毫秒。

如果使用回调函数处理按键，因为键值已经作为参数传递给回调函数，则无需再呼叫此函数。

使用方法示例

例 1. 普通单键

下面的 Sketch 使用软件串口，使用数字 I/O 的 11 脚为 RX。(同时设 12 脚为 TX，因为软件串口初始化时需要同时设置 RX 和 TX，但 TX 在本例中没有用到) 软件监测按键的变化，如果有新的按键，则将键值在硬件串口 Serial 上打印出来，可以用串口监视器(Serial Monitor)查看:

```
#include <SoftwareSerial.h>
#include <bc_key_scan.h>

SoftwareSerial   swSerial(11, 12);   // 创建软件串口实例，11 脚为 RX，12 脚为 TX
BcKeyScan        Keypad(swSerial);   // 创建驱动库实例，使用软件串口

void setup() {
    Serial.begin(9600);              // 硬件串口初始化
    swSerial.begin(9600);            // 软件串口初始化
}

void loop() {
    Keypad.checkChanges();                  // 更新键盘状态
    if (Keypad.isKeyChanged() == true) {    // 如果有新按键
          Serial.println(Keypad.getKeyValue());   // 将键值打印在硬件串口
    }
    delay(10);                       // 延时 10ms
}
```

例 2. 长按键

下面的 Sketch 和例 1 基本相同，增加了对长按键的支持。

```
#include <SoftwareSerial.h>
#include <bc_key_scan.h>

SoftwareSerial   swSerial(11, 12);   // 创建软件串口实例，11 脚为 RX，12 脚为 TX
BcKeyScan        Keypad(swSerial);   // 创建驱动库实例，使用软件串口

// 定义长按键
const unsigned char  lp1[2] = { 1, 120 };  // 长按键 1 定义，检测'1'键长按，长按键键值 120
const unsigned char  lp2[2] = { 5, 121 };  // 长按键 2 定义，检测'5'键长按，长按键键值 121
```

```
const unsigned char* LPList[2] = { lp1, lp2 };  // 长按键列表

void setup() {
    Serial.begin(9600);              // 硬件串口初始化
    swSerial.begin(9600);            // 软件串口初始化
    Keypad.defLongpressKey(LPList, 2);  // 定义长按键
    Keypad.setLongpressCount(300);     // 定义长按键时间长度为 3s(10ms*300)
}

void loop() {
    Keypad.checkChanges();                  // 更新键盘状态
    if (Keypad.isKeyChanged() == true) {    // 如果有新按键
          Serial.println(Keypad.getKeyValue());   // 将键值打印在硬件串口
    }
    Keypad.longpressTick();          // 长按键计数
    delay(10);                       // 延时 10ms
}
```

例 3. 长按键+组合键

在例 2 的基础上进一步增加了组合键:

```
#include <SoftwareSerial.h>
#include <bc_key_scan.h>

SoftwareSerial   swSerial(11, 12);   // 创建软件串口实例，11 脚为 RX, 12 脚为 TX
BcKeyScan        Keypad(swSerial);   // 创建驱动库实例，使用软件串口

// 定义长按键
const unsigned char  lp1[2] = { 1, 120 };  // 长按键 1 定义，检测'1'键长按，长按键键值 120
const unsigned char  lp2[2] = { 5, 121 };  // 长按键 2 定义，检测'5'键长按，长按键键值 121
const unsigned char* LPList[2] = { lp1, lp2 };  // 长按键列表

// 定义组合键
const unsigned char  cb1[4] = { 2, 122, 0, 1 };  // 组合键 1,由 0,1 两个键组合，键值 122
const unsigned char  cb2[4] = { 2, 123, 8, 12 }; // 组合键 2,由 8,12 两个键组合，键值 123
const unsigned char* CBList[2] = { cb1, cb2 };   // 组合键列表

void setup() {
    Serial.begin(9600);              // 硬件串口初始化
    swSerial.begin(9600);            // 软件串口初始化
```

```
    Keypad.defLongpressKey(LPList, 2);  // 定义长按键
    Keypad.setLongpressCount(300);      // 定义长按键时间长度为 3s(10ms*300)
    Keypad.defCombinedKey(CBList, 2);  // 定义组合键
}

void loop() {
    Keypad.checkChanges();                  // 更新键盘状态
    if (Keypad.isKeyChanged() == true) {    // 如果有新按键
          Serial.println(Keypad.getKeyValue());   // 将键值打印在硬件串口
    }
    Keypad.longpressTick();           // 长按键计数
    delay(10);                        // 延时 10ms
}
```