# 嵌入式Linux应用开发文档框架

## GPIO编程

### 9.1 GPIO编程基础介绍

GPIO(General-Purpose IO Ports)，即通用IO接口。GPIO的使用较为简单，主要分为输入和输出两种功能。GPIO主要用于实现一些简单设备的控制。在作为输入型GPIO的情况下，我们可以将该IO连接外部按键或者传感器，用于检测外部状态。当作为输出时，我们可以通过输出高低电平来控制外部设备的运转。

由于GPIO的功能多种多样，我们需要首先将引脚设置为GPIO。设置为GPIO之后，我们需要设置GPIO的方向。当设置为输出时，我们可以控制输出高电平或者低电平。当设置为输入时，我们可以读取GPIO的电平来判断外部输入电平的高低。

### 9.2 GPIO编程软件接口

GPIO编程有多种实现方式，在这里，我们通过sysfs方式来实现GPIO的控制实现。

如果要通过sysfs方式控制gpio，首先需要底层内核的支持。为了实现内核对sysfs gpio的支持，我们需要在内核中进行设置。在编译内核的时候，加入 Device Drivers-> GPIO Support ->/sys/class/gpio/… (sysfs interface)。作为GPIO的引脚，不允许在内核中被用作其他用途。

在系统正常运行之后，我们可以在/sys/class/gpio下看到sysfs控制相关的接口。有三种类型的接口， 分别是控制接口，GPIO信号和GPIO控制器三种接口。这部分的具体介绍可参考《kernel\Documentation\gpio\sysfs.txt》。

#### 9.2.1 控制接口

控制接口用于实现在用户空间对GPIO的控制，主要包括/sys/class/gpio/export和/sys/class/gpio/unexport两个接口。这这两个控制接口都是只写的，/sys/class/gpio/export实现将GPIO控制从内核空间导出到用户空间，/sys/class/gpio/unexport用于实现取消GPIO控制从内核空间到用户空间的导出。

下面以引脚编号为19的GPIO为例进行说明，在/sys/class/gpio/目录下，我们执行"echo 19 > export"之后，将会产生一个”gpio19”节点来控制引脚编号为19的GPIO。我们执行"echo 19 > unexport"之后，将会删除之前通过export产生的”gpio19”节点。为了使用gpio，我们需要首先使用/sys/class/gpio/export导出gpio引脚编号。完成使用之后，通过/sys/class/gpio/unexport删除掉之前导出的gpio引脚。

#### 9.2.2 GPIO信号

GPIO信号，即为GPIO本身，其路径为/sys/class/gpio/gpioN/,拥有多个属性。通过对这些属性进行控制，就可以实现对GPIO的控制

* “direction”属性，读取的值为”in”或者”out”。通过对该属性写入”in”或者”out”可以设置该GPIO为输入或者输出。如果直接写入”out”，则会使GPIO直接输出低电平。也可以通过写入”low”或者”high”来直接设置输出低电平或者高电平。
* ”value”属性，用于读取输入电平或者控制输出电平。如果GPIO为输出，则对value写入0为输出低电平，写入非0为输出高电平。如果设置为输入的话，则读到0表示输入为低电平，1为高电平。
* ”edge”属性，用于设置触发电平，只有在GPIO可以设置为中断输入引脚时才会出现该属性

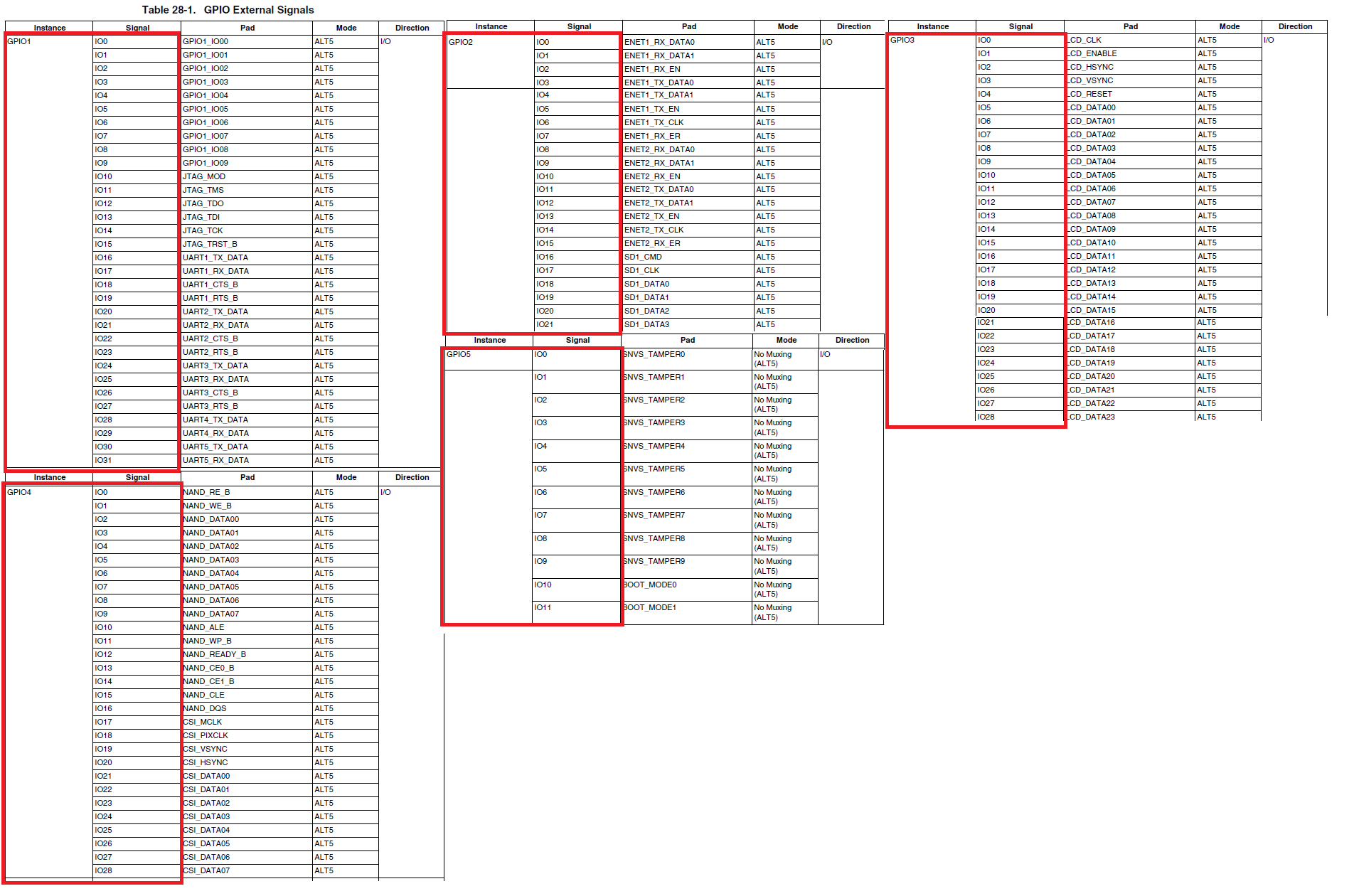
#### 9.2.2 GPIO控制器

GPIO控制器，用于表示GPIO 控制实现的初始GPIO，其路径为/sys/class/gpio/gpiochipN/。比如/sys/class/gpio/gpiochip42/ 则表示实现GPIO控制器的初始化编号为42。GPIO控制器的属性为只读属性，包括base、label和ngpio等多个。

* ”base”属性，和gpiochipN的N代表的含义相同，表示被该组GPIO控制器实现的第一个GPIO.
* ” ngpio”属性，用于表示该控制器支持多少个GPIO，支持的GPIO编号为从N到N+ngpio-1
* ” label”属性，用于判断控制器，并不总是唯一的

### 9.3 IMX6ULL开发板GPIO编号的确定

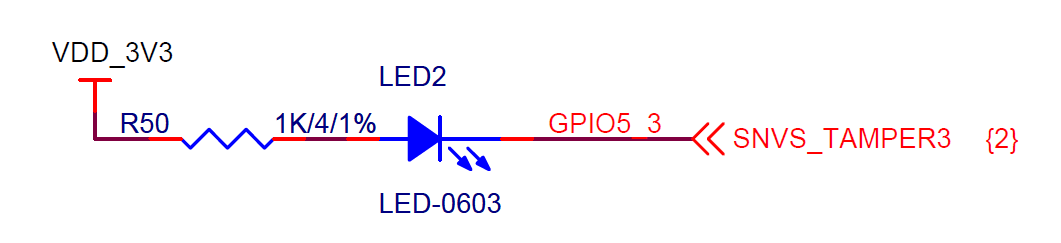
每个芯片可以有N组GPIO，每组GPIO最多有32个GPIO，即最多有N\*32个GPIO。但是在实际设计中，每组的GPIO数量各有不同。在IMX6ULL中，实际每组拥有的GPIO数量如下图所示，具体详见《IMX6ULLRM.pdf》手册1347页。



从上图可以看到，在IMX6ULL中，共有5组GPIO，起始GPIO组为GPIO1。因此在实际GPIO编号计算中，第一组GPIO1对应的编号为0~31。以此类推，IMX6ULL的GPION\_X（N=1~5，X=0~31对应的编号实际为（N-1）\*32+X。接下来，我们以板载的LED和按键各自对应的GPIO为例来说明如何在实际应用中计算GPIO编号。

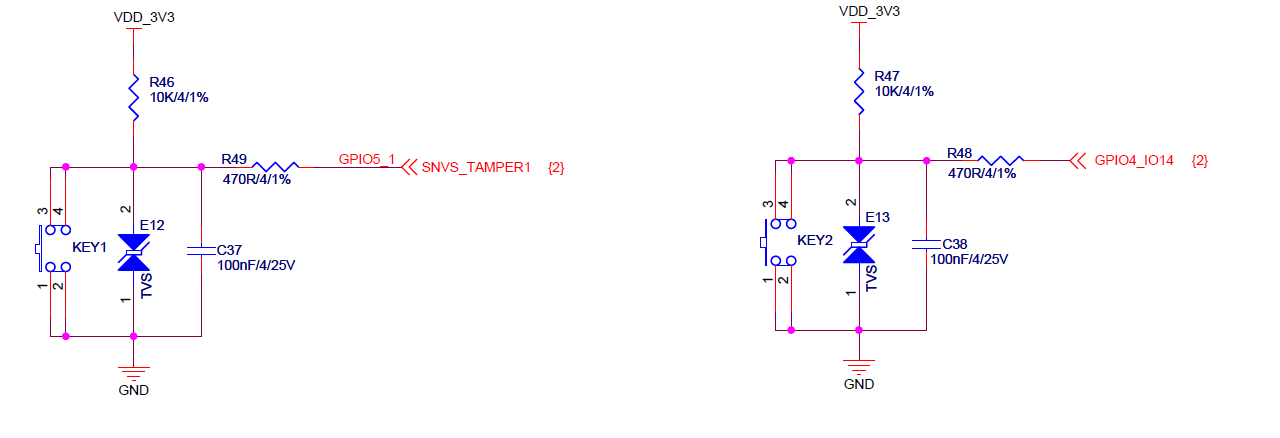
#### 9.3.1 LED的GPIO编号计算

从原理图中找到对应LED的设计，具体的连接如下图所示。从图中我们可以看到，LED连接到的GPIO为GPIO5\_3，其对应的GPIO编号实际为（5-1）\*32+3 = 131。因此，我们如果要在sys\_gpio中操作LED，我们就需要将编号131的GPIO进行导出。



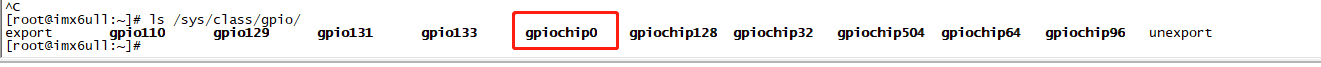
#### 9.3.2 按键的GPIO编号计算

从原理图中找到对应按键的设计，底板有2个按键，具体的连接如下图所示。从图中我们可以看到，两个按键连接到的GPIO分别为GPIO5\_1和GPIO4\_14，第一个按键KEY1对应的GPIO编号为（5-1）\*32+1 = 129，第二个按键KEY2对应的GPIO编号为（4-1）\*32+14=110。因此，我们如果要在sys\_gpio中读取按键KEY1和KEY2的值，，我们就需要将编号129和110的GPIO进行导出。



#### 9.3.3 特殊情况下的GPIO编号计算

在有些情况下，起始的gpiochipN不是gpiochip0。这个时候 ，我们就需要在原有的GPIO编号基础上加上起始gpiochipN值进行计算。下图所示的为其实gpiochip为gpiochip0的情况。



### 9.4 实际编程操作

在实际操作中，我们使用LED和按键实现了GPIO输出和输入的实验，相关的实验过程和相关代码如下。

#### 9.4.1 导出GPIO口

为了导出GPIO口，我们需要向/sys/class/gpio/export写入需要导出得引脚编号。在使用之后，我们也可以使用/sys/class/gpio/unexport取消导出引脚编号。

导出引脚编号的实现代码如下所示，具体详见《sysfs\_gpio\_1\_export\_gpio sysfs\_gpio\_export.c》的sysfs\_gpio\_export()函数。

|  |
| --- |
| 32 int sysfs\_gpio\_export(unsigned int gpio)  33 {  34 int fd, len;  35 char buf[MAX\_BUF];  36 // /sys/class/gpio/export  37 fd = open( "/sys/class/gpio/export", O\_WRONLY);//打开文件  38 if (fd < 0) {  39 perror("gpio/export");  40 return fd;  41 }  42  43 len = snprintf(buf, sizeof(buf), "%d", gpio);//从数字变换为字符串，即1 变为”1“  44 write(fd, buf, len);//将需要导出的GPIO引脚编号进行写入  45 close(fd);//关闭文件  46  47 return 0;  48 } |

取消导出引脚编号的实现代码如下所示，具体详见《sysfs\_gpio\_export.c》的sysfs\_gpio\_unexport()函数。

|  |
| --- |
| 59 int sysfs\_gpio\_unexport(unsigned int gpio)  60 {  61 int fd, len;  62 char buf[MAX\_BUF];  63 // /sys/class/gpio/unexport  64 fd = open("/sys/class/gpio/unexport", O\_WRONLY);//打开文件  65 if (fd < 0) {  66 perror("gpio/export");  67 return fd;  68 }  69  70 len = snprintf(buf, sizeof(buf), "%d", gpio);//从数字变换为字符串，即1 变为”1“  71 write(fd, buf, len);//将需要取消导出的GPIO引脚编号进行写入  72 close(fd);//关闭文件  73 return 0;  74 } |

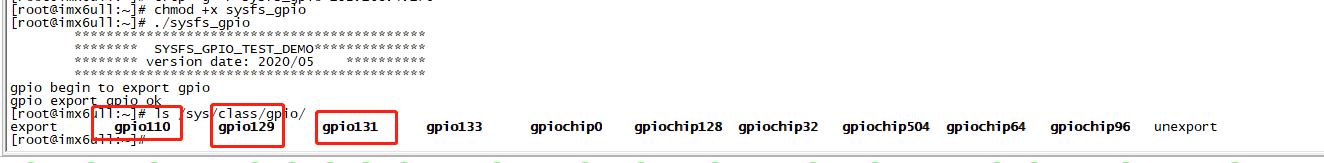
在实现导出和取消导出引脚编号的函数之后，我们来实现具体的引脚编号的导出。LED和按键各自对应的引脚编号如下所示

|  |
| --- |
| 11 #define GPIO4\_14 110  12 #define GPIO5\_1 129  13 #define GPIO5\_3 131  14  15 #define GPIO\_KEY1 GPIO4\_14  16 #define GPIO\_KEY2 GPIO5\_1  17 #define GPIO\_LED GPIO5\_3 |

在确定了各自对应的引脚编号，我们就可以进行导出了。具体实现代码在程序文件《sysfs\_gpio\_1\_export\_gpio/sysfs\_gpio\_export.c》中main函数，下为对应代码部分，我们将LED和按键对应的引脚都进行了导出。

|  |
| --- |
| 183 int main(int argc, char \*\*argv) {  184 unsigned int i;  185 unsigned int value1,value2;  186  187 printf("\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  188 printf("\t\*\*\*\*\*\*\*\* SYSFS\_GPIO\_TEST\_DEMO\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  189 printf("\t\*\*\*\*\*\*\*\* version date: 2020/05 \*\*\*\*\*\*\*\*\*\*\n");  190 printf("\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  191  192 printf("gpio begin to export gpio\r\n");  193 sysfs\_gpio\_export(GPIO\_KEY1);//export gpio key1  194 sysfs\_gpio\_export(GPIO\_KEY2);//export gpio key2  195 sysfs\_gpio\_export(GPIO\_LED);//export gpio led  196 printf("gpio export gpio ok\r\n");  197  198  199 return 0;  200 } |

在将代码编译之后，我们将代码在板卡上进行运行。代码运行之后的的结果如下图所示，可以看到成功的将GPIO110、GPIO129和GPIO131进行了导出。



#### 9.4.2 设置GPIO方向

为了实现导出的引脚的方向设置，我们需要对/sys/class/gpio/gpioN/direction写入不同的值。写入“in”则表示设置为输入，写入“out”则表示设置为输出。设置引脚编号的的实现代码如下所示，具体详见《sysfs\_gpio\_2\_export\_gpio sysfs\_gpio\_export.c》的sysfs\_gpio\_set\_dir ()函数。

|  |
| --- |
| 86 int sysfs\_gpio\_set\_dir(unsigned int gpio, unsigned int out\_flag)  87 {  88 int fd, len;  89 char buf[MAX\_BUF];  90 // /sys/class/gpio/gpioN/direction  91 len = snprintf(buf, sizeof(buf), SYSFS\_GPIO\_DIR "/gpio%d/direction", gpio);  92  93 fd = open(buf, O\_WRONLY);//打开文件  94 if (fd < 0) {  95 perror(buf);  96 return fd;  97 }  98  99 if (out\_flag)//为1，则写入“out"，即设置为输出  100 write(fd, "out", 4);  101 else//为0，则写入“in"，即设置为输入  102 write(fd, "in", 3);  103  104 close(fd);//关闭文件  105 return 0;  106 } |

在实现引脚方向的设置函数之后，我们分别针对按键和LED设置各自不同的方向。将按键设置为输入“IN”,将LED设置为输出“out”，对应的代码如下图所示。相关的代码在程序文件《sysfs\_gpio\_2\_export\_gpio/sysfs\_gpio\_export.c》中main函数，下为对应代码部分。

|  |
| --- |
| 183 int main(int argc, char \*\*argv) {  184 unsigned int i;  185 unsigned int value1,value2;  186  187 printf("\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  188 printf("\t\*\*\*\*\*\*\*\* SYSFS\_GPIO\_TEST\_DEMO\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  189 printf("\t\*\*\*\*\*\*\*\* version date: 2020/05 \*\*\*\*\*\*\*\*\*\*\n");  190 printf("\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  191  192 printf("begin to export gpio and direction\r\n");  193 sysfs\_gpio\_export(GPIO\_KEY1);//export gpio key1  194 sysfs\_gpio\_export(GPIO\_KEY2);//export gpio key2  195 sysfs\_gpio\_export(GPIO\_LED);//export gpio led  196  197 sysfs\_gpio\_set\_dir(GPIO\_KEY1, 0);//set as input  198 sysfs\_gpio\_set\_dir(GPIO\_KEY2, 0);//set as input  199 sysfs\_gpio\_set\_dir(GPIO\_LED, 1);//set as output  200 printf(" export gpio and direction ok\r\n");  201  202  203  204 return 0;  205 } |

在将代码编译之后，我们将代码在板卡上进行运行。代码运行之后的的结果如下图所示，我们可以看到按键GPIO110和GPIO129的方向设置成了输入，LED2的GPIO131设置成了输入。



#### 9.4.3 GPIO输出实验-LED输出控制

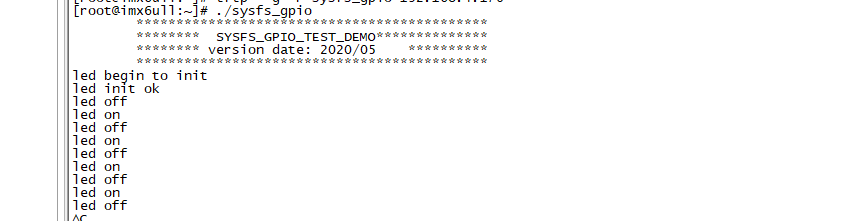
为了设置引脚的输出电平高低，我们需要对/sys/class/gpio/gpioN/value写入不同的值。写入‘1’则表示输出高电平，写入‘0’则表示输出低电平。设置引脚输出高低电平的的实现代码如下所示，具体详见《sysfs\_gpio\_3\_export\_gpio sysfs\_gpio\_export.c》的sysfs\_gpio\_set\_value ()函数。

|  |
| --- |
| 119 int sysfs\_gpio\_set\_value(unsigned int gpio, unsigned int value)  120 {  121 int fd, len;  122 char buf[MAX\_BUF];  123 // /sys/class/gpio/gpioN/value  124 len = snprintf(buf, sizeof(buf), SYSFS\_GPIO\_DIR "/gpio%d/value", gpio);  125  126 fd = open(buf, O\_WRONLY);//打开文件  127 if (fd < 0) {  128 perror(buf);  129 return fd;  130 }  131  132 if (value)//为1，则写入“1"，即设置为输出高电平  133 write(fd, "1", 2);  134 else//为0，则写入“0"，即设置为输出低电平  135 write(fd, "0", 2);  136  137 close(fd);//关闭文件  138 return 0;  139 } |

在实现引脚输出电平的控制函数之后，我们来实现LED的控制。我们通过将“1”或“0”写入value来控制GPIO输出高电平或者低电平，具体相关的代码在程序文件《sysfs\_gpio\_3\_export\_gpio/sysfs\_gpio\_export.c》中main函数，下为对应代码部分。

|  |
| --- |
| 183 int main(int argc, char \*\*argv) {  184 unsigned int i;  185 unsigned int value1,value2;  186  187 printf("\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  188 printf("\t\*\*\*\*\*\*\*\* SYSFS\_GPIO\_TEST\_DEMO\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  189 printf("\t\*\*\*\*\*\*\*\* version date: 2020/05 \*\*\*\*\*\*\*\*\*\*\n");  190 printf("\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  191  192 printf("led begin to init\r\n");  193 sysfs\_gpio\_export(GPIO\_LED);//export gpio led  194  195 sysfs\_gpio\_set\_dir(GPIO\_LED, 1);//set as output  196 printf("led init ok\r\n");  197  198  199 /\* Confirm INIT\_B Pin as High \*/  200 while(1)  201 {  202  203  204 sysfs\_gpio\_set\_value(GPIO\_LED, 1);//output high  205 printf("led off\r\n");  206 usleep(500000); //delay  207 sysfs\_gpio\_set\_value(GPIO\_LED, 0);//output low  208 printf("led on\r\n");  209 usleep(500000);//delay  210 }  211  212 sysfs\_gpio\_unexport(GPIO\_LED);//unexport gpio led  213  214 return 0;  215 } |

在将代码编译之后，我们将代码在板卡上进行运行。代码运行之后的的结果如下图所示， 可以看到规律性的打印LED控制信息（实物可以看到LED灯闪烁）。



#### 9.4.4 GPIO输入试验-按键值读取

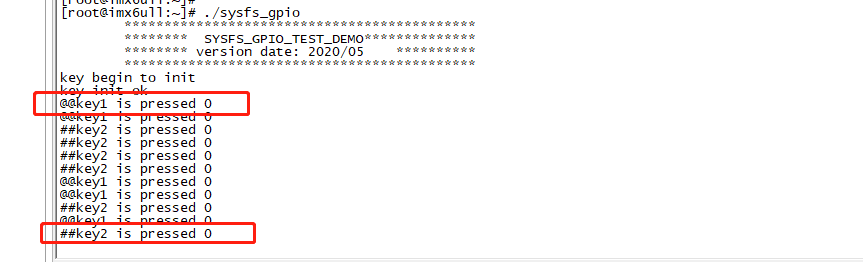
为了读取引脚输入的电平高低，我们需要读取/sys/class/gpio/gpioN/value的值。读到的是‘1’则表输入为高电平，读到的是‘0’则表示输入为低电平。读取引脚输入电平的、的的实现代码如下所示，具体详见《sysfs\_gpio\_4\_export\_gpio sysfs\_gpio\_export.c》的sysfs\_gpio\_get\_value ()函数。

|  |
| --- |
| 152 int sysfs\_gpio\_get\_value(unsigned int gpio, unsigned int \*value)  153 {  154 int fd, len;  155 char buf[MAX\_BUF];  156 char ch;  157 // /sys/class/gpio/gpioN/value  158 len = snprintf(buf, sizeof(buf), SYSFS\_GPIO\_DIR "/gpio%d/value", gpio);  159  160 fd = open(buf, O\_RDONLY);//打开文件  161 if (fd < 0) {  162 perror("gpio/get-value");  163 return fd;  164 }  165  166 read(fd, &ch, 1);//读取外部输入电平  167  168 if (ch != '0') {//为'1'，则设置为1，即输入为高电平  169 \*value = 1;  170 } else {//为'0'，则设置为0，即输入为低电平  171 \*value = 0;  172 }  173  174 close(fd);//关闭文件  175 return 0;  176 } |

在实现引脚电平读取函数之后，我们来实现外部按键值得读取，我们通过读取value的值来读取按键值，具体相关的代码在程序文件《sysfs\_gpio\_4\_export\_gpio/sysfs\_gpio\_export.c》中main函数，下为对应代码部分。

|  |
| --- |
| 183 int main(int argc, char \*\*argv) {  184 unsigned int i;  185 unsigned int value1,value2;  186  187 printf("\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  188 printf("\t\*\*\*\*\*\*\*\* SYSFS\_GPIO\_TEST\_DEMO\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  189 printf("\t\*\*\*\*\*\*\*\* version date: 2020/05 \*\*\*\*\*\*\*\*\*\*\n");  190 printf("\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  191  192 printf("key begin to init\r\n");  193 sysfs\_gpio\_export(GPIO\_KEY1);//export gpio key1  194 sysfs\_gpio\_export(GPIO\_KEY2);//export gpio key2  195  196 sysfs\_gpio\_set\_dir(GPIO\_KEY1, 0);//set as input  197 sysfs\_gpio\_set\_dir(GPIO\_KEY2, 0);//set as input  198  199 printf("key init ok\r\n");  200  201  202 /\* Confirm INIT\_B Pin as High \*/  203 while(1)  204 {  205  206 sysfs\_gpio\_get\_value(GPIO\_KEY1, &value1); //read key1 value  207 //printf("@@key1 value 1is %d \n\r",value1);  208 if(value1==0)//key1 pressed  209 {  210 printf("@@key1 is pressed 0\n\r");  211 }  212 sysfs\_gpio\_get\_value(GPIO\_KEY2, &value2);//read key2 value  213 //printf("##key2 value 1is %d \n\r",value2);  214 if(value2==0)//key2 pressed  215 {  216 printf("##key2 is pressed 0\n\r");  217 }  218 usleep(100000);//delay  219  220 }  221  222 sysfs\_gpio\_unexport(GPIO\_KEY1);//unexport gpio key1  223 sysfs\_gpio\_unexport(GPIO\_KEY2);//unexport gpio key2  224  225  226 return 0;  227 } |

在将代码编译之后，我们将代码在板卡上进行运行。代码运行之后的的结果如下图所示，我们可以看到在按键KEY1和KEY2按下之后打印的值各有不同。



#### 9.4.5 LED和按键控制实验

在前几个实验中，我们分别实现了LED和按键各自得控制。在这个实验中，我们将前几个实验进行整合，控制LED得闪烁，并读取按键得值。当按键按下时，打印相关信息。具体相关的代码在程序文件《sysfs\_gpio\_5\_export\_gpio/sysfs\_gpio\_export.c》中main函数，下为对应代码部分

|  |
| --- |
| 183 int main(int argc, char \*\*argv) {  184 unsigned int i;  185 unsigned int value1,value2;  186  187 printf("\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  188 printf("\t\*\*\*\*\*\*\*\* SYSFS\_GPIO\_TEST\_DEMO\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  189 printf("\t\*\*\*\*\*\*\*\* version date: 2020/05 \*\*\*\*\*\*\*\*\*\*\n");  190 printf("\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  191  192 printf("led&key begin to init\r\n");  193 sysfs\_gpio\_export(GPIO\_KEY1);//export gpio key1  194 sysfs\_gpio\_export(GPIO\_KEY2);//export gpio key2  195 sysfs\_gpio\_export(GPIO\_LED);//export gpio led  196 sysfs\_gpio\_set\_dir(GPIO\_KEY1, 0);//set as input  197 sysfs\_gpio\_set\_dir(GPIO\_KEY2, 0);//set as input  198 sysfs\_gpio\_set\_dir(GPIO\_LED, 1);//set as output  199 printf("led&key init ok\r\n");  200  201  202 /\* Confirm INIT\_B Pin as High \*/  203 while(1)  204 {  205  206 sysfs\_gpio\_get\_value(GPIO\_KEY1, &value1); //read key1 value  207 //printf("@@key1 value 1is %d \n\r",value1);  208 if(value1==0)//key1 pressed  209 {  210 printf("@@key1 is pressed 0\n\r");  211 }  212 sysfs\_gpio\_get\_value(GPIO\_KEY2, &value2);//read key2 value  213 //printf("##key2 value 1is %d \n\r",value2);  214 if(value2==0)//key2 pressed  215 {  216 printf("##key2 is pressed 0\n\r");  217 }  218 //led flash  219 sysfs\_gpio\_set\_value(GPIO\_LED, 1);  220 printf("LED OFF\n\r");  221 usleep(500000);  222 sysfs\_gpio\_set\_value(GPIO\_LED, 0);  223 printf("LED ON\n\r");  224 usleep(500000);  225 }  226  227 sysfs\_gpio\_unexport(GPIO\_KEY1);//unexport gpio key1  228 sysfs\_gpio\_unexport(GPIO\_KEY2);//unexport gpio key2  229 sysfs\_gpio\_unexport(GPIO\_LED);//unexport gpio led  230  231 return 0;  232 } |

在将代码编译之后，我们将代码在板卡上进行运行。代码运行之后的的结果如下图所示，可以看到LED闪烁，按键KEY1和KEY2按下之后打印的值各有不同（因为LED的闪烁导致按键需要经过一次LED闪烁之后才能读取，因此按键必须一直按着才能读取到值的变化）。

