USB HID 键盘

Application Note for 32-bit NuMicro® Family

Document Information

Abstract	此应用文档是在说明如何让开发工程师能够快速导入一般的 USB HID 键盘的入门应用，以节省在键盘扫描程序开发与测试的时间，进而让开发工程师有更多的时间与精力在开发其他键盘上相关的功能或特性。
Apply to	NuMicro® M452/M453 series.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

Table of Contents

1 大纲 5

1.1 M453 系列的主要特性 5

1.2 USB HID 键盘应用方块图 6

2 M453 USB HID 键盘功能管脚规划 7

3 USB HID 键盘软件说明 9

3.1 主程序 9

3.2 系统初始化 11

3.3 设定 UART 13

3.4 设定 LED GPIO 13

3.5 设定 Key GPIO 14

3.6 设定 USB 17

3.7 设定 Timer 20

3.8 键扫描 22

3.9 按键中断 27

3.10 按键处理 29

3.11 更新键盘数据 36

3.12 HID 类别的Request 37

3.13 更新键盘 LED 指示灯 39

3.14 USB HID 键盘描述符 44

4 修订历史 52

List of Figures

Figure 1-1 M453 USB HID 键盘应用方块图 6
Figure 2-1 键盘矩阵管脚与字符对应图 8
Figure 2-2 LED and UART 管脚对应图 8
Figure 3-1主程序流程图 9
Figure 3-2 键盘矩阵输入输出管脚示意图 15
Figure 3-3 USB Device 端点规划图 17
Figure 3-4 各端点 SRAM 数据缓存区规划图 18
Figure 3-5 键扫描输出信号与输入中断时序图 23
Figure 3-6 描述符架构图 45

List of Tables

Table 2-1 管脚功能与 GPIO 模式..... 7

1 大纲

此应用说明文档，是以 M453 系列为主控制器来开发 USB HID 键盘的应用，其中使用 16 根 GPIO 管脚当作脉冲扫描的输出，以及 8 根 GPIO 管脚当作按键事件的输入，来扫描一个 16x8 的键盘矩阵，可以快速检测到使用者按下哪个键，并实时回报给 PC USB Host，来当作一般键盘输入的应用参考。

并且利用 3 根 GPIO 管脚来当作 Caps Lock，Scroll Lock 和 Number Lock On/Off 状态 LED 的显示控制。

1.1 M453 系列的主要特性

NuMicro® M453 系列 32 位微控制器是内嵌 ARM® Cortex®-M4F 内核，适用于需要丰富的通讯接口和快速计算能力的工业控制和应用领域。

NuMicro® M453 系列内置 Cortex®-M4F 内核，并扩展了 DSP 功能和浮点运算单元，运行最高频率达 72 MHz，内置 256K/128K 字节flash 和 32K 字节SRAM。带有丰富的外设，如 USB OTG/Device，定时器，看门狗定时器，RTC，PDMA，EBI，UART，Smart Card，SPI，I^2S，I^2C，CAN，PWM Timer，GPIO，12-位 ADC，12-位 DAC，模拟比较器，温度传感器，低压复位和掉电检测功能。

1.2 USB HID 键盘应用方块图

Figure 1-1 显示 M453 USB HID 键盘方块图以及在此应用所使用到的外设。

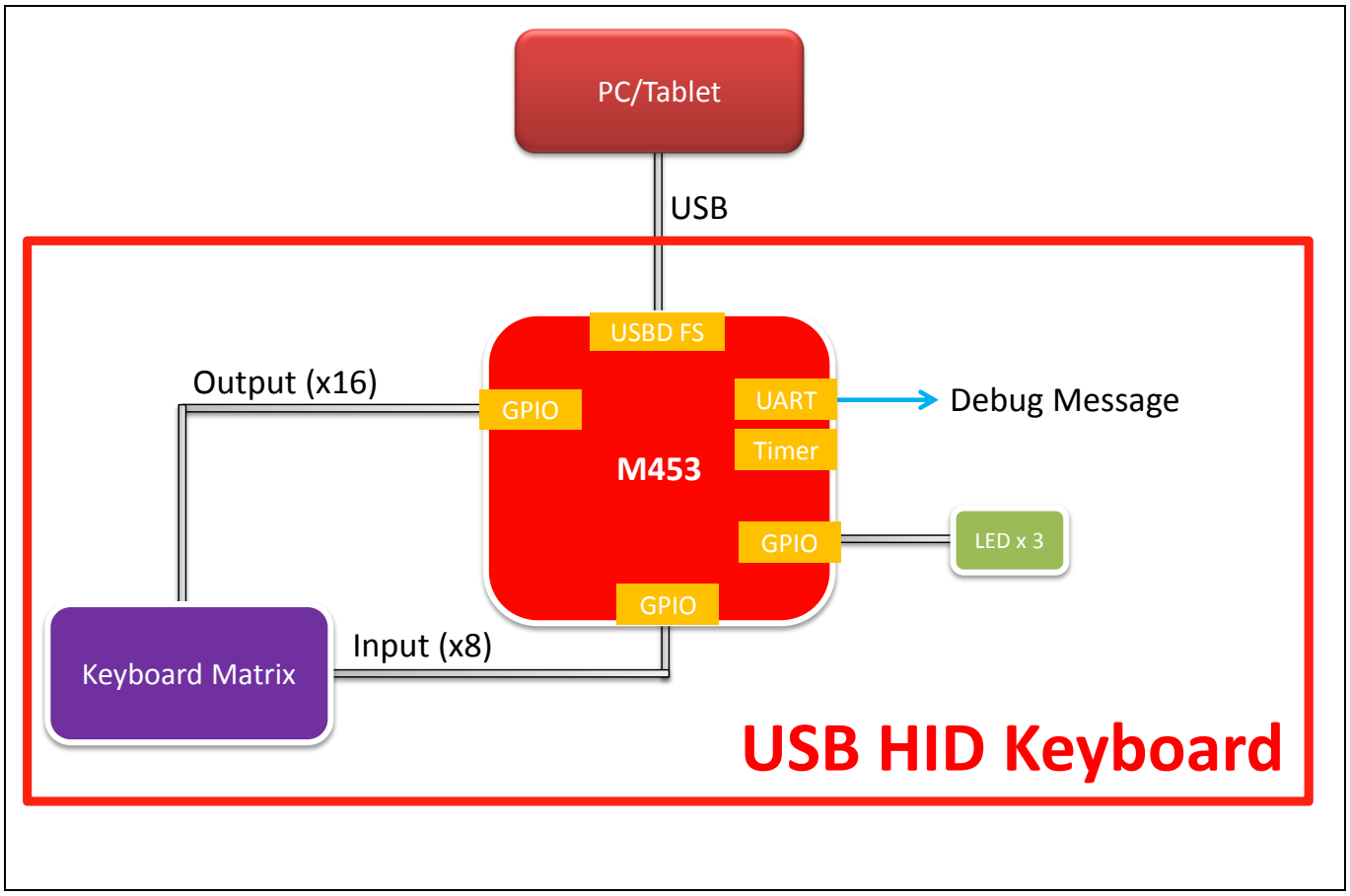


Figure 1-1 M453 USB HID 键盘应用方块图

2 M453 USB HID 键盘功能管脚规划

此 USB HID 键盘应用是采用新唐 M453 系列的微控制器。Table 2-1 说明了 GPIO 管脚功能与 GPIO 模式的设定。

GPIO 名称	功能	IO 模式	GPIO 名称	功能	IO 模式
PD.0	键盘矩阵输入 0	Quasi (In)	PA.13	键盘矩阵输出 7	Quasi (Out)
PD.1	键盘矩阵输入 1	Quasi (In)	PA.14	键盘矩阵输出 8	Quasi (Out)
PD.2	键盘矩阵输入 2	Quasi (In)	PA.15	键盘矩阵输出 9	Quasi (Out)
PD.3	键盘矩阵输入 3	Quasi (In)	PC.8	键盘矩阵输出 10	Quasi (Out)
PD.4	键盘矩阵输入 4	Quasi (In)	PC.9	键盘矩阵输出 11	Quasi (Out)
PD.5	键盘矩阵输入 5	Quasi (In)	PC.10	键盘矩阵输出 12	Quasi (Out)
PD.6	键盘矩阵输入 6	Quasi (In)	PC.11	键盘矩阵输出 13	Quasi (Out)
PD.7	键盘矩阵输入 7	Quasi (In)	PC.12	键盘矩阵输出 14	Quasi (Out)
PB.9	键盘矩阵输出 0	Quasi (Out)	PC.13	键盘矩阵输出 15	Quasi (Out)
PB.10	键盘矩阵输出 1	Quasi (Out)	PB.4	Caps Lock LED	Output
PC.0	键盘矩阵输出 2	Quasi (Out)	PB.6	Num Lock LED	Output
PC.1	键盘矩阵输出 3	Quasi (Out)	PB.7	Scr Lock LED	Output
PC.2	键盘矩阵输出 4	Quasi (Out)	PA.2	UART 0 TXD	Output
PC.3	键盘矩阵输出 5	Quasi (Out)	PA.3	UART 0 RXD	Input
PC.4	键盘矩阵输出 6	Quasi (Out)			

Table 2-1 管脚功能与 GPIO 模式

键盘矩阵是利用 16 根 GPIO 管脚当作输出，以及 8 根 GPIO 管脚当作输入，来扫描一个 16x8 的矩阵式键盘，Figure 2-1 显示键盘矩阵使用的管脚与字符的对应图。

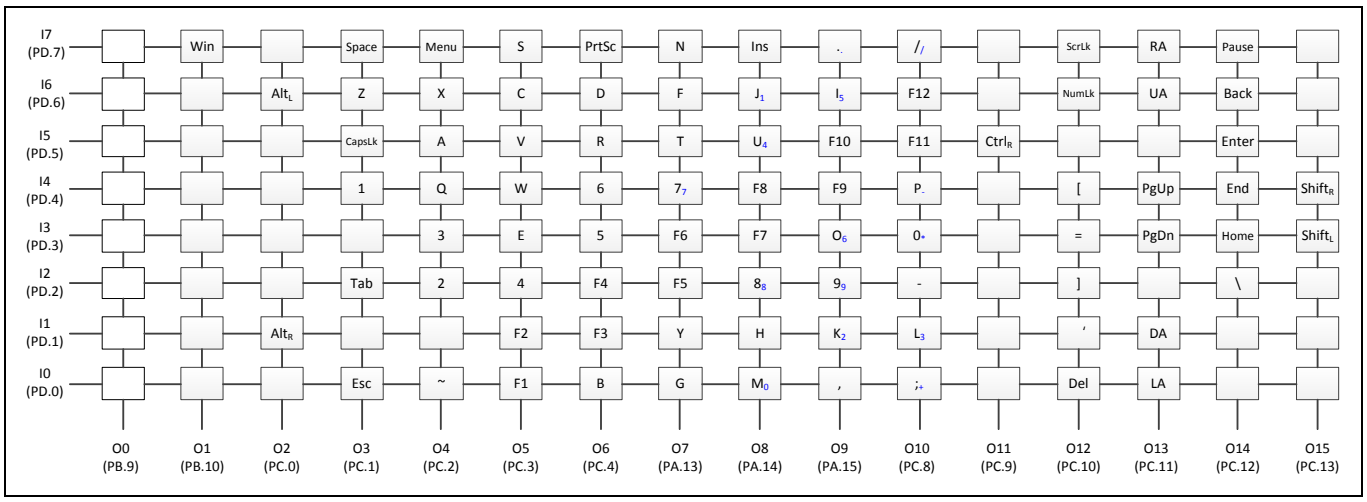


Figure 2-1 键盘矩阵管脚与字符对应图

另外，用 3 根 GPIO 管脚来当作 Caps Lock，Scroll Lock 和 Number Lock On/Off 状态 LED 的显示控制，以及 1 根 GPIO 管脚来当 UART 的 TXD 方便调试信息的显示，Figure 2-2 为 LED 与 UART 的管脚对应图。

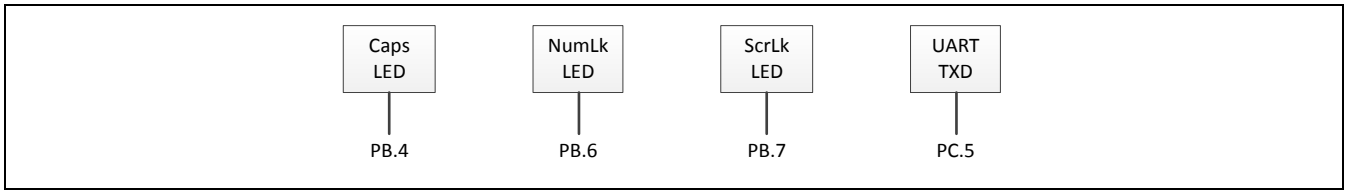


Figure 2-2 LED and UART 管脚对应图

3 USB HID 键盘软件说明

3.1 主程序

USB键盘的主程序是做整个应用系统与外设的初始化动作，再等待相关定时器的中断标志，做键盘的扫描与按键的处理，最后更新键盘数据到相对应端点在 USB SRAM 的数据缓存区，并实时回报给 PC 的 USB Host，使 PC 可以做字符的输入，Figure 3-1 为主程序的流程图。

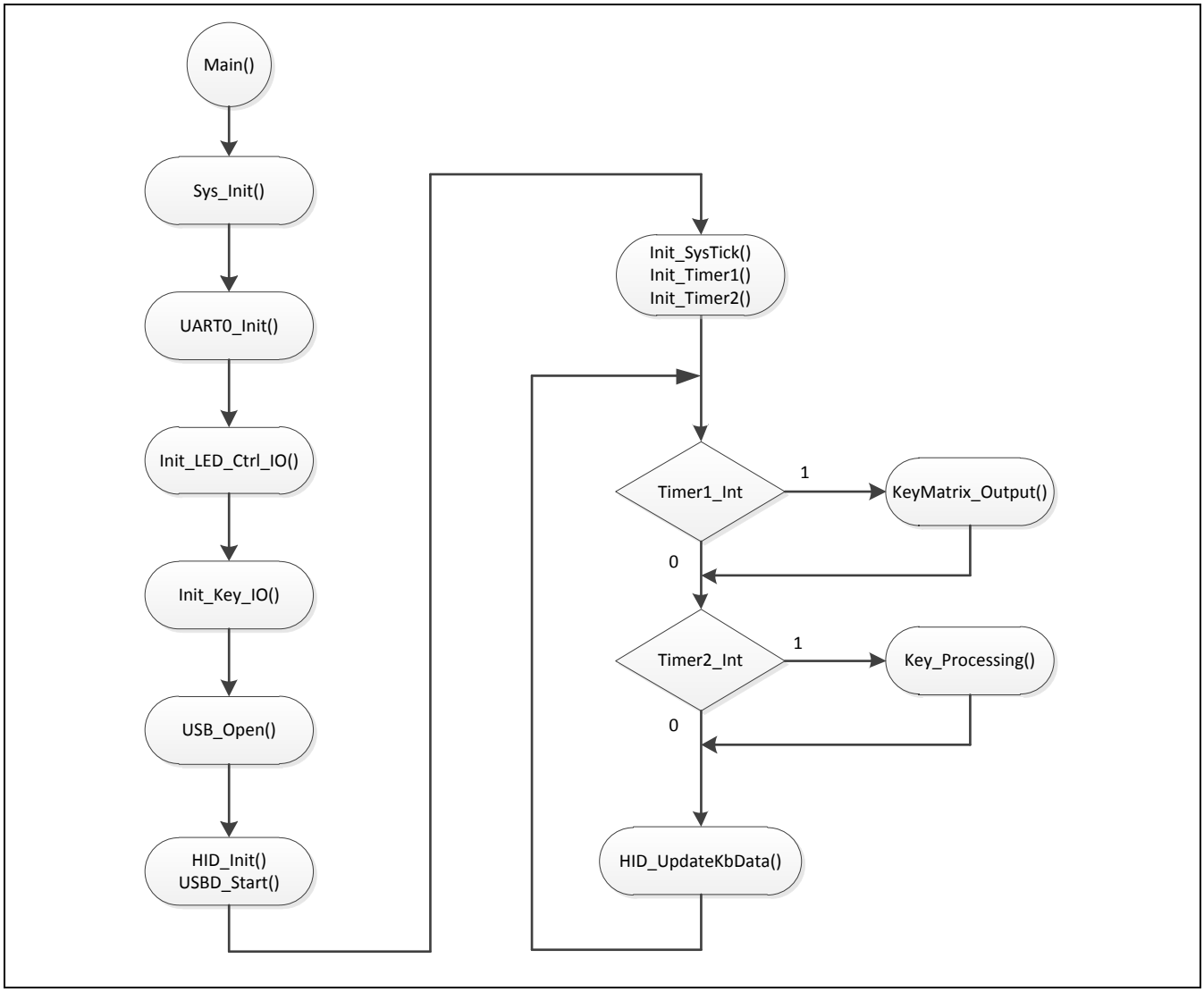


Figure 3-1主程序流程图

```
uint8_t volatile g_u8EP2Ready = 0;
uint8_t g_u8InitialBuffer = 0;

extern volatile   uint8_t    Timer1_Int;      // Timer1 Interrupt flag
extern volatile   uint8_t    Timer2_Int;      // Timer2 Interrupt flag

/*---------------------------------------------------------------------------------------*/
/*  Main Function                                                                        */
/*---------------------------------------------------------------------------------------*/
int32_t main(void)
{
    /* Unlock protected registers */
    SYS_UnlockReg();

    /* Init Syetem */
    SYS_Init();

    /* Lock protected registers */
    SYS_LockReg();

    /* Init UART0 to show messages for debugging */
    UART0_Init();

    printf("\n");
    printf("+------------------------------------------------------------+\n");
    printf("|          NuMicro USB HID Keyboard Sample Code              |\n");
    printf("+------------------------------------------------------------+\n");
    printf("PLL = %d MHz, HCLK = %d MHz.\n", CLK_GetPLLClockFreq(),
CLK_GetHCLKFreq());

    /* Init LED Control IO */
    Init_LED_Ctrl_IO();

    /* Init key IO for keyboard matrix */
    Init_KEY_IO();

    /* USB Device Open */
    USBD_Open(&gsInfo, HID_ClassRequest, NULL);

    /* Endpoint configuration */
    HID_Init();
```

```
    USBD_Start();
    NVIC_EnableIRQ(USBD_IRQn);

    /* Initial Systick, Timer1 and Timer2 */
    Init_SysTick();
    Init_Timer1();
    Init_Timer2();

    /* Start to IN data */
    g_u8EP2Ready = 1;

    while(1)
    {
            /* Key scanning */
            if(Timer1_Int == 1){
                Timer1_Int = 0;
                KeyMatrix_Output();
            }

            /* Key processing */
            if(Timer2_Int == 1){
                Timer2_Int = 0;
                Key_Processing();
            }

            /* Update Key Code */
            HID_UpdateKbData();

    }
}
```

3.2 系统初始化

在主程序中，首先会对系统进行初始设定，包含系统时钟源的选择、使能启用的外设的时钟源，并且设定通用管脚的功能。

```
//********************************
// Initial System
//
//********************************
void SYS_Init(void)
```

```
{

    /*---------------------------------------------------------------------------*/
    /* Init System Clock                                                         */
    /*---------------------------------------------------------------------------*/

    /* Enable Internal RC 22.1184 MHz clock */
    CLK_EnableXtalRC(CLK_PWRCTL_HIRCEN_Msk);

    /* Waiting for Internal RC clock ready */
    CLK_WaitClockReady(CLK_STATUS_HIRCSTB_Msk);

    /* Switch HCLK clock source to Internal RC and HCLK source divide 1 */
    CLK_SetHCLK(CLK_CLKSEL0_HCLKSEL_HIRC, CLK_CLKDIV0_HCLK(1));

    /* Enable external XTAL 12 MHz clock */
    CLK_EnableXtalRC(CLK_PWRCTL_HXTEN_Msk);

    /* Waiting for external XTAL clock ready */
    CLK_WaitClockReady(CLK_STATUS_HXTSTB_Msk);

    /* Set Flash Access Delay */
    FMC->FTCTL |= FMC_FTCTL_FOM_Msk;

    /* Set core clock */
    CLK_SetCoreClock(48000000);

   /* Systick clock source from XHT */
    CLK->CLKSEL0 &= ~(CLK_CLKSEL0_STCLKSEL_Msk);
    CLK->CLKSEL0 |= CLK_CLKSEL0_STCLKSEL_HXT;

    /* Enable module clock */
    CLK_EnableModuleClock(UART0_MODULE);
    CLK_EnableModuleClock(TMR1_MODULE);
    CLK_EnableModuleClock(TMR2_MODULE);
    CLK_EnableModuleClock(USBD_MODULE);

    /* Select module clock source */
    CLK_SetModuleClock(UART0_MODULE, CLK_CLKSEL1_UARTSEL_HXT, CLK_CLKDIV0_UART(1));
    CLK_SetModuleClock(TMR1_MODULE, CLK_CLKSEL1_TMR1SEL_HXT, 0);
    CLK_SetModuleClock(TMR2_MODULE, CLK_CLKSEL1_TMR2SEL_HXT, 0);
```

```
    CLK_SetModuleClock(USBD_MODULE, 0, CLK_CLKDIV0_USB(2));

    /* Enable USB LDO33 */
    SYS->USBPHY = SYS_USBPHY_LDO33EN_Msk;

    /*---------------------------------------------------------------------------------*/
    /* Init I/O Multi-function                                                         */
    /*---------------------------------------------------------------------------------*/

    /* Set GPA multi-function pins for UART0 TXD(PA.2) and RXD(PA.3) */
    SYS->GPA_MFPL &= ~(SYS_GPA_MFPL_PA2MFP_Msk | SYS_GPA_MFPL_PA3MFP_Msk);
    SYS->GPA_MFPL |= (SYS_GPA_MFPL_PA2MFP_UART0_TXD | SYS_GPA_MFPL_PA3MFP_UART0_RXD);

}
```

3.3 设定 UART

设定 UART 0 波特率为 115200，方便程序打印调试的信息。

```
void UART0_Init(void)
{
    /*---------------------------------------------------------------------------------*/
    /* Init UART                                                                       */
    /*---------------------------------------------------------------------------------*/
    /* Reset UART0 module */
    SYS_ResetModule(UART0_RST);

    /* Configure UART0 and set UART0 Baudrate */
    UART_Open(UART0, 115200);

}
```

3.4 设定 LED GPIO

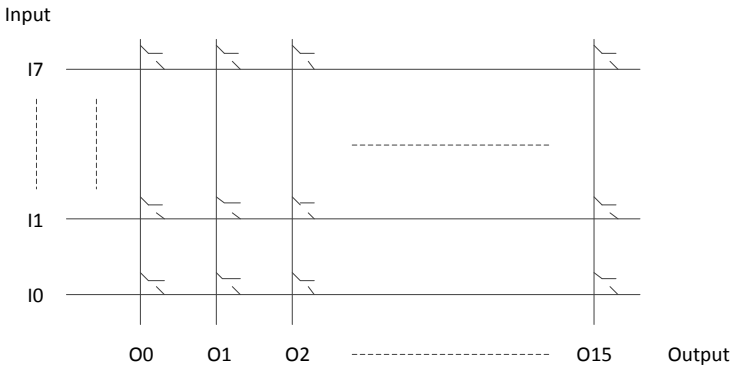
设定 3 根 GPIO 管脚当作 Caps Lock，Scroll Lock 和 Number Lock On/Off 状态 LED 的显示控制。

```
#define CAPLED PB4
#define NUMLED PB6
#define SCRLED PB7
```

```
extern volatile   uint8_t    LED_Status[2];        // LED Status
       volatile   uint8_t    NumLk_flag = 0;       // 1= Number Lock, 0= not

//*******************************
// Initial LED Control IO
//
// GPB.4 = CAPLED as Output mode, =1 LED off, 0=LED on
// GPB.6 = NUMLED as Output mode, =1 LED off, 0=LED on
// GPB.7 = SCRLED as Output mode, =1 LED off, 0=LED on
//*******************************
void Init_LED_Ctrl_IO(void)
{
  /* Set PB.4 (CAPLED) as Output mode and High */
  GPIO_SetMode(PB, BIT4, GPIO_MODE_OUTPUT);
  CAPLED = 1;

  /* Set PB.6 (NUMLED) as Output mode and High */
  GPIO_SetMode(PB, BIT6, GPIO_MODE_OUTPUT);
  NUMLED = 1;

  /* Set PB.7 (SCRLED) as Output mode and High */
  GPIO_SetMode(PB, BIT7, GPIO_MODE_OUTPUT);
  SCRLED = 1;

}
```

3.5 设定 Key GPIO

设定 8 根 GPIO 管脚当作键盘矩阵的输入脚，并使能此 8 根 GPIO 下降沿触发的中断，也开启硬件 de-bounce 的功能。另设定 16 根 GPIO 管脚当作键盘矩阵的输出脚来输出键扫描的低脉冲信号，Figure 3-2 为键盘矩阵输入输出管脚示意图。



Output: O0, O1, O2,…, O15 output low-pulse (20 us) sequentially
Input: I0 or I1 … or I7 will be interrupted when a key be pressed

Figure 3-2 键盘矩阵输入输出管脚示意图

```
//*******************************
// Initial KEY I/O
//
// <INPUT>   GPD(0, 1, 2, 3, 4, 5, 6, 7)
// GPD.0     <== MX0
// GPD.1     <== MX1
// GPD.2     <== MX2
// GPD.3     <== MX3
// GPD.4     <== MX4
// GPD.5     <== MX5
// GPD.6     <== MX6
// GPD.7     <== MX7
//
// <OUTPUT>  GPA(13, 14, 15), GPB(9, 10), GPC(0, 1, 2, 3, 4, 8, 9, 10, 11, 12, 13)
// GPB.9     ==> MY0
// GPB.10    ==> MY1
// GPC.0     ==> MY2
// GPC.1     ==> MY3
// GPC.2     ==> MY4
// GPC.3     ==> MY5
// GPC.4     ==> MY6
// GPA.13    ==> MY7
// GPA.14    ==> MY8
// GPA.15    ==> MY9
```

```
// GPC.8     ==> MYA
// GPC.9     ==> MYB
// GPC.10    ==> MYC
// GPC.11    ==> MYD
// GPC.12    ==> MYE
// GPC.13    ==> MYF
//*******************************
void Init_KEY_IO(void)
{
  uint8_t i;

  /* Set GPD.0~7 as Quasi-bidirection mode */
  GPIO_SetMode(PD, BIT0|BIT1|BIT2|BIT3|BIT4|BIT5|BIT6|BIT7, GPIO_MODE_QUASI);
  /* Enable GPD.0~7 debounce function */
  GPIO_ENABLE_DEBOUNCE(PD, BIT0|BIT1|BIT2|BIT3|BIT4|BIT5|BIT6|BIT7);
  /* Set GPX.i falling-edge trigger and enable interrupt */
  for(i=0;i<=7;i++)
    GPIO_EnableInt(PD, i, GPIO_INT_FALLING);
  /* Set GPD Interrupt Priority */
  NVIC_EnableIRQ(GPD_IRQn);
  NVIC_SetPriority(GPD_IRQn, 3);

  /* Set debounce time = 1/48MHz * 256 = 5.33us */
  GPIO_SET_DEBOUNCE_TIME(GPIO_DBCTL_DBCLKSRC_HCLK, GPIO_DBCTL_DBCLKSEL_256);

  /*==================================*/
  /* Set All Output Keys as Quasi High */
  /*==================================*/
  /* Set GPA.13~15 as Quasi-bidirection mode and High */
  GPIO_SetMode(PA, BIT13 | BIT14 | BIT15, GPIO_MODE_QUASI);
  PA13 = 1;
  PA14 = 1;
  PA15 = 1;

  /* Set GPB.9~10 as Quasi-bidirection mode and High */
  GPIO_SetMode(PB, BIT9 | BIT10, GPIO_MODE_QUASI);
  PB9 = 1;
  PB10 = 1;

  /* Set GPC.0~4 and GPC.8~13 as Quasi-bidirection mode and High */
```

```
  GPIO_SetMode(PC, BIT0 | BIT1 | BIT2 | BIT3 | BIT4 | BIT8 | BIT9 | BIT10 | BIT11 |
BIT12 | BIT13, GPIO_MODE_QUASI);
  PC0 = 1;
  PC1 = 1;
  PC2 = 1;
  PC3 = 1;
  PC4 = 1;
  PC8 = 1;
  PC9 = 1;
  PC10 = 1;
  PC11 = 1;
  PC12 = 1;
  PC13 = 1;


}
```

3.6 设定 USB

规划 USB 使用的端点及其传输型态，以及各端点在 USB SRAM buffer 里相对应的地址与大小。

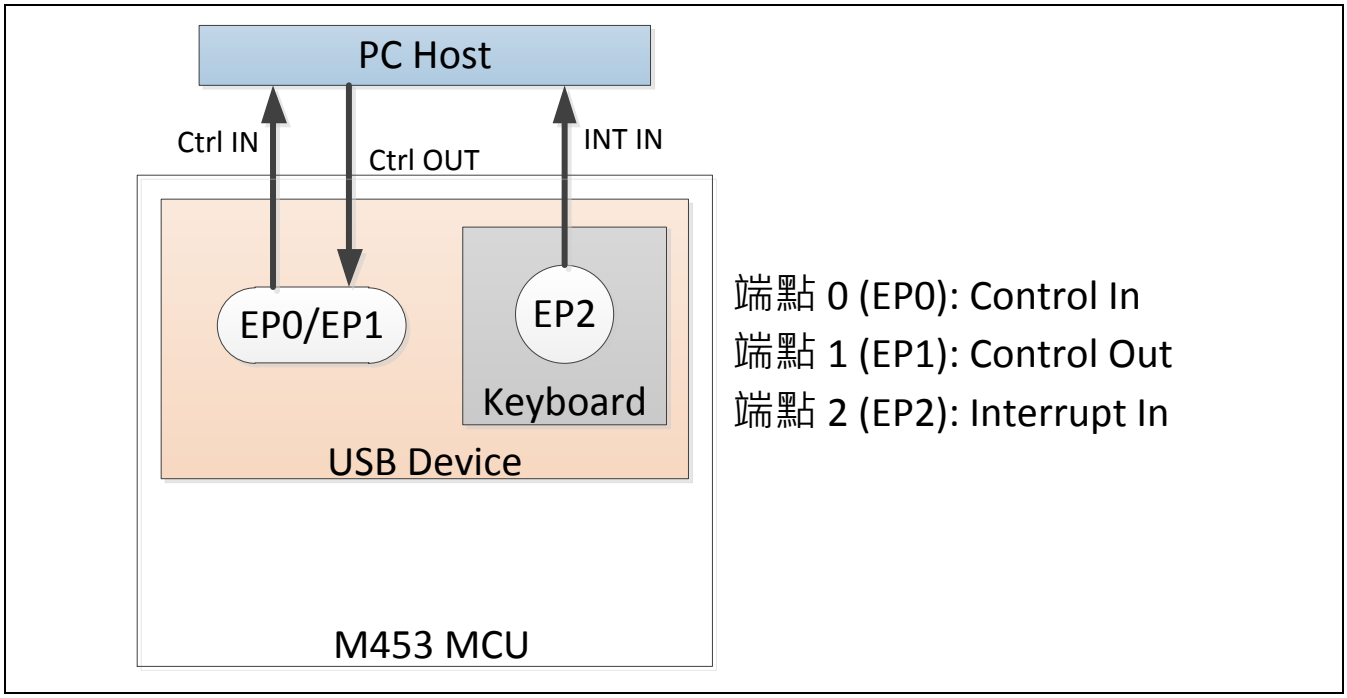


Figure 3-3 USB Device 端点规划图

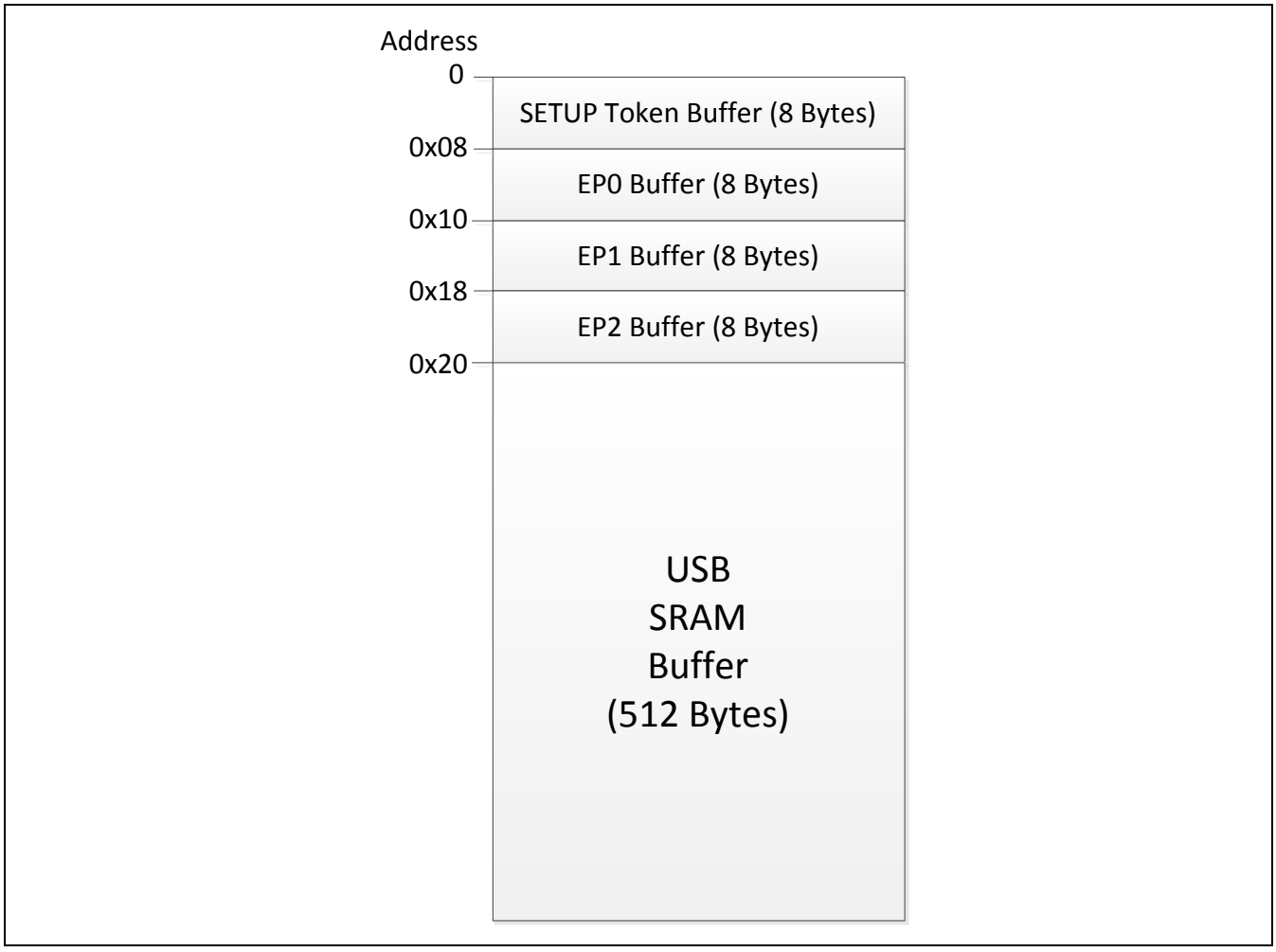


Figure 3-4 各端点 SRAM 数据缓存区规划图

```
/**
  * @brief      This function makes USBD module to be ready to use
  *
  * @param[in]  param           The structure of USBD information.
  * @param[in]  pfnClassReq     USB Class request callback function.
  * @param[in]  pfnSetInterface USB Set Interface request callback function.
  *
  * @return     None
  *
  * @details    This function will enable USB controller, USB PHY transceiver and
pull-up resistor of USB_D+ pin. USB PHY will drive SE0 to bus.
  */
void USBD_Open(const S_USBD_INFO_T *param, CLASS_REQ pfnClassReq, SET_INTERFACE_REQ
pfnSetInterface)
{
```

```
    g_usbd_sInfo = param;
    g_usbd_pfnClassRequest = pfnClassReq;
    g_usbd_pfnSetInterface = pfnSetInterface;

    /* get EP0 maximum packet size */
    g_usbd_CtrlMaxPktSize = g_usbd_sInfo->gu8DevDesc[7];

    /* Initial USB engine */
    USBD->ATTR = 0x7D0;
    /* Force SE0 */
    USBD_SET_SE0();
}

/*--------------------------------------------------------------------------*/
/**
  * @brief  USBD Endpoint Config.
  * @param  None.
  * @retval None.
  */
void HID_Init(void)
{
    /* Init setup packet buffer */
    /* Buffer range for setup packet -> [0 ~ 0x7] */
    USBD->STBUFSEG = SETUP_BUF_BASE;

    /*****************************************************/
    /* EP0 ==> control IN endpoint, address 0 */
    USBD_CONFIG_EP(EP0, USBD_CFG_CSTALL | USBD_CFG_EPMODE_IN | 0);
    /* Buffer range for EP0 */
    USBD_SET_EP_BUF_ADDR(EP0, EP0_BUF_BASE);

    /* EP1 ==> control OUT endpoint, address 0 */
    USBD_CONFIG_EP(EP1, USBD_CFG_CSTALL | USBD_CFG_EPMODE_OUT | 0);
    /* Buffer range for EP1 */
    USBD_SET_EP_BUF_ADDR(EP1, EP1_BUF_BASE);

    /*****************************************************/
    /* EP2 ==> Interrupt IN endpoint, address 1 */
    USBD_CONFIG_EP(EP2, USBD_CFG_EPMODE_IN | INT_IN_EP_NUM);
    /* Buffer range for EP2 */
    USBD_SET_EP_BUF_ADDR(EP2, EP2_BUF_BASE);
```

```
}

/**
  * @brief    This function makes USB host to recognize the device
  *
  * @param    None
  *
  * @return   None
  *
  * @details  Enable WAKEUP, FLDET, USB and BUS interrupts. Disable software-
disconnect function after 100ms delay with SysTick timer.
  */
void USBD_Start(void)
{
    CLK_SysTickDelay(100000);
    /* Disable software-disconnect function */
    USBD_CLR_SE0();

    /* Clear USB-related interrupts before enable interrupt */
    USBD_CLR_INT_FLAG(USBD_INT_BUS | USBD_INT_USB | USBD_INT_FLDET | USBD_INT_WAKEUP);

    /* Enable USB-related interrupts. */
    USBD_ENABLE_INT(USBD_INT_BUS | USBD_INT_USB | USBD_INT_FLDET | USBD_INT_WAKEUP);
}
```

3.7 设定 Timer

初始化 System Timer (SysTick)，以及设定 1 组定时器 (Timer1) 当作键扫描的间隔时间，另 1 组定时器 (Timer2) 当作 key code 回报给 PC Host 的时间周期。

```
//*******************************
// Initial SysTick
//
//
//*******************************
void Init_SysTick(void)
{
  /* Reset current value and control registers to disable SysTick counting */
  SysTick->VAL   = 0;          // Reset current value register
  SysTick->CTRL = 0;          // Reset control register
```

```
}

//*******************************
// Initial Timer1 (5ms)
//
// for key scan
//
// Clock Source = 12 MHz crystal
// Periodic Mode
//*******************************
void Init_Timer1(void)
{
  /* Reset Timer1 */
  SYS_ResetModule(TMR1_RST);

  /* Time out period = (Period of Timer clock input) * (8-bit Prescale + 1) * (24-bit TCMP) = 0.0000833ms * 1 * 60000 = 5ms */
  TIMER1->CTL |= TIMER_CTL_RSTCNT_Msk;
  TIMER1->CTL &= ~TIMER_CTL_PSC_Msk;
  TIMER1->CMP = 60000;

  /* Set Timer1 as Periodic mode */
  TIMER1->CTL &= ~TIMER_CTL_OPMODE_Msk;
  TIMER1->CTL |= TIMER_PERIODIC_MODE;

  /* Enable Timer1 interrupt */
  TIMER1->CTL |= TIMER_CTL_INTEN_Msk;
  NVIC_EnableIRQ(TMR1_IRQn);

  /* Set Timer1 priority */
  NVIC_SetPriority(TMR1_IRQn, 3);

  /* Set Timer1 enable flag */
  Timer1_En = 1;

  /* Enable Timer1 */
  TIMER1->CTL |= TIMER_CTL_CNTEN_Msk;

}
```

```
//*****************************
// Initial Timer2 (30ms)
//
// for key report
//
// Clock Source = 12 MHz crystal
// Periodic Mode
//*****************************
void Init_Timer2(void)
{
  /* Reset Timer2 */
  SYS_ResetModule(TMR2_RST);

  /* Time out period = (Period of Timer clock input) * (8-bit Prescale + 1) * (24-bit TCMP) = 0.0000833ms * 40 * 9000 = 30ms */
  TIMER2->CTL |= TIMER_CTL_RSTCNT_Msk;
  TIMER2->CTL &= ~TIMER_CTL_PSC_Msk;
  TIMER2->CTL |= (39 << TIMER_CTL_PSC_Pos);
  TIMER2->CMP = 9000;

  /* Set Timer2 as Periodic mode */
  TIMER2->CTL &= ~TIMER_CTL_OPMODE_Msk;
  TIMER2->CTL |= TIMER_PERIODIC_MODE;

  /* Enable Timer2 interrupt */
  TIMER2->CTL |= TIMER_CTL_INTEN_Msk;
  NVIC_EnableIRQ(TMR2_IRQn);

  /* Set Timer2 priority */
  NVIC_SetPriority(TMR2_IRQn, 3);

  /* Clear Timer2 enable flag */
  Timer2_En = 0;

  /* Disable Timer2 */
  TIMER2->CTL &= ~TIMER_CTL_CNTEN_Msk;

}
```

3.8 键扫描

以固定时间 (Timer1 定时器设定值 5ms) 在 16 根 GPIO 输出管脚上依序输出键扫描的低脉冲信号，当用户有按下键时，该键的输入输出端会从原本的开路状态而形成短路，若恰巧键扫描的低脉冲输出信号刚好扫至该键，该键的输入管脚即会复制此低脉冲信号而产生中断，Figure 3-5 为键扫描的低脉冲信号输出与输入中断的时序图。

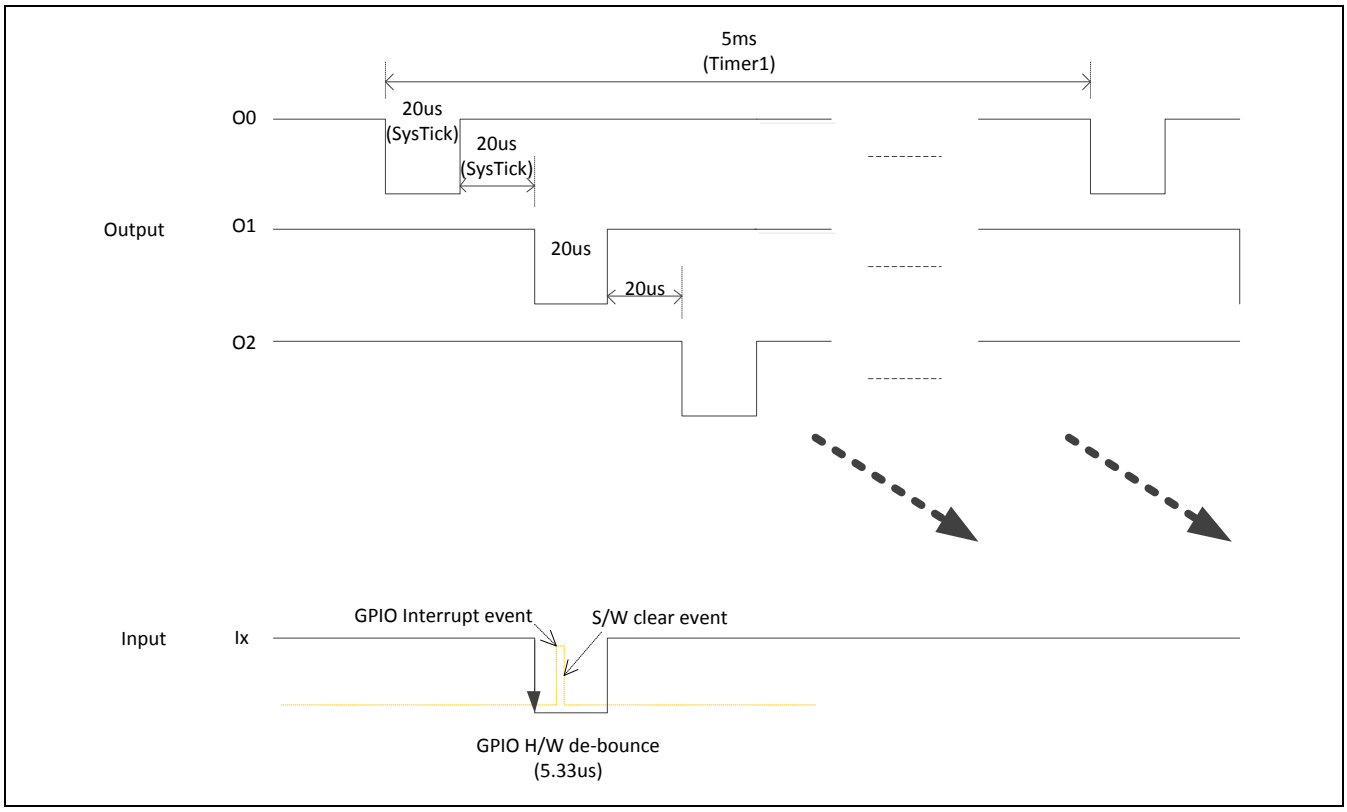


Figure 3-5 键扫描输出信号与输入中断时序图

```
//*******************************
// SysTick_20us
//
//
//*******************************
void SysTick_20us(void)
{
  /* Clear SysTick interrupt flag */
  SysTick_Int = 0;

  /* Set SysTick registers and enable SysTick counting */
  SysTick->LOAD = 225;  // Set reload value register, 225*(1/12MHz) = 18.8us + ISR
latency = 20us
```

```
  SysTick->VAL   = 0;          // Reset current value register
  SysTick->CTRL = SysTick_CTRL_TICKINT_Msk | SysTick_CTRL_ENABLE_Msk;

  /* Wait SysTick interrupt */
  while(!SysTick_Int);

}

//*******************************
// KeyMatrix Output
//
// <OUTPUT>  GPA(13, 14, 15), GPB(9, 10), GPC(0, 1, 2, 3, 4, 8, 9, 10, 11, 12, 13)
// GPB.9     ==> MY0
// GPB.10    ==> MY1
// GPC.0     ==> MY2
// GPC.1     ==> MY3
// GPC.2     ==> MY4
// GPC.3     ==> MY5
// GPC.4     ==> MY6
// GPA.13    ==> MY7
// GPA.14    ==> MY8
// GPA.15    ==> MY9
// GPC.8     ==> MYA
// GPC.9     ==> MYB
// GPC.10    ==> MYC
// GPC.11    ==> MYD
// GPC.12    ==> MYE
// GPC.13    ==> MYF
//*******************************
void KeyMatrix_Output(void)
{
  /* MY0 = GPB.9 */
  MY_out = 0x00;
  PB9 = 0;
  SysTick_20us();
  PB9 = 1;
  SysTick_20us();

  /* MY1 = GPB.10 */
  MY_out = 0x01;
  PB10 = 0;
```

```
    SysTick_20us();
    PB10 = 1;
    SysTick_20us();

    /* MY2 = GPC.0 */
    MY_out = 0x02;
    PC0 = 0;
    SysTick_20us();
    PC0 = 1;
    SysTick_20us();

    /* MY3 = GPC.1 */
    MY_out = 0x03;
    PC1 = 0;
    SysTick_20us();
    PC1 = 1;
    SysTick_20us();

    /* MY4 = GPC.2 */
    MY_out = 0x04;
    PC2 = 0;
    SysTick_20us();
    PC2 = 1;
    SysTick_20us();

    /* MY5 = GPC.3 */
    MY_out = 0x05;
    PC3 = 0;
    SysTick_20us();
    PC3 = 1;
    SysTick_20us();

    /* MY6 = GPC.4 */
    MY_out = 0x06;
    PC4 = 0;
    SysTick_20us();
    PC4 = 1;
    SysTick_20us();

    /* MY7 = GPA.13 */
    MY_out = 0x07;
```

```
PA13 = 0;
SysTick_20us();
PA13 = 1;
SysTick_20us();

/* MY8 = GPA.14 */
MY_out = 0x08;
PA14 = 0;
SysTick_20us();
PA14 = 1;
SysTick_20us();

/* MY9 = GPA.15 */
MY_out = 0x09;
PA15 = 0;
SysTick_20us();
PA15 = 1;
SysTick_20us();

/* MYA = GPC.8 */
MY_out = 0x0A;
PC8 = 0;
SysTick_20us();
PC8 = 1;
SysTick_20us();

/* MYB = GPC.9 */
MY_out = 0x0B;
PC9 = 0;
SysTick_20us();
PC9 = 1;
SysTick_20us();

/* MYC = GPC.10 */
MY_out = 0x0C;
PC10 = 0;
SysTick_20us();
PC10 = 1;
SysTick_20us();

/* MYD = GPC.11 */
```

```
  MY_out = 0x0D;
  PC11 = 0;
  SysTick_20us();
  PC11 = 1;
  SysTick_20us();

  /* MYE = GPC.12 */
  MY_out = 0x0E;
  PC12 = 0;
  SysTick_20us();
  PC12 = 1;
  SysTick_20us();

  /* MYF = GPC.13 */
  MY_out = 0x0F;
  PC13 = 0;
  SysTick_20us();
  PC13 = 1;
  SysTick_20us();

}
```

3.9 按键中断

当有按键时，GPIO 硬件会产生输入的中断，并跳至其中断服务程序做事件的处理。

```
//*******************************
// GPD IRQ Handler
//
// <INPUT>  GPD(0,1,2,3,4,5,6,7)
// GPD.0    <== MX0
// GPD.1    <== MX1
// GPD.2    <== MX2
// GPD.3    <== MX3
// GPD.4    <== MX4
// GPD.5    <== MX5
// GPD.6    <== MX6
// GPD.7    <== MX7
//*******************************
void GPD_IRQHandler(void)
{
```

```
uint32_t u32GPDStatus;
uint8_t temp;

/* Keep the interrupt events */
u32GPDStatus = PD->INTSRC;

/* Clear the interrupt events */
PD->INTSRC = u32GPDStatus;

/* Enable Timer2 */
if(Timer2_En == 0){
      TIMER2->CTL |= TIMER_CTL_CNTEN_Msk;
      Timer2_int_cnt = 0;
      Timer2_En = 1;
}

//printf("GPD Interrupt! GPD_INTSRC:0x%04x. \n", u32GPDStatus);
temp = (u32GPDStatus & 0xff);

if(Timer2_En == 1){
      switch(MY_out){
            case  0x00:
                        Yn_X_temp[0] |= temp;
                        break;
            case  0x01:
                        Yn_X_temp[1] |= temp;
                        break;
            case  0x02:
                        Yn_X_temp[2] |= temp;
                        break;
            case  0x03:
                        Yn_X_temp[3] |= temp;
                        break;
            case  0x04:
                        Yn_X_temp[4] |= temp;
                        break;
            case  0x05:
                        Yn_X_temp[5] |= temp;
                        break;
            case  0x06:
                        Yn_X_temp[6] |= temp;
```

```
                        break;
            case  0x07:
                        Yn_X_temp[7] |= temp;
                        break;
            case  0x08:
                        Yn_X_temp[8] |= temp;
                        break;
            case  0x09:
                        Yn_X_temp[9] |= temp;
                        break;
            case  0x0A:
                        Yn_X_temp[10] |= temp;
                        break;
            case  0x0B:
                        Yn_X_temp[11] |= temp;
                        break;
            case  0x0C:
                        Yn_X_temp[12] |= temp;
                        break;
            case  0x0D:
                        Yn_X_temp[13] |= temp;
                        break;
            case  0x0E:
                        Yn_X_temp[14] |= temp;
                        break;
            case  0x0F:
                        Yn_X_temp[15] |= temp;
                        break;
            default:
                        break;
      }
 }

}
```

3.10 按键处理

以固定时间 (Timer2 定时器设定值 30ms) 来处理按键，查出被按键的 key code 表后，并存放于 KeyCode_Buf[0..7] 内。

```
//*******************************
```

```
// Key Processing
//
//
//*******************************
void Key_Processing()
{
  /* Key judge */
  if(Key_Judge() == 1){
        /* Clear Timer2 Interrupt counter */
        Timer2_int_cnt = 0;

        /* Set KeyCode_Rdy */
        KeyCode_Rdy = 1;

        /* Clear Key_Released */
        Key_Released = 0;

  }
  else if(Timer2_int_cnt > 0){
        /* Disable Timer2 */
        TIMER2->CTL &= ~TIMER_CTL_CNTEN_Msk;
        Timer2_En = 0;

        /* MY_out out */
        MY_out = 0xff;

        /* Set Key_Released */
        Key_Released = 1;
  }

}

const uint8_t KeyCode_Board[16][8] = {
 // MX7,  MX6,  MX5,  MX4,  MX3,  MX2,  MX1,  MX0
  {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}, // MY0
  {0xE3, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}, // MY1, GUI
  {0x00, 0xE2, 0x00, 0x00, 0x00, 0x00, 0xE6, 0x00}, // MY2, Alt
  {0x2C, 0x1D, 0x39, 0x1E, 0x00, 0x2B, 0x00, 0x29}, // MY3
  {0x65, 0x1B, 0x04, 0x14, 0x20, 0x1F, 0x00, 0x35}, // MY4
  {0x16, 0x06, 0x19, 0x1A, 0x08, 0x21, 0x3B, 0x3A}, // MY5
  {0x46, 0x07, 0x15, 0x23, 0x22, 0x3D, 0x3C, 0x05}, // MY6
```

```
  {0x11, 0x09, 0x17, 0x24, 0x3F, 0x3E, 0x1C, 0x0A}, // MY7
  {0x49, 0x0D, 0x18, 0x41, 0x40, 0x25, 0x0B, 0x10}, // MY8
  {0x37, 0x0C, 0x43, 0x42, 0x12, 0x26, 0x0E, 0x36}, // MY9
  {0x38, 0x45, 0x44, 0x13, 0x27, 0x2D, 0x0F, 0x33}, // MYA
  {0x00, 0x00, 0xE0, 0x00, 0x00, 0x00, 0x00, 0x00}, // MYB, Ctrl
  {0x47, 0x53, 0x00, 0x2F, 0x2E, 0x30, 0x34, 0x4C}, // MYC
  {0x4F, 0x52, 0x00, 0x4B, 0x4E, 0x00, 0x51, 0x50}, // MYD
  {0x48, 0x2A, 0x28, 0x4D, 0x4A, 0x31, 0x00, 0x00}, // MYE
  {0x00, 0x00, 0x00, 0xE5, 0xE1, 0x00, 0x00, 0x00}  // MYF, Shift
};

// Define Modifiers bits
#define M_LEFT_CTRL     0x01  // 0xE0
#define M_LEFT_SHIFT    0x02  // 0xE1
#define M_LEFT_ALT      0x04  // 0xE2
#define M_LEFT_GUI      0x08  // 0xE3
#define M_RIGHT_CTRL    0x10  // 0xE4
#define M_RIGHT_SHIFT   0x20  // 0xE5
#define M_RIGHT_ALT     0x40  // 0xE6
#define M_RIGHT_GUI     0x80  // 0xE7

// Define Keyboard KeyCode of Ctrl, Shift, Alt, GUI and Fn keys
#define LEFT_CTRL       0xE0
#define LEFT_SHIFT      0xE1
#define LEFT_ALT        0xE2
#define LEFT_GUI        0xE3
#define RIGHT_CTRL      0xE4
#define RIGHT_SHIFT     0xE5
#define RIGHT_ALT       0xE6
#define RIGHT_GUI       0xE7

// Define Keyboard KeyCode of the keys that support under Num Lock
#define KC_7      0x24   // 7
#define KC_8      0x25   // 8
#define KC_9      0x26   // 9
#define KC_0      0x27   // 0
#define KC_U      0x18   // U
#define KC_I      0x0C   // I
#define KC_O      0x12   // O
#define KC_P      0x13   // P
#define KC_J      0x0D   // J
```

```
#define KC_K      0x0E   // K
#define KC_L      0x0F   // L
#define KC_M      0x10   // M
#defineKC_SC      0x33   // ;
#defineKC_DT      0x37   // .
#defineKC_FS      0x38   // /

// Define Keypad KeyCode 0f 0 ~ 9, +, -, *, / and .
#defineKP_DIV     0x54   // /
#defineKP_MLP     0x55   // *
#defineKP_MIN     0x56   // -
#defineKP_PLS     0x57   // +
#defineKP_ETR     0x58   // Enter
#define KP_1      0x59   // 1
#define KP_2      0x5A   // 2
#define KP_3      0x5B   // 3
#define KP_4      0x5C   // 4
#define KP_5      0x5D   // 5
#define KP_6      0x5E   // 6
#define KP_7      0x5F   // 7
#define KP_8      0x60   // 8
#define KP_9      0x61   // 9
#define KP_0      0x62   // 0
#defineKP_DT      0x63   // .

//*******************************
// Key Judge
//
//*******************************
uint8_t Key_Judge(void)
{
  uint8_t i, bbit;
  int8_t  j;
  uint8_t Key_code;
  uint8_t Key_num = 2;

  /* Clear key code buffer */
  for(i=0;i<=7;i++)
    KeyCode_Buf[i] = 0;

  /* Clear Key_Pressed flag */
```

```
  Key_Pressed = 0;

  /* === Get Key Code === */
  for(i=0;i<16;i++){
    bbit = 0x01;
    if(Yn_X[i] != 0){
      for(j=7;j>=0;j--){
        if(Yn_X[i] & bbit){
          Key_Pressed = 1;

          Key_code = KeyCode_Board[i][j];

          //DBG_PRINT("<%02x,%02x> = [%02x] \n", i, j, Key_code);

          /* Modifiers and Fn keys */
          if((Key_code == LEFT_CTRL)  || (Key_code == LEFT_SHIFT)  || (Key_code ==
LEFT_ALT)  || (Key_code == LEFT_GUI)  || \
             (Key_code == RIGHT_CTRL) || (Key_code == RIGHT_SHIFT) || (Key_code ==
RIGHT_ALT) || (Key_code == RIGHT_GUI)){

              switch(Key_code){
                case    LEFT_CTRL:
                        KeyCode_Buf[0] |= M_LEFT_CTRL;
                        break;
                case    LEFT_SHIFT:
                        KeyCode_Buf[0] |= M_LEFT_SHIFT;
                        break;
                case    LEFT_ALT:
                        KeyCode_Buf[0] |= M_LEFT_ALT;
                        break;
                case    LEFT_GUI:
                        KeyCode_Buf[0] |= M_LEFT_GUI;
                         break;
                case    RIGHT_CTRL:
                        KeyCode_Buf[0] |= M_RIGHT_CTRL;
                        break;
                case    RIGHT_SHIFT:
                        KeyCode_Buf[0] |= M_RIGHT_SHIFT;
                        break;
                case    RIGHT_ALT:
                        KeyCode_Buf[0] |= M_RIGHT_ALT;
```

```
                        break;
                case    RIGHT_GUI:
                        KeyCode_Buf[0] |= M_RIGHT_GUI;
                        break;
              }
            }else if((Key_code == KC_7) || (Key_code == KC_8)  || (Key_code == KC_9) ||
(Key_code == KC_0)  || (Key_code == KC_U)  || \
              (Key_code == KC_I) || (Key_code == KC_O) || (Key_code == KC_P) ||
(Key_code == KC_J)  || (Key_code == KC_K)  || \
              (Key_code == KC_L) || (Key_code == KC_SC) || (Key_code == KC_M) ||
(Key_code == KC_DT) || (Key_code == KC_FS)){

                if(NumLk_flag == 1){

                   switch(Key_code){
                     // 7
                     case    KC_7:
                             Key_code = KP_7;
                             break;
                     // 8
                     case    KC_8:
                             Key_code = KP_8;
                             break;
                     // 9
                     case    KC_9:
                             Key_code = KP_9;
                             break;
                     // 0
                     case    KC_0:
                             Key_code = KP_MLP;    // *
                             break;
                     // U
                     case    KC_U:
                             Key_code = KP_4;
                             break;
                     // I
                     case    KC_I:
                             Key_code = KP_5;
                             break;
                     // O
                     case    KC_O:
                             Key_code = KP_6;
```

```
                    break;
                // P
                case    KC_P:
                        Key_code = KP_MIN;    // -
                        break;
                // J
                case    KC_J:
                        Key_code = KP_1;
                        break;
                // K
                case    KC_K:
                        Key_code = KP_2;
                        break;
                // L
                case    KC_L:
                        Key_code = KP_3;
                        break;
                // ;
                case    KC_SC:
                        Key_code = KP_PLS;    // +
                        break;
                // M
                case    KC_M:
                        Key_code = KP_0;
                        break;
                // .
                case    KC_DT:
                        Key_code = KP_DT;
                        break;
                // /
                case    KC_FS:
                        Key_code = KP_DIV;    // /
                        break;
              }
              KeyCode_Buf[2] = Key_code;
              Key_num = 8;
            }
            else{
              if(Key_code != 0x00){
                KeyCode_Buf[Key_num] = Key_code;
                Key_num++;
```

```
                    }
                  }
                }
                /* Other keys */
                else{
                  if(Key_code != 0x00){
                    KeyCode_Buf[Key_num] = Key_code;
                    Key_num++;
                  }
                }
                /* Max bytes of Keycode = 1+1+6 ((Modifier(1)+Constant(1)+Keycode(6)) */
                if(Key_num >= 8)
                  break;
              }
              /* Shift-left bbit */
               bbit = bbit << 1;
            }
            /* Clear Yn_X*/
            Yn_X[i] = 0;
        }
    }

  return Key_Pressed;

}
```

3.11 更新键盘数据

把存放在 KeyCode_Buf[0..7] 的新按键数据更新到端点 2 (EP2) 在 USB SRAM 所在的 buffer 里，等待 PC Host 再发 Interrupt In 时，USB 装置硬件会自动传送此新输入的键数据。

```
void HID_UpdateKbData(void)
{
  uint8_t *buf;

  /* To check if previous report data is processed or not */
  if(g_u8EP2Ready == 0)
    return;

  /* Report data will be put into USB buffer */
  buf = (uint8_t *)(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP2));
```

```
  /* Check KB keycode is ready or not? */
  if(KeyCode_Rdy == 1){
        /* Clear KeyCode_Rdy flag */
        KeyCode_Rdy = 0;

        /* Check KeyCode is equal to Previous KeyCode or not? */
        /* If yes, then return. */
        /* If not, store the KeyCode_Buf to Pre_KeyCode_Buf. */
        if(Compare_KeyCode() == 1)
              return;

        /* Update new report data */
        USBD_MemCopy(buf, (uint8_t *)&KeyCode_Buf, 8);
        g_u8EP2Ready = 0;
        USBD_SET_PAYLOAD_LEN(EP2, 8);
  }
  else if(Key_Released == 1){
        /* Clear Key_Released flag */
        Key_Released = 0;

        /* Update new report data */
        USBD_MemCopy(buf, (uint8_t *)&keycode_temp[0], 8);

        /* Clear Pre_KeyCode_Buf[0]~[7] */
        Clear_Pre_KeyCode_Buf();

        g_u8EP2Ready = 0;
        USBD_SET_PAYLOAD_LEN(EP2, 8);

  }
}
```

3.12 HID 类别的 Request

在 HID_ClassRequest() 函数中，假设 Caps Lock，Scroll Lock 或者 Number Lock 的状态因有按键而改变时，PC Host 收到这三个键中的任一个键的输入后，会发出 HID 类别 request 中的 “SET_REPORT” 命令给 USB 装置，此时 USB 装置要准备接收 Host 传送新的 Caps Lock，Scroll Lock 和 Number Lock 的状态数据，并设定一个 ‘SetReport_flag’ 的标志。

```
void HID_ClassRequest(void)
```

```
{
    uint8_t buf[8];

    USBD_GetSetupPacket(buf);

    if(buf[0] & 0x80)    /* request data transfer direction */
    {
        // Device to host
        switch(buf[1])
        {
            case GET_REPORT:
            {
                break;
            }
            case GET_IDLE:
            {
                break;
            }
            case GET_PROTOCOL:
            {
                break;
            }
            default:
            {
                /* Setup error, stall the device */
                USBD_SetStall(EP0);
                USBD_SetStall(EP1);
                break;
            }
        }
    }
    else
    {
        // Host to device
        switch(buf[1])
        {
            case SET_REPORT:
            {
                if(buf[3] == 2)
                {
                    USBD_PrepareCtrlOut((uint8_t *)&LED_Status[0], 1);
```

```
                    /* Request Type = Output */
                    USBD_SET_DATA1(EP1);
                    USBD_SET_PAYLOAD_LEN(EP1, buf[6]);

                    /* Status stage */
                    USBD_PrepareCtrlIn(0, 0);

                    /* Set SetReport_flag */
                    SetReport_flag = 1;
                }
                break;
            }
            case SET_IDLE:
            {
                /* Status stage */
                USBD_SET_DATA1(EP0);
                USBD_SET_PAYLOAD_LEN(EP0, 0);
                break;
            }
            case SET_PROTOCOL:
            {
                break;
            }
            default:
            {
                // Stall
                /* Setup error, stall the device */
                USBD_SetStall(EP0);
                USBD_SetStall(EP1);
                break;
            }
        }
    }
}
```

3.13 更新键盘 LED 指示灯

在 USBD_IRQHandler() 中断服务程序里，端点 1 (EP1 为 Control Out) 接收到 PC Host 送出 OUT 的数据时，USB Device 硬件会自动产生 EP1 的中断事件标志，而且此 OUT 的数据即为

Caps Lock，Scroll Lock 和 Number Lock 的最新 On/Off 状态数据，此时 MCU 可根据此 On/Off 的状态数据来开启或关闭各 Caps Lock，Scroll Lock 与 Number Lock 的 LED 指示灯。

```
void USBD_IRQHandler(void)
{
    uint32_t u32IntSts = USBD_GET_INT_FLAG();
    uint32_t u32State = USBD_GET_BUS_STATE();

//------------------------------------------------------------------
    if(u32IntSts & USBD_INTSTS_FLDET)
    {
        // Floating detect
        USBD_CLR_INT_FLAG(USBD_INTSTS_FLDET);

        if(USBD_IS_ATTACHED())
        {
            /* USB Plug In */
            USBD_ENABLE_USB();
        }
        else
        {
            /* USB Un-plug */
            USBD_DISABLE_USB();
        }
    }

//------------------------------------------------------------------
    if(u32IntSts & USBD_INTSTS_BUS)
    {
        /* Clear event flag */
        USBD_CLR_INT_FLAG(USBD_INTSTS_BUS);

        if(u32State & USBD_STATE_USBRST)
        {
            /* Bus reset */
            USBD_ENABLE_USB();
            USBD_SwReset();
        }
        if(u32State & USBD_STATE_SUSPEND)
        {
```

```
            /* Enable USB but disable PHY */
            USBD_DISABLE_PHY();
        }
        if(u32State & USBD_STATE_RESUME)
        {
            /* Enable USB and enable PHY */
            USBD_ENABLE_USB();
        }
    }
//------------------------------------------------------------------
    if(u32IntSts & USBD_INTSTS_USB)
    {
        // USB event
        if(u32IntSts & USBD_INTSTS_SETUP)
        {
            // Setup packet
            /* Clear event flag */
            USBD_CLR_INT_FLAG(USBD_INTSTS_SETUP);

            /* Clear the data IN/OUT ready flag of control end-points */
            USBD_STOP_TRANSACTION(EP0);
            USBD_STOP_TRANSACTION(EP1);

            USBD_ProcessSetupPacket();
        }

        // EP events
        if(u32IntSts & USBD_INTSTS_EP0)
        {
            /* Clear event flag */
            USBD_CLR_INT_FLAG(USBD_INTSTS_EP0);

            // control IN
            USBD_CtrlIn();
        }

        if(u32IntSts & USBD_INTSTS_EP1)
        {
            /* Clear event flag */
            USBD_CLR_INT_FLAG(USBD_INTSTS_EP1);
```

```
            // control OUT
            USBD_CtrlOut();

           if(SetReport_flag == 1){
             SetReport_flag = 0;

              // Change LED Status
             Change_LED_OnOff();
           }
        }

        if(u32IntSts & USBD_INTSTS_EP2)
        {
            /* Clear event flag */
            USBD_CLR_INT_FLAG(USBD_INTSTS_EP2);

            // Interrupt IN
            EP2_Handler();
        }

        if(u32IntSts & USBD_INTSTS_EP3)
        {
            /* Clear event flag */
            USBD_CLR_INT_FLAG(USBD_INTSTS_EP3);
        }

        if(u32IntSts & USBD_INTSTS_EP4)
        {
            /* Clear event flag */
            USBD_CLR_INT_FLAG(USBD_INTSTS_EP4);
        }

        if(u32IntSts & USBD_INTSTS_EP5)
        {
            /* Clear event flag */
            USBD_CLR_INT_FLAG(USBD_INTSTS_EP5);
        }

        if(u32IntSts & USBD_INTSTS_EP6)
        {
            /* Clear event flag */
```

```
            USBD_CLR_INT_FLAG(USBD_INTSTS_EP6);
        }

        if(u32IntSts & USBD_INTSTS_EP7)
        {
            /* Clear event flag */
            USBD_CLR_INT_FLAG(USBD_INTSTS_EP7);
        }
    }
    /* clear unknown event */
    USBD_CLR_INT_FLAG(u32IntSts);
}

//*******************************
// Change LED OnOff
//
//*******************************
void Change_LED_OnOff(void)
{
  /* NumLk LED */
  if(LED_Status[0] & 0x01){
        NUMLED = 0;
        NumLk_flag = 1;
  }else{
        NUMLED = 1;
        NumLk_flag = 0;
  }

  /* Caps LED */
  if(LED_Status[0] & 0x02)
        CAPLED = 0;
  else
        CAPLED = 1;

  /* ScrLk LED */
  if(LED_Status[0] & 0x04)
        SCRLED = 0;
  else
        SCRLED = 1;
```

```
}
```

3.14 USB HID 键盘描述符

Figure 3-6 为此 USB HID 键盘应用所需声明的相关描述符架构图，程序里需要事先建置这些描述符表，以便 PC Host 来枚举此 USB Device 时，程序软件能回复 Host 相对应的描述符。

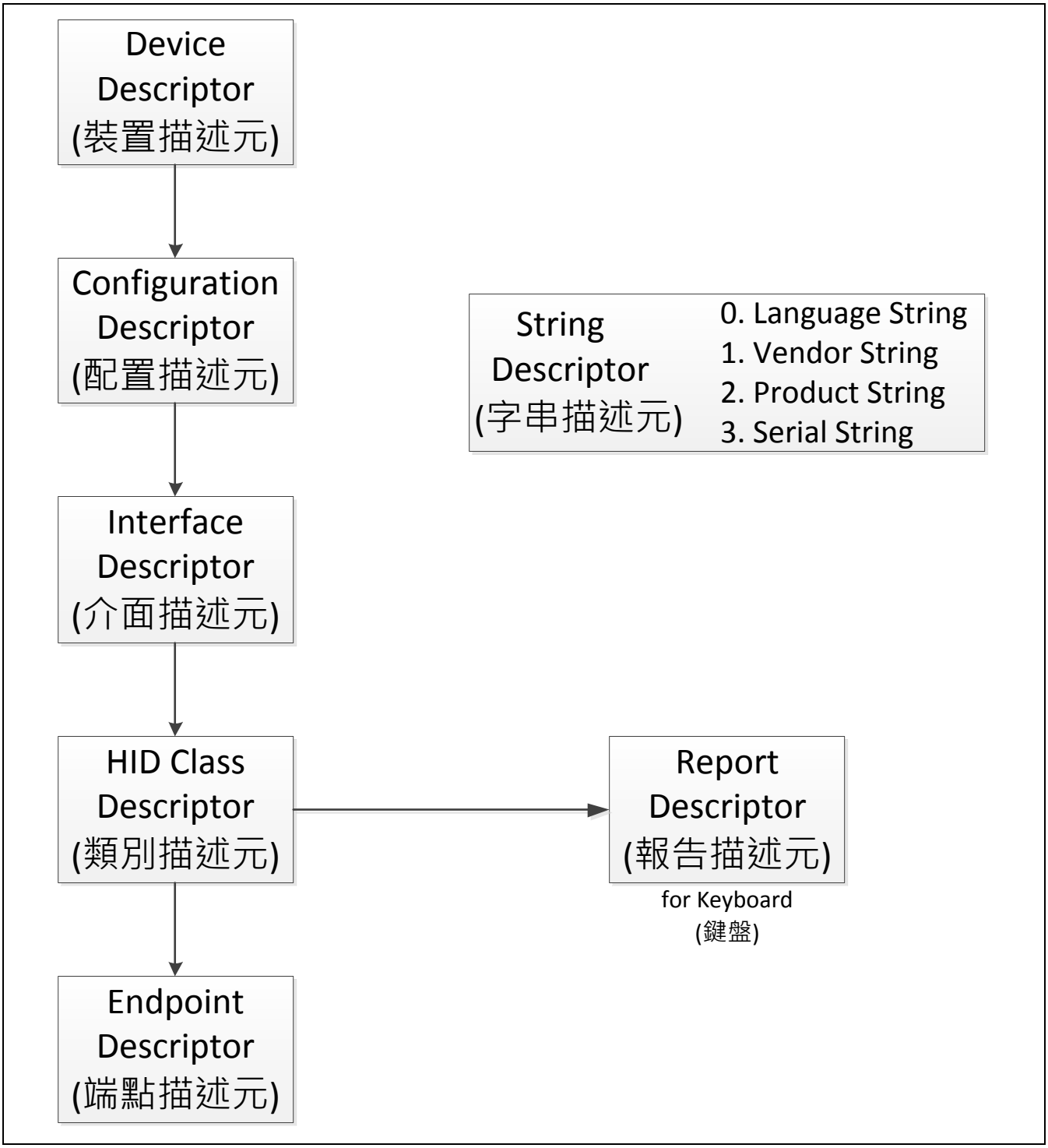


Figure 3-6 描述符架构图

```
typedef struct s_usbd_info
{
    const uint8_t *gu8DevDesc;            /*!< Pointer for USB Device Descriptor
*/
```

```
    const uint8_t *gu8ConfigDesc;         /*!< Pointer for USB Configuration
Descriptor   */
    const uint8_t **gu8StringDesc;        /*!< Pointer for USB String Descriptor
pointers */
    const uint8_t **gu8HidReportDesc;     /*!< Pointer for USB HID Report Descriptor
*/
    const uint32_t *gu32HidReportSize;    /*!< Pointer for HID Report descriptor Size
*/
    const uint32_t *gu32ConfigHidDescIdx; /*!< Pointer for HID Descriptor start index
*/

} S_USBD_INFO_T;

extern const S_USBD_INFO_T gsInfo;

/* Define the vendor id and product id */
#define USBD_VID        0x0416
#define USBD_PID        0xB000

/*!<Define HID Class Specific Request */
#define GET_REPORT          0x01
#define GET_IDLE            0x02
#define GET_PROTOCOL        0x03
#define SET_REPORT          0x09
#define SET_IDLE            0x0A
#define SET_PROTOCOL        0x0B

/*!<USB HID Interface Class protocol */
#define HID_NONE            0x00
#define HID_KEYBOARD        0x01
#define HID_MOUSE           0x02

/*!<USB HID Class Report Type */
#define HID_RPT_TYPE_INPUT      0x01
#define HID_RPT_TYPE_OUTPUT     0x02
#define HID_RPT_TYPE_FEATURE    0x03

/* Define EP maximum packet size */
#define EP0_MAX_PKT_SIZE    8
#define EP1_MAX_PKT_SIZE    EP0_MAX_PKT_SIZE
#define EP2_MAX_PKT_SIZE    8
```

```
#define SETUP_BUF_BASE  0
#define SETUP_BUF_LEN   8
#define EP0_BUF_BASE    (SETUP_BUF_BASE + SETUP_BUF_LEN)
#define EP0_BUF_LEN     EP0_MAX_PKT_SIZE
#define EP1_BUF_BASE    (SETUP_BUF_BASE + SETUP_BUF_LEN)
#define EP1_BUF_LEN     EP1_MAX_PKT_SIZE
#define EP2_BUF_BASE    (EP1_BUF_BASE + EP1_BUF_LEN)
#define EP2_BUF_LEN     EP2_MAX_PKT_SIZE

/* Define the interrupt In EP number */
#define INT_IN_EP_NUM   0x01

/* Define Descriptor information */
#define HID_DEFAULT_INT_IN_INTERVAL     1
#define USBD_SELF_POWERED               0
#define USBD_REMOTE_WAKEUP              0
#define USBD_MAX_POWER                  50  /* The unit is in 2mA. ex: 50*2mA=100mA */

#define LEN_CONFIG_AND_SUBORDINATE     (LEN_CONFIG+LEN_INTERFACE+LEN_HID+LEN_ENDPOINT)

/*!<USB HID Report Descriptor */
const uint8_t HID_KeyboardReportDescriptor[] =
{
    0x05, 0x01,     /* Usage Page(Generic Desktop Controls) */
    0x09, 0x06,     /* Usage(Keyboard) */
    0xA1, 0x01,     /* Collection(Application) */
    0x05, 0x07,     /* Usage Page(Keyboard/Keypad) */
    0x19, 0xE0,     /* Usage Minimum(0xE0) */
    0x29, 0xE7,     /* Usage Maximum(0xE7) */
    0x15, 0x00,     /* Logical Minimum(0x0) */
    0x25, 0x01,     /* Logical Maximum(0x1) */
    0x75, 0x01,     /* Report Size(0x1) */
    0x95, 0x08,     /* Report Count(0x8) */
    0x81, 0x02,     /* Input (Data) => Modifier byte */
    0x95, 0x01,     /* Report Count(0x1) */
    0x75, 0x08,     /* Report Size(0x8) */
    0x81, 0x01,     /* Input (Constant) => Reserved byte */
    0x95, 0x05,     /* Report Count(0x5) */
    0x75, 0x01,     /* Report Size(0x1) */
    0x05, 0x08,     /* Usage Page(LEDs) */
    0x19, 0x01,     /* Usage Minimum(0x1) */
```

```
    0x29, 0x05,     /* Usage Maximum(0x5) */
    0x91, 0x02,     /* Output (Data) => LED report */
    0x95, 0x01,     /* Report Count(0x1) */
    0x75, 0x03,     /* Report Size(0x3) */
    0x91, 0x01,     /* Output (Constant) => LED report padding */
    0x95, 0x06,     /* Report Count(0x6) */
    0x75, 0x08,     /* Report Size(0x8) */
    0x15, 0x00,     /* Logical Minimum(0x0) */
    0x25, 0x65,     /* Logical Maximum(0x65) */
    0x05, 0x07,     /* Usage Page(Keyboard/Keypad) */
    0x19, 0x00,     /* Usage Minimum(0x0) */
    0x29, 0x65,     /* Usage Maximum(0x65) */
    0x81, 0x00,     /* Input (Data) */
    0xC0            /* End Collection */
};

/*---------------------------------------------------------------------------*/
/*!<USB Device Descriptor */
const uint8_t gu8DeviceDescriptor[] =
{
    LEN_DEVICE,     /* bLength */
    DESC_DEVICE,    /* bDescriptorType */
    0x10, 0x01,     /* bcdUSB */
    0x00,           /* bDeviceClass */
    0x00,           /* bDeviceSubClass */
    0x00,           /* bDeviceProtocol */
    EP0_MAX_PKT_SIZE,   /* bMaxPacketSize0 */
    /* idVendor */
    USBD_VID & 0x00FF,
    (USBD_VID & 0xFF00) >> 8,
    /* idProduct */
    USBD_PID & 0x00FF,
    (USBD_PID & 0xFF00) >> 8,
    0x00, 0x00,     /* bcdDevice */
    0x01,           /* iManufacture */
    0x02,           /* iProduct */
    0x03,           /* iSerialNumber - no serial */
    0x01            /* bNumConfigurations */
};

/*!<USB Configure Descriptor */
```

```
const uint8_t gu8ConfigDescriptor[] =
{
    LEN_CONFIG,     /* bLength */
    DESC_CONFIG,    /* bDescriptorType */
    /* wTotalLength */
    LEN_CONFIG_AND_SUBORDINATE & 0x00FF,
    (LEN_CONFIG_AND_SUBORDINATE & 0xFF00) >> 8,
    0x01,           /* bNumInterfaces */
    0x01,           /* bConfigurationValue */
    0x00,           /* iConfiguration */
    0x80 | (USBD_SELF_POWERED << 6) | (USBD_REMOTE_WAKEUP << 5),/* bmAttributes */
    USBD_MAX_POWER,         /* MaxPower */

    /* I/F descr: HID */
    LEN_INTERFACE,  /* bLength */
    DESC_INTERFACE, /* bDescriptorType */
    0x00,           /* bInterfaceNumber */
    0x00,           /* bAlternateSetting */
    0x01,           /* bNumEndpoints */
    0x03,           /* bInterfaceClass */
    0x01,           /* bInterfaceSubClass */
    HID_KEYBOARD,   /* bInterfaceProtocol */
    0x00,           /* iInterface */

    /* HID Descriptor */
    LEN_HID,        /* Size of this descriptor in UINT8s. */
    DESC_HID,       /* HID descriptor type. */
    0x10, 0x01,     /* HID Class Spec. release number. */
    0x00,           /* H/W target country. */
    0x01,           /* Number of HID class descriptors to follow. */
    DESC_HID_RPT,   /* Descriptor type. */
    /* Total length of report descriptor. */
    sizeof(HID_KeyboardReportDescriptor) & 0x00FF,
    (sizeof(HID_KeyboardReportDescriptor) & 0xFF00) >> 8,

    /* EP Descriptor: interrupt in. */
    LEN_ENDPOINT,   /* bLength */
    DESC_ENDPOINT,  /* bDescriptorType */
    (INT_IN_EP_NUM | EP_INPUT), /* bEndpointAddress */
    EP_INT,         /* bmAttributes */
    /* wMaxPacketSize */
```

```
    EP2_MAX_PKT_SIZE & 0x00FF,
    (EP2_MAX_PKT_SIZE & 0xFF00) >> 8,
    HID_DEFAULT_INT_IN_INTERVAL     /* bInterval */
};

/*!<USB Language String Descriptor */
const uint8_t gu8StringLang[4] =
{
    4,              /* bLength */
    DESC_STRING,    /* bDescriptorType */
    0x09, 0x04
};

/*!<USB Vendor String Descriptor */
const uint8_t gu8VendorStringDesc[] =
{
    16,
    DESC_STRING,
    'N', 0, 'u', 0, 'v', 0, 'o', 0, 't', 0, 'o', 0, 'n', 0
};

/*!<USB Product String Descriptor */
const uint8_t gu8ProductStringDesc[] =
{
    26,
    DESC_STRING,
    'H', 0, 'I', 0, 'D', 0, ' ', 0, 'K', 0, 'e', 0, 'y', 0, 'b', 0, 'o', 0, 'a', 0,
'r', 0, 'd', 0
};

const uint8_t gu8StringSerial[26] =
{
    26,             // bLength
    DESC_STRING,    // bDescriptorType
    'A', 0, '0', 0, '2', 0, '0', 0, '1', 0, '4', 0, '0', 0, '9', 0, '0', 0, '3', 0,
'0', 0, '2', 0
};

const uint8_t *gpu8UsbString[4] =
{
    gu8StringLang,
```

```
    gu8VendorStringDesc,
    gu8ProductStringDesc,
    gu8StringSerial
};

const uint8_t *gpu8UsbHidReport[3] =
{
    HID_KeyboardReportDescriptor,
    NULL,
    NULL
};

const uint32_t gu32UsbHidReportLen[3] = {
    sizeof(HID_KeyboardReportDescriptor),
    0,
    0
};

const uint32_t gu32ConfigHidDescIdx[3] = {
    (LEN_CONFIG+LEN_INTERFACE),
    0,
    0
};

const S_USBD_INFO_T gsInfo =
{
    gu8DeviceDescriptor,
    gu8ConfigDescriptor,
    gpu8UsbString,
    gpu8UsbHidReport,
    gu32UsbHidReportLen,
    gu32ConfigHidDescIdx
};
```

4 修订历史

Date	Revision	Description
2016.06.20	1.00	1. 初次发行版本.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.