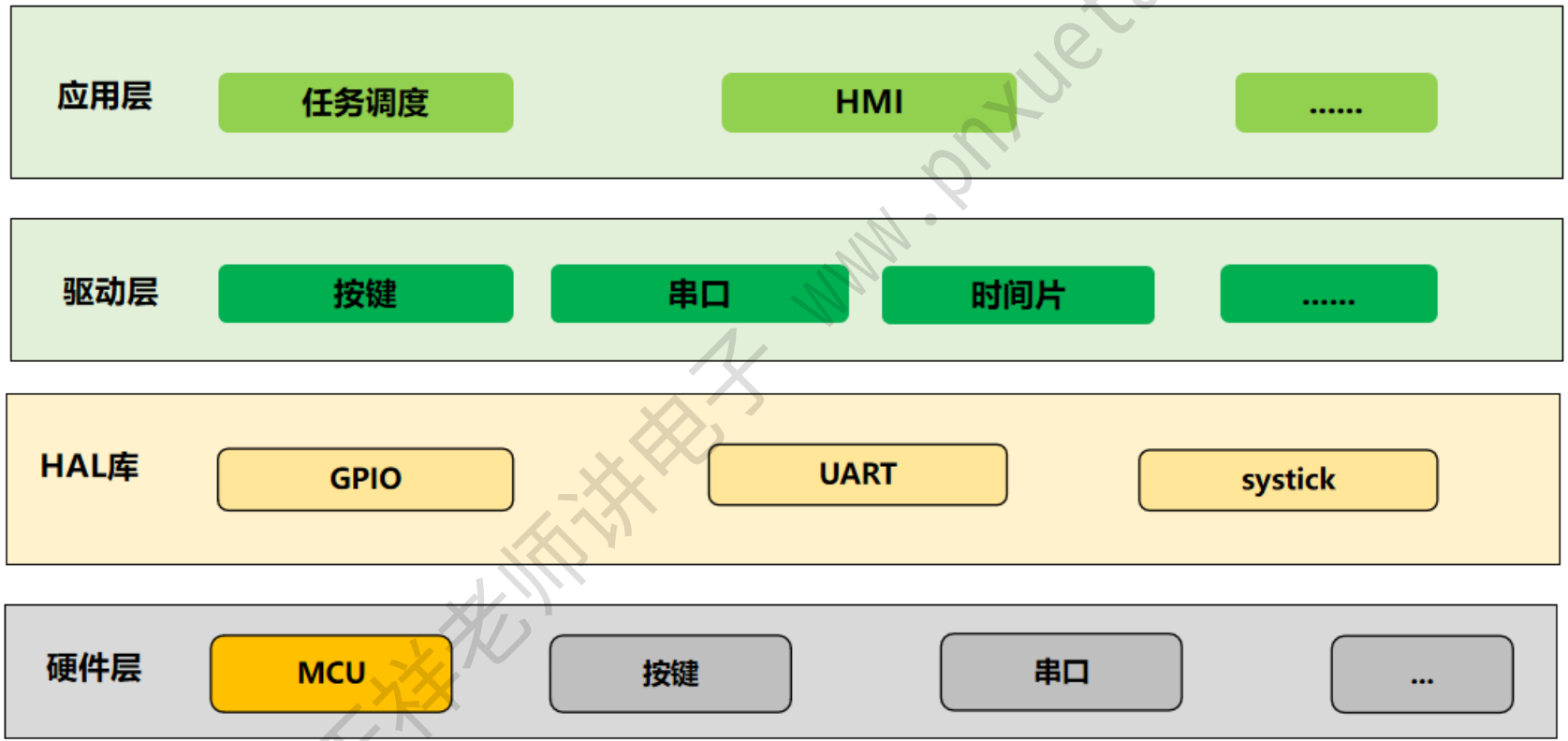
嵌入式C语言之-

基于状态机按键扫描

按键任务软件架构



裸机任务调度方案3，按需分配，软件架构更优

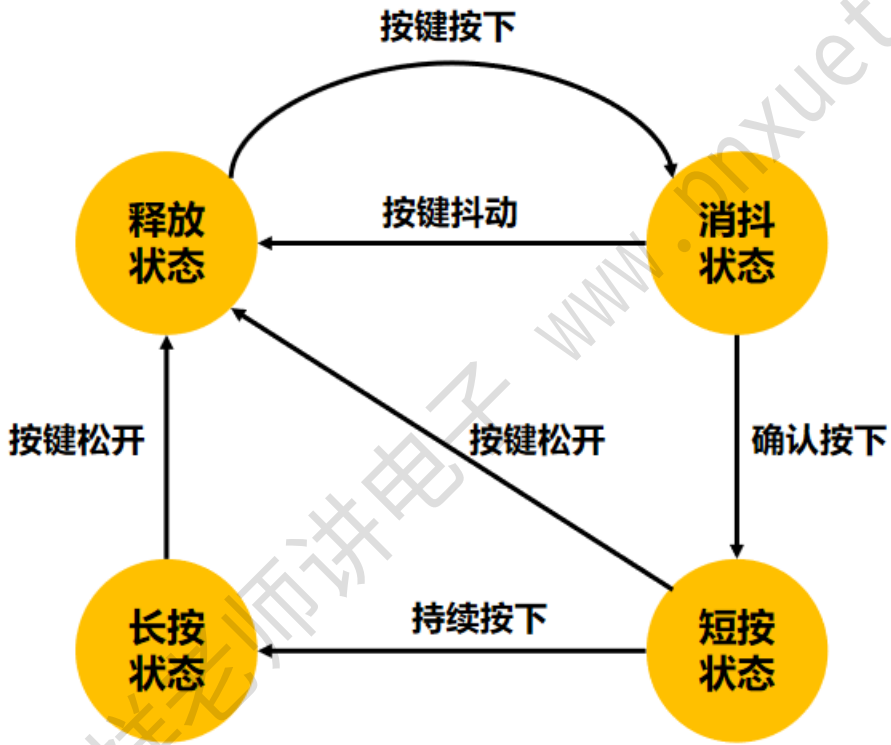
```
typedef struct
{
        uint8_t run;                    //任务状态：Run/Stop
        uint16_t TIMCount;              //时间片周期，用于递减计数
        uint16_t TRITime;               //时间片周期，用于重载
        void (*TaskFuncCb)(void);  //函数指针，保存任务函数地址
} TaskComps_t;
```

```
static TaskComps_t TaskComps[] =
{
        //状态  计数  周期  函数
        //{0, 10, 10, SensorTask},                  /* task 1 Period：  1000ms*/
        {0, 10, 10, HmiTask},                   /* task 2 Period：  10ms */
        /* Add new task here */
};
```

按键扫描，采用延时死等的方法来实现软件消抖

```
if (RESET == gd_eval_key_state_get(KEY1))
{
    /* delay 50ms for software removing jitter */
    delay_1ms(50);
    if(RESET == gd_eval_key_state_get(KEY1))
    {
        printf("key1 is short pressed!\n");
    }
}
```

基于状态机实现按键定时扫描



基于状态机实现按键定时扫描

systick.c代码

```
static uint64_t g_sysRunTime = 0;       //系统运行时间（ms）

void SysTick_Handler(void)
{
        g_sysRunTime++;         //系统运行时间加1
        pTaskScheduleFunc();
}

uint64_t GetSysTime(void)
{
    return g_sysRunTime;
}
```

基于状态机实现按键定时扫描

- hmi_app.c代码

```
#define KEY_NONE_PRESS      0x00
#define KEY1_SHORT_PRESS    0x01
#define KEY1_LONG_PRESS     0x81
#define KEY2_SHORT_PRESS    0x02
#define KEY2_LONG_PRESS     0x82
#define KEY3_SHORT_PRESS    0x03
#define KEY3_LONG_PRESS     0x83
```

```
void HmiTask(void)
{
        KeyScanProcess();
        uint8_t keyVal = GetKeyVal();
        switch (keyVal)
        {
                case KEY1_SHORT_PRESS:
                        printf("key1 is short pressed!\n");
                        break;
                case KEY1_LONG_PRESS:
                        printf("key1 is long pressed!\n");
                        break;
                case KEY2_SHORT_PRESS:
                        ...
        }
}
```

基于状态机实现按键定时扫描

- key_drv.c代码

```
typedef struct
{
        rcu_periph_enum rcu;
        uint32_t gpio;
        uint32_t pin;
} Key_GPIO_t;

/* GPIO和PIN定义 */
static const Key_GPIO_t g_gpioList[] = {
        {RCU_GPIOA, GPIOA, GPIO_PIN_0},
        {RCU_GPIOG, GPIOG, GPIO_PIN_13},
        {RCU_GPIOG, GPIOG, GPIO_PIN_14}
};
```

```
#define KEY_NUM_MAX (sizeof(g_gpioList) / sizeof(g_gpioList[0]))

void KeyInit(void)
{
    for (uint8_t i = 0; i < KEY_NUM_MAX; i++)
    {
        rcu_periph_clock_enable(g_gpioList[i].rcu);
        gpio_init(g_gpioList[i].gpio, GPIO_MODE_IPU,
                GPIO_OSPEED_50MHZ, g_gpioList[i].pin);
    }
}
```

基于状态机实现按键定时扫描

- key_drv.c代码

```
static uint8_t g_keyVal = 0;

uint8_t GetKeyVal(void)
{
    return g_keyVal;
}
```

```
void KeyScanProcess(void)
{
    uint8_t res = 0;

    for (uint8_t i = 0; i < KEY_NUM_MAX; i++)
    {
        res = KeyScan(i);
        if (res != 0)
        {
            g_keyVal = res;
            break;
        }
        g_keyVal = 0;
    }
}
```

基于状态机实现按键定时扫描

- key_drv.c代码

```
typedef enum
{
        KEY_RELEASE = 0,
        KEY_COMFIRM,
        KEY_SHORTPRESS,
        KEY_LONGPRESS
} KEY_STATE;

typedef struct
{
        KEY_STATE keyState;
        uint64_t  prvTimeCount;
        uint64_t  curTimeCount;
} Key_Info_t;
static Key_Info_t keyInfo[KEY_NUM_MAX] = {0};
```

```
#define SHORT_PRESS_TIME     30
//判定短按的时间长度，软件消抖 30ms
#define LONG_PRESS_TIME      1000
//判定长按的时间长度 1000ms
```

基于状态机实现按键定时扫描

- key_drv.c代码

```
static uint8_t KeyScan(uint8_t KeyIndex)
{
    uint8_t keyPress;
    keyPress = gpio_input_bit_get(g_gpioList[KeyIndex].gpio, g_gpioList[KeyIndex].pin);

    switch (keyInfo[KeyIndex].keyState)
    {
        case KEY_RELEASE:                    // 释放状态：判断有无按键按下
            if (!keyPress)                   // 有按键按下
            {
                keyInfo[KeyIndex].prvTimeCount = GetSysTime();    // 获取系统运行时间
                keyInfo[KeyIndex].keyState = KEY_COMFIRM;         // 然后进入 消抖状态
            }
            break;
```

基于状态机实现按键定时扫描

```
case KEY_COMFIRM:                    // 消抖状态
        if (!keyPress)
        {
           keyInfo[KeyIndex].curTimeCount = GetSysTime();              //获取系统运行时间
           if(keyInfo[KeyIndex].curTimeCount - keyInfo[KeyIndex].prvTimeCount >= SHORT_PRESS_TIME)
           {
                keyInfo[KeyIndex].keyState = KEY_SHORTPRESS;   // 如果按键时间超过消抖时间，先设置为 短按状态
           }
        }
        else
       {
            keyInfo[KeyIndex].keyState = KEY_RELEASE;      // 如果按键时间没有超过，判定为误触
       }

        break;
```

基于状态机实现按键定时扫描

```
case KEY_SHORTPRESS:                    // 短按状态：继续判定按键是短按，还是长按
  if(keyPress)                          // 如果按键在 设定的长按时间 内松开，则判定为短按
  {
    keyInfo[KeyIndex].keyState = KEY_RELEASE;     // 设置 释放状态
    return (KeyIndex + 1);        // 返回按键码值，三个按键短按对应0x01 02 03
  }
  else
  {
    keyInfo[KeyIndex].curTimeCount = GetSysTime();              //获取系统运行时间
    if(keyInfo[KeyIndex].curTimeCount - keyInfo[KeyIndex].prvTimeCount >= LONG_PRESS_TIME)
                                                                // 如果按键时间超过 设定的长按时间，为长按
    {
      keyInfo[KeyIndex].keyState = KEY_LONGPRESS;     // 设置 长按状态
    }
  }
  break;
```

基于状态机实现按键定时扫描

```
    case KEY_LONGPRESS:
        if (keyPress)
        {
            keyInfo[KeyIndex].keyState = KEY_RELEASE;        //  按键松开后,设置 释放状态，进行下一次按键的判定
            return (0x80 + KeyIndex + 1);    // 返回按键码值，三个按键长按对应0x81 82 83
        }
        break;

      default:
          keyInfo[KeyIndex].keyState = KEY_RELEASE;
          break;
   }

   return  0;
}
```

基于状态机实现按键定时扫描

```
static uint64_t g_sysRunTime = 0;       //系统运行时间（ms）

void SysTick_Handler(void)
{
        g_sysRunTime++;         //系统运行时间加一
        pTaskScheduleFunc();
}

uint64_t GetSysTime(void)
{
    return g_sysRunTime;
}
```

按键实现切换任务模块

```
static void (*ptestFunc[])(void) =
{
        LedDrvTest,
        BeepDrvTest,
        EepromDrvTest,
        IrDrvTest,
        NorFlashDrvTest
};

#define TEST_FUNC_NUM
(sizeof(ptestFunc) / sizeof(ptestFunc[0]))
```

```
void HmiTask(void)
{
        static int8_t s_testIndex = -1;
        KeyScanProcess();
        uint8_t keyVal = GetKeyVal();
        switch (keyVal)
        {
                case KEY1_SHORT_PRESS:
                        printf("key1 is short pressed!\n");
                        s_testIndex++;
                        s_testIndex %= TEST_FUNC_NUM;

                        (ptestFunc[s_testIndex])();

                        break;
```

THANK YOU!