使用红外遥控器

RT-Thread 评估板 RealTouch 裸机例程

版本号：1.0.0

日期：2012/9/15

修订记录

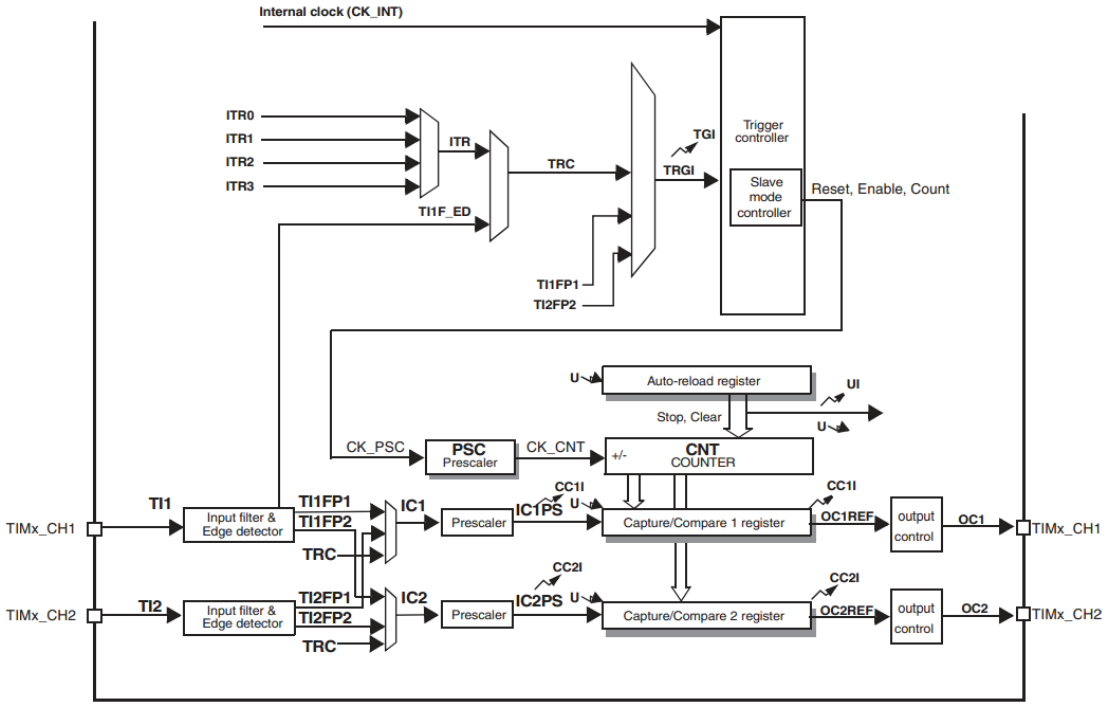
日期	作者	修订历史
2012/9/15	Heyuanjie87	添加例程说明

红外遥控器

红外常用在短距离的遥控设备中，它造价低制造工艺简单因此应用非常广泛。然而红外数据传输格式众多，甚至各个厂商都可以自定一套协议。在简单的应用中，比如只使用四个方向键和两个控制键（确定、返回）的情况下，如果对具体某种制式解码，当更换另一种制式的遥控器后又需要编写另一种解码程序，这样将耗费大量时间。本文利用 STM32 的 PWM 输入功能非常容易地实现了一种自学习型红外识别方法。通过简单的红外学习过程后可适应于多种遥控器。

1. STM32 PWM 输入简介

图一　定时器 9 结构图



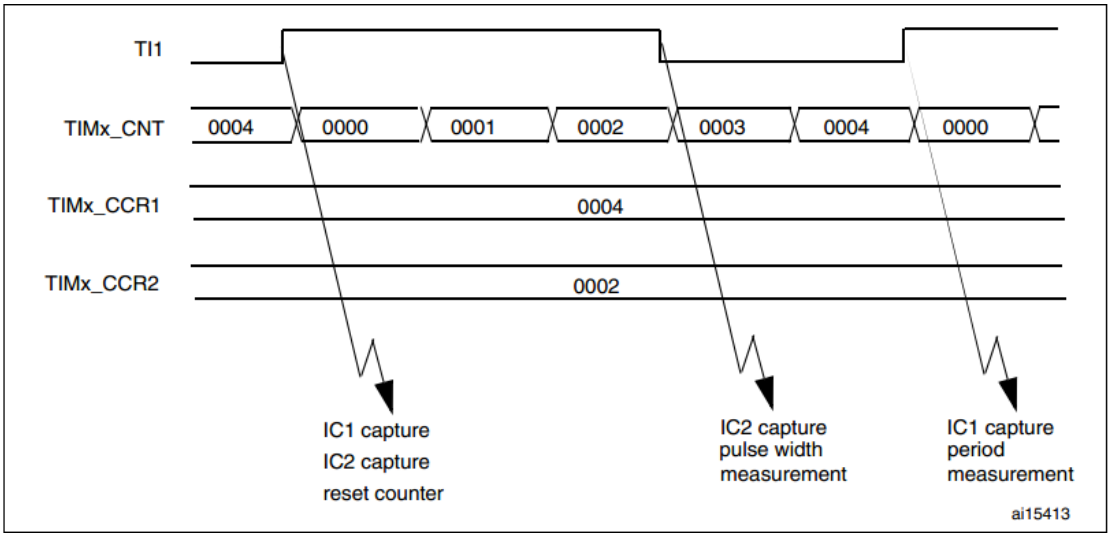
PWM 输入是输入捕获的一个特殊应用，输入捕获就是当连接到定时器的引脚上产生电平变化时对应的捕获装置会立即将当前计数值复制到另一个寄存器中。你可以开启捕获中断然后在中断处理函数中读出保存的计数值。

与输入捕获不同的是 PWM 输入模式会将同一个输入信号（TI1 或 TI2）连接到两个捕获装置（IC1 和 IC2）。这两个捕获装置一个捕获上升沿一个捕获下降沿。TI1FP1、TI2FP2 它们中的一个被选择为触发输入且从模式控制器被配置为复位模式。

下面将演示以 TI1 作为输入端，TIMx_CCR1 作为周期测量 TIMx_CCR2 作为脉宽测量的设置过程。

1) 为 TIMx_CCR1 选择激活输入：写 TIMx_CCMR1 寄存器的 CC1S 位域为’01’（TI1 被选择）。
2) 为 TI1FP1 选择激活极性：设置 CC1P 和 CC1NP 位域为’00’（上升沿激活）。
3) 为 TIMx_CCR2 选择激活输入：设置 TIMx_CCMR1 寄存器的 CC2S 位域为’10’（TI1 被选择）。
4) 为 TI1FP2 选择激活极性：设置 CC2P 与 CC2NP 位域为’11’（下降沿激活）。
5) 选择有效的触发输入：设置 TImx_SMCR 寄存器的 TS 位域为’101’（TI1FP1 被选择）。
6) 配置从模式控制器为复位模式：设置 TIMx_SMCR 寄存器的 SMS 位域为’100’。
7) 使能捕获：设置 TIMx_CCER 寄存器的 CC1E 位和 CC2E 位为 1。

图二. PWM 输入模式时序



1) 当第一次上升沿到达时 IC1 捕获 TIMx_CCR1 的值为当前计数值 4，IC2 不会捕获 TIMx_CCR2 保持不变，计数器复位从 0 开始计数。
2) 第一个下降沿到达时 IC2 捕获 TIMx_CCR2 的值为 2 表示脉冲宽度。当上升再次到达时 TIMx_CCR1 的值就表示脉冲周期了（注意：第一次上升沿捕获的是个随机值）。

2. 本例程所需硬件资源

1) IO 扩展板，使用上面的红外接收模块 KS0338。

2） GPIO_PE6，连接到红外接收模块的输出端。
3） 定时器 9，捕获脉冲周期和脉冲宽度。
4） 红外遥控器（需自备），不能使用空调遥控器。因为空调遥控器按同一个键后发出的脉冲序列不是固定的，它发出的脉冲与当前温度、风速等具体设置有关。

3. 例程

代码一　初始化

```
/* irda.c */

static volatile key_t _key[NUM_KEY + 1];

static void key_init(void)
{
    int i;

    for (i = 0; i < NUM_KEY + 1; i++)
    {
        memset((uint8_t*)&_key[i], 0, sizeof(key_t));
        _key[i].flag = KEY_FLAG_DISABLE;
    }

}

void ir_init(void)
{
    key_init();

    /* GPIO 初始化 */
    {
        GPIO_InitTypeDef GPIO_InitStructure;

        RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOE, ENABLE);

        /* TIM9 通道 2：PE6 */
        GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_6;
        GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_AF;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
        GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
        GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_UP ;
```

```
        GPIO_Init(GPIOE, &GPIO_InitStructure);

        /* 连接 TIM9 到复用功能引脚 */
        GPIO_PinAFConfig(GPIOE, GPIO_PinSource6, GPIO_AF_TIM9);
    }
    /* TIM9 NVIC 设置 */
    {
        NVIC_InitTypeDef NVIC_InitStructure;

        NVIC_InitStructure.NVIC_IRQChannel = TIM1_BRK_TIM9_IRQn;
        NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority =
0;
        NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
        NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
        NVIC_Init(&NVIC_InitStructure);
    }
    /* TIM9 初始化 */
    {
        TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
        TIM_ICInitTypeDef TIM_ICInitStructure;

        RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM9, ENABLE);
        /* APB2 上的定时器输入时钟为 168M，840000 为计数时钟 */
        TIM_TimeBaseStructure.TIM_Prescaler = 168000000 / 840000
- 1;

        TIM_TimeBaseStructure.TIM_CounterMode =
TIM_CounterMode_Up;
        TIM_TimeBaseStructure.TIM_Period = 0xFFFF;
        TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
        TIM_TimeBaseStructure.TIM_RepetitionCounter = 0;

        TIM_TimeBaseInit(TIM9, &TIM_TimeBaseStructure);

        /* PWM 输入设置 */
        TIM_ICInitStructure.TIM_Channel = TIM_Channel_2;
        /* 通道 2 下降沿捕获，通道 1 上升沿捕获 */
        TIM_ICInitStructure.TIM_ICPolarity =
TIM_ICPolarity_Falling;
        TIM_ICInitStructure.TIM_ICSelection =
TIM_ICSelection_DirectTI;
        TIM_ICInitStructure.TIM_ICPrescaler = TIM_ICPSC_DIV1;
        TIM_ICInitStructure.TIM_ICFilter = 0x8;
```

```
        TIM_PWMIConfig(TIM9, &TIM_ICInitStructure);

        /* 选择输入触发: TI2FP2 */
        TIM_SelectInputTrigger(TIM9, TIM_TS_TI2FP2);

        /* 选择从模式: Reset Mode */
        TIM_SelectSlaveMode(TIM9, TIM_SlaveMode_Reset);

TIM_SelectMasterSlaveMode(TIM9,TIM_MasterSlaveMode_Enable);

        /* 使能定时器 */
        TIM_Cmd(TIM9, ENABLE);

        /* 上升沿和定时器溢出产生中断 */
        TIM_ITConfig(TIM9, TIM_IT_CC1 | TIM_IT_Update, ENABLE);
    }
  }
```

首先是按键初始化，这里定义了 6 个按键：上、下、左、右、确定、返回。初始化后，按键是不可用的。因此这里指定了按键标识为 KEY_FLAG_DISABLE（参见工程）。接下来就是对 GPIO 和定时器以及 NVIC 的初始化，例程中用到的 PE6 是连接到 TIM9 的 TI2 的。通道 2 下降沿捕获、通道 1 上升沿捕获，产生中断是在上升沿。因此，通道 2 的值为周期、通道 1 的值为低电平宽度。同时也开启了定时器溢出中断，溢出中断会周期性的产生并对溢出中断的个数进行了计数，这里用作时间基准以确定各个脉冲是在哪个时刻产生的。

代码二　按键等待学习

```
  /* main.c */

  int main(void)
  {
      int key;

      uart_init();

      ir_init();
      wait_for_keystudy();

      /* 省略其它代码 */
  }

  /*********************************************/

  /* irda.c */

  void wait_for_keystudy(void)
  {
     debug("need key study,please press any key\n");
     while (_ir_status == remote_mode_disable);

     for (_key_in_study = 0; _key_in_study < NUM_KEY; _key_in_study
++)
     {
          _key[_key_in_study].key = _key_in_study;

        switch (_key_in_study)
        {
        case 0: /* up */
           debug("press up\n\r");
           _key[_key_in_study].flag = KEY_FLAG_READY;
           while (_key[_key_in_study].flag != KEY_FLAG_ENABLE);
           break;

        case 1: /* down */
           debug("press down\n\r");
           _key[_key_in_study].flag = KEY_FLAG_READY;
           while (_key[_key_in_study].flag != KEY_FLAG_ENABLE);
           break;
```

```
        case 2: /* left */
            debug("press left\n\r");
            _key[_key_in_study].flag = KEY_FLAG_READY;
            while (_key[_key_in_study].flag != KEY_FLAG_ENABLE);
            break;

        case 3: /* right */
            debug("press right\n\r");
            _key[_key_in_study].flag = KEY_FLAG_READY;
            while (_key[_key_in_study].flag != KEY_FLAG_ENABLE);
            break;

        case 4: /* enter */
            debug("press enter\n\r");
            _key[_key_in_study].flag = KEY_FLAG_READY;
            while (_key[_key_in_study].flag != KEY_FLAG_ENABLE);
            break;

        case 5: /* back */
            debug("press back\n\r");
            _key[_key_in_study].flag = KEY_FLAG_READY;
            while (_key[_key_in_study].flag != KEY_FLAG_ENABLE);
            break;
        }
    }

    _key_in_study = 0;
    _key[NUM_KEY].flag = KEY_FLAG_READY;
    _ir_status = remote_mode_enable;

     debug("study over\n\r");
}
```

当程序启动时会提示按任意键，当按任意键后会提示“press up”等，按完六个不同的键后学习完成。

代码二　脉冲采样

```
  /* irda.c */

  void TIM1_BRK_TIM9_IRQHandler(void)
  {
     static uint32_t Pulse_Count =0; /* 统计脉冲个数 */
      static uint32_t Tick = 0; /* 定时器溢出中断计数 */
     uint16_t ccr1 = 0;
     uint16_t ccr2 = 0;

     /* 下降沿 */
     if ((TIM9->SR & TIM_IT_CC2) && !(TIM9->SR & TIM_IT_Update))
     {
         /* 周期 */
        ccr2 = TIM_GetCapture2(TIM9);
     }
     /* 上升沿 */
     if (TIM9->SR & TIM_IT_CC1)
     {
         /* 低电平宽度 */
        ccr1 = TIM_GetCapture1(TIM9);

             /* 按键准备好学习 */
        if ((_ir_status == remote_mode_study) &&
(_key[_key_in_study].flag & KEY_FLAG_READY))
        {
           if (Pulse_Count < CODE_SAMPLING_MAX)
           {
              _key[_key_in_study].period[Pulse_Count] = ccr2;
              _key[_key_in_study].width[Pulse_Count] = ccr1;
           }

           Pulse_Count ++;
                    /* 学习中 */
           _key[_key_in_study].flag |= KEY_FLAG_STUDY;
        }
        /* 键值被读取后才可接受下一次按键信号 */
        if ((_ir_status == remote_mode_enable) &&
(_key[NUM_KEY].flag & KEY_FLAG_READY))
        {
           if (Pulse_Count < CODE_SAMPLING_MAX)
           {
              _key[NUM_KEY].period[Pulse_Count] = ccr2;
```

```
                _key[NUM_KEY].width[Pulse_Count] = ccr1;

            }
                     /* 读到按键信号 */
            _key[NUM_KEY].flag |= KEY_FLAG_ENABLE;
            Pulse_Count ++;
        }
        /* 脉冲的当前时间 */
        Tick = _timer_tick;
    }
    /* 计数器溢出 */
    if (TIM9->SR & TIM_IT_Update)
    {
        TIM_ClearITPendingBit(TIM9, TIM_IT_Update);
             /* 记录时间 */
        _timer_tick ++;

             /* 有任意键按下后才开始学习 */
        if ((_ir_status == remote_mode_disable) && (((_timer_tick
- Tick) == CODE_SAMPLING_END) && Tick != 0))
        {
            _ir_status = remote_mode_study;
        }
        /* 学习完成 */
        if ((_key[_key_in_study].flag & KEY_FLAG_STUDY) &&
((_timer_tick - Tick) == CODE_SAMPLING_END))
        {
            _key[_key_in_study].flag = KEY_FLAG_ENABLE;
            _key[_key_in_study].pulse_count = Pulse_Count;
            /* 清除脉冲计数 */
            Pulse_Count = 0;
        }
        /* 学习完成后有键被按下 */
        if ((_key[NUM_KEY].flag & KEY_FLAG_ENABLE) &&
((_timer_tick - Tick) == CODE_SAMPLING_END))
        {
            _key[NUM_KEY].flag |= KEY_FLAG_PRESS;
            _key[NUM_KEY].flag &= ~KEY_FLAG_READY;
            _key[NUM_KEY].pulse_count = Pulse_Count;

            Pulse_Count = 0;
        }
    }
  }
```

脉冲采样是放在中断处理函数中执行的，主要对脉冲周期、脉冲宽度、脉冲个数进行收集。按键的学习过程以及学习完成后对按键脉冲的采集都是放在上升沿中断处理中的。判断学习完成以及按键脉冲结束是放在定时器溢出中断中处理的。

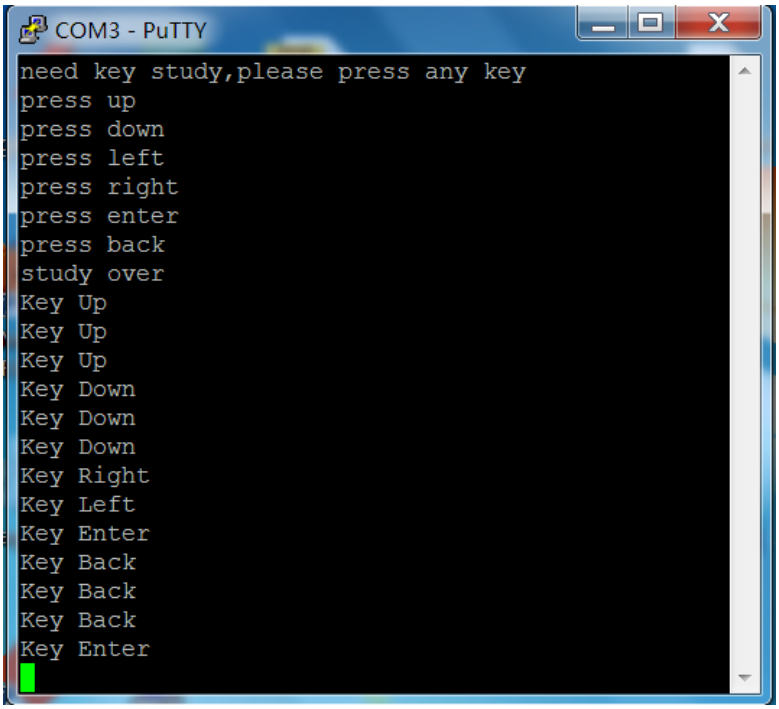
代码三　读取键值

```
  /* irda.c */

  int32_t ir_readkey(void)
  {
     int k,n,s;

     if (_key[NUM_KEY].flag & KEY_FLAG_PRESS)
     {
        for(k = 0; k < NUM_KEY; k ++)
        {
                 /* 应对比的脉冲个数 */
            s = (_key[k].pulse_count < CODE_SAMPLING_MAX) ?
_key[k].pulse_count : CODE_SAMPLING_MAX;
            /* period[0]是个随机数，忽略 */
                    for (n = 1; n < s; n++)
            {
                        /* 对脉冲宽度和脉冲周期进行比较 */
                if ((abs(_key[k].period[n] -
_key[NUM_KEY].period[n]) > CODE_TOLERANCE_MAX) || \
                     (abs(_key[k].width[n] -
_key[NUM_KEY].width[n]) > CODE_TOLERANCE_MAX))
                {
                   //debug("<%d,%d,%d>",abs(_key[k].cycle[n] -
_key[NUM_KEY].cycle[n]),n,k);

//debug("<%d,%d,%d>\n\n\r",abs(_key[k].width[n] -
_key[NUM_KEY].width[n]),n,k);

                   break;
                }
                            /* 完全匹配 */
                if (n == (s - 1))
                   goto _ok;
            }
        }
        /* 清除标志 */
```

```
        _key[NUM_KEY].flag &= ~KEY_FLAG_PRESS;
             /* 可以接受下次按键 */
        _key[NUM_KEY].flag |= KEY_FLAG_READY;
        /* 匹配不成功 */
        return (-1);

_ok:
        _key[NUM_KEY].flag &= ~KEY_FLAG_PRESS;
        _key[NUM_KEY].flag |= KEY_FLAG_READY;
        /* 成功读取 */
        return (k);
    }
    /* 没有键被按下 */
    return (-1);
}
```

当学习完成后，按键的脉冲信息放在了_key[NUM_KEY]中。读取键值时将_key[NUM_KEY]中的信息与_key[0…5]依次进行比较。需要注意的是对脉冲的采样误差需要计算好，不然就很难识别到按键。

下载运行



下载程序后首先会提示“press any key”。这时程序正在等待按键学习，按遥控器任意键后会提示“press up”，按“上”键后依次按完其它键学习完成。这时你可以按学习过的键以及没学习过的键进行测试了，用红外遥控器来控制 RealTouch 吧。