# M251/M252 PWM Capture Rotary

NuMicro® 32 位系列微控制器范例程序代码介绍

## 文件信息

| | |
|---|---|
| 应用简述 | 本范例程序代码使用 M251/M252 PWM Capture 来检测 Rotary Encoder Switch 旋转方向及计数 |
| BSP 版本 | M251/M252/M254/M256/M258 Series BSP CMSIS V3.02.007 |
| 开发平台 | NuMaker-M252SD |

*The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.*

*Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.*

*All data and specifications are subject to change without notice.*

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

www.nuvoton.com

# 1. 概述

此范例代码使用 M251/M252 系列以抓取 Rotary Encoder Switch 所产生 A 相和 B 相方波信号，藉此判断 A 相和 B 相的相对位置来确认其旋转方向和记录旋转位置并通过一组 UART 打印出撷取结果。

## 1.1 原理

PWM capture 主要通过设定中断，中断方式可选择上升沿触发、下降沿触来测量外部信号的周期、频率、占空比。每个信道中可记录最近一次触发时的计数器值，上升沿触发的计数值及下降沿触计数值。

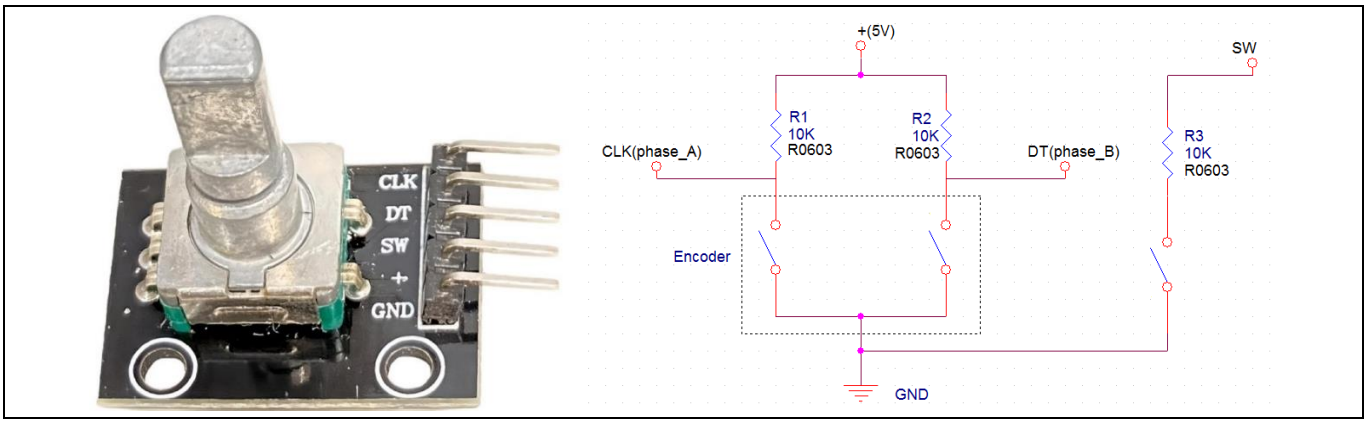


图 1-1 Rotary Encoder Switch

图 1-1 为左侧为 Rotary Encoder Switch 的实体组件，右侧为 Rotary Encoder Switch 的电路，其中有「GND」、「+」、「 SW」、「DT」、「CLK」5 个 PIN，接法为「GND」接电路中的 VSS，「+」接 DC 5V，「 SW」为按键开关功能此范例中不使用，「DT」和「CLK」会产生时序不一样的方波。旋转方向为顺时针时「CLK」产生的方波会超前「DT」产生的方波如图 1-2 及如图 1-3 所示，分别为起始上升沿触发和起始下降沿触发；反之旋转方向为逆时针时「CLK」产生的方波会落后「DT」产生的方波如图 1-4 及如图 1-5 所示，分别为起始上升沿触发和起始下降沿触发，本范例的 Rotary Encoder Switch 旋转一圈「DT」和「CLK」会各自产生 15 个方波，如图 1-6 及图 1-7 所示。

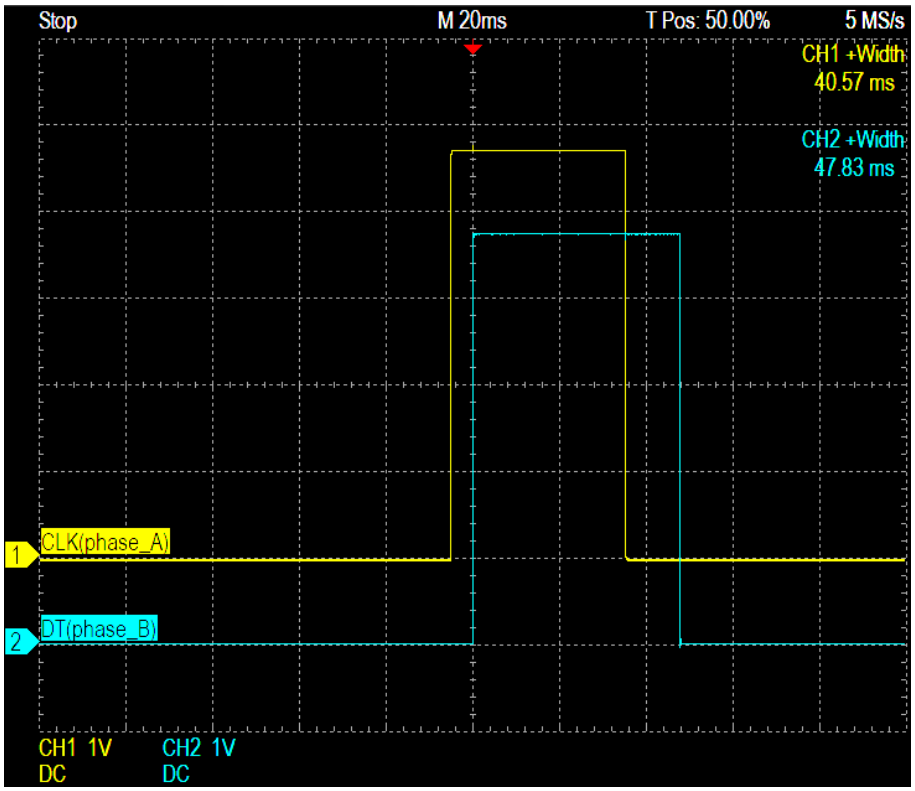


图 1-2 Rotary Encoder Switch 顺时针旋转（起始上升沿触发）

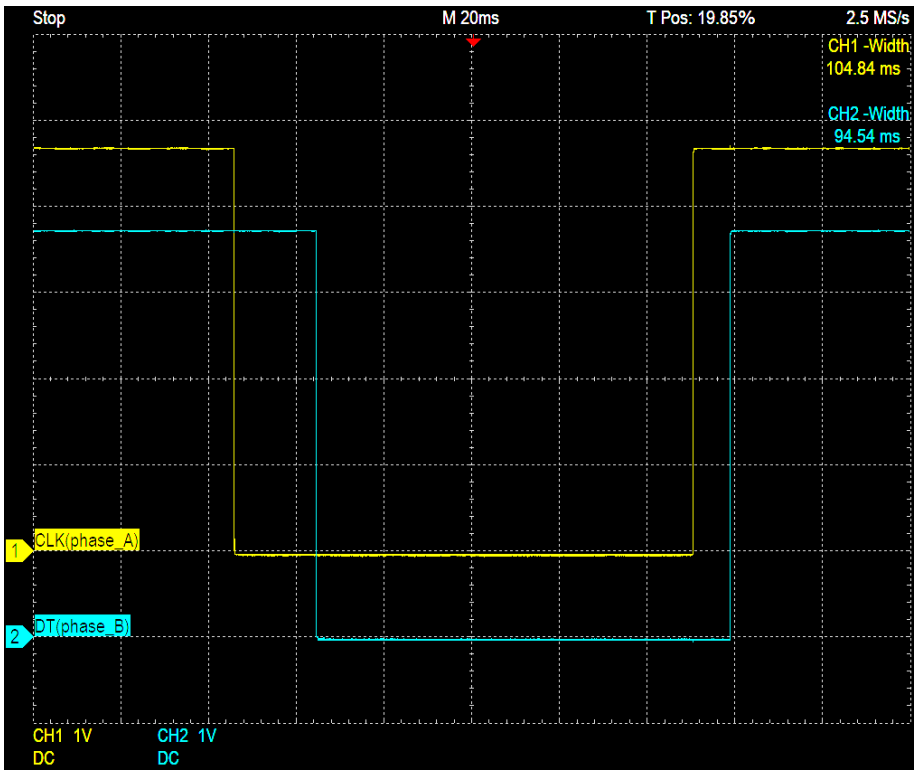


图 1-3 Rotary Encoder Switch 顺时针旋转（起始下降沿触发）

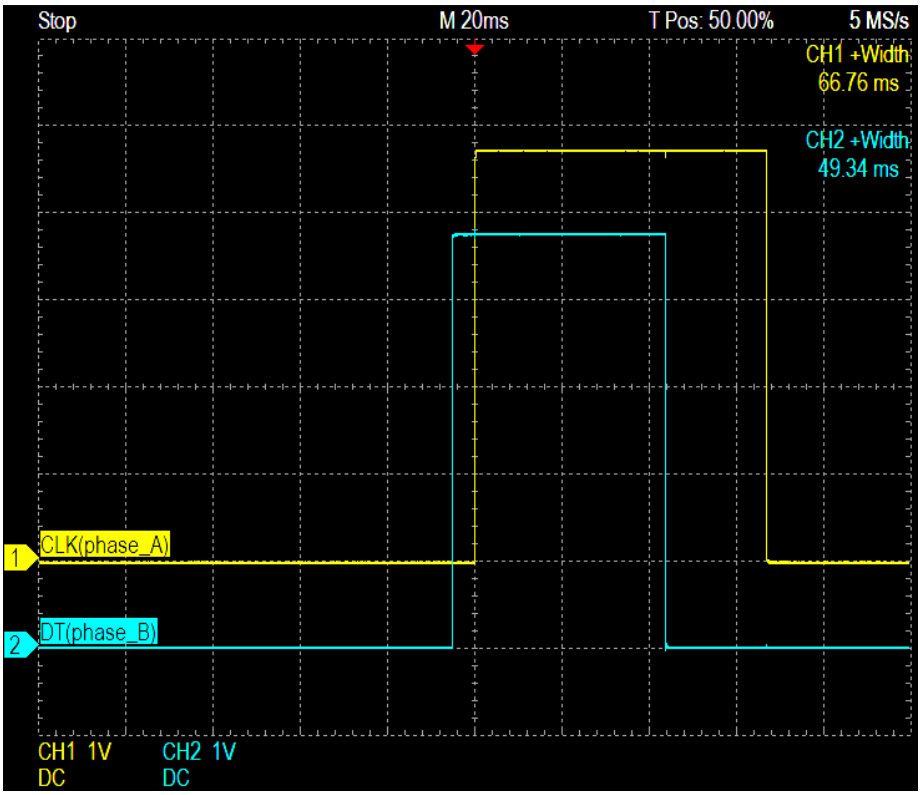


图 1-4 Rotary Encoder Switch 逆时针旋转（起始上升沿触发）

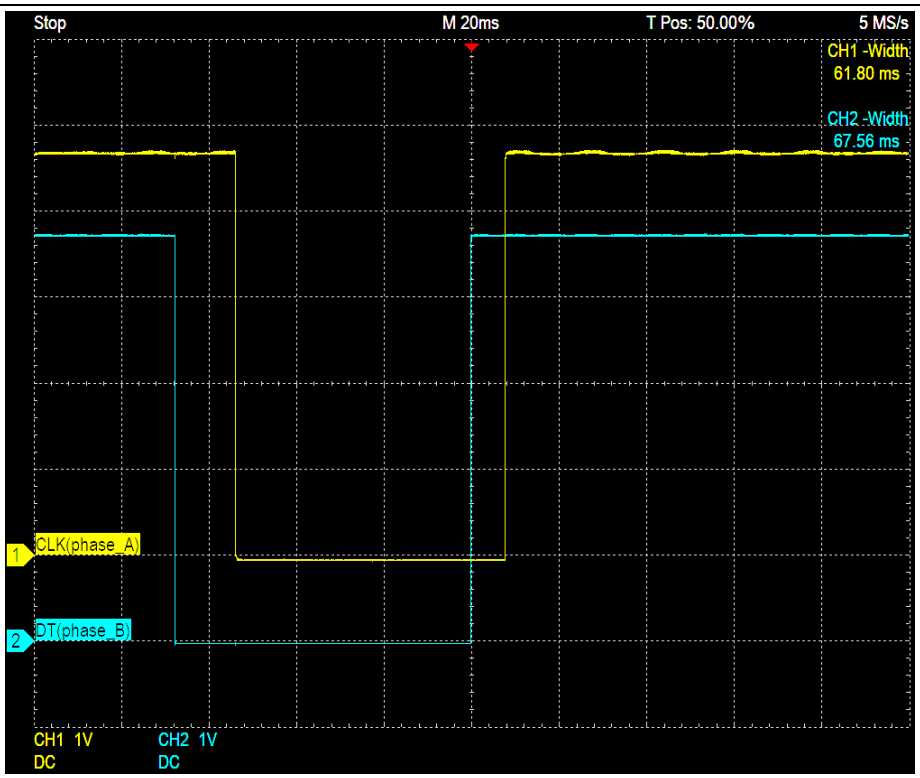


图 1-5 Rotary Encoder Switch 逆时针旋转（起始下降沿触发）

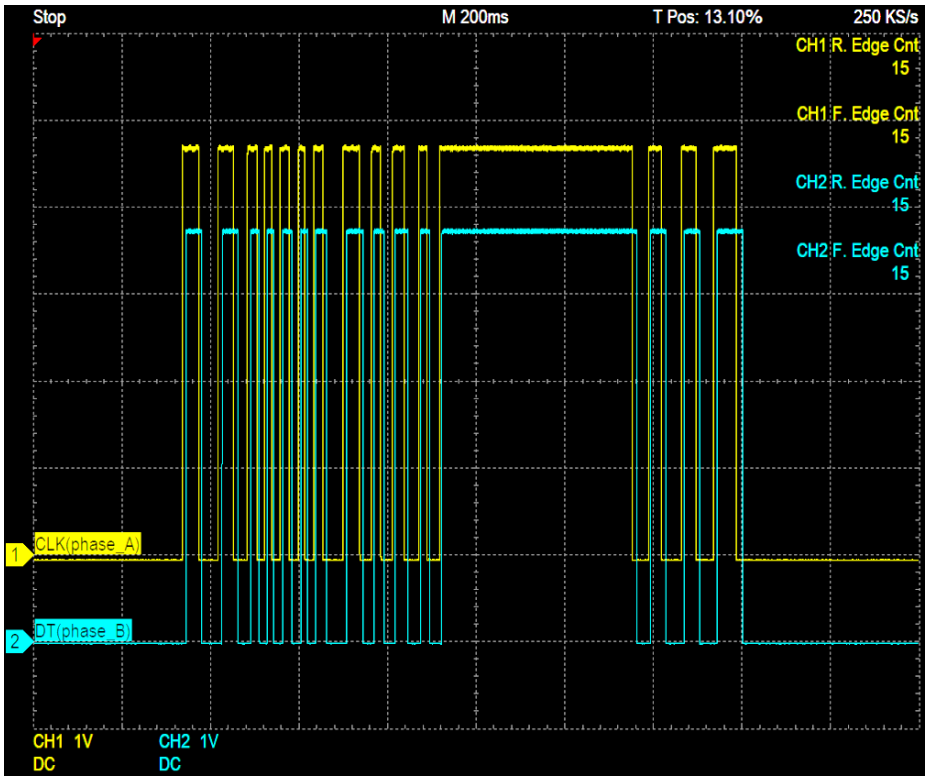


图 1-6 Rotary Encoder Switch 顺时针旋转一圈

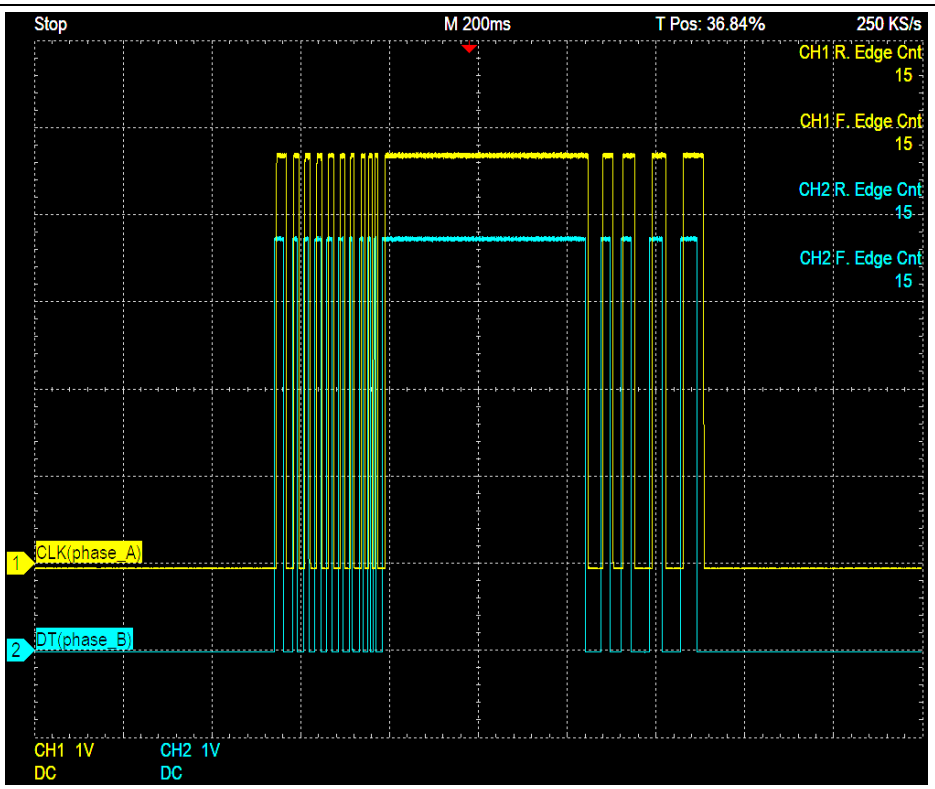


图 1-7 Rotary Encoder Switch 逆时针旋转一圈

由于 Rotary Encoder Switch 是机械式开关，所产生的方波 CLK(phase_A)和 DT(phase_B)会不一样，如图 1-2 所示 CLK(phase_A)的正脉冲宽为 40.57 ms、DT(phase_B) 正脉冲宽为 47.83 ms；如图 1-3 所示 CLK(phase_A)的负脉波宽为 104.84 ms、DT(phase_B) 负脉波宽为 94.54 ms，从上述观察得知同一次中旋转产生 CLK(phase_A)和 DT(phase_B)两方波时间误差较大不适合使用 PWM Capture 自带的 CAPDATn(上升沿触发计数值 RCAPDATn、下降沿触发计数值 FCAPDATn)来判断方波超前或是落后。

故本范例使用 PWM Capture 将 CLK(phase_A)和 DT(phase_B)产生的上升沿触发中断、下降沿触发中断搭配旋转编码器方向查找表（正交译码器）且记录前一个状态 CLK(phase_A)及 DT(phase_B)，当前状态 Current state = (phase_A << 1) | phase_B；也就是说将 phase_A 状态当 MSB，phase_B 状态当 LSB，组合成 2 Bits 的 Data 就是当前状态 Current state，如图 1-8 和图 1-9 分别为顺时针旋转和逆时针旋转产生的当前状态值 Current state。

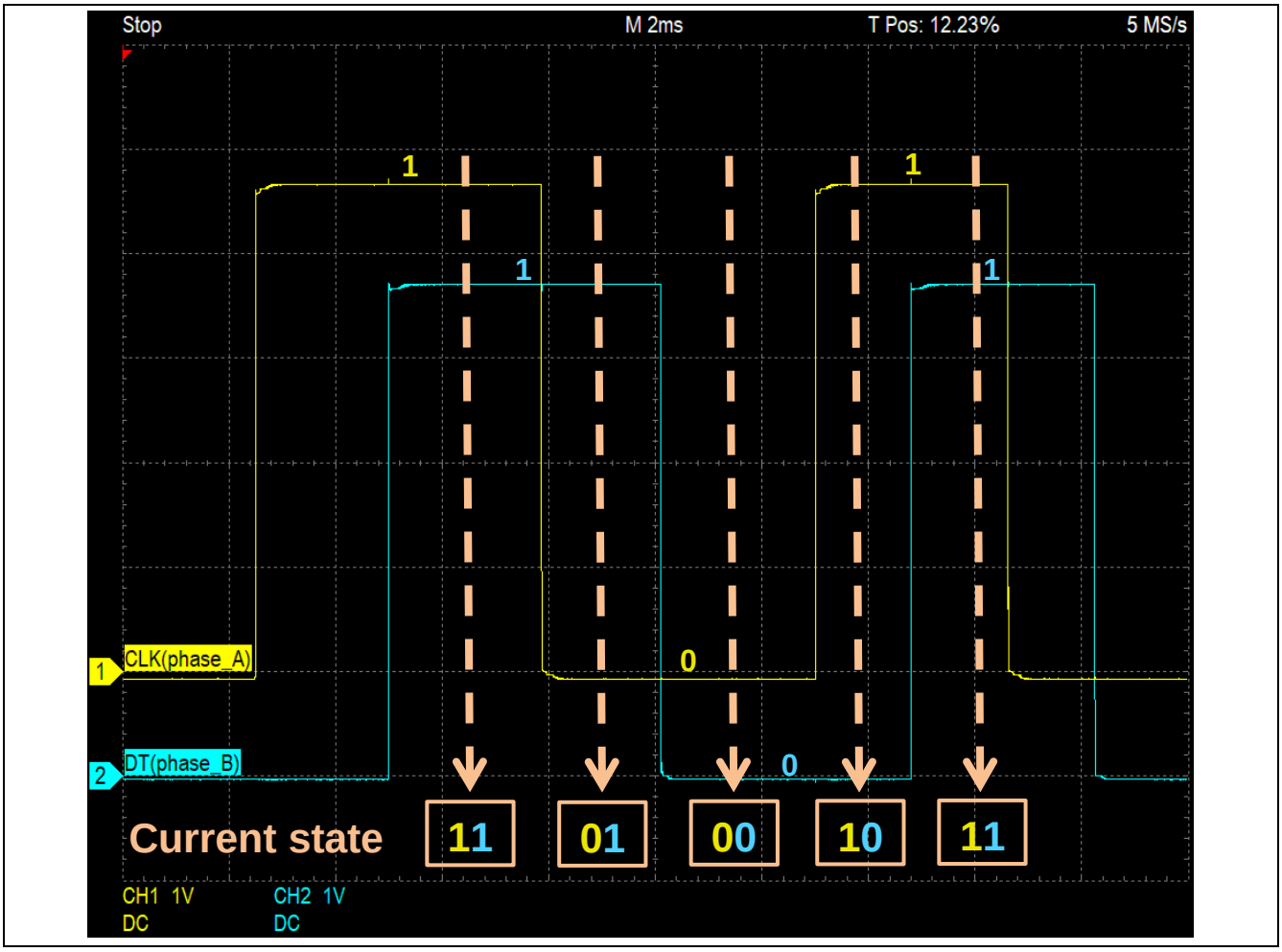


图 1-8 Rotary Encoder Switch 顺时针旋转

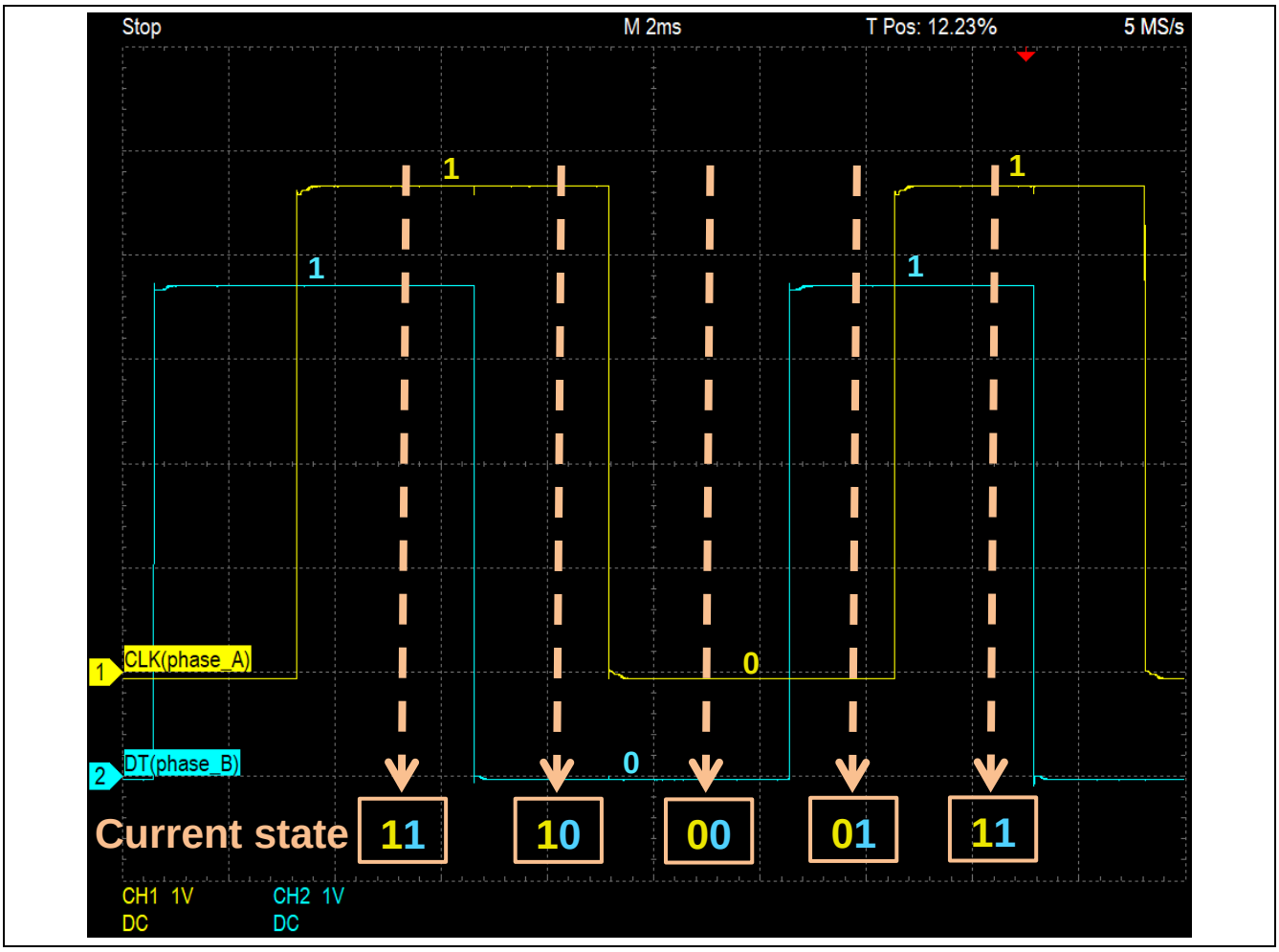


图 1-9 Rotary Encoder Switch 逆时针旋转

顺时针旋转当前状态 Current state 11→01、01→00、00→10、10→11 有 4 种顺序，逆时针旋转 11→10、10→00、00→01、01→11 也是有 4 种顺序，顺时针旋转与逆时针旋转总共有 8 种顺序，将其当前状态 Current state 和上一个状态 Last state 组合起来，可作为旋转状态查表依据如表 1-1；上一个状态 Last state 的 2 Bits 为 MSB，当前状态 Current state 的 2 Bits 为 LSB，也就是 Transition = (Last state << 2) | current state 。

| 顺时针旋转 | 逆时针旋转 |
|---|---|
| 1101 | 1110 |
| 0100 | 1000 |
| 0010 | 0001 |
| 1011 | 0111 |

表 1-1 顺时针旋转与逆时针旋转的上一个状态和当前状态表

| 索引<br>(十进制) | 上一个状态|当前状态<br>Transition =<br>(Last state << 2) \| Current state<br>(二进制) | 有效性 | 旋转方向 | 0=无旋转/无效<br>1=顺时针旋转<br>-1=逆时针旋转 |
|---|---|---|---|---|
| 0 | 0000 | 无 | 无改变 | 0 |
| 1 | 0001 | 有 | 逆时针旋转 | -1 |
| 2 | 0010 | 有 | 顺时针旋转 | 1 |
| 3 | 0011 | 无 | 无 | 0 |
| 4 | 0100 | 有 | 顺时针旋转 | 1 |
| 5 | 0101 | 无 | 无 | 0 |
| 6 | 0110 | 无 | 无 | 0 |
| 7 | 0111 | 有 | 逆时针旋转 | -1 |
| 8 | 1000 | 有 | 逆时针旋转 | -1 |
| 9 | 1001 | 无 | 无 | 0 |
| 10 | 1010 | 无 | 无改变 | 0 |
| 11 | 1011 | 有 | 顺时针旋转 | 1 |
| 12 | 1100 | 无 | 无 | 0 |
| 13 | 1101 | 有 | 顺时针旋转 | 1 |
| 14 | 1110 | 有 | 逆时针旋转 | -1 |
| 15 | 1111 | 无 | 无改变 | 0 |

表 1-2 上一个状态和当前状态组合的 4 Bits 所有情况

如表 1-2 为上一个状态和当前状态组合的 4 Bits 所有情况，有 8 种为有效编码另外 8 种为无效编码，由于 Rotary Encoder Switch 为机械开关，在旋转途中可能会有弹跳现象，弹跳产生的噪声会是上述表 1-2 无效编码情况，只需记录有效编码情况可用于除弹跳现象，最后将顺时针旋转替换为「1」，逆时针旋转替换为「-1」，无效输出替换为「0」。

## 1.2 执行结果

本范例程序代码会有三种执行结果说明，第一种为单纯顺时针旋转，第二种为单为单纯逆时针旋转，第三种为顺时针旋转与逆时针交替旋转。

在程序中 phase_A 和 phase_B 分别使用两种触发中断，分别为上升沿触发中断和下降沿触发中断，

如图 1-10 所示，

- 时序①时为 phase_A 的上升沿触发
- 时序②时为 phase_B 的上升沿触发
- 时序③时为 phase_A 的上升沿触发
- 时序④时为 phase_B 的上升沿触发
- 时序⑤时为 phase_A 的上升沿触发
- 时序⑥时为 phase_B 的上升沿触发
- 时序⑦时为 phase_A 的下降沿触发
- 时序⑧时为 phase_B 的下降沿触发

上述情况使用 phase_A 和 phase_B 都会进入上升沿触发中断和下降沿触发中断，所以在连续 2 次检测 Transition 值都为顺时针时才是真的顺时针旋转一格，在 Rotary Encoder Switch 的 Count 值才加1；反之连续2次检测Transition值都为逆时针时才是真的逆时针旋转一格，在Rotary Encoder Switch 的 Count 值才减 1。

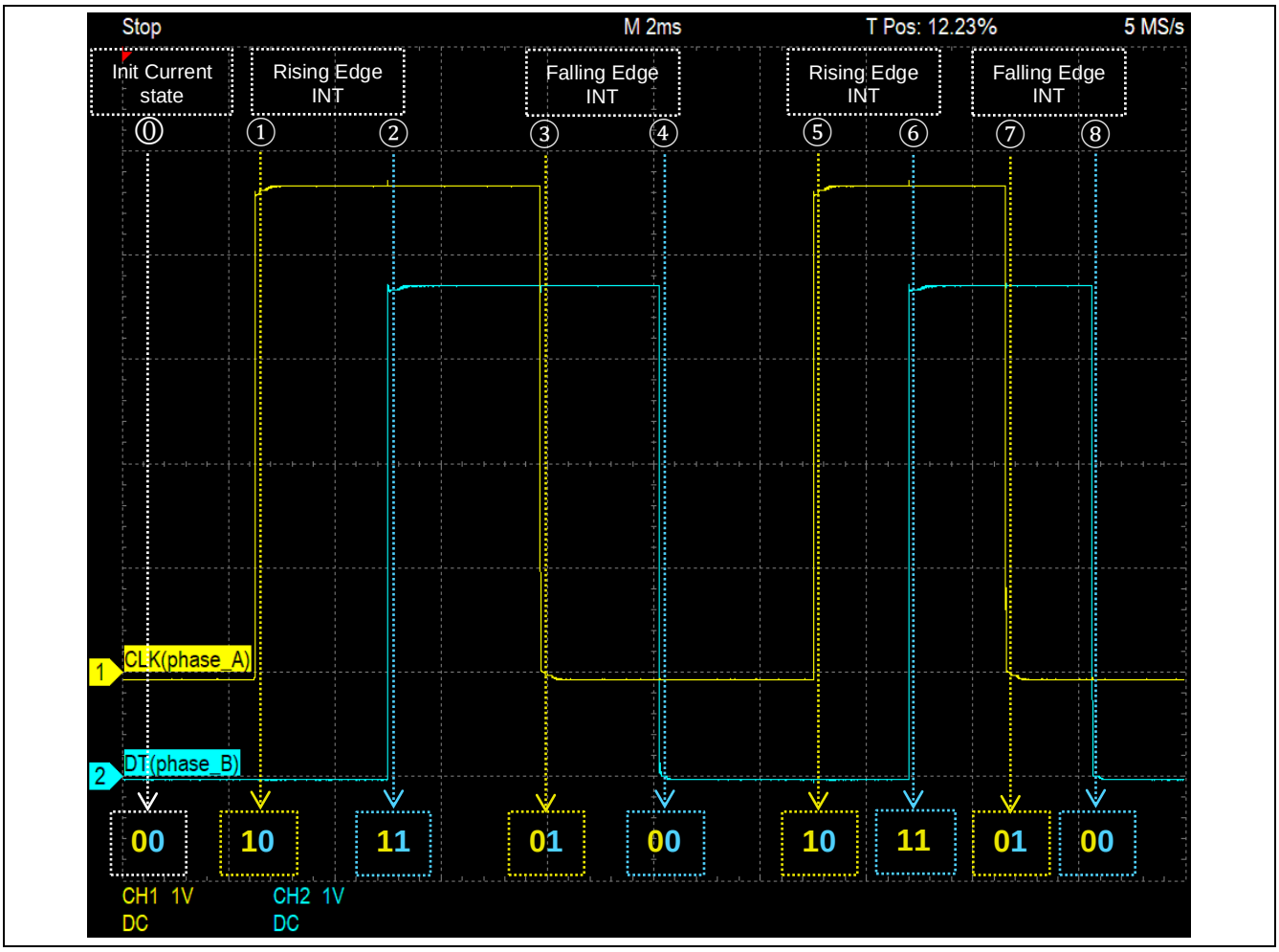


图 1-10 Rotary Encoder Switch 顺时针旋转

Current state = (phase_A << 1) | phase_B,  Transition = (Last state << 2) | Current state

- 时序⓪时初始化 Current state=00(二进制)
- 时序①时 Transition=0010(二进制)，索引= 2(十进制) ，查表 1-2 为顺时针旋转→**“1”**
- 时序②时 Transition=1011(二进制) ，索引=11(十进制)，查表 1-2 为顺时针旋转→**“1”**

**→ 连续 2 次出现 “1” → 显示“ Clockwise(CW)  Count= 1”**

- 时序③时 Transition=1101(二进制)，索引=13(十进制)，查表 1-2 为顺时针旋转→**“1”**
- 时序④时 Transition=0100(二进制)，索引= 4(十进制) ，查表 1-2 为顺时针旋转→**“1”**

**→ 连续 2 次出现 “1” → 显示“ Clockwise(CW)  Count= 2”**

- 时序⑤时 Transition=0010(二进制)，索引= 2(十进制) ，查表 1-2 为顺时针旋转→“1”
- 时序⑥时 Transition=1011(二进制)，索引=11(十进制)，查表 1-2 为顺时针旋转→“1”

**→ 连续 2 次出现 “1” → 显示“ Clockwise(CW)  Count= 3”**

- 时序⑦时 Transition=1101(二进制)，索引=13(十进制)，查表 1-2 为顺时针旋转→“1”
- 时序⑧时 Transition=0100(二进制)，索引= 4(十进制) ，查表 1-2 为顺时针旋转→“1”

**→ 连续 2 次出现 “1” →显示“ Clockwise(CW)  Count= 4”**

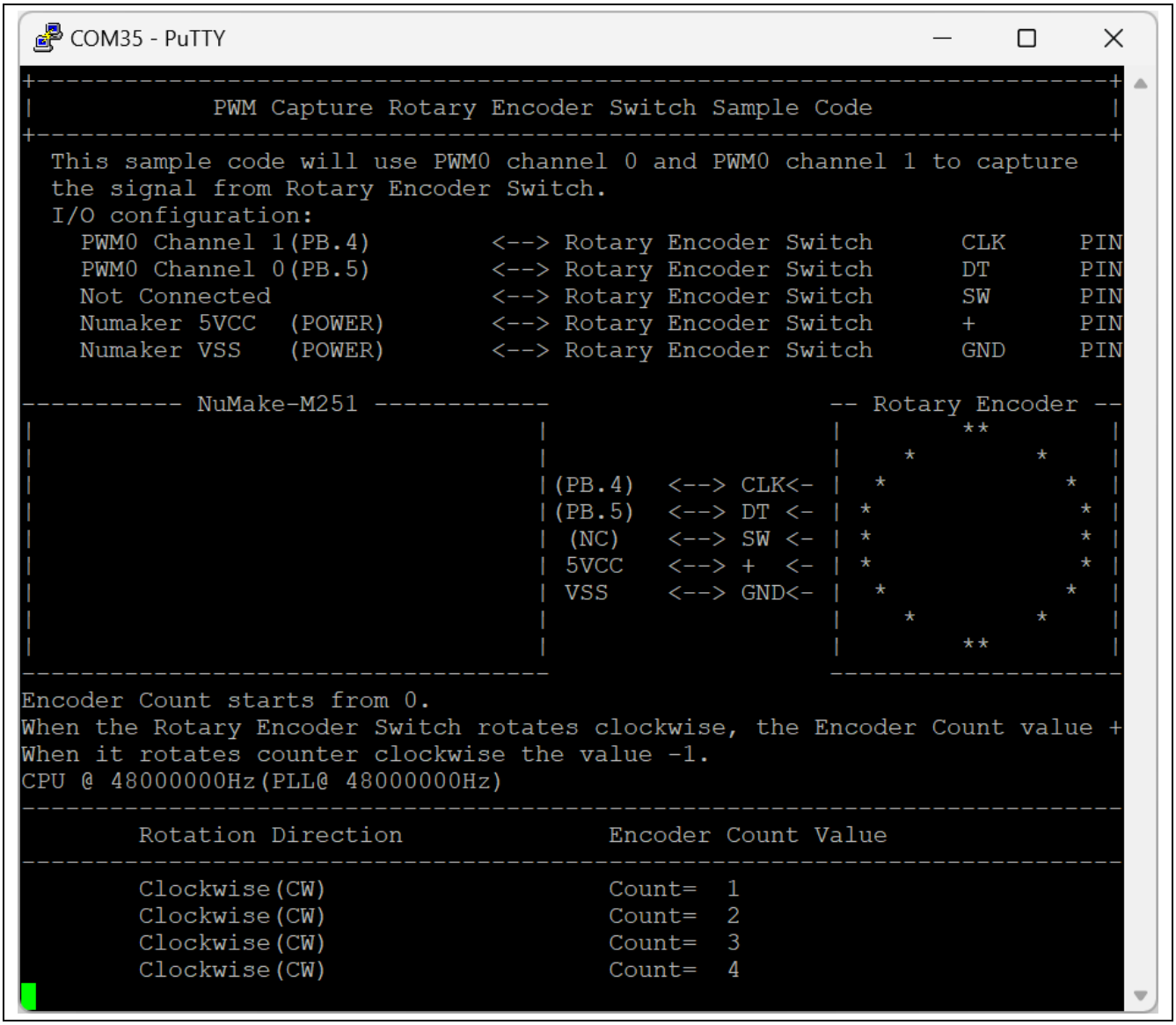


图 1-11 顺时针旋转执行结果

如图 1-10 Rotary Encoder Switch 顺时针旋转时序⓪→⑧执行结果为图 1-11 所示。

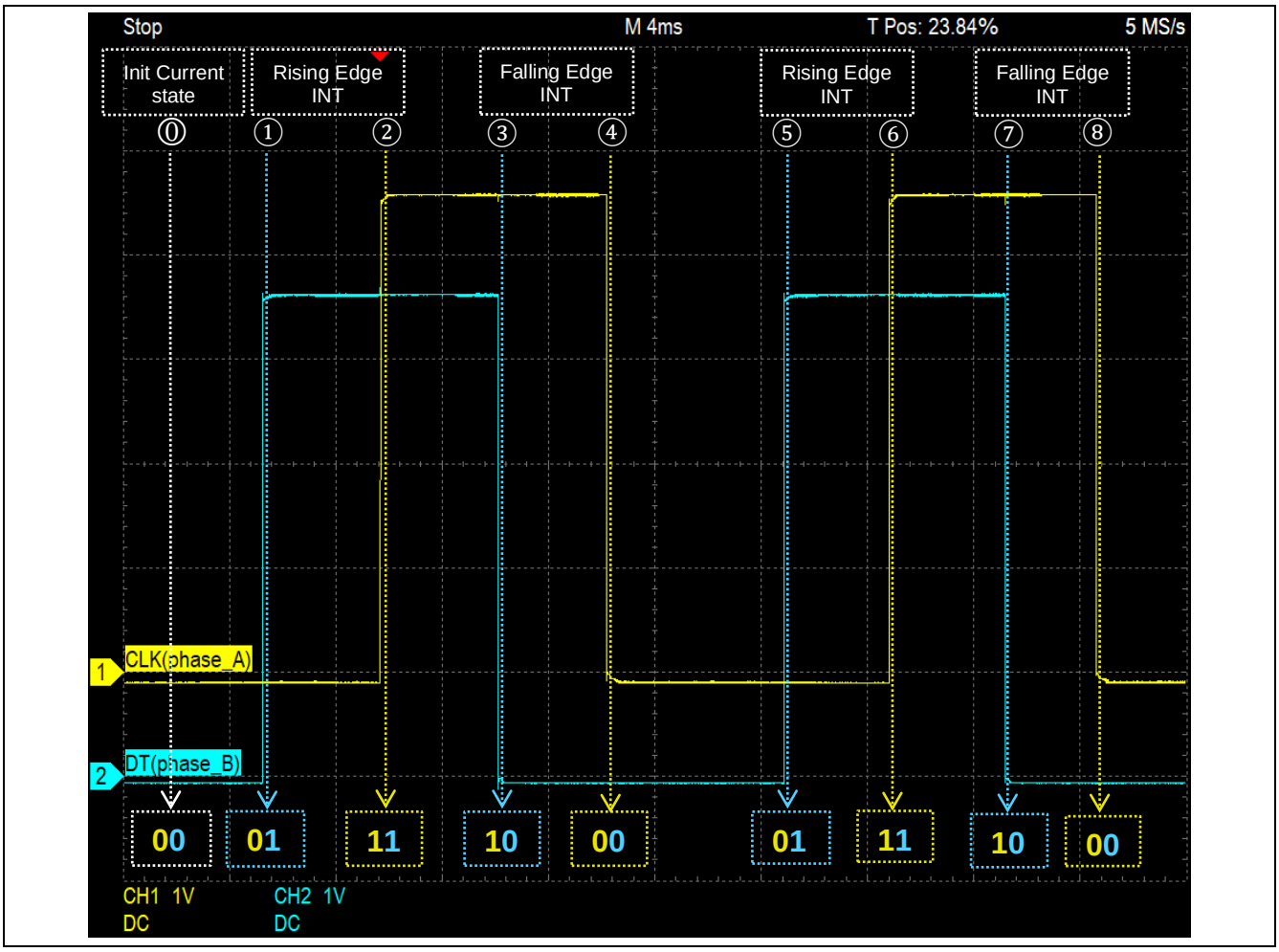


图 1-12 Rotary Encoder Switch 逆时针旋转

Current state = (phase_A << 1) | phase_B,  Transition = (Last state << 2) | Current state

- 时序⓪时初始化 Current state=00(二进制)
- 时序①时 Transition=0001(二进制)，索引= 1(十进制) ，查表 1-2 为逆时针旋转→“-1”
- 时序②时 Transition=0111(二进制)，索引= 7(十进制) ，查表 1-2 为逆时针旋转→“-1”

**→ 连续 2 次出现“-1” → 显示“ Counter-Clockwise(CCW)  Count= -1”**

- 时序③时 Transition=1110(二进制)，索引=14(十进制)，查表 1-2 为逆时针旋转→“-1”
- 时序④时 Transition=1000(二进制)，索引= 8(十进制) ，查表 1-2 为逆时针旋转→“-1”

**→ 连续 2 次出现“-1” → 显示“ Counter-Clockwise(CCW)  Count= -2”**

- 时序⑤时 Transition=0001(二进制)，索引= 1(十进制) ，查表 1-2 为逆时针旋转→“-1”
- 时序⑥时 Transition=0111(二进制)，索引= 7(十进制) ，查表 1-2 为逆时针旋转→“-1”

**→ 连续 2 次出现“-1” → 显示“ Counter-Clockwise(CCW)  Count= -3”**

- 时序⑦时 Transition=1110(二进制)，索引=14(十进制)，查表 1-2 为逆时针旋转→“-1”
- 时序⑧时 Transition=1000(二进制)，索引= 8(十进制) ，查表 1-2 为顺时针旋转→“-1”

**→ 连续 2 次出现“-1” → 显示“ Counter-Clockwise(CCW)  Count= -4”**

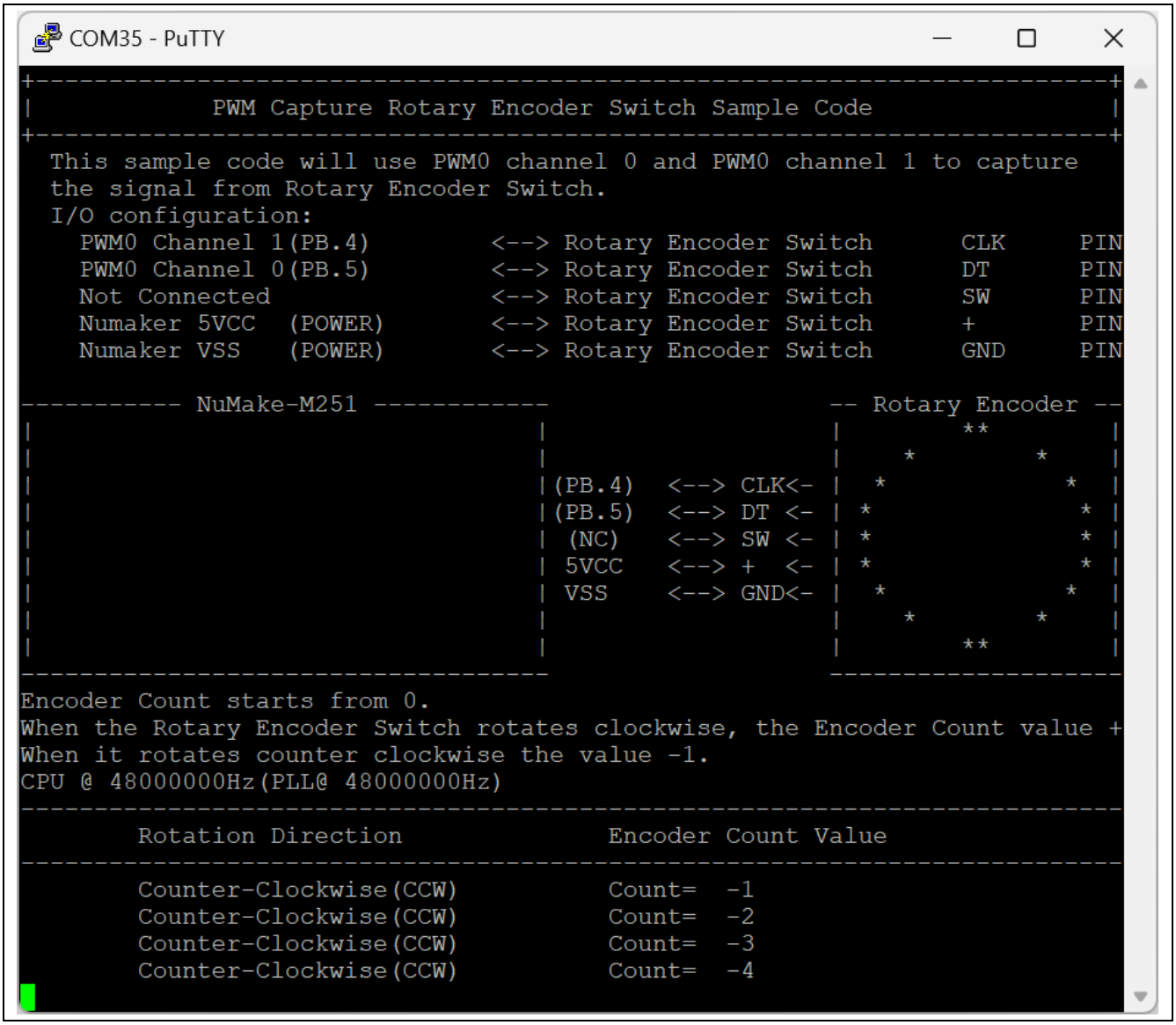


图 1-13 逆时针旋转执行结果

如图 1-12 Rotary Encoder Switch 逆时针旋转时序⓪→⑧执行结果为图 1-13 所示。

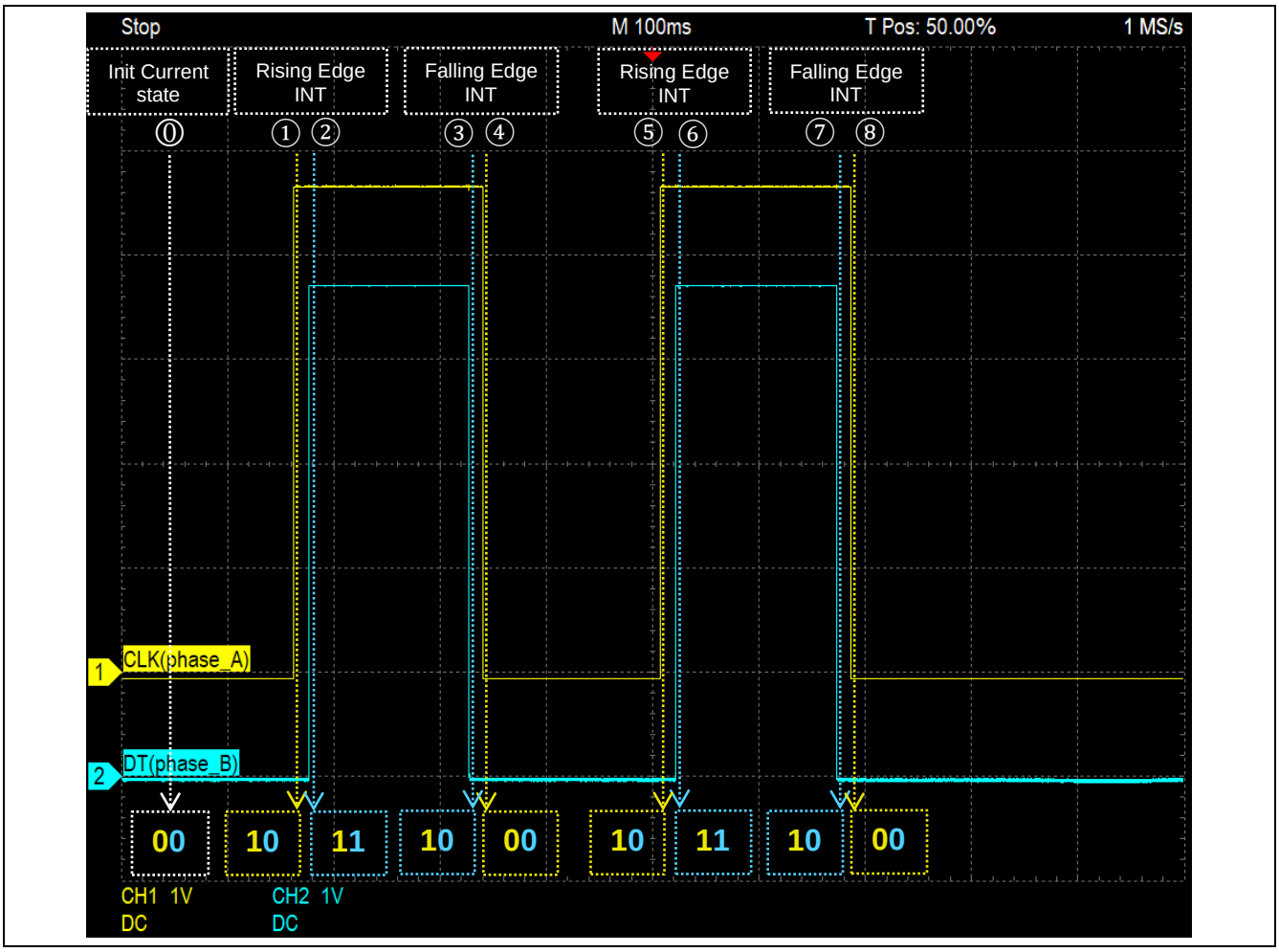


图 1-14 Rotary Encoder Switch 顺时针旋转与逆时针旋转交替

Current state = (phase_A << 1) | phase_B,  Transition = (Last state << 2) | Current state

- 时序⓪时初始化 Current state=00(二进制)
- 时序①时 Transition=0010(二进制)，索引= 2(十进制) ，查表 1-2 为顺时针旋转→“1”
- 时序②时 Transition=1011(二进制)，索引=11(十进制)，查表 1-2 为顺时针旋转→“1”

**→ 连续 2 次出现“1” → 显示“ Clockwise(CW)  Count= 1”**

- 时序③时 Transition=1110(二进制)，索引=14(十进制)，查表 1-2 为逆时针旋转→“-1”
- 时序④时 Transition=1000(二进制)，索引= 8(十进制) ，查表 1-2 为逆时针旋转→“-1”

**→ 连续 2 次出现“-1” → 显示“ Counter-Clockwise(CCW)  Count= 0”**

- 时序⑤时 Transition=0010(二进制)，索引= 2(十进制) ，查表 1-2 为顺时针旋转→“1”

- 时序⑥时 Transition=1011(二进制)，索引=11(十进制)，查表 1-2 为顺时针旋转→“1”

**→ 连续 2 次出现“1” → 显示“ Clockwise(CW)  Count= 1”**

- 时序⑦时 Transition=1110(二进制)，索引=14(十进制)，查表 1-2 为逆时针旋转→“-1”
- 时序⑧时 Transition=1000(二进制)，索引= 8(十进制) ，查表 1-2 为逆时针旋转→“-1”

**→ 连续 2 次出现“-1” → 显示“ Counter-Clockwise(CCW)  Count= 0”**

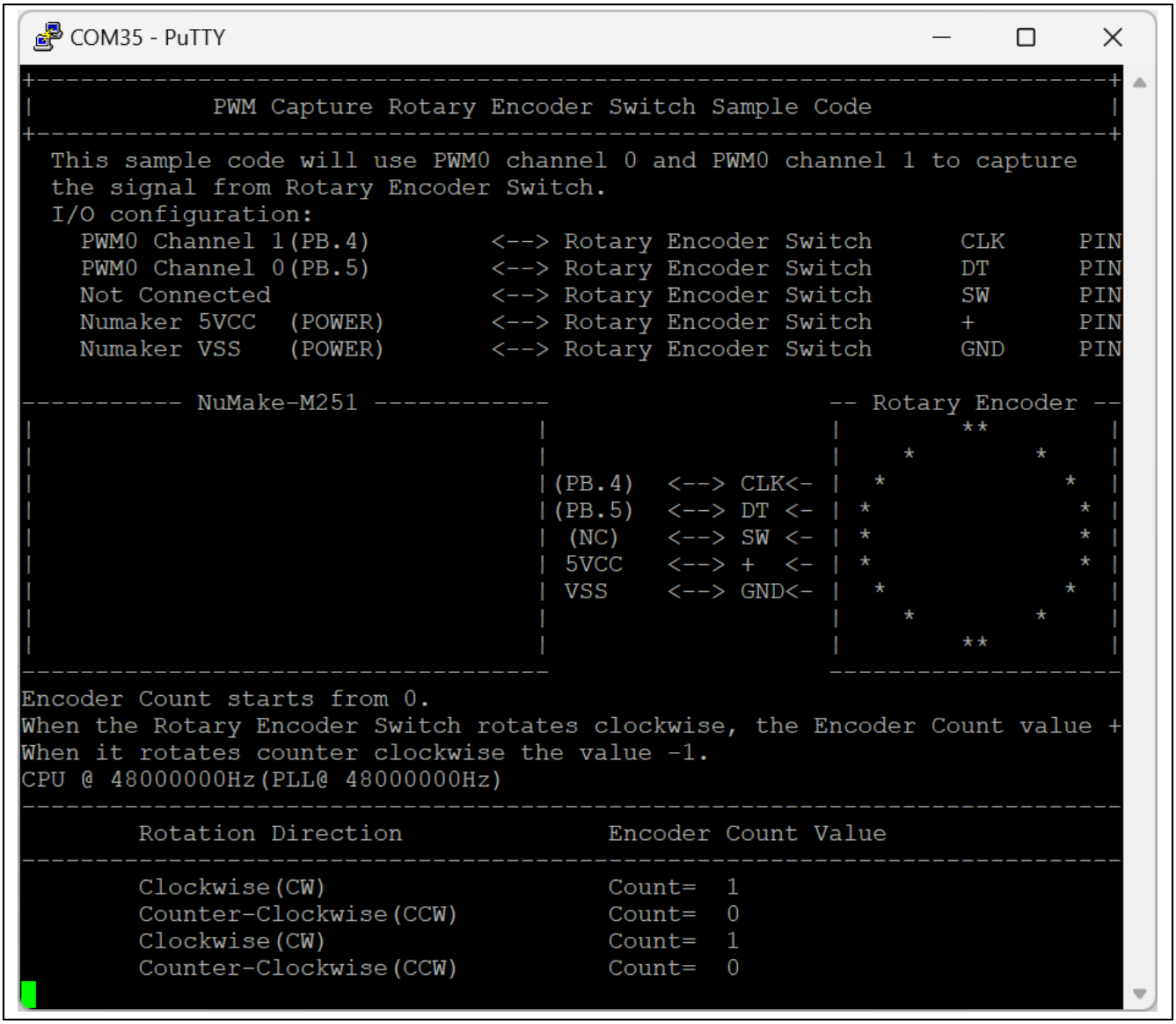


图 1-15 顺时针旋转与逆时针旋转交替执行结果

如图 1-14 Rotary Encoder Switch顺时针旋转与逆时针旋转交替时序⓪→⑧执行结果为图 1-15所示。

## 2. 程序代码介绍

代码工作流程如下：

1. main.c 中初始 M251/M252/M254/M256/M258 系列相关外设，包含 HIRC、UART0、PWM0，并在 main.c 主循环中去判断 Rotary Enconder Switch 是否有旋转。
2. 使用 PWM0 CH0 和 CH1 的 Capture 中断检测 Rotary Enconder Switch A/B 相位的上升沿/下降沿中断。
3. 通过状态表判断方向变化，并累加变化量以滤除抖动干扰。
4. 每次稳定的增减 2 次变化才判定为一次有效旋转，并记录事件。
5. 使用循环队列 event_queue 存储方向和计数信息。

在 while(1){...} 主循环中，M251/M252 会去从事件队列中取出 1 笔 Rotary Enconder Switch 数据并显示方向与计数值，用来实时显示旋转方向与对应的计数。

```
int main(void)
{
    /* Initialize system and clocks */
    SYS_Init();

      /* Initialize UART for printing */
    UART0_Init();

    /* Initialize PWM capture for encoder */
    PWM0_Capture_Init();

    EncoderEvent evt;
    printf("+-----------------------------------------------------------------------------+\n");
    printf("|            PWM Capture Rotary Encoder Switch Sample Code                    |\n");
    printf("+-----------------------------------------------------------------------------+\n");
    ...
    printf("-------------------------------------------------------------------------------\n");
    printf("\tRotation Direction\t\tEncoder Count Value\n");
    printf("-------------------------------------------------------------------------------\n");
    while (1)
    {
        /* If there is an event in the queue, print it */
        if (dequeue_event(&evt))
        {
            if (evt.Direction == 1)
            {
                printf("\tClockwise(CW)\t\t\tCount=\t%d\n", evt.Count);
            }
            else if (evt.Direction == -1)
            {
                printf("\tCounter-Clockwise(CCW)\t\tCount=\t%d\n", evt.Count);
            }
        }
    }
}
```

PWM0 channel 0 and channel 1 判断上升沿触发中断和下降沿触发中断，进入中断后先初始 phases A 和 phases B 状态，然后将当前状态 current_state 记录下来(2 Bits 值)，下一个中断进来后将目前 current_state 和刚刚上一个中断记录起来 current_state 结合在一起为 4 Bits 值(transition)，将进入中断得到的 transition 值去查表(transition_table)便可知旋转方向，连续 2 次中断进入后判断为旋转同方向才为此旋转方向。

```
void PWM0_P0_IRQHandler(void)
{
    /* Get capture interrupt flags for channel 0 and channel 1 */
    uint32_t flag_state_ch0 = PWM_GetCaptureIntFlag(PWM0, 0);
    uint32_t flag_state_ch1 = PWM_GetCaptureIntFlag(PWM0, 1);

    /* Check if channel 0 has a rising edge capture interrupt */
    if (flag_state_ch0 & 0x01)
    {
        /* Handle encoder event on channel 0 rising edge */
        HandleEncoderEvent();

        /* Clear channel 0 rising edge capture interrupt flag */
        PWM_ClearCaptureIntFlag(PWM0, 0, PWM_CAPTURE_INT_RISING_LATCH);
    }
    /* Check if channel 0 has a falling edge capture interrupt */
    if (flag_state_ch0 & 0x02)
    {
        /* Handle encoder event on channel 0 falling edge */
        HandleEncoderEvent();

        /* Clear channel 0 falling edge capture interrupt flag */
        PWM_ClearCaptureIntFlag(PWM0, 0, PWM_CAPTURE_INT_FALLING_LATCH);
    }
    /* Check if channel 1 has a rising edge capture interrupt */
    if (flag_state_ch1 & 0x01)
    {
        /* Handle encoder event on channel 1 rising edge */
        HandleEncoderEvent();

        /* Clear channel 1 rising edge capture interrupt flag */
        PWM_ClearCaptureIntFlag(PWM0, 1, PWM_CAPTURE_INT_RISING_LATCH);
    }
    /* Check if channel 1 has a falling edge capture interrupt */
    if (flag_state_ch1 & 0x02)
    {
        /* Handle encoder event on channel 1 falling edge */
        HandleEncoderEvent();

        /* Clear channel 1 falling edge capture interrupt flag */
        PWM_ClearCaptureIntFlag(PWM0, 1, PWM_CAPTURE_INT_FALLING_LATCH);
    }
}

void HandleEncoderEvent()
{
    /* Read current states of encoder phases A and B from pins */
    uint8_t phase_A = PB4 & 1;
    uint8_t phase_B = PB5 & 1;

    /* Encode current state */
    uint8_t current_state = (phase_A << 1) | phase_B;

     /* Form 4-bit transition key from last state and current state */
```

```
    uint8_t transition = (last_state << 2) | current_state;

    /* Lookup direction using transition table */
    int8_t dir = transition_table[transition];

    /* If movement is valid */
    if (dir != 0)
    {
      transition_sum += dir;
      if (transition_sum >= 2)
      {
        /* Increase count on clockwise rotation */
        encoder_count++;
        enqueue_event(1, encoder_count);
        transition_sum = 0;
      }
      else if (transition_sum <= -2)
      {
        /* Decrease count on counter-clockwise rotation */
        encoder_count--;
        enqueue_event(-1, encoder_count);
        transition_sum = 0;
      }
    }
    /* Update last state */
    last_state = current_state;
}
```

transition_table 为判断 Rotary Encoder Switch 旋转方向有效性。

```
const int8_t transition_table[16] =
{
     0,  /*  0: 00 -> 00 (no change) */
    -1,  /*  1: 00 -> 01 (CCW)       */
     1,  /*  2: 00 -> 10 (CW)        */
     0,  /*  3: 00 -> 11 (invalid)   */

     1,  /*  4: 01 -> 00 (CW)        */
     0,  /*  5: 01 -> 01 (no change) */
     0,  /*  6: 01 -> 10 (invalid)   */
    -1,  /*  7: 01 -> 11 (CCW)       */

    -1,  /*  8: 10 -> 00 (CCW)       */
     0,  /*  9: 10 -> 01 (invalid)   */
     0,  /* 10: 10 -> 10 (no change) */
     1,  /* 11: 10 -> 11 (CW)        */

     0,  /* 12: 11 -> 00 (invalid)   */
     1,  /* 13: 11 -> 01 (CW)        */
    -1,  /* 14: 11 -> 10 (CCW)       */
     0   /* 15: 11 -> 11 (no change) */
};
```

PWM0_Capture_Init 设定 PWM0 使用 PIN，如使用 PB4 和 PB5 作为检测 Rotary Enconder Switch 产生的脉波，并初始化 PB4 和 PB5 的值。

```
void PWM0_Capture_Init(void)
{
    /* Enable PWM0 module clock */
    CLK_EnableModuleClock(PWM0_MODULE);

    /* Set PWM0 clock source to PCLK0 with no divider */
    CLK_SetModuleClock(PWM0_MODULE, CLK_CLKSEL2_PWM0SEL_PCLK0, 0);

    /* Set multi-function pins PB.4 and PB.5 for PWM0 CH1 and CH0 */
    SYS->GPB_MFPL &= ~(SYS_GPB_MFPL_PB4MFP_Msk | SYS_GPB_MFPL_PB5MFP_Msk);
    SYS->GPB_MFPL |= (SYS_GPB_MFPL_PB4MFP_PWM0_CH1 | SYS_GPB_MFPL_PB5MFP_PWM0_CH0);

    /* Configure PWM0 channel 0 as capture input with 1000 unit time and 0 prescaler */
    PWM_ConfigCaptureChannel(PWM0, 0, 1000, 0);

    /* Enable capture function and interrupts for channel 0 */
    PWM_EnableCapture(PWM0, PWM_CH_0_MASK);
    PWM_EnableCaptureInt(PWM0, 0, PWM_CAPTURE_INT_RISING_LATCH |
PWM_CAPTURE_INT_FALLING_LATCH);

    /* Configure PWM0 channel 1 as capture input with 1000 unit time and 0 prescaler */
    PWM_ConfigCaptureChannel(PWM0, 1, 1000, 0);

    /* Enable capture function and interrupts for channel 1 */
    PWM_EnableCapture(PWM0, PWM_CH_1_MASK);
    PWM_EnableCaptureInt(PWM0, 1, PWM_CAPTURE_INT_RISING_LATCH |
PWM_CAPTURE_INT_FALLING_LATCH);

    /* Enable PWM0 interrupt in NVIC */
    NVIC_EnableIRQ(PWM0_P0_IRQn);

     /* Set the interrupt priority for PWM0_P0_IRQn to level 1
     (lower number means higher priority) */
    NVIC_SetPriority(PWM0_P0_IRQn, 1);

    /* Initialize last_state to the current state of the encoder */
    last_state = (PB4 ? 1 : 0) << 1 | (PB5 ? 1 : 0);
}
```

使用 EncoderEvent struct 记录旋转编码器（Rotary Enconder Switch）事件的数据，记录旋转的方向和计数值。

```
typedef struct
{
    volatile int8_t Direction;              /* Direction of rotation */
    volatile int Count;                     /* Encoder count */
} EncoderEvent;
```

旋转编码器的事件（旋转方向与计数值）加入到事件队列（queue）中。

```
void enqueue_event(int8_t Direction, int Count)
{
    uint8_t next_tail = (queue_tail + 1) % QUEUE_SIZE;
    if (next_tail != queue_head)
    {
        event_queue[queue_tail].Direction = Direction;
        event_queue[queue_tail].Count = Count;
        queue_tail = next_tail;
    }
}
```

从队列中取出 1 笔 Rotary Enconder Switch 事件资料（旋转方向与计数值）。

```
int dequeue_event(EncoderEvent* event)
{
    if (queue_head == queue_tail)
        return 0;
    *event = event_queue[queue_head];
    queue_head = (queue_head + 1) % QUEUE_SIZE;
    return 1;
}
```

# 3. 软件与硬件需求

## 3.1 软件需求

- BSP 版本
  - M251_M252_M254_M256_M258_Series_BSP_CMSIS_V3.02.007
- IDE 版本
  - Keil uVision V5.38.0.0

## 3.2 硬件需求

- 电路组件
  - NuMaker-M252SD V1.1
- 线路示意图及 Rotary Encoder 接法
  - 将 NuMaker-M252SD 的相关 PIN 与 Rotary Encoder Switch 相关 PIN 对接如下:

| | | | |
|---|---|---|---|
| PWM0 Channel 1(PB.4) | ↔ Rotary Encoder Switch | CLK | PIN |
| PWM0 Channel 0(PB.5) | ↔ Rotary Encoder Switch | DT | PIN |
| Not Connected | ↔ Rotary Encoder Switch | SW | PIN |
| Numaker 5VCC (POWER) | ↔ Rotary Encoder Switch | + | PIN |
| Numaker VSS (POWER) | ↔ Rotary Encoder Switch | GND | PIN |

  - 将 NuMaker-M252SD 中 Nu-Link2-Me 的 USB 与 PC USB 对接，以显示范例程序代码的执行结果。

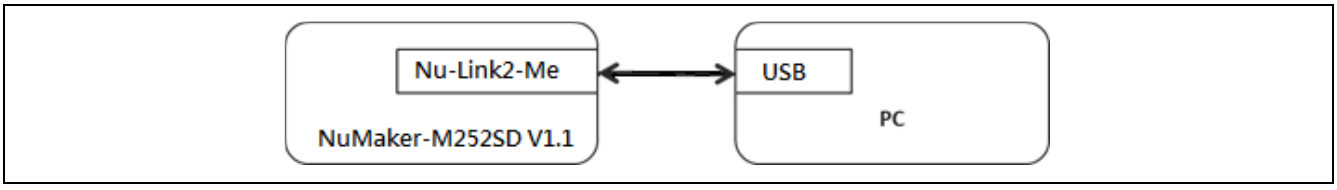


图 3-1 线路示意图

# 4. 目录信息

目录结构如下图所示。

| | |
|---|---|
| EC_M251_M252_PWM_Capture_Rotary_Encoder_V1.00 | |
| Library | Sample code header and source files |
| CMSIS | Cortex® Microcontroller Software Interface Standard (CMSIS) by Arm® Corp. |
| Device | CMSIS compliant device header file |
| StdDriver | All peripheral driver header and source files |
| SampleCode | |
| ExampleCode | |
| M251_M252_PWM_Capture_Rotary_Encoder | Source file of example code |

图 4-1 目录信息

# 5. 范例程序执行

1. 根据目录信息章节进入 ExampleCode 路径中的 KEIL 文件夹，双击 M251_M252_PWM_Capture_Rotary_Encoder.uvprojx。
2. 进入编译模式接口
   - 编译
   - 下载程序代码至内存
   - 进入 / 离开除错模式
3. 进入除错模式接口
   - 执行程序代码

# 6. 修订纪录

| Date | Revision | Description |
|---|---|---|
| 2025.06.16 | 1.00 | 初始发布。 |

## 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.**

*Please note that all data and specifications are subject to change without notice.*
*All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.*