M251/M252 PWM Capture Rotary

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	This example code uses the M251/M252 PWM Capture to detect the rotation direction and count of a Rotary Encoder Switch.
BSP Version	M251/M252/M254/M256/M258 Series BSP CMSIS V3.02.007
Hardware	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. Overview

This sample code uses the M251/M252 series to capture the A and B phase square wave signals generated by a rotary encoder switch. By determining the relative position of phases A and B, it identifies the rotation direction and records the rotation position, then outputs the captured results via UART.

1.1 Principle

PWM capture primarily operates through interrupt configuration. The interrupt can be triggered by either rising edge or falling edge to measure the period, frequency, and duty cycle of an external signal. Each channel can record the most recent counter value when triggered, including the counter values at the rising edge and falling edge events.

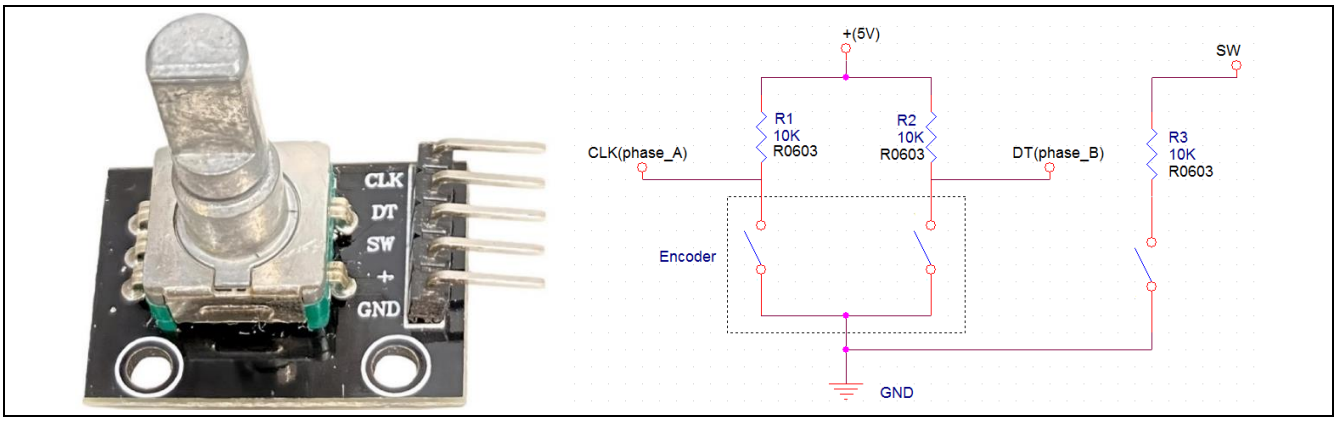

Figure 1-1 Rotary Encoder Switch

Figure 1-1 shows the physical component of the Rotary Encoder Switch on the left, and the circuit diagram of the Rotary Encoder Switch on the right. It has five pins: “GND”, “+”, “SW”, “DT”, and “CLK”. The wiring is as follows: “GND” is connected to VSS in the circuit, “+”, is connected to DC 5V, and “SW” is a push-button switch function which is not used in this example. “DT” and “CLK” generate square waves with different timing sequences.

When rotated clockwise, the square wave generated by “CLK” leads the one generated by “DT”, as shown in Figure 1-2 and Figure 1-3, which illustrate edge detection starting from the rising edge and falling edge respectively. Conversely, when rotated counterclockwise, the square wave generated by “CLK” lags that of “DT”, as shown in Figure 1-4 and Figure 1-5, also corresponding to rising and falling edge detection.

In this example, one full rotation of the Rotary Encoder Switch results in 15 square waves each on “DT” and “CLK”, as shown in Figure 1-6 and Figure 1-7.

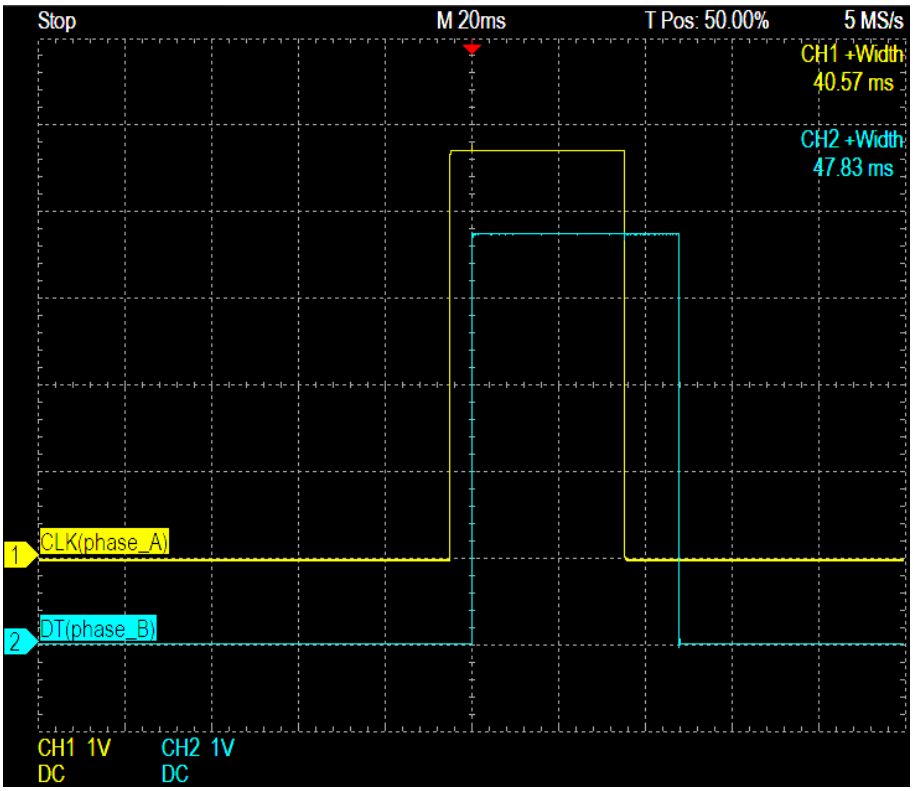

Figure 1-2 Rotary Encoder Switch Clockwise Rotation (Initial Rising Edge Trigger)

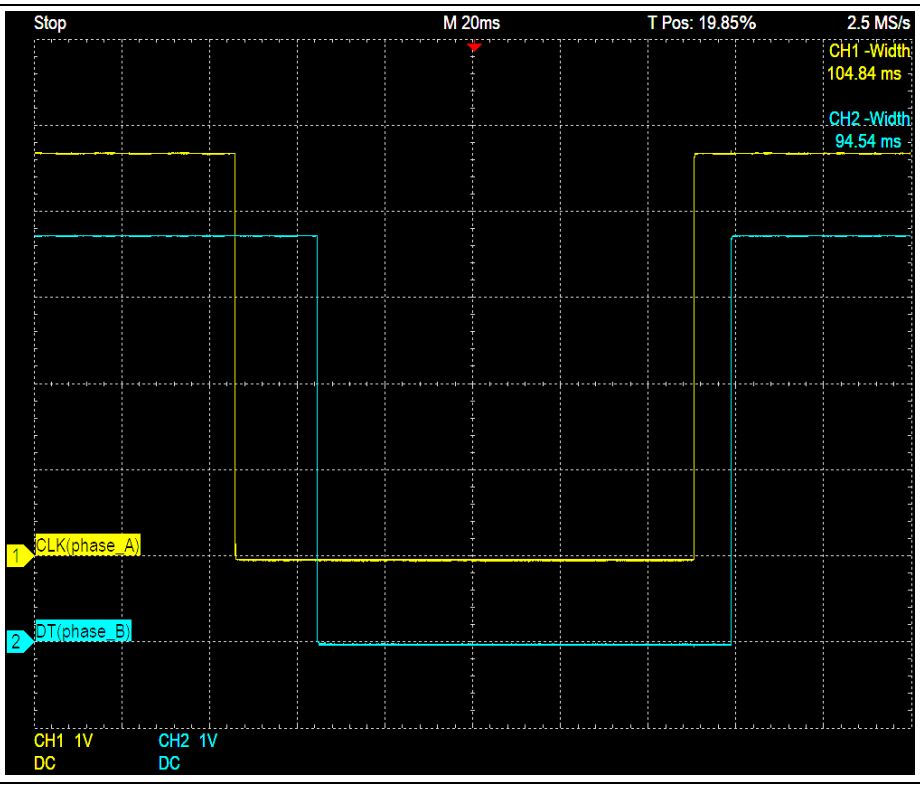

Figure 1-3 Rotary Encoder Switch Clockwise Rotation (Initial Falling Edge Trigger)

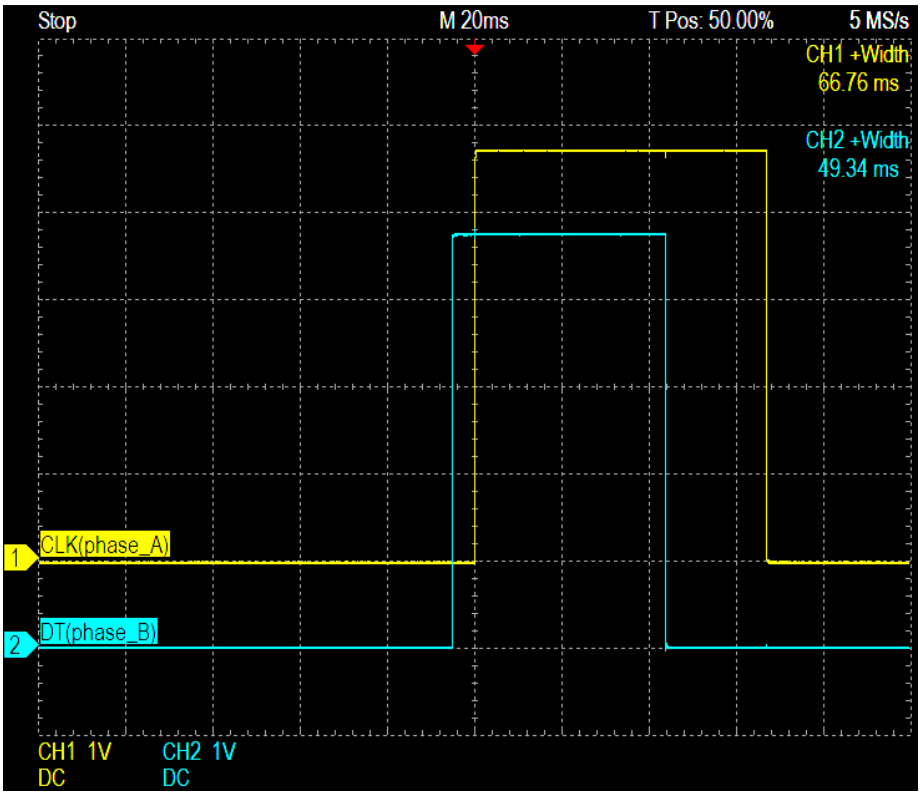

Figure 1-4 Rotary Encoder Switch Counterclockwise Rotation (Initial Rising Edge Trigger)

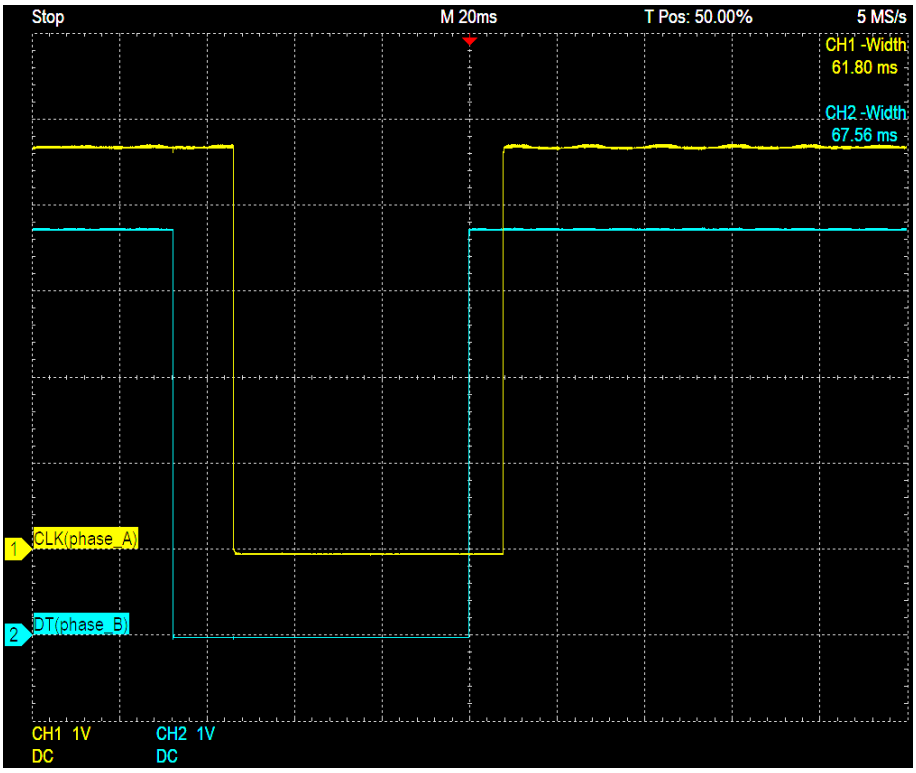

Figure 1-5 Rotary Encoder Switch Counterclockwise Rotation (Initial Falling Edge Trigger)

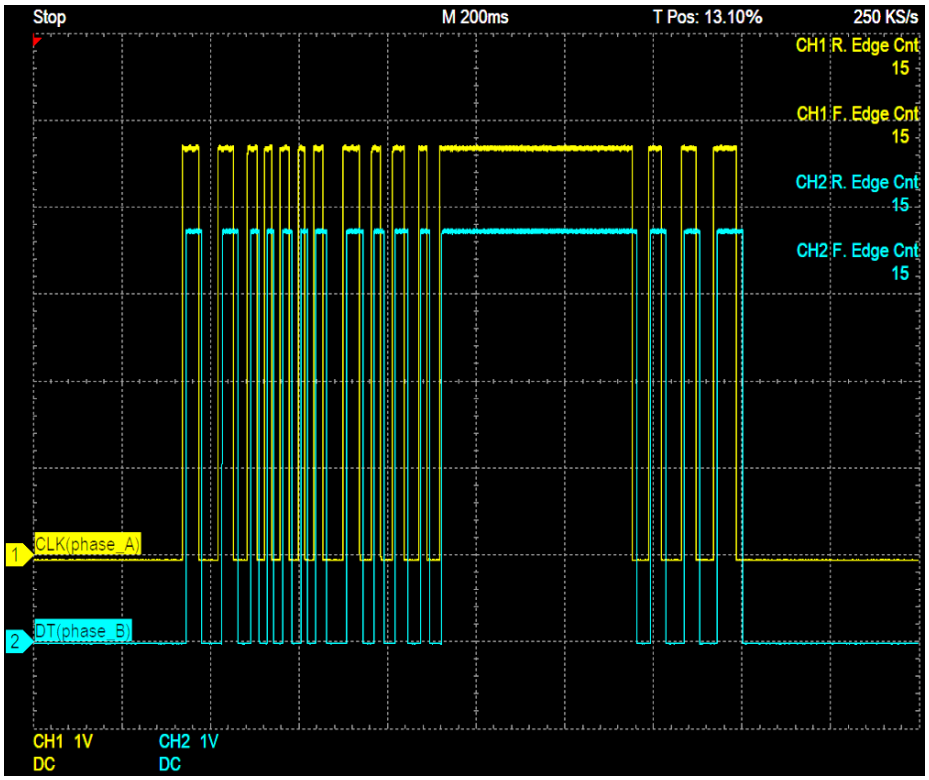

Figure 1-6 Rotary Encoder Switch Clockwise Rotation (One Full Turn)

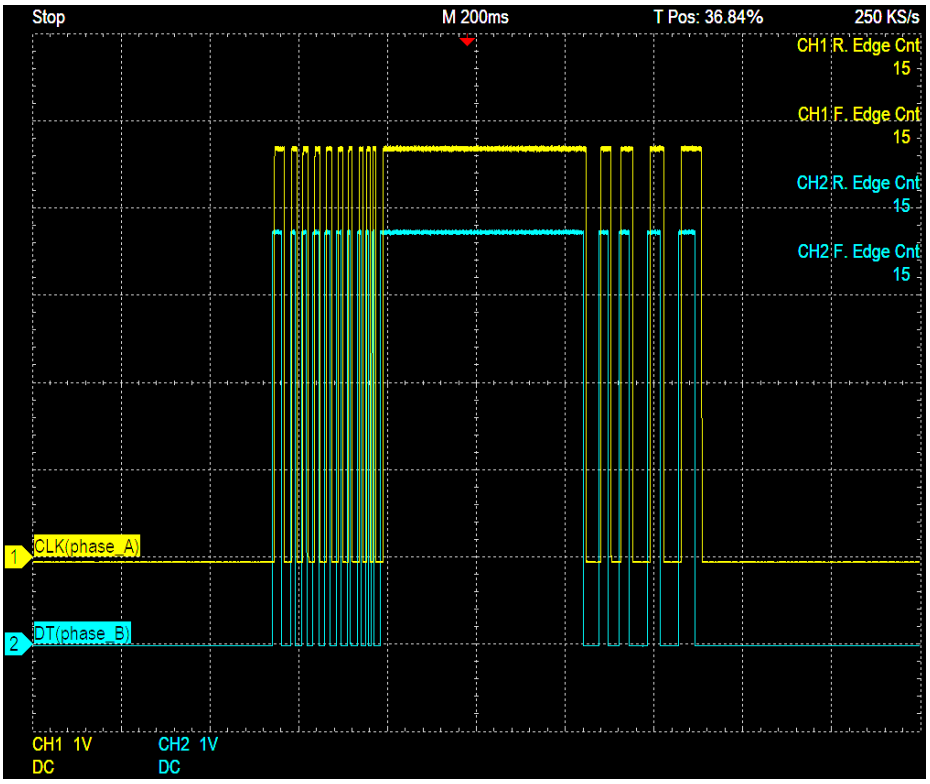

Figure 1-7 Rotary Encoder Switch Counterclockwise Rotation (One Full Turn)

Because the Rotary Encoder Switch is a mechanical switch, the generated square wave CLK (phase_A) and DT (phase_B) differ. As shown in Figure 1-2, the positive pulse width of CLK (phase_A) is 40.57 ms, while the positive pulse width of DT (phase_B) is 47.83 ms; as shown in Figure 1-3, the negative pulse width of CLK (phase_A) is 104.84 ms, and the negative pulse width of DT (phase_B) is 94.54 ms. From these observations, it can be seen that the timing difference between the CLK (phase_A) and DT (phase_B) square wave generated during the same rotation is quite large, making it unsuitable to use the PWM Capture's built-in CAPDATn values (rising edge capture count RCAPDATn and falling edge capture count FCAPDATn) to determine whether the square wave leads or lags.

Therefore, this example uses PWM Capture to handle rising edge interrupts and falling edge interrupts generated by CLK (phase_A) and DT (phase_B), combined with a rotary encoder direction lookup table (quadrature decoder). It records the previous states of CLK (phase_A) and DT (phase_B). The current state is calculated as Current state = (phase_A << 1) | phase_B; In other words, the phase_A status is treated as the MSB, and the phase_B status as the LSB, forming a 2-bits data representing the current state. Figure 1-8 and Figure 1-9 show the current state values generated during clockwise and counterclockwise rotation, respectively.

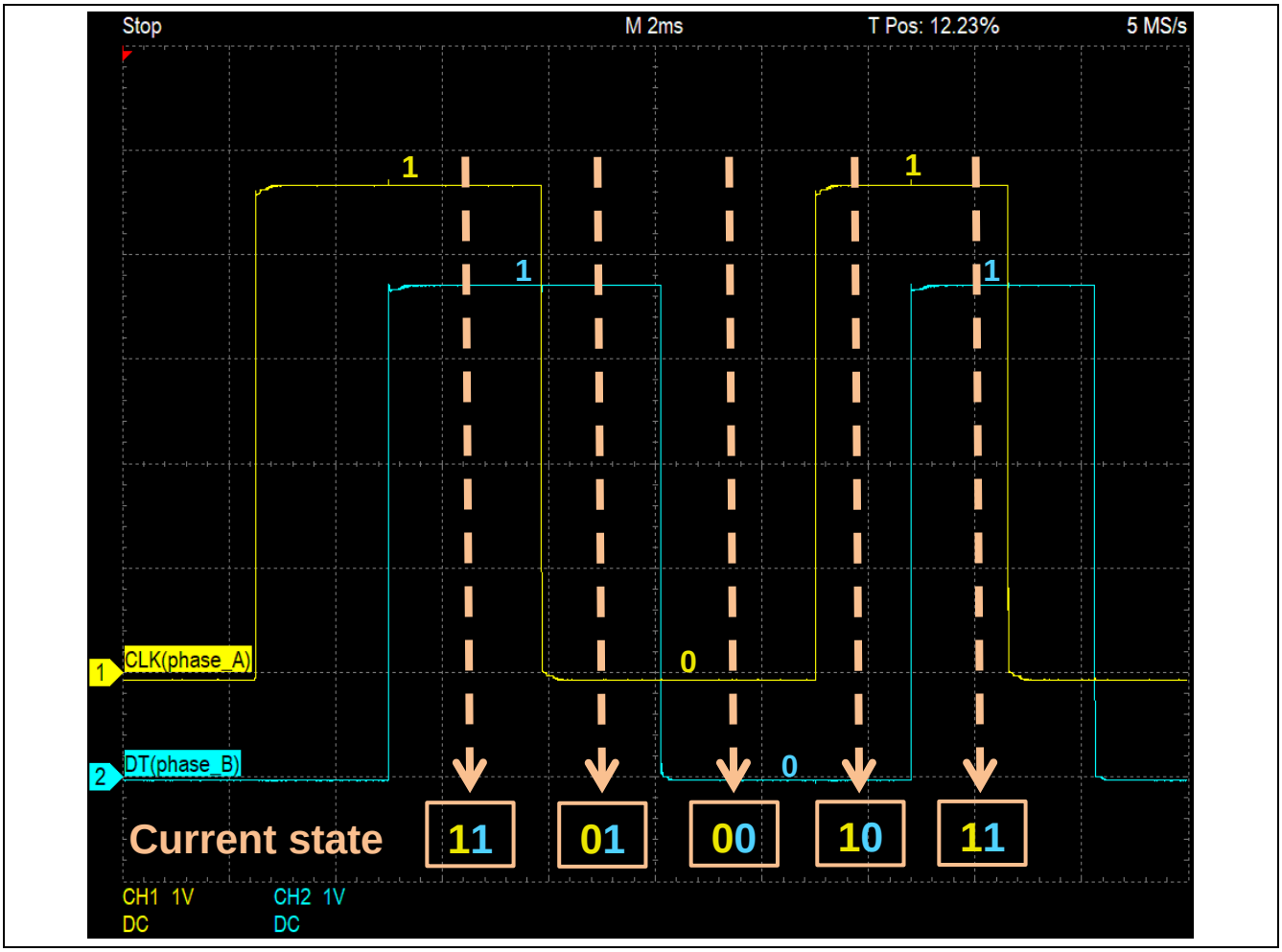

Figure 1-8 Rotary Encoder Switch Clockwise Rotation

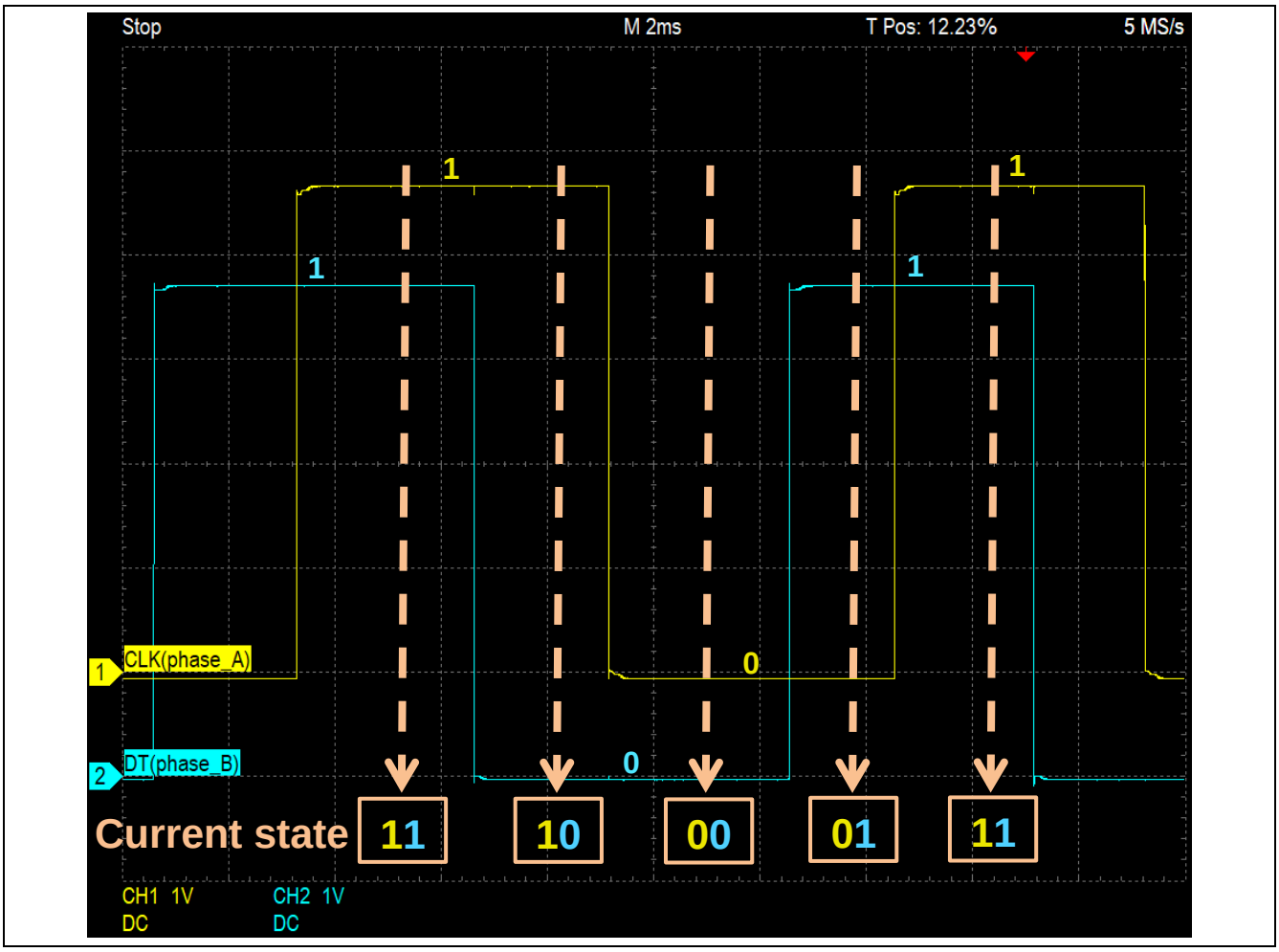

Figure 1-9 Rotary Encoder Switch Counterclockwise Rotation

Clockwise rotation of the current state (Current state) follows four sequences: 11→01, 01→00, 00→10, 10→11. Similarly, counterclockwise rotation also has four sequences: 11→10, 10→00, 00→01, 01→11. In total, there are 8 transition patterns for both clockwise and counter-clockwise rotation. By combining the current state (Current state) and the previous state (Last state), we can create a reference for a rotation state lookup table, as shown in Table 1-1.The 2 bits of the previous state (Last state) are used as the most significant bits (MSB), and the 2 bits of the current state (Current state) are used as the least significant bits (LSB). That is, the transition value can be calculated as: Transition = (Last state << 2) | Current state.

Clockwise rotation (CW)	Counterclockwise rotation (CCW)
1101	1110
0100	1000
0010	0001
1011	0111

Table 1-1 Previous State and Current State for Clockwise and Counterclockwise Rotation

Index (Decimal)	Previous State | Current State Transition = (Last state << 2) | Current state (Binary)	Valid/ Invalid	Rotation Direction (CW/CCW)	0= No change / Valid 1= CW -1= CCW
0	0000	Invalid	No change	0
1	0001	Valid	CCW	-1
2	0010	Valid	CW	1
3	0011	Invalid	Invalid	0
4	0100	Valid	CW	1
5	0101	Invalid	Invalid	0
6	0110	Invalid	Invalid	0
7	0111	Valid	CCW	-1
8	1000	Valid	CCW	-1
9	1001	Invalid	Invalid	0
10	1010	Invalid	No change	0
11	1011	Valid	CW	1
12	1100	Invalid	Invalid	0
13	1101	Valid	CW	1
14	1110	Valid	CCW	-1
15	1111	Invalid	No change	0

Table 1-2 All Possible 4-Bit Combinations of Previous State and Current State

Table 1-2 All Possible 4-Bit Combinations of Previous State and Current State shows all possible 4-bit combinations of the previous state and current state. Among them, 8 are valid codes and the other 8 are invalid codes. Since the Rotary Encoder Switch is a mechanical switch, bouncing may occur during rotation. The noise caused by bouncing corresponds to the invalid codes in Table 1-2. By only recording valid codes, this method can be used to eliminate bounce noise. Finally, clockwise rotation(CW) is assigned a value of “1”, counterclockwise rotation(CCW) is assigned “-1”, and invalid outputs are assigned “0”.

1.2 Demo Result

This example code provides three types of execution results: the first is purely clockwise rotation, the second is purely counterclockwise rotation, and the third is alternating between clockwise and counterclockwise rotation.

In the program, phase_A and phase_B each use two types of trigger interrupts: rising edge interrupt and falling edge interrupt, as shown in Figure 1-10.

- At timing ① is the rising edge trigger of phase_A
- At timing ② is the rising edge trigger of phase_B
- At timing ③ is the falling edge trigger of phase_A
- At timing ④ is the falling edge trigger of phase_B

- At timing ⑤ is the rising edge trigger of phase_A
- At timing ⑥ is the rising edge trigger of phase_B
- At timing ⑦ is the falling edge trigger of phase_A
- At timing ⑧ is the falling edge trigger of phase_B

In the above scenario, both phase_A and phase_B will trigger both rising and falling edge interrupts. Therefore, only when two consecutive detected Transition values are clockwise can it be confirmed as one valid step of clockwise rotation, and the Rotary Encoder Switch count value is incremented by 1. Conversely, only when two consecutive detected Transition values are counterclockwise can it be confirmed as one valid step of counterclockwise rotation, and the Rotary Encoder Switch count value is decremented by 1.

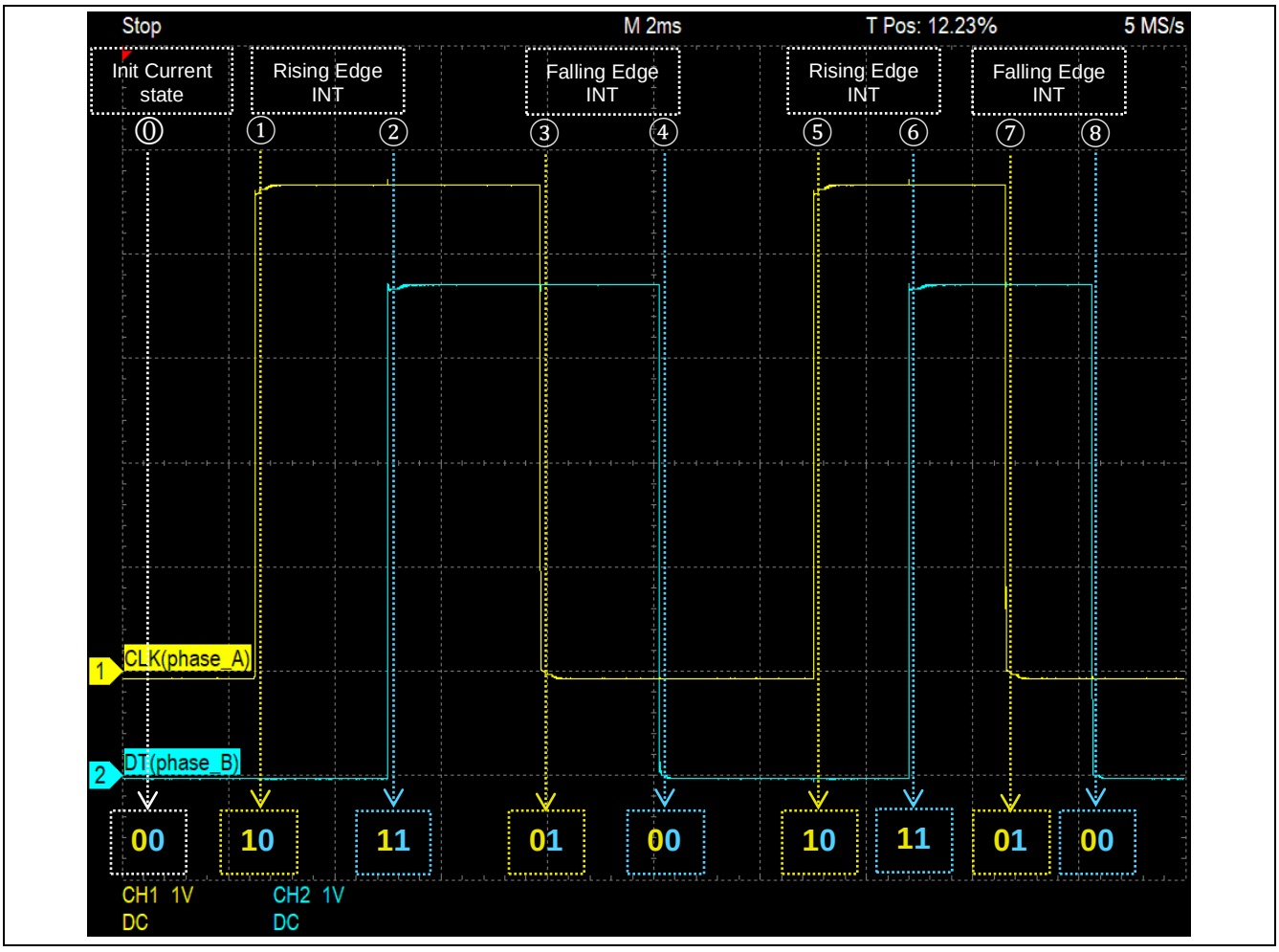

Figure 1-10 Rotary Encoder Switch Clockwise Rotation

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

- At timing⓪ the Current state is initialized to 00 (BIN).
- At timing①, Transition=0010(BIN), Index= 2(DEC), lookup in Table 1-2→ CW→ “1”
- At timing②, Transition=1011 (BIN), Index=11(DEC), lookup in Table 1-2→ CW → “1”

→Two consecutive "1" s detected →Print "Clockwise (CW) Count = 1"

- At timing③, Transition=1101(BIN), Index=13(DEC), lookup in Table 1-2→ CW → "1"
- At timing④, Transition=0100(BIN), Index= 4(DEC), lookup in Table 1-2→ CW → "1"

→Two consecutive "1" s detected →Print "Clockwise (CW) Count = 2"

- At timing⑤, Transition=0010(BIN), Index= 2(DEC), lookup in Table 1-2→ CW → "1"
- At timing⑥, Transition=1011(BIN), Index=11(DEC), lookup in Table 1-2→ CW → "1"

→ Two consecutive "1" s detected →Print "Clockwise (CW) Count = 3"

- At timing⑦, Transition=1101 (BIN), Index=13 (DEC), lookup in Table 1-2→ CW → "1"
- At timing⑧, Transition=0100 (BIN), Index= 4 (DEC), lookup in Table 1-2→ CW → "1"

→ Two consecutive "1" s detected →Print "Clockwise (CW) Count = 4"

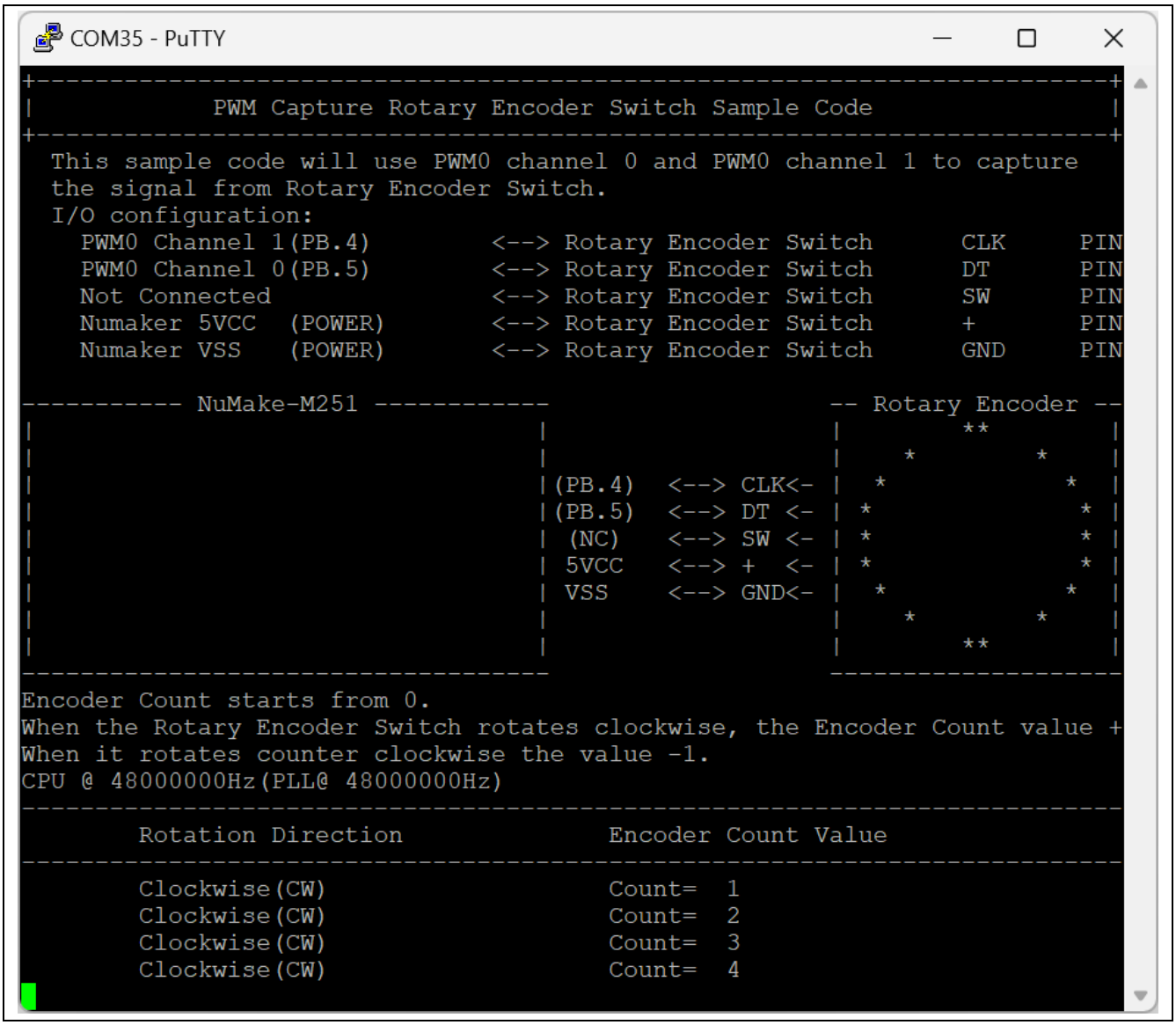

Figure 1-11 Clockwise Rotation Execution Result

Figure 1-10 Rotary Encoder Switch Clockwise Rotation Sequence ⓪→⑧ Execution Result is Shown in Figure 1-11.

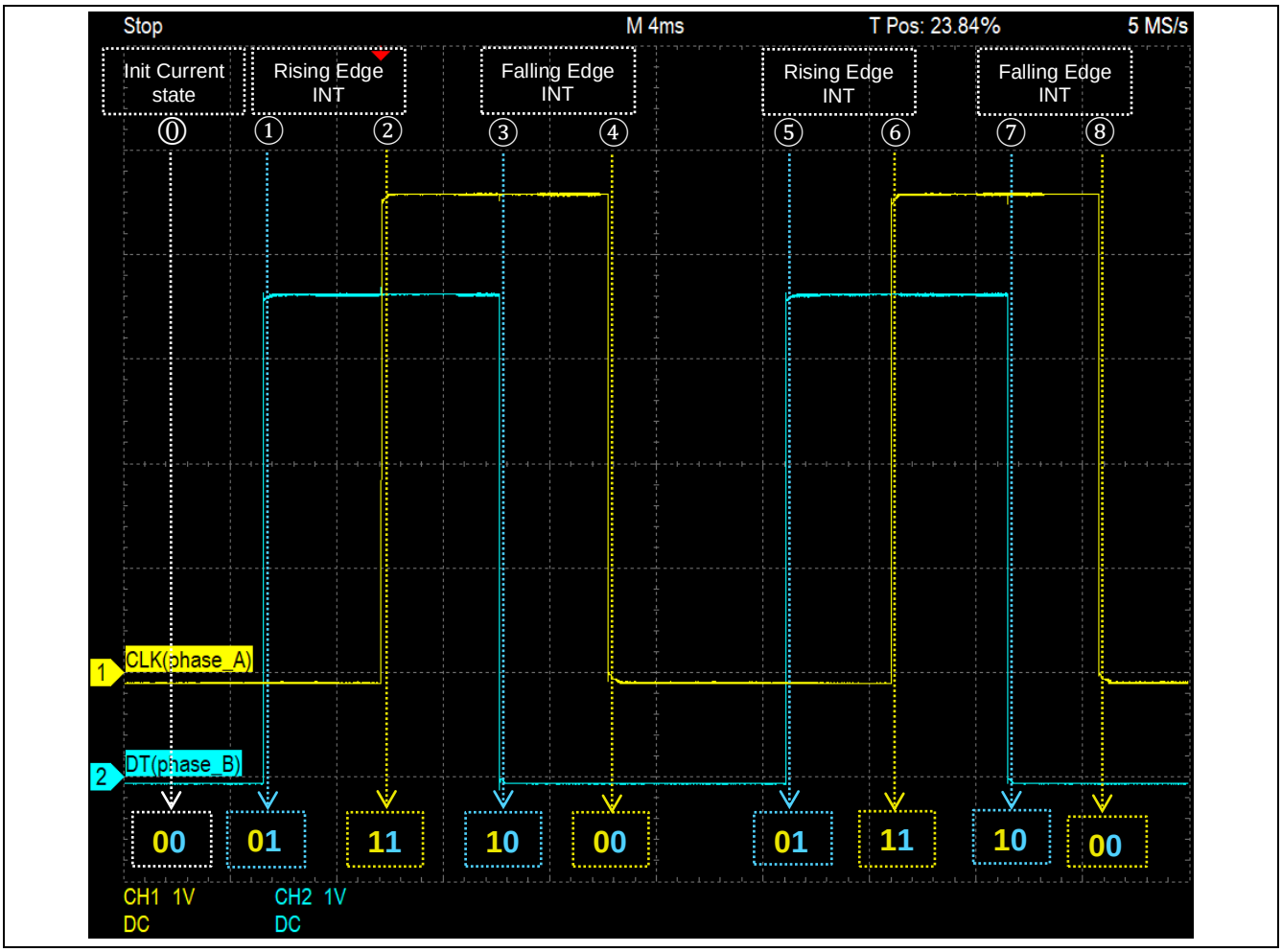

Figure 1-12 Rotary Encoder Switch Counterclockwise Rotation

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

- At timing⓪ the Current state is initialized to 00 (BIN).
- At timing①, Transition=0001(BIN), Index=1(DEC), lookup in Table 1-2→ CCW → "-1"
- At timing②, Transition=0111(BIN), Index=7(DEC), lookup in Table 1-2→ CCW → "-1"

→Two consecutive "-1" s detected →Print "Counter-Clockwise (CCW) Count = -1"

- At timing③, Transition=1110(BIN), Index=14(DEC), lookup in Table 1-2→ CCW→ "-1"
- At timing④, Transition=1000(BIN), Index= 8(DEC), lookup in Table 1-2→ CCW → "-1"

→Two consecutive "-1" s detected →Print "Counter-Clockwise (CCW) Count = -2"

- At timing⑤, Transition=0001(BIN), Index=1(DEC), lookup in Table 1-2→ CCW → "-1"
- At timing⑥, Transition=0111 (BIN), Index=7(DEC), lookup in Table 1-2→ CCW → "-1"

→ **Two consecutive "-1" s detected →Print "Counter-Clockwise (CCW) Count = -3"**

- At timing⑦, Transition=1110 (BIN), Index=14 (DEC), lookup in Table 1-2→ CCW → "-1"
- At timing⑧, Transition=1000 (BIN), Index= 8(DEC), lookup in Table 1-2→ CCW → "-1"

→ **Two consecutive "-1" s detected →Print "Counter-Clockwise (CCW) Count = -4"**

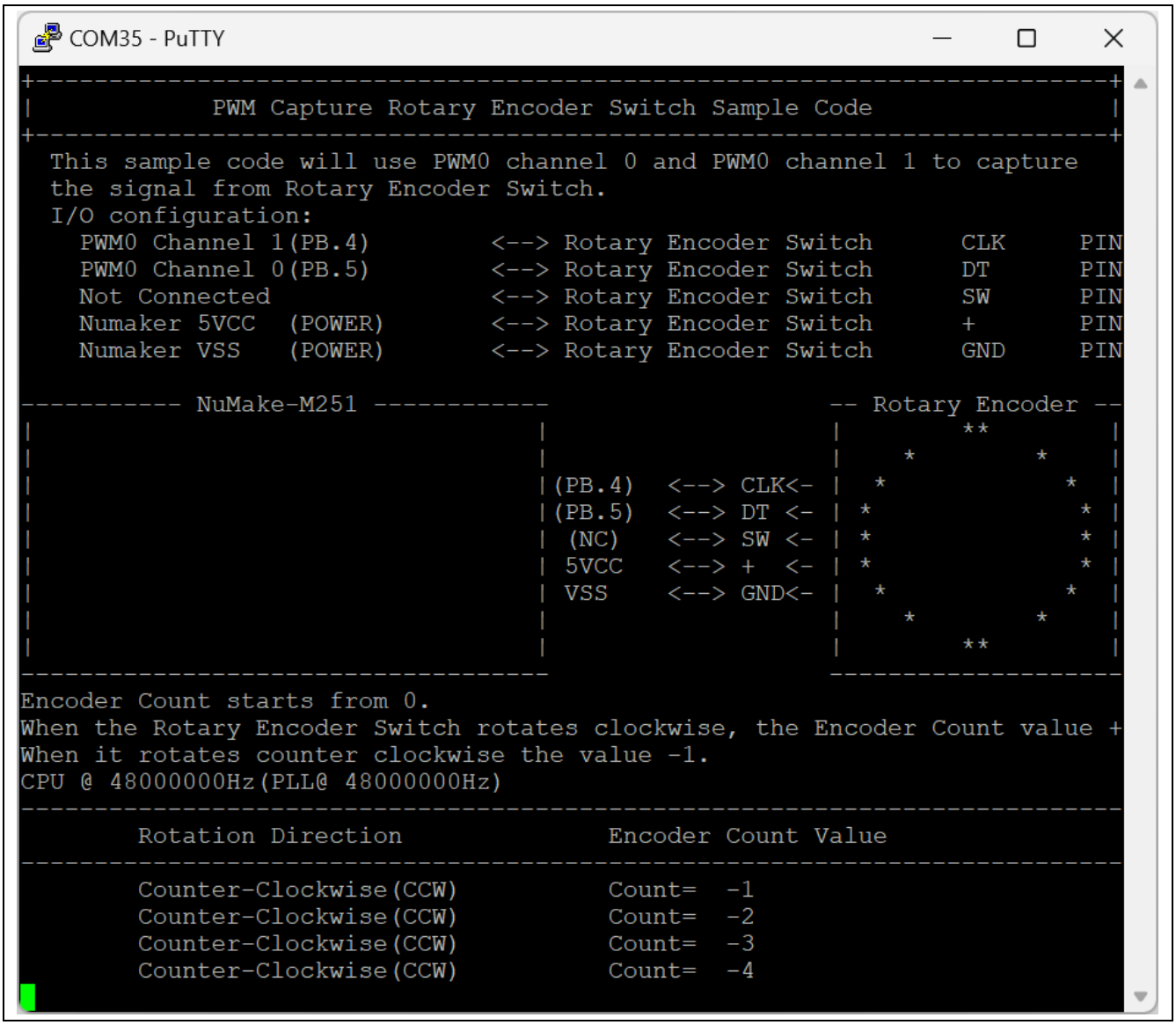

Figure 1-13 Counterclockwise Rotation Execution Result

As shown in Figure 1-12, the execution result of the Rotary Encoder Switch rotating counterclockwise from sequence ⓪ to ⑧ is shown in Figure 1-13.

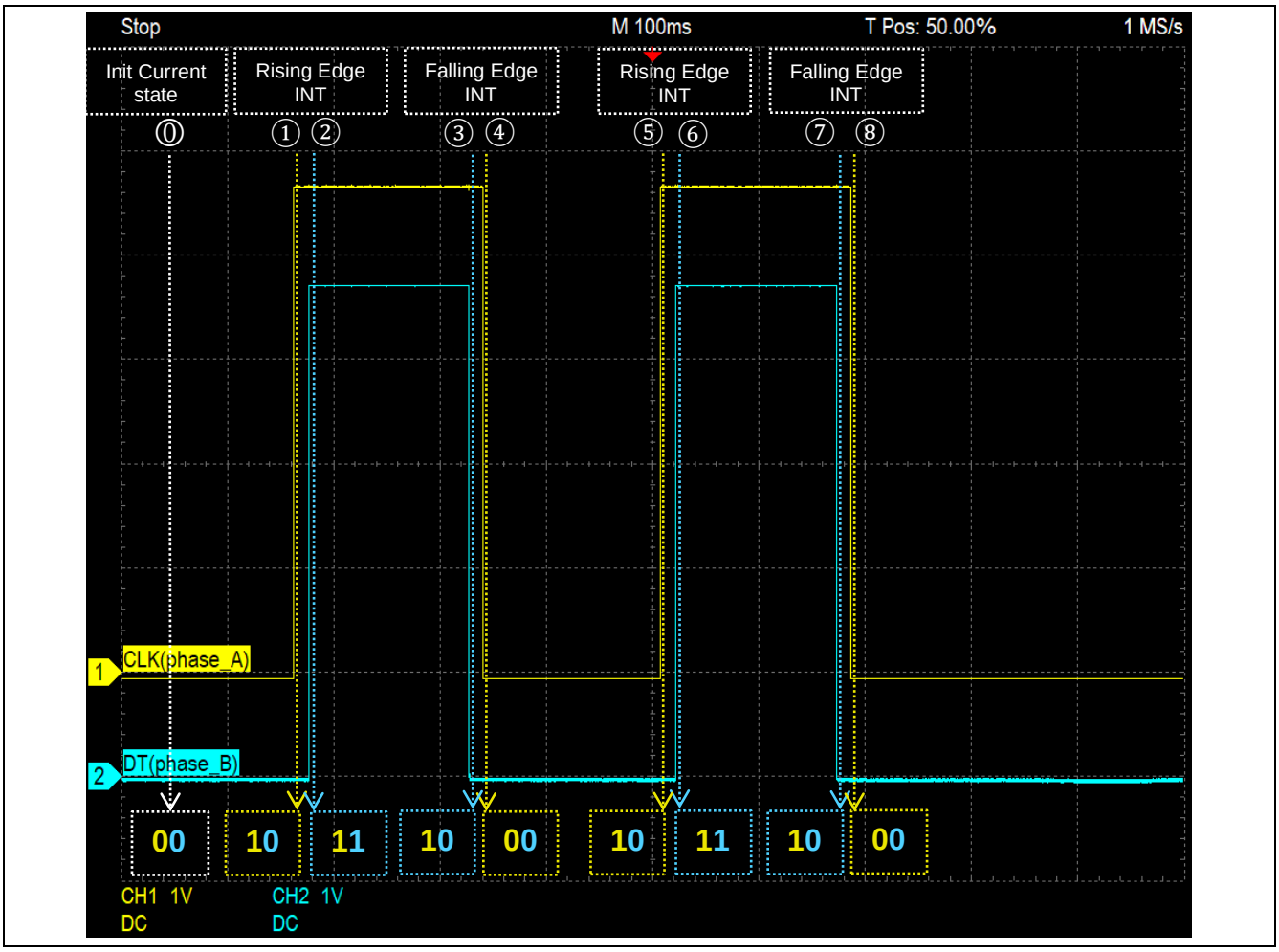

Figure 1-14 Rotary Encoder Switch Clockwise and Counterclockwise Alternating Rotation

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

- At timing⓪the Current state is initialized to 00 (BIN).
- At timing①, Transition=0010(BIN), Index= 2(DEC), lookup in Table 1-2→ CW→ "1"
- At timing②, Transition=1011 (BIN), Index=11(DEC), lookup in Table 1-2→ CW → "1"

→Two consecutive "1" s detected →Print "Clockwise (CW) Count = 1"

- At timing③, Transition=1110(BIN), Index=14(DEC), lookup in Table 1-2→ CCW→ "-1"
- At timing④, Transition=1000(BIN), Index= 8(DEC), lookup in Table 1-2→ CCW → "-1"

→Two consecutive "-1" s detected →Print "Counter-Clockwise(CCW) Count = 0"

- At timing⑤, Transition=0010(BIN), Index= 2(DEC), lookup in Table 1-2→ CW → "1"
- At timing⑥, Transition=1011(BIN), Index=11(DEC), lookup in Table 1-2→ CW → "1"

→ Two consecutive "1" s detected →Print "Clockwise (CW) Count = 1"

- At timing⑦, Transition=1110 (BIN), Index=14 (DEC), lookup in Table 1-2→ CCW → "-1"
- At timing⑧, Transition=1000 (BIN), Index= 8(DEC), lookup in Table 1-2→ CCW → "-1"

→ Two consecutive "-1" s detected →Print "Counter-Clockwise (CCW) Count = 0"

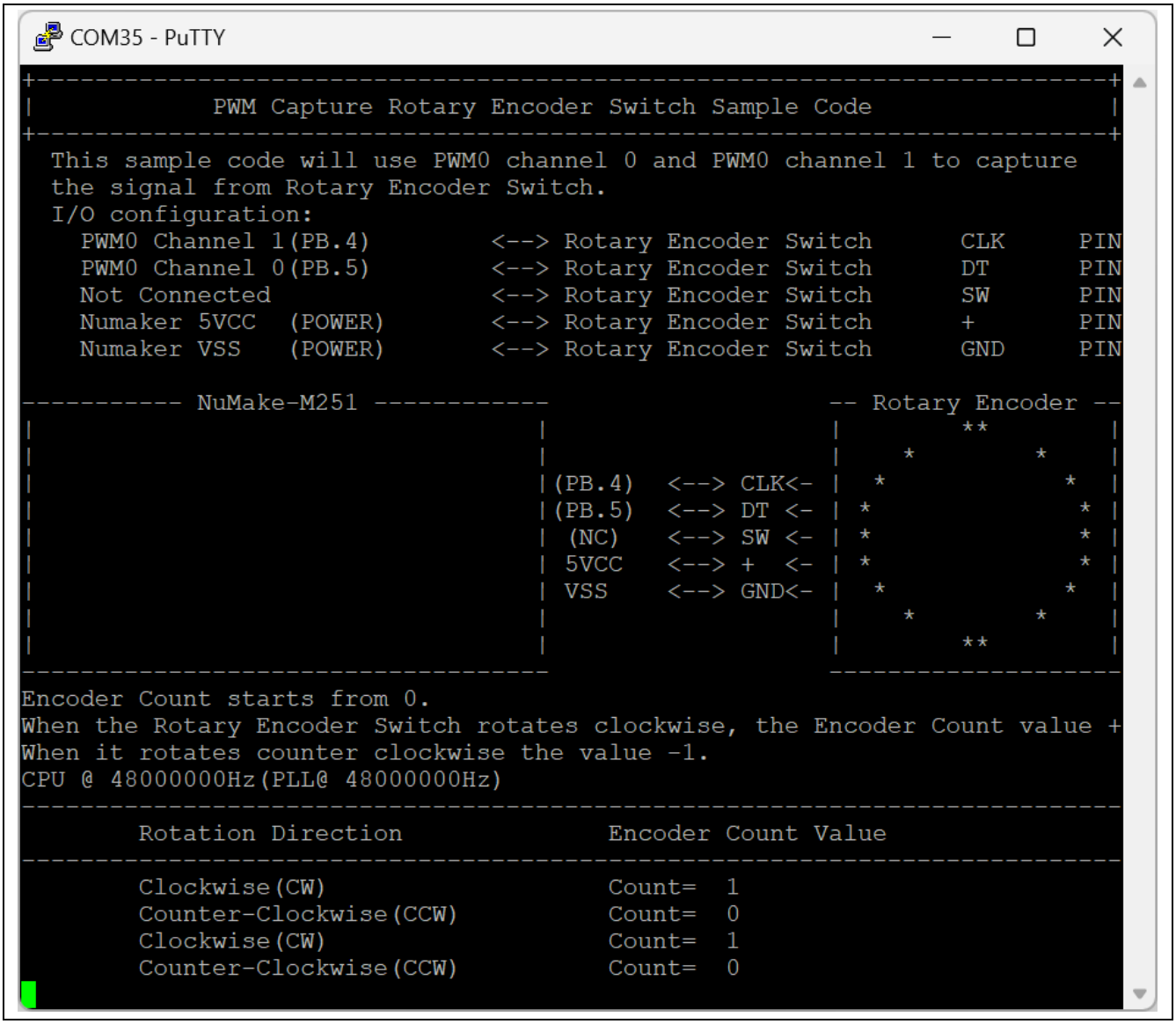

Figure 1-15 Execution Result of Alternating Clockwise and Counterclockwise Rotation

As shown in Figure 1-14, the execution result of the Rotary Encoder Switch alternating between clockwise and counterclockwise rotation from sequence ⓪ to ⑧ is illustrated in Figure 1-15.

2. Code Description

The code workflow is as follows:

1. In *main.c*, initialize the peripherals related to the M251/M252/M254/M256/M258 series, including HIRC, UART0, and PWM0. The main loop in main.c checks whether the Rotary Encoder Switch has rotated.
2. Use the capture interrupts of PWM0 CH0 and CH1 to detect the rising and falling edges of the Rotary Encoder Switch Phase A/B signals.
3. Determine the direction of rotation using a state table and accumulate the changes to filter out bouncing noise.
4. Only when there are two consecutive stables ± changes is it considered a valid rotation, and the event is recorded.
5. A circular queue event_queue is used to store direction and count information

In the while(1){…} main loop, the M251/M252 retrieves one piece of Rotary Encoder Switch data from the event queue and displays the direction and count value in real time, allowing immediate indication of the rotation direction and corresponding count.

```
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 detect rising edge and falling edge interrupts. Upon entering an interrupt, the states of phase A and phase B are first initialized, then the current state (current_state) is recorded as a 2-bit value. When the next interrupt occurs, the current current_state is combined with the previously recorded current_state from the last interrupt to form a 4-bit value (transition). By looking up this transition value in the transition table (transition_table), the rotation direction can be determined. Only when two consecutive interrupts indicate the same rotation direction is that direction considered valid.

```
void PWM0_P0_IRQH 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;
}
```

The transition_table is used to determine the validity of the rotation direction of the Rotary Encoder Switch.

```
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 configures PWM0 to use specific pins; for example, when using PB4 and PB5 to detect pulses generated by the Rotary Encoder Switch, it also initializes the values of PB4 and PB5.

```
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);
}
```

Use a EncoderEvent struct to record Rotary Encoder Switch event data, including the rotation direction and count value.

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

The rotary encoder events (rotation direction and count value) are added to the event queue.

```
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;
    }
}
```

Retrieve one Rotary Encoder Switch event data (rotation direction and count value) from the queue.

```
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. Software and Hardware Requirements

3.1 Software Requirements

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

3.2 Hardware Requirements

- Circuit components
  - NuMaker-M252SD V1.1
- Pin Connect
  - Connect the relevant pins of the NuMaker-M252SD to the corresponding pins of the Rotary Encoder Switch as follows

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

  - Connect the Nu-Link2-Me USB on the NuMaker-M252SD to the PC USB to display the execution results of the example code.

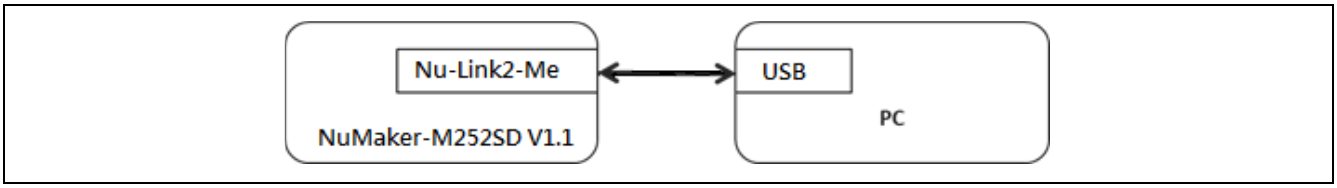

Figure 3-1 NuMaker-M252SDPin Connect

4. Directory Information

The directory structure is shown below.

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

Figure 4-1 Directory Structure

5. Example Code Execution

1. Browse the sample code folder as described in the Directory Information section and double-click *M251_M252_PWM_Capture_Rotary_Encoder.uvprojx*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6. Revision History

Date	Revision	Description
2025.06.16	1.00	Initial version.

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.