Using M251 TIMER Capture to Get Frequency of a Signal

Example Code Introduction for 32-bit NuMicro® Family

Information

Application	This sample code uses M251 TIMER capture function to get frequency of an input signal.
BSP Version	M251_M252_M254_M256_M258_Series_BSP_CMSIS_V3.02.001
Hardware	NuMaker-M251SD Ver1.1

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 TIMER2 to capture the input signal frequency and duty cycle through the capture function, and TIMER0 to output digital signal through the PWM function. The input signal frequency range is 20 ~ 80 kHz. Connect these two pins to confirm the capture frequency and duty cycle. Finally, use a UART to show the result.

1.1 Principle

1.1.1 Get Signal's Frequency Calculation through TIMER Capture Function

The TIMER capture mainly sets interrupts, and calculates the captured frequency value and duty cycle according to the number of interrupts. The interrupt mode of TIMER capture can choose rising edge trigger, falling edge trigger, high level cycle time, or low level cycle time. Combined with the system clock frequency, the value of the fetch frequency can be obtained. This example code sets the system clock to 24MHz, so a Clock time is 0.042us, as shown in Figure 1-1. A clock time of TIMER is 0.042us, too. The TIMER capture interrupt mode is set to high level cycle time and low level cycle time respectively. Capture the counting times of TIMER when the input signal is at a high level, and the counting times of TIMER is at a low level, as shown in Figure 1-2. After adding these two counting times, multiply by 0.042us of the Clock time to get the cycle time of a input signal, as shown in Figure 1-3. Finally, the reciprocal of the cycle time is the input frequency.

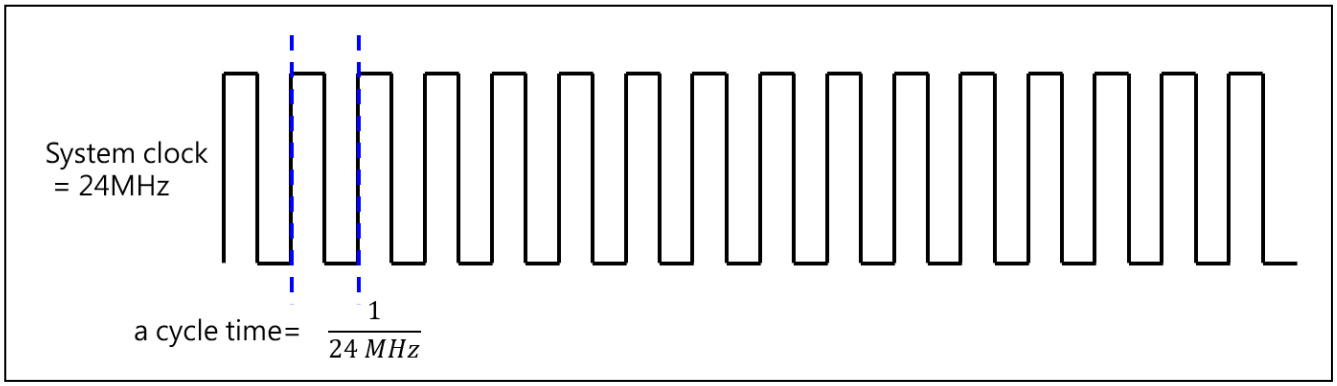

Figure 1-1 Cycle Time of System Clock

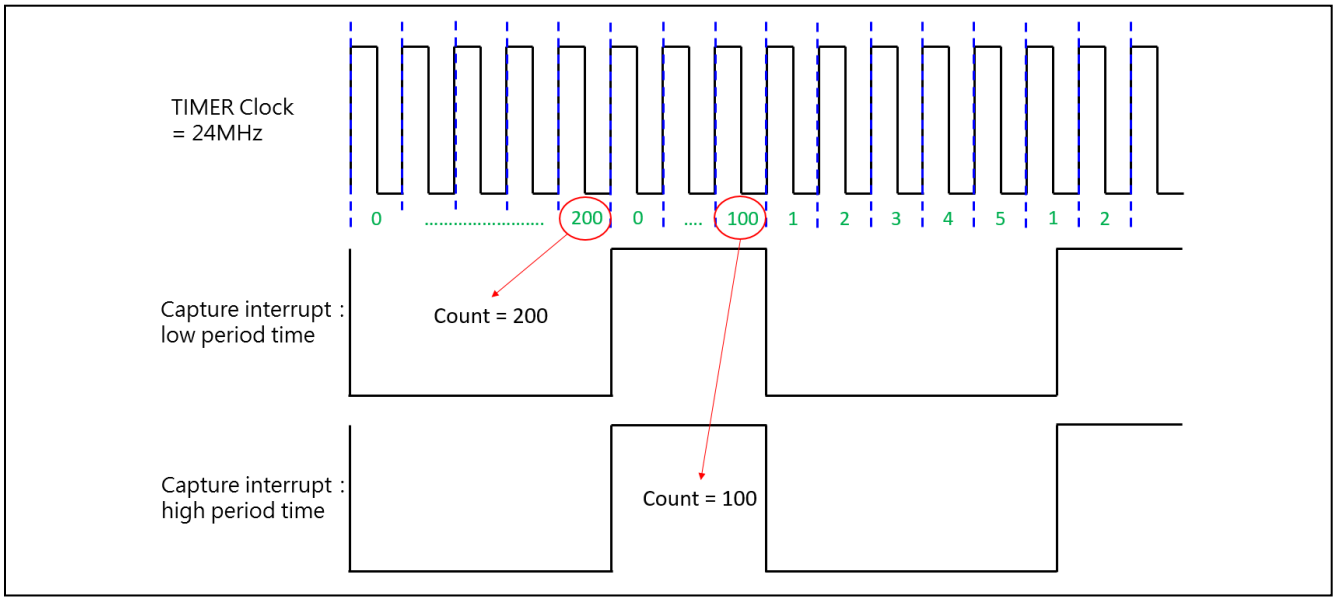

Figure 1-2 Interrupt Mode of TIMER Capture

1.1.2 Get Signal's Duty Cycle Calculation through TIMER Capture Function

The TIMER capture interrupt method is set to the high-level cycle time and the low-level cycle time respectively. Reset the count value every time a Capture interrupt occurs. Finally, calculate the count value captured by the high-level cycle time and low-level cycle time to get the duty cycle of the input signal, as shown in Figure 1-3.

$$Duty\ cycle = \frac{\text{High level cycle time Capture count value}}{\text{High level cycle time Capture count value} + \text{Low level cycle time Capture count value}} \times 100\%$$

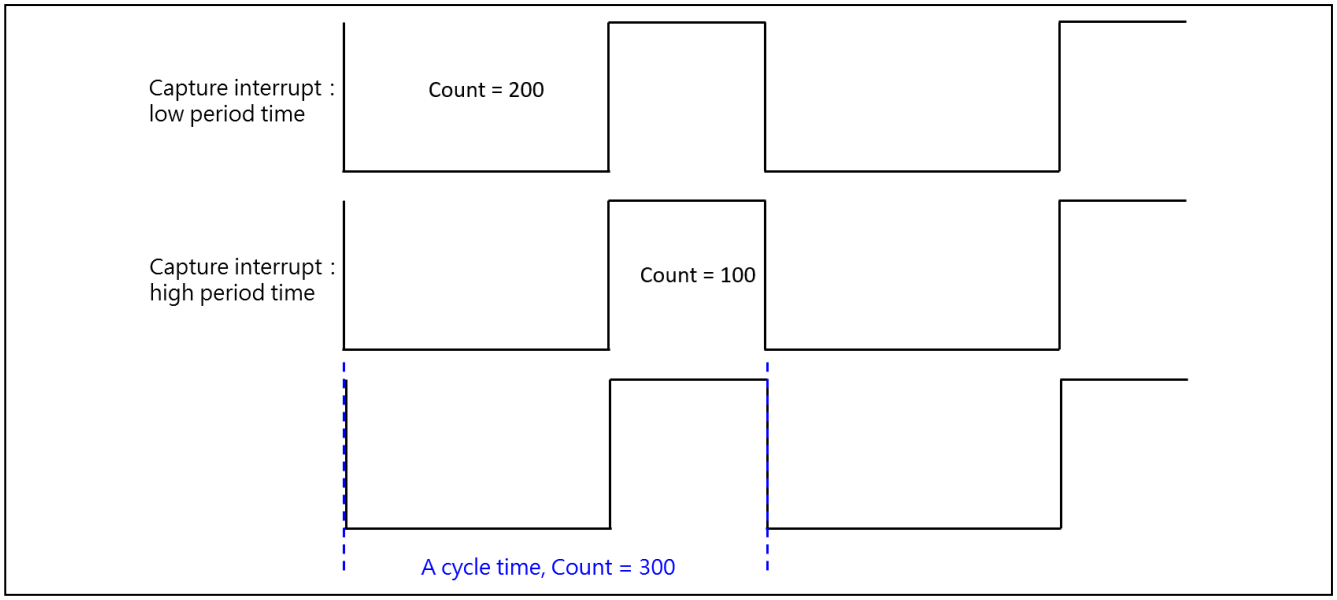

Figure 1-3 TIMER Capture Cycle Time Count of a Signal

1.2 Demo Result

TIMER0 turns on the PWM function, outputs 20 Hz and 80 kHz signals respectively, and the duty cycle is set to 10%, 50% and 100% respectively. TIMER2 activates the Capture function, captures the signal frequency and duty cycle output by TIMER0, confirms whether the calculation result is correct, and displays the result through the terminal software Please refer to Figure 1-4 ~ Figure 1-15.

1.2.1 PWM Signal (Frequency = 20 Hz, Duty Cycle = 10%)

Figure 1-4 is the waveform when the PWM frequency is 20 Hz and the duty cycle is 10%. Figure 1-5 is the actual capture signal result of TIMER2, which is presented through the terminal software.

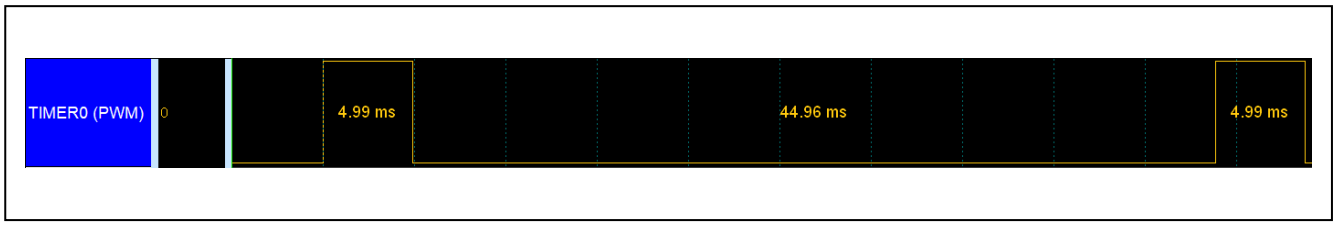

Figure 1-4 TIMER0 Output Frequency = 20 Hz, Duty Cycle = 10%

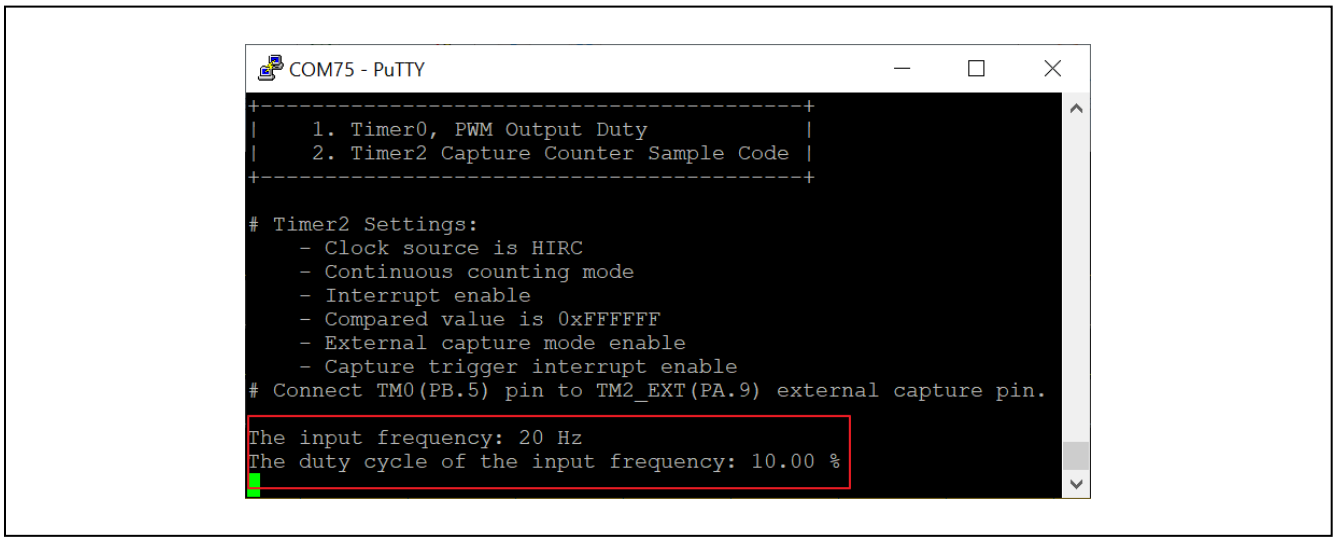

Figure 1-5 TIMER2 Capture Result (Frequency = 20 Hz, Duty Cycle = 10%)

1.2.2 PWM Signal (Frequency = 20 Hz, Duty Cycle = 50%)

Figure 1-6 is the waveform when the PWM frequency is 20 Hz and the duty cycle is 50%. Figure 1-7 is the actual capture signal result of TIMER2, which is presented through the terminal software.

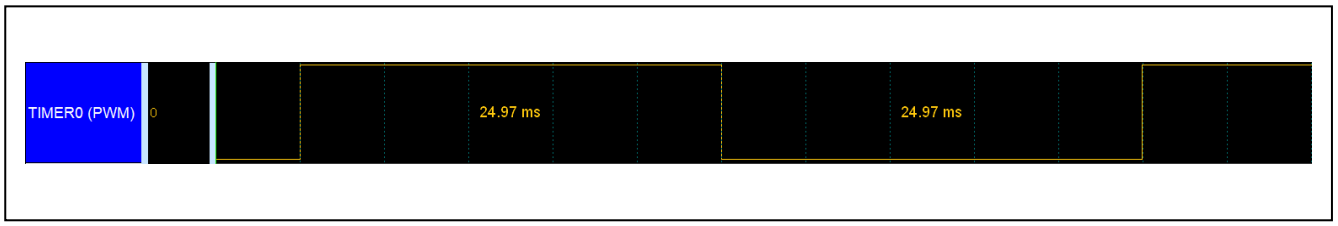

Figure 1-6 TIMER0 Output Frequency = 20 Hz, Duty Cycle = 50%

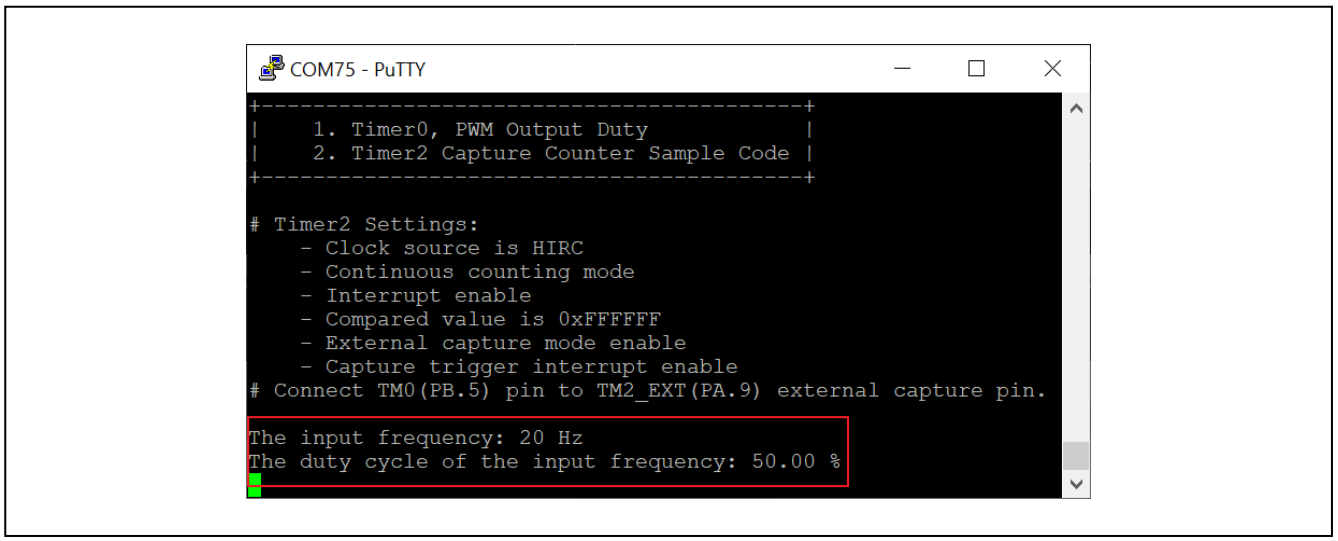

Figure 1-7 TIMER2 Capture Result (Frequency = 20 Hz, Duty Cycle = 50%)

1.2.3 PWM Capture Result (Frequency = 20 Hz, Duty Cycle = 90%)

Figure 1-8 is the waveform when the PWM frequency is 20 Hz and the duty cycle is 90%. Figure 1-9 is the actual capture signal result of TIMER2, which is presented through the terminal software.

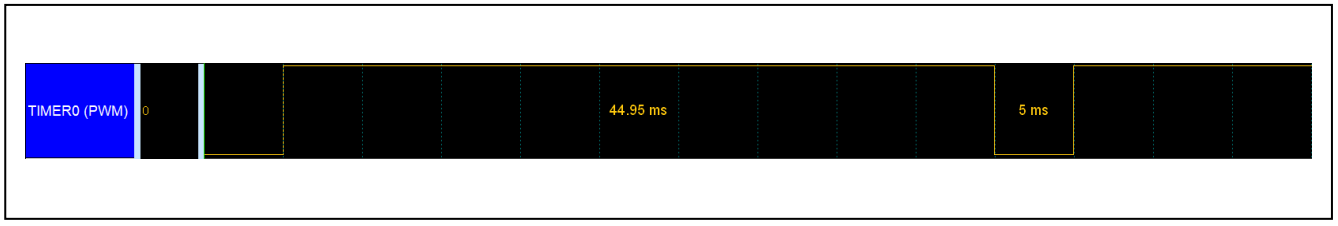

Figure 1-8 TIMER0 Output Frequency = 20 Hz, Duty Cycle = 90%

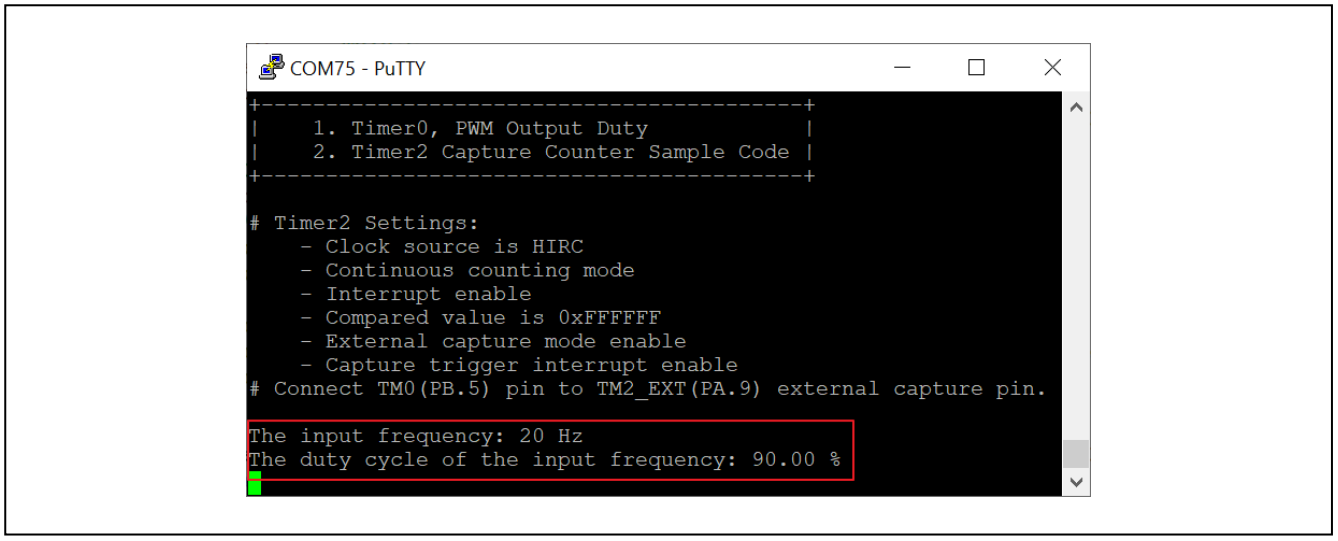

Figure 1-9 TIMER2 Capture Result (Frequency = 20 Hz, Duty Cycle = 90%)

1.2.4 PWM Capture Result (Frequency = 80 kHz, Duty Cycle = 10%)

Figure 1-10 is the waveform when the PWM frequency is 80 kHz and the duty cycle is 10%. Figure 1-11 is the actual capture signal result of TIMER2, which is presented through the terminal software.

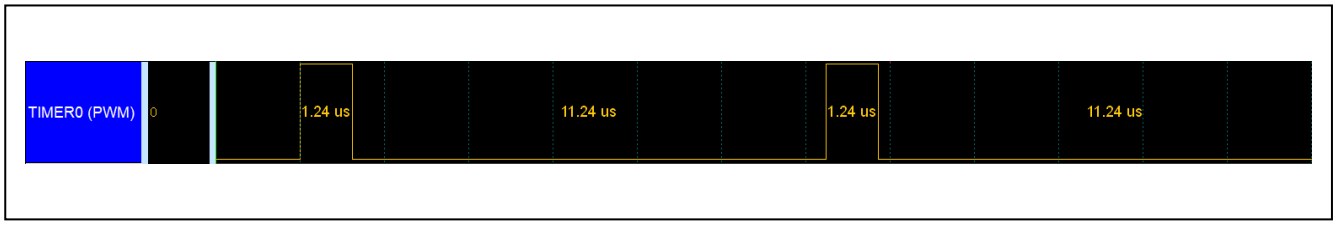

Figure 1-10 TIMER0 Output Frequency = 80 kHz, Duty Cycle = 10%

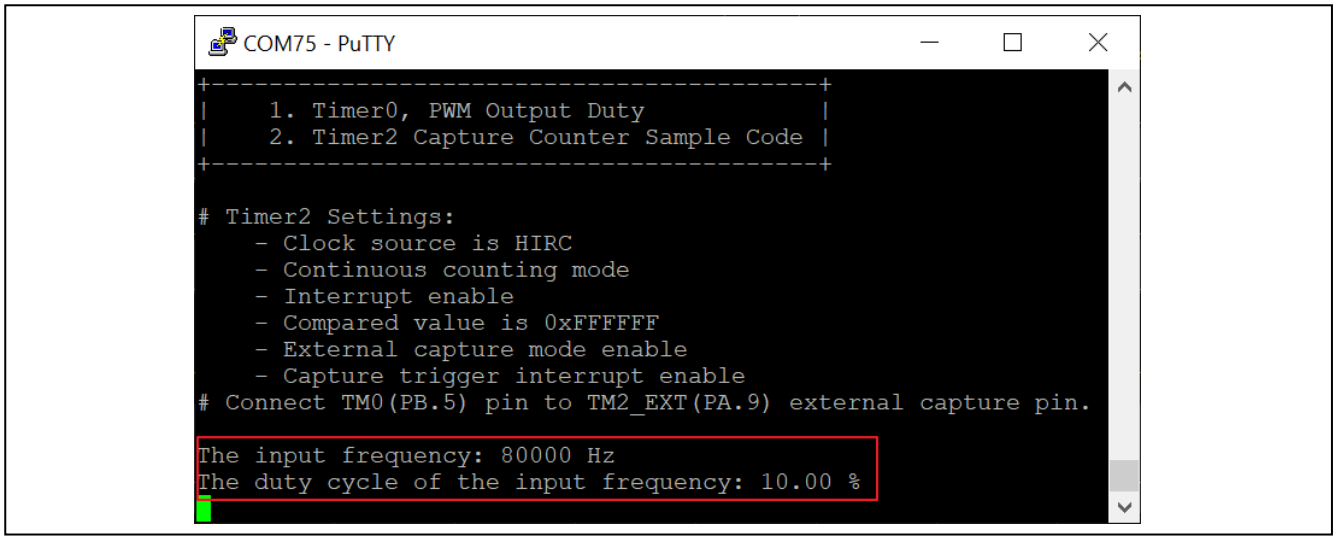

Figure 1-11 TIMER2 Capture Result (Frequency = 80 kHz, Duty Cycle = 10%)

1.2.5 PWM Capture Result (Frequency = 80 kHz, Duty Cycle = 50%)

Figure 1-12 is the waveform when the PWM frequency is 80 kHz and the duty cycle is 50%. Figure 1-13 is the actual capture signal result of TIMER2, which is presented through the terminal software.

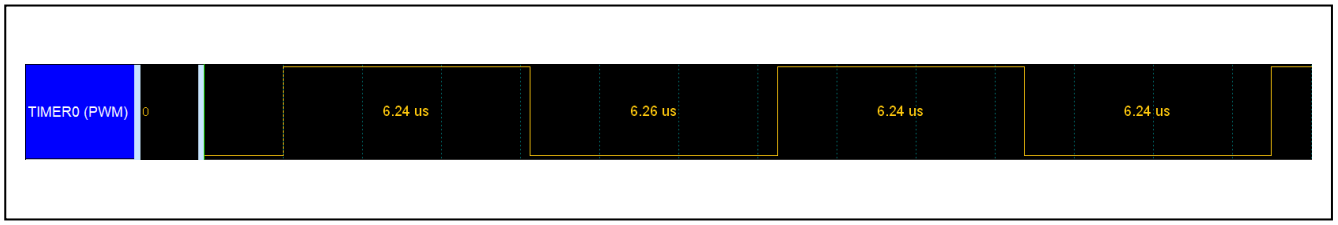

Figure 1-12 TIMER0 Output Frequency = 80 kHz, Duty Cycle = 50%

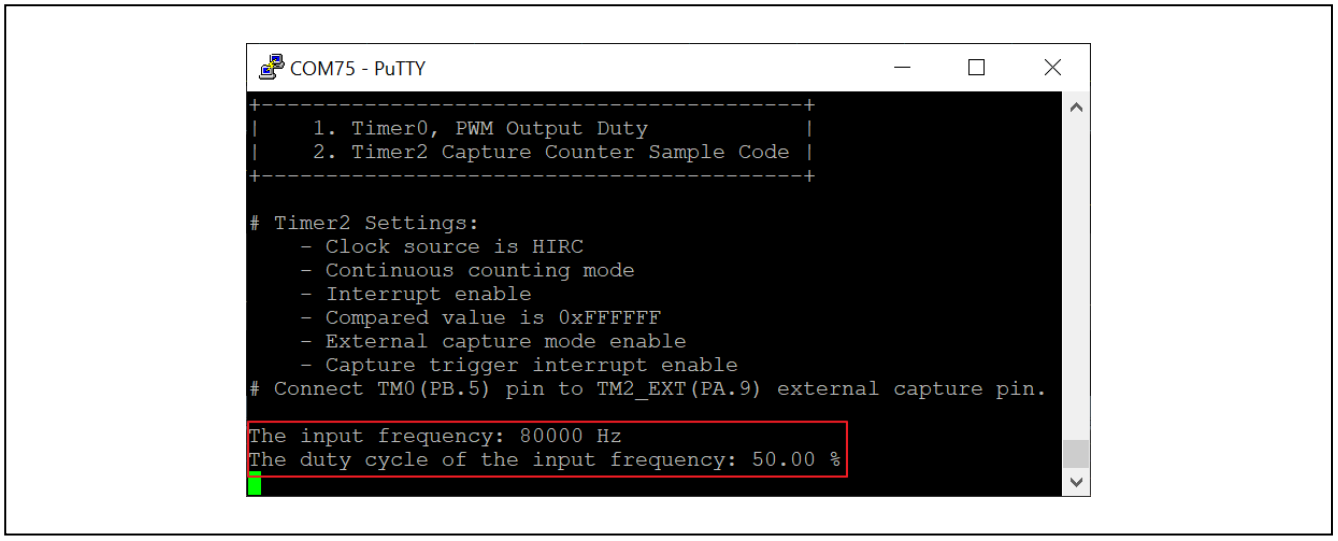

Figure 1-13 TIMER2 Capture Result (Frequency = 80 kHz, Duty Cycle = 50%)

1.2.6 PWM Capture Result (Frequency = 80 kHz, Duty Cycle = 90%)

Figure 1-14 is the waveform when the PWM frequency is 80 kHz and the duty cycle is 90%. Figure 1-15 is the actual capture signal result of TIMER2, which is presented through the terminal software.

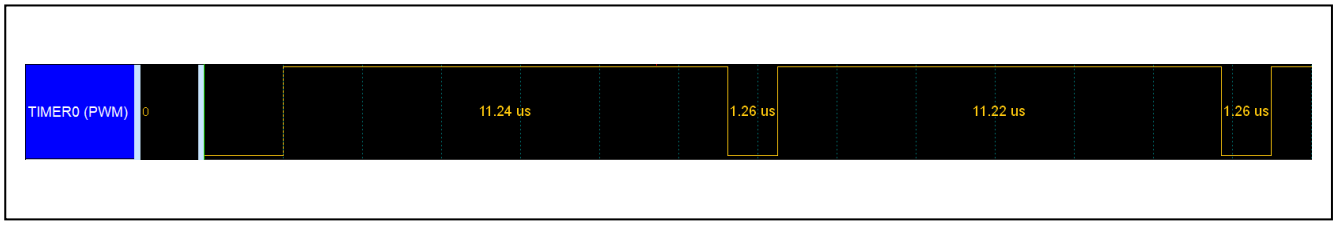

Figure 1-14 TIMER0 Output Frequency = 80 kHz, Duty Cycle = 90%

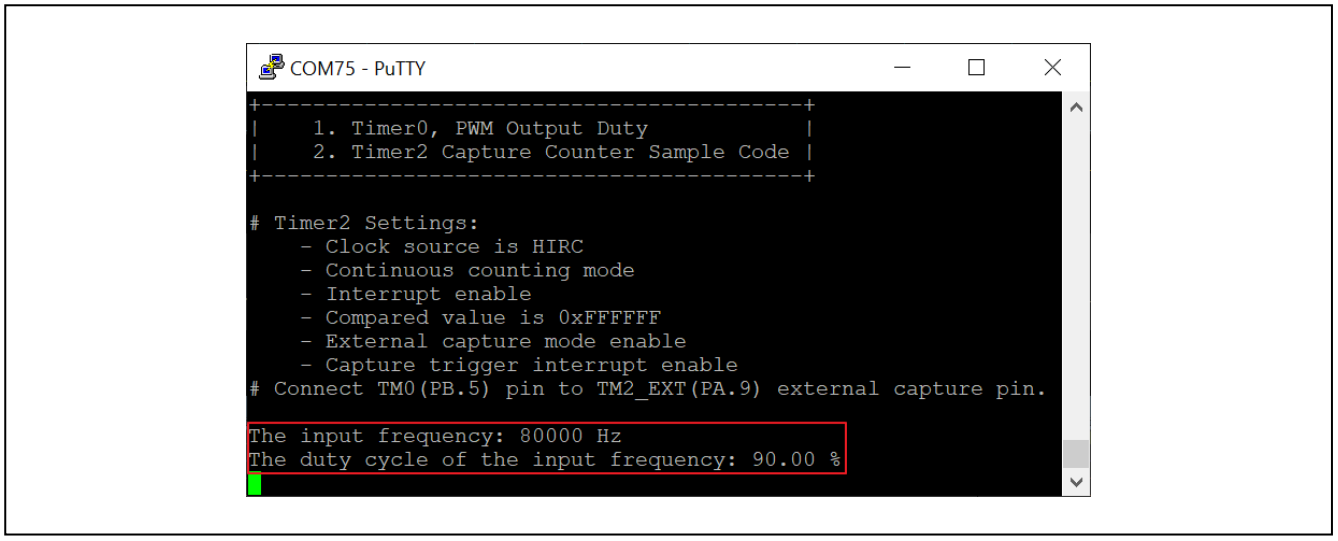

Figure 1-15 TIMER2 Capture Result (Frequency = 80 kHz, Duty Cycle = 90%)

2. Code Description

2.1 Main Code Description

The steps are listed below:

1. Initialize related peripherals of the M251/M252/M254/M256/M258 series in main.c, including HIRC, UART0, TIMER0, TIMER1 and TIMER2.
2. Determine whether the set frequency and duty cycle of TIMER0 are within the required range, and enable the PWM function of TIMER0, the Capture function of TIMER2, and the interrupt function of TIMER2 Capture.
3. Set the interrupt mode of TIMER2 Capture to TIMER_CAPTURE_EVENT_GET_LOW_PERIOD, and the count value captured can be used to calculate the low cycle time of the capture signal, and stored in the g_au32TMRINTValue[ ] array after capture. The index 0 of the g_au32TMRINTValue[ ] array is used to store this count value.
4. Set the interrupt mode of TIMER2 Capture to TIMER_CAPTURE_EVENT_GET_HIGH_PERIOD, and the captured count value can be used to calculate the high cycle time of the capture signal, and stored in the g_au32TMRINTValue[ ] array after capture. The index 1 of the g_au32TMRINTValue[ ] array is used to store this count value.
5. Take out the count value stored in the g_au32TMRINTValue[ ] array, and calculate the captured signal frequency and duty cycle respectively.

Main code:

```
int main(void)
{
    /* Init System, peripheral clock and multi-function I/O */
    SYS_Init();

    /* Init UART0 for printf and test */
    UART0_Init();

printf("+-----------------------------------------+\n");
printf("|    1. Timer0, PWM Output Duty           |\n");
    printf("|    2. Timer2 Capture Counter Sample Code |\n");
    printf("+-----------------------------------------+\n\n");
    printf("# Timer2 Settings:                          \n");
    printf("    - Clock source is PLL                   \n");
    printf("    - Continuous counting mode              \n");
    printf("    - Interrupt enable                      \n");
    printf("    - Compared value is 0xFFFFFF            \n");
    printf("    - External capture mode enable          \n");
    printf("    - Capture trigger interrupt enable      \n");
    printf("# Connect TM0(PB.5) pin to TM2_EXT(PA.9) external capture pin.\n\n");

    printf(" Please set frequency, then press the ENTER.: (Range: 20 ~ 80k Hz)\n");
    printf(" Frequency: ");

    while (1)
```

```
{
    if(g_u8FinishSet == TRUE)
    {
    if((((int)g_u32Frequency < 20) || ((int)g_u32Frequency > 80000)) ||
        (((int)g_u32Duty < 0) || ((int)g_u32Duty > 100)))
    {
        printf(" Please check the frequency (range: 20 ~ 80k Hz) and duty cycle
            (range: 0 ~ 100 %%) setting! \n\n");

        g_u8FinishSet = FALSE;

        printf(" Please set frequency, then press the ENTER.: \n");
        printf(" Frequency: ");
    }
    else
    {
        /* Enable Timer2 NVIC */
        NVIC_EnableIRQ(TMR2_IRQn);

        /* Change Timer to PWM counter mode */
        TPWM_ENABLE_PWM_MODE(TIMER0);

        /* Enable output of PWM_CH0 */
        TPWM_ENABLE_OUTPUT(TIMER0, TPWM_CH0);

        /* Set Timer0 PWM output frequency and duty cycle in up count type */
        if (TPWM_ConfigOutputFreqAndDuty(TIMER0, g_u32Frequency, g_u32Duty)
            != g_u32Frequency)
        {
            printf("\n Set the frequency different from the user\n");
        }

        TPWM_START_COUNTER(TIMER0);

        TIMER_Delay(TIMER1, 500000);

        Get_PWMSignal();

        printf(" Please set frequency, then press the ENTER.: \n");
        printf(" Frequency: ");

        g_u32CaptureCount = 0;
        g_u8FinishSet = FALSE;
    }
    }
}
}
```

The Get_PWMSignal( ) function is used to set the Capture mode of TIMER2, set the Low period and High period modes respectively, and enable the Capture interrupt. When TIMER2 enters the Capture interrupt, the captured counter value will be stored in the au32TMRINTValue[ ] array, and finally through the au32TMRINTValue[ ] the value stored in the array is used to calculate the input signal frequency value and Duty value. Please refer to the code below.

```
void Get_PWMSignal(void)
```

```
{
    /* Step1. Capture low period of input frequency */
    TIMER_SET_PRESCALE_VALUE(TIMER2, 0);
    TIMER_SET_CMP_VALUE(TIMER2, 0xFFFFFF);
    TIMER_EnableCapture(TIMER2, TIMER_CAPTURE_COUNTER_RESET_MODE,
        TIMER_CAPTURE_EVENT_GET_LOW_PERIOD);
    TIMER_EnableCaptureInt(TIMER2);
    g_u32CaptureFinish = 0;

    /* Start Timer0 PWM counter and Timer2 counting*/
    TIMER_Start(TIMER2);
    while(g_u32CaptureFinish == 0);

    TIMER_StopCapture(TIMER2);
    TIMER_Stop(TIMER2);

    while (TIMER_IS_ACTIVE(TIMER2)){};
    TIMER_ClearCaptureIntFlag(TIMER2);

    /* Step2. Capture high period of input frequency */
    TIMER_SET_PRESCALE_VALUE(TIMER2, 0);
    TIMER_SET_CMP_VALUE(TIMER2, 0xFFFFFF);
    TIMER_EnableCapture(TIMER2, TIMER_CAPTURE_COUNTER_RESET_MODE,
        TIMER_CAPTURE_EVENT_GET_HIGH_PERIOD);
    TIMER_EnableCaptureInt(TIMER2);
    g_u32CaptureFinish = 0;

    /* Start Timer0 PWM counter and Timer2 counting*/
    TIMER_Start(TIMER2);
    while(g_u32CaptureFinish == 0);

    TIMER_StopCapture(TIMER2);
    TIMER_Stop(TIMER2);

    while (TIMER_IS_ACTIVE(TIMER2)){};
    TIMER_ClearCaptureIntFlag(TIMER2);

    printf("\n The input frequency: %d Hz\n", (int)PLL_CLOCK / (g_au32TMRINTValue[1] +
        g_au32TMRINTValue[0]));
```

```
    printf(" The duty cycle of the input frequency: %.2f %%\n\n\n",
        (float)(g_au32TMRINTValue[1] * 100) / (g_au32TMRINTValue[1] +
        g_au32TMRINTValue[0]));
}
```

Declare variables and arrays here to control the count value obtained when TIMER2 capture generates an interrupt.

```
#define PLL_CLOCK 24000000UL
#define RX_BUFFER_SIZE   50
#define RX_TIMEOUT_CNT   50
volatile uint32_t g_au32TMRINTCount = 0, g_u32CaptureFinish = 0, g_u32CaptureCount = 0;
volatile uint32_t g_au32TMRINTValue[2] = {0};
```

The TIMER2 capture interrupt code is as follows:

The main code interrupt mode is set to TIMER_CAPTURE_EVENT_GET_LOW_PERIOD and TIMER_CAPTURE_EVENT_GET_HIGH_PERIOD respectively. And sequentially through the TIMER_GetCaptureData( ) function, the obtained count value is stored in the g_au32TMRINTValue[ ] array.

```
void TMR2_IRQHandler(void)
{
    if (TIMER_GetCaptureIntFlag(TIMER2))
    {
    /* Clear Timer2 capture trigger interrupt flag */
    TIMER_ClearCaptureIntFlag(TIMER2);

    if(g_au32TMRINTCount == 0)
    {
        g_au32TMRINTCount = 1;
    }
    else
    {
        g_au32TMRINTValue[g_captureCount++] = TIMER_GetCaptureData(TIMER2);

        g_au32TMRINTCount = 0;

        g_captureFinish = 1;
    }
    }
}
```

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M251_M252_M254_M256_M258_Series_BSP_CMSIS_V3.02.001
- IDE version
  - Keil uVersion 5.26

3.2 Hardware Requirements

- Circuit components
  - NuMaker-M251SD Ver1.1
- Pin Connect

PB.5 of M251 is set to TIMER0, and the PWM function is turned on to output external signals. PA.9 is set to TIMER2_EXT, and the Capture function is turned on to capture the frequency and duty cycle of the signal generated by TIMER0.

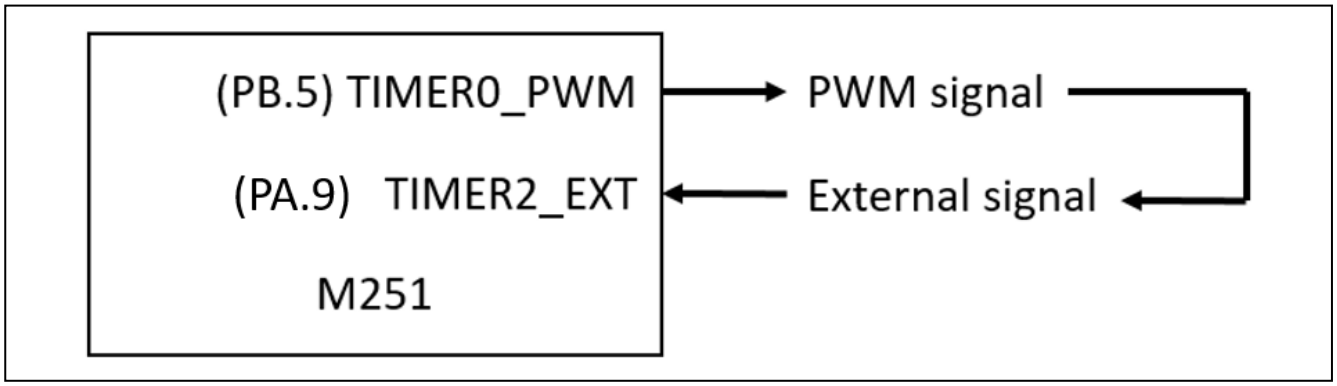

Figure 3-1 Pin connect

4. Directory Information

The directory structure is shown below.

EC_ M251_TIMER_Capture_Input_Signal_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
Sample_Code	
ExampleCode	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_TIMER_Capture_Input_Signal.uvprojx.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6. Revision History

Date	Revision	Description
2023.01.18	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.