PWM Fan Control

Example Code Introduction for 32-bit NuMicro® Family

Information

Application	This code uses Mini51 PWM Channel 0 (P2.2) to control 4-wire PWM fan speed and uses Timer 0 external capture pin (P3.2) to detect fan speed in RPM.
BSP Version	Mini51DE_Series_BSP_CMSIS_v3.02.000
Hardware	NuTiny-SDK-Mini54 V3.1 4-wire PWM Fan

1 Function Description

1.1 Introduction

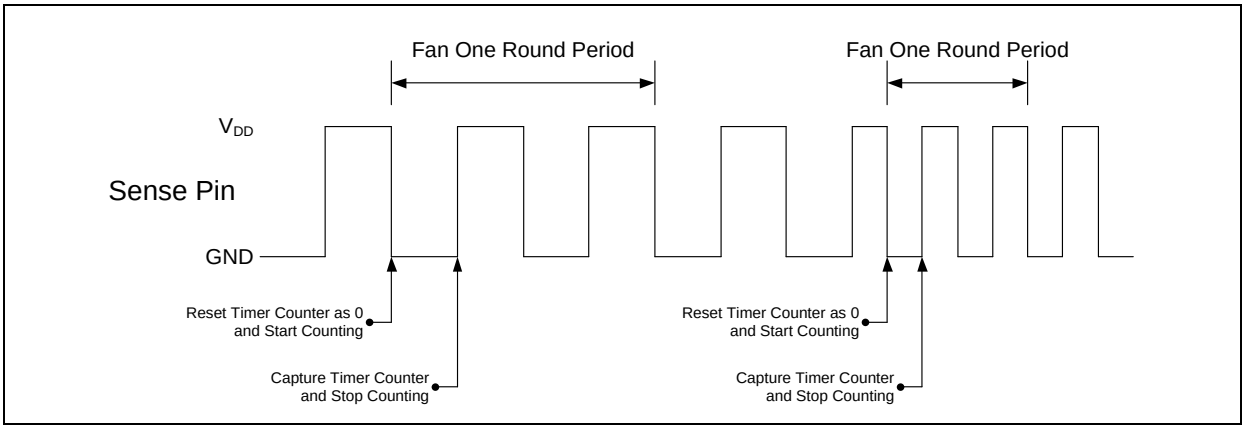
This example code uses PWM Channel 0 (P2.2) to control 4-wire PWM fan speed and uses Timer 0 external capture pin (P3.2) to detect fan speed in RPM. Inputting PWM waveform to the PWM fan control pin can control fan speed. By changing PWM duty ratio, PWM fan can operate in suitable speed. The PWM fan sense pin outputs tachometer signal, which changes voltage level when rotating from one pole to another. By calculating the time which fan takes when rotating one round, user can get the fan speed in RPM.

To use this example code, user should connect PWM channel 0 (P2.2) to PWM fan control pin and connect Timer 0 external capture pin (P3.2) to fan sense pin.

1.2 Principle

To operate in suitable speed for different situation, the 4-wire PWM fan has control pin to receive PWM waveform and to control the MOSFET switch. When increasing PWM duty ratio, the MOSFET switch turns on longer and fan rotates faster. Based on the specification Intel released, the PWM output frequency should be 25 kHz.

User can also get the current fan speed. The 4-pole motor PWM fan sense pin outputs tachometer signal shown below:



When fan motor rotates from one pole to another, the tachometer signal changes voltage level. User can connect Timer 0 external capture pin (T0EX) to fan sense pin and enable trigger counting capture mode with level change trigger. When T0EX detects the high to low transition, Timer 0 resets counter value as zero and starts counting. When T0EX detects the

low to high transition, Timer 0 stops counting and captures the counter value to Timer Capture Data Register (TCAP). Since the PWM fan sense pin is open-drain type, user has to use pull-up resistor on it.

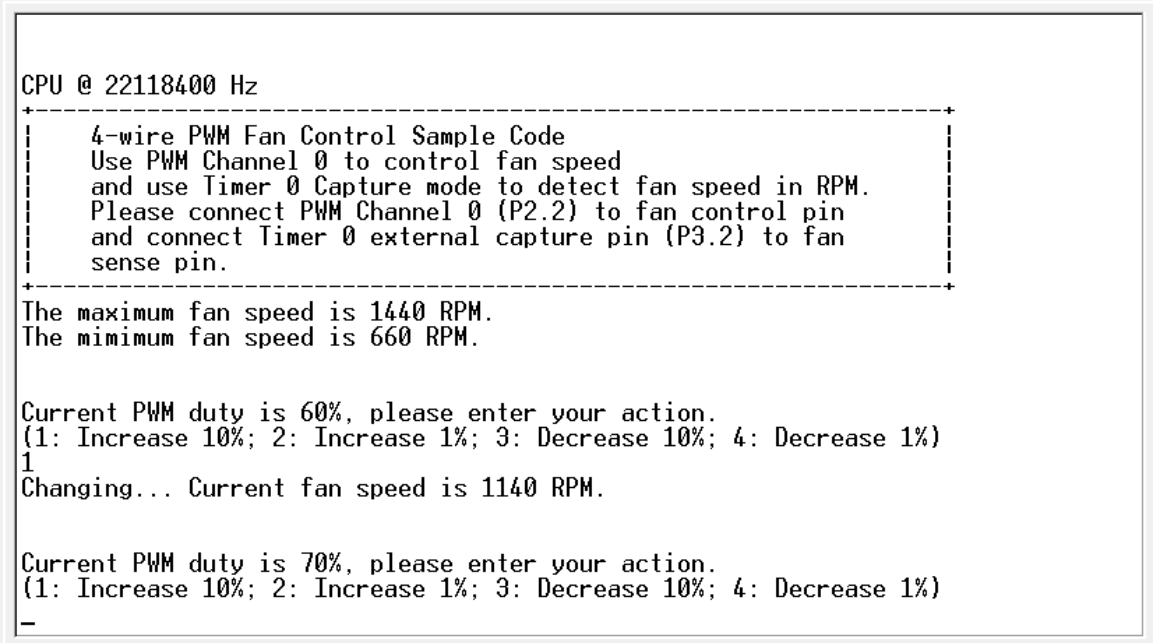
By using equation shown below, user can calculate the fan speed in RPM.

$$Fan\ Speed\ in\ RPM\ = \frac{Timer\ 0\ Clock\ Source\ Frequency * 60}{(Motor\ Pole\ Number) * (Timer\ 0\ Capture\ Value)}$$

1.3 Demo Result

This example code sets the duty 100% PWM waveform to detect the maximum fan speed firstly. Then it sets the duty 0% PWM waveform to detect the minimum fan speed. It takes three seconds delay time to wait fan stable every time if the PWM duty is changed. After above action, user can input further action that increases 10% or 1% PWM duty or decreases 10% or 1% PWM duty.

The result is shown below picture:



2 Code Description

Initialize Timer 0 and enable trigger counting capture mode with level change trigger.

```
void TMR0_Init(void)
{
    /*---------------------------------------------------------------------------------------*/
    /* Init Timer 0                                                                          */
    /*---------------------------------------------------------------------------------------*/
    /* Reset IP */
    SYS_ResetModule(TMR0_RST);

    /* Give a dummy target frequency here. Will over write capture resolution with macro
*/
    TIMER_Open(TIMER0, TIMER_PERIODIC_MODE, 1000000);

    /* Update prescale to set proper resolution */
    TIMER_SET_PRESCALE_VALUE(TIMER0, 0);

    /* Set compare value as large as possible, so don't need to worry about counter
overrun too frequently */
    TIMER_SET_CMP_VALUE(TIMER0, 0xFFFFFF);

    /* Configure Timer 0 trigger counting mode and level change trigger.
       The high to low transition on Timer 0 external capture pin is detected to
       reset TDR as 0 and then starts counting, while low to high transition stops
       counting. */
    TIMER_EnableCapture(TIMER0, TIMER_CAPTURE_TRIGGER_COUNTING_MODE,
TIMER_CAPTURE_FALLING_THEN_RISING_EDGE);

    /* Enable T0EX debounce function */
    TIMER0->TEXCON |= TIMER_TEXCON_TEXDB_Msk;

    /* Enable timer interrupt */
    TIMER_EnableCaptureInt(TIMER0);
    NVIC_EnableIRQ(TMR0_IRQn);
}
```

Initialize PWM Channel 0 frequency to 25 kHz.

```
void PWM_Init(void)
{
    /*---------------------------------------------------------------------------------------*/
```

```
    /* Init PWM Channel 0                                                     */
    /*------------------------------------------------------------------------*/
    /* Reset IP */
    SYS_ResetModule(PWM_RST);

    /* Set PWM Channel 0 frequency to 25 kHz and initial duty 100% */
    PWM_ConfigOutputChannel(PWM, 0, 25000, 100);

    /* Enable PWM Channel 0 output */
    PWM_EnableOutput(PWM, BIT0);
}
```

Stop Timer 0 when it captures the counter value and calculate the fan speed in Timer 0 interrupt handler.

```
void TMR0_IRQHandler(void)
{
    uint32_t u32_FanRPM;

    /* Stop Timer 0 */
    TIMER_Stop(TIMER0);

    /* Calculate RPM = (Timer 0 Clock Source Frequency / (Fan_Pole * Timer 0 Capture
Value)) * 60 */
    u32_FanRPM = (22118400 / (Fan_Pole * TIMER_GetCaptureData(TIMER0))) * 60;
    printf("%d RPM.\n", u32_FanRPM);

    /* Clear Timer 0 Capture interrupt flag */
    TIMER_ClearCaptureIntFlag(TIMER0);

    /* Set Detect done flag*/
    g_u8DetectFlag = 1;
}
```

Test maximum fan speed and minimum fan speed.

```
    /* Test maximum fan speed */
    /* Start PWM Channel 0 */
    PWM_Start(PWM, BIT0);
    /* Wait 3 seconds */
    Delay_mS(3000);
    /* Start Timer 0 to detect the time which fan takes when rotating one round */
    g_u8DetectFlag = 0;
    printf("The maximum fan speed is ");
    TIMER_Start(TIMER0);
```

```
    while (!g_u8DetectFlag);

    /* Test maximum fan speed */
    /* Force PWM Channel 0 output low */
    PWM->PHCHG = PWM->PHCHGNXT & ~(PWM_PHCHGNXT_PWM0_Msk | PWM_PHCHGNXT_D0_Msk);
    /* Wait 3 seconds */
    Delay_mS(3000);
    /* Start Timer 0 to detect the time which fan takes when rotating one round */
    g_u8DetectFlag = 0;
    printf("The mimimum fan speed is ");
    TIMER_Start(TIMER0);

    while (!g_u8DetectFlag);
```

Increase or decrease PWM duty based on user input action.

```
    while (1)
    {
        printf("\n\nCurrent PWM duty is %d%%, please enter your action.\n", g_u8PWMDuty);
        printf("(1: Increase 10%%; 2: Increase 1%%; 3: Decrease 10%%; 4: Decrease
1%%)\n");
        u8Action = getchar();
        printf("%c\n", u8Action);

        switch (u8Action)
        {
            case '1':
            {
                /* Increase PWM duty 10% */
                (g_u8PWMDuty > 90) ? (g_u8PWMDuty = 100) : (g_u8PWMDuty += 10);

                break;
            }

            case '2':
            {
                /* Increase PWM duty 1% */
                (g_u8PWMDuty > 99) ? (g_u8PWMDuty = 100) : (g_u8PWMDuty++);

                break;
            }
```

```
            case '3':
            {
                /* Decrease PWM duty 10% */
                (g_u8PWMDuty < 10) ? (g_u8PWMDuty = 0) : (g_u8PWMDuty -= 10);

                break;
            }

            case '4':
            {
                /* Decrease PWM duty 1% */
                (g_u8PWMDuty < 1) ? (g_u8PWMDuty = 0) : (g_u8PWMDuty--);

                break;
            }

            default:
            {
                continue;
            }
        }

        /* Set PWM Channel 0 frequency and duty */
        if (g_u8PWMDuty > 0)
        {
            PWM->PHCHG |= PWM_PHCHGNXT_PWM0_Msk;
            PWM_ConfigOutputChannel(PWM, 0, 25000, g_u8PWMDuty);
        }
        else
            PWM->PHCHG = PWM->PHCHGNXT & ~(PWM_PHCHGNXT_PWM0_Msk | PWM_PHCHGNXT_D0_Msk);

        /* Wait 3 seconds */
        printf("Changing... ");
        Delay_mS(3000);
        /* Start Timer 0 to detect the time which fan takes when rotating one round */
        g_u8DetectFlag = 0;
        printf("Current fan speed is ");
        TIMER_Start(TIMER0);

        while (!g_u8DetectFlag);
    }
```

3 Software and Hardware Environment

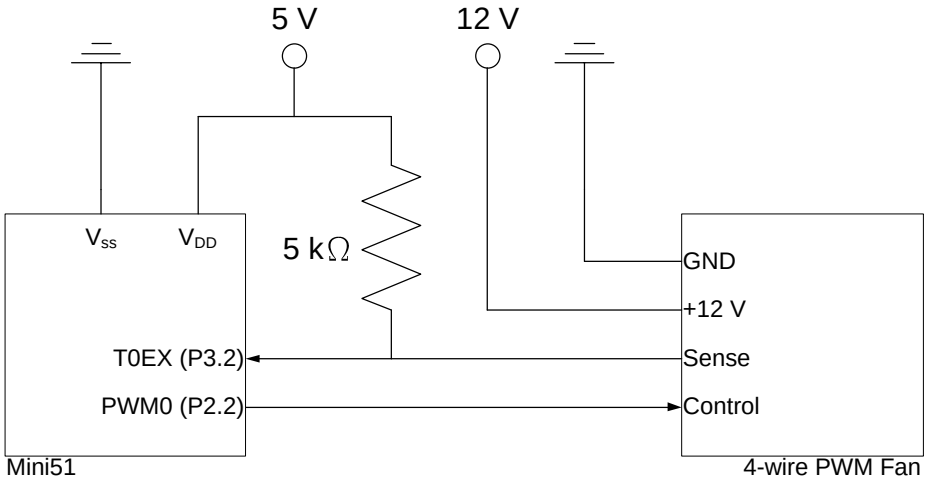
Software Environment

- BSP version
  - Mini51DE Series BSP CMSIS v3.02.000
- IDE version
  - Keil uVersion 5.26

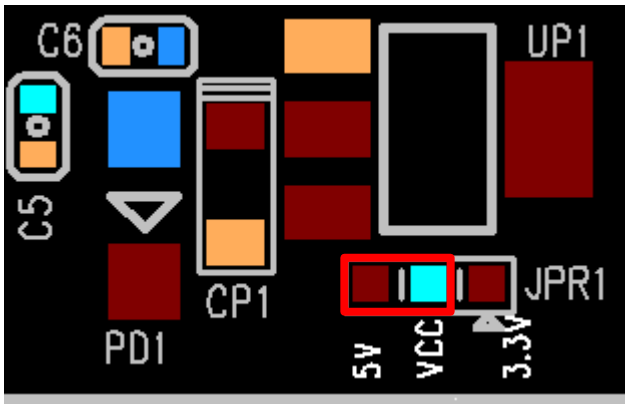
Hardware Environment

- Circuit components
  - NuTiny-SDK-Mini54 V3.1
  - 4-wire PWM Fan
- Diagram

Mini51 uses PWM Channel 0 (P2.2) to connect with 4-wire PWM fan control pin to control fan speed. Mini51 also uses Timer 0 external capture pin (T0EX) (P3.2) to connect with 4-wire PWM fan sense pin to detect fan speed. There should also be a 5 kΩ pull-up resistor on sense pin.



To make NuTiny-SDK-Mini54 operates in 5 V, user also needs to change 0 Ω resistor on JPR1 in Nu-Link-Me from 3.3 V to 5 V.



4 Directory Information

- EC_Mini51_PWM_Fan_Control_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 — Source file of example code

5 How to Execute Example Code

1. Browsing into sample code folder by Directory Information (section 4) and double click Mini51_PWM_Fan_Control.uvproj.
2. Enter Keil compile mode
   a. Build
   b. Download
   c. Start/Stop debug session
3. Connect 12 V power supply to 4-wire PWM fan.
4. Connect 4-wire PWM fan control and sense pin to NuTiny-SDK-Mini54.
5. Enter debug mode
   a. Run
6. Use terminal tool to get fan speed message and input action to change current PWM duty ratio.

6 Revision History

Date	Revision	Description
July 25, 2019	1.00	1. Initially issued.

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.