Mini51DE PWM Trigger ADC and PWM Brake

Example Code Introduction for 32-bit NuMicro® Family

Information

Application	This document describes how to use PWM trigger ADC and PWM brake function to stop PWM output.
BSP Version	Mini51DE Series BSP CMSIS V3.02.000
Hardware	NuTiny-EVB-Mini51_V2.1

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

www.nuvoton.com

1 Function Description

1.1 Introduction

This example code demonstrates two PWM functions. The first one is PWM triggers ADC and the second is PWM brake. PWM sends trigger request to ADC every 1 msec. After receiving the trigger request, ADC converts channel 0. ADC comparator 0 and 1 monitor the conversion result of ADC channel 0. When the result meets the condition of comparator 0 or 1, generates interrupt and prints related message. While the program executes, user can change the input voltage of pin INT1 to generate PWM brake. Then PWM stops output and ADC stops conversion.

1.2 Demo Result

We compile the example code and download it to the NuTiny-EVB-Mini51_V2.1 development board. PWM triggers ADC to convert channel 0 every 1 msec. When the conversion result of ADC channel 0 meets the condition of ADC comparator 0 or 1, prints the interrupt message to UART console, as shown in Figure 1.

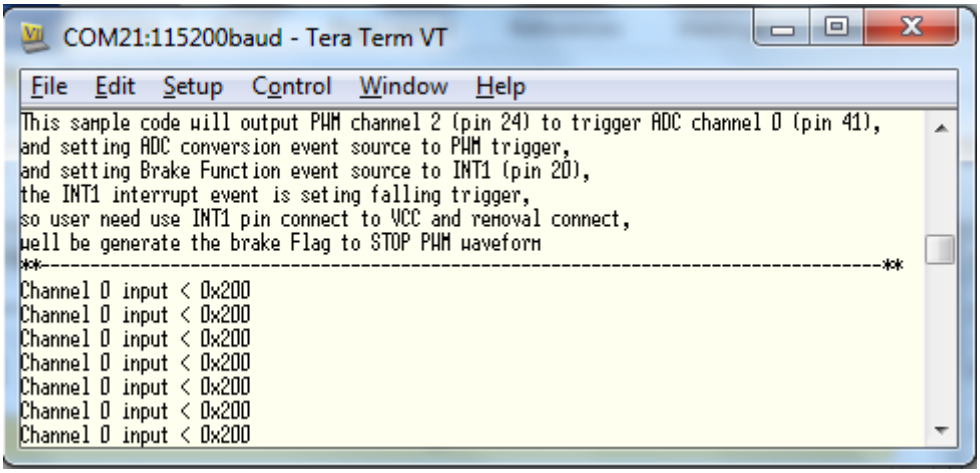


Figure 1 PWM trigger ADC every 1 msec

Pin INT1 works as brake source of PWM. After connecting Pin INT1 to VCC then connecting to VSS, INT1 interrupt occurs and print “INT1” message to UART console. At the same time, PWM brakes and stop output. Since PWM stops, ADC also stops conversion. As shown in Figure 2.

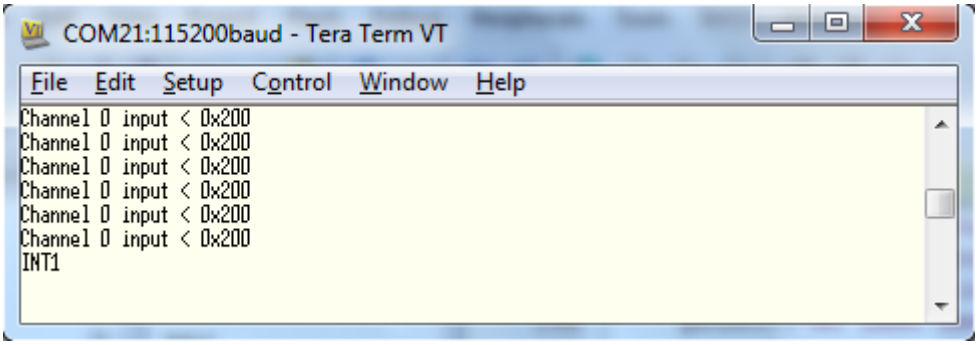


Figure 2 Pin INT1 generates PWM brake and stops ADC conversion

2 Code Description

In this example, we have to configure UART, ADC and PWM clock setting. Following is the related code segment in SYS_Init() of main.c

```
    /* Enable IP clock */
    CLK_EnableModuleClock(UART_MODULE);
    CLK_EnableModuleClock(ADC_MODULE);
    CLK_EnableModuleClock(PWM23_MODULE);

    /* Select IP clock source */
    CLK_SetModuleClock(UART_MODULE, CLK_CLKSEL1_UART_S_XTAL, CLK_CLKDIV_UART(1));
    CLK_SetModuleClock(ADC_MODULE, CLK_CLKSEL1_ADC_S_HIRC, CLK_CLKDIV_ADC(4));
    CLK_SetModuleClock(PWM23_MODULE, CLK_CLKSEL1_PWM23_S_HCLK, 0);
```

In main routine, initializing ADC as following

- Set ADC trigger source to PWM and trigger delay to 1 HCLK
- Configure and enable comparator 0 to monitor channel 0 input less than 0x200 and comparator 1 to monitor channel 0 input greater or equal to 0x200
- Configure and enable both CMP0 and CMP1 interrupt

```
    /* Set ADC trigger source to PWM and trigger delay to 1 HCLK */
    ADC_EnableHWTrigger(ADC, ADC_TRIGGER_BY_PWM, 1);

    /* Enable ADC interrupt */
    ADC_EnableInt(ADC, ADC_ADF_INT);

    /* Clear ADC interrupt */
    ADC_CLR_INT_FLAG(ADC, u32Flag);

    /* Configure and enable Comparator 0 to monitor channel 0 input less than 0x200, match
count 16 */
    ADC_ENABLE_CMP0(ADC, 0, ADC_CMP_LESS_THAN, 0x200, 16);
    /* Configure and enable Comparator 1 to monitor channel 0 input greater or equal to
0x200, match count 16 */
    ADC_ENABLE_CMP1(ADC, 0, ADC_CMP_GREATER_OR_EQUAL_TO, 0x200, 16);

    /* Enable ADC comparator 0 and 1 interrupt */
    ADC_EnableInt(ADC, ADC_CMP0_INT);
    ADC_EnableInt(ADC, ADC_CMP1_INT);
    NVIC_EnableIRQ(ADC_IRQn);
```

ADC_IRQHandler() is ADC interrupt service routine. Whenever ADC interrupt occurs, CPU executes ADC_IRQHandler(). In ADC_IRQHandler(), using ADC_GET_INT_FLAG() to get interrupt status and print related message. Then using ADC_CLR_INT_FLAG() to clear ADC interrupt flag.

```
void ADC_IRQHandler(void)
{
    /* Get ADC comparator interrupt flag */
    u32Flag = ADC_GET_INT_FLAG(ADC, ADC_CMP0_INT | ADC_CMP1_INT);

    if(u32Flag & ADC_CMP0_INT)
        printf("Channel 0 input < 0x200\n");
    if(u32Flag & ADC_CMP1_INT)
        printf("Channel 0 input >= 0x200\n");

    ADC_CLR_INT_FLAG(ADC, u32Flag);
}
```

In main routine, PWM initial codes executes after ADC initialization. Following are PWM initializations.

- Configure the frequency of PWM channel 2 to 1000 Hz
- PWM triggers ADC when PWM channel 2 counter match CNR. Hence, PWM triggers ADC every 1 msec.
- PWM break occurs when the input voltage of pin INT1 has falling edge changes.

```
   /* PWM2 frequency is 1000Hz, duty 50% */
    PWM_ConfigOutputChannel(PWM, 2, 1000, 50);
    PWM_EnableDeadZone(PWM, 2, 200);

    /* Set PWM channel 2 counter matching CNR trigger ADC */
    PWM->TRGCON0 |= PWM_TRGCON0_CNT2TRGEN_Msk;
    PWM_SET_ALIGNED_TYPE(PWM, 2, PWM_CENTER_ALIGNED);

    /* Enable output of PWM channel 2 */
    PWM_EnableOutput(PWM, 1 << 2);

    /* Enable PWM channel 2 period interrupt */
    PWM_EnablePeriodInt(PWM, 2, PWM_PERIOD_INT_MATCH_CNR);
    NVIC_EnableIRQ(PWM_IRQn);

    /* PWM PFBCON: PWMBKO2 Mask */
```

```
    PWM->PFBCON |= PWM_PFBCON_PWMBKO2_Msk;

    /* PWM brake Flag clear */
    PWM->PFBCON |= PWM_PFBCON_BKF_Msk;

    /* PWM BRKIE */
    PWM->PIER |= PWM_PIER_BRKIE_Msk;
    PWM->PHCHGNXT |= PWM_PHCHGNXT_CE1_Msk;

    /* Comparator 0 as fault break 1 source */
    PWM->PFBCON |= PWM_FB1_EINT1;
    PWM->PFBCON &= ~PWM_PFBCON_CPO1BKEN_Msk;

    /* Enable PWM channel 2 counter */
    PWM_Start(PWM, 1 << 2);
```

In PWM interrupt service routine, PWM_IRQHandler(), toggle channel 2 output every 1 second. Clear channel 2 period and trigger ADC interrupt flag.

```
void PWM_IRQHandler(void)
{
    static uint32_t cnt;
    static uint32_t out;

    /* Channel 2 frequency is 1000Hz, every 1 second enter this IRQ handler 1000 times. */
    if(++cnt == 1000)
    {
        if(out)
            PWM_EnableOutput(PWM, BIT2);
        else
            PWM_DisableOutput(PWM, BIT2);
        out ^= 1;
        cnt = 0;
    }
    /* Clear channel 2 period and trigger ADC interrupt flag */
    PWM_ClearPeriodIntFlag(PWM, 2);
    PWM_ClearADCTriggerFlag(PWM, 2, PWM_TRIGGER_ADC_CNTR_IS_CNR);
}
```

In main routine, enable pin INT1(P5.2) interrupt. It works as break source of PWM.

```
/* EN P5.2 INT1 Falling */
    P5->IEN |= 0x04;
    NVIC_EnableIRQ(EINT1_IRQn);
```

In EINT1 interrupt service routine, EINT1_IRQHandler(), a debug I/O P5.5 outputs low. Print a message to UART console. Clear INT1interrupt flag and break flag.

```
void EINT1_IRQHandler(void)
{
    P55=0;  /* debug IO */
    printf("INT1");
    P5->ISRC |= 0x04;   /* CLR INT1 interrupt Flag */
    PWM->PFBCON |= PWM_PFBCON_BKF_Msk;  /* CLR Brake Function Flag */
}
```

3 Software and Hardware Environment

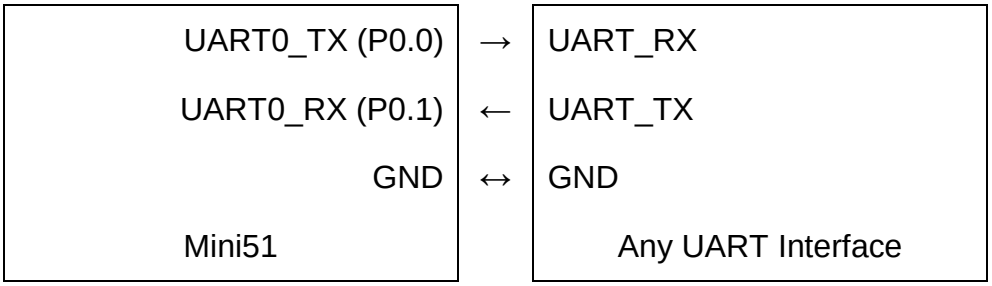
Software Environment

- BSP version
  - Mini51DE Series BSP CMSIS V3.02.000
- IDE version
  - Keil uVersion 5.26

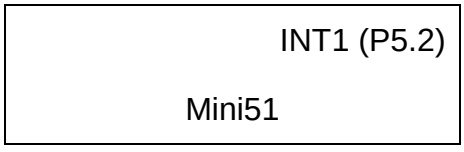
Hardware Environment

- Circuit components
  - NuTiny-EVB-Mini51_V001
- Diagram

We use printf function to output ADC CMP0 and CMP1 interrupt message. The UART0 TX RX pins are P0.0 and P0.1 respectively.



Set P5.2 to INT1 mode for PWM Brake function.



Set P5.3 to AIN0 for ADC channel 0 input pin.

4 Directory Information

- EC_Mini51_PWM_Trigger_ADC_and_PWM_Brake_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 section 4 Directory Information and double click PWM_Trigger_ADC_and_PWM_Brake.uvproj.
2. Enter Keil compile mode
   a. Build
   b. Download
   c. Start/Stop debug session
3. Enter debug mode
   a. Run

6 Revision History

Date	Revision	Description
Jun. 17, 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.