

Chapter 8

PWM Generation using Advanced-control Timer (ADTM)

The Advanced-control timer Module contains a powerful 16-bit auto-reload timer, TIM1, which can be used to measure pulse width or generate pulses, PWM waves, etc. It is used in motor control, power supply, etc.

Main Features of ADTM

The main features of the advanced-control timer TIM1 include.

- 16-bit auto-reload counter supporting incremental counting mode, decremental counting mode and incremental and decremental counting mode.
- 16-bit prescaler with dynamically adjustable crossover coefficients from 1 to 65536.
- Support for **four independent comparison capture channels**.
- Each comparison capture channel supports multiple operating modes, such as: input capture, output comparison, **PWM generation** and single pulse output.
- **Complementary outputs supporting programmable dead time**.
- Support for external signals to control the timer.
- Support for updating the timer after a defined period using a repeat counter.
- Support for resetting the timer or placing it in the OK state using the brake signal.
- Support for the use of DMA in multiple modes.
- Support for incremental encoders.
- Support cascading and synchronization between timers.

PWM Output Mode

PWM output mode is one of the basic functions of the timer. PWM output mode is most commonly used to determine the PWM frequency using the reload value and the duty cycle using the capture comparison register. Set 110b or 111b in the OCxM field to use PWM mode 1 or mode 2, set the OCxPE bit to enable the preload register, and finally set the ARPE bit to enable automatic reload of the preload register. Since the value of the preload register can only be sent to the shadow register when an update event occurs, the UG bit needs to be set to initialize all registers before the core counter starts counting. In PWM mode, the core counter and the compare capture register are always comparing, and depending on the CMS bit, the timer is able to output edge-aligned or center-aligned PWM signals.

Edge alignment: When edge alignment is used, the core counter is incremented or decremented, and in the PWM mode 1 scenario, OCxREF is high when the core counter value is greater than the compare capture register, and low when the core counter value is less than the compare capture register (e.g., when the core counter grows to the value of R16_TIMx_ATRLR and reverts to all zeros).

Central alignment: When using the central alignment modes, the core counter runs in alternating incremental and decremental count modes, and OCxREF makes rising and falling jumps when the values of the core counter and the compare capture register match. However, the comparison flags are set at different times in the three central alignment modes. When using the central alignment modes, it is best to generate a software update flag (set the UG bit) before starting the core counter.

Example Code: For Single Channel PWM Output:

```
#include "debug.h"

/* PWM Output Mode Definition */
#define PWM_MODE1    0
#define PWM_MODE2    1

/* PWM Output Mode Selection */
// #define PWM_MODE PWM_MODE1
#define PWM_MODE PWM_MODE2

/*****
 * @fn      TIM1_OutCompare_Init
 *
 * @brief   Initializes TIM1 output compare.
 *
 * @param   arr - the period value.
 *          psc - the prescaler value.
 *          ccp - the pulse value.
 *
 * @return  none
 */
void TIM1_PWMOut_Init(u16 arr, u16 psc, u16 ccp)
{
    GPIO_InitTypeDef GPIO_InitStructure={0};
    TIM_OCInitTypeDef TIM_OCInitStructure={0};
    TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure={0};

    RCC_APB2PeriphClockCmd( RCC_APB2Periph_GPIOD | RCC_APB2Periph_TIM1,
    ENABLE );

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init( GPIOD, &GPIO_InitStructure );

    TIM_TimeBaseInitStructure.TIM_Period = arr;
    TIM_TimeBaseInitStructure.TIM_Prescaler = psc;
    TIM_TimeBaseInitStructure.TIM_ClockDivision = TIM_CKD_DIV1;
    TIM_TimeBaseInitStructure.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_TimeBaseInit( TIM1, &TIM_TimeBaseInitStructure );

    #if (PWM_MODE == PWM_MODE1)
        TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;

    #elif (PWM_MODE == PWM_MODE2)
        TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM2;

    #endif

    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
    TIM_OCInitStructure.TIM_Pulse = ccp;
    TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
    TIM_OC1Init( TIM1, &TIM_OCInitStructure );
```

```

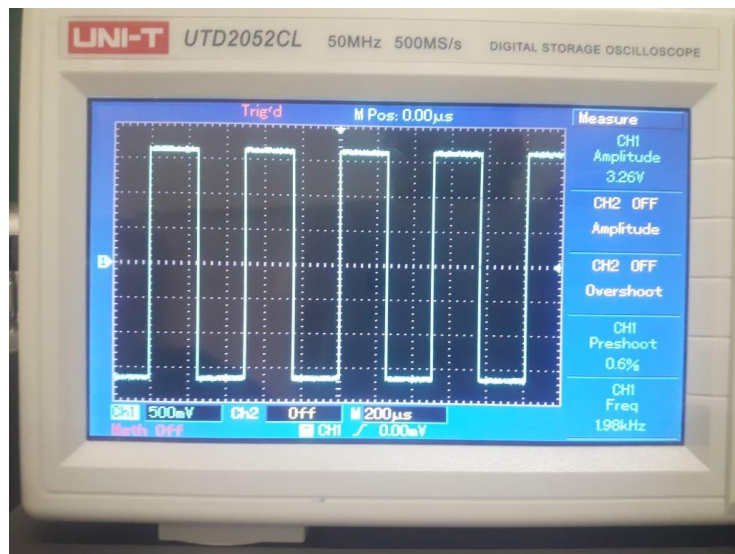
    TIM_CtrlPWMOutputs(TIM1, ENABLE );
    TIM_OC1PreloadConfig( TIM1, TIM_OCPreload_Disable );
    TIM_ARRPreloadConfig( TIM1, ENABLE );
    TIM_Cmd( TIM1, ENABLE );
}
/*****
* @fn      main
*
* @brief    Main program.
*
* @return   none
*/
int main(void)
{
    SystemCoreClockUpdate();
    Delay_Init();
    #if (SDI_PRINT == SDI_PR_OPEN)
        SDI_Printf_Enable();
    #else
        USART_Printf_Init(115200);
    #endif
    printf("SystemClk:%d\r\n",SystemCoreClock);
    printf( "ChipID:%08x\r\n", DBGMCU_GetCHIPID() );

    TIM1_PWMOut_Init( 100, 48000-1, 50 );

    while(1);
}

```

Output:



PWM output at Multiple Channels:

In order to get PWM on Multiple Channels we initialize the required channels. Each channel has its own Capture and Compare Register and a Pre-scaler Register for setting up the duty cycle independently. As the input clock for all the Channels of the same Timer is same so the frequency at all channels will be same.

Code: PWM output at Multiple Channels (Same Duty Cycle):

```
#include "debug.h"

/* PWM Output Mode Definition */
#define PWM_MODE1    0
#define PWM_MODE2    1

/* PWM Output Mode Selection */
#define PWM_MODE PWM_MODE2

/*****
 * @fn          TIM1_PWMOut_Init
 *
 * @brief       Initializes TIM1 output compare.
 *
 * @param       arr - the period value.
 *              psc - the prescaler value.
 *              ccp - the pulse value.
 *
 * @return      none
 */
void TIM1_PWMOut_Init(u16 arr, u16 psc, u16 ccp)
{
    GPIO_InitTypeDef GPIO_InitStructure={0};
    TIM_OCInitTypeDef TIM_OCInitStructure={0};
    TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure={0};

    RCC_APB2PeriphClockCmd( RCC_APB2Periph_GPIOC | RCC_APB2Periph_GPIOD |
                             RCC_APB2Periph_AFIO | RCC_APB2Periph_TIM1, ENABLE
);

    // Remap TIM1
    GPIO_PinRemapConfig(GPIO_FullRemap_TIM1, ENABLE);

    // TIM1_CH1 (PC4)
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    // TIM1_CH2 (PC7)
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    // TIM1_CH3 (PC5)
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
    GPIO_Init(GPIOC, &GPIO_InitStructure);
}
```

```

// TIM1_CH4 (PD4)
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOD, &GPIO_InitStructure);

TIM_TimeBaseInitStructure.TIM_Period = arr;
TIM_TimeBaseInitStructure.TIM_Prescaler = psc;
TIM_TimeBaseInitStructure.TIM_ClockDivision = TIM_CKD_DIV1;
TIM_TimeBaseInitStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM1, &TIM_TimeBaseInitStructure);

#if (PWM_MODE == PWM_MODE1)
    TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;

#elif (PWM_MODE == PWM_MODE2)
    TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM2;

#endif

TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = ccp;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;

// Configure TIM1_CH1 (PC4)
TIM_OC1Init(TIM1, &TIM_OCInitStructure);

// Configure TIM1_CH2 (PC7)
TIM_OC2Init(TIM1, &TIM_OCInitStructure);

// Configure TIM1_CH3 (PC5)
TIM_OC3Init(TIM1, &TIM_OCInitStructure);

// Configure TIM1_CH4 (PD4)
TIM_OC4Init(TIM1, &TIM_OCInitStructure);

TIM_CtrlPWMOutputs(TIM1, ENABLE);

TIM_OC1PreloadConfig(TIM1, TIM_OCPreload_Disable);
TIM_OC2PreloadConfig(TIM1, TIM_OCPreload_Disable);
TIM_OC3PreloadConfig(TIM1, TIM_OCPreload_Disable);
TIM_OC4PreloadConfig(TIM1, TIM_OCPreload_Disable);

TIM_ARRPreloadConfig(TIM1, ENABLE);
TIM_Cmd(TIM1, ENABLE);
}

/*****
* @fn      main
*
* @brief   Main program.
*
* @return  none
*/
int main(void)
{
    SystemCoreClockUpdate();
    Delay_Init();
}

```

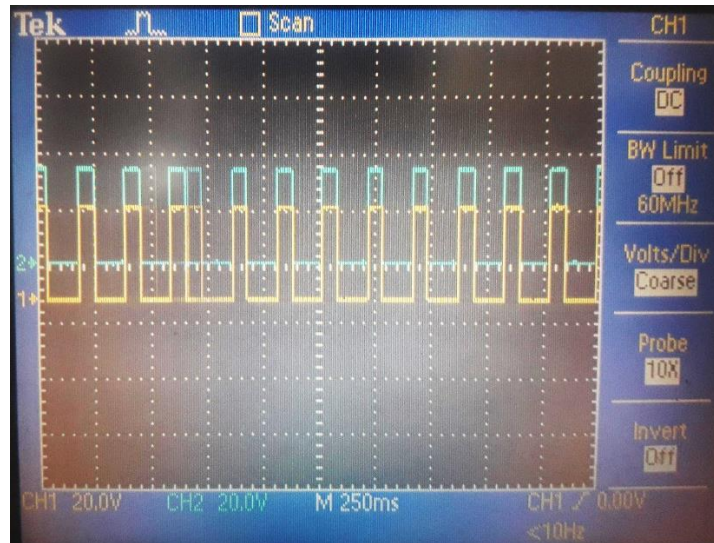
```

#if (SDI_PRINT == SDI_PR_OPEN)
    SDI_Printf_Enable();
#else
    USART_Printf_Init(115200);
#endif
printf("SystemClk:%d\r\n", SystemCoreClock);
printf("ChipID:%08x\r\n", DBGMCU_GetCHIPID());

TIM1_PWMOut_Init(100, 48000-1, 77.7);

while(1);
}

```



Code: PWM Output on Multiple Channels with Different Duty Cycle:

```

#include "debug.h"

/* PWM Output Mode Definition */
#define PWM_MODE1 0
#define PWM_MODE2 1

/* PWM Output Mode Selection */
#define PWM_MODE PWM_MODE2

/*****
 * @fn      TIM1_3Phase_PWMOut_Init
 *
 * @brief   Initializes TIM1 output compare for three-phase PWM.
 *
 * @param   arr - the period value.
 *          psc - the prescaler value.
 *          ccp1 - the pulse value for channel 1.
 *          ccp2 - the pulse value for channel 2.
 *          ccp3 - the pulse value for channel 3.
 *
 * @return  none
 */
void TIM1_3Phase_PWMOut_Init(u16 arr, u16 psc, u16 ccp1, u16 ccp2, u16 ccp3)

```

```

{
    GPIO_InitTypeDef GPIO_InitStructure={0};
    TIM_OCInitTypeDef TIM_OCInitStructure={0};
    TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure={0};

    RCC_APB2PeriphClockCmd( RCC_APB2Periph_GPIOC | RCC_APB2Periph_GPIOD |
                             RCC_APB2Periph_AFIO | RCC_APB2Periph_TIM1, ENABLE
);

    // Remap TIM1
    GPIO_PinRemapConfig(GPIO_FullRemap_TIM1, ENABLE);

    // TIM1_CH1 (PC4)
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    // TIM1_CH2 (PC7)
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    // TIM1_CH3 (PC5)
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    TIM_TimeBaseInitStructure.TIM_Period = arr;
    TIM_TimeBaseInitStructure.TIM_Prescaler = psc;
    TIM_TimeBaseInitStructure.TIM_ClockDivision = TIM_CKD_DIV1;
    TIM_TimeBaseInitStructure.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_TimeBaseInit(TIM1, &TIM_TimeBaseInitStructure);

    #if (PWM_MODE == PWM_MODE1)
        TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
    #elif (PWM_MODE == PWM_MODE2)
        TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM2;
    #endif

    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;

    // Configure TIM1_CH1 (PC4)
    TIM_OCInitStructure.TIM_Pulse = ccp1;
    TIM_OC1Init(TIM1, &TIM_OCInitStructure);

    // Configure TIM1_CH2 (PC7)
    TIM_OCInitStructure.TIM_Pulse = ccp2;
    TIM_OC2Init(TIM1, &TIM_OCInitStructure);

    // Configure TIM1_CH3 (PC5)
    TIM_OCInitStructure.TIM_Pulse = ccp3;
    TIM_OC3Init(TIM1, &TIM_OCInitStructure);

    TIM_CtrlPWMOutputs(TIM1, ENABLE);

    TIM_OC1PreloadConfig(TIM1, TIM_OCPreload_Disable);
    TIM_OC2PreloadConfig(TIM1, TIM_OCPreload_Disable);
    TIM_OC3PreloadConfig(TIM1, TIM_OCPreload_Disable);

```

```

    TIM_ARRPreloadConfig(TIM1, ENABLE);
    TIM_Cmd(TIM1, ENABLE);
}

/*****
 * @fn      main
 *
 * @brief    Main program.
 *
 * @return   none
 */
int main(void)
{
    SystemCoreClockUpdate();
    Delay_Init();
#ifdef (SDI_PRINT == SDI_PR_OPEN)
    SDI_Printf_Enable();
#else
    USART_Printf_Init(115200);
#endif
    printf("SystemClk:%d\r\n", SystemCoreClock);
    printf("ChipID:%08x\r\n", DBGMCU_GetCHIPID());

    TIM1_3Phase_PWMOut_Init(100, 48000-1, 10, 20, 30); // Example duty cycles

    while(1);
}

```

Output for PWM Output on Multiple Channels with Different Duty Cycle:

