Using PWM with the STM32F051C6 microcontroller

1. Definitions

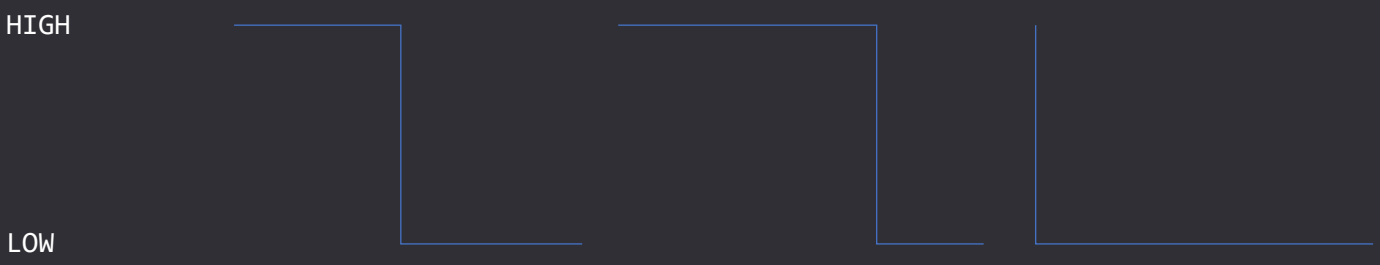
Pulse Width Modulation – varying(modulating) how wide a signal pulse is HIGH

PWM signals are used for transitional control of systems which have discrete states. To demonstrate how this can be useful, let's consider an example that suspiciously sounds 'markety'.

Imagine having a sliding switch that lets control the brightness of a light bulb, instead of one that lets go from OFF to ON and vice versa only. With a PWM signal, we can make your imagination come to live by add a transitional change between these two states, going from OFF to FULLY ON and vice versa, thus giving you control over the brightness of your bulbs. Now you can imagine the benefits that come with having control over the brightness of light bulbs at your dorm or home.

PWM signals provide this control because we can vary the average output. This is achieved by changing the duty cycle of the signal.

Duty cycle – the portion of time for each pulse on a generated signal is HIGH.



Duty cycle: 50% 75% 0%

The average output of the signal is calculated as follows

$$\text{signal output} = \text{DutyCycle} \times (\text{signal max} - \text{signal min})$$

Replacing the light bulb from our earlier example with an LED, and supposing that we use the STM32F051C6 uC to generate the PWM signal, the signal will have max of 3V3 and a min of 0V. Now, if we set the duty cycle to 50%, the STM32F051C6 uC will output 1.65 V to the LED and hence the LED will shine at 50% of its brightness.

To learn how to use PWM with the STM32F052C6 microcontroller, let's start by having a look at which pins have this capability then the relevant registers.

2. PWM pins on the STM32F051C6

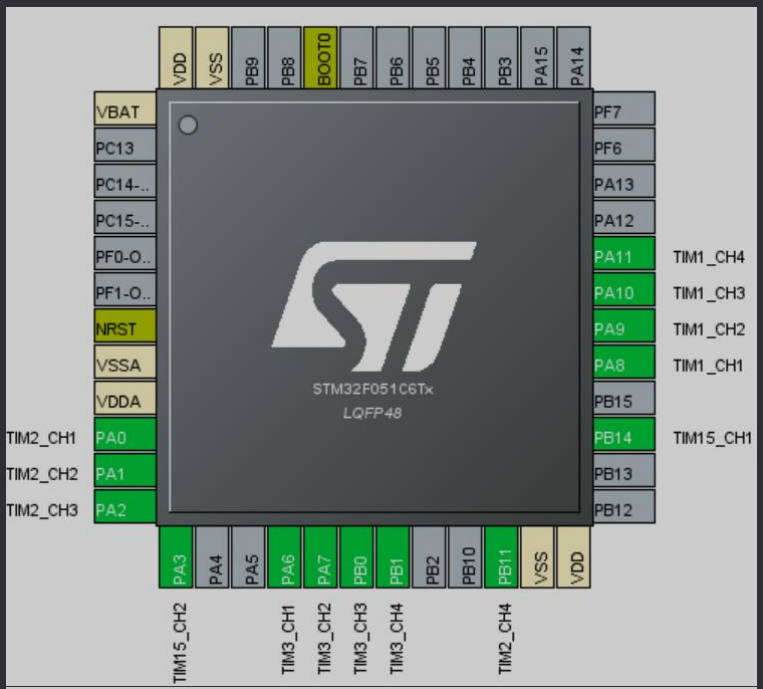


Figure 1 Pins with PWM capability on the STM32F051C6

The pins with PWM capability can be identified with *TIMx_CHx* from the **Pinout descriptions** section on the STMF0xx datasheet, alternatively, STM32CubeMX can be used, as was in generating Figure 1.

3. PWM registers on the STM32F051C6

'Pulse Width Modulation mode allows you to generate a signal with a frequency determined by the value of the TIMx_ARR register and a duty cycle determined by the value of the TIMx_CCRx register. The PWM mode can be selected independently on each channel (one PWM per OCx output) by writing '110' (PWM mode 1) or '111' (PWM mode 2) in the OCxM bits in the TIMx_CCMRx register. You must enable the corresponding preload register by setting the OCxPE bit in the TIMx_CCMRx register, and eventually the auto-reload preload register (in upcounting or center-aligned modes) by setting the ARPE bit in the TIMx_CR1 register. As the preload registers are transferred to the shadow registers only when an update event occurs, before starting the counter, you have to initialize all the registers by setting the UG bit in the TIMx_EGR register. OCx polarity is software programmable using the CCxP bit in the TIMx_CCER register. It can be programmed as active high or active low. OCx output is enabled by a combination of the CCxE, CCxNE, MOE, OSSI and OSSR bits (TIMx_CCER and TIMx_BDTR registers). Refer to the TIMx_CCER register description for more details. In PWM mode (1 or 2), TIMx_CNT and TIMx_CCRx are always compared to determine whether TIMx_CCRx ≤ TIMx_CNT or TIMx_CNT ≤ TIMx_CCRx (depending on the direction of the counter). The timer is able to generate PWM in edge-aligned mode or center-aligned mode depending on the CMS bits in the TIMx_CR1 register.' Extract from STM32F0xx Reference Manual

DISCLAIMER: The list of registers below is not exhaustive but meant for simple PWM Applications. For complex applications see extract above.

Advanced High-Performance Bus RCC -> AHBENR	To enable the Portx on which the PWM pin is located
Mode Register GPIOx -> MODER	To configure the pin on Portx to alternate function mode
Advanced Peripheral Bus RCC -> APB1ENR	To connect clock signal to the TIMERx of the selected pin
Alternate Function Register GPIOx->AFR[0] for pins whose last number < 8 e.g PA6 GPIOx->AFR[1] for pins whose last number > 7 e.g PA11	To enable PWM channel of the TIMER on the selected pin
Prescale and Auto Reload Register TIMx -> PSC TIMx -> ARR	To set the PWM frequency of TIMERx
Capture/Compare Mode Register TIMx -> CCMR1 for channels 1 and 2 TIMx -> CCMR2 for channels 3 and 4	To set the PWM mode for TIMERx
Capture/Compare Register TIMx -> CCRx	To control the duty cycle on channelx of TIMERx
Control Register TIMx -> CR1	To start counter on TIMERx

4. Example

In this example I will do a walkthrough on how to control the brightness of the LED on PB0 of the STM32F051 UCT Dev Board using PWM.

1. Select pin

In this example the pin is preselected, now we check if it has PWM capability. Looking at Figure 1, it can be noticed that PB0 does have PWM functionality through CHANNEL3 of TIMER3(TIM3_CH3).

2. Enable port on which the selected pin is found

PB0 is on Port B, so we'll have to enable Port B using the AHB enable register.

```
RCC -> AHBENR |= RCC_AHBENR_GPIOBEN;
```

3. Configure pin to alternate function mode

To configure the mode of a pin we use the mode register. Each pin can be configured to one of the four I/O modes by writing its corresponding number, see below:

	MODER	
I/O mode	bit 1	bit 0
input	0	0
output	0	1
Alt function	1	0
Analog	1	1

So, to configure PB0 to alternate function mode, we write

```
GPIOB -> MODER |= GPIO_MODER_MODER0_1; // write 1 to bit 1 of PB0
GPIOB -> MODER &= ~(GPIO_MODER_MODER0_0); // write 0 to bit 0 of PB0
```

4. Connect clock signal to TIMER of pin

Now we need to connect clock signal to TIMER3.

```
RCC -> APB13NR |= RCC_APB1ENR_TIM3EN;
```

5. Enable PWM channel connected to pin

Table 14. Alternate functions selected through GPIOA_AFR registers for port A

Pin name	AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7
PA0		USART2_CTS	TIM2_CH1_ETR	TSC_G1_IO1				COMP1_OUT
PA1	EVENTOUT	USART2_RTS	TIM2_CH2	TSC_G1_IO2				
PA2	TIM15_CH1	USART2_TX	TIM2_CH3	TSC_G1_IO3				COMP2_OUT
PA3	TIM15_CH2	USART2_RX	TIM2_CH4	TSC_G1_IO4				
PA4	SPI1_NSS, I2S1_WS	USART2_CK		TSC_G2_IO1	TIM14_CH1			
PA5	SPI1_SCK, I2S1_CK	CEC	TIM2_CH1_ETR	TSC_G2_IO2				
PA6	SPI1_MISO, I2S1_MCK	TIM3_CH1	TIM1_BKIN	TSC_G2_IO3		TIM16_CH1	EVENTOUT	COMP1_OUT
PA7	SPI1_MOSI, I2S1_SD	TIM3_CH2	TIM1_CH1N	TSC_G2_IO4	TIM14_CH1	TIM17_CH1	EVENTOUT	COMP2_OUT
PA8	MCO	USART1_CK	TIM1_CH1	EVENTOUT				
PA9	TIM15_BKIN	USART1_TX	TIM1_CH2	TSC_G4_IO1				
PA10	TIM17_BKIN	USART1_RX	TIM1_CH3	TSC_G4_IO2				
PA11	EVENTOUT	USART1_CTS	TIM1_CH4	TSC_G4_IO3				COMP1_OUT
PA12	EVENTOUT	USART1_RTS	TIM1_ETR	TSC_G4_IO4				COMP2_OUT
PA13	SWDIO	IR_OUT						
PA14	SWCLK	USART2_TX						
PA15	SPI1_NSS, I2S1_WS	USART2_RX	TIM2_CH1_ETR	EVENTOUT				

Table 15. Alternate functions selected through GPIOB_AFR registers for port B

Pin name	AF0	AF1	AF2	AF3
PB0	EVENTOUT	TIM3_CH3	TIM1_CH2N	TSC_G3_IO2
PB1	TIM14_CH1	TIM3_CH4	TIM1_CH3N	TSC_G3_IO3
PB2				TSC_G3_IO4
PB3	SPI1_SCK, I2S1_CK	EVENTOUT	TIM2_CH2	TSC_G5_IO1
PB4	SPI1_MISO, I2S1_MCK	TIM3_CH1	EVENTOUT	TSC_G5_IO2
PB5	SPI1_MOSI, I2S1_SD	TIM3_CH2	TIM16_BKIN	I2C1_SMBA
PB6	USART1_TX	I2C1_SCL	TIM16_CH1N	TSC_G5_IO3
PB7	USART1_RX	I2C1_SDA	TIM17_CH1N	TSC_G5_IO4
PB8	CEC	I2C1_SCL	TIM16_CH1	TSC_SYNC
PB9	IR_OUT	I2C1_SDA	TIM17_CH1	EVENTOUT
PB10	CEC	I2C2_SCL	TIM2_CH3	TSC_SYNC
PB11	EVENTOUT	I2C2_SDA	TIM2_CH4	TSC_G6_IO1
PB12	SPI2_NSS	EVENTOUT	TIM1_BKIN	TSC_G6_IO2
PB13	SPI2_SCK		TIM1_CH1N	TSC_G6_IO3
PB14	SPI2_MISO	TIM15_CH1	TIM1_CH2N	TSC_G6_IO4
PB15	SPI2_MOSI	TIM15_CH2	TIM1_CH3N	TIM15_CH1N

AFRL (AFR[0])

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFR7[3:0]				AFR6[3:0]				AFR5[3:0]				AFR4[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFR3[3:0]				AFR2[3:0]				AFR1[3:0]				AFR0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

AFRH (AFR[1])

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFR15[3:0]				AFR14[3:0]				AFR13[3:0]				AFR12[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFR11[3:0]				AFR10[3:0]				AFR9[3:0]				AFR8[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

PB0 is connected to channel 3 of TIMER 3, so, we have to enable channel 3 of TIMER 3. From Table 15 above, adopted from STM32F0xx datasheet, it can be noticed that TIM3_CH3 is alternate function 1 on PB0. To enable a specific alt function on a pin, we make use of the below;

	bit 3	bit 2	bit 1	bit 0
AF0	0	0	0	0
AF1	0	0	0	1
AF2	0	0	1	0
AF3	0	0	1	1
AF4	0	1	0	0
AF5	0	1	0	1
AF6	0	1	1	0
AF7	0	1	1	1

So, to enable AF1 on PB0, we write:

```
GPIOB -> AFR[0] |= (0b0001 <<(0)*4);
```

6. Set PWM frequency

The STM32F051C6 AHB and APB clock signals can be configured to max frequency of 48MHz, but on startup, the internal RC 8MHz oscillator is selected as the default. We will continue with this example assuming the frequency of 8MHz. To configure the PWM frequency, the formula below is used.

$$f_{PWM} = \frac{8\ MHz}{(TIMx{-}>\ PSC + 1) \cdot (TIMx{-}>\ ARR + 1)}$$

TIMx -> PSC is a 16-bit wide number (all PWM TIMERs) [max = 65 535]
TIMx -> ARR is a 16-bit wide number (all except for TIMER2 which has 32-bit wide ARR)

Suppose we want a PWM frequency of 1kHz(which is pretty common in the DIY community), Then, TIMx -> ARR = 7 999 and TIMx -> PSC = 0(default) should cut it. Therefore, we write:

```
TIM3 -> ARR = 7999;
```

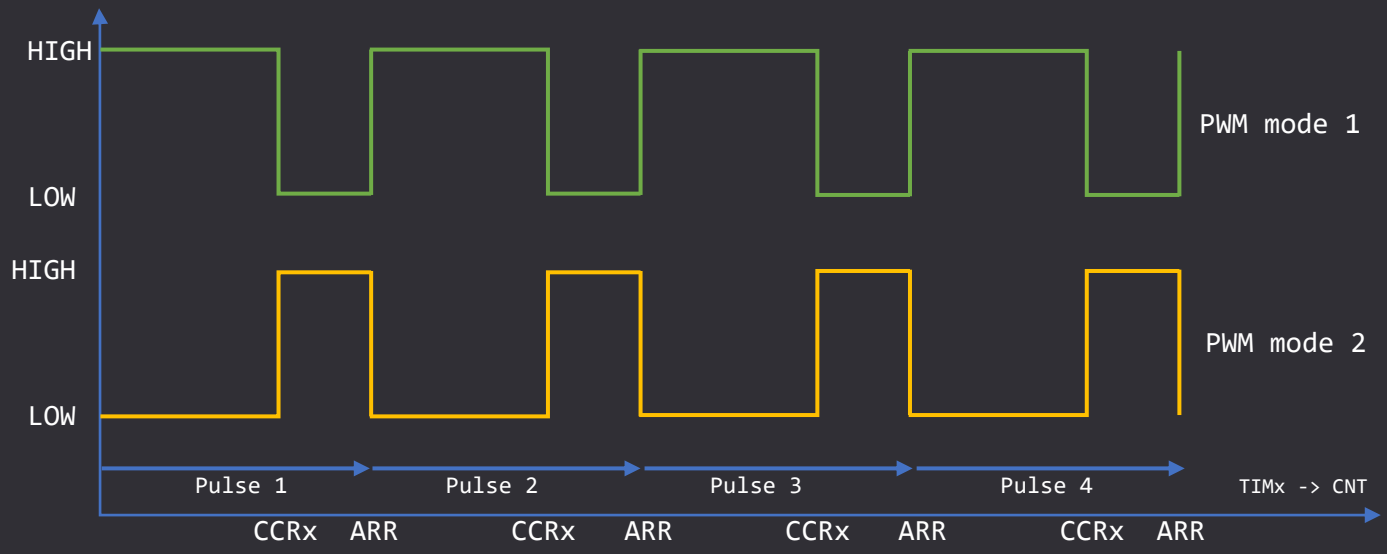
7. Configure PWM mode and enable

	Count direction	Channel active if	Channel inactive if
PWM mode 1	Upcounting	TIMx -> CNT < TIMx -> CCRx	TIMx -> CNT > TIMx -> CCRx
	Downcounting	TIMx -> CNT < TIMx -> CCRx	TIMx -> CNT > TIMx -> CCRx
PWM mode 2	Upcounting	TIMx -> CNT > TIMx -> CCRx	TIMx -> CNT < TIMx -> CCRx
	Downcounting	TIMx -> CNT > TIMx -> CCRx	TIMx -> CNT < TIMx -> CCRx

The diagram below demonstrates the two PWM modes. The duty cycle is given by

$$\text{DutyCycle} = \frac{\text{TIMx} \rightarrow \text{CCRx}}{\text{TIMx} \rightarrow \text{ARR}} \times 100\%$$

Notice in PWM mode 1, the duty cycle is 75% but 25% in PWM mode 2 for the same TIMx -> CCRx and TIMx -> ARR values.



	OCxM		
	bit 2	bit 1	bit 0
PWM mode 1	1	1	0
PWM mode 2	1	1	1

In this example it is sensible to use mode 1, we therefore write:

```
TIM3 -> CCMR2 |= ( TIM3_CCMR2_OC3M_2 | TIM3_CCMR3_OC3M_1 );
```

8. Set the duty cycle

As mentioned in step 7, the duty cycle of the PWM signal is determined using the formula;

$$\text{DutyCycle} = \frac{\text{TIMx} \rightarrow \text{CCRx}}{\text{TIMx} \rightarrow \text{ARR}} \times 100\%$$

In step 6, we set the value of TIM3 -> ARR to 7999 and will remain the same, so a value of TIM3 -> CCR3 will determine the duty cycle. A value of 7999 to TIMx -> CCR3

i.e `TIM3 -> CCR3 = 7999;`

will set the duty cycle to 100%. If TIMx -> CCR3 > TIM3 -> ARR, the duty cycle is still recorded as 100%. Now, whatever value we write to TIMx -> CCR3, will control the brightness of the LED on PB0, but not quite yet. We first have to start the counter.

9. Start the counter

```
TIM3 -> CR1 |= TIM3_CR1_CEN;
```

10. Example code

```
#include <STM32F0xx.h>

void main(void)
{
	RCC -> AHBENR |= RCC_AHBENR_GPIOBEN; // enable Port B

	/* Configure pin PB0 to alt function mode */
	GPIOB -> MODER |= GPIO_MODER_MODER0_1;
	GPIOB -> MODER &= ~GPIO_MODER_MODER0_0;

	RCC -> APB1ENR |= RCC_APB1ENR_TIM3EN; // connect clock signal to TIM3
	GPIOB -> AFR[0] |= (0b0001 >> 4);  // enable PWM Channel 3 of TIM3

	TIM3 -> ARR = 7999; // set PWM frequency to 1kHz
	TIM3 -> CCMR2 |= ( TIM3_CCMR2_OC3M_2 | TIM3_CCMR2_OC3M_1); // select PWM mode 1
	TIM3 -> CCR3 = 0; // initialise duty cycle to 0%
	while(1)
	{
		/* increase brightness from 0 to full then bring it down back, repeat */
		for( int i = 0; i < 8000; i++)
		{
			TIM3 -> CCR3 = i;
		}
		for( int j = 8000; j >=0; j--)
		{
			TIM3 -> CCR3 = j;
		}
	}
}
```

As you will notice, this does not give complete control over controlling the brightness of the LED because you can’t easily and conveniently change the brightness and have it remain there. In future we will learn how to control the duty cycle with one of the potentiometers on the STMF051 UCT Dev Board.