Capítulo 10

Timers no STM32F407

10.1. Introdução aos *timers*

O tempo é uma grandeza extremamente importante no mundo dos sistemas embarcados. Todos os microcontroladores têm um ou mais *timers* (temporizadores) de hardware embutidos, projetados para funcionar paralelamente e independentemente da CPU.

Um *timer* é fundamentalmente um contador, cujo conteúdo é automaticamente incrementado/decrementado a cada pulso de entrada. Se os pulsos recebidos tiverem uma frequência fixa, contá-los se torna uma função de contagem do tempo, o que chamamos de temporização. Geralmente, o oscilador principal do microcontrolador utiliza um cristal de quartzo para a geração do sinal de clock do sistema. Embora esta não seja a solução mais simples para a geração de um sinal de clock, existem muitas razões para usá-la. A frequência de osciladores com cristais de quartzo é precisamente definida e muito estável, de modo que gera pulsos sempre da mesma largura e duração, o que os torna ideais para serem usados como base para a medição do tempo pelos *timers*.

Se for necessário medir o tempo entre dois eventos, por exemplo, basta contar a quantidade de pulsos gerados pelo oscilador no intervalo desejado. Na Figura 1 é mostrado, de forma simplificada, como funciona a medição de intervalos de tempo utilizando um timer interno do microcontrolador. Se o valor armazenado no timer é "*Number A*" (*A*) quando se inicia a medição, e "*Number B*" (*B*), quando se termina, então, o tempo transcorrido é igual ao resultado da subtração *B-A* multiplicado pelo período de oscilação dos pulsos. Por exemplo, quando o oscilador de quartzo funciona a 1 MHz, o contador é incrementado a cada 1 µs, e o tempo transcorrido seria calculado como (*B-A)* 1 µs.*

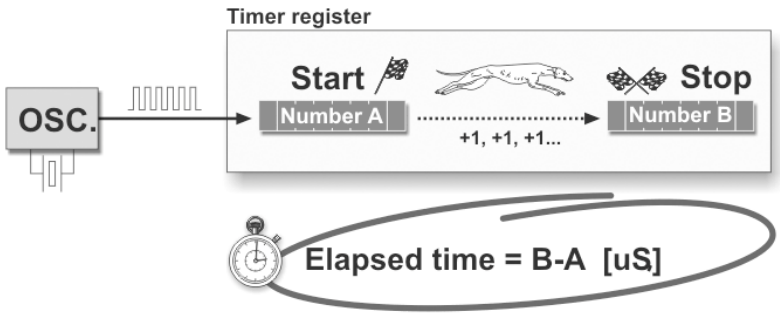

Figura 1 – Operação dos timers internos medindo um intervalo de tempo.

Os *timers* também podem ser configurados para contar pulsos de entrada que não têm uma frequência fixa. Nesse caso, o *timer* está, de fato, agindo como um contador simples. Obviamente, trata-se do mesmo circuito eletrônico, capaz de operar em dois modos diferentes (*timer* ou contador). A única diferença é que, neste último caso, os pulsos contados são oriundos de um pino de entrada do microcontrolador e sua duração (largura) é indefinida. Dessa forma, eles não podem ser usados para medir o tempo, mas são usados para outros fins, tais como contagem de produtos em uma linha de montagem, contagem do número de rotações por segundo do eixo de um motor, a quantidade de vezes que um botão foi pressionado, etc.

Normalmente, além do registrador que contém o valor da contagem propriamente dito, os *timers* têm outros registradores usados para controlar o modo de contagem e indicar o seu status de operação. Os registradores de controle possuem bits de habilitação e configuração para ativar e desativar os recursos do timer. A lógica de controle define o tipo e a versatilidade de um *timer*.

Os *timers* são especificados pela quantidade de bits dos seus registradores de contagem. Eles costumam ter 8, 12, 16, 24 ou 32 bits. Não é incomum encontrar temporizadores de tamanhos diferentes no mesmo microcontrolador. Por exemplo, a maioria dos microcontroladores AVR (utilizados na plataforma Arduino) possui temporizadores de 8 e 16 bits, enquanto os Cortex M4 possuem temporizadores de 16, 24 e 32 bits. Um *timer* de 8 bits pode contar 2^8=256 pulsos de entrada. Um *timer* de 16 bits expande essa capacidade para 2^{16}=65536 pulsos de entrada, enquanto um *timer* de 32 bits conta 2^{32}, ou mais de 4 bilhões de pulsos de entrada. Se o número máximo de contagem for ultrapassado, o temporizador será automaticamente reiniciado e a contagem vai começar do zero novamente. Esta condição é chamada de estouro (*overflow*) do *timer*.

Na Figura 1, por exemplo, se o *timer* tiver 8 bits, é fácil medir intervalos de tempo de até 256 µs. Essa limitação, entretanto, pode ser facilmente superada de diversas maneiras, como por meio do uso de um oscilador mais lento, aplicando-se um divisor ao sinal de clock, usando um *timer* com uma maior quantidade de bits ou usando recursos de interrupção.

O gerenciamento de clock em microcontroladores mais complexos pode ser um pouco complicado, mas, para começar, vamos imaginar que o sinal de clock do sistema (o clock da CPU) é o que aciona os *timers*. Esse sinal de clock pode passar por um divisor (*prescaler*) e a saída do *prescaler* é a entrada de clock no *timer*. Alterando o fator pelo qual o *prescaler* divide o clock recebido, podemos alterar a frequência de clock do *timer*, mesmo que o clock do sistema permaneça o mesmo. A capacidade de alterar a frequência de clock do temporizador permite escolher a frequência mais adequada para os trabalhos de temporização necessários. Um sinal de clock de *timer* mais rápido fornece uma resolução de tempo mais alta, mas um tempo máximo de temporização mais curto, enquanto um sinal de clock de *timer* mais lento fornece uma resolução de tempo mais baixa, mas um tempo máximo de temporização mais longo.

Alguns *prescalers* oferecem apenas alguns fatores de divisão do sinal de clock pré-determinados, para que possamos obter opções de frequência como *F* (o clock do sistema), *F/2*, *F/4*, *F/8*, *F/16*, etc. Outros *prescalers* são completamente configuráveis e podem dividir o clock do sistema por qualquer valor inteiro. Por exemplo, *prescalers* de 16 bits podem dividir o clock do sistema por qualquer valor inteiro entre 1 e 65536. Esses divisores são muito mais flexíveis, mas apresentam uma maior complexidade de uso e operação. Em alguns casos, um *prescaler* é normalmente compartilhado por vários *timers*, de modo que não pode ser independentemente configurado para cada *timer*.

A operação dos *timers* pode gerar diversos eventos de interrupção, pode também fornecer sinais de disparo periódicos para outros periféricos, como conversores ADC, pode ser usada para medição de largura de pulsos de sinais de entrada (captura de entrada) ou a geração de formas de onda de saída (comparação de saída), como sinais PWM. A contagem e a temporização permitem aplicações como controle do brilho de sistemas de iluminação, controle do ângulo do eixo de servomotores, controle de velocidade de rotação de motores, sincronização da comunicação entre dois sistemas, recebimento de dados de sensores que são transmitidos em PWM (*Pulse Width Modulation* - Modulação por Largura de Pulso), criação de temporizadores (como em um aparelho doméstico de microondas) ou simplesmente a criação de um recurso genérico de temporização.

O estouro de um *timer*, por exemplo, pode ser configurado para gerar uma solicitação de interrupção, o que dá possibilidades completamente novas ao projetista. Por exemplo, em um relógio digital, o estado dos registradores usados para a contagem de segundos, minutos, horas ou dias pode ser alterado em uma rotina de atendimento à interrupção gerada precisamente a cada um segundo. A Figura 2 ilustra o exemplo de operação com o uso de interrupção em um *timer*. Se a interrupção por estouro do *timer* estiver habilitada, a rotina de atendimento à interrupção pode incrementar um registrador adicional. O conteúdo desse registrador adicional é usado para controlar determinadas funções no programa principal. A figura mostra que o clock do oscilador é dividido por um fator N pelo *prescaler*, aumentando o tempo de contagem do *timer*.

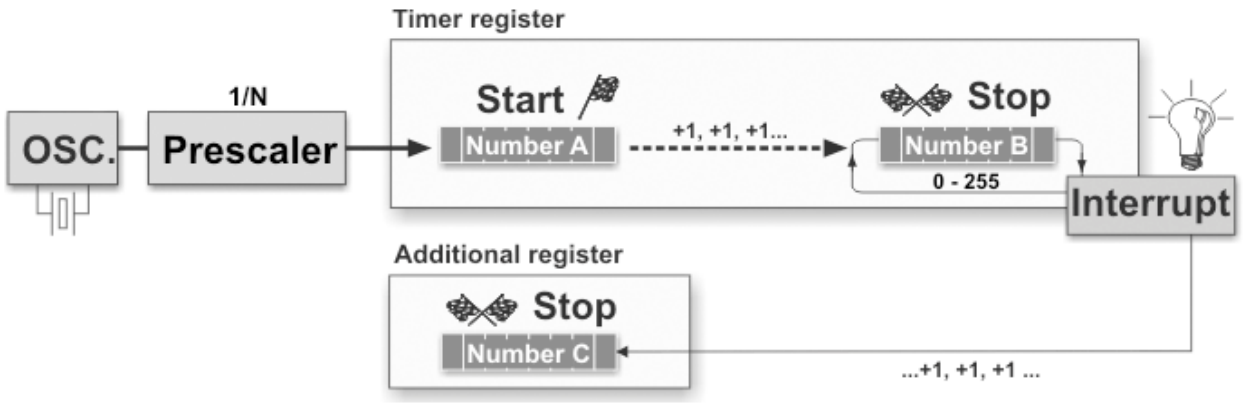

Figura 2 – Uso de prescaler e interrupção no timer interno.

Uma rotina de atendimento à interrupção de um *timer* é um alicerce fundamental para a maioria dos sistemas operacionais. Um *timer* com interrupções periódicas na ordem de 1 ms, por exemplo, é útil para uma variedade de tarefas, pois fornece boa capacidade de resposta às ações humanas, enquanto ainda permite que a maioria dos ciclos da CPU seja dedicada para outros trabalhos.

Outra aplicação útil que se pode fazer com um *timer* é usar a interrupção de estouro para gravar um novo valor inicial no *timer* (o próprio estouro definirá o temporizador como 0). Por exemplo, se em cada estouro escrevermos um valor de 100 em um *timer* de 8 bits, agora, em vez de ter 256 pulsos de clock entre cada estouro, teremos apenas 156 e o *timer* agora conta de 100 a 255, e não mais de 0 a 255.

10.2. *Timers* no STM32F407

O STM32F407 fornece vários *timers* que podem ser usados em uma variedade de aplicações, incluindo dois *timers* PWM para controle de motores, além de dois *timers* de 32 bits de uso geral. Os tipos de *timers* incluem dois de controle avançado, dez de uso geral, dois básicos e dois de vigilância e monitoramento (*watchdog timer*). Todos os *timers* podem ser congelados durante o modo de depuração.

A Tabela 1 sumariza as principais características dos *timers* do STM32F407.

Tabela 1. Comparação das características dos *timers*

Tipo do timer	Timer	Resolução	prescaler	Canais de captura/comparação	Frequência máxima da interface (MHz)	Frequência máxima de contagem (MHz)
Controle avançado	TIM1, TIM8	16	1-65536	4	84	168
Uso geral	TIM2, TIM5	32		4	42	84
	TIM3, TIM4	16		4	42	84
	TIM9	16		2	84	168
	TIM10, TIM11	16		1	84	168
	TIM12	16		2	42	84
	TIM13, TIM14	16		1	42	84
básico	TIM6, TIM7	16		0	42	84

Timers de controle avançado (TIM1, TIM8)

Os *timers* de controle avançado (TIM1, TIM8) de 16 bits podem ser vistos como geradores de sinais PWM trifásicos multiplexados de 6 canais. Eles têm saídas PWM complementares com tempo morto (*dead time*) programável. Podem ser considerados como *timers* completos. Seus 6 canais independentes podem ser usados para:

- captura de entrada
- comparação de saída
- geração de sinais PWM

Se configurados como *timer* padrão de 16 bits, eles têm os mesmos recursos que os *timers* de uso geral. Se configurados como geradores de sinal PWM de 16 bits, eles têm capacidade total de modulação (0-100%).

Os *timers* de controle avançado podem trabalhar juntos com os *timers* de uso geral por meio do recurso *Timer Link* para sincronização ou encadeamento de eventos.

Timers de uso geral

Existem dez temporizadores de uso geral sincronizáveis.

- TIM2, TIM3, TIM4, TIM5

O STM32F407 inclui quatro *timers* de uso geral com todos os recursos: TIM2, TIM3, TIM4 e TIM5. Os *timers* TIM2 e TIM5 são baseados em um contador de recarga automática de 32 bits e em um *prescaler* de 16 bits. Os *timers* TIM3 e TIM4 são baseados em um contador crescente/decrescente de recarga automática de 16 bits e em

um *prescaler* de 16 bits. Todos eles possuem 4 canais independentes para captura de entrada e comparação de saída, saídas PWM ou modo de um pulso.

Os *timers* de uso geral TIM2, TIM3, TIM4, TIM5 podem trabalhar juntos ou com os outros *timers* de uso geral e os *timers* de controle avançado TIM1 e TIM8, através do recurso *Timer Link* para sincronização ou encadeamento de eventos. Eles ainda são capazes de manipular sinais de *encoders* em quadratura (incremental).

• TIM9, TIM10, TIM11, TIM12, TIM13 e TIM14

Esses *timers* são baseados em um contador de recarga automática de 16 bits e em um *prescaler* de 16 bits. O TIM10, o TIM11, o TIM13 e o TIM14 possuem um canal independente, enquanto o TIM9 e o TIM12 possuem dois canais independentes para captura/comparação, PWM ou saída de modo de um pulso. Eles podem ser sincronizados com os *timers* de uso geral com todos os recursos dos timers TIM2, TIM3, TIM4, TIM5. Eles também podem ser usados para geração de base de tempo simples.

***Timers* básicos TIM6 e TIM7**

Esses *timers* são usados principalmente para geração de gatilhos e formas de onda para o DAC. Eles também podem ser usados como uma base de tempo genérica de 16 bits.

O STM32F407 ainda possui os seguintes *timers* de uso específico:

***Timer* cão de guarda independente (*independent watchdog*)**

O *watchdog* independente é baseado em um contador de 12 bits e um *prescaler* de 8 bits. O clock é fornecido por um circuito RC interno independente de 32 kHz e, como opera independentemente do clock principal, pode operar nos modos de espera. Ele pode ser usado como um cão de guarda para resetar o dispositivo quando ocorrer um travamento de software ou como um *timer* genérico para o gerenciamento de tempo limite de alguma aplicação.

***Timer* cão de guarda de janela (*Window watchdog*)**

O *watchdog* de janela é baseado em um contador de 7 bits. Pode ser usado como um cão de guarda para resetar o dispositivo quando ocorrer um problema de travamento de software. É alimentado a partir do clock principal. Possui recursos de interrupção e o contador pode ser congelado no modo de depuração.

***Timer* do sistema (SysTick)**

Este *timer* é dedicado para uso em sistemas operacionais, mas também pode ser usado como um timer de uso geral em aplicações que não exijam um sistema operacional. Ele é baseado em um contador decrescente de 24 bits, com capacidade de carregamento automático, sistema de interrupção quando o contador atinge o valor zero e fonte de clock programável.

10.3 *Timer* SysTick

O *timer* do sistema, também chamado de System Tick, ou SysTick, é usado para gerar interrupções em intervalos fixos regulares programáveis, como ilustrado na Figura 3.

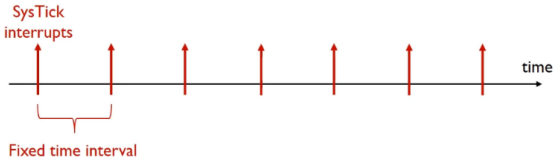

Figura 3 – Uso do SysTick para geração de interrupções em intervalos fixos regulares e programáveis.

Esse temporizador é integrado como parte do NVIC e, se habilitado, pode gerar a exceção SysTick (exceção número #15). O temporizador SysTick é um contador decrescente simples de 24 bits e pode ser utilizado para medição de tempo, como na criação de funções de atraso, ou como base de tempo para execução de tarefas periódicas, como na verificação do status de operação de periféricos.

Em sistemas embarcados complexos, uma interrupção periódica é necessária para garantir que o *kernel* do sistema operacional possa ser invocado regularmente, por exemplo, para gerenciamento de tarefas e troca de contexto. Isso permite que um processador lide com diferentes tarefas em diferentes intervalos de tempo (*multitasking*). Por exemplo, sistemas operacionais de tempo real (*RTOS – Real Time Operating System*) se baseiam nesse temporizador para realizar o agendamento de mudanças de contexto (*scheduling*).

Na Figura 4 é apresentado o modo de operação do SysTick.

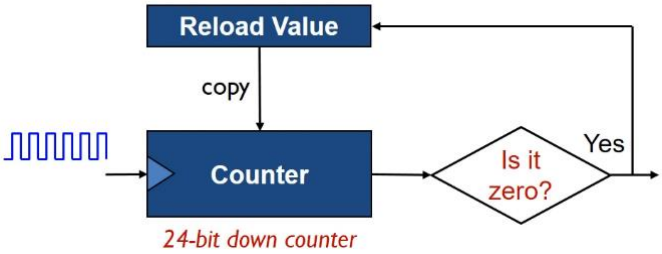

Figura 4 – Modo de operação do SysTick.

Como dito anteriormente, o SysTick é baseado em um contador (*Counter*) decrescente de 24 bits. Quando o contador recebe os pulsos de clock, seu conteúdo é decrementado desde o valor armazenado no registrador *Reload Value* até zero. Depois que o contador atinge o valor zero, o conteúdo do registrador *Reload Value* é copiado automaticamente no contador, a contagem inicia novamente e o processo se repete indefinidamente, até que o contador seja desabilitado. Se configurada, quando o contador fizer a transição do valor 1 para o valor 0, o SysTick gera uma requisição de interrupção.

O diagrama da Figura 5 mostra um exemplo de como o valor do contador muda quando o conteúdo de *Reload Value* é 6. O valor a ser recarregado no contador é armazenado no registrador SysTick_LOAD. O contador conta de forma decrescente desde 6 até 0. O contador tem um total de 7 valores únicos de contagem. Quando o contador faz a transição do valor 1 para 0, o timer pode gerar uma requisição de interrupção, se configurada. No ciclo de clock seguinte, o contador é reinicializado com 6 e nos ciclos de clock subsequentes o processo de contagem decrescente se repete.

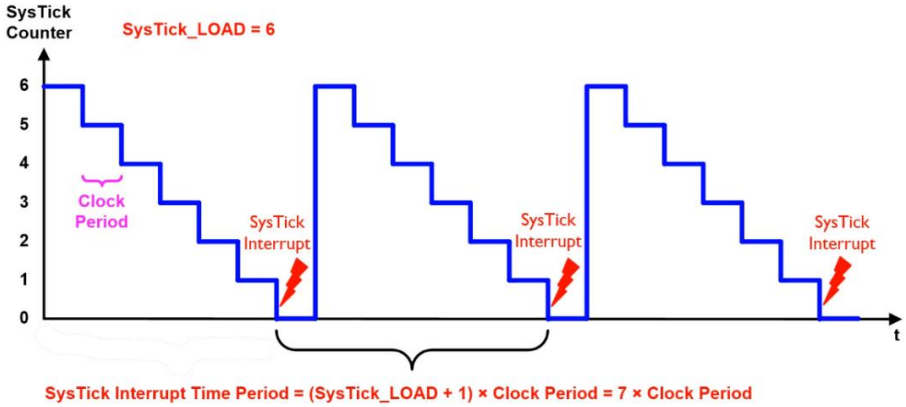

Figura 5 – Comportamento do contador quando o conteúdo de Reload Value é 6.

Na Figura 5 é possível verificar que o intervalo entre as requisições de interrupção do SysTick é:

$$SysTick\ Interrupt\ Time\ Period = (SysTick\_LOAD + 1) * Clock\ Period$$

A operação do SysTick é configurada pelo registrador de controle SysTick_CTRL, cuja organização é mostrada na Figura 6. Nesse registrador, apenas 4 bits são utilizados [16 e 2:0], incluindo um bit de status e três bits de controle. Quando setados, o bit 0 (ENABLE) habilita a contagem e o bit 1 (TICKINT) habilita a requisição de interrupções. O bit 2 (Clock Source) seleciona a fonte de clock do contador e o bit 16 (COUNTFLAG) é setado automaticamente pelo hardware quando a contagem muda de 1 para 0, podendo gerar uma requisição de interrupção se o bit 1 estiver setado.

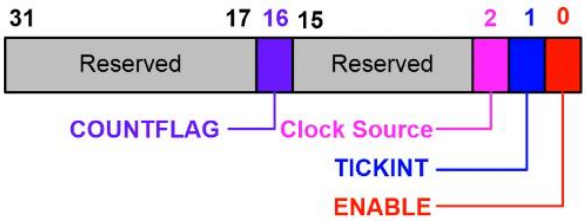

Figura 6 – Registrador SysTick_CTRL.

O diagrama de blocos da Figura 7 mostra o hardware completo do SysTick. Se o bit 2 (*Clock Source*) estiver setado, o clock do barramento AHB (máximo de 168 MHz no STM32F407) é selecionado como fonte de clock do SysTick. Se esse bit estiver resetado, o clock do contador é o clock do barramento AHB dividido por 8.

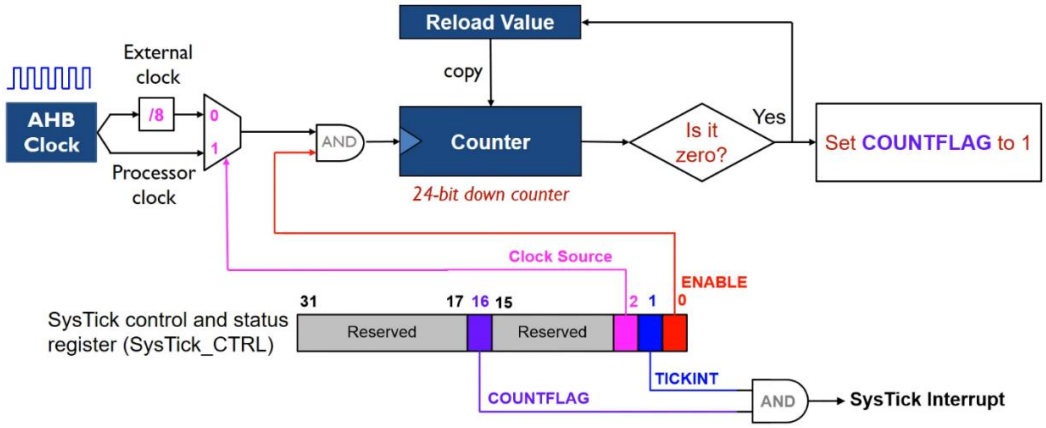

Figura 7 – Diagrama de blocos do timer SysTick.

A seguir, é mostrado um exemplo de código de configuração do SysTick para gerar interrupções periódicas a cada *ticks* pulsos de clock do barramento AHB.

```
void SysTick_Initializer(uint32_t ticks)
{
  SysTick->CTRL = 0;              //desabilita o SysTick
  SysTick->LOAD = ticks-1;        //carrega o registrador Reload Value
  SysTick->VAL = 0;               //zera a contagem do contador
  SysTick->CTRL = 0b111;          //liga o Systick, habilita a interrupção e seleciona
                                  //a fonte de clock como AHB Clock
}
```

A função de atendimento à interrupção do SysTick é definida no CMSIS como:

```
void SysTick_Handler(void)
{
  //código a ser executado periodicamente
}
```

10.4 *Timer* TIM2

O *timer* TIM2 é um contador crescente/decrescente de contagem livre. O *timer* conta continuamente até que seja desabilitado. O processo de contagem reinicia automaticamente quando o contador chega a zero, no modo de contagem decrescente, ou quando atinge um valor limite máximo, no modo de contagem crescente. O software pode selecionar a frequência de contagem para que o incremento ou decremento automático ocorra a uma velocidade desejada.

O *timer* TIM2 consiste em um contador de recarga automática de 32 bits acionado por um *prescaler* programável. Ele pode ser usado para uma variedade de finalidades, incluindo a medição da duração de pulsos de entrada (captura de entrada) ou a geração de formas de onda de saída (comparação de saída e PWM). O *timer* é completamente independente e não compartilha nenhum recurso com outros *timers*.

As principais características do *timer* TIM2 incluem:

- Recarregamento automático de 32 bits;
- Contagem crescente, decrescente ou crescente/decrescente;
- *Prescaler* programável de 16 bits (divide a frequência de clock por qualquer fator entre 1 e 65536);
- 4 canais independentes para captura de entrada, comparação de saída e geração de sinais PWM;
- Circuito de sincronização para controlar o *timer* com sinais externos e interconectar vários *timers*;
- Eventos de interrupção no estouro, inicialização (por software ou gatilho interno/externo), disparo (partida e parada), captura de entrada, comparação de saída;
- Suporte a circuitos incrementais (*encoder* em quadratura).

Na Figura 8 é mostrado o diagrama de blocos do *timer* TIM2.

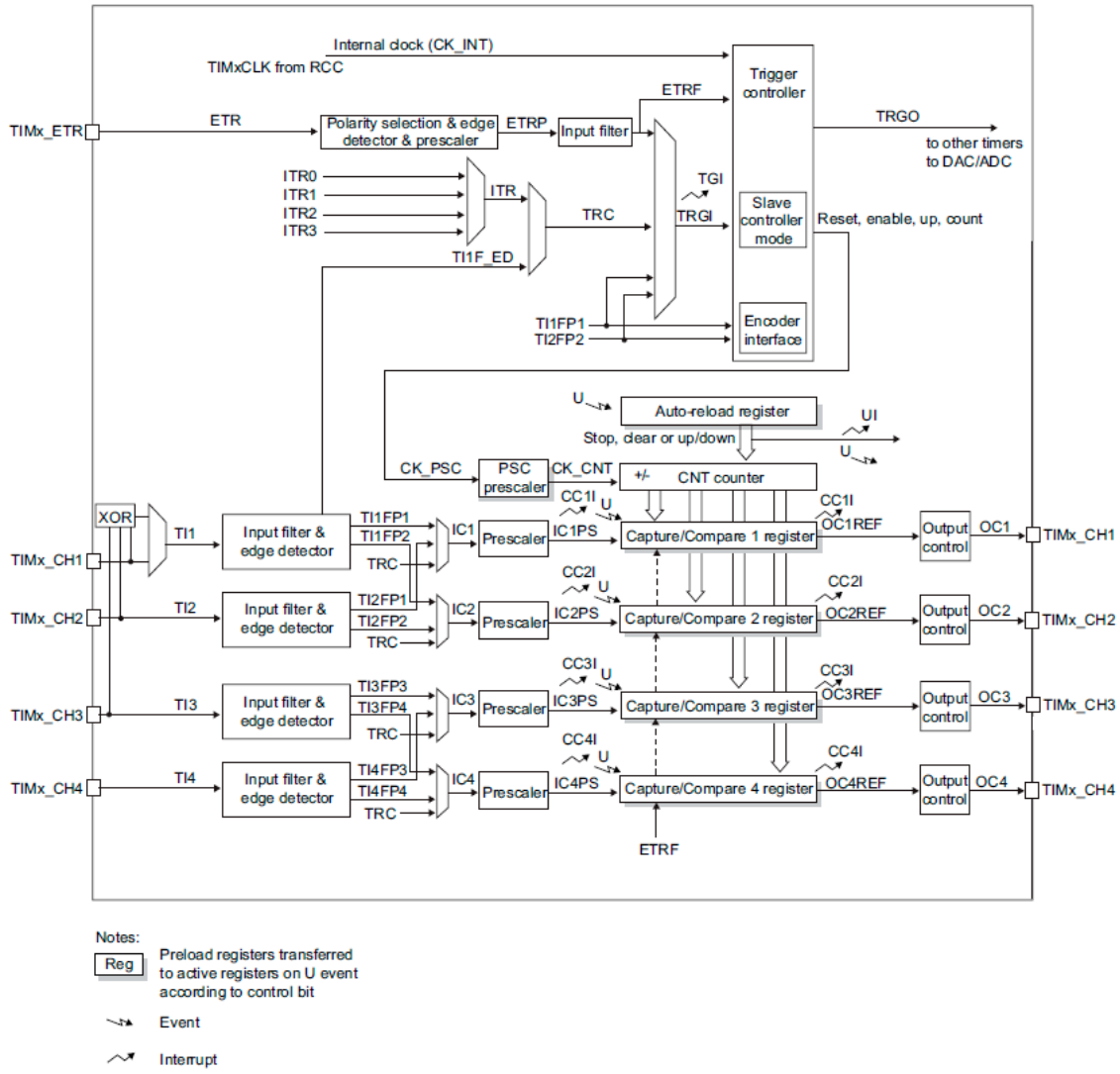

Figura 8 – Diagrama de blocos do timer TIM2.

O bloco principal do *timer* programável é um contador crescente/decrescente de 32 bits (*CNT counter*) com o seu registrador de recarga automática (*Auto-reload register - ARR*). O clock do contador pode ser dividido por um *prescaler* (*PSC prescaler*) de 16 bits. O contador, o registrador de recarga automática e o registrador do *prescaler* podem ser gravados ou lidos por software, mesmo quando em execução.

Assim, a unidade de base temporal inclui:

- Registrador do contador (CNT)
- Registrador do *prescaler* (PSC):
- Registrador de recarga automática (ARR)

O registrador de recarga automática possui pré-carregamento. Escrever ou ler no registrador de recarga automática acessa, na verdade, o registrador de pré-carregamento. O conteúdo do registrador de pré-carregamento é transferido permanentemente para o registrador de recarga automática em cada evento de atualização (*Update Event - UEV*). Um *update event* pode ser disparado por software ou causado por um *overflow* ou *underflow*, que serão abordados mais adiante.

A entrada de clock do contador (CK_CNT) é alimentada pela saída do prescaler, que é acionada apenas quando o bit de habilitação do contador (CEN) no registrador CR1 do módulo TIM2 está setado.

O *prescaler* pode dividir a frequência de entrada (CK_PSC) por qualquer fator entre 1 e 65536. Ele é baseado em um contador de 16 bits controlado por um registrador também de 16 bits (TIM_PSC). Pode ser alterado a qualquer momento pois esse registrador de controle é armazenado em buffer. A nova taxa de divisão do *prescaler* é levada em consideração apenas no próximo evento de atualização.

Se o *timer* opera no modo de comparação de saída (*output compare*), como mostrado na Figura 9, o comparador constantemente compara o valor do contador com um valor constante e gera um sinal de saída ou uma solicitação de interrupção se os valores forem iguais. O software pode programar o valor constante para controlar a periodicidade das solicitações de interrupção ou do sinal de saída.

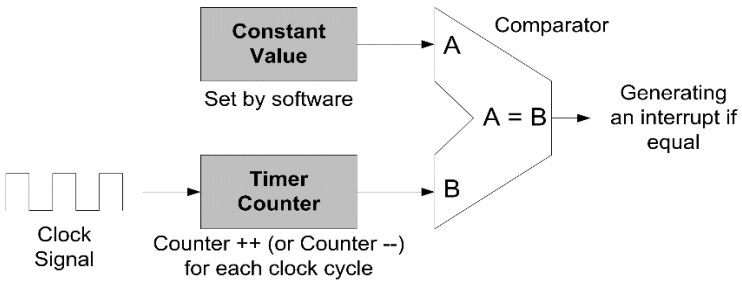

Figura 9 – Timer usado no modo de comparação de saída.

Se o *timer* opera no modo de captura de entrada (*input capture*), como mostrado na Figura 10, o hardware automaticamente captura e armazena o valor do contador em um registrador especial (chamado CCR – *Capture/Compare Register*) e gera uma solicitação de interrupção quando o evento monitorado ocorre no circuito detector de borda (*Edge Detector*). Tipicamente, a rotina de atendimento à interrupção precisa copiar o valor do registrador CCR para uma variável do usuário que armazena o tempo de eventos passados. Então, o software deve calcular a diferença entre dois valores registrados para encontrar o tempo transcorrido entre os dois eventos.

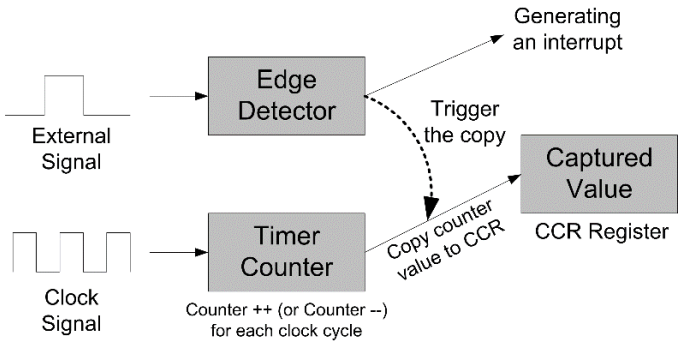

Figura 10 – Timer usado no modo de captura de entrada.

O *timer* TIM2 tem três diferentes modos de contagem: contagem crescente, decrescente e contagem alinhada ao centro (*center-aligned*). A contagem do *timer* nos diferentes modos pode ser representada pelos gráficos mostrados na Figura 11.

• No modo de contagem crescente, o contador inicia a contagem em 0 e vai até o valor armazenado no registrador ARR, então reinicia a contagem;
• No modo de contagem decrescente, o contador inicia a contagem a partir do valor armazenado no registrador ARR, vai decrementando até atingir 0 e em seguida reinicia a partir do valor de ARR;
• No modo de contagem alinhada ao centro, que executa a contagem crescente e decrescente alternadamente, o contador inicia a contagem a partir de 0, é incrementado até atingir o valor armazenado em ARR e, em seguida, é decrementado até 0, reiniciando o processo de contagem.

O período de contagem é controlado pela frequência de clock do contador (CK_CNT) e pelo valor armazenado no registrador ARR. Para as contagens crescente e decrescente, o período de contagem nos modos crescente ou decrescente é:

$$Período = (1 + ARR) * \frac{1}{CK\_CNT}$$

Para o modo de contagem alinhado ao centro, o período da forma de onda triangular é:

$$Período = 2 * ARR * \frac{1}{CK\_CNT}$$

É importante destacar que os gráficos da Figura 11 não são formas de onda de sinais gerados em nenhum pino de saída ou em qualquer periférico interno, mas apenas ilustram o comportamento da contagem no registrador CNT.

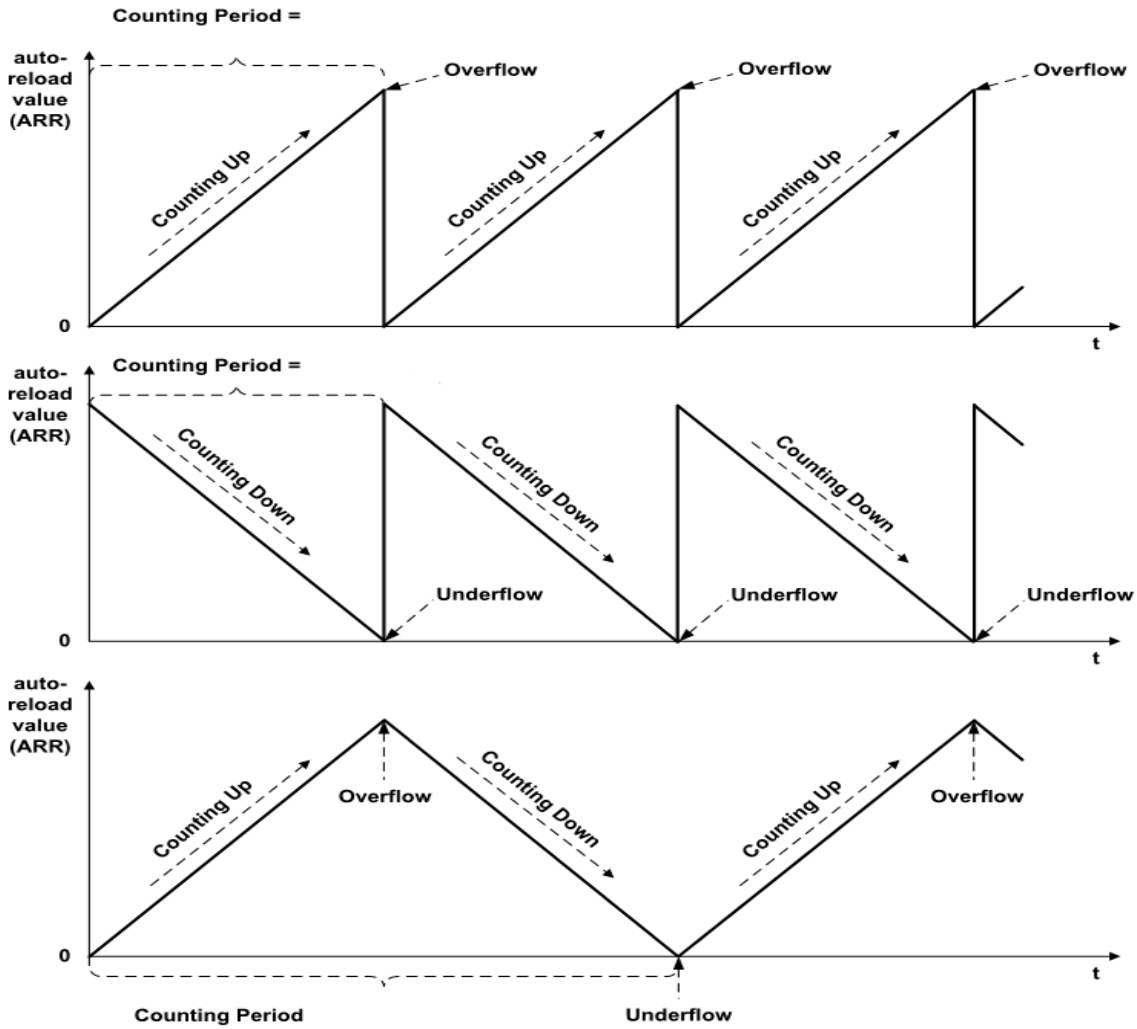

Figura 11 – Modos de contagem do timer TIM2.

O *timer* TIM2 tem dois eventos de atualização (*update events*): *overflow* e *underflow*. Como se vê na Figura 11, no modo de contagem crescente, o *overflow* ocorre quando o valor do contador muda do valor máximo para 0. No modo de contagem decrescente, o *underflow* ocorre quando o contador é recarregado com o valor do registrador ARR. No modo de contagem alinhada ao centro, o *underflow* e *overflow* ocorrem alternadamente.

Ao usar o *timer* para medir longos intervalos de tempo, o software deve considerar o *overflow* e o *underflow* para evitar a subestimação do intervalo medido. A rotina de atendimento à interrupção pode verificar os bits de status apropriados no registrador de status do *timer* para contar quantas vezes o *underflow* ou *overflow* ocorreram.

O sinal de clock do TIM2 é oriundo do clock dos *timers* do barramento APB1 (84MHz), a menos que seja escolhida outra fonte pelos bits TS[2:0] no registrador SMCR (*Slave Mode Control Register*) do módulo TIM2. As fontes alternativas são:

• Modo de clock externo 1: pino de entrada externo (TIx)
• Modo de clock externo 2: entrada de disparo externo (ETR), oriundo de outro *timer*.

Timer em contagem livre

A maneira mais simples de utilizar um *timer* é fazê-lo contar livremente e usar sua contagem como base para temporização de eventos. Esse tipo de configuração permite criar bases de tempo precisas e consistentes para a concepção de funções de atraso (*delays*). Nessa configuração, nenhum recurso de interrupção ou de disparos especiais é necessário. A configuração de um *timer* para esse fim geralmente inclui, mas não se limita, a:

• Selecionar uma fonte de clock
• Definir um fator de divisão para o *prescaler*
• Configurar os bits de controle para especificar o modo de operação
• Ativar interrupções, caso sejam usadas
• Configurar condições de disparo, caso o timer esteja acionando um outro periférico
• Configurar conexões de I/O se o contador estiver conectado a um pino de entrada ou saída.

No código abaixo, é mostrado um exemplo de configuração para usar o *timer* TIM2 para contar livremente. A configuração seleciona o clock interno (2*APB1 = 84 MHz) e o divide pelo fator do *prescaler+1* (= 84), o que produz pulsos de clock a cada 1µs (base de tempo) na entrada do contador.

```
void TIM2_Setup()
{
  RCC->APB1ENR |= 1;              //liga o clock da interface do Timer2
  TIM2->CR1 &= ~(1 << 4);         //contagem crescente
  TIM2->PSC = 83;                 //prescaler para pulsos a cada 1uS
  TIM2->EGR = 1;                  //update event para escrever o valor do prescaler
  TIM2->CR1 |= 1;                 //habilita o timer
}
```

A configuração acima permite a concepção das funções de atraso mostradas abaixo. A função `Delay_us()` produz uma espera ociosa pela quantidade de tempo (em microssegundos) especificada como parâmetro. A função `Delay_ms()` produz uma espera ociosa pela quantidade de tempo (em milissegundos) especificada como parâmetro. O atraso em cada função é obtido a partir da leitura do registrador de contagem, CNT. Quando o registrador CNT atinge o valor especificado no parâmetro, a função é encerrada.

```
//Criação de atraso em us
void Delay_us(uint32_t delay)
{
  TIM2->CNT = 0;                  //inicializa o contador com 0
  while(TIM2->CNT < delay);       //aguarda o tempo passar
}

//Criação de atraso em ms
void Delay_ms(uint32_t delay)
{
  uint32_t max = 1000*delay;
  TIM2->CNT = 0;                  //inicializa o contador com 0
  while(TIM2->CNT < max);         //aguarda o tempo passar
}
```

10.5 Modulação por largura de pulso (*PWM-Pulse Width Modulation*) e *timers*

Comumente conhecida por sua sigla em inglês, PWM (*Pulse-Width Modulation*), a modulação de um sinal por largura de pulso é uma técnica digital que envolve a modulação da razão cíclica de um sinal digital (*duty cycle*) para transportar qualquer informação sobre um canal de comunicação ou controlar o valor da potência entregue a uma carga.

A maneira mais simples de se controlar a potência aplicada a uma carga é por meio da sua associação em série com um reostato (resistor variável), conforme mostra a Figura 12. Variando-se a resistência do reostato, podem-se modificar a corrente e a tensão na carga e, portanto, a potência aplicada a ela.

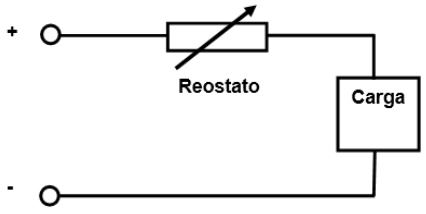

Figura 12 – Controle de potência em uma carga com o uso de reostato.

A grande desvantagem deste tipo de controle, denominado "linear", é que a queda de tensão no reostato multiplicada pela corrente que ele controla representa uma grande quantidade de energia dissipada em forma de calor. O elemento de controle passa a dissipar mais potência que a aplicada na própria carga em determinadas condições. Além dessa perda ser inadmissível, essa abordagem exige que o componente usado no controle seja capaz de dissipar elevadas potências, o que o torna grande e caro.

Uma outra opção de controle linear de potência se dá pela substituição do reostato por transistores, ou circuitos integrados, polarizados adequadamente de forma a variar a potência na carga pelo controle direto da corrente de comando. Mesmo assim, a potência dissipada pelo dispositivo que controla a corrente principal ainda pode ser bastante elevada. Esta potência depende da corrente e da queda de tensão no dispositivo e, da mesma forma, em certas condições, pode ser maior que a própria potência aplicada à carga.

Na eletrônica moderna, é fundamental que se busque um rendimento com pequenas perdas e a ausência de grandes dissipadores que ocupam muito espaço, principalmente quando circuitos de alta potência estão sendo controlados. Dessa forma, são necessárias outras configurações de maior rendimento, como as que fazem uso da tecnologia de chaveamento por PWM, empregada em inversores para carros elétricos, inversores de frequência e fotovoltaicos, fontes chaveadas, controle de brilho e iluminação em monitores de vídeo , e muitas outras aplicações.

Para entendermos como funciona essa tecnologia, partimos do circuito mostrado na Figura 13, formado por um interruptor eletrônico de ação muito rápida e uma carga.

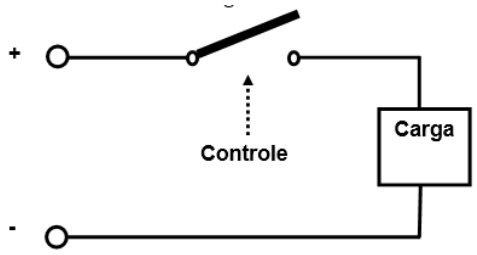

Figura 13 – Controle de potência em uma carga com o uso de interruptor eletrônico.

Quando o interruptor está aberto não há corrente na carga e a potência aplicada é nula. No instante em que o interruptor é fechado, a carga recebe a tensão total da fonte e a potência aplicada é máxima. Então, como fazer para obter uma potência intermediária aplicada à carga, digamos 50%? Uma ideia é fazer com que a chave seja aberta e fechada rapidamente de modo que a cada ciclo ela esteja 50% do tempo aberta e 50% do tempo fechada. Isso significa que, em média, teremos metade do tempo com tensão na carga e metade do tempo sem tensão, conforme mostrado na Figura 14.

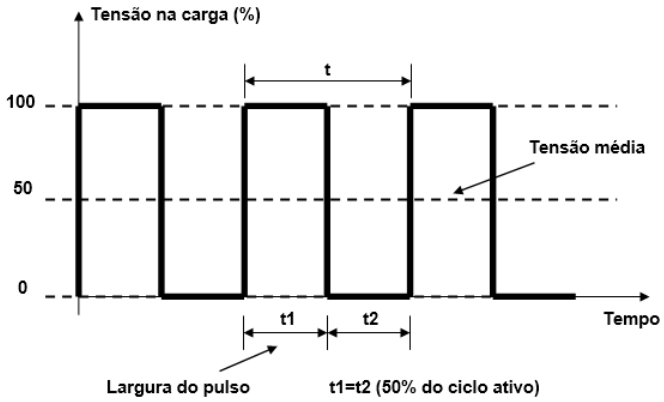

Figura 14– Princípio de funcionamento do PWM.

A tensão média aplicada à carga, é, neste caso, 50% da tensão de entrada. Veja que o interruptor fechado define uma largura de pulso de tensão pelo tempo em que ele fica nesta condição (t1), e um intervalo entre pulsos pelo tempo em que ele fica aberto (t2). Os dois tempos juntos definem um período (t) e, portanto, uma frequência de chaveamento. A relação entre o tempo de duração do pulso (t1) e a duração de um ciclo completo de operação do interruptor (t) nos define o ciclo ativo (*duty cycle*), que neste caso é de 50%.

O período (t_{PWM}) de um sinal PWM é a soma dos tempos ativo (t1) e inativo (t2), isto é:

$$t_{PWM} = t1 + t2,$$

a frequência do sinal PWM (f_{PWM}) é, então, definida como:

$$f_{PWM} = \frac{1}{t_{PWM}},$$

e o *duty cycle*, por sua vez, é definido como:

$$duty\ cycle = \frac{t1}{t_{PWM}} * 100\%$$

Variando-se a largura do pulso e mantendo o período t constante, teremos ciclos ativos diferentes e, com isso, podemos controlar a tensão e a potência médias aplicadas à carga. Assim, quando a largura do pulso varia de zero até o máximo, a potência também varia. Este é o princípio usado no controle PWM: modulamos a largura do pulso de modo a controlar o ciclo ativo da tensão aplicada a uma carga e, com isso, a potência aplicada a ela. Na Figura 15 são apresentados sinais PWM para diferentes larguras de pulso e diferentes valores médios de tensão (*V average*) aplicada à carga.

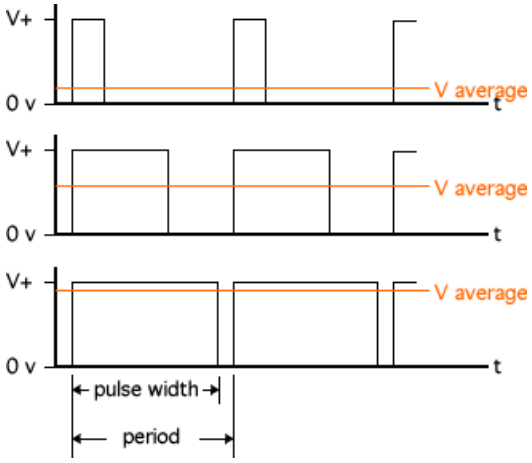

Figura 15 – Sinal PWM com diferentes larguras de pulso.

Na prática, o interruptor é algum dispositivo de estado sólido que possa abrir e fechar o circuito rapidamente como, por exemplo, um transistor bipolar, um FET de potência, um IGBT ou um TRIAC.

Na operação de um controle por PWM, existem diversas vantagens a serem consideradas e alguns pontos para os quais o projetista deve ficar atento para não desperdiçar estas vantagens. Na condição de aberto, nenhuma corrente circula pelo dispositivo de controle e, portanto, sua dissipação é nula. Na condição de fechado, teoricamente, a queda de tensão é nula, e ele também não dissipa nenhuma potência. Isso significa que, na teoria, os controles PWM não dissipam potência alguma e, portanto, consistem em soluções ideais para este tipo de aplicação.

Na prática, entretanto, isso não ocorre. Em primeiro lugar, os dispositivos usados no controle não são capazes de abrir e fechar o circuito instantaneamente. Eles precisam de um tempo para mudar de estado e, sua resistência sobe de um valor muito pequeno até um valor muito alto, ou o contrário. Neste intervalo de tempo, a queda de tensão e a corrente através do dispositivo não são nulas e uma boa quantidade de calor poderá ser gerada, de acordo com a carga controlada. Dependendo da frequência de controle e da resposta do dispositivo usado, uma boa quantidade de energia poderá ser perdida neste processo de comutação. Entretanto, mesmo com este problema, a potência dissipada em um controle PWM ainda é muito menor do que em um circuito de controle linear com potência equivalente (a eficiência de controle de potência com PWM pode chegar a 98%). Dispositivos de chaveamento podem ser suficientemente rápidos para permitir que projetos de controle de potências elevadas sejam implementados sem a necessidade de grandes dissipadores de calor.

Outro problema que poderá surgir vem do fato de que os transistores de efeito de campo, ou os bipolares, usados em comutação não se comportam como resistências nulas quando fechados. Os transistores podem apresentar uma queda de tensão de alguns volts quando saturados. Deve-se observar, em especial, o caso dos FETs de potência que são, às vezes, considerados comutadores perfeitos, com resistências de fração de ohm entre o dreno e a fonte quando saturados ($Rds_{(on)}$), mas, não é exatamente isso que se verifica na prática.

A baixíssima resistência de um FET de potência quando saturado só é válida para uma excitação de porta feita com uma tensão relativamente alta. Assim, dependendo da aplicação, principalmente nos circuitos de baixa tensão, os transistores de potência bipolares ou mesmo os IGBTs podem ser ainda melhores que os FETs de potência.

O controle de potência por PWM possui diversas aplicações práticas, tais como: controle de velocidade de motores, controle de luminosidade de lâmpadas e LEDS, controle do posicionamento de servo motores, controle de temperatura em processos térmicos, sistemas analógicos de telecomunicações, onde a informação modula a largura do pulso, conversores DAC, inversores fotovoltaicos, entre outros.

Módulo de comparação de saída (*compare output*) no STM32F407 para geração de sinais PWM

Na Figura 16 temos o diagrama de blocos do módulo de comparação de saída de um *timer*. O registrador de captura e comparação (CCR) armazena o valor que deve ser comparado com o contador do *timer* (CNT).

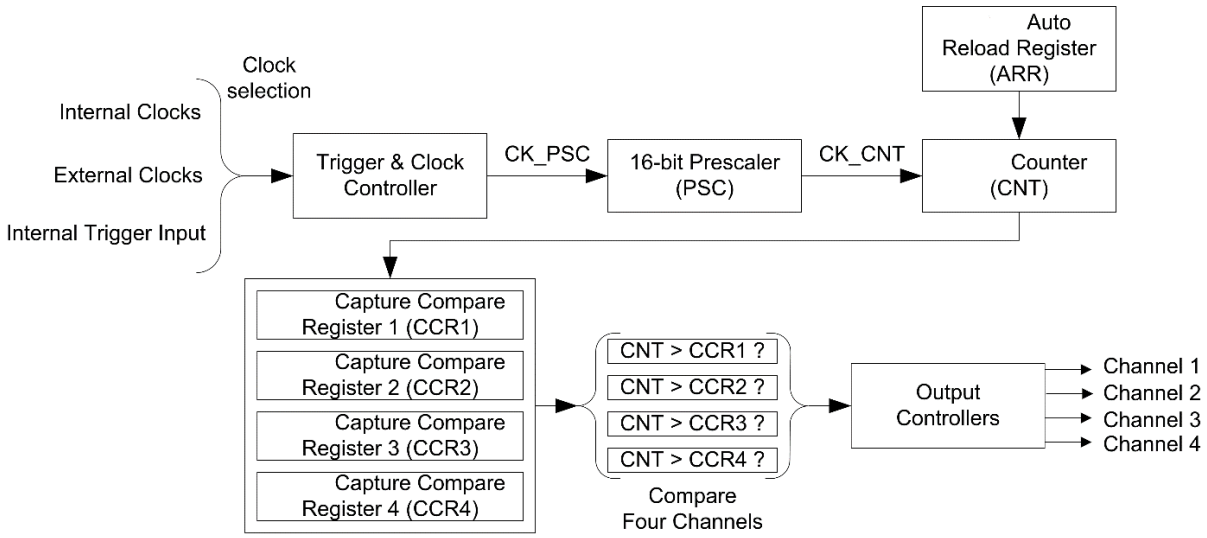

Figura 16 – Diagrama de blocos do módulo de comparação de saída.

O módulo de comparação de saída possui 4 canais que compartilham o mesmo *timer*. Portanto, o hardware compara o valor do registrador do timer (CNT) com 4 registradores CCR simultaneamente e gera 4 sinais de saída independentes, baseado no resultado da comparação. O software pode programar como as saídas se comportam quando o registrador do timer for igual ao valor armazenado no registrador CCR. A saída pode ter diferentes comportamentos, dependendo do modo de comparação de saída escolhido, selecionado nos bits OCM[2:0] do registrador CCMR (*Capture/Compare Mode Register*) do módulo do *timer*, conforme mostrado na Tabela 2.

Tabela 2. Comportamento das saídas do módulo de comparação

Modo de comparação de saída (OCM)	**OCM[2:0]**	**Comportamento da saída**
Modo de temporização	0b000	Sem efeito
Modo ativo	0b001	Lógica HIGH se CNT=CCR
Modo inativo	0b010	Lógica LOW se CNT=CCR
Modo de alternância	0b011	Alterna se CNT=CCR
Modo inativo forçado	0b100	Sempre LOW
Modo ativo forçado	0b101	Sempre HIGH
Modo PWM 1	0b110	Contagem crescente: Lógica HIGH se CNT<CCR, lógica LOW caso contrário Contagem decrescente: Lógica HIGH se CNT≤CCR, lógica LOW caso contrário
Modo PWM 2	0b111	Contagem crescente: Lógica HIGH se CNT≥CCR, lógica LOW caso contrário Contagem decrescente: Lógica HIGH se CNT>CCR, lógica LOW caso contrário

No modo de temporização, o resultado da comparação de CNT com CCR não tem qualquer efeito sobre a saída. Os modos ativo e inativo setam ou resetam a saída, respectivamente, quando o valor de CNT e CCR são iguais. O modo de alternância inverte o estado da saída quando os valores dos registradores são iguais. Os modos forçados ativo e inativo fazem a saída permanecer em HIGH e LOW, respectivamente.

Na Figura 17, por exemplo, é mostrada a saída do canal de referência (OCREF) quando ela é configurada no modo de alternância. Neste exemplo, o contador conta de 0 até o valor armazenado no registrador ARR. O sinal OCREF é sempre ativo em HIGH. Entretanto, o software pode mudar a representação da lógica binária de saída no bit de polaridade no registrador CCER (*Capture/Compare Enable Register*), fazendo com que a saída seja OCREF ou sua negação.

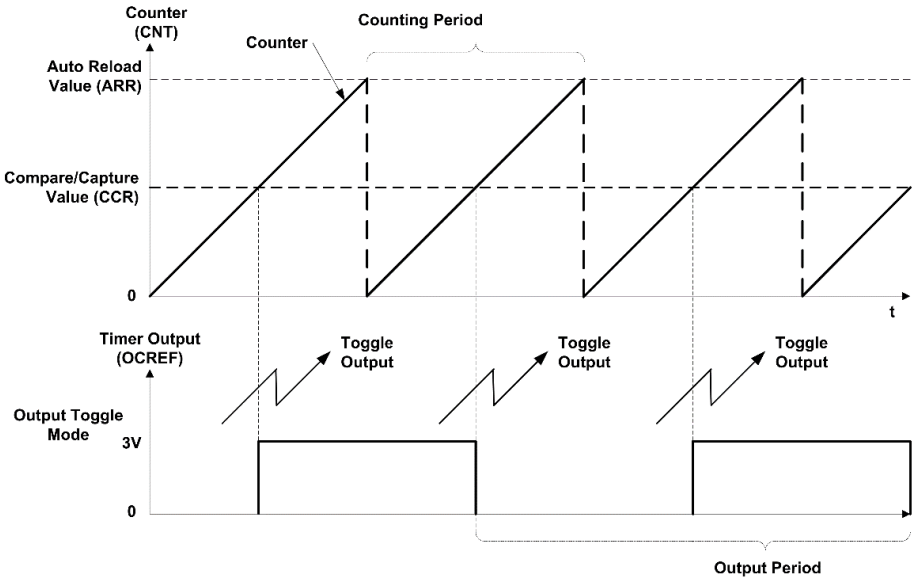

Figura 17 – Sinal de saída OCREF no modo de alternância do módulo de comparação de saída.

Se uma interrupção de *timer* for habilitada, uma solicitação de interrupção é gerada se CNT se iguala a CCR ou então quando CNT tem um evento de *overflow* ou *underflow*. A rotina de atendimento à interrupção do timer deve analisar os bits do registrador de status para determinar que evento ocorreu. O bit UIF é setado quando ocorre um evento de atualização (*overflow* e *underflow*) e o bit CCIF é setado quando CNT=CCR.

Modo de saída PWM

Um sinal de saída PWM pode ser obtido usando o módulo de comparação de saída no STM32F407. Três fatores determinam as características do sinal:

• A comparação entre os registradores CNT e CCR
• O modo de saída PWM
• O bit de polaridade

A saída de cada canal (OCREF) sempre usa o nível HIGH como a saída ativa, conforme apresentado na Tabela 2. Entretanto, o pino de saída do canal pode usar uma lógica inversa dependendo do bit de polaridade.

Cada canal do módulo de comparação de saída pode gerar um sinal PWM. Todos os canais compartilham o mesmo contador e o mesmo registrador ARR. Com isto, todos os sinais PWM produzidos pelo mesmo timer têm o mesmo período e frequência. Entretanto, o *duty cycle* pode ser diferente porque cada canal tem seu próprio registrador CCR. Na Figura 18 são mostrados dois sinais de saída obtidos com modo PWM 2 com *duty cycles* diferentes (1/3 e 1/2), obtidos quando os valores do registrador CCR são diferentes.

Para a contagem crescente ou decrescente, o período do sinal PWM é determinado por:

$$t_{PWM} = (1 + ARR) * t_{timer} \quad ,$$

onde t_{timer} é o período do sinal de clock do timer, enquanto o *duty cycle* pode ser definido por:

$$duty\ cycle = \frac{CCR}{1 + ARR}$$

10.5 Exemplo de configuração do timer 5 para geração de sinais PWM

O trecho de código da página seguinte pode ser usado para configurar o *timer* TIM5 para gerar um sinal PWM (modo PWM 1) no canal de saída 3. Esse canal é conectado ao pino PA2, que deve ser inicialmente configurado no modo de função alternativa (AF), cuja função deve ser o TIM5 (função 2). O *timer* 5 (TIM5) possui as mesmas características do *timer* 2 (TIM2) e, por isso, não serão rediscutidas aqui novamente.

Desejamos ter uma resolução de *duty cycle* de 1%. Isso significa dizer que poderemos variar o valor do ciclo ativo de 0% a 100% em passos de 1%. Para isso, o valor do registrar ARR deve ser feito igual a 99. Além disso, desejamos ter uma frequência PWM igual a 2kHz, o que nos leva a escolher o valor de PSC igual a 419.

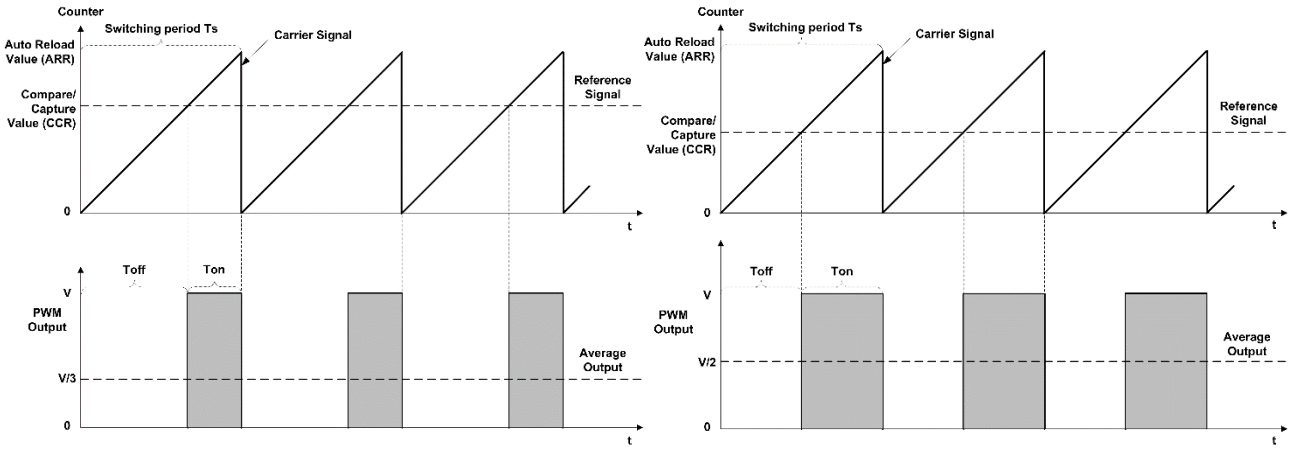

Figura 18 – Sinais PWM com duty cycle de 1/3 (esquerda) e 1/2 (direita).

```
RCC->AHB1ENR |= 1;                    //habilita o clock do GPIOA

GPIOA->MODER |= (0b10 << 4);          //pino PA2 como função alternativa
GPIOA->AFR[0] |= (0b0010 << 8);       //função alternativa 2 (TIM5)

RCC->APB1ENR |= (1 << 3);             //liga o clock do Timer5
TIM5->CR1 &= ~(1 << 4);               //contador crescente
TIM5->ARR = 99;                       //auto reload=99
TIM5->PSC = 419;                      //prescaler para pulsos a cada 5uS
TIM5->CCMR2 |= (0b11 << 5);           //seleciona PWM modo 1
TIM5->CCMR2 |= (1 << 3);              //habilita o pré-carregamento de CCR3
TIM5->CCER |= (1 << 8);               //habilita a saída
TIM5->EGR = 1;                        //update event para escrever PSC e ARR
TIM5->CR1 |= 1;                       //habilita o timer
```

A linha de código abaixo, por sua vez, permite a atualização do *duty cycle* do sinal PWM, uma vez que ela altera o valor do registrador CCR. A variável duty_cycle pode assumir qualquer valor inteiro entre 0 e 100.

```
TIM5->CCR3 = duty_cycle;
```

10.6 Saídas de disparo (TRGO – *Trigger Output*) dos timers no STM32F407

Os *timers* do STM32F407 possuem uma saída de sinal chamada TRGO (*Trigger Output*) que pode ser utilizada para sincronizar a operação de outros *timers* ou disparar diversos eventos em outros periféricos. Por exemplo, pode-se utilizar a saída TRGO de um *timer* como sinal de clock de outro timer por meio do recurso *Timer Link* para encadeamento de temporizadores, aumentando a capacidade de contagem ou temporização.

Uma técnica bem empregada para se obter taxas de amostragem estáveis nos processos de conversão analógico para digital ou digital para analógico é a utilização da saída TRGO de um *timer* para disparar automaticamente uma conversão nos conversores ADC ou DAC. Nesse caso, o conversor é configurado para receber pulsos de disparo de conversão de um *timer* específico que é, por sua vez, configurado para operar numa temporização adequada à taxa de amostragem requerida, fornecendo o sinal de disparo em TRGO em cada instante de amostragem.

Quando um *timer* é selecionado como temporizador mestre, o sinal de saída TRGO pode ser usado pelo periférico escravo (quando configurado). A fonte do sinal TRGO pode ser selecionada na lista a seguir:

• Reset: o bit UG do registrador EGR (*Event Generation Register*) do módulo do *timer* é usado como saída de disparo
• Enable: o sinal de habilitação do timer é utilizado como saída de disparo. É usado para acionar vários temporizadores ao mesmo tempo, ou para controlar um intervalo de tempo no qual um *timer* escravo deve estar habilitado;
• Atualização: o evento de atualização (*Update Event*) é selecionado como saída do trigger (TRGO). Por exemplo, um temporizador mestre pode ser usado como *prescaler* para um temporizador escravo, ou para disparar eventos em outros periféricos como conversões no DAC ou ADC;

• Pulso de comparação: a saída TRGO envia um pulso positivo quando a *flag* CC1IF for setado (mesmo que já esteja setado) assim que ocorrer uma captura ou uma correspondência de comparação;
OC1Ref: o sinal OC1REF é utilizado como saída de disparo;
OC2Ref: sinal OC2REF é utilizado como saída de disparo;
OC3Ref: sinal OC3REF é utilizado como saída de disparo;
OC4Ref: O sinal OC4REF é usado como saída de disparo.

Para configurar um *timer* no modo mestre:

1.Configure o *timer* para a temporização desejada;
2. Selecione a fonte do sinal de disparo a ser utilizada escrevendo nos bits MMS[2:0] do registrador CR2;
3. Selecione o que gera um evento de atualização do timer no bit URS do registrador CR1.

O trecho de código mostrado abaixo mostra um exemplo de aplicação de uso de saídas de disparo de um timer. O código configura o *timer* 3 para gerar pulsos de disparo em TRGO a cada evento de atualização. O *timer* é configurado para realizar uma contagem crescente e estourar a cada 200ms, gerando um evento de atualização. Em seguida, o conversor ADC1 é configurado para receber pulsos de disparo de conversão oriundos do sinal TRGO do *timer* 3. Além disso, é habilitada a interrupção por fim de conversão (*EOC – End Of Conversion*). A rotina de atendimento à interrupção do ADC1 é então usada para fazer a leitura do valor convertido do canal 0. ■

```
//Configuração de GPIO
RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN; //habilita o clock do GPIOA
GPIOA->MODER |= 0b11;                //pino PA0 como entrada analógica

//Configuração do TIMER3
RCC->APB1ENR |= RCC_APB1ENR_TIM3EN;    //liga o clock do Timer3
TIM3->CR1 &= ~TIM_CR1_DIR;             //contador crescente
TIM3->PSC = 419;                       //prescaler para pulsos a cada 5uS
TIM3->ARR = 40000;                     //estouros a cada 200ms
TIM3->CR2 = 0b010 << 4;                //master mode (TRGO a cada estouro do timer)
TIM3->CR1 |= TIM_CR1_URS;              //overflow gera um gatilho em TRGO
TIM3->EGR = 1;                         //update para escrever o valor do prescaler
TIM3->CR1 |= TIM_CR1_CEN;              //habilita o timer

//Configuração do ADC1
RCC->APB2ENR |= RCC_APB2ENR_ADC1EN;    //liga o clock da interface digital do ADC1
ADC->CCR |= 0b01 << 16;                //prescaler /4
ADC1->SQR1 &= ~(0xF << 20);            //conversão de apenas um canal
ADC1->SQR3 &= ~(0x1F);                 //seleção do canal a ser convertido (IN_0)
ADC1->CR2 |= 0b1000 << 24;             //TIM3_TRGO como fonte de gatilho do ADC1
ADC1->CR2 |= ADC_CR2_EXTEN_0;          //disparo externo em rising edge no ADC1
ADC1->CR1 |= 1 << 5;                   //habilita interrupção de EOC
NVIC_EnableIRQ(ADC_IRQn);              //habilita a interrupção do ADC no NVIC
ADC1->CR2 |= ADC_CR2_ADON;             //liga o conversor ADC1
```

```
//ISR do ADC
void ADC_IRQHandler(void)
{
   printf("Leitura = %ld\n", ADC1->DR);//leitura e impressão do valor convertido
}
```