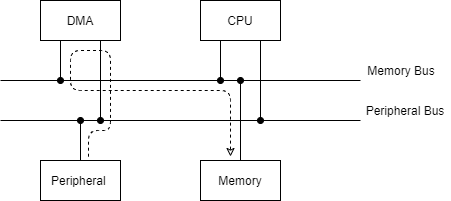
Periféricos Utilizados

1. DMA

Uma unidade de Acesso à Memória Direta (DMA) é um elemento lógico digital que pode ser usado em conjunto com o microprocessador para executar as operações de transferência de memória. Deste modo reduz-se, significativamente, a carga da unidade central de processamento (CPU). Os dispositivos compartilham o barramento de memória e os barramentos de periféricos com o processador (CPU), tal como mostrado na Figura.



Exemplo de transferência do DMA

No diagrama, o dispositivo DMA lê o valor de um periférico a partir do barramento do periférico e grava na memória através do barramento de memória. Na STM o princípio é o mesmo, mas com os vários barramentos de periféricos.

Na STM32F767ZI existem dois controladores DMA (DMA1 e DMA2) que possuem dezasseis *streams* (canais de acesso à memória) no total, oito para cada. Cada *stream* pode teraté dezasseis canais de requisição independentes, configuráveis, como apresentado na figura.

|  |
| --- |
| Uma imagem com mesa  Descrição gerada automaticamente |
| (a) |
| Uma imagem com mesa  Descrição gerada automaticamente |
| (b) |

Mapa de pedido de DMA

A priorização da *stream* pode ser definida pelo utilizador em quatro níveis: baixa, média, alta e muito alta. Além disso, cada *stream* pode ser configurada com um tamanho específico de oito, dezasseis ou trinta e dois bits. Na figura, pode constatar-se que as transferências permitidas são entre memórias, entre o periférico e a memória e entre a memória e o periférico. É de notar que apenas o controlador do DMA 2 está habilitado a executar transferências entre memórias.

Uma imagem com mesa

Descrição gerada automaticamente

Endereços da fonte e do destino

O DMA pode ser configurado em dois modos de transferência de dados: modo direto, em que a transferência de dados é feita de forma imediata, ou modo *first in first out* (FIFO)*,* em que os dados são armazenados temporariamente antes de serem transmitidos para a memória, sendo possível selecionar entre 25%, 50%, 75% ou 100% da capacidade total da FIFO. Em relação aos tamanhos dos dados, estes podem variar entre um byte, *half word* (2 *bytes* consecutivos) ou *word* (4 *bytes* consecutivos). Outras potencialidades do DMA surgem no facto de os apontadores se auto incrementarem, possibilitando a escrita de várias posições de memória consecutivas e a reprogramação ser automática, ou seja, poder usar o modo circular. De forma a inicializar o DMA entre um periférico específico e uma memória, é necessário configurar o canal apropriadamente e definir todas as configurações necessárias de acordo com a figura.

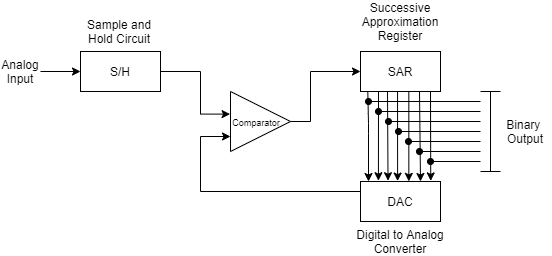
Uma imagem com mesa

Descrição gerada automaticamente

Configurações possíveis do DMA

1. ADC

Um *Analog to Digital Converter* (ADC) converte sinais analógicos em sinais digitais, permitindo aos microcontroladores comunicar com o mundo real. Deste modo, o processador consegue fazer cálculos e tomar decisões com base nos valores amostrados. A STM32F767ZI possui três ADCs de aproximação sucessiva, com dezasseis canais cada, e uma resolução, configurável, de doze, dez, oito ou seis bits. Um conversor por aproximação sucessiva (Figura) possui um comparador e um *Digital to Analog Converter* (DAC) interno para aproximar, sucessivamente, o valor de saída do ADC ao valor de entrada.



O ADC pode ser configurado em *Single-channel* ou *Multichannel* (*scan mode*). No primeiro, apenas é lido um canal, ao passo que, no segundo, são lidos vários canais sucessivamente. Além disso, pode ser configurado em *single conversion mode* e *continuous conversion mode*. No primeiro, o ADC realiza uma única conversão e armazena o resultado no registo de dados (DR). No segundo, o ADC inicializa uma nova conversão logo que a conversão em curso termine. As possíveis configurações destes modos de operação estão apresentadas na Figura.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| (a) | (b) | (c) | (d) |

Modo de funcionamento independente (a) *Single-channel*, *single conversion mode;* (b) Multichannel*, single conversion mode; (c)* *Single-channel, continuous conversion mode;* (d) *Multichannel, continuous conversion mode*

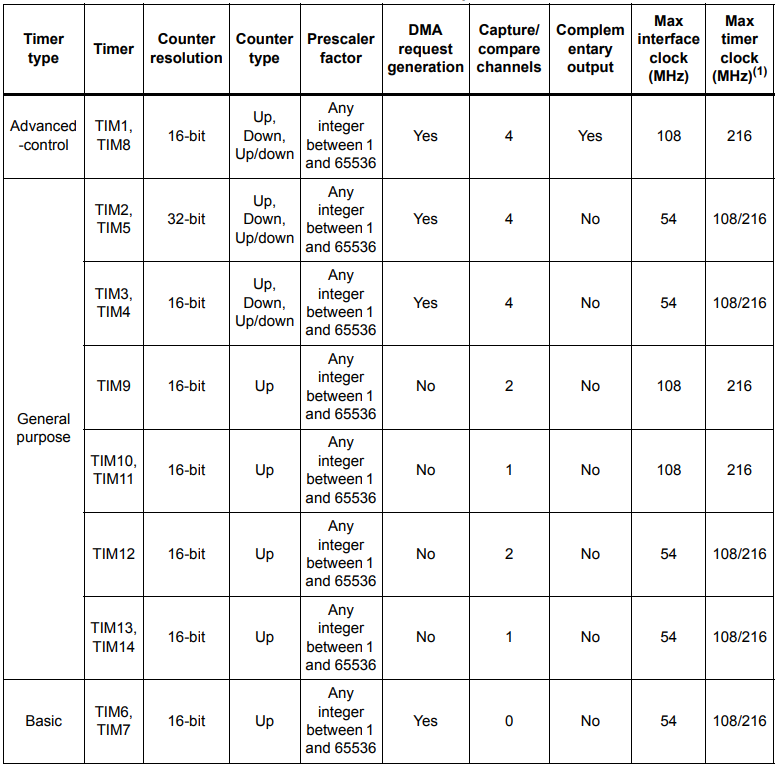
O ADC possui um único registo de dados, por isso, quando se lê múltiplas entradas analógicas, é essencial que os dados sejam lidos entre duas amostras. Isto pode ser realizado através de *polling,* interrupção ou DMA. Interrupções e DMA podem ser disparados no final de cada conversão. Quanto à gama de valores de entrada do ADC, este nunca não suporta valores de tensão negativas nem superiores a 3,3 V.

Outros modos de operação -> Dizer que não se vai especificar

1. TIMER

Um *timer* é um dispositivo de *hardware* capaz de medir uma base de tempo. Pode ser usado, por exemplo, para ativar eventos a frequências ou atrasos conhecidos, gerar sinais a várias frequências, sinais de saída modulados por largura de pulso (PWM) e medir pulsos de entrada.

A STM32F767ZI tem dois *timers* básicos (TIM6 e TIM7), oito *timers* do tipo *general-purpose* (TIM2, TIM3, TIM4, TIM5, TIM9, TIM10, TIM11, TIM12, TIM13 e TIM14) e dois *timers* do tipo *advanced-control* (TIM1 e TIM8). Os *timers* básicos são os mais simples. Consistem em contadores, com contagem crescente, de 16-bits com *auto-reload* controlados por um *prescaler* programável. Podem ser usados como *timers* genéricos para gerar bases de tempo, circuitos de sincronização para despoletar *digital to analog converters* (DAC) e gerar interrupções/DMA através de *update events*. Os *General purpose* estão divididos em vários grupos. Um dos grupos é composto pelos *timers* TIM2, TIM3, TIM4 e TIM5. Consistem em contadores, de contagem crescente, decrescente e crescente/decrescente, de 16‑bits (TIM3 e TIM4) ou 32-bits (TIM2 e TIM5) com *auto-reload* controlados por um *prescaler* programável. Cada um destes *timers* possui quatro canais independentes que podem ser configurados como *output compare, one-pulse mode output, input capture* e PWM *generation*. Podem ser usados para gerar interrupções/DMA através de *update events, trigger events, input capture events* e *output compare events.* Outro grupo é constituído pelos *timers* TIM9 e TIM12. Consistem em contadores, de contagem crescente, de 16‑bits com *auto-reload* controlados por um *prescaler* programável. Cada um destes *timers* possui dois canais independentes que podem ser configurados como *output compare, one-pulse mode output, input capture* e PWM *generation*. Podem ser usados para gerar interrupções através de *update events, trigger events, input capture events* e *output compare events.* O último grupo é constituído pelos *timers* TIM10, TIM11, TIM13 e TIM14. Consistem em contadores, de contagem crescente, de 16‑bits com *auto-reload* controlados por um *prescaler* programável. Cada um destes *timers* possui um único canal independente que pode ser configurado como *output compare, one-pulse mode output, input capture* e PWM *generation*. Podem ser usados para gerar interrupções através de *update events, input capture events* e *output compare events.* Estas informações podem ser observadas de forma mais compacta na figura



Todos os timers estão mapeados em barramentos do mapa de memória. Na tabela pode‑se ver os barramentos a que cada timer pertence, bem a frequência máxima do barramento.

Mapeamento dos *Timers*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Timer | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| APB1(108 MHz) |  | x | x | x | x | x | x |  |  |  |  | x | x | x |
| APB2(216 MHz) | x |  |  |  |  |  |  | x | x | x | x |  |  |  |

Para configurar o *timer* com a frequência desejada é necessário configurar os valores de *prescaler* de *preload*. O *prescaler* divide a frequência de oscilação do barramento, ao passo que o *preload* define o número de contagens do *timer*. Por exemplo, se o barramento tiver uma frequência de 108 MHz e o *prescaler* um valor de 108 – 1 (a contagem do registo começa em zero), a frequência do timer será de 108 MHz / 108 = 1 MHz, o que equivale a uma base de tempo de 1 us. Se o valor de *preload* for 10 – 1, o timer terá uma frequência de 1 MHz / 10= 100 KHz, ou seja, um período de 1 / 100 KHz =10 us. Tendo em conta estes parâmetros e o valor da frequência do barramento do timer em questão (figura), é possível calcular a frequência do *timer* usando a Equação.

|  |  |  |
| --- | --- | --- |
|  |  | (2.4) |

1. UART

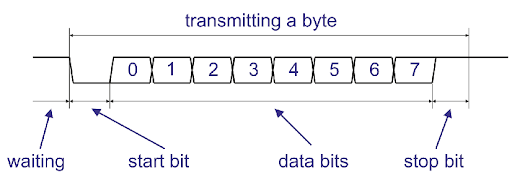
A *universal asynchronous receiver transmitter* (UART) é uma comunicação série largamente usada para comunicar entre dois dispositivos, como sistemas embebidos, microcontroladores e computadores. Na comunicação UART, a comunicação direta entre dois dispositivos faz-se através de duas linhas de dados como representado na figura.

Uma imagem com texto, relógio

Descrição gerada automaticamente

Esquema de ligação entre 2 UARTs

Este periférico transmite dados de forma assíncrona, ou seja, não necessita de um sinal de *clock* para se manter sincronizado. Em vez disso, o emissor acrescenta bits ao pacote a ser transmitidosinalizando o início e o fim dos dados a ser transferidos, tal como mostrado na figura.



Estrutura de um pacote de dados enviado por UART

Quanto ao recetor, após detetar o *start* bit, começará a ler os bits que se seguem a uma determinada frequência, denominado *baud rate*. Expresso em bits por segundo (bps), este é definido como sendo uma medida de velocidade de transmissão. Ambas as UART têm de operar ao mesmo *baud rate* para não ocorrer erros na transmissão, antes da inicialização da comunicação, ambas as UARTs têm de ser configuradas com o mesmo *baud rate* e com a mesma estrutura de pacotes.

Na STM32F767ZI encontram-se disponíveis 8 módulos UART do quais 4 (USART1, USART2, USART3 e USART6) podem ser configurados em modo síncrono. A figura mostra as funcionalidades que cada UART, presente na placa, possui.

Uma imagem com mesa

Descrição gerada automaticamente

Figura de funcionalidades existentes para cada UART

Cada UART pode ser configurada para comunicação assíncrona ou síncrona (no caso das UARTs previamente referidas), pode ser ainda configurada em *full-duplex* ou *half-duplex*, com baud-rate até 27 Mbits/s (quando *clock source* da UART é o *clock* do sistema) ou com deteção automática do baud rate. A estrutura dos pacotes de dados é modificável, podendo escolher ter 7,8 ou 9 bits de dados, 1 ou 2 stop bits e uma definição da ordem da transmissão de dados com MSB-*first* (bit mais significativo primeiro) ou LSB-*first* (bit menos significativo primeiro). Para além disso, existem mais 14 *interrupt flags* para controlo de fluxo de dados e erros. Possui, ainda, a possibilidade de funcionar com o DMA para comunicação continua e para *buffers* de dados recebidos e transmitidos, com o intuito de serem guardados ou adquiridos da SRAM. As UARTs podem também configuradas para a utilização do padrão de comunicação série RS-232 ou o padrão RS-485. Os principais elementos das UARTs localizadas na STM32F767ZI, ~~e~~ bem como as suas interações encontram-se descritas na figura.

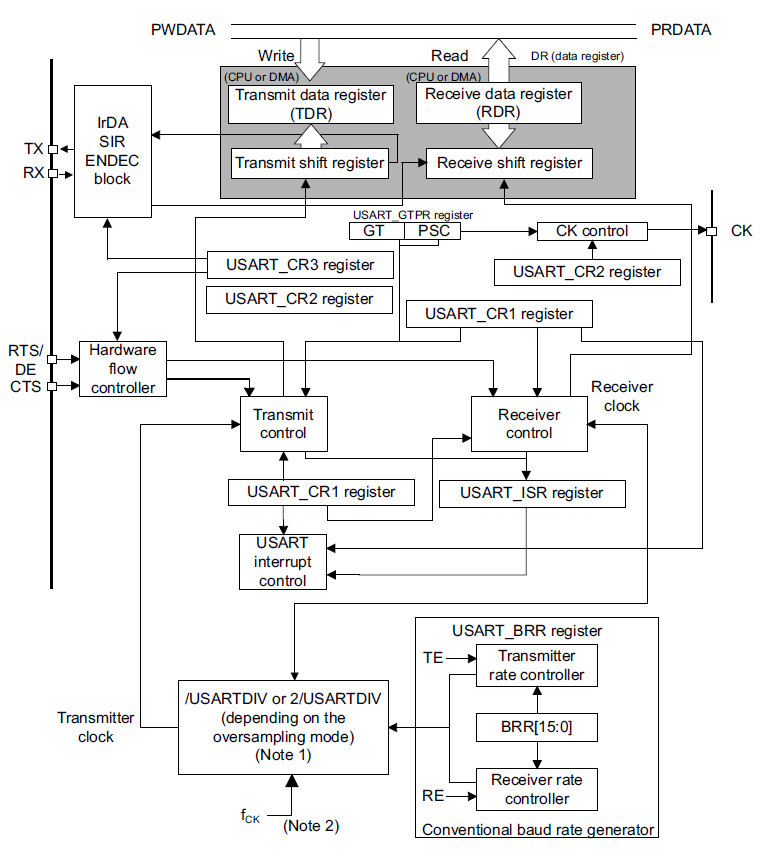


Diagrama de blocos da USARTs

1. SPI

A *Serial Peripheral Interface* (SPI) é um protocolo de comunicação de curto alcance criado para a troca de dados entre microcontroladores ou entre microcontroladores e sensores. O SPI é síncrono e necessita que o transmissor e o recetor estejam sincronizados através de uma linha de *clock* partilhada. Este suporta transferências de dados em três modos: *full-duplex* (2 linhas de dados), *half-duplex* (1 linha de dados bidirecional) ou *simplex* (1 linhas unidirecional).

Os dispositivos conectados por SPI encontram-se numa relação *master-slave.* O *master* é, tipicamente, um microcontrolador que envia instruções ao *slave*. Este, em vários casos, trata-se de um sensor, um *chip* de memória ou um display.

De modo a proceder à comunicação *full-duplex* entre um dispositivo *master* e um dispositivo *slave,* estes devem ser conectados segundo a configuração apresentada na figura.

Uma imagem com texto

Descrição gerada automaticamente

Esquema de ligação entre Master e Slave.

O protocolo SPI especifica 4 linhas lógicas para a transferência de dados:

* SCLK: *Serial Clock*, uma saída proveniente do *master* para sincronizar as transferências de dados nas linhas MISO e MOSI;
* MOSI: *Master Out Slave In*, Saída de dados do *master* com destino ao *slave*;
* MISO: *Master In Slave Out*, Saída de dados do *slave* com destino ao *master*;
* SS: *Slave Select*, através desta linha o *master* pode selecionar com qual dispositivo pretende comunicar.

A STM32F767ZI possui 6 interfaces SPI que suportam comunicação *full-duplex*, *half-duplex* e *simplex*, permitindo enviar blocos de dados de 4 a 16 bits. Além disso, o *clock* e a ordem dos dadosé configurável por software, existem *interrupt flags* dedicadas à transmissão e receção de dados e cada interface possui 2 FIFOs de 32 bits (uma para receção e outra para a transmissão) com capacidade de uso conjunto com o DMA. Os principais elementos da interface SPI localizada na STM32F767ZI e as suas interações encontram-se descritas na figura.

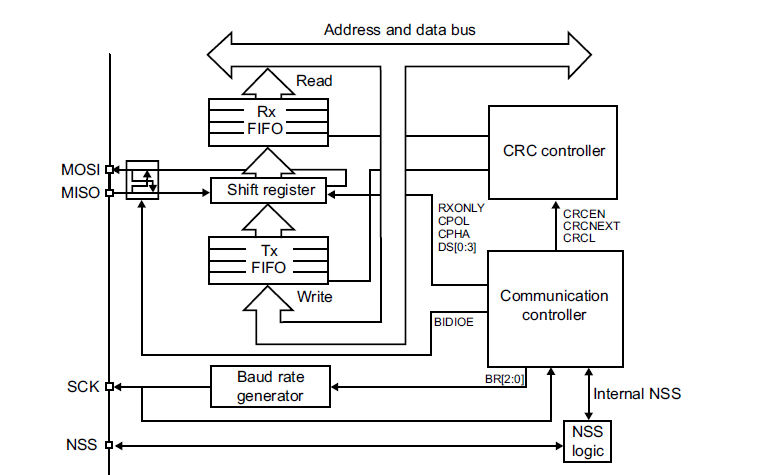


Diagrama de blocos das interfaces SPI

Tabela Periféricos

|  |  |  |
| --- | --- | --- |
| Configuração ADC2 (com base no stm32cube) | | |
| *Mode* | | *Independent Mode* |
| *Clock Prescaler* | | *PCLK2 divide by 4* |
| *Resolution* | | 12-bits |
| *Data Alignment* | | *Right alignment* |
| *Scan Conversion Mode* | | *Enable* |
| *Conversion Mode* | | *Continuous* |
| *DMA Continuous Requests* | | *Enable* |
| *End Of Conversion Selection* | | *EOC flag at the end of single channel conversion* |
| *Number Of Conversion* | | 1 |
| *External Trigger Conversion Source* | | *Regular conversion* |
| Rank 1 | *Channel* | 1 |
| *Sample Time* | 3 *Cycles* |
| *Global Interrupt* | | *Enable* |

|  |  |  |
| --- | --- | --- |
| Configuração ADC2 (com base no stm32cube) | | |
| *Mode* | | *Independent Mode* |
| *Clock Prescaler* | | *PCLK2 divide by 4* |
| *Resolution* | | 12-bits |
| *Data Alignment* | | *Right alignment* |
| *Scan Conversion Mode* | | *Enable* |
| *Conversion Mode* | | *Continuous* |
| *DMA Continuous Requests* | | *Enable* |
| *End Of Conversion Selection* | | *EOC flag at the end of single channel conversion* |
| *Number Of Conversion* | | 6 |
| *External Trigger Conversion Source* | | *Regular conversion* |
| *Rank 1* | *Channel* | 10 |
| *Sample Time* | 3 *Cycles* |
| *Rank 2* | *Channel* | 13 |
| *Sample Time* | 3 *Cycles* |
| *Rank 3* | *Channel* | 9 |
| *Sample Time* | 3 *Cycles* |
| *Rank 4* | *Channel* | 15 |
| *Sample Time* | 3 *Cycles* |
| *Rank 5* | *Channel* | 8 |
| *Sample Time* | 3 *Cycles* |
| *Rank 6* | *Channel* | 6 |
| *Sample Time* | 3 *Cycles* |
| *Global Interrupt* | | *Enable* |

|  |  |  |
| --- | --- | --- |
| Configuração DMA2 (com base no stm32cube) | | |
| ADC3 | *Stream* | 1 |
| *Direction* | *Peripheral to memory* |
| *Priority* | *Low* |
| *Mode* | *Circular* |
| *Increment Address* | *Memory* |
| *Data Witdth* | *Word* |
| ADC2 | *Stream* | 2 |
| *Direction* | *Peripheral to memory* |
| *Priority* | *Low* |
| *Mode* | *Circular* |
| *Increment Address* | *Memory* |
| *Data Witdth* | *Word* |

Em relação aos *timers*, foram usados quatro para aplicações distintas. Sempre que o robô deteta uma linha terminal, significa que, pouco tempo depois, terá de ler o RFID do quarto ou cruzamento correspondente. Sendo assim, é necessário um temporizador que permite encaminhar o robô para o estado de erro caso se ultrapasse o tempo estabelecido e o cartão RFID não seja lido. O tempo estabelecido foi de 1 segundo. A tabela apresenta as configurações usadas para que este periférico funcionasse como desejado.

|  |  |
| --- | --- |
| Configuração TIM3 (com base no stm32cube) | |
| **Counter Settings** | |
| Prescaler | 60000 - 1 |
| Counter Mode | Up |
| Counter Period | 1800 - 1 |
| Internal clock divison | No division |
| Auto-reload preload | Enable |
| **Trigger Output** | |
| Master/Slave Mode | Disable |
| Trigger Event Selection | Update Event |
| **NVIC** | |
| Global Interrupt | Enable |

O controlo dos motores afeta, diretamente, a tensão de alimentação do motor, ou seja, com um sinal de PWM. Sendo assim, foi necessário utilizar um *timer*, TIM4 no caso, que possibilitasse a geração destes. Para o efeito, selecionou-se dois canais (3 e 4) e configurou-se em *PWM Generation.*

|  |  |  |
| --- | --- | --- |
| Configuração TIM4 (com base no stm32cube) | | |
| **Counter Settings** | | |
| *Prescaler* | | 108 - 1 |
| *Counter Mode* | | *Up* |
| *Counter Period* | | 100 - 1 |
| *Internal clock divison* | | *No division* |
| *Auto-reload preload* | | *Enable* |
| ***Trigger Output*** | | |
| *Master/Slave Mode* | | *Disable* |
| *Trigger Event Selection* | | *Reset* |
| ***Channel 3*** | *Mode* | 1 |
| *Pulse* | 0 |
| *Output compare preload* | *Enable* |
| *Fast mode* | *Diasble* |
| *CH Polarity* | *High* |
| ***Channel* 4** | *Mode* | 1 |
| *Pulse* | 0 |
| *Output compare preload* | *Enable* |
| *Fast mode* | *Diasble* |
| *CH Polarity* | *High* |
| ***NVIC*** | | |
| *Global Interrupt* | | *Enable* |

Para uma implementação, de um controlador PID, em microcontrolador é necessária uma versão digital do mesmo. Com isto, recorre-se a um *timer,* neste caso o mais básico – TIM6, para definir a frequência de amostragem dos valores lidos pelos sensores. Neste caso, optou-se por um período de amostragem de 10 ms.

|  |  |
| --- | --- |
| Configuração TIM6 (com base no stm32cube) | |
| **Counter Settings** | |
| Prescaler | 10800 - 1 |
| Counter Mode | Up |
| Counter Period | 100 - 1 |
| Auto-reload preload | Enable |
| **Trigger Output** | |
| Trigger Event Selection | Update Event |
| **NVIC** | |
| Global Interrupt | Enable |

O robô contém um botão de pressão que permite, ao utilizador, iniciar a marcha. Assim, para evitar possíveis leituras erradas efetuou-se o *debounce* do botão. Para este ser possível de implementar tem-se que recorrer a um timer. Neste caso usou-se o TIM13 com uma base de tempo de 50 ms.

|  |  |
| --- | --- |
| Configuração TIM13 (com base no stm32cube) | |
| **Counter Settings** | |
| Prescaler | 10800 - 1 |
| Counter Mode | Up |
| Counter Period | 500 - 1 |
| Internal clock divison | No division |
| Auto-reload preload | Enable |
| **NVIC** | |
| Global Interrupt | Enable |

---------------------------------------------------------------------------------------\*\*--------------------------------------------------------------------------------------

TIM 6 -> SAMPLE TIME  
TIM 3 -> TIMEOUT

TIM 13 -> DEBOUNCE

TIM 4 -> PWM