# Aplication Note - Configuração da árvore de clock em tempo de execução em um STM32F103rb Nucleo

Universidade Federal de Minas Gerais

Pietro Zanetti
Pedro Bahia

Agosto 2021

# Índice

# 1 Introdução

O Núcleo STM32F103RB apresenta uma série de clocks destinados a diversas funcionalidades organizados na árvore de clock. Sua configuração adequada é essencial para a previsibilidade e corretude do programa. Este Application Note visa orientar na configuração inicial da árvore de clock de um Núcleo STM32F103RB por meio da interface gráfica STM32CubeMX e na posterior alteração dos valores dos clock em meio à execução. O código completo utilizados no exemplo desenvolvido estará na página do GitHub fornecida ao final deste trabalho e os itens necessários para a reprodução serão os seguintes:

- Hardware: Computador e Núcleo STM32F103RB ou similiar
- Software: STM32CubeMX
- IDE: SW4STM32 - System Workbench for STM32

# 2 Árvore de Clock

Existem ao todo cinco origens distintas para o clock utilizado, que podem ser ligadas ou desligadas de forma independente. Quatro delas, (HSI, HSE, LSE e LSI) diferem entre si quanto a velocidade, high-speed e low-speed, e a origem, intern e extern, sendo o intern um oscilador RC e o extern um Crystal/Ceramic Resonator ou uma fonte externa. Enquanto isso, o quinto (PLL) é um gerador de clock que utiliza o sinal do HSE ou HSI para gerar um clock de maior frequência e garantir a sincronia entre sinais de entrada e saída. Por padrão, o HSI é utilizado após a reinicialização como o clock do sistema (SYSCLK).
Assim que a fonte do sinal e sua respectiva frequência são escolhidas, os pré-escalares devem ser definidos. Eles serão utilizados como divisores do sinal inicial nas ramificações dos clocks, ramificações essas cuja interface destino é previamente determinada. Há ainda o multiplicador do PLL que pode ser definido.[1]
A figura 1 ilustra a árvore de clock do STM32F103RB com suas devidas fontes, pré-escalares e interfaces de destino. Nota-se nela as restrições quanto aos valores possíveis de cada elemento.

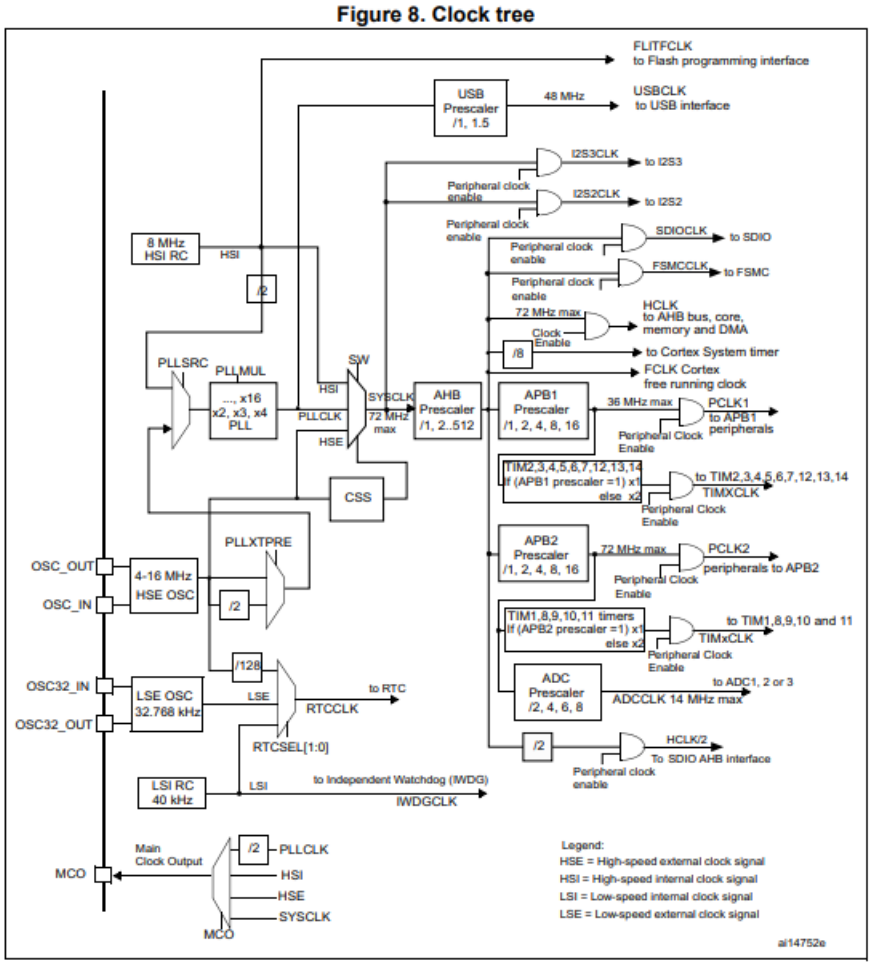


Figura 1: Árvore de Clock.[1]

# 3 Configuração da Árvore de Clock

## 3.1 Configuração inicial usando STM32CubeMX

A figura 2 é uma captura de tela da árvore de clock do STM32CubeMX, que pode ser acessada pela aba "Clock Configuration" do programa. Por meio dela defini-se todos os atributos iniciais do clock, como fontes, valores de frequência e pré-escalares.Como saída, é possível obter diversos valores de frequências para diferentes fins a partir de um valor inicial de SYSCLK e dos pré-escalares AHB Prescaler, APB1 Prescaler e APB2 Prescaler. As marcações coloridas indicam cada parâmetro utilizado para a configuração do SYSCLK. Em vermelho estão as fontes de clock a ser selecionadas e suas respectivas frequências

a serem definidas. Nota-se a inscrição abaixo das fontes com os valores limites de frquência. Marcado em rosa está o MUX em que a fonte do SYSCLK é selecionada. Já em verde estão os pré-escalares que podem ser alterados entre valores pré-definidos. Por fim, em preto, estão os diversos sinais de saída com o periférico ou interface ao qual se destinam. Além das marcações, pré-escalares e multiplicadores individuais encontram-se presentes a depender da ramificação a ser configurada. Os timers, por exemplo, podem ser multiplicados por um valor assim que são divididos pelo pré-escalar APBX.

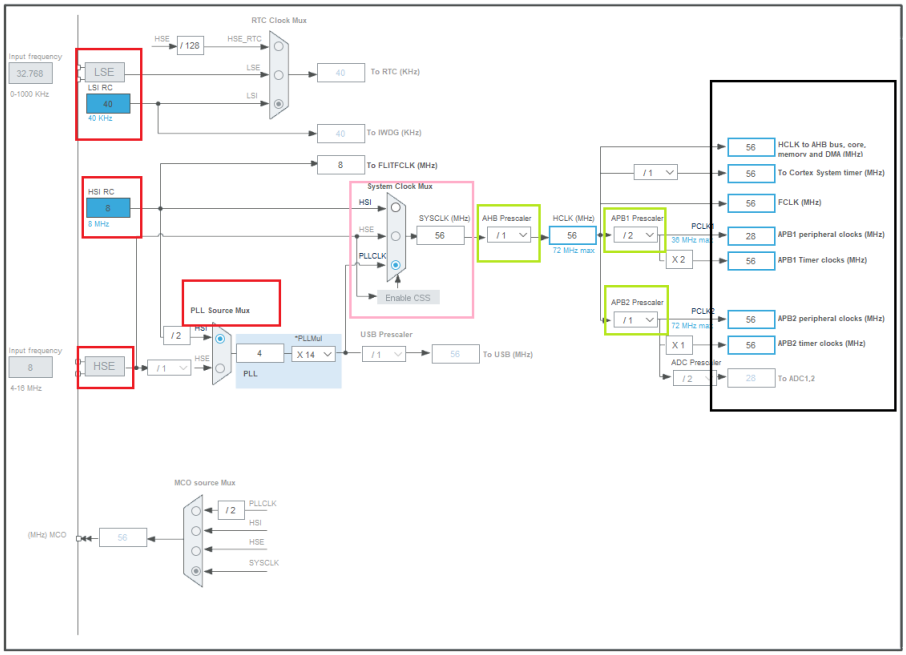


Figura 2: Configuração da árvore de clock

Assim que os parâmetros são devidamente selecionados e o código é gerado, a função **void SystemClockConfig(void)** contendo os valores definidos é criada no SW4STM32. Ao chamá-la no início da execução do programa, os clocks serão devidamente inicializados.

## 3.2 Alteração dos valores de clock em tempo de execução

Para que os valores da frequência de clock sejam alterados durante a execução do programa é, então, necessário que esses sejam acessados à partir de um registrador específico, o RCC_CFGR (Figura 3), e escritos os valores desejados nos bits 7:4 (AHB Prescaler, figura 3), 10:8 (APB1 Prescaler, figura 4) e 13:11 (APB2 Prescaler, figura 5).

### 7.3.2 Clock configuration register (RCC_CFGR)

Address offset: 0x04

Reset value: 0x0000 0000

Access: 0 ≤ wait state ≤ 2, word, half-word and byte access

1 or 2 wait states inserted only if the access occurs during clock source switch.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reserved | | | | | MCO[2:0] | | | Res. | USB PRE | PLLMUL[3:0] | | | | PLL XTPRE | PLL SRC |
| | | | | | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADCPRE[1:0] | | PPRE2[2:0] | | | PPRE1[2:0] | | | HPRE[3:0] | | | | SWS[1:0] | | SW[1:0] | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | r | r | rw | rw |

Figura 3: Clock configuration register (RCC_CFGR).[1]

Bits 7:4 **HPRE:** AHB prescaler

Set and cleared by software to control the division factor of the AHB clock.
0xxx: SYSCLK not divided
1000: SYSCLK divided by 2
1001: SYSCLK divided by 4
1010: SYSCLK divided by 8
1011: SYSCLK divided by 16
1100: SYSCLK divided by 64
1101: SYSCLK divided by 128
1110: SYSCLK divided by 256
1111: SYSCLK divided by 512

*Note: The prefetch buffer must be kept on when using a prescaler different from 1 on the AHB clock. Refer to Reading the Flash memory section for more details.*

Figura 4: AHB Prescaler.[1]

Bits 10:8 **PPRE1:** APB low-speed prescaler (APB1)

Set and cleared by software to control the division factor of the APB low-speed clock (PCLK1).
Warning: the software has to set correctly these bits to not exceed 36 MHz on this domain.
0xx: HCLK not divided
100: HCLK divided by 2
101: HCLK divided by 4
110: HCLK divided by 8
111: HCLK divided by 16

Figura 5: APB1 Prescaler.[1]

Bits 13:11 **PPRE2:** APB high-speed prescaler (APB2)

Set and cleared by software to control the division factor of the APB high-speed clock (PCLK2).
0xx: HCLK not divided
100: HCLK divided by 2
101: HCLK divided by 4
110: HCLK divided by 8
111: HCLK divided by 16

Figura 6: APB2 Prescaler.[1]

No projeto, em linguagem C, o registrador pode ser acessado da seguinte maneira:

```
RCC -> CFGR | = RCC_CFGR_HPRE_DIVX; // AHB prescaler
RCC ->CFGR | = RCC_CFGR_PPRE1_DIVX; // APB1 presacaler
RCC -> CFGR | = RCC_CFGR_PPRE2_DIVX; // prescaler APB2
```

A terminação 'X' indica os possíveis valores para a divisão, os quais podem ser encontrado dentro do arquivo stm32f103xb.h, no caso do microcontrolador utilizado nessa AP, ou equivalente. Esse valor também pode ser escrito diretamente à partir de seu correspondente em bit, mostrado nas figuras 4, 5 e 6.
Por fim, a função **SystemCoreClockUpdate()** deve ser chamada para que o clock seja devidamente atualizado.

# 4 Exemplo Prático

Para exemplificar os conceitos abordados, foi elaborada uma montagem, em que o LED interno da placa do STM32 pisca em diversas frequências ao longo da execução. O timer TIM1 foi escolhido para enviar um sinal de interrupção assim que um número de clocks forem contados. Assim que tal sinal é enviado, a função **HAL_GPIO_TogglePin** altera o estado do Led fazendo-o piscar.

Inicialmente configura-se a árvore de clock a partir da interface gráfica STM32CubeMX como na imagem seguinte. A origem do clock do sistema foi definida como o PLL com frequência de 56MHz e os pré-escalares foram escolhidos para que o sinal de saída tivesse a maior frequência possível para esse valor, 56 MHz, como especificado na figura 7.

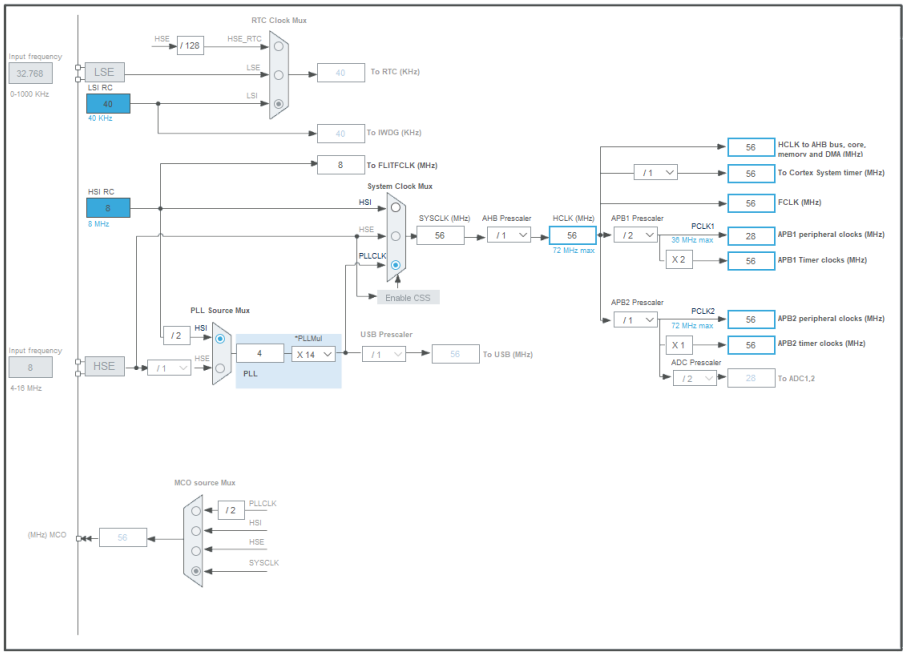


Figura 7: Configuração da árvore de clock.

A configuração do timer que será utilizado para tratar a representação visual do clock, em velocidade em que o LED pisca, foi feita de acordo com a figura 8.

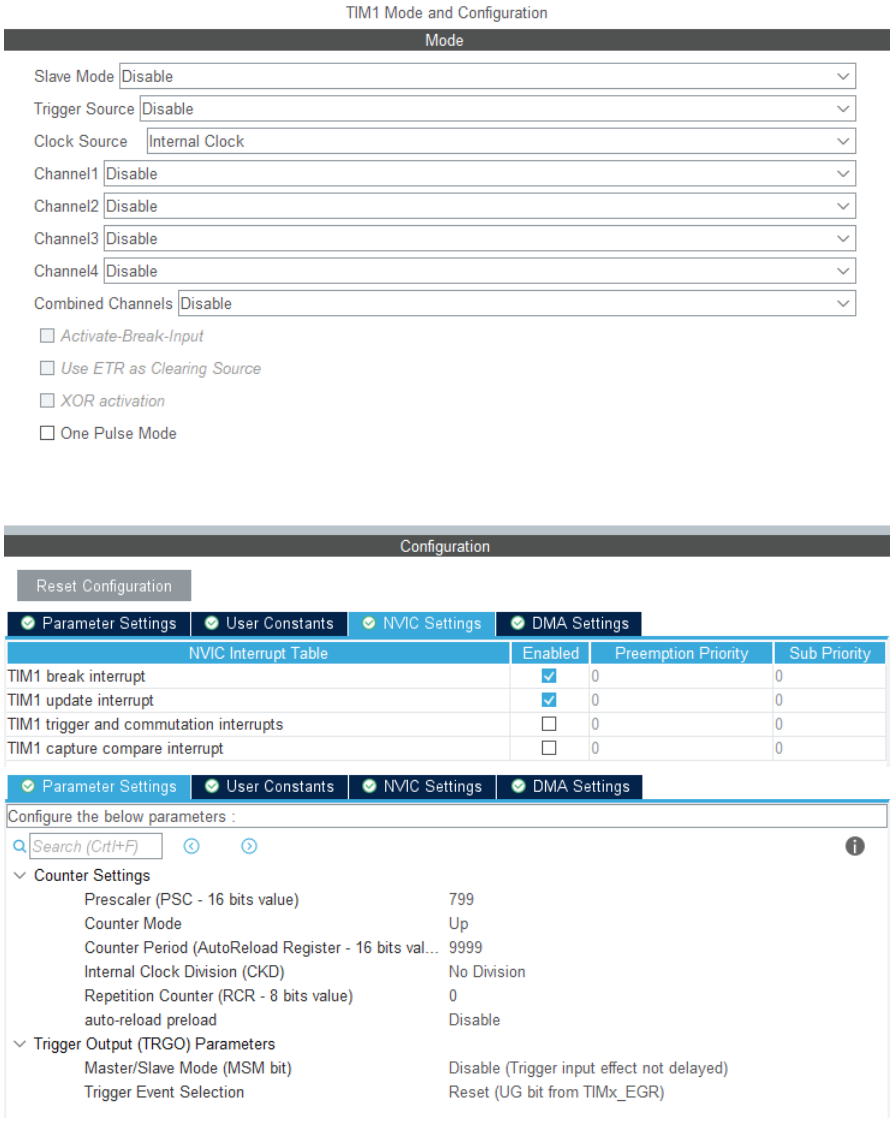


Figura 8: TIM1 Mode and Configuration

Após as configurações iniciais a partir do STM32CubeMX, um projeto para SW4STM32 foi gerado. No arquivo *main.c* foi criada a função *makeItGoSlow*(*uint32_t div*) em que o valor do prescaler AHB é alterado em função do parâmetro e, então, o clock do sistema é atualizado:

```
1 void makeItGoSlow(uint32_t div)
2 {
```

```
 RCC->CFGR |= div;  //AHB prescaler

 SystemCoreClockUpdate();

}
```

Para o uso do timer, dentro do arquivo $stm32f1xx\_it.c$ foi escrito em seu handler para que piscasse o LED toda vez que ocorresse uma interrupção.

```
void TIM1_UP_IRQHandler(void)
{
  /* USER CODE BEGIN TIM1_UP_IRQn 0 */
  HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_5);
  /* USER CODE END TIM1_UP_IRQn 0 */
  HAL_TIM_IRQHandler(&htim1);
  /* USER CODE BEGIN TIM1_UP_IRQn 1 */

  /* USER CODE END TIM1_UP_IRQn 1 */
}
```

Por fim, dentro do loop infinito do programa principal ($main.c$) chama-se a função criada com valores de AHB prescaler cada vez maiores, ou seja, uma frequência cada vez menor e em seguida utiliza-se um delay para que o led possa piscar algumas vezes.

```
  while (1)
  {
    /* USER CODE END WHILE */
    makeItGoFast(RCC_CFGR_HPRE_DIV1);
    HAL_Delay(1000);
    makeItGoFast(RCC_CFGR_HPRE_DIV2);
    HAL_Delay(1000);
    makeItGoFast(RCC_CFGR_HPRE_DIV4);
    HAL_Delay(1000);
    makeItGoFast(RCC_CFGR_HPRE_DIV8);
    HAL_Delay(1000);
    SystemClock_Config();
    /* USER CODE BEGIN 3 */
  }
```

# 5 Conclusão

Nesse Application Note, foram apresentadas duas maneiras de configurar o clock de um programa. A primeira por meio da interface STM32CubeMX e a segunda utilizando o SW4STM32, que permitia a alteração dos valores durante a execução. Discorreu-se igualmente a cerca do funcionamento da árvore de clock para melhor compreensão. Como meio ilustrativo, um exemplo simples da aplicação foi desenvolvido em que um LED piscava em diferentes frequências à medida que o clock era alterado. Seus resultados, assim como os códigos utilizados, encontram-se no projeto do GitHub encontrados no link https://github.com/pietrolzanetti/AN_Clock_Tree.
Com isso, o objetivo inicial do documento foi cumprido.

# References

[1] *RM0008 - Reference Manual STM32F103xx.*