

- Núcleo do ARM Cortex – hardware...

(Registradores, ULA, GPIOs, pipeline, barramentos e interconexão com vários periféricos)

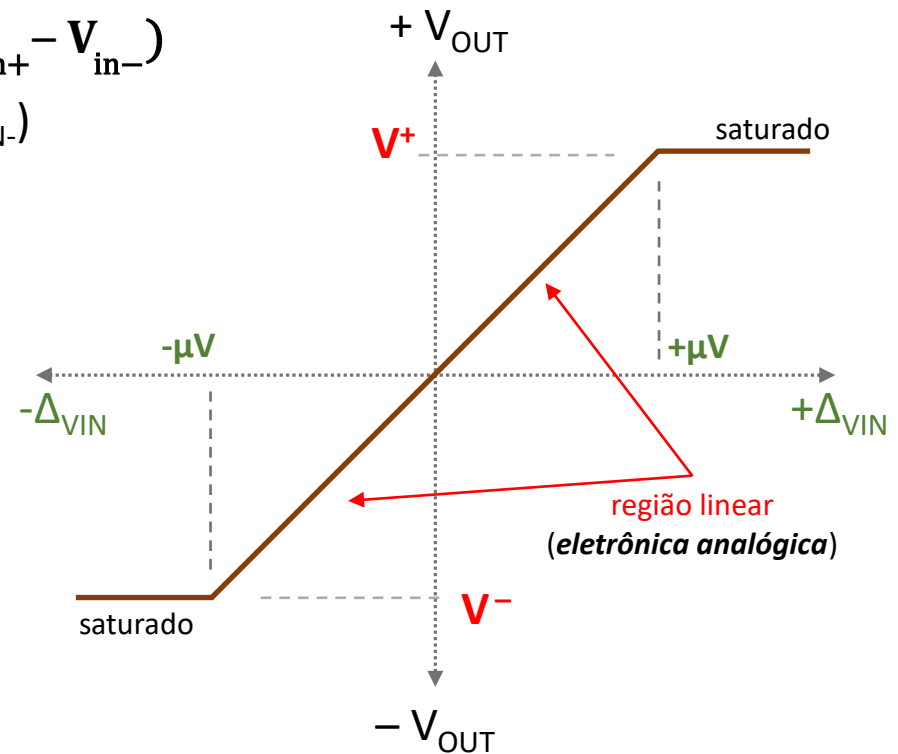
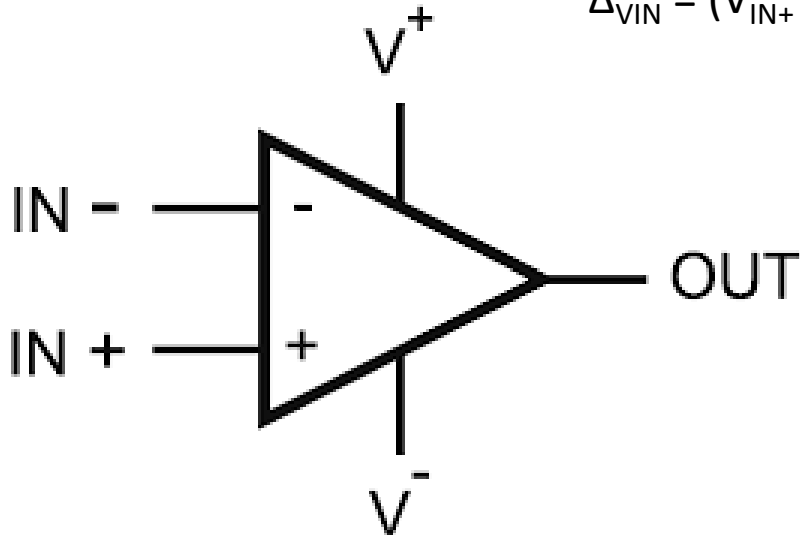
- Conceitos de programação – e.g. superloop, funções, assembly...
- Conceito de Interrupção – programação do NVIC
- Programação de periféricos (GPIO, timers, SysTick, PWM...)
- *Comunicação paralela - falamos de barramentos (interfaceamento paralelo)*
- *Comunicação serial*
  
- HOJE: conversão DAC e ADC
- DMA
- JTAG

Amplificador Operacional (AO) é um componente criado com transistores e resistores mas que permite ao engenheiro eletrônico abstrair de seus componentes internos e considera-lo um bloco operacional – um **componente** especial!

- 4) OpAmps têm ALTO GANHO (*o ideal seria ganho infinito*), o real é da ordem de  $10^5$  a  $10^6$   
Chamando  $G$  o ganho do amplificador, a saída OUT pode ser calculada como:

$$V_{OUT} = -G \cdot (V_{in+} - V_{in-})$$

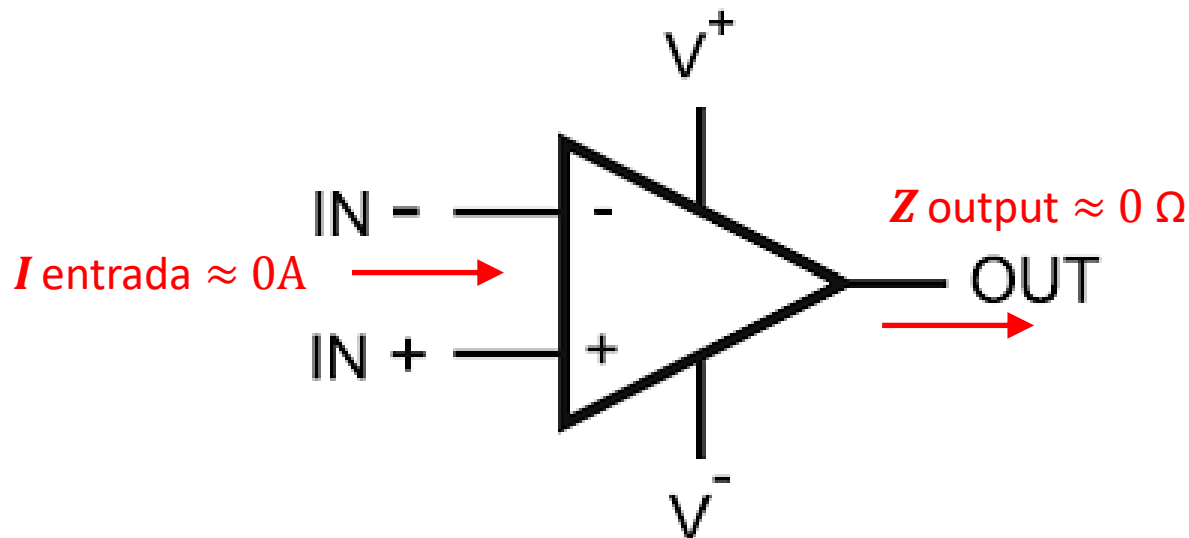
$$\Delta_{VIN} = (V_{IN+} - V_{IN-})$$



- RECORDANDO: OpAmp permite ao engenheiro eletrônico abstrair de seus componentes internos e considerá-lo um bloco operacional – um **componente** especial ideal!

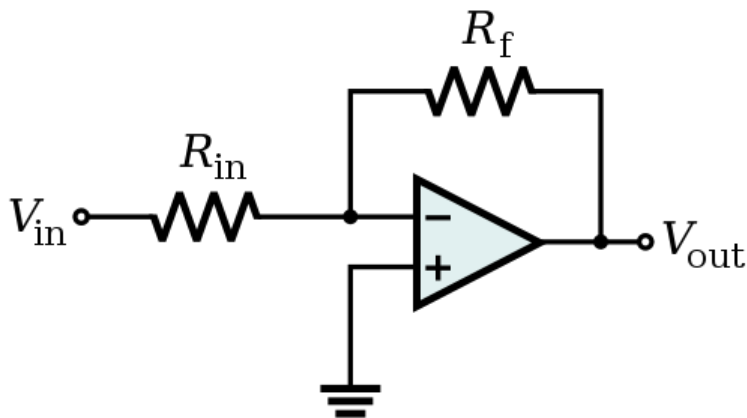
5) OpAmps têm **entradas** com **ALTA IMPEDÂNCIA** (ou seja, alta resistência de entrada): (idealmente, a resistência seria infinita, mas na prática são alguns MΩ). Isso significa que, na prática, a **corrente nas entradas do OpAmp são desprezíveis...**

6) OpAmps tem **BAIXA IMPEDÂNCIA** na **saída** (ou seja, baixa resistência de saída - ideal seria zero, mas na prática tem alguns Ω). Isso significa que um OpAmp deveria fornecer altas correntes de saída (na prática, fornece de alguma mA até alguns A).



Amplificador Operacional (AO) é um componente criado com transistores e resistores mas que permite ao engenheiro eletrônico abstrair de seus componentes internos e considera-lo um bloco operacional – um **componente** especial!

➤ Amplificador inversor



$$\begin{aligned}
 I &= \frac{V_{in}}{R_{in}} \\
 I &= \frac{0 - V_{out}}{R_f}
 \end{aligned}
 \left. \vphantom{\begin{aligned} I &= \frac{V_{in}}{R_{in}} \\ I &= \frac{0 - V_{out}}{R_f} \end{aligned}} \right\} \frac{-V_{out}}{R_f} = \frac{V_{in}}{R_{in}}$$

$$-V_{out} = V_{in} \frac{R_f}{R_{in}}$$

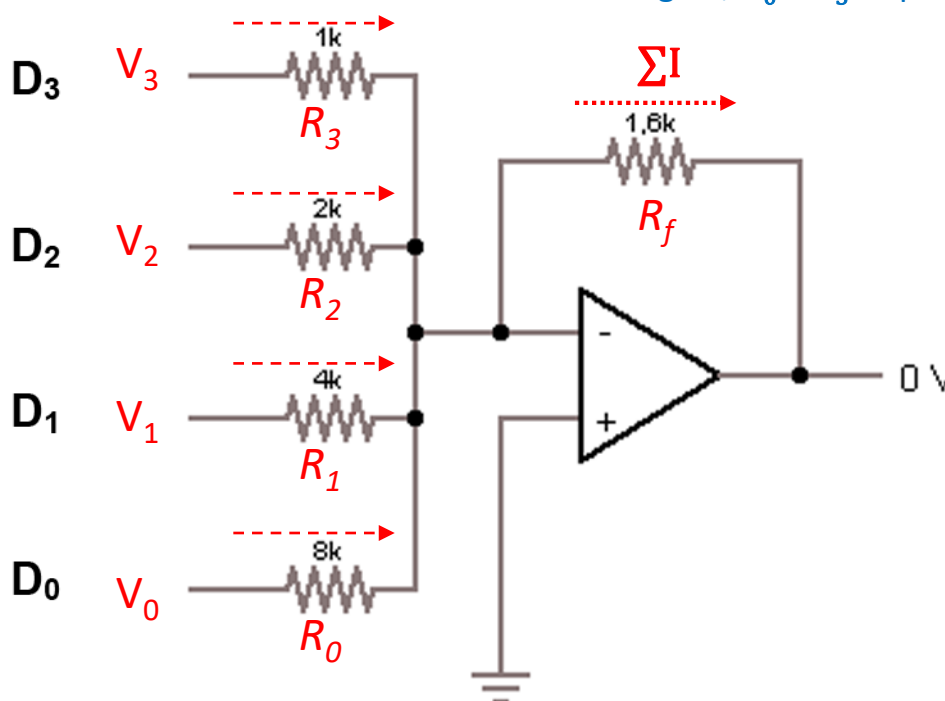
a) Amplificador inversor

$$V_{out} = -V_{in} \cdot \left(\frac{R_f}{R_{in}}\right)$$

## ➤ Somador inversor:

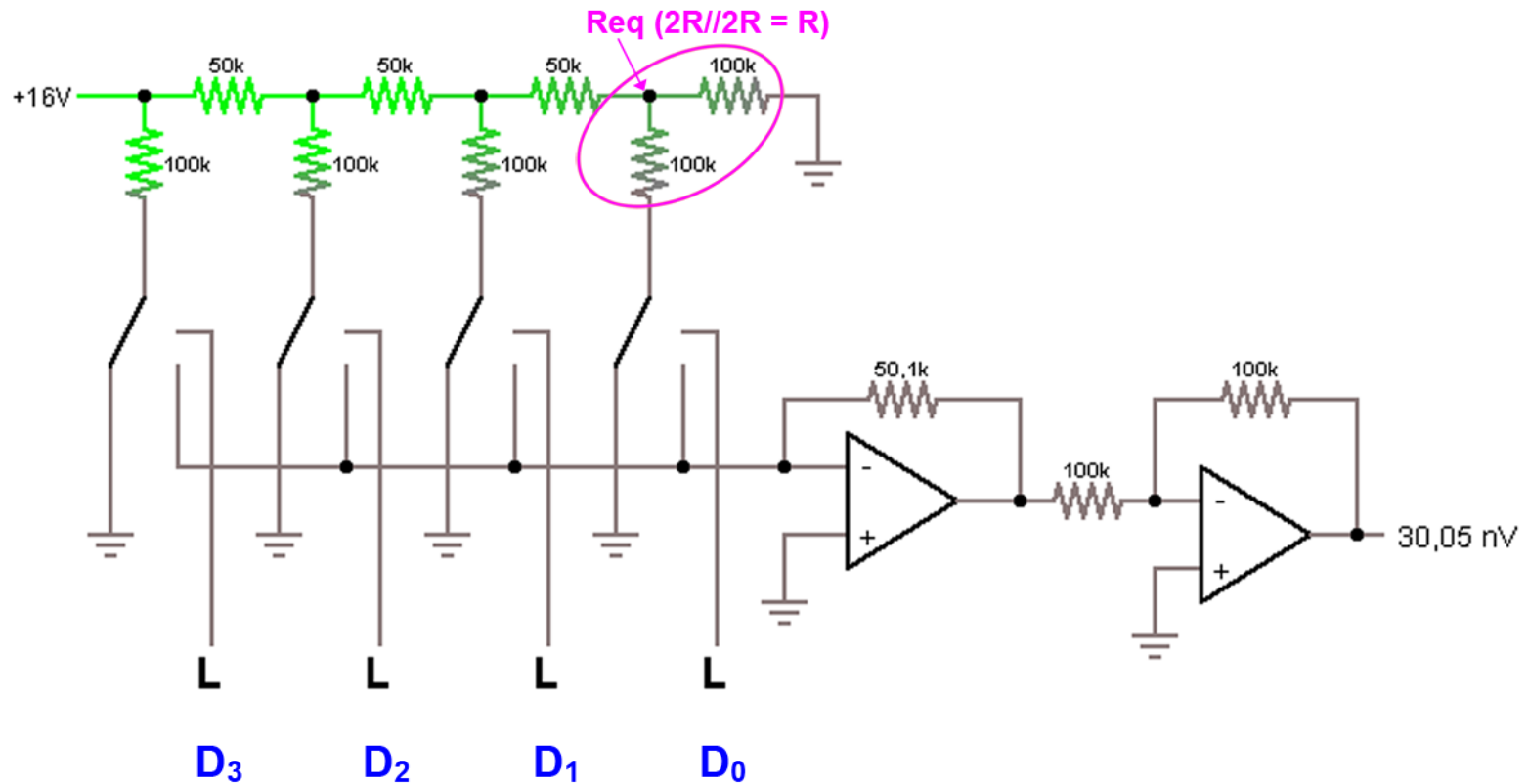
$$V_{out} = -\left[ V_3 \left( \frac{R_f}{R_3} \right) + V_2 \left( \frac{R_f}{R_2} \right) + V_1 \left( \frac{R_f}{R_1} \right) + V_0 \left( \frac{R_f}{R_0} \right) \right]$$

OBS: em eletrônica digital,  $V_0 \dots V_3$  só podem ser "1" ou "0"

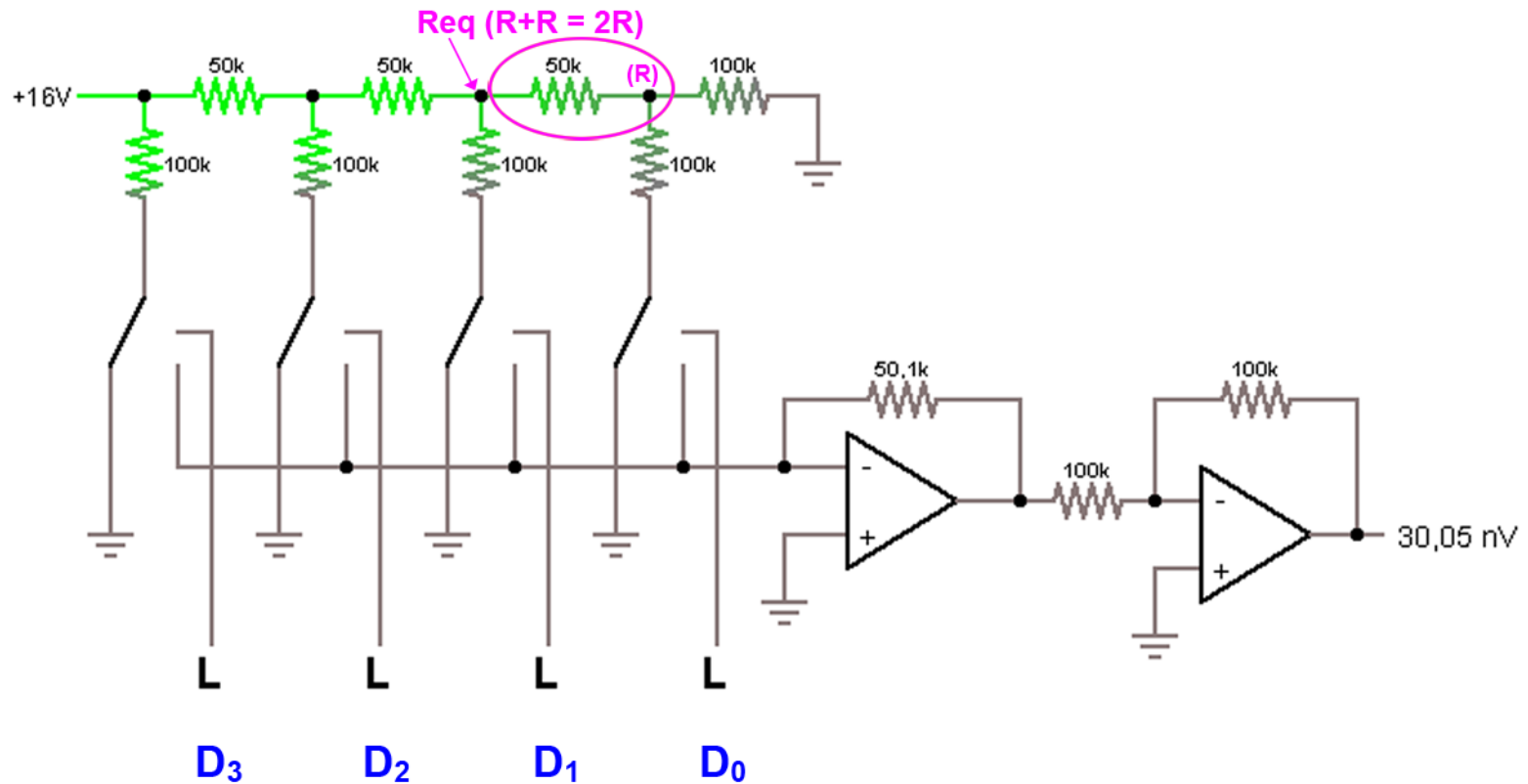


Um circuito somador inversor SOMA as correntes que entram em cada uma das entradas do somador, porque todas as correntes só tem um caminho para passar: o resistor  $R_f$ . Assim, podemos considerar que o somador SOMA os valores de tensão  $V_0 \dots V_N$  das entradas, ponderadas pela relação  $R_f/R_i$  (sendo  $i$  o índice da entrada)

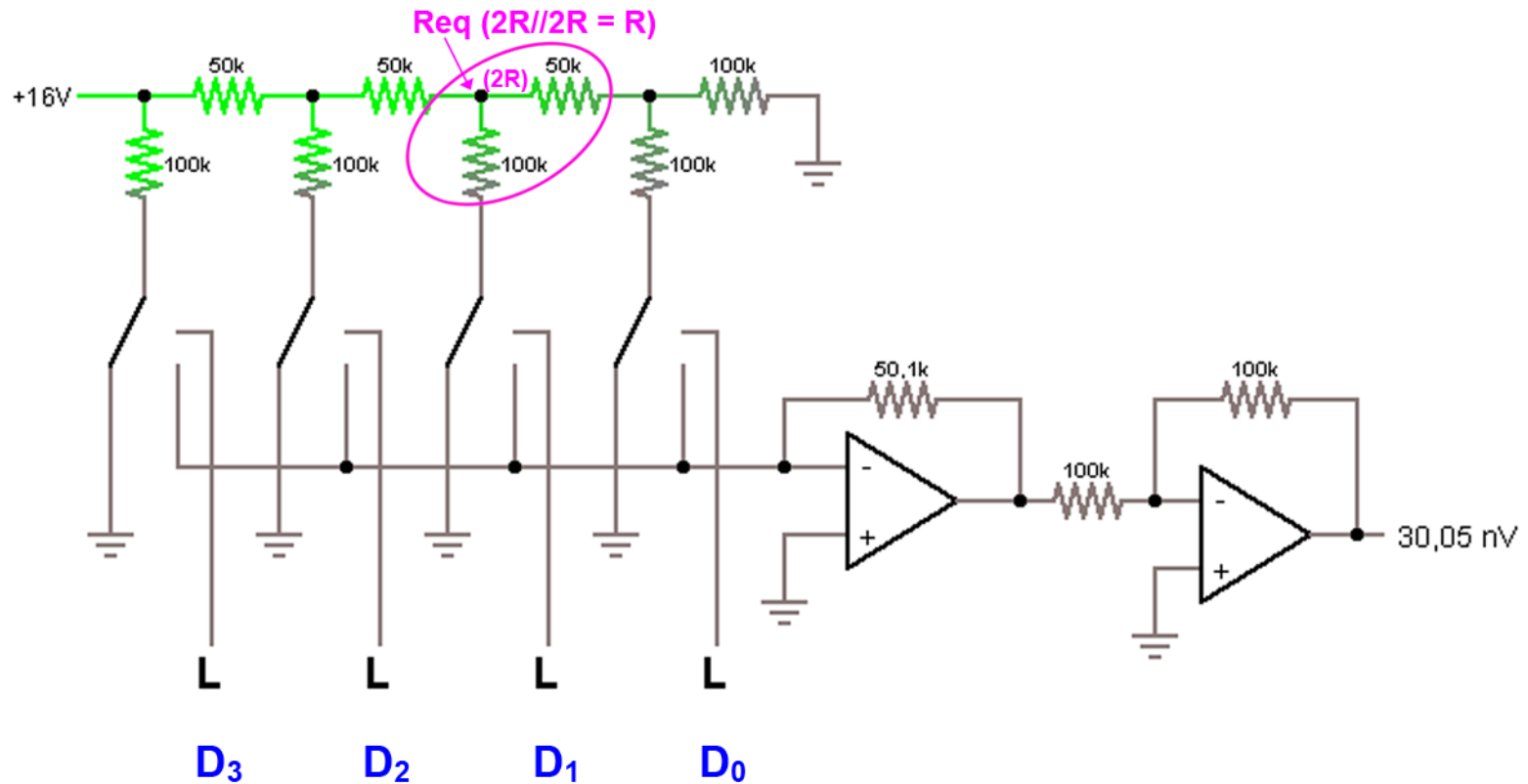
Conversor DIGITAL → ANALÓGICO malha R-2R:



## Conversor DIGITAL → ANALÓGICO malha R-2R:

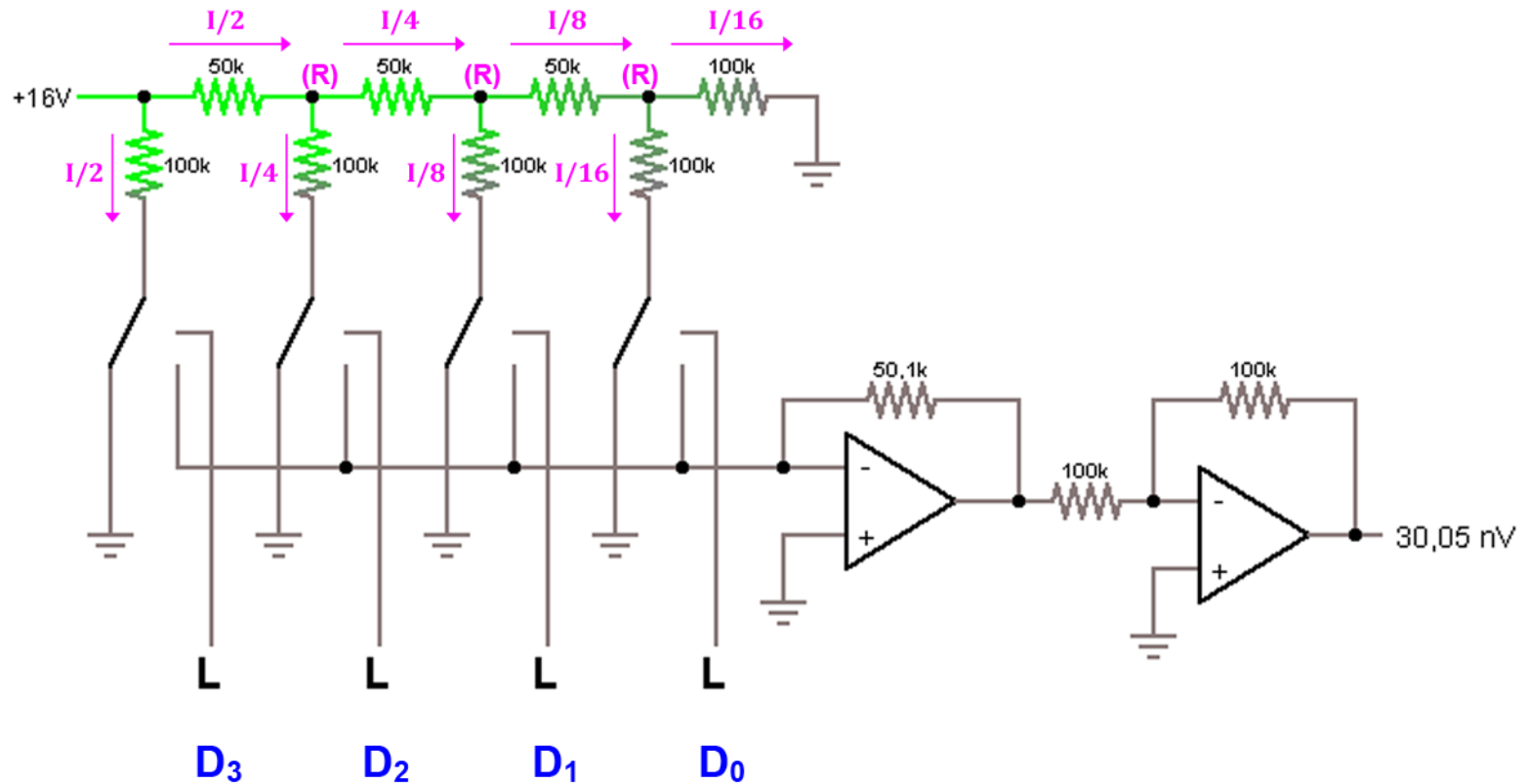


## Conversor DIGITAL → ANALÓGICO malha R-2R:

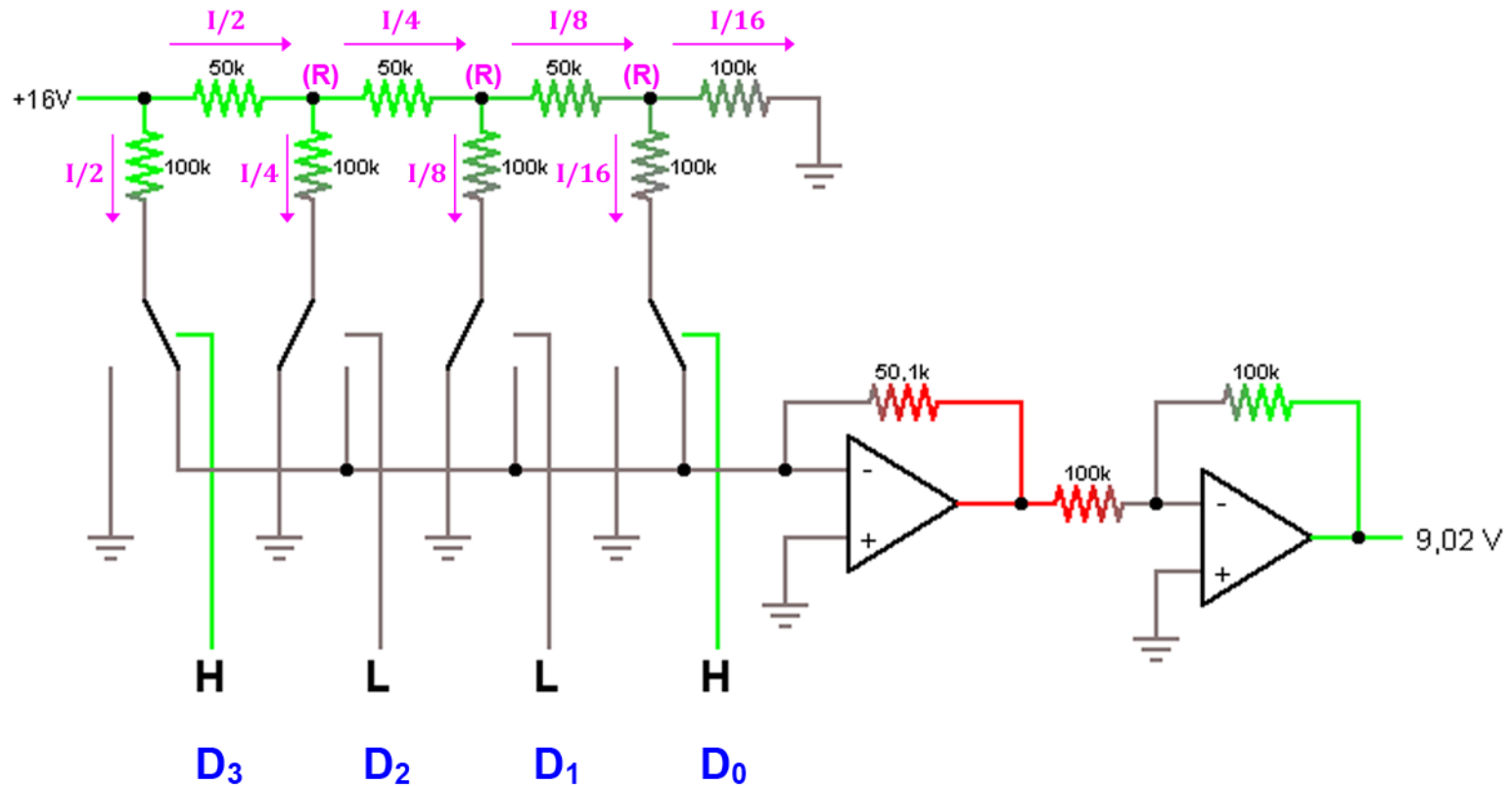


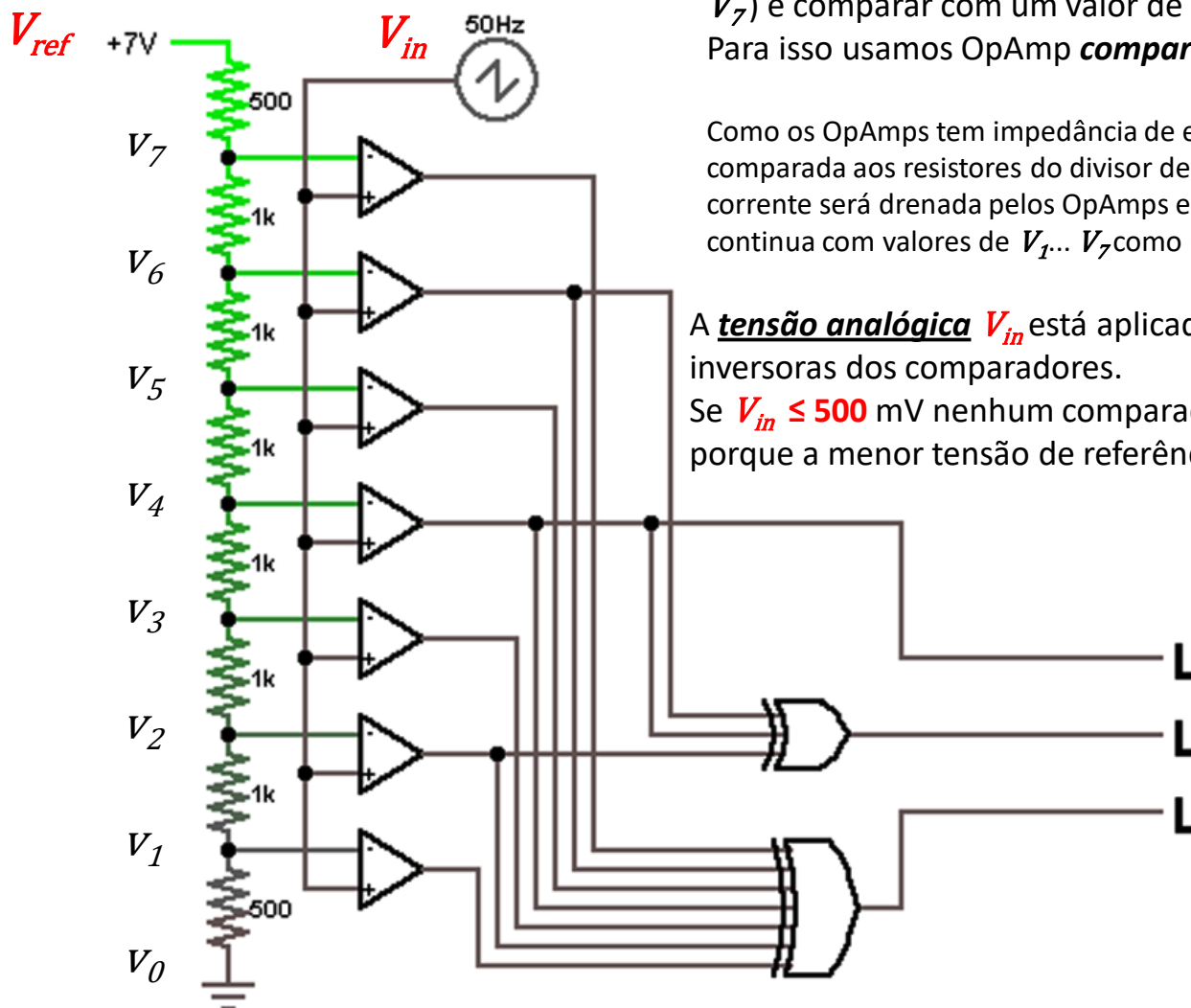


## Conversor DIGITAL → ANALÓGICO malha R-2R:



## Conversor DIGITAL → ANALÓGICO malha R-2R:



Conversor ANALÓGICO → DIGITAL (*flash*):

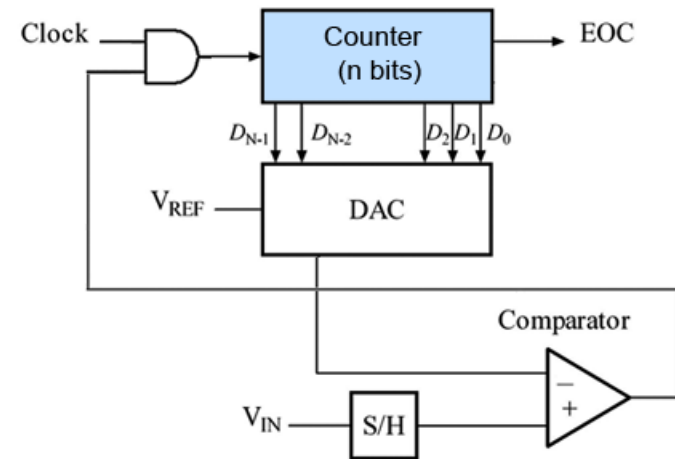
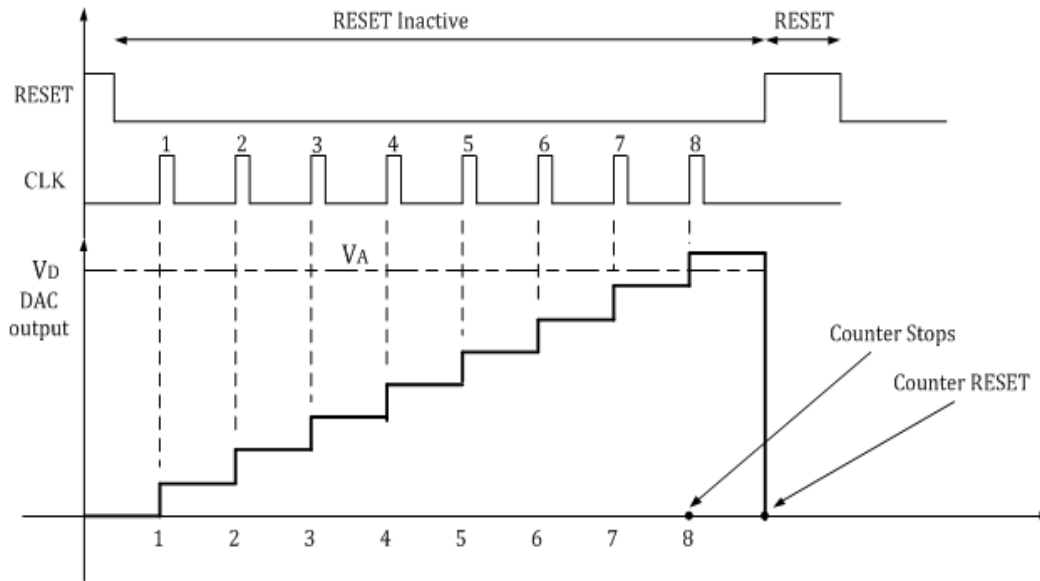
Vamos usar esta “escada” de níveis de tensão (de  $V_1$  até  $V_7$ ) e comparar com um valor de entrada analógico. Para isso usamos OpAmp **comparadores de tensão**.

Como os OpAmps tem impedância de entrada muito alta comparada aos resistores do divisor de tensão, quase nenhuma corrente será drenada pelos OpAmps e o divisor de tensão continua com valores de  $V_1 \dots V_7$  como calculado anteriormente.

A tensão analógica  $V_{in}$  está aplicada a todas as entradas inversoras dos comparadores.

Se  $V_{in} \leq 500$  mV nenhum comparador tem saída = ‘1’, porque a menor tensão de referência é  $V_1 = 500$  mV.

## Conversor ANALÓGICO → DIGITAL (single-slope e counter-slop ):



Conversor A/D counter-slope

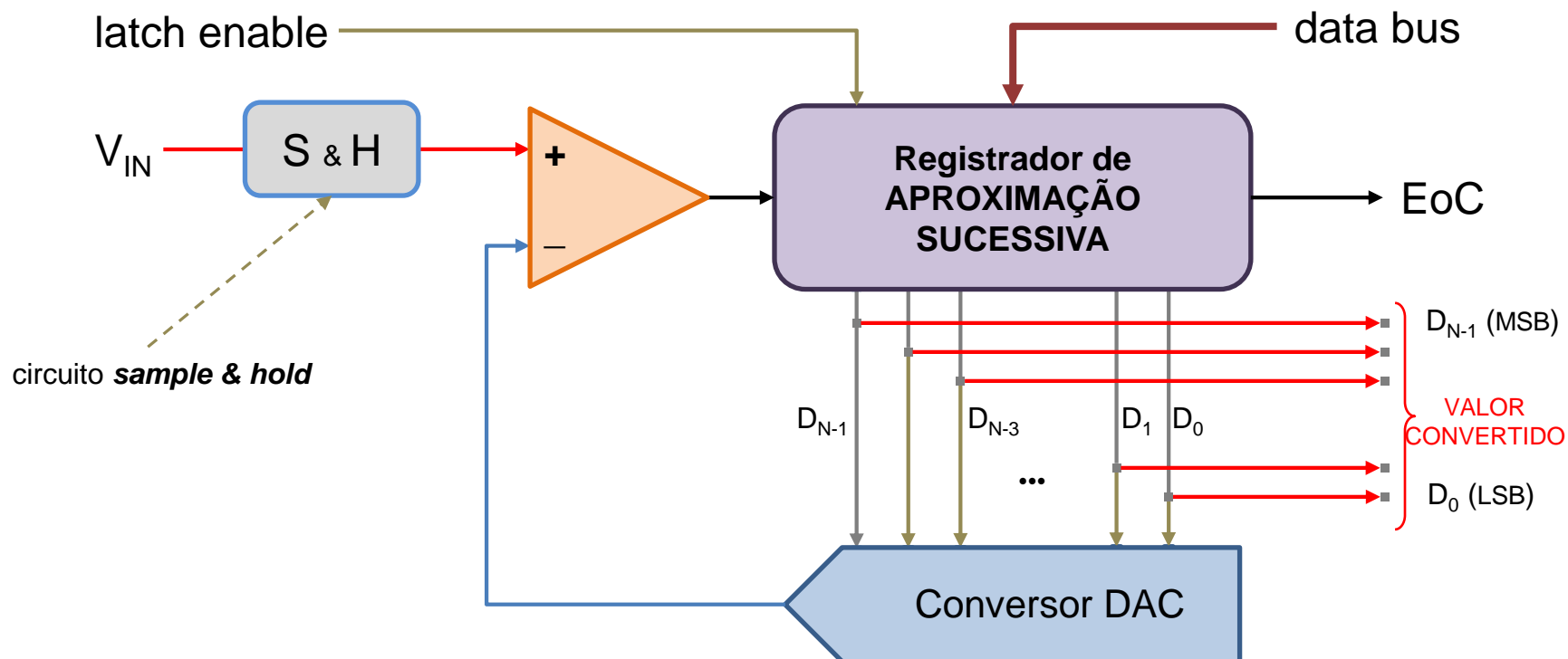
### PROS

- ▲ baixo custo
- ▲ bom para aplicações em baixa resolução (como alguns níveis de luminosidade em LED).

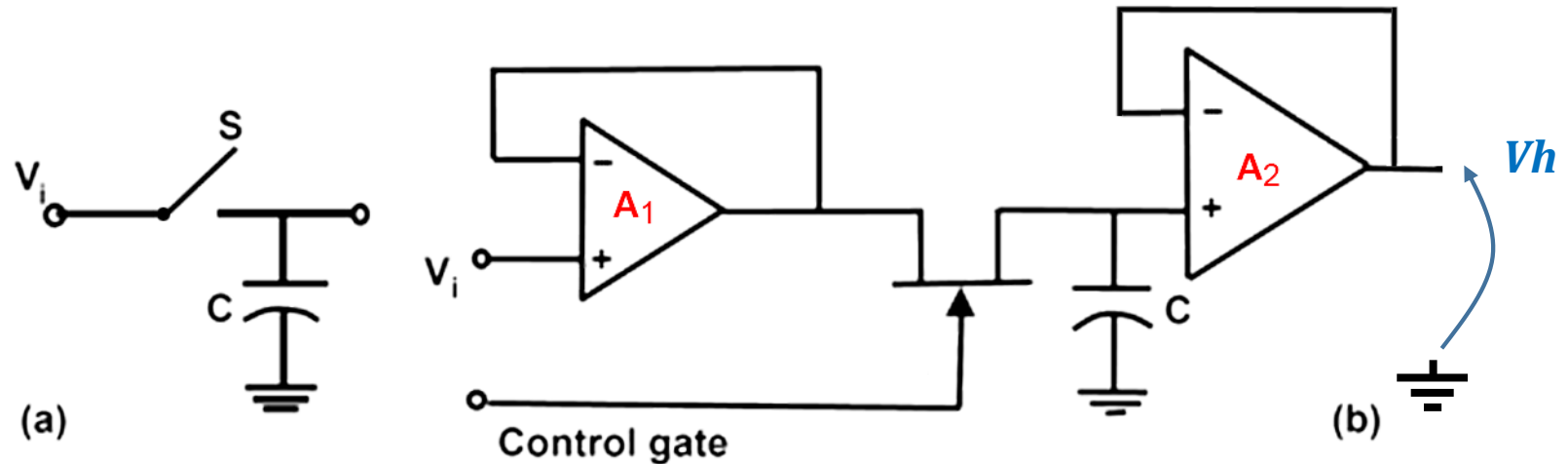
### CONS

- ▼ resultado da conversão é sensível a erros nos valores dos componentes
- ▼ lento
- ▼ efeitos adversos devido a "ruídos"
- ▼ a precisão depende do uso de componentes externos de precisão

## Conversor ANALÓGICO → DIGITAL (*aproximação sucessiva - SAR*):

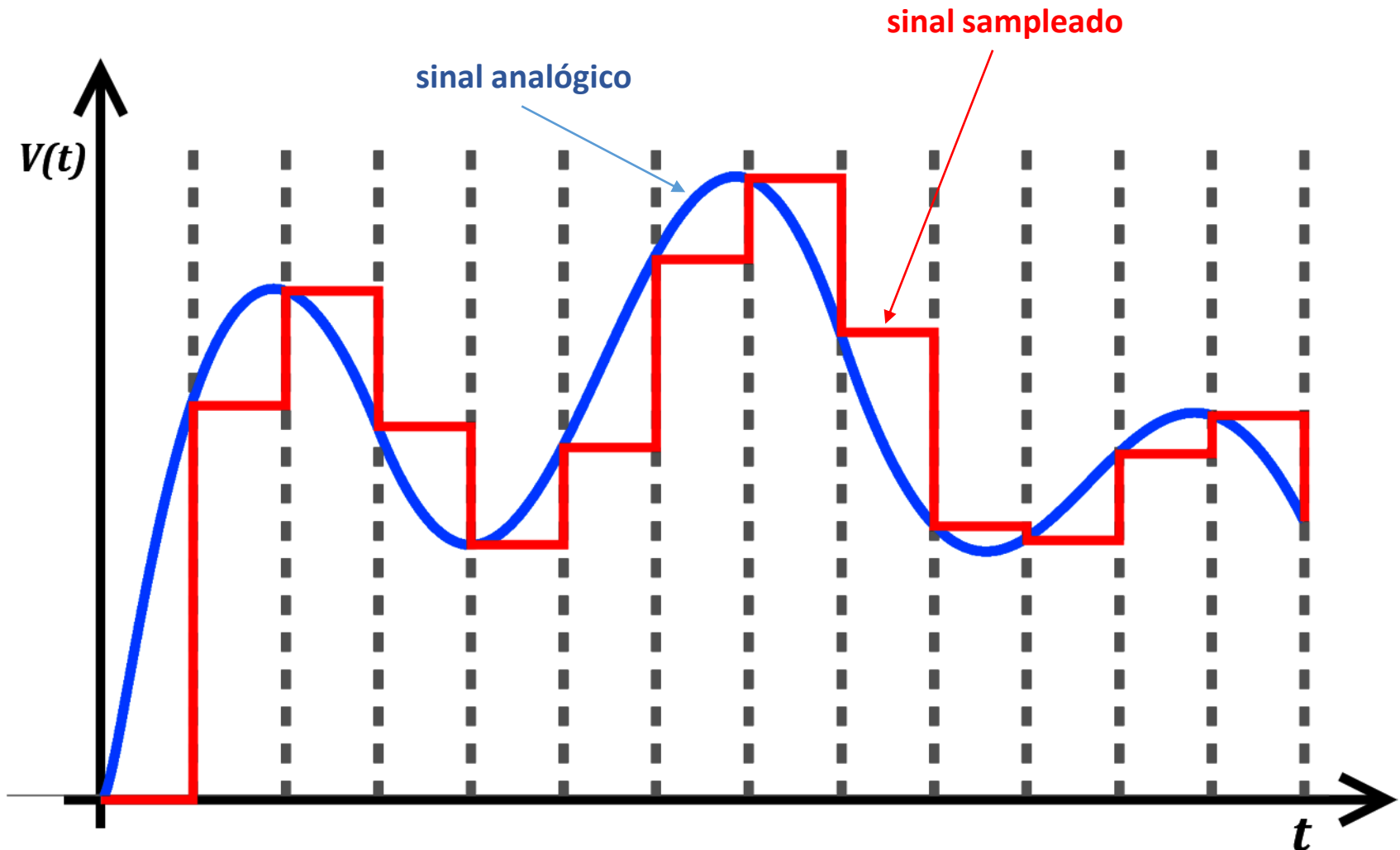


Em vez de gerar uma rampa contínua (*com um contador*), usa-se um **registrador** que faz **aproximações sucessivas**, do MSB ao LSB de forma a chegar rapidamente ao valor de tensão na saída do conversor DAC que seja igual à entrada analógica  $V_{IN}$ .  
 Sample & Hold (S&H) é um circuito que retém o valor de  $V_{IN}$  durante a conversão.  
 É um conversor **rápido** (necessita no máximo N pulsos de clock para N bits no conversor).

Conversor ANALÓGICO → DIGITAL: circuito eletrônico *sample & hold*

- (a) A chave **S** fecha e carrega o capacitor **C** durante a tomada da amostra. Depois a chave é aberta e o capacitor retém a tensão durante a conversão.
- (b) eletronicamente, um amplificador “copia” a tensão **Vi** (tensão analógica de entrada) e um transistor FET faz o papel de chave. O sinal “control gate” ativa o transistor. O capacitor **C** carrega rapidamente via **A1** (lembrar que o OpAmp tem baixa impedância de saída). Contudo, o capacitor **C** quase não descarrega depois que o FET é desativado (lembrar que o OpAmp A2 tem altíssima impedância de entrada). O sinal de saída **Vh** é o sinal que será aplicado ao comparador (ou outro circuito do conversor) como o sinal de **tensão hold** – estável para ser convertido pelo conversor.

Conversor ANALÓGICO → DIGITAL: circuito eletrônico *sample & hold*









- ARM (STM32F103C8) **tem 2 conversores** analógico/digital:
  - 12 bits aproximação sucessiva (usa 14 clks devido ao Sample & Hold)
  - 18 MUX analógicos – **até 16 canais externos** e 2 internos (*nosso 10 canais*)
  - Gera interrupção ao final da conversão
  - Conversão simples ou contínua
  - *Sampling time* (taxa de leitura) programável para cada canal
  - Opcionalmente, pode-se disparar conversão por sinal externo
  - Tempo de conversão (1,17 μs quando STM32F103xx = 24 MHz)
  - Pode gerar DMA durante uma conversão regular para um canal
- Fonte para ADC : 2,4 V até 3,6 V
- Valor do sinal analógico:  $V_{REF-}$  (terra analógico)  $\leq V_{in} \leq V_{REF+}$

◇ Conversores ADC no ARM – STM32F103

- Cada μC ARM (STM32F1xxx) tem no mínimo 2 conversores AD (ADC) cujo diagrama de bloco é mostrado à direita:

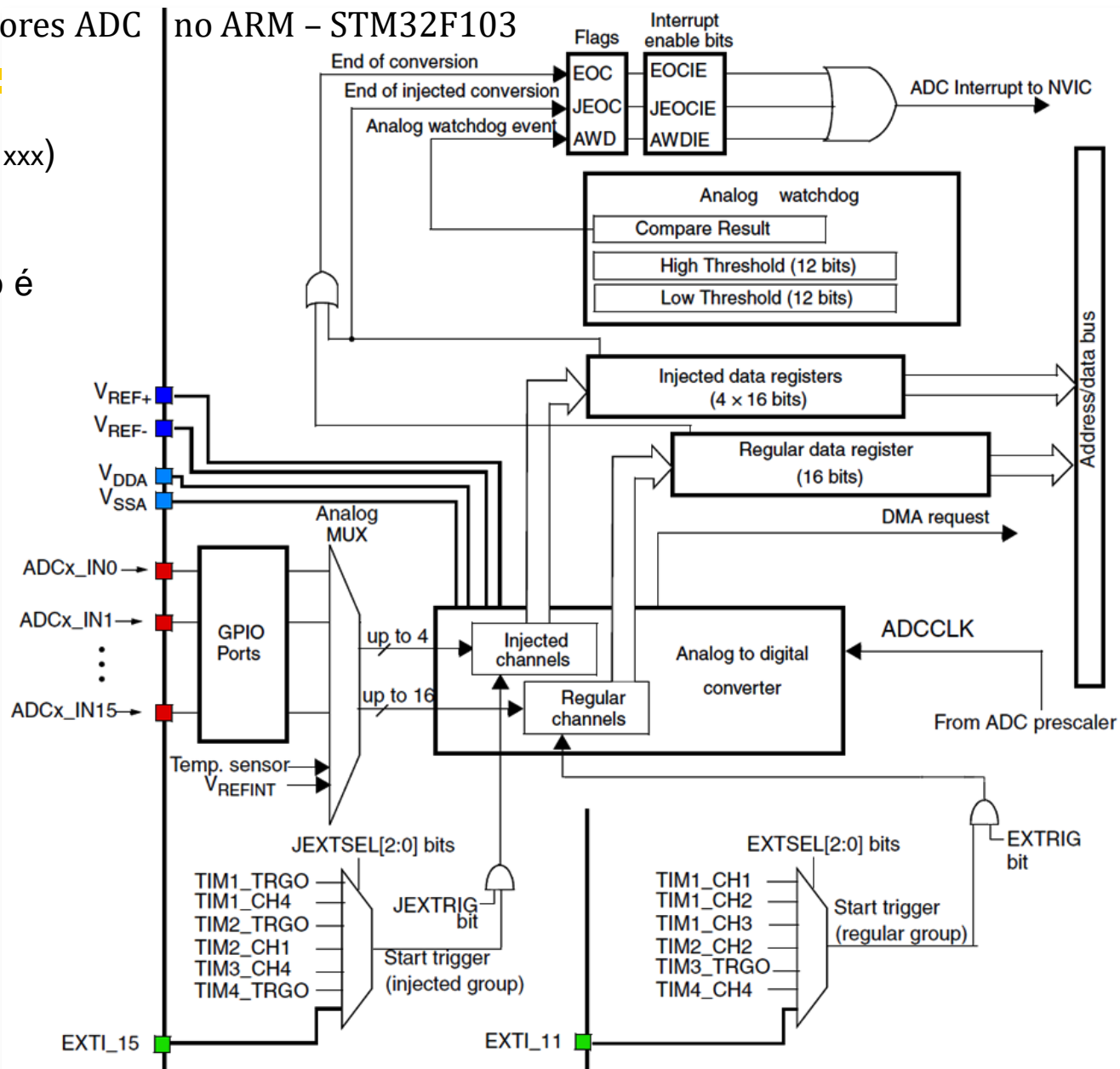


Figure 24. Single ADC block diagram

# ◇ Conversores ADC no ARM – STM32F103

- Cada μC ARM (STM32F1xxx) tem no mínimo 2 conversores AD (ADC) cujo diagrama de bloco é mostrado à direita:

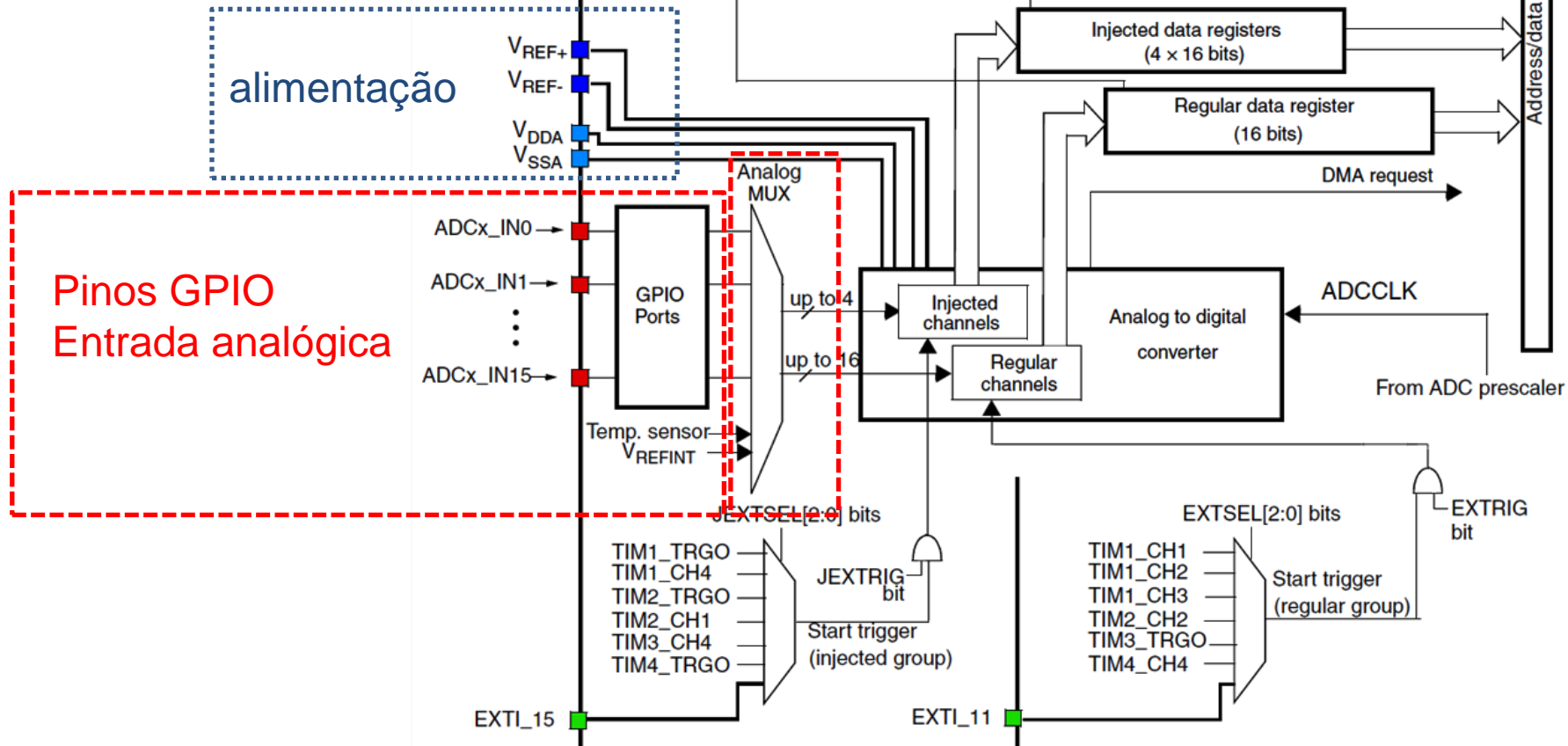


Figure 24. Single ADC block diagram

◇ Conversores ADC no ARM – STM32F103

- Cada μC ARM (STM32F1xxx) tem no mínimo 2 conversores AD (ADC) cujo diagrama de bloco é mostrado à direita:

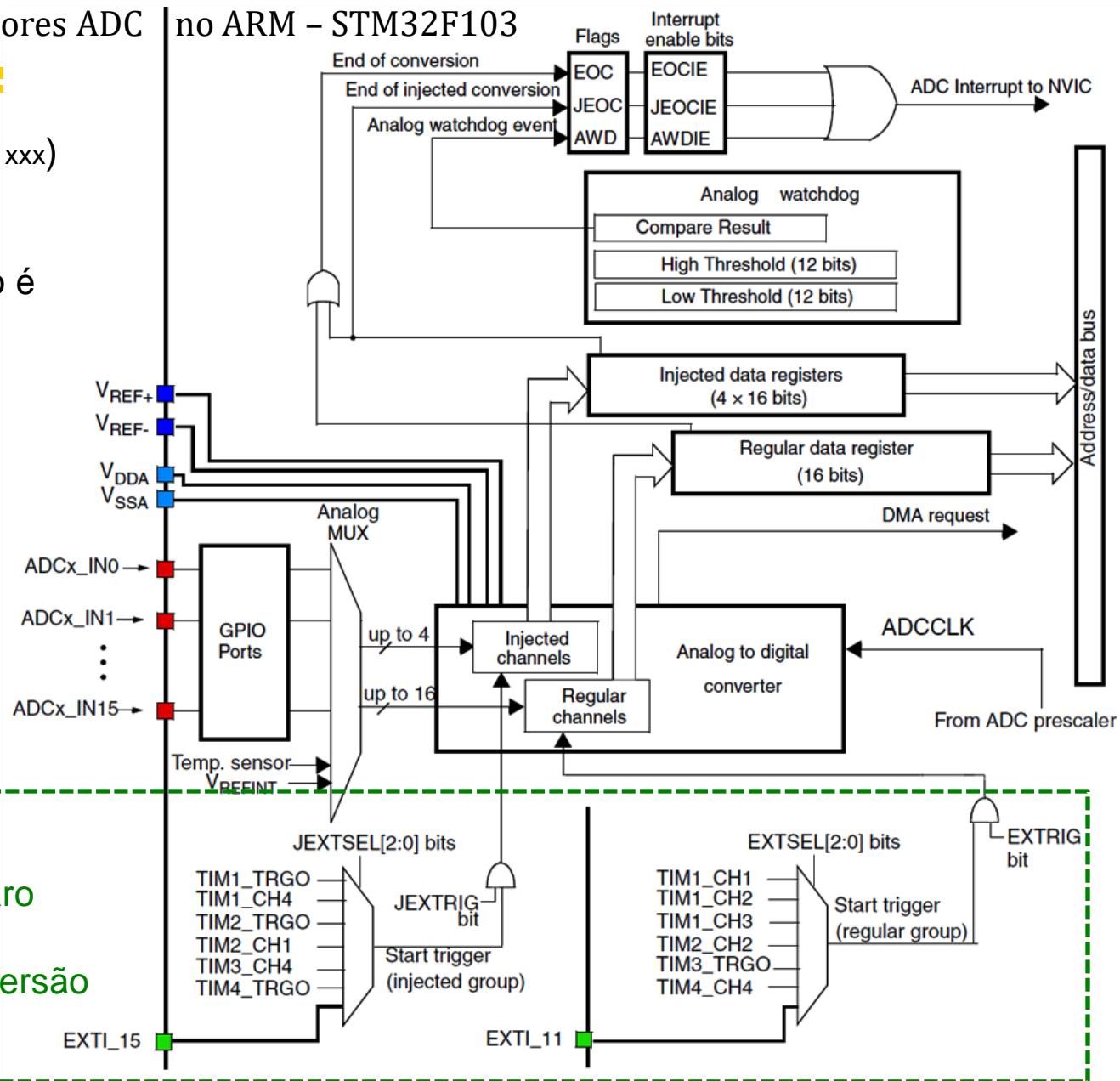


Figure 24. Single ADC block diagram

◇ Conversores ADC no ARM – STM32F103

- Cada μC ARM (STM32F1xxx) tem no mínimo 2 conversores AD (ADC) cujo diagrama de bloco é mostrado à direita:

Temos que programar:

- Freq ADCCLK
- Pino GPIO analógico
- Sequencia de canais (*injected=4 // regular=16 convs*)
- Condição de disparo
- Interrupção ou DMA
- Analog Watchdog
- (calibração – se necessário)

Além do clock de bus, das GPIO e timers (já vistos em aulas anteriores...)

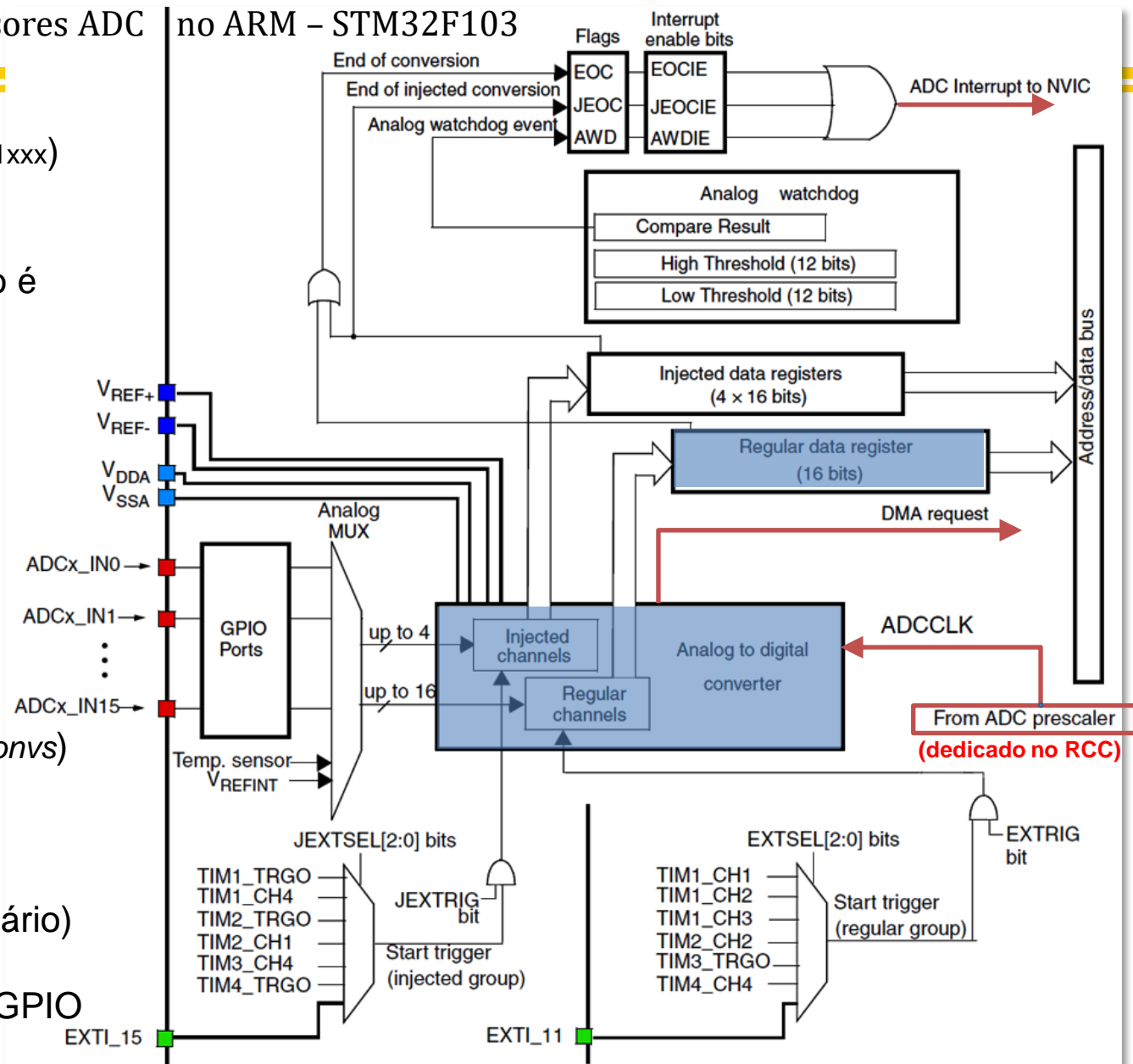


Figure 24. Single ADC block diagram

## ➤ MODOS DE CONVERSÃO:

### **Modo de conversão única**

No modo de conversão única, o ADC faz uma conversão apenas e pára.

### **Modo de conversão contínua**

No modo de conversão contínua, o ADC inicia outra conversão assim que finaliza a conversão anterior. Após cada conversão:

- Se um **canal normal** foi convertido:
  - Os dados convertidos são armazenados no registrador ADC\_DR de 16 bits
  - O flag (sinalizador) EOC (*End Of Conversion*) está configurado (setado)
  - Uma interrupção é gerada se o EOCIE estiver configurado.
- Se um **canal injetado** foi convertido:
  - Os dados convertidos são armazenados no registrador ADC\_DRJ1 de 16 bits
  - O flag JEOC (*End Of Conversion Injected*) está configurado
  - Uma interrupção é gerada se o bit JEOCIE estiver configurado.

## ➤ MODOS DE CONVERSÃO:

**Modo de conversão única:** o ADC faz uma conversão apenas e pára.

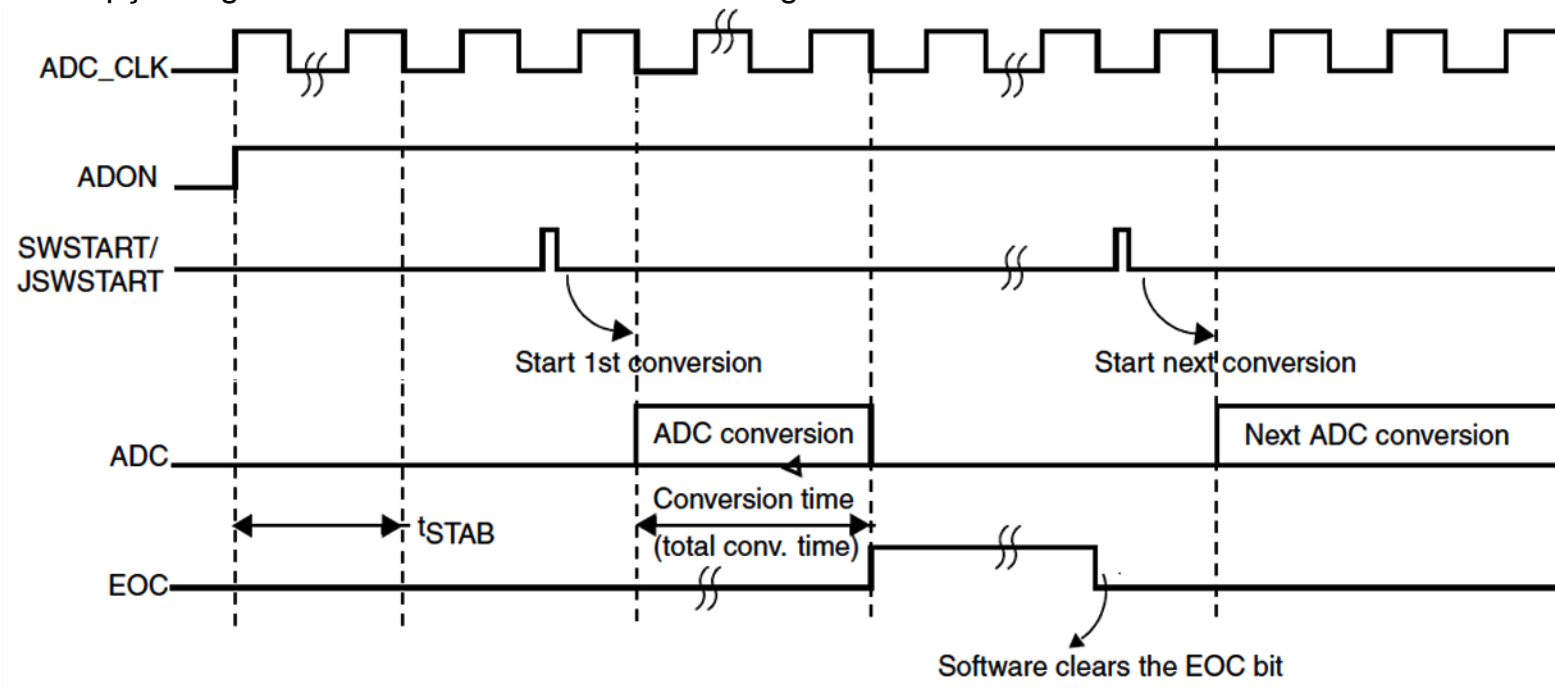
**Modo de conversão contínua:** o ADC inicia outra conversão assim que finaliza a anterior.

Após cada conversão: • Se um **canal normal** foi convertido:

- Os dados convertidos são armazenados no registrador ADC\_DR de 16 bits
- O flag (sinalizador) EOC (*End Of Conversion*) está configurado (setado)
- Uma interrupção é gerada se o EOCIE estiver configurado.

• Se um **canal injetado** foi convertido:

- Os dados convertidos são armazenados no registrador ADC\_DRJ1 de 16 bits
- O flag JEOC (*End Of Conversion Injected*) está configurado
- Uma interrupção é gerada se o bit JEOCIE estiver configurado.





## ➤ MODOS DE CONVERSÃO

**(Scan mode) Modo de varredura** : usado para digitalizar um grupo de canais analógicos. O modo SCAN pode ser selecionado configurando o bit SCAN no registro ADC\_CR1. Uma vez que este bit está configurado, o ADC verifica todos os canais selecionados nos registros ADC\_SQRx (para canais regulares) ou no ADC\_JSQR (para canais injetados). Uma única conversão é realizada para cada canal do grupo. Após cada final de conversão, o próximo canal do grupo é convertido automaticamente. Se o CONT bit estiver SETADO, a conversão não pára no último canal de grupo selecionado, mas continua novamente a partir do primeiro canal de grupo selecionado. Ao usar o modo de varredura, o bit DMA deve ser configurado e *o controlador de acesso direto à memória é usado para transferir os dados de canais de grupo regulares para SRAM* após cada atualização do registro ADC\_DR. O canal injetado é sempre armazenado nos registros ADC\_JDRx.

**(Discontinuous mode) Modo descontínuo**: usado para converter uma sequência curta de  $n$  conversões ( $n \leq 8$ ), parte da seq. de conversões selecionadas nos registros ADC\_SQRx.

Ex:  $n = 3$ , canais a serem convertidos = 0, 1, 2, 3, 6, 7, 9, 10

1º trigger: sequência convertida 0, 1, 2. Um evento EOC é gerado em cada conversão

2º trigger : sequência convertida 3, 6, 7. Um evento EOC é gerado em cada conversão

3º trigger : sequência convertida 9, 10. Um evento EOC é gerado em cada conversão

4º trigger : sequência convertida 0, 1, 2. Um evento EOC é gerado em cada conversão

*(ver no manual... Analog watchdog , Temperature sensor, calibração, DMA request)*



O básico do bloco ADC STM32:

As palavras "modo" e "conversão" no datasheet STM32 são um pouco ambíguas, a menos que você as tome ao pé da letra.

➤ Por exemplo, existem dois *modos* de operação que são:

*Modo independente.* É assim como o uso típico do ADC. Cada unidade ADC está operando sozinha e sem nenhuma dependência mútua.

*Modo duplo.* Neste modo, dois ADCs são convertidos simultaneamente ou com algum atraso (literalmente insignificante). Duas unidades ADC funcionando juntas como se fossem uma única unidade.

➤ Com relação à *conversão A/D*, ela pode ser:

*Conversão única (Single Conversion).* Uma conversão de amostra em um determinado instante.

*Conversão contínua (Continuous Conversion).* Coleta e conversão de amostras sem parar, continuamente.

*Conversão descontínua (Discontinuous Conversion).* Conversão sequencial de alguns canais em um grupo.

*Conversão de digitalização (Scan Conversion).* Amostragem sequencial e conversão de uma série de canais um após o outro.

➤ Para iniciar a conversão A/D, uma unidade ADC precisa ser estimulada com um sinal de disparo:

*Disparador de software.* Faz conversão conforme demanda do programa codificado.

*Disparador de hardware.* Faz conversão de acordo com eventos de (e.g. interrupções externas ou eventos temporizados).

O básico do bloco ADC STM32:

- ✓ As conversões A/D são feitas em grupos.
- ✓ Os membros do grupo são canais ADC e não precisam ser múltiplos canais. Um grupo pode consistir de apenas um canal.
- ✓ Dentro destes grupos, os canais ADC são convertidos em uma base agendada. O bom é que, ao contrário da maioria dos μC, podemos programar quais canais pertencem a um grupo e com qual sequência da conversão A/D será iniciada.
- ✓ Também podemos definir o tempo de amostragem para cada canal separadamente.

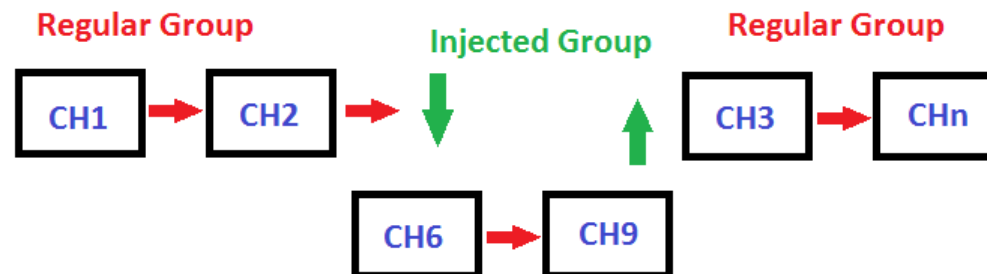
➤ Os grupos ADC podem ser categorizados da seguinte forma:

Grupo regular. Um determinado grupo fixo de canais ADC será regularmente convertido. Até 16 canais podem estar presentes em um grupo regular. *Um grupo regular é semelhante a um código que está sendo executado no loop principal.*



Regular Group Conversion

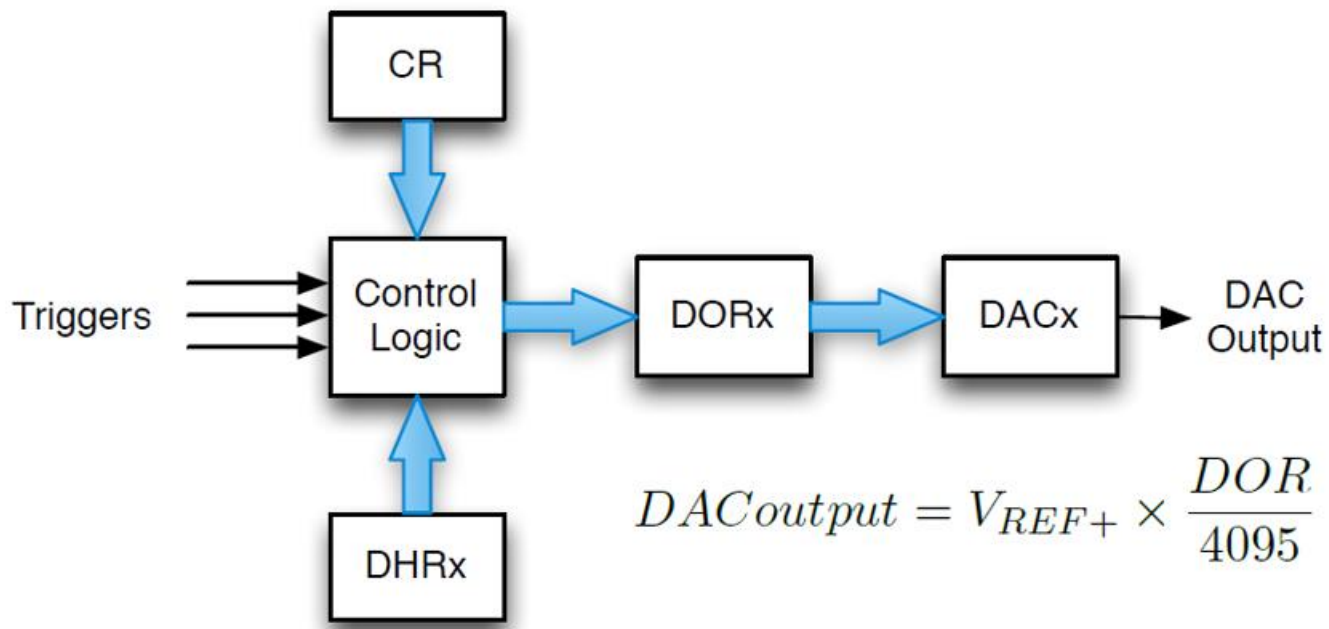
Grupo Injetado. Este grupo pode interromper uma conversão de grupo regular, pois **tem maior prioridade** sobre o primeiro. Até 4 canais podem estar presentes em um grupo injetado. Quando um grupo injetado está presente ou injetado em um grupo regular, todas as conversões de grupos regulares são interrompidas temporariamente. O grupo injetado é processado primeiro e depois os grupos regulares são retomados. *Um grupo injetado é análogo a ter uma interrupção sobre um código em execução.* Sozinho sem um grupo regular, um grupo injetado se comportará como um grupo regular.



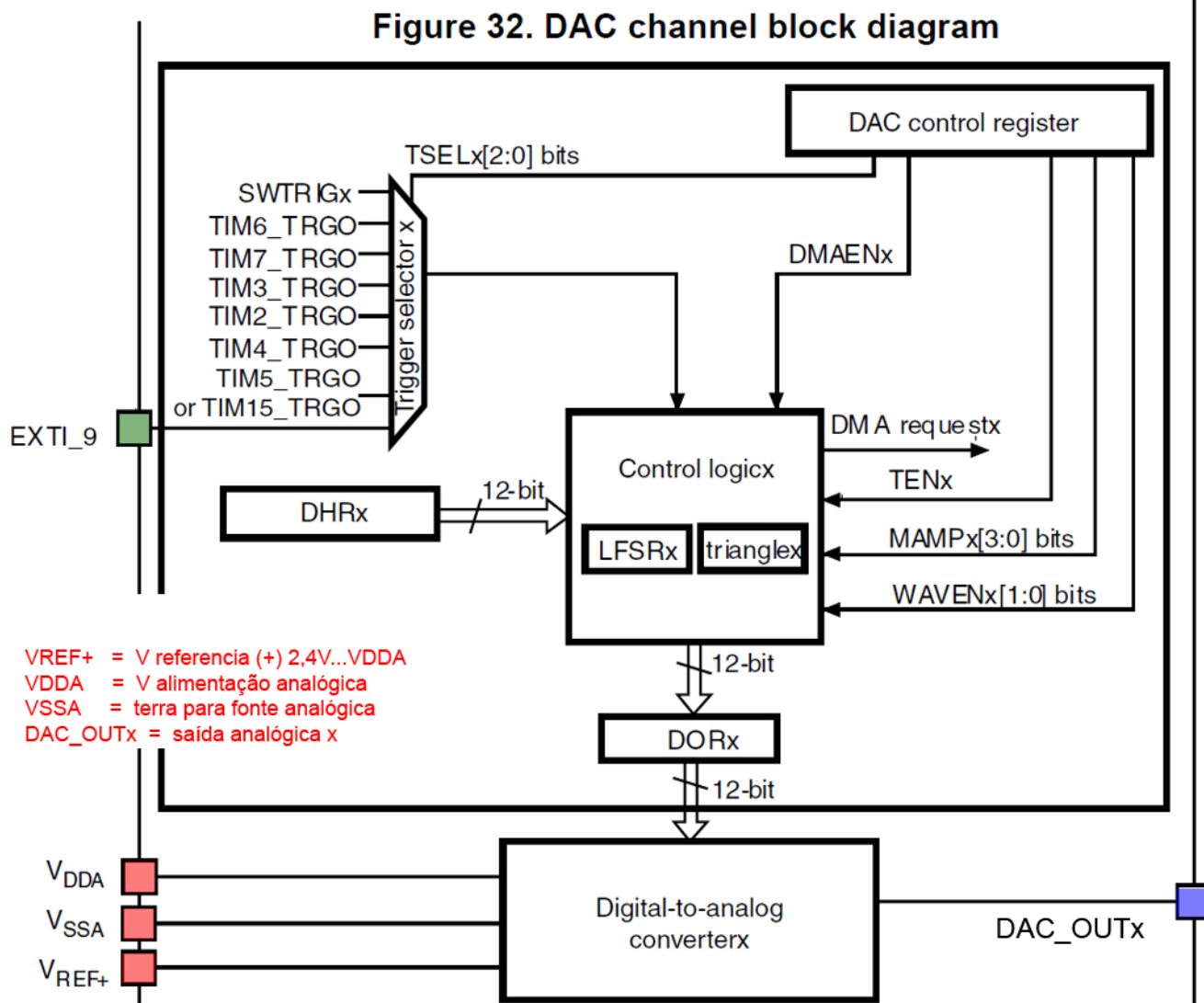
- Em alguns Cis da família STM32F1xx existem **2 conversores digital → analógico** configuráveis para 8 ou 12 bits, cujas saídas são nos pinos PA4 (DAC1) e PA5 (DAC2)...

Nosso chip **STM32F103C8.. não tem DAC !**

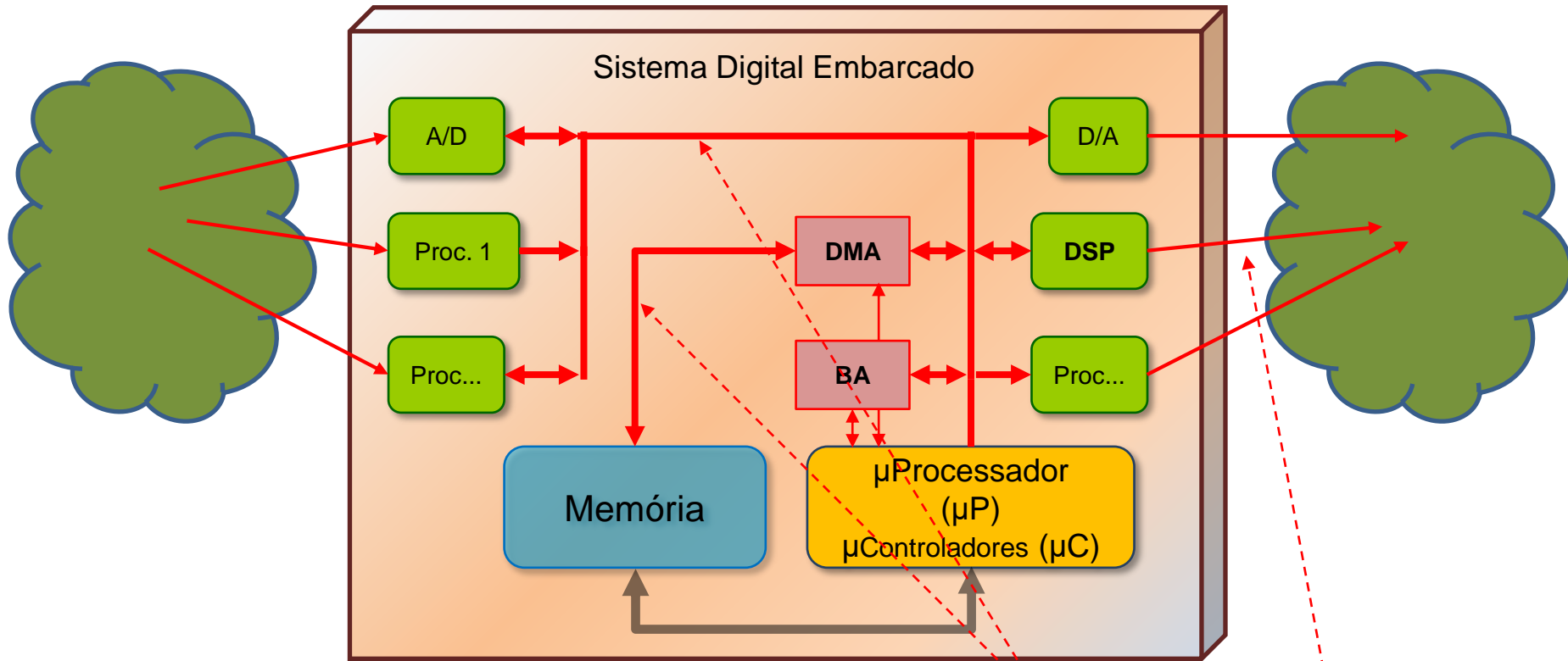
(devemos usar o PWM e um filtro externo – ou mudar o chip se for realmente necessário)



- Nos CIs ARM que possuem DAC, cada canal DAC tem o seguinte bloco:
- Os DACs também podem ser programados para gerar pseudo-ruído (usa LFSR - Linear feedback shift register) e podem gerar ondas triangulares, dente de serra, etc. (ver manual\0.



- Introdução DMA
- Introdução ao DMA O acesso direto à memória (DMA) é usado para fornecer transferência de dados de alta velocidade entre periféricos e memória, bem como da memória para a memória. Os dados podem ser movidos rapidamente pelo DMA sem qualquer ação de CPU. Isso mantém os recursos da CPU livres para outras operações.
- Os dois controladores DMA têm 12 canais no total (7 para DMA1 e 5 para DMA2), cada um dedicado a gerenciar pedidos de acesso a memória de um ou mais periféricos.
- Cada controlador possui um **árbitro** para tratar a prioridade entre os pedidos DMA.



Sist. Embarcados são compostos por processadores, memória, **barramentos** e **redes de comunicação** que interconectam processadores e memória, formando um subconjunto de **interfaceamento**.

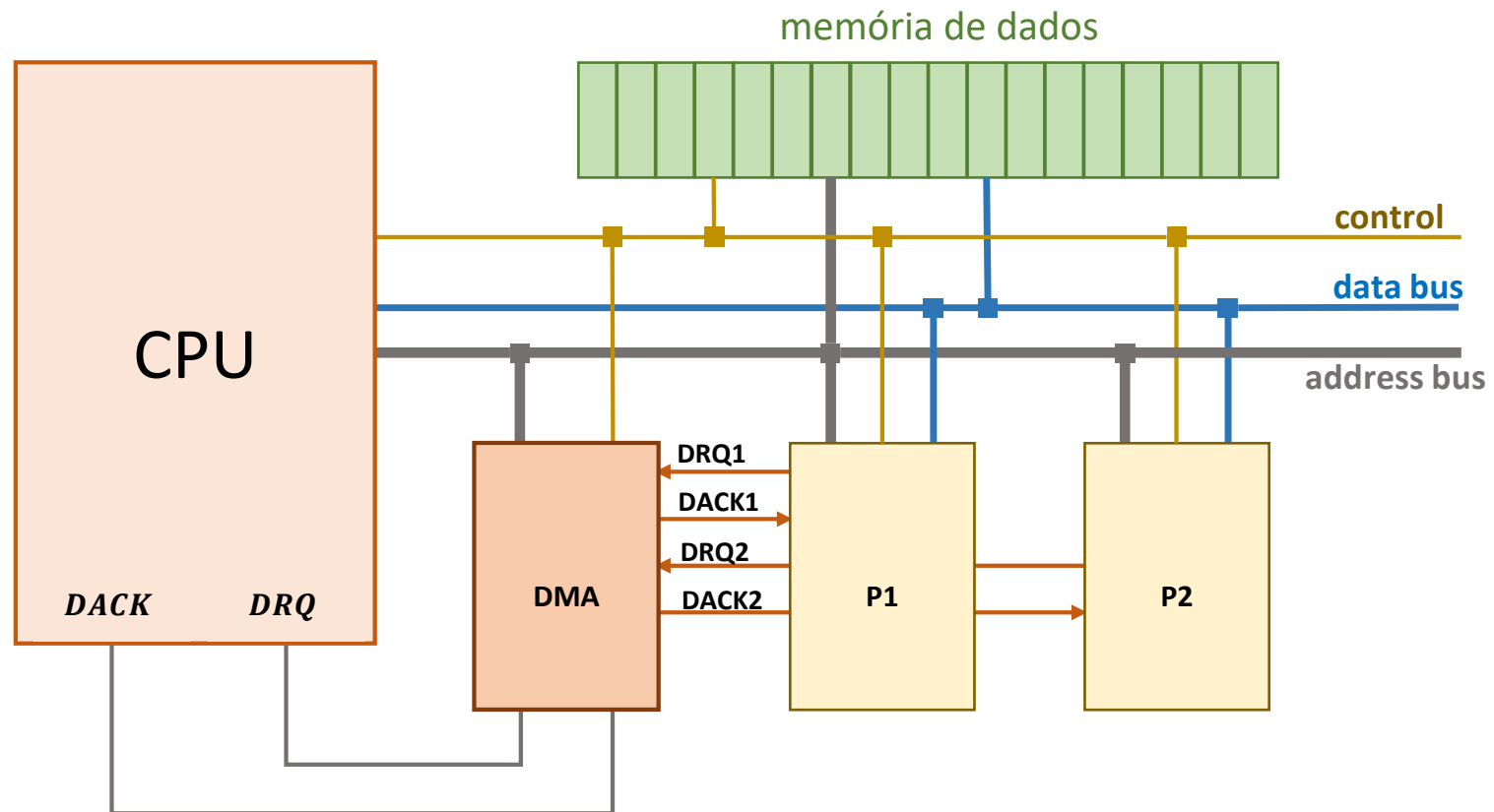
Processadores: - dedicados (standard – periféricos, ou customizados – criados pelo projetista )  
 - especializados (processadores de propósito específico)  
 - de uso geral (μProcessadores ou μControladores).

Armazenagem: implementado por sistemas de **memória**.

Interfaceamento: implementado por **barramentos** e **redes** de comunicação entre sistemas.

- Características principais do DMA
- • 12 canais independentes configuráveis (solicitações): 7 para DMA1 e 5 para DMA2;
- • Cada um dos 12 canais está conectado à solicitações DMA de hardware dedicado, o gatilho do software é suportado em cada canal. Essa configuração é feita por software;
- • As *prioridades* entre os pedidos de canais de um DMA são programáveis por software (e tem 4 níveis: ***muito alta***, ***alta***, ***média***, ***baixa***) ou *hardware* em caso de igualdade (a solicitação 1 tem prioridade sobre a solicitação 2, etc.);
- • Tamanho independente de transferência de origem e destino (***byte***, ***meia palavra***, ou ***palavra-32 bits***), emulando pacotes ou não. Os endereços de origem / destino devem estar alinhados no tamanho do dado (ou pacote);
- • Suporte para gerenciamento de buffer circular;
- • 3 sinalizadores de eventos (*DMA Half Transfer*, *DMA Transfer completo* e *DMA Transfer Error*) logicamente ou juntos em uma única solicitação de interrupção para cada canal;
- • Transferência de memória para memória, de periféricos para memória, de memória para periféricos, e de periféricos para periféricos;
- • Acesso aos periféricos Flash, SRAM, APB1, APB2 e AHB como fonte e destino;
- • Número programável de dados a serem transferidos: até 65536.

- Vários periféricos produzem dados em grande quantidade
- Usar a CPU para transferir cada bloco de dados para a RAM pode ser muito lento...
- Melhor seria usar um processador dedicado, especializado em transferir dados rapidamente;
- Esse processador se chama **Direct Memory Access controller (DMA)**





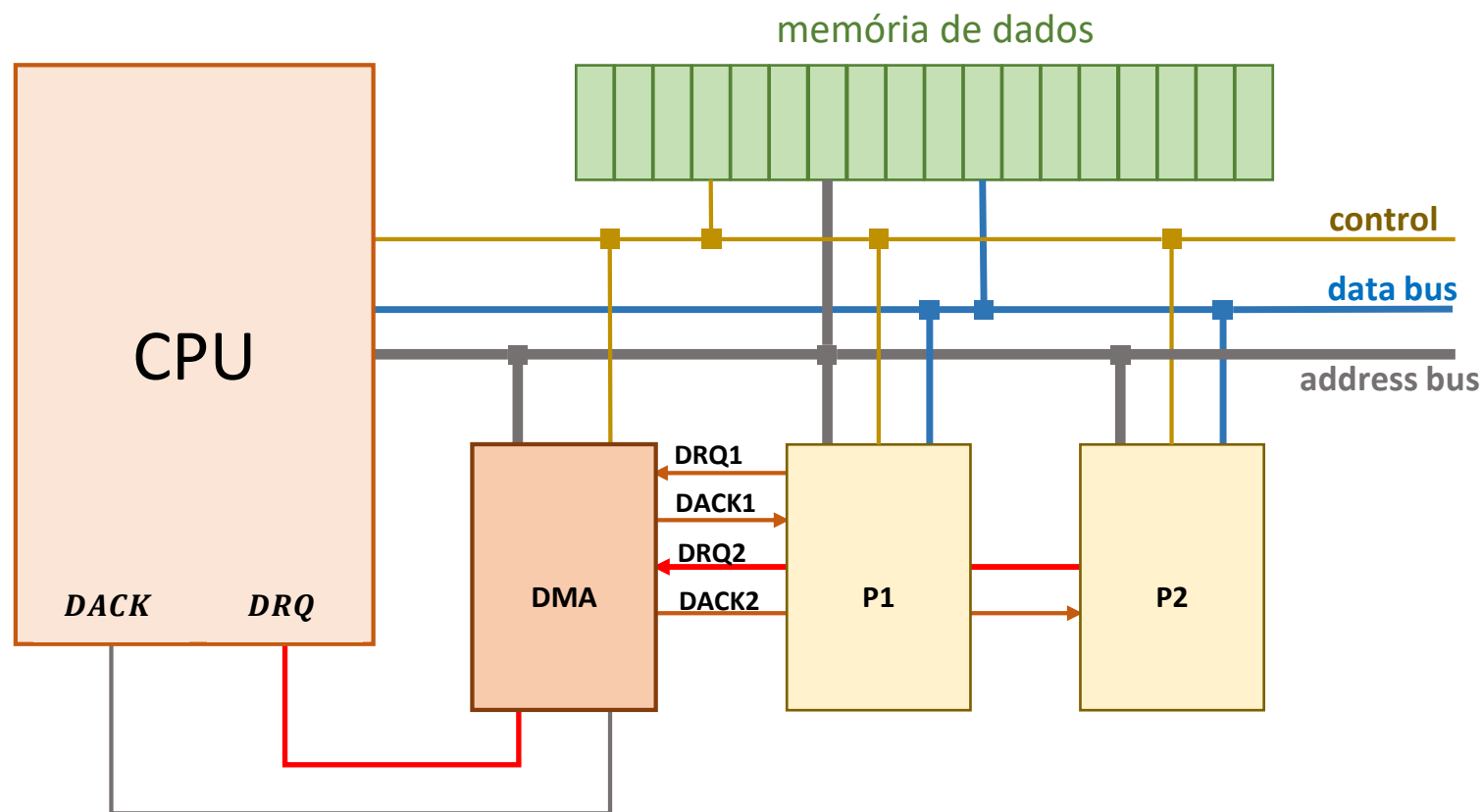
➤ Como funciona:

a) Para cada periférico, a CPU primeiro **configura** o DMAc com pelo menos 3 parâmetros:

- Endereço do periférico de onde os dados devem ser lidos / escritos
- Endereço da memória onde os dados devem ser escritos / lidos
- A quantidade total de bytes que será transferida na operação.

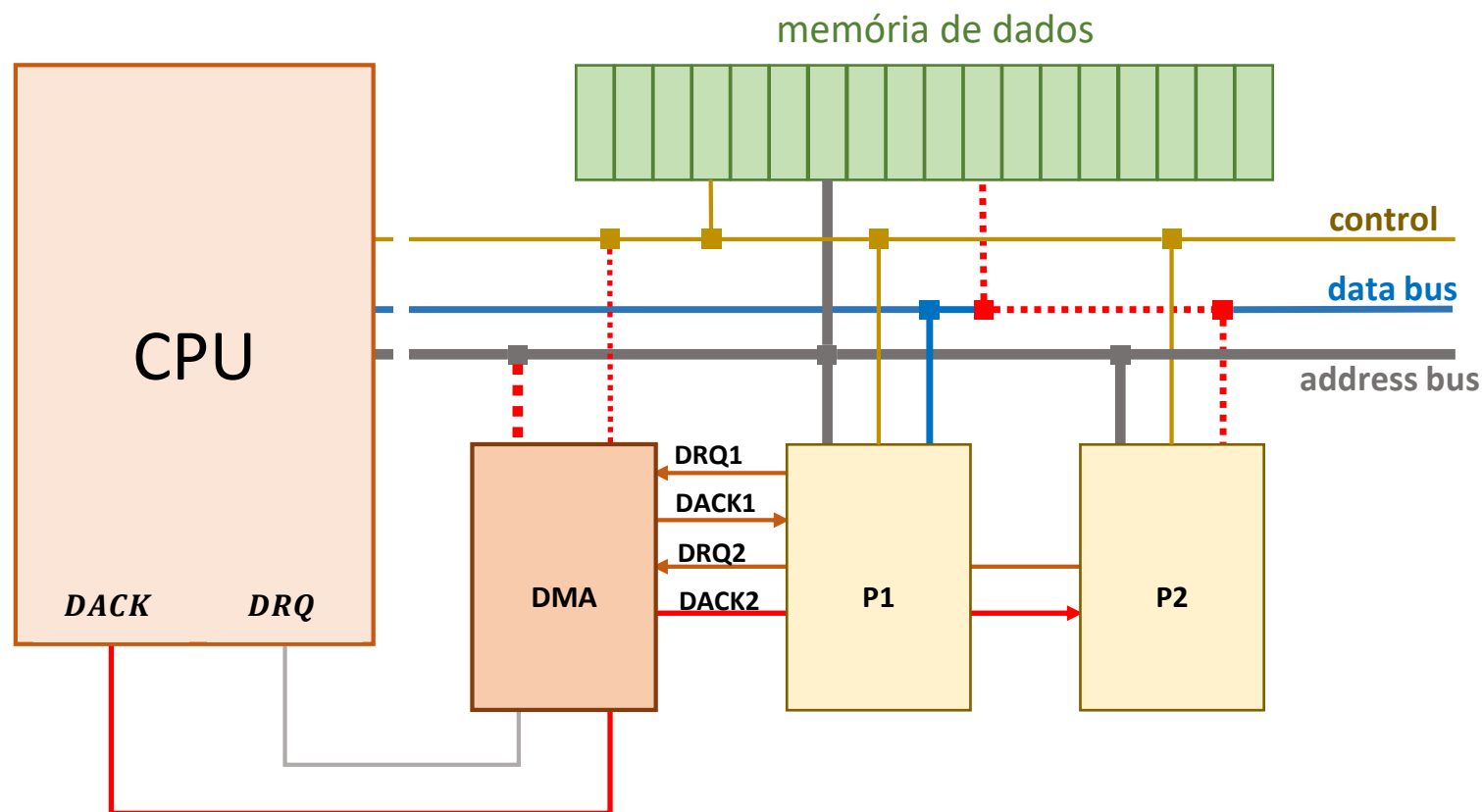
b) Periféricos geralmente têm buffers de memória; cheio, o periférico solicita serviço ao DMAc acionando **DRQx**

c) O DMAc solicita controle do bus ativando **DRQ** (data request) da CPU... ao final da instrução atual, a CPU pode liberar o controle do *data bus*, *address bus*, e os sinais de *controle*. Para avisar ao DMAc a CPU ativa **DACK** (data acknowledge).



## ➤ Como funciona:

- d) O DMAc então avisa o periférico que o serviço dele será atendido ativando DACKx (data acknowledge)
- e) A CPU então coloca todos os sinais dos barramentos em 3-state (nesse momento a CPU pode executar instruções na cache interna. No caso do ARM, a matriz de barramentos permite busca de instrução e até acesso à memória)
- f) O DMAc assume o controle do bus de endereços e os sinais de controle – os dados são fornecidos pelo periférico!
- g) Quando chega ao final da transferência do bloco o DMAc desativa DACKx para o periférico,
- h) Pode ser que o DMAc peça interrupção à CPU para que ela organize (fique ciente) dados novos na memória.



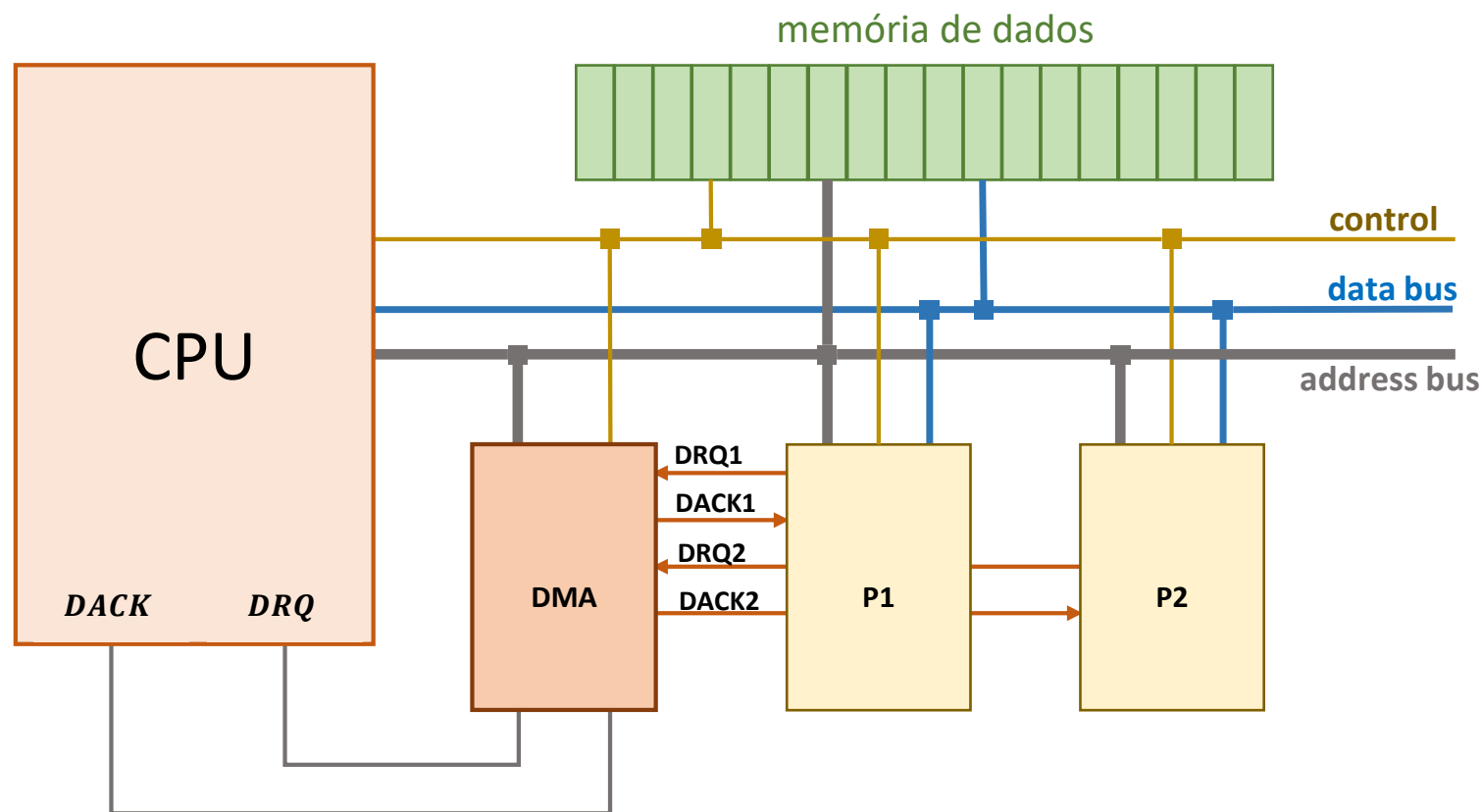
➤ Como funciona:

i) A CPU atende um ISR do DMA e retoma o controle do barramento (continua a execução do programa principal).

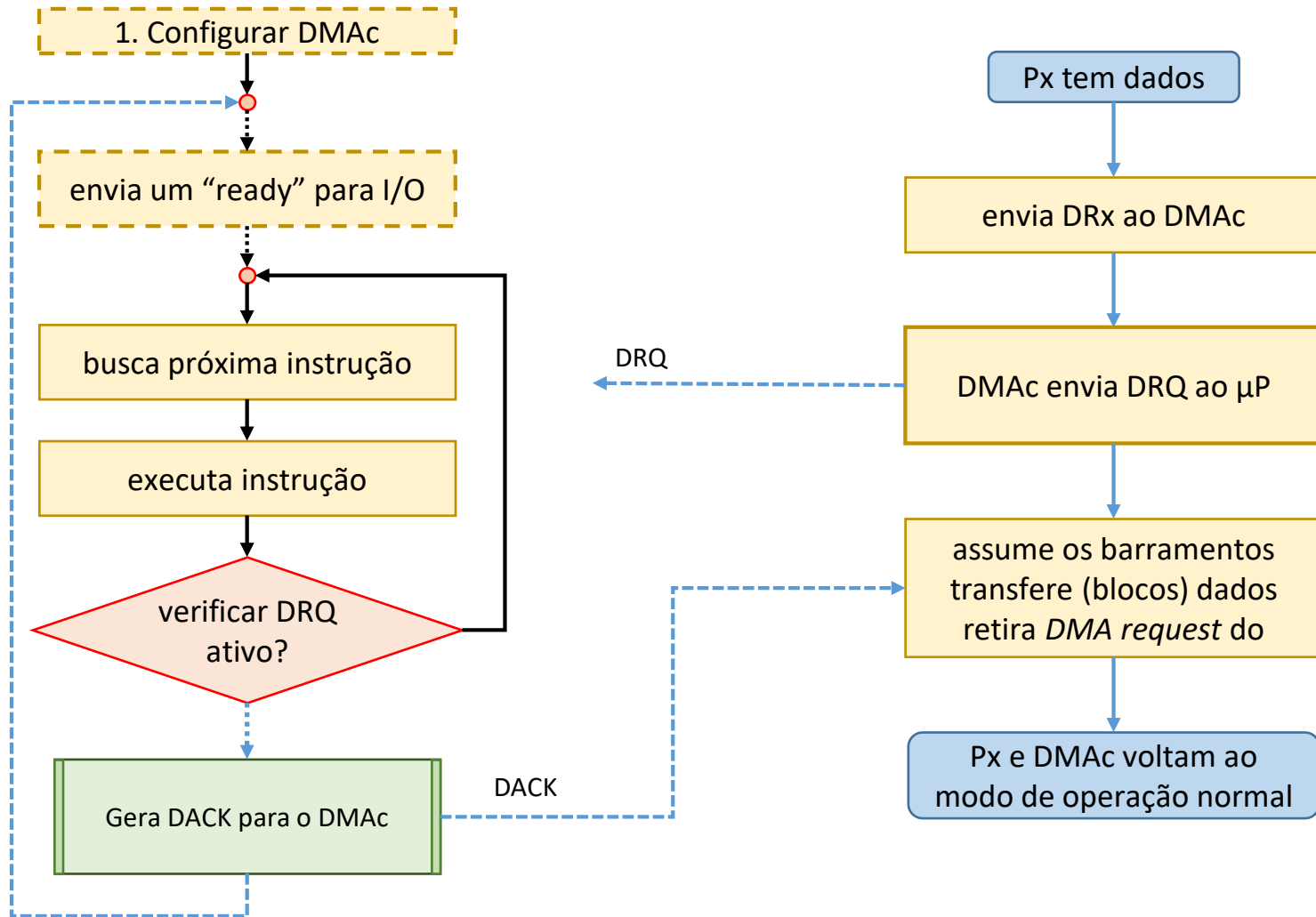
Note-se que a transferência usando DMA pode ser de um único byte ou de um bloco (buffer) de memória.

Se a CPU roda o programa principal e acontecer um DMA (especialmente 1 palavra apenas) ela não precisa salvar conteúdo na pilha, nem recuperá-los ao retorno da interrupção. Isso aconteceria em cada palavra/byte transferido.

Se a CPU tem um cache primário (L1) e/ou um cache secundário razoável (L2) é possível que DMA aconteça em paralelo à execução do programa principal – Isso é especialmente verdade nos sistemas mais modernos (e.g. ARM)...

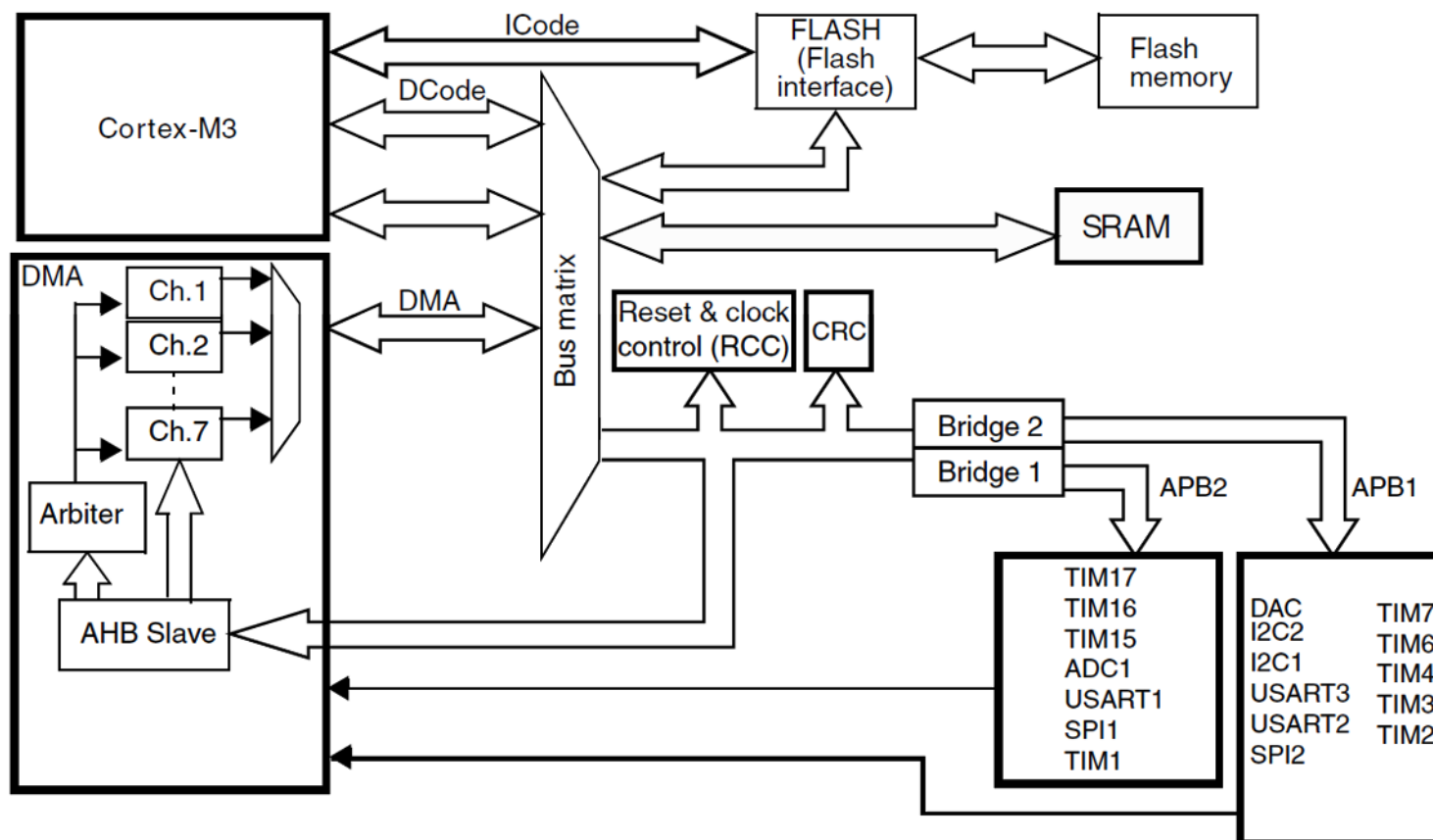


- Ciclo de instruções (*pipeline*) do μP com uma transferência em DMA...



➤ Diagrama de blocos do controlador **DMA** do STM32

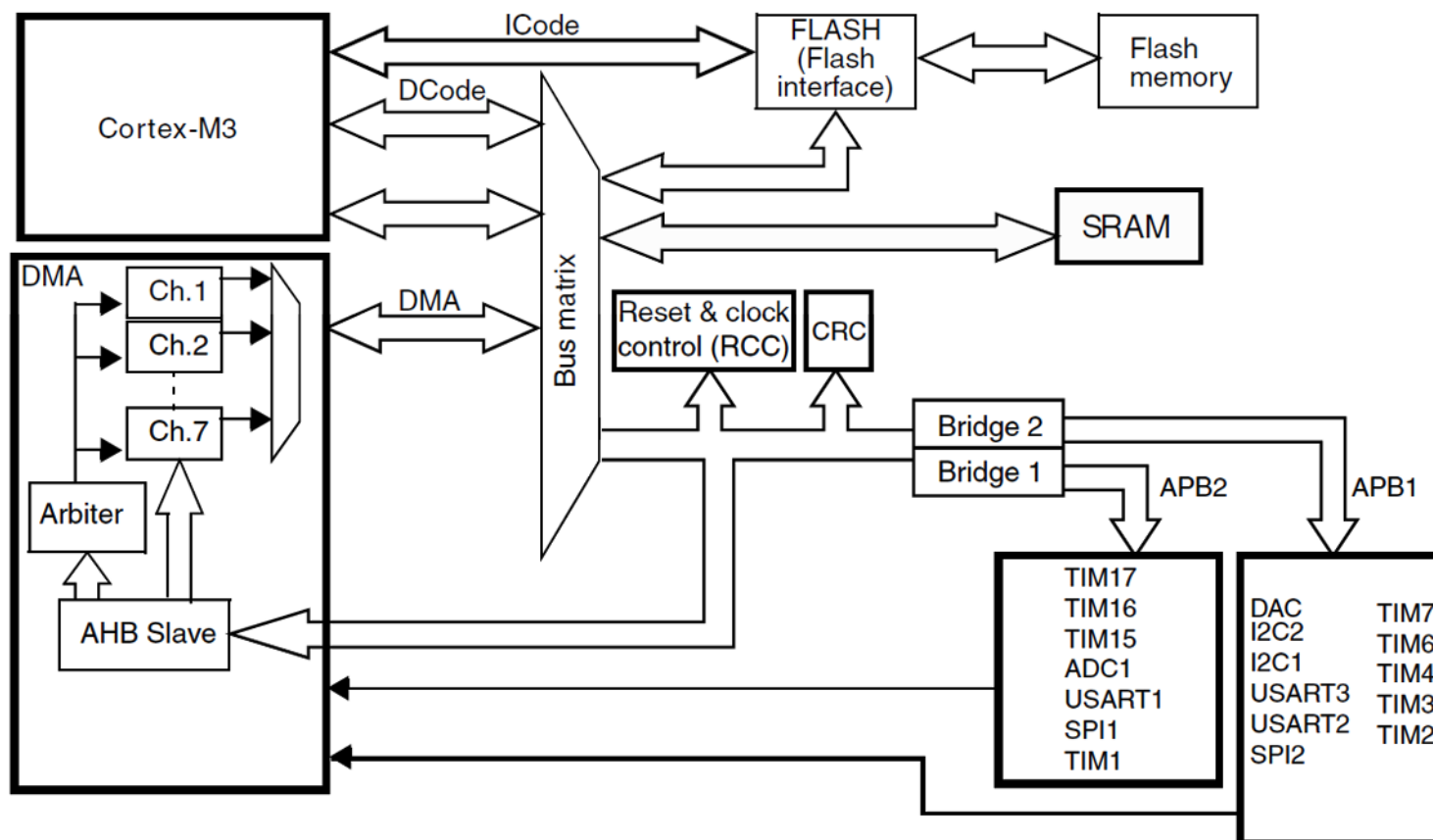
**Figure 20. DMA block diagram in low and medium- density Cat.1 and Cat.2 STM32F100xx devices**



A solicitação DMA pode interromper o acesso da CPU ao **bus system** por alguns ciclos de bus, quando a CPU e o DMA estão direcionados ao mesmo destino (memória ou periférico).

➤ Diagrama de blocos do controlador **DMA** do STM32

**Figure 20. DMA block diagram in low and medium- density Cat.1 and Cat.2 STM32F100xx devices**



A **bus matrix** implementa o agendamento round-robin, garantindo assim pelo menos 1/2 da largura de banda do barramento (tanto para memória/periférico quanto para a CPU).

- **CADA transferência DMA consiste em três operações:**
  - • **O carregamento** de dados do *registrador de dados* (ou um *endereço de memória*) através de um registro de endereço (endereço de memória). O endereço inicial usado para a 1ª transferência é o endereço base/memória programado no registro DMA\_CPARx ou DMA\_CMARx;
  - • **O armazenamento** de dados carregados em um *registro de dados* (ou localização da *memória*) endereçados por um *registro interno*. O endereço inicial usado para a primeira transferência é o endereço base / memória programado no registro DMA\_CPARx ou DMA\_CMARx;
  - • O **pós-decremento do registro DMA\_CNDTRx**, que contém o número de transações que ainda precisam ser executadas.
- **Canais DMA**
  - Cada canal pode lidar com a transferência DMA entre o registrador de um periférico (localizado num endereço fixo) e um endereço de memória. A quantidade de dados a serem transferidos (até 65535) é programável. O registro que contém a quantidade de itens de dados a serem transferidos é diminuído após cada transação.
  - Os ponteiros dos periféricos e da memória podem ser automaticamente pós-incrementados após cada transação dependendo dos bits PINC e MINC no registro DMA\_CCRx.





**Table 55. Summary of DMA2 requests for each channel**

Peripherals	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5
SPI3	SPI3_RX	SPI3_TX	-	-	
UART4	-	-	UART4_RX	-	UART4_TX
UART5	UART5_TX	-	-	UART5_RX	-
TIM5	TIM5_CH4 TIM5_TRIG	TIM5_CH3 TIM5_UP	-	TIM5_CH2	TIM5_CH1
TIM6/ DAC_Channel1	-	-	TIM6_UP/ DAC_Channel1	-	-
TIM7/ DAC_Channel2	-	-	-	TIM7_UP/ DAC_Channel2	-

*Obrigado...*

*Até a próxima aula.*