

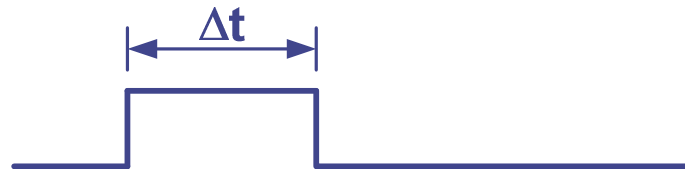
Aula 10

- *Timers*
 - Aplicações
 - Princípio de funcionamento
- *Timers* no PIC32
 - Estrutura e funcionamento
 - Geração de sinais PWM (*output compare module*)
- *Watchdog timer*

José Luís Azevedo, Arnaldo Oliveira, Tomás Silva, Bernardo Cunha

Introdução

- Um *timer* é um dispositivo periférico de suporte que permite, no essencial, a medição de tempo, partindo de uma referência temporal conhecida
- Alguns exemplos de aplicações típicas de *timers*:
 - geração de um evento com uma duração controlada;
exemplo: geração de um impulso com uma duração definida, Δt

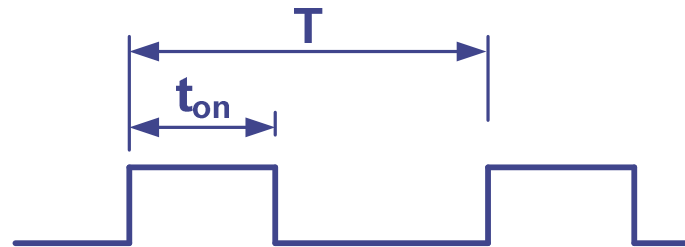


- geração de um evento periódico com período controlado;
exemplo: geração de um impulso com um período definido, T



Introdução

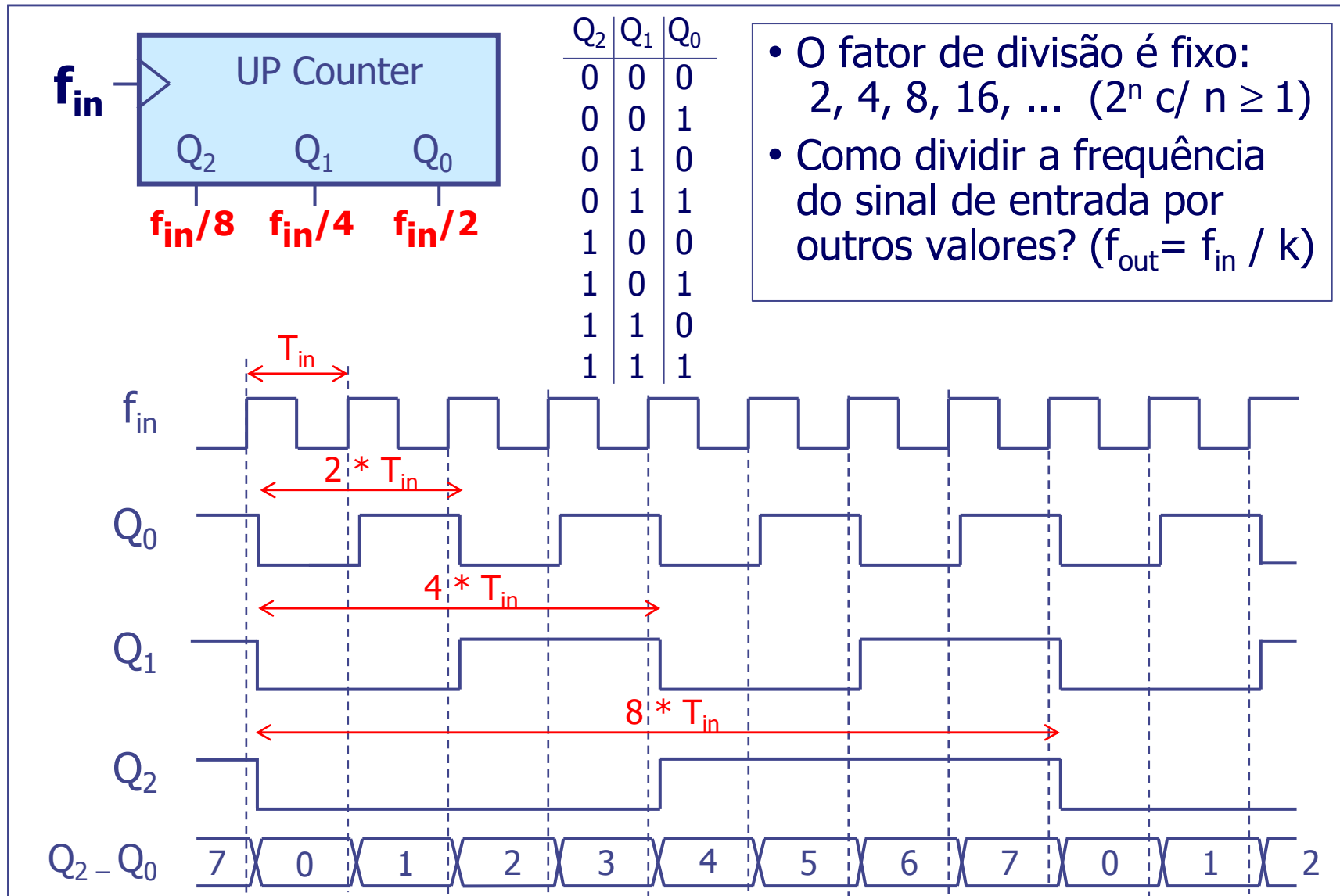
- Geração de um evento periódico com período e duração controlados. Exemplo: geração de um sinal periódico com um período de 10 ms e um "duty-cycle" de 40%:



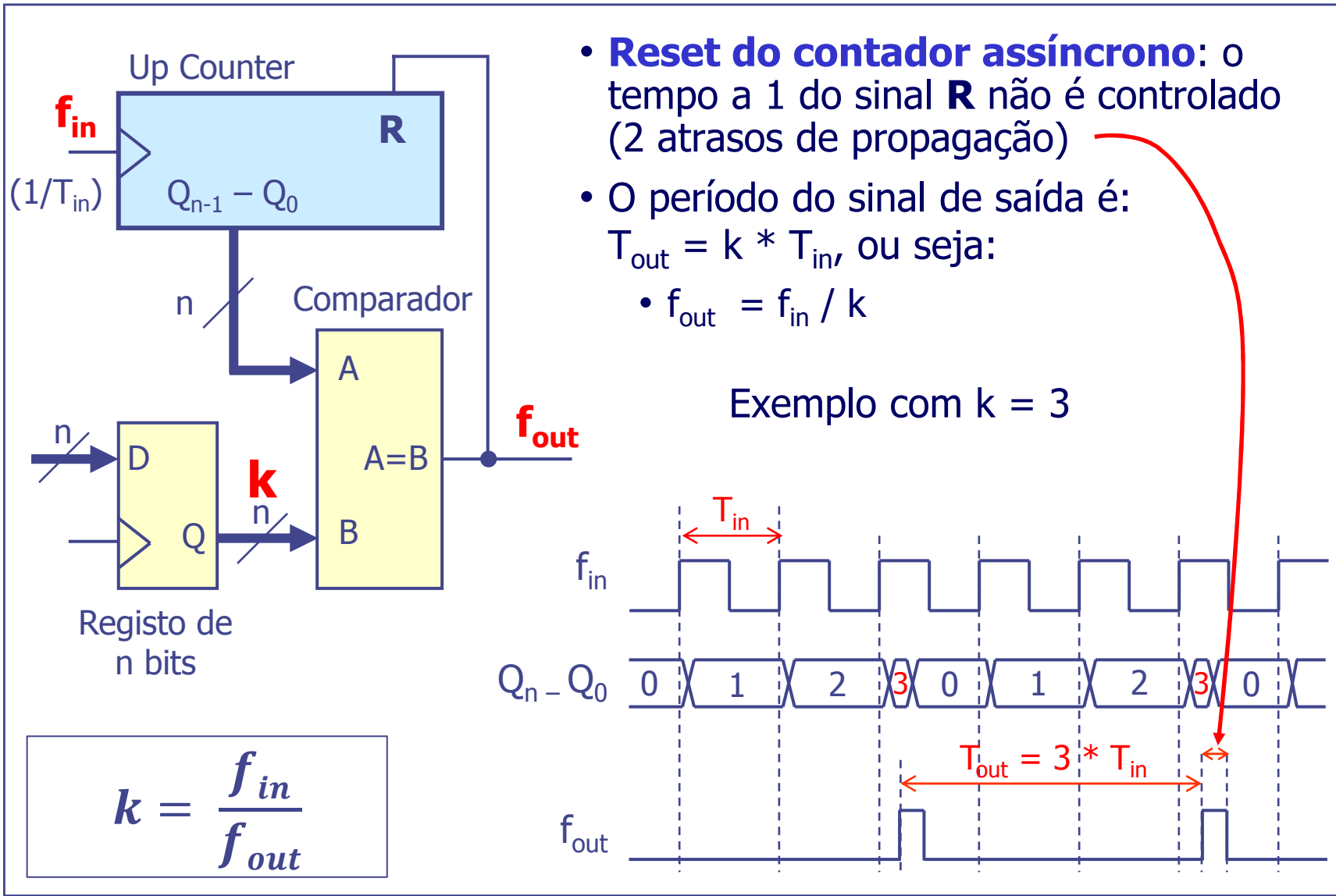
$$\text{Duty-cycle} = (t_{on} / T) * 100 \text{ [\%]}$$

- "ton" é o tempo durante o qual o sinal está no nível lógico 1, num período
- a possibilidade de alterar o valor de "ton" sem alterar o valor de T é útil em muitas situações e designa-se por **PWM** (**Pulse Width Modulation** – modulação por largura de pulso)
- O funcionamento dos *timers* baseia-se sempre na contagem de ciclos de um sinal de relógio com frequência conhecida

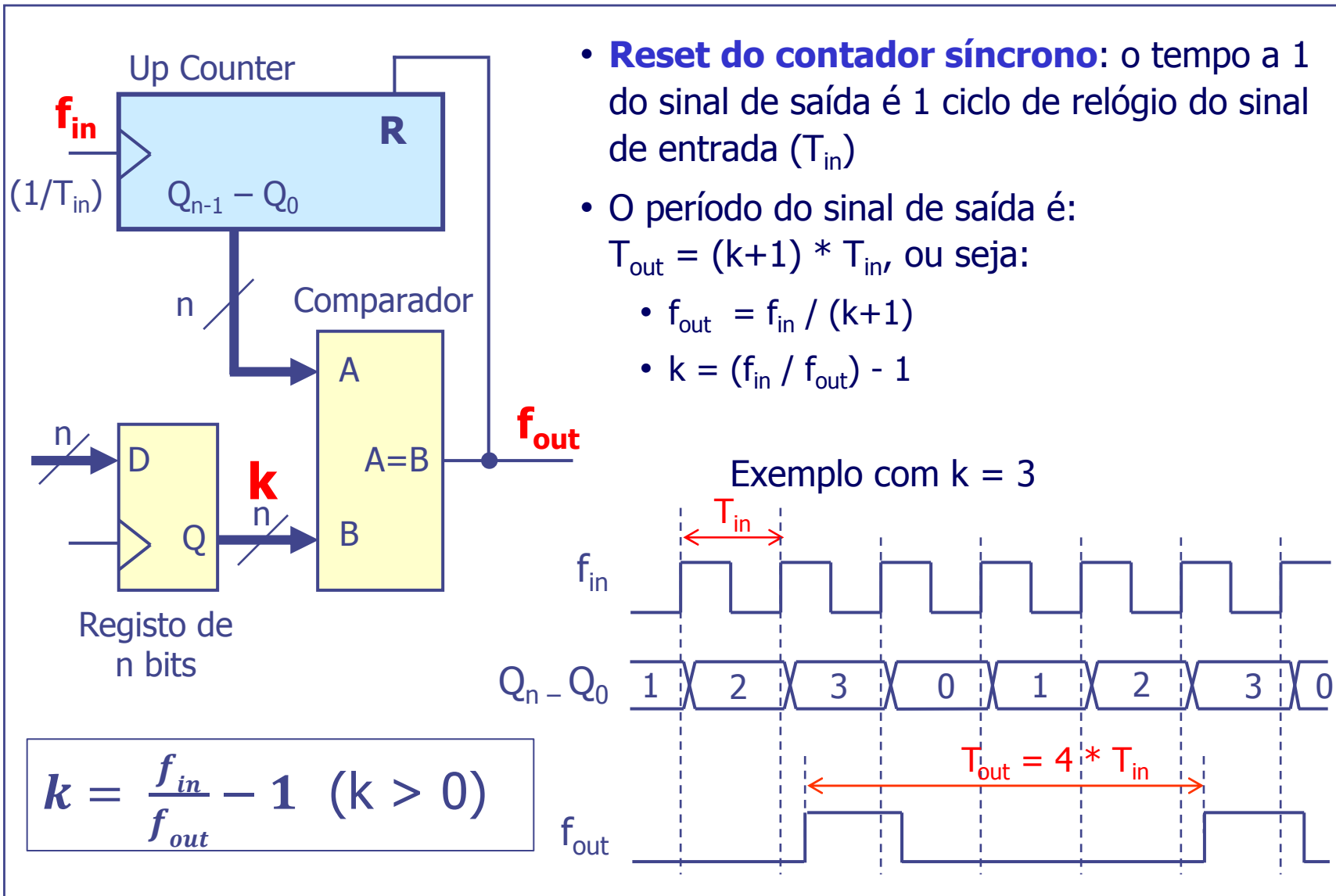
Divisão de frequência



Divisão de frequência (versão 1)



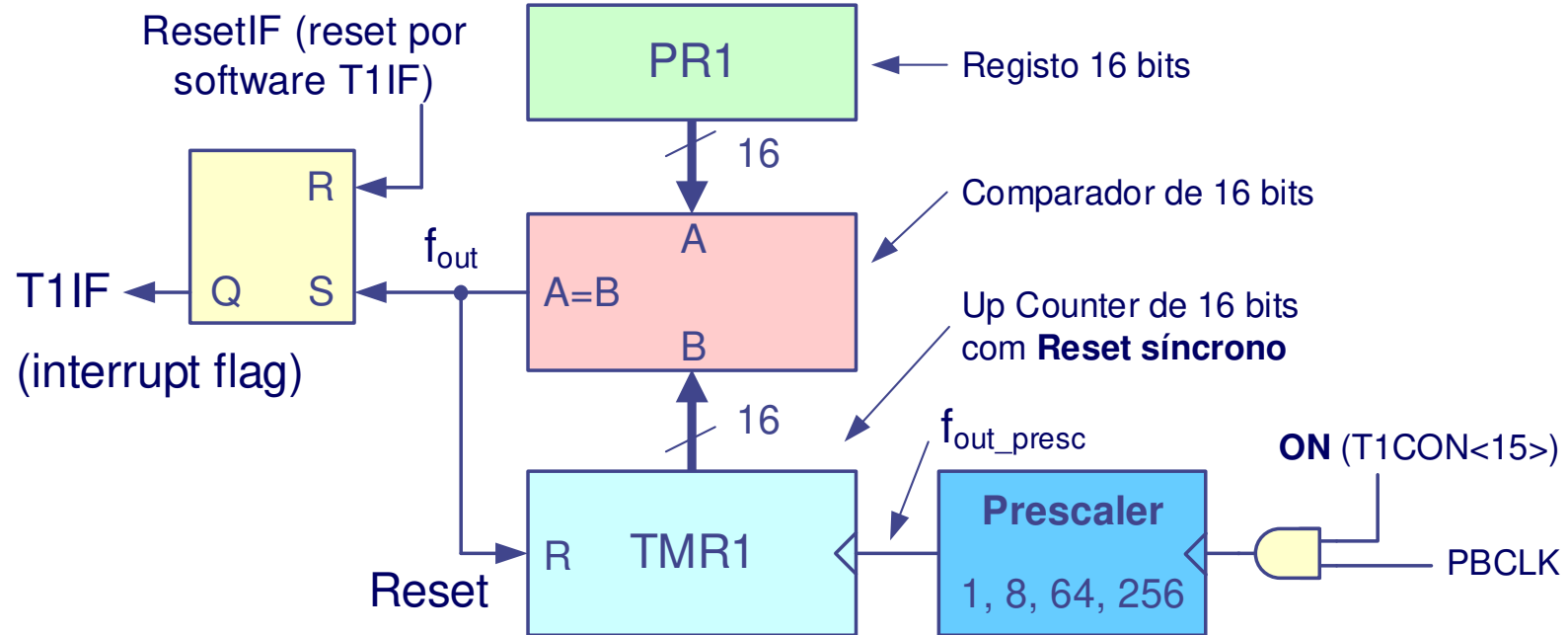
Divisão de frequência (versão 2)



Timers no PIC32

- A série PIC32MX7xx disponibiliza **5 timers de 16 bits**, designados por **T1, T2, T3, T4** e **T5**
- T2, T3, T4 e T5 têm a mesma estrutura e apresentam o mesmo modelo de programação. São designados pelo fabricante como **timers tipo B**
- T2 a T5 podem ser agrupados 2 a 2 para formar 2 *timers* de 32 bits (T2 e T3 e/ou T4 e T5)
- O T1 é designado como **timer tipo A**; tem uma estrutura semelhante aos restantes e pequenas diferenças no modelo de programação
- A frequência-base de entrada para os *timers* é dada pelo *Peripheral Bus Clock* (**PBCLK**). Na placa DETPIC32 a frequência de PBCLK é metade da frequência de CPU, i.e. **PBCLK = 20 MHz**
- Os *timers* do PIC32 não têm saída acessível no exterior. Podem ser usados para gerar interrupções (todos) ou como base de tempo para a geração de sinais com "duty-cycle" configurável (T2 e T3)

PIC32 – *timer* tipo A (estrutura simplificada)



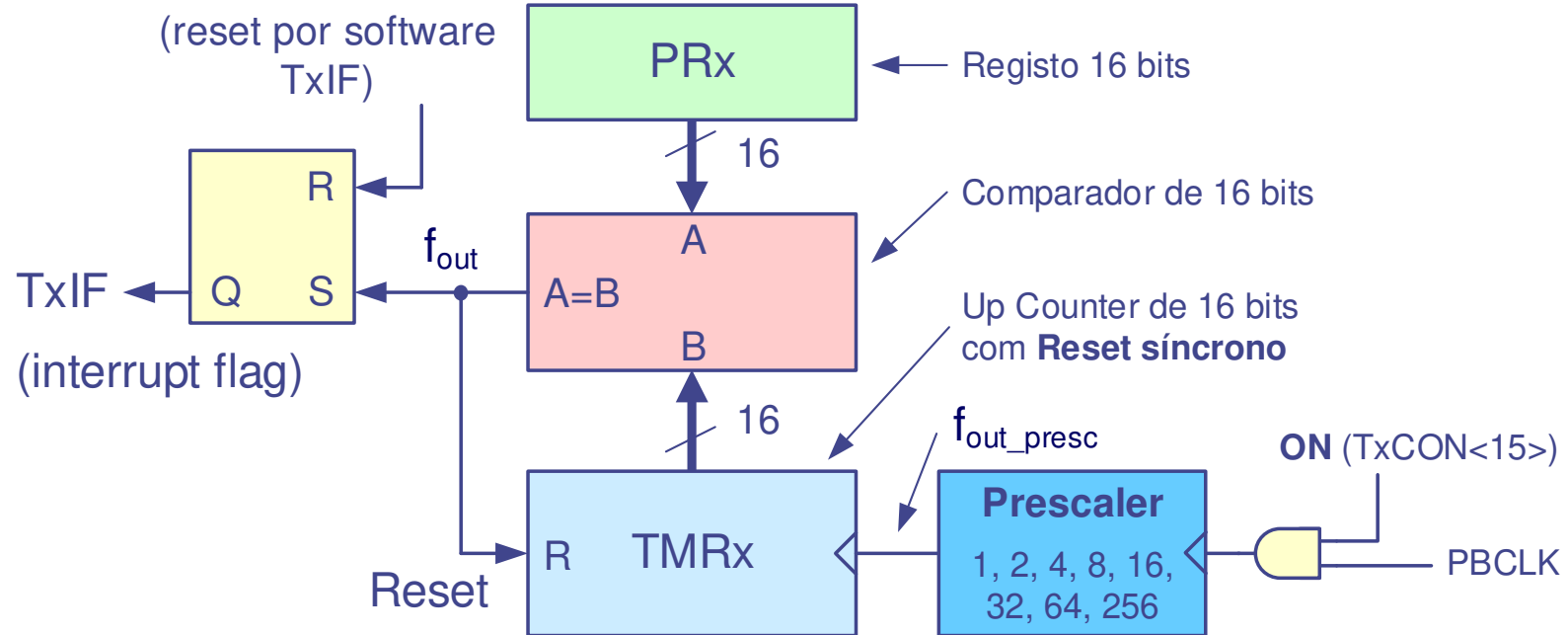
- Frequência à entrada do contador de 16 bits:

$$f_{out_presc} = \frac{PBCLK}{K_{prescaler}}$$

- Frequência à saída do *timer* (ativação de T1IF):

$$f_{out} = \frac{f_{out_presc}}{PR1 + 1}$$

PIC32 – *timers* tipo B (estrutura simplificada)



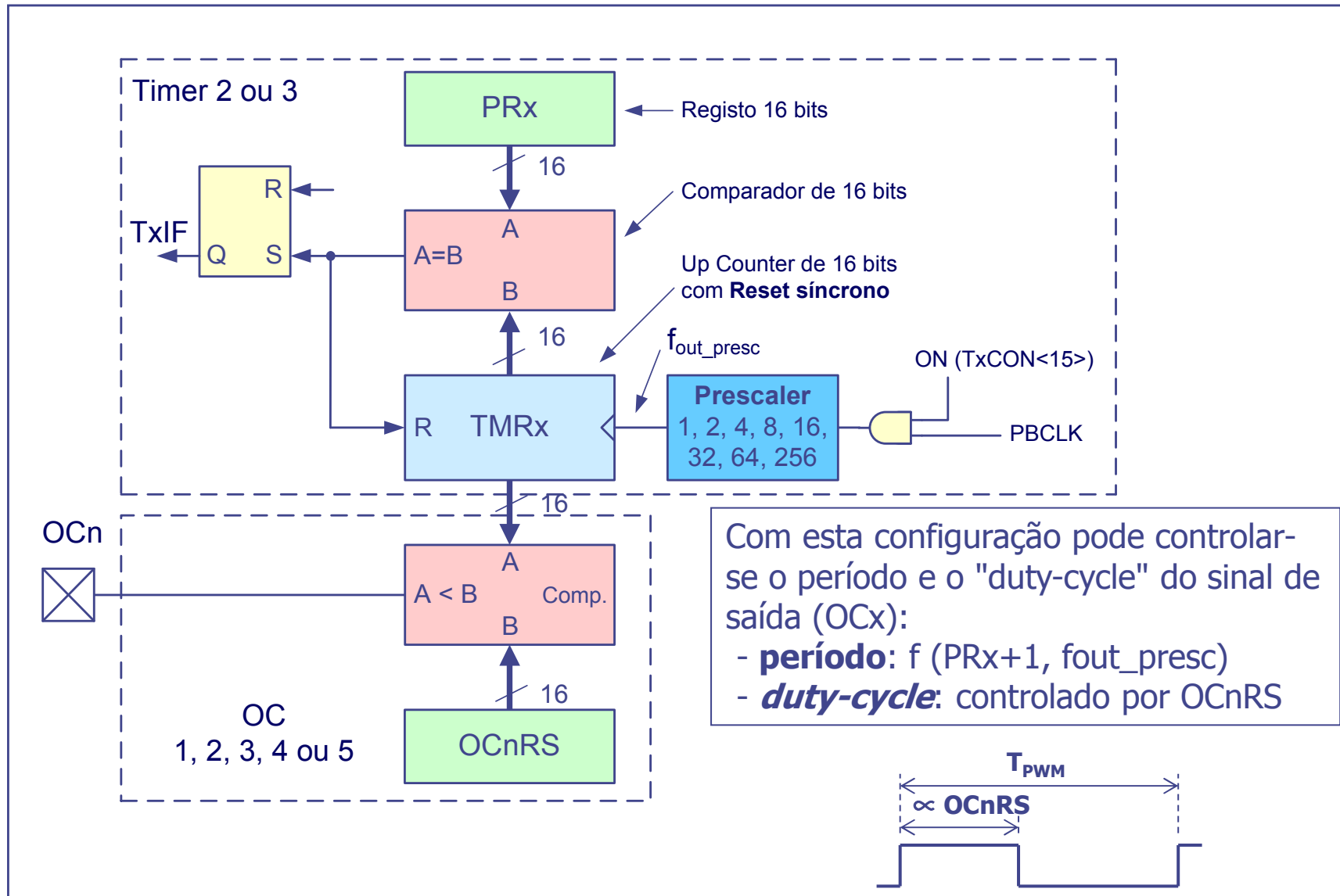
- Frequência à entrada do contador de 16 bits:

$$f_{out_presc} = \frac{PBCLK}{K_{prescaler}}$$

- Frequência à saída do *timer* (ativação de TxIF):

$$f_{out} = \frac{f_{out_presc}}{PRx + 1}$$

PIC32 – controlo de período e "duty-cycle"

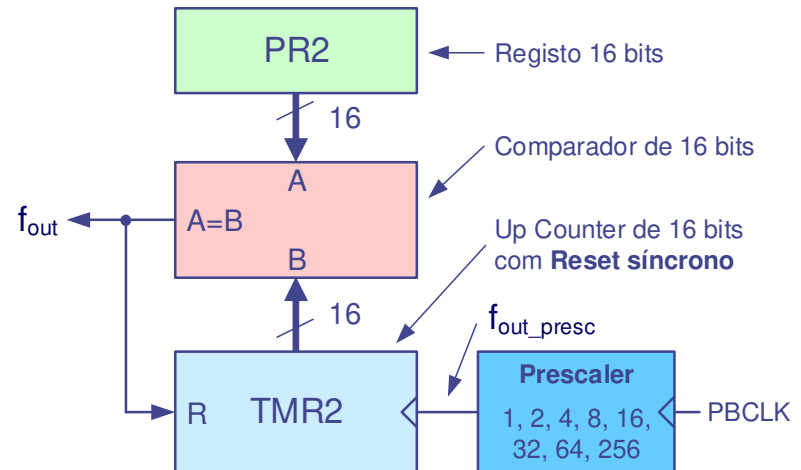


Exercício

Calcular as constantes para gerar um sinal PWM com uma frequência de 8 Hz e um *duty-cycle* de 20%, usando T2 como referência e OC1 como saída (PBCLK = 20 MHz)

$$f_{out} = \frac{f_{out_presc}}{PR2 + 1} = \frac{K_{prescaler}}{PR2 + 1}$$

$$K_{prescaler} = \frac{PBCLK}{(PR2 + 1) * f_{out}}$$



1. Cálculo da constante de divisão do *prescaler*

$$K_{prescaler} \geq \left\lceil \frac{PBCLK}{(65535 + 1) * 8} \right\rceil = 39$$

$$K_{prescaler} = 64$$

Valor máximo da constante PR2

Exercício (continuação)

2. Cálculo da constante de divisão do *timer* (PR2), com Kprescaler=64

$$f_{out_presc} = \frac{PBCLK}{K_{prescaler}} = \frac{20 * 10^6}{64} = 312500 \text{ Hz}$$

$$PR2 = \left(\frac{f_{out_presc}}{f_{out}} \right) - 1 = \frac{312500}{8} - 1 = 39062$$

3. Cálculo de OC1RS (*duty-cycle* de 20%):

$$OC1RS = \frac{(PR2 + 1) * dutyCycle}{100} = \frac{(39062 + 1) * 20}{100} = 7813$$

- 3.1 Alternativa ao cálculo de OC1RS:

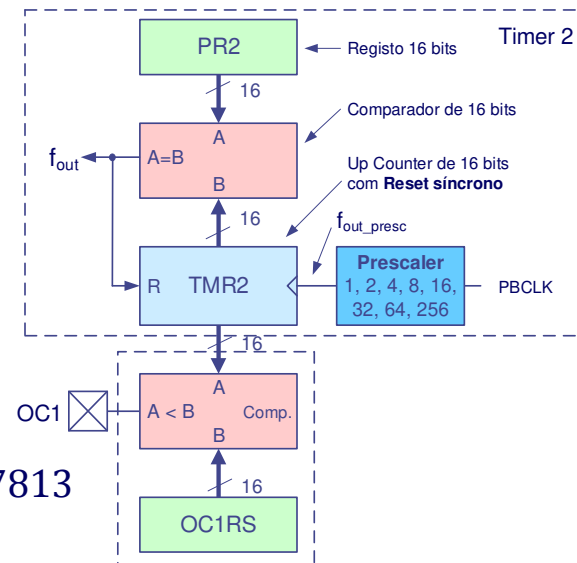
$$t_{ON} = 0.2 * T_{out} = \frac{0.2}{8} = 25 \text{ ms}$$

tempo a 1 (t_{ON}) do sinal de saída

$$T_{IN} = \frac{1}{f_{out_presc}} = \frac{1}{312500} = 3.2 \text{ us}$$

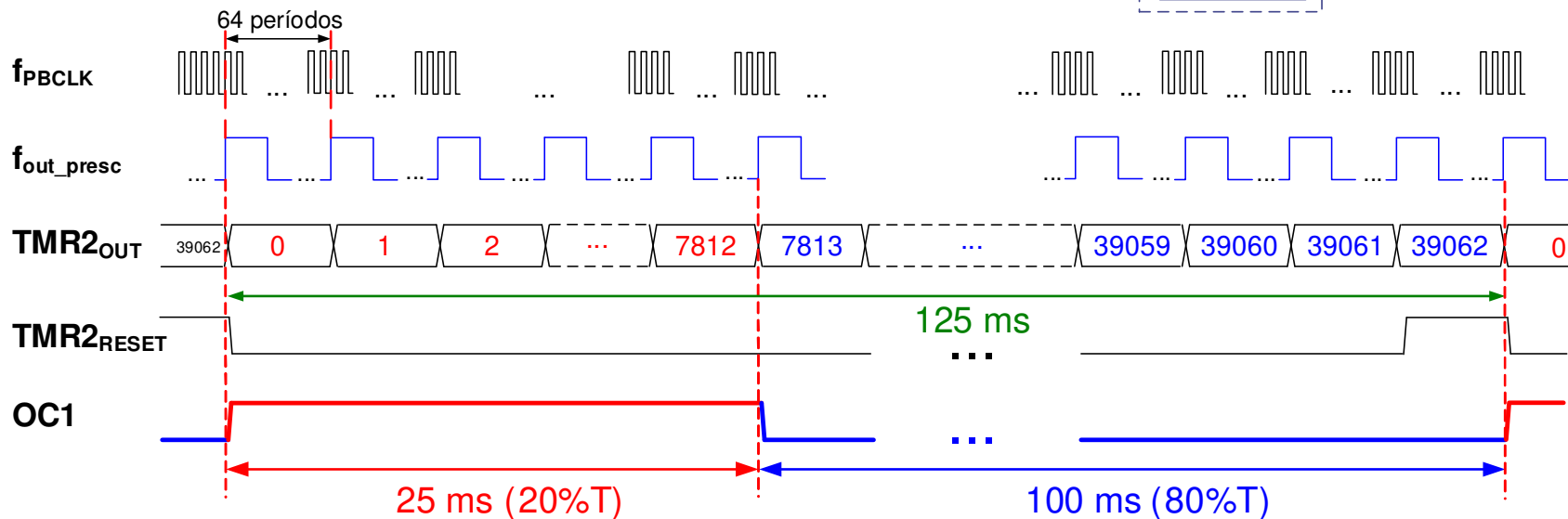
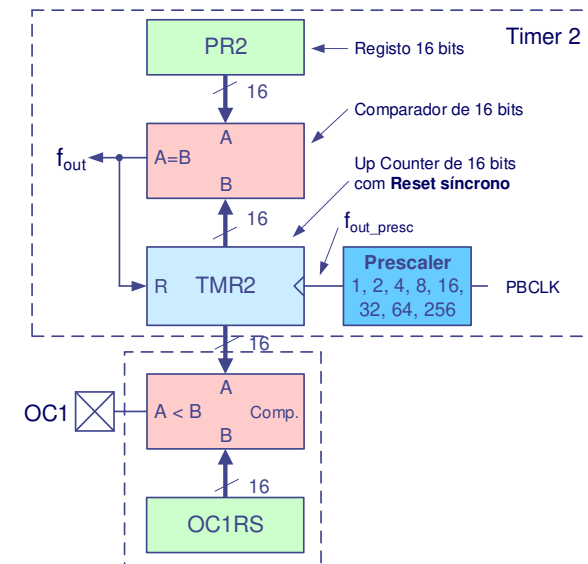
período do sinal à entrada do *timer*

$$OC1RS = \frac{t_{ON}}{T_{IN}} = \frac{25 * 10^{-3}}{3.2 * 10^{-6}} = 7813$$



Exercício (comportamento temporal)

- Settings:
 - Pre-scale factor = 64
 - PR2 value = 39062
 - OC1RS = 7813
- $f_{PBCLK} = 20 \text{ MHz}$
- $f_{out_presc} = 312.5 \text{ KHz}$
- $TMR2(OC1)_{period} = 125 \text{ ms}$ ($f=8\text{Hz}$)
- Duty-cycle = 20%

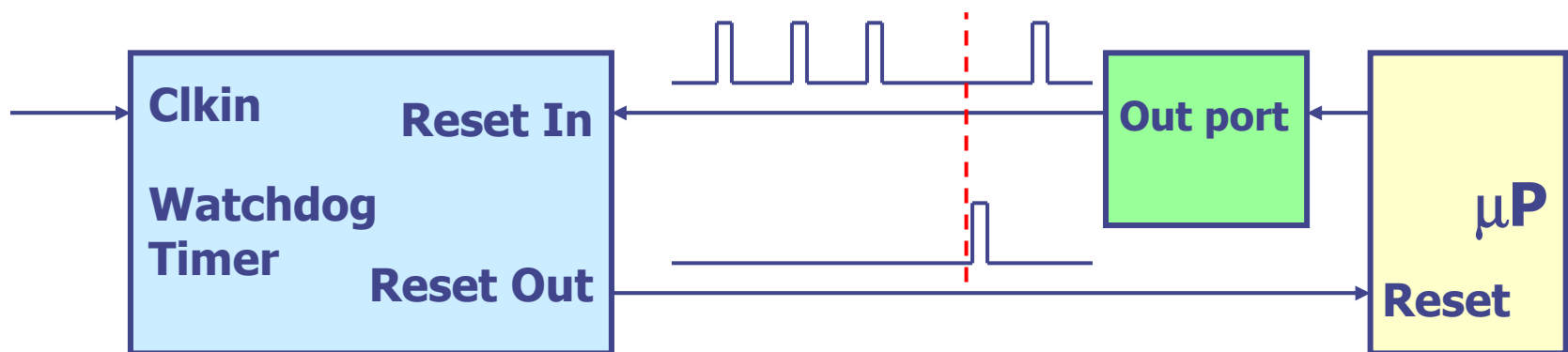


PIC32 – Resolução do sinal PWM

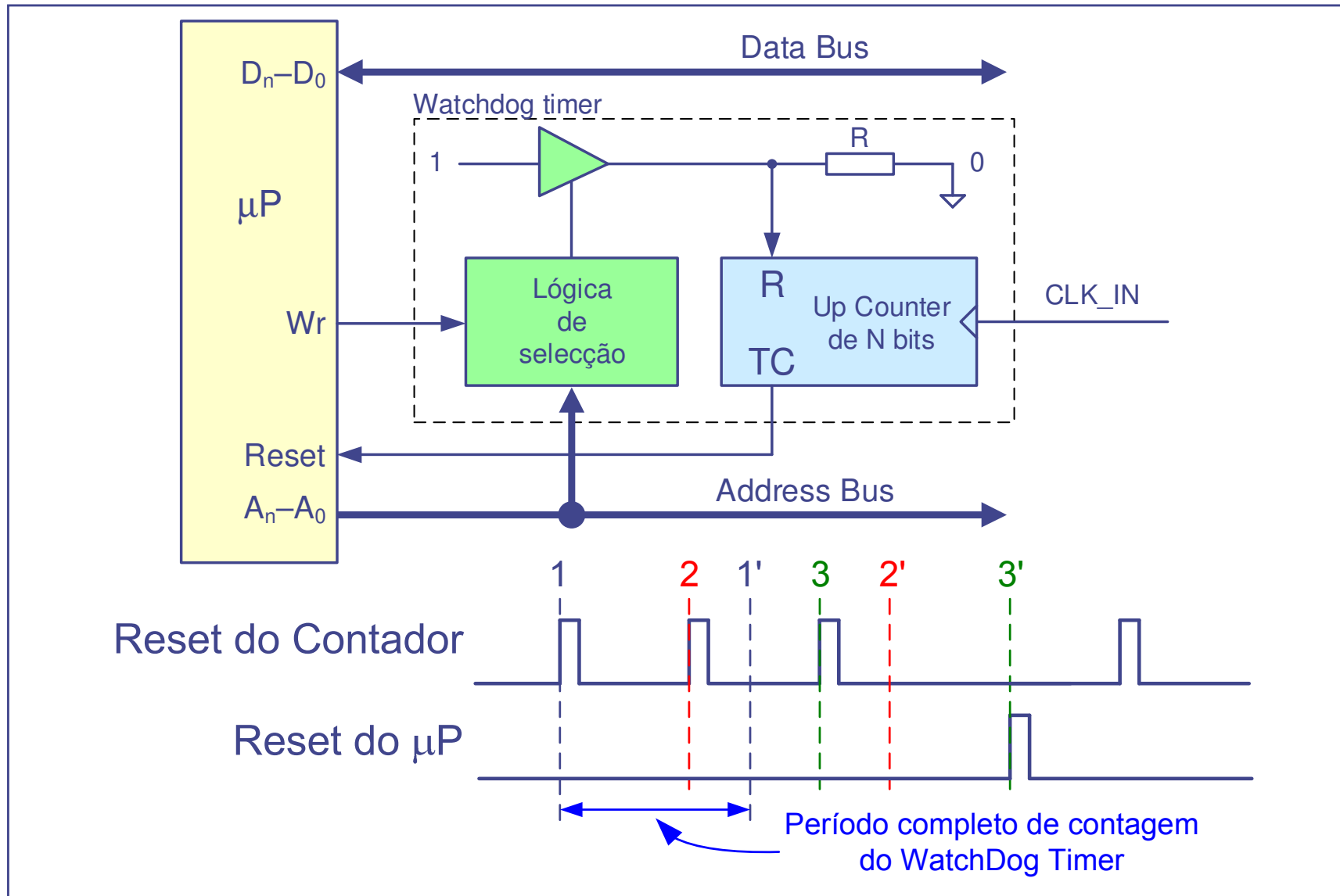
- A resolução de um sinal PWM dá uma medida do número de níveis com que se pode variar o *duty-cycle* do sinal
- Pode ser definida como:
 - **Resolução = $\log_2 (T_{\text{PWM}} / T_{\text{IN}})$**
 - em que T_{PWM} é o período do sinal PWM gerado e T_{IN} é o período do sinal à entrada do gerador de PWM
- Para o caso do PIC32:
 - **Resolução = $\log_2 (T_{\text{PWM}} / (T_{\text{PBCLK}} * \text{Prescaler}))$** , ou, mais simplesmente:
 - **Resolução = $\log_2 (\text{PRx} + 1)$**
- Exercícios:
 - determine o valor das constantes PRx e OCnRS para a geração de um sinal com uma frequência de 100 Hz e 25% de *duty-cycle*, supondo PBCLK = 20 MHz, maximizando a resolução do sinal PWM; determine o valor das constantes para um *duty-cycle* de 80%
 - determine a resolução do sinal PWM que obteve; determine a resolução do sinal de PWM do exemplo do slide anterior

Watchdog Timer (temporizador "cão de guarda")

- Sistemas baseados em microprocessador podem assegurar funções de controlo críticas que não podem falhar
- Como garantir que um crash do microprocessador não compromete o funcionamento global do sistema?
- Um *watchdog timer* tem como função monitorizar a operação do microprocessador e, em caso de falha, forçar o seu reinício
- Situação mais comum: se o processador não atuou a entrada de *Reset* do *watchdog timer* ao fim de um tempo pré-determinado o *watchdog timer* força o *Reset* do microprocessador



Watchdog Timer – implementação



Watchdog Timer – exemplo de utilização

- A aplicação no microcontrolador executa em ciclo infinito
- O *watchdog timer* é ativado quando o programa inicia. O *reset* da sua contagem é feito regularmente no corpo do ciclo (no exemplo, `clearWatchdogTimer()`)

```
void main(void)
{
    enableWatchdogTimer();
    (...)
    while(1)
    {
        (...)
        clearWatchdogTimer();
    }
}
```

- Caso haja uma falha no processador que implique a quebra de execução do ciclo, o *watchdog timer* deixa de ser reiniciado e, nessa situação, força um *reset* ao processador

Exercícios

1. Pretende-se gerar um sinal com uma frequência de 85 Hz. Usando o Timer T2 e supondo PBCLK = 50 MHz:
 - calcule o valor mínimo da constante de divisão a aplicar ao *prescaler* e indique qual o valor efetivo dessa constante
 - calcule o valor da constante PR2
2. Repita o exercício anterior, supondo que se está a usar o Timer T1
3. Pretende-se gerar um sinal com uma frequência de 100 Hz e 25% de "duty-cycle". Usando o módulo "output compare" OC5 e como base de tempo o Timer T3 e supondo ainda PBCLK = 40 MHz:
 - determine o valor efetivo da constante de *prescaler* que maximiza a resolução do sinal PWM
 - determine o valor das constantes PR3 e OC5RS
 - determine a resolução do sinal de PWM obtido

Anexos

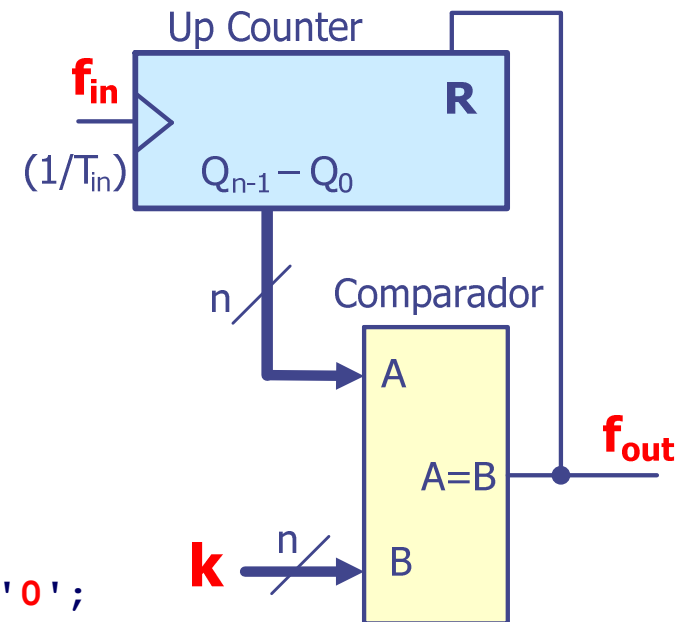
- Exemplos de modelação em VHDL de:
 - divisor de frequência genérico
 - *timer* tipo A do PIC32
 - gerador de PWM do PIC32

Exemplo de um divisor de frequência (VHDL)

```

entity FreqDivider is
    generic(N      : positive := 16);
    port( fin      : in  std_logic;
          k        : in  std_logic_vector(N-1 downto 0);
          fout     : out std_logic);
end FreqDivider;
architecture synchronous of FreqDivider is
    signal s_counter, s_k : natural range 0 to ((2 ** N)-1) := 0;
begin
    s_k <= to_integer(unsigned(k));
    process(fin)
    begin
        if(rising_edge(fin)) then
            if(s_counter = s_k) then
                s_counter <= 0;
            else
                s_counter <= s_counter + 1;
            end if;
        end if;
    end process;
    fout <= '1' when s_counter = s_k else '0';
end synchronous;

```



PIC32 – Modelação em VHDL do *timer* tipo A

```
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.numeric_std.all;
```

```
entity TimerPIC32_A is
```

```
    port ( PBCLK      : in std_logic;  
          T1On       : in std_logic;  
          ResetIF    : in std_logic;  
          Presc      : in std_logic_vector(1 downto 0);  
          PR1        : in std_logic_vector(15 downto 0);  
          T1IF       : out std_logic);
```

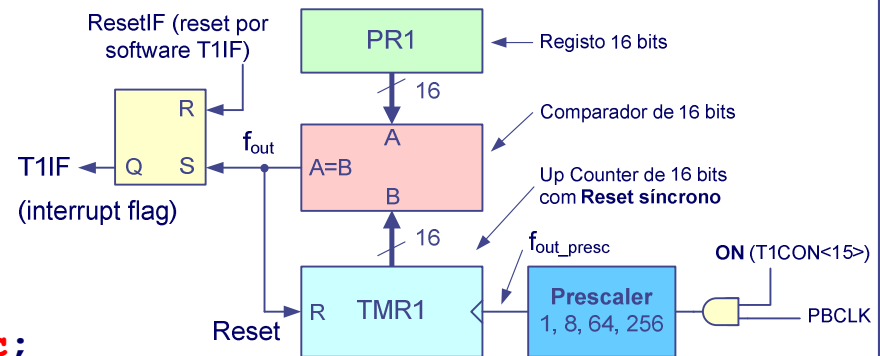
```
end TimerPIC32_A;
```

```
architecture synchronous of TimerPIC32_A is
```

```
    signal s_counter, s_pr1 : natural range 0 to (2**16-1) := 0;  
    signal s_precounter : unsigned(7 downto 0);  
    signal s_foutPresc : std_logic;
```

```
begin
```

```
-- (continua)
```

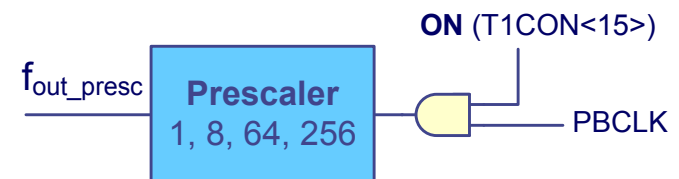


PIC32 – Modelação em VHDL do *timer* tipo A

```
-- Prescaler (divide PBCLK frequency by 1, 8, 64 or 256)
process(PBCLK, T1On, s_precounter, s_kprescale)
begin
    if(rising_edge(PBCLK)) then
        if(T1On = '1') then
            s_precounter <= s_precounter + 1;
        end if;
    end if;
end process;

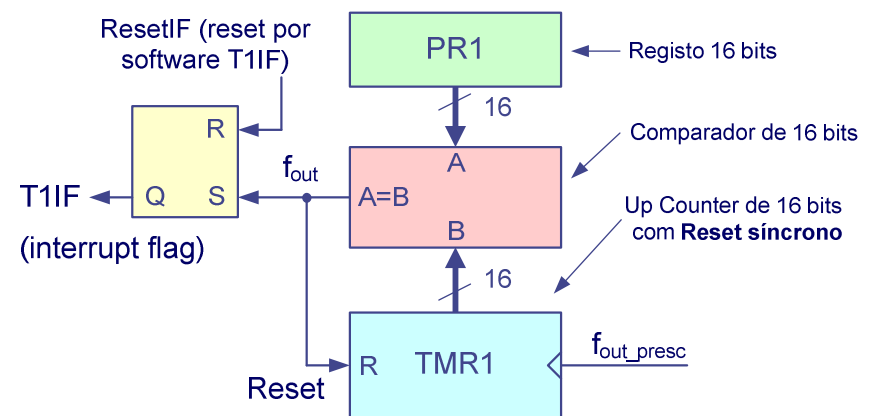
s_foutPresc <= PBCLK          when Presc = "00" else -- Div 1
              s_precounter(2) when Presc = "01" else -- Div 8
              s_precounter(5) when Presc = "10" else -- Div 64
              s_precounter(7); -- Div 256
```

-- (continua)



PIC32 – Modelação em VHDL do *timer* tipo A

```
-- Timer (input clock is the signal produced by the prescaler)
s_pr1 <= to_integer(unsigned(PR1));
process(s_foutPresc, ResetIF)
begin
    if(rising_edge(s_foutPresc)) then
        if(s_counter = s_pr1) then
            T1IF <= '1'; s_counter <= 0;
        else
            s_counter <= s_counter + 1;
        end if;
    end if;
end if;
if(ResetIF = '1') then
    T1IF <= '0';
end if;
end process;
end synchronous;
```



PIC32 – Modelação em VHDL do gerador de PWM

```
entity FreqDividerDC is
    port( foutPresc : in std_logic;
          PRx       : in std_logic_vector(15 downto 0);
          OCnRS     : in std_logic_vector(15 downto 0);
          OCn       : out std_logic);
end FreqDividerDC;
architecture synchronous of FreqDividerDC is
    signal s_counter : natural range 0 to (2**16-1) := 0;
    signal s_prx, s_ocnrs : natural range 0 to (2**16-1);
begin
    s_ocnrs <= to_integer(unsigned(OCnRS));
    s_prx <= to_integer(unsigned(PRx));
    process(foutPresc)
    begin
        if(rising_edge(foutPresc)) then
            if(s_counter = s_prx) then
                s_counter <= 0;
            else
                s_counter <= s_counter + 1;
            end if;
        end if;
    end process;
    OCn <= '1' when s_counter < s_ocnrs) else '0';
end synchronous;
```

