

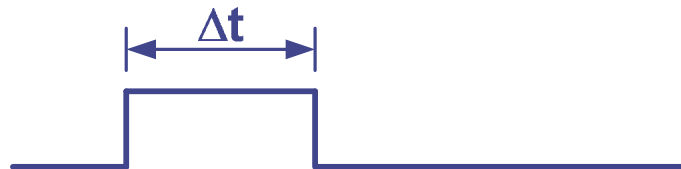
Aula 10

- *Timers*
 - Aplicações
 - Princípio de funcionamento
 - Controlo da frequência e "duty-cycle" do sinal de saída
- *Timers* no PIC32
 - Estrutura e funcionamento
 - Geração de sinais PWM (*output compare module*)
- *Watchdog timer*

José Luís Azevedo, Arnaldo Oliveira, Tomás Oliveira e Silva, Nuno Lau

Introdução

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

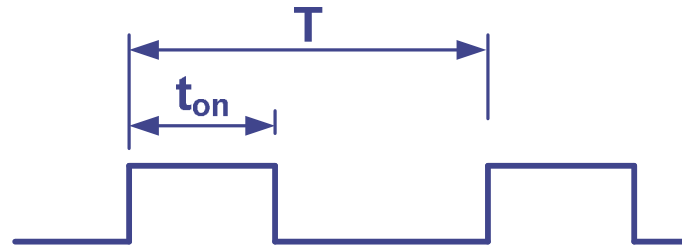


- geração de um evento periódico com período controlado.
Exemplo: geração de um pulso com um período definido, T



Introdução

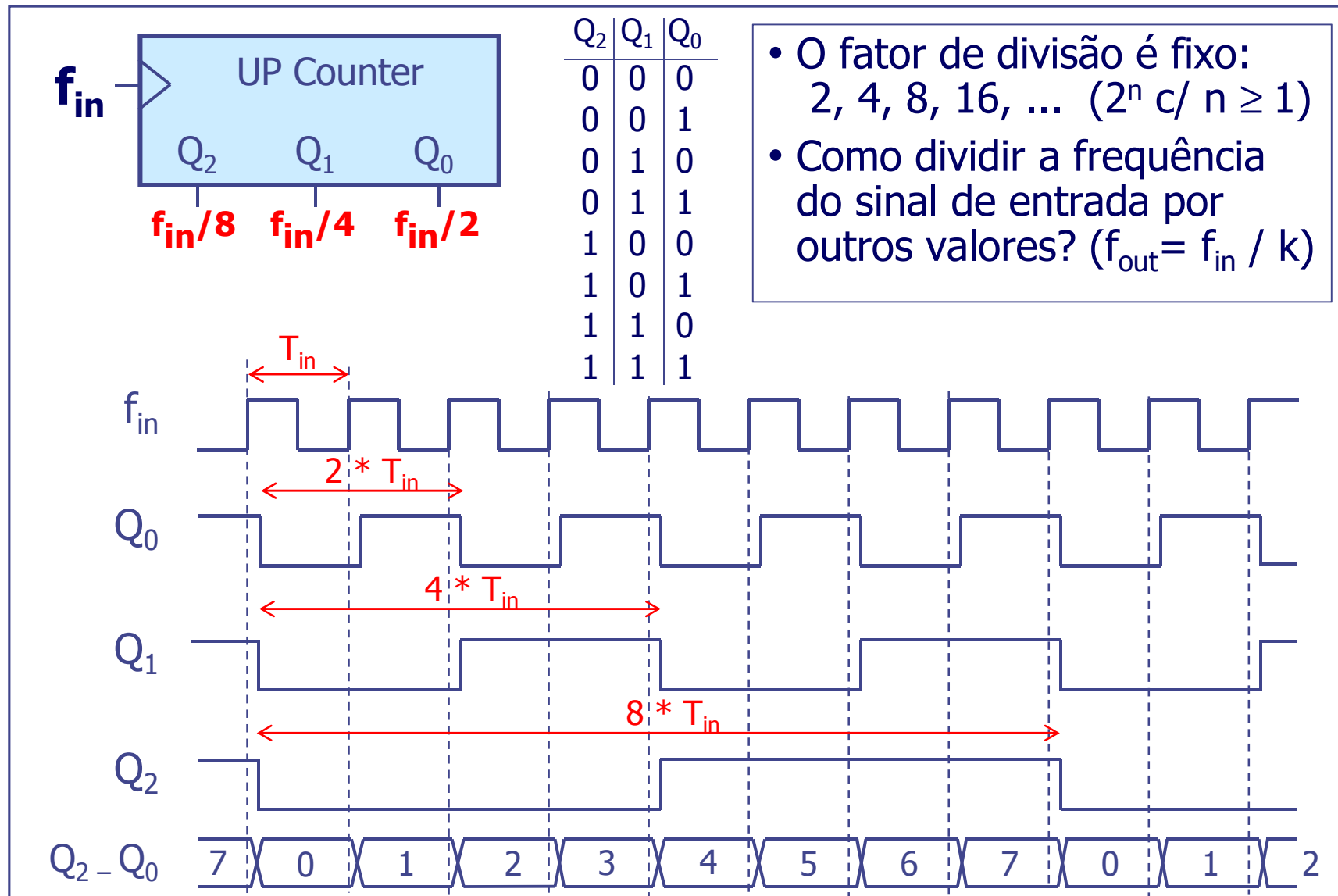
- Geração de um evento periódico com período e duração controlados. Exemplo: geração de um sinal periódico com um período de 10 ms e um "duty-cycle" de 40%:



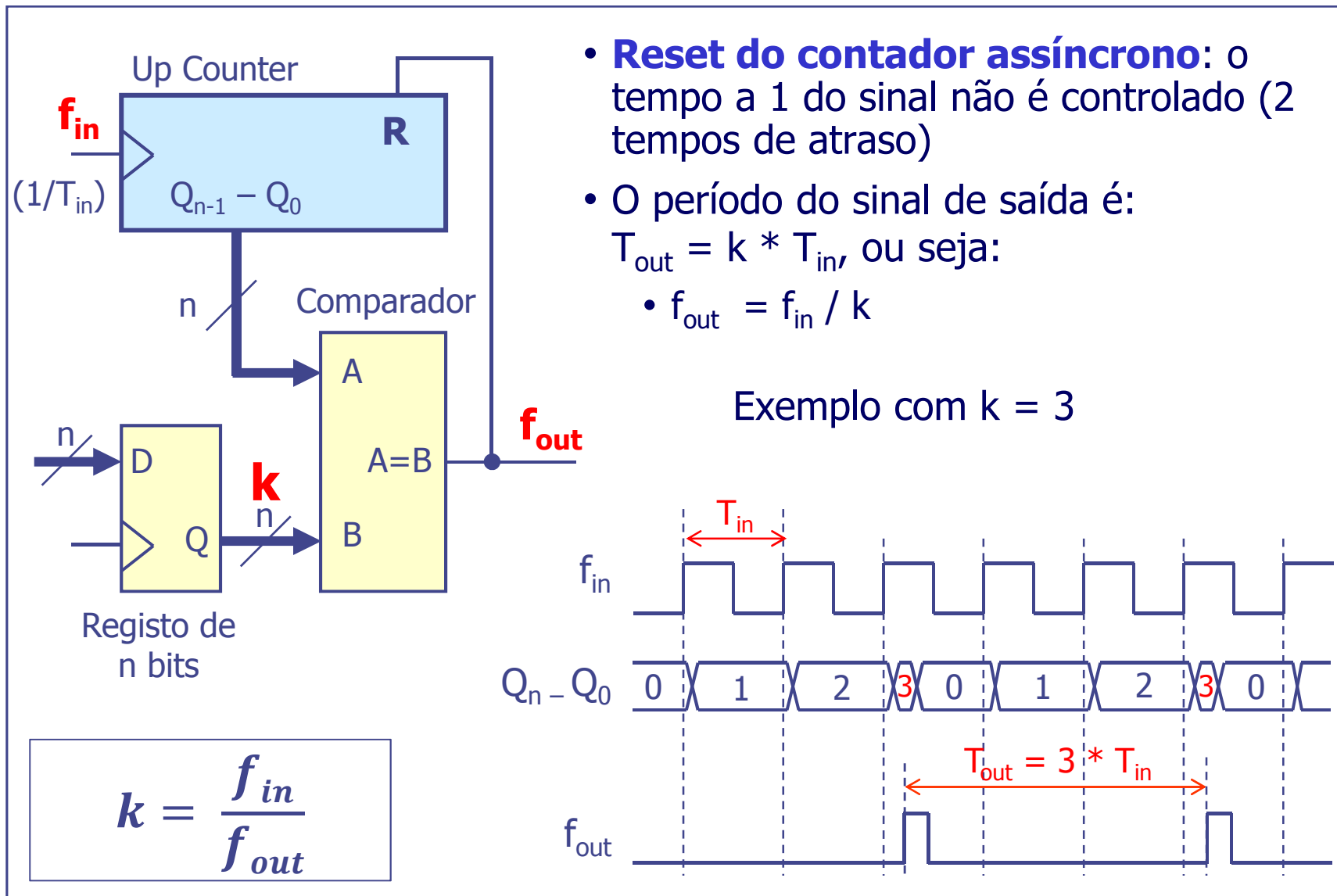
$$\text{Duty-cycle} = (t_{\text{on}} / T) * 100 \text{ [\%]}$$

- "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

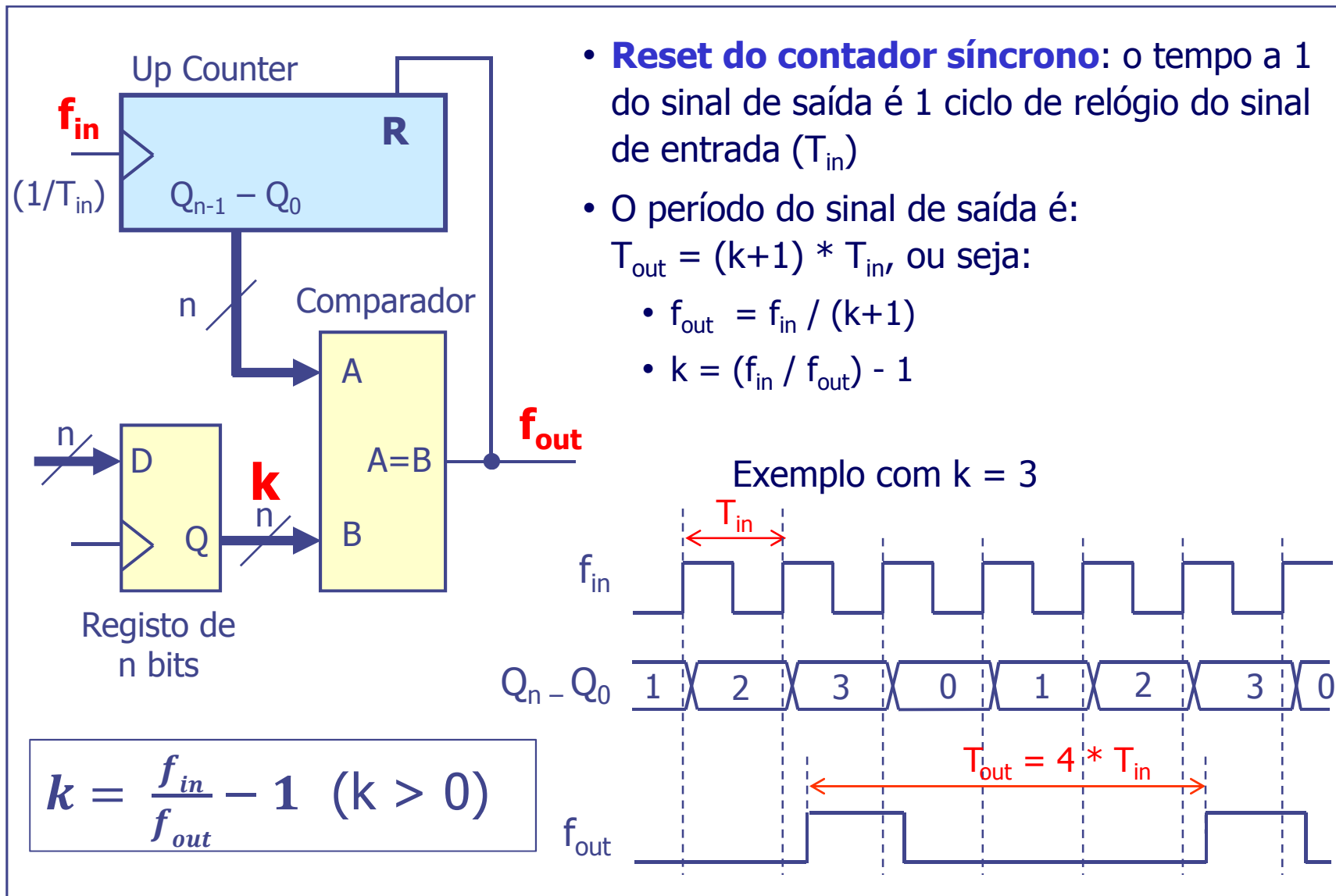
Divisão de frequência



Divisão de frequência (versão 1)

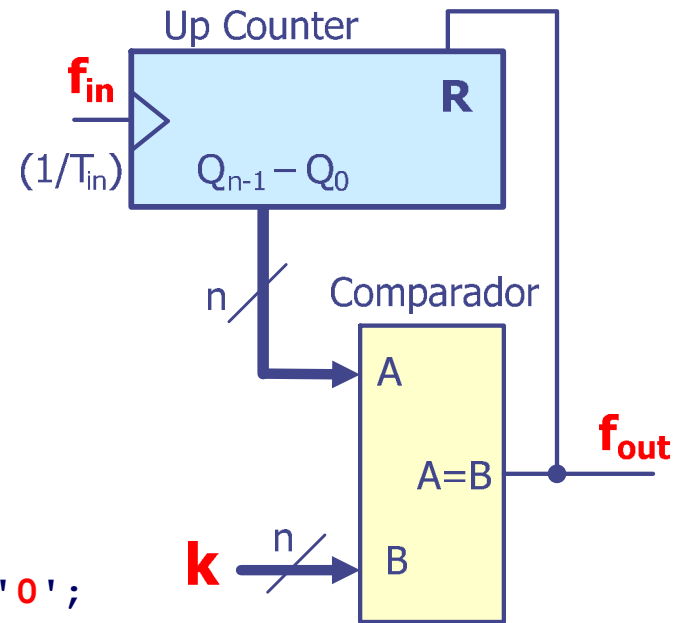


Divisão de frequência (versão 2)



Exemplo de um divisor de frequência (VHDL)

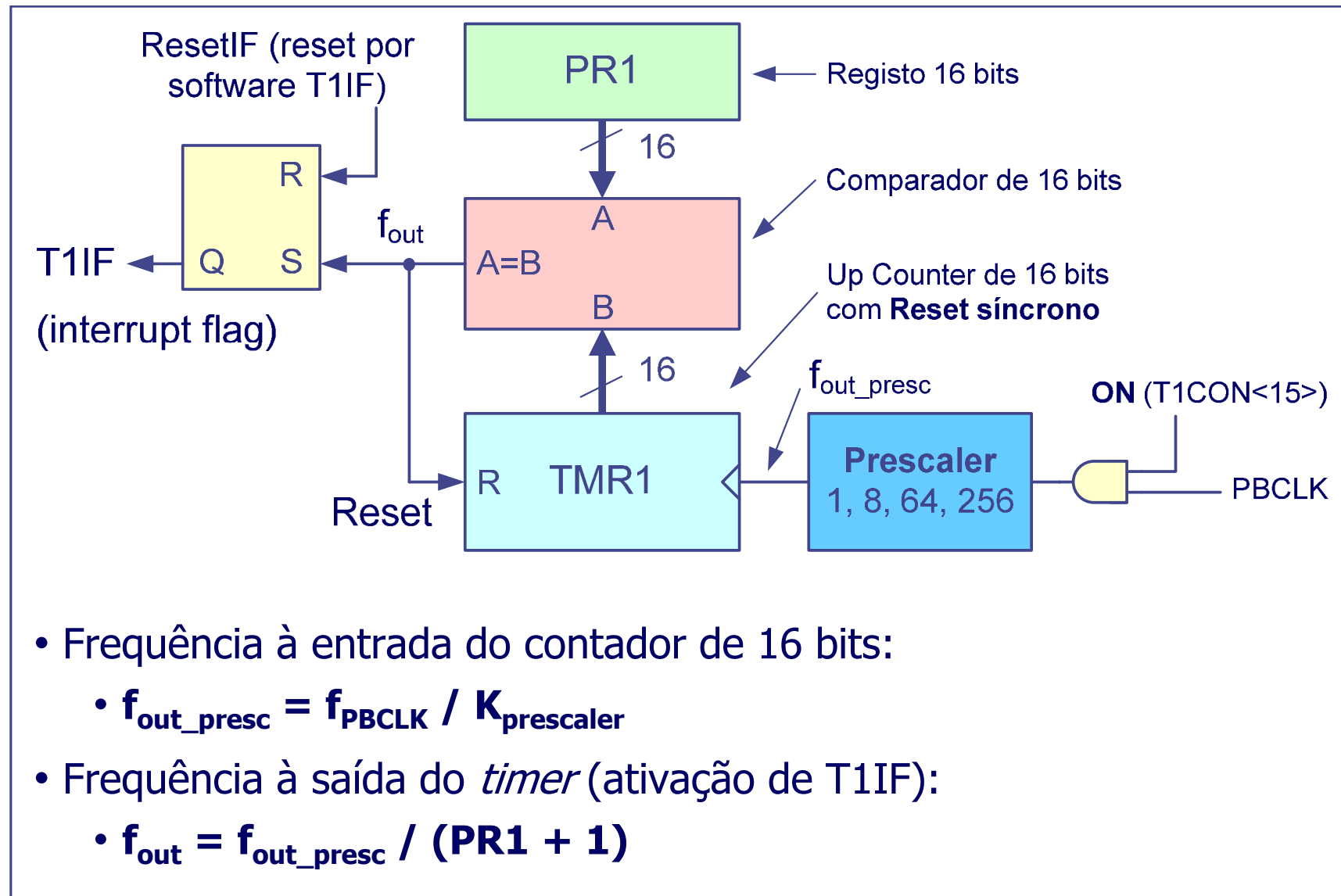
```
entity FreqDivider is
    generic(N      : positive := 16);
    port( fin      : in  std_logic;
          k        : in  std_logic_vector(N-1 downto 0);
          fout     : out std_logic);
end FreqDivider;
architecture synchronous of FreqDivider is
    signal s_counter, s_k : natural range 0 to ((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;
```



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
- 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 f_{PBCLK} é metade da frequência de CPU, i.e. **$f_{\text{PBCLK}} = 20 \text{ MHz}$**
- Os *timers* do PIC32 não têm saída acessível no exterior. Podem ser usados para gerar interrupções ou como base de tempo para a geração de sinais com "duty-cycle" configurável

PIC32 – *timer* tipo A (estrutura simplificada)



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);
```

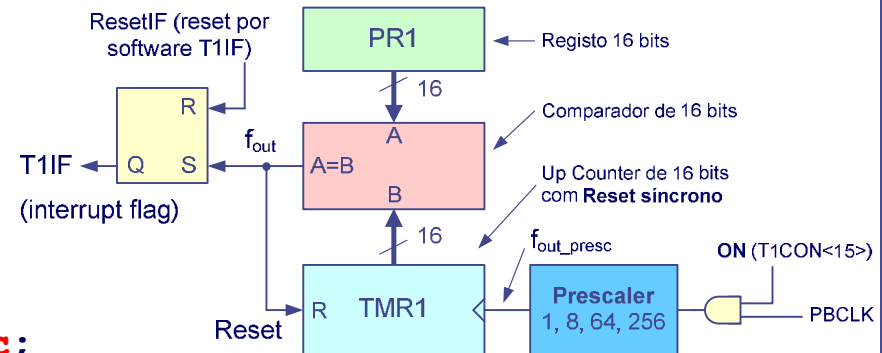
```
end TimerPIC32_A;
```

```
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)
```

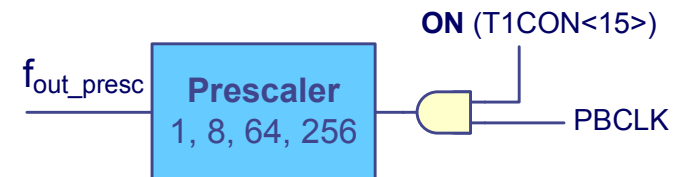


PIC32 – Modelação em VHDL do *timer* tipo A

```
-- 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;

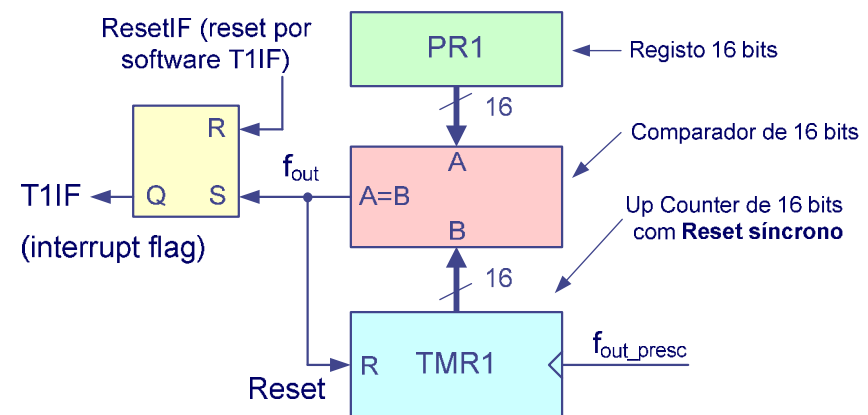
s_foutPresc <= PBCLK          when Presc = "00" else -- Div 1
                s_precounter(2) when Presc = "01" else -- Div 8
                s_precounter(5) when Presc = "10" else -- Div 64
                s_precounter(7);                       -- Div 256
```

-- (continua)

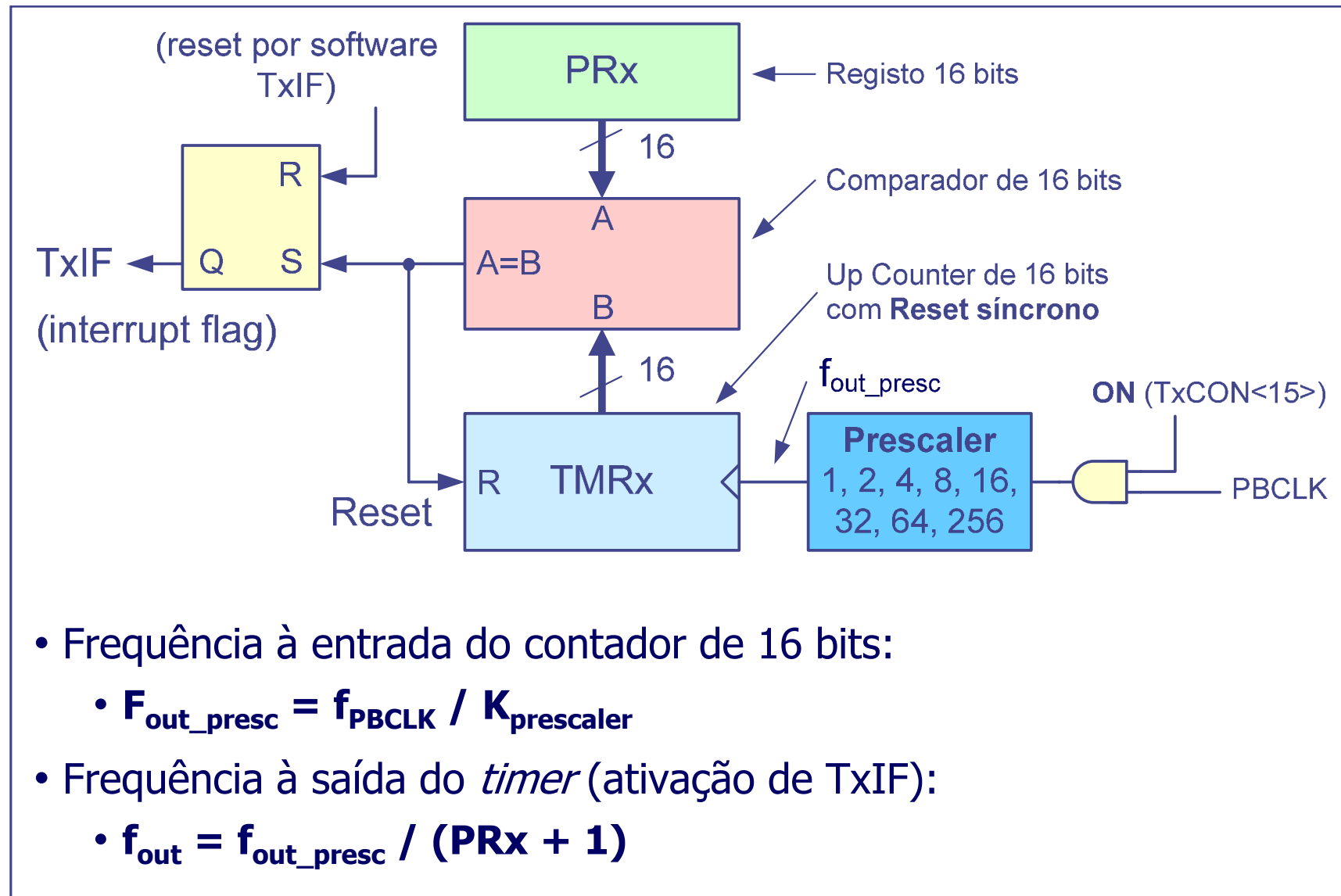


PIC32 – Modelação em VHDL do *timer* tipo A

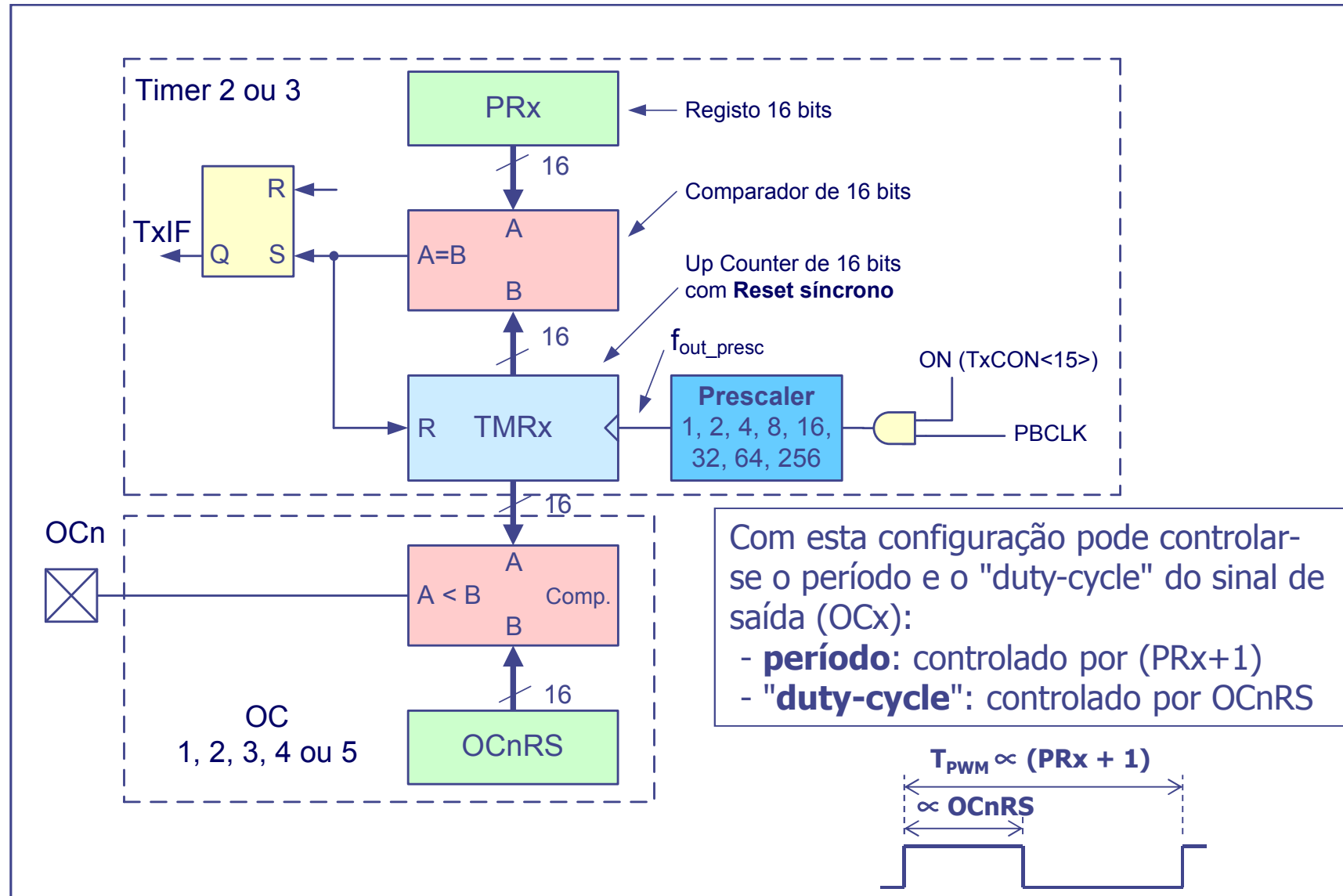
```
-- Timer (input clock is the signal produced by the prescaler)
s_pr1 <= to_integer(unsigned(PR1));
process(s_foutPresc, ResetIF)
begin
    if(rising_edge(s_foutPresc)) then
        if(s_counter = s_pr1) then
            T1IF <= '1'; s_counter <= 0;
        else
            s_counter <= s_counter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



PIC32 – *timers* tipo B (estrutura simplificada)



PIC32 – controlo de período e "duty-cycle"

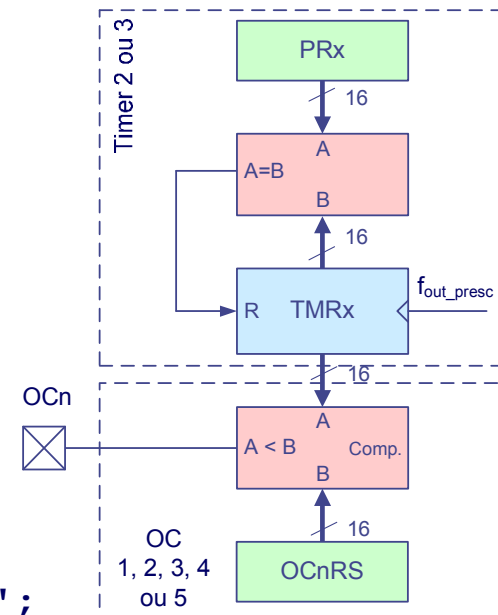


PIC32 – Modelação em VHDL do gerador de PWM

```

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;

```



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)

$$\begin{aligned}f_{\text{out}} &= f_{\text{out_presc}} / (\text{PR2} + 1) \\&= (f_{\text{PBCLK}} / K_{\text{prescaler}}) / (\text{PR2} + 1) \\&= f_{\text{PBCLK}} / (K_{\text{prescaler}} * (\text{PR2} + 1)) \\K_{\text{prescaler}} &= f_{\text{PBCLK}} / ((\text{PR2} + 1) * f_{\text{out}})\end{aligned}$$

1. Cálculo da constante de divisão do *prescaler*

$$K_{\text{prescaler}} \geq \lceil f_{\text{PBCLK}} / ((65535+1) * 8) \rceil = 39$$

$$K_{\text{prescaler}} = 64$$

Valor máximo da constante PR2

Exercício (continuação)

2. Cálculo da constante de divisão do *timer* (PR2)

Com prescaler = 64, $f_{\text{OUT_PRESC}} = 20000000 / 64 = 312500 \text{ Hz}$

$$\text{PR2} = (f_{\text{OUT_RESC}} / f_{\text{OUT}}) - 1 = (312500 / 8) - 1 = 39062$$

3. Cálculo de OC1RS:

$$\text{OC1RS} = ((\text{PR2} + 1) * \text{duty_cycle}) / 100 = (39062 + 1) * 0.20$$

$$\text{OC1RS} = 7813$$

Alternativa ao cálculo de OC1RS:

Tempo a 1 (t_{ON}) do sinal de saída, $t_{\text{ON}} = 0.20 * (1 / 8) = 25 \text{ ms}$

Período do sinal de entrada do *timer*, $T_{\text{IN}} = 1 / 312500 = 3.2 \text{ } \mu\text{s}$

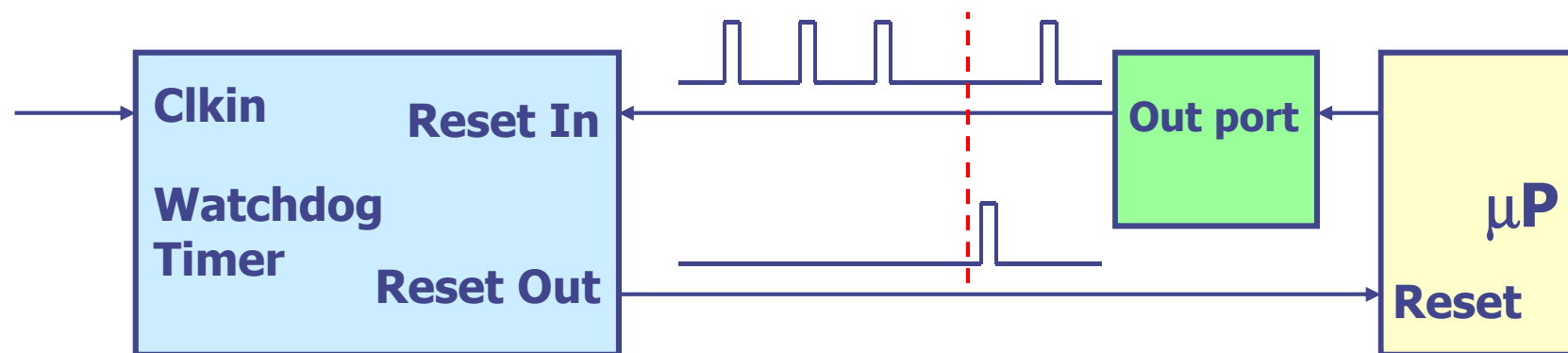
$$\text{OC1RS} = t_{\text{ON}} / T_{\text{IN}} = 25000 / 3.2 = 7813$$

PIC32 – Resolução do sinal PWM

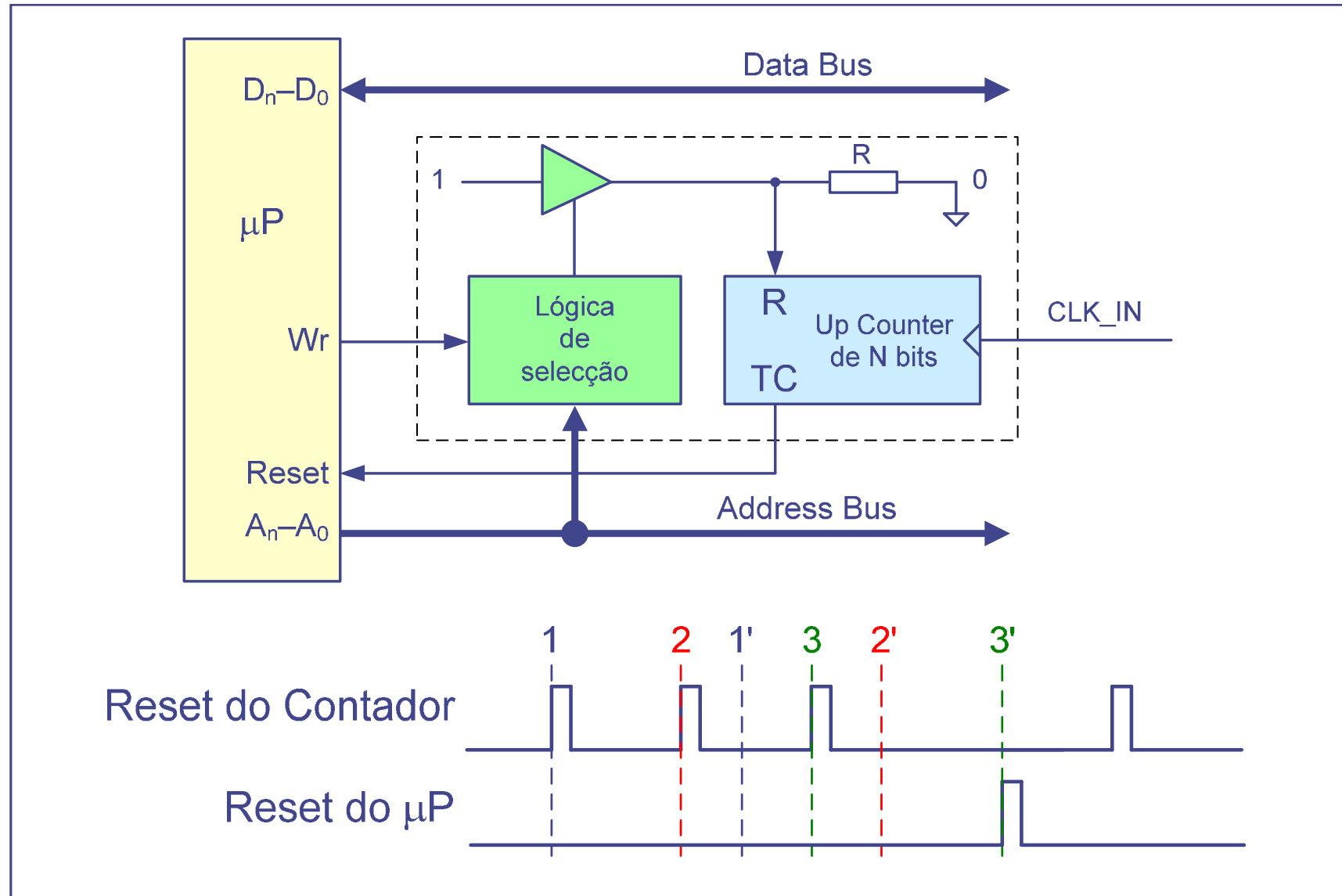
- A resolução de um sinal PWM dá uma medida do número de níveis com que se pode variar o "duty-cycle" do sinal
- Pode ser definido como:
 - **Resolução = $\log_2 (T_{PWM} / T_{IN})$**
 - em que T_{PWM} é o período do sinal PWM gerado e T_{IN} é o período do sinal à entrada do gerador de PWM
- Para o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



"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