

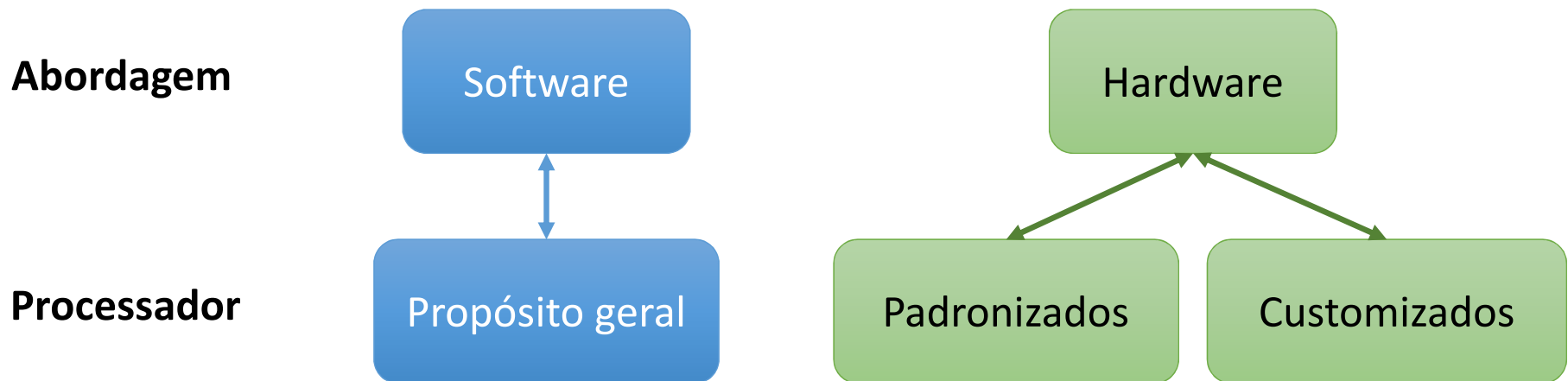
# EA075 - Introdução ao projeto de sistemas embarcados (1S/2016) ::

## Periféricos: Processadores Dedicados Padronizados

Prof. Christian Esteve Rothenberg  
[chesteve@dca.fee.unicamp.br](mailto:chesteve@dca.fee.unicamp.br)

# Introdução

- **Processador dedicado:** sistema digital projetado para realizar uma tarefa **específica** de computação.
- Algumas tarefas de computação são tão comuns que **processadores dedicados padronizados** (também chamados de **periféricos**) estão disponíveis como “itens de prateleira”.
  - “*Off-the-shelf*”: pré-desenvolvido para uma tarefa comum



# Objetivos do capítulo

Processadores Dedicados **Padronizados : Periféricos**

- Temporizadores, contadores e watchdog
- UART
- Modulador de largura de pulsos (PWM)
- LCD
- Controladores de teclado
- Motor de passos
- Conversores A/D e D/A

# Porque utilizar um periférico

Um projetista de sistemas embarcados pode escolher utilizar um processador dedicado padronizado para implementar parte de uma funcionalidade desejada do sistema e, com isso, obter alguns benefícios:

- Em comparação a um Customizado
  - Custo NRE baixo
  - Custo unitário pequeno (grandes volumes)
- Em comparação a um de Uso Geral
  - Melhor desempenho
  - Baixo consumo
  - Tamanho

# Porque utilizar um periférico

**Trade-off:** Quando o projeto já utiliza um processador de uso geral:

- Adicionar um processador padronizado, ao invés de adicionar as funcionalidades em software **aumenta consumo e tamanho**
- Por outro lado, **melhora desempenho**, pois libera o processador de uso geral para outras tarefas

# Temporizadores / Contadores

- **Temporizador:** circuito capaz de medir intervalos de tempo.
  - Pode gerar eventos temporais – por exemplo, para manter o sinal verde em um semáforo por 10 s.
  - Pode medir o tempo decorrido entre a ocorrência de eventos – por exemplo, para computar a velocidade de um carro.

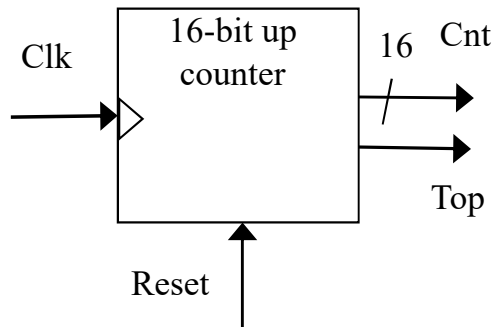
# Temporizadores / Contadores

- **Como medir o tempo?**

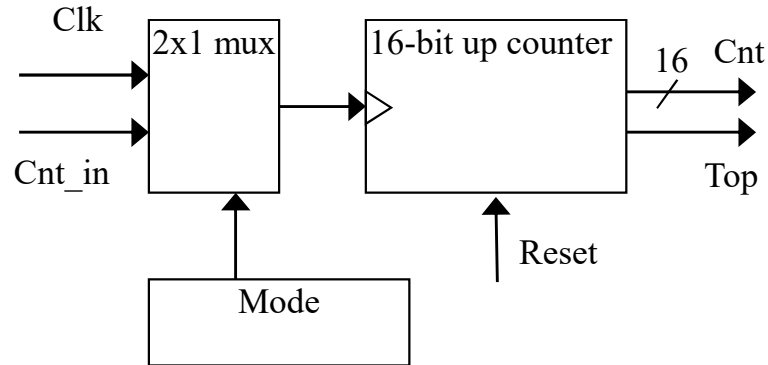
- Contando os pulsos que aparecem em um sinal de relógio de entrada que possui um período/frequência conhecido.
- **Exemplo:** Se a frequência do relógio é igual a 1 MHz e são contabilizados 2000 pulsos, o tempo decorrido é igual a 2 ms.

# Temporizador: Estruturas

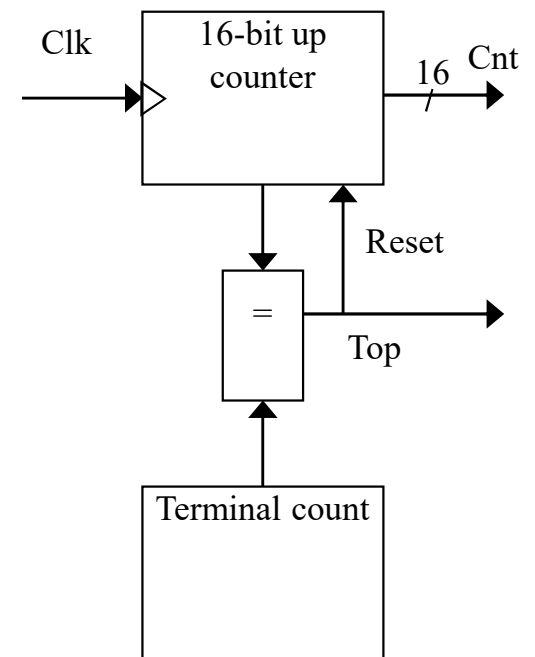
**Temporizador Básico**



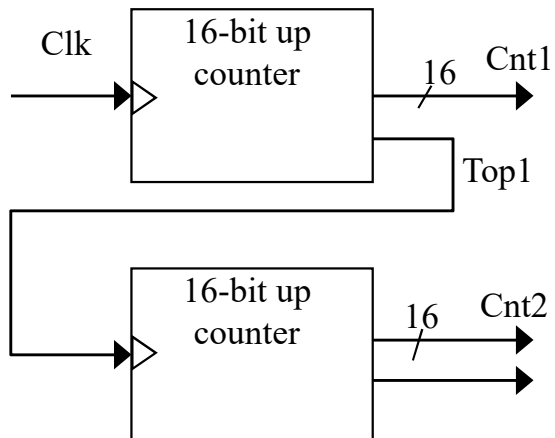
**Temporizador/Contador**



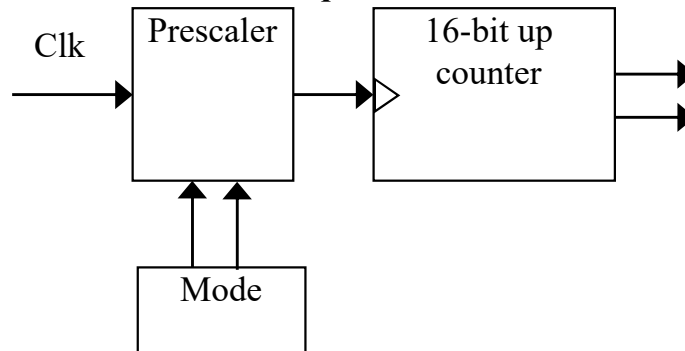
**Temporizador com valor final de contagem**



**Temporizador em cascata (16/32-bit)**

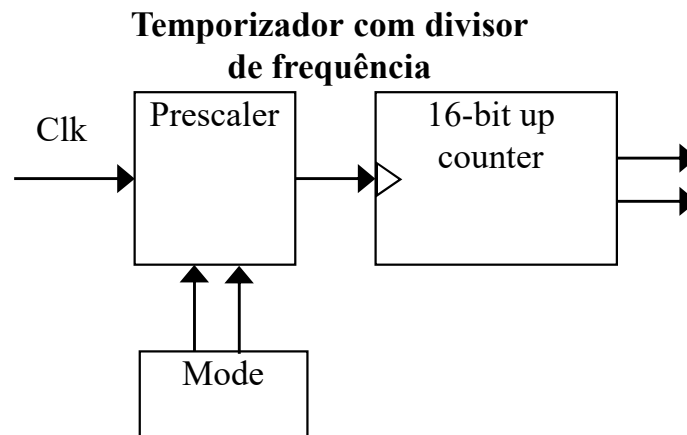
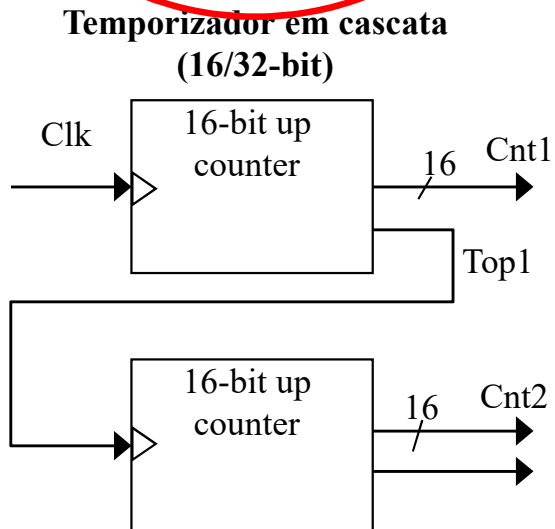
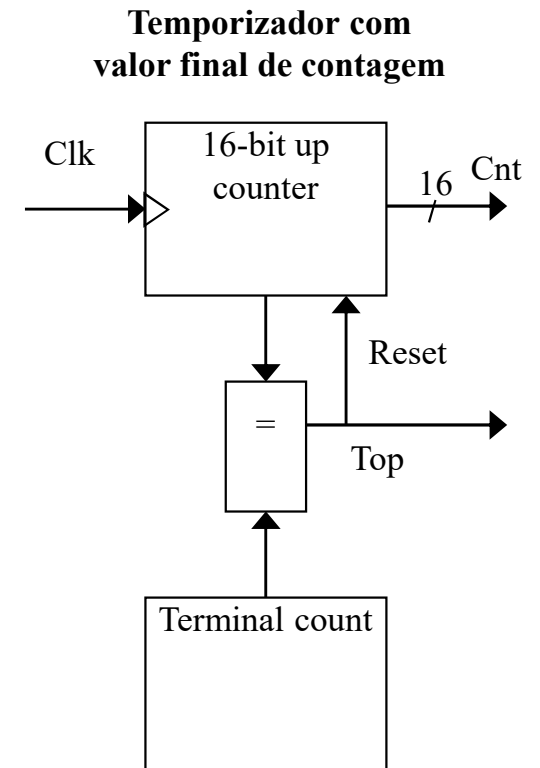
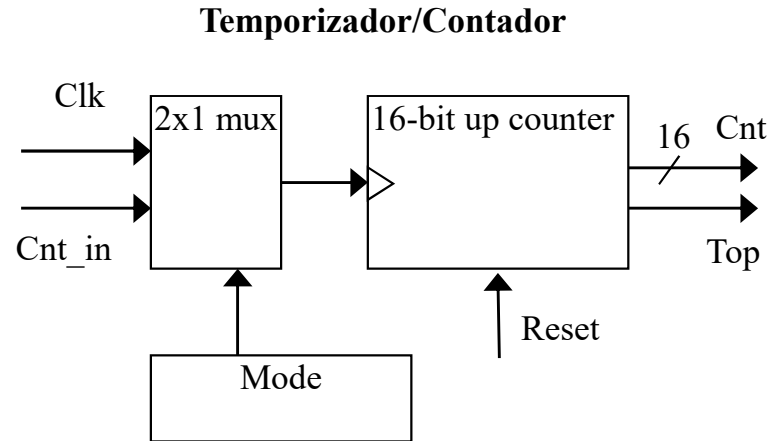
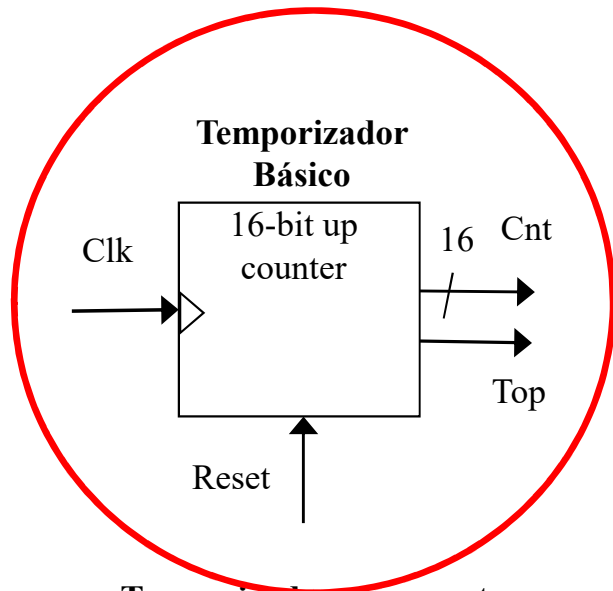


**Temporizador com divisor de frequência**



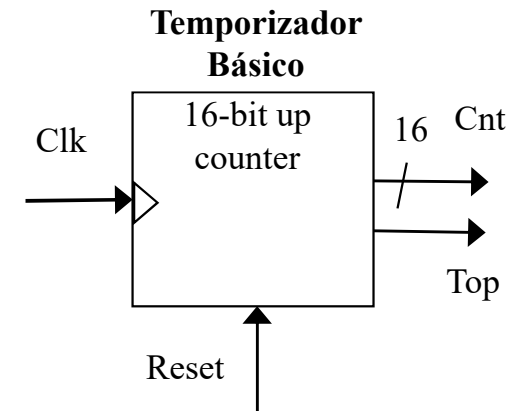


# Temporizador: Estruturas

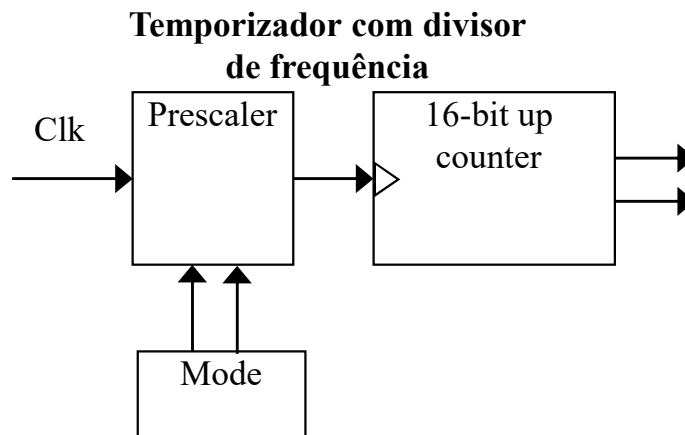
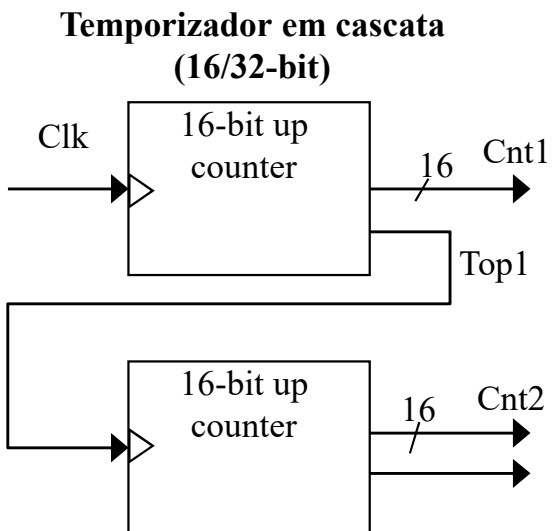
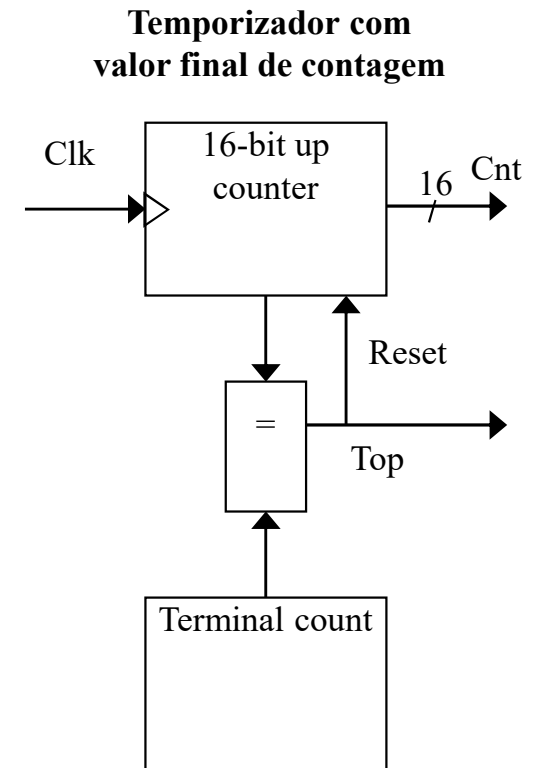
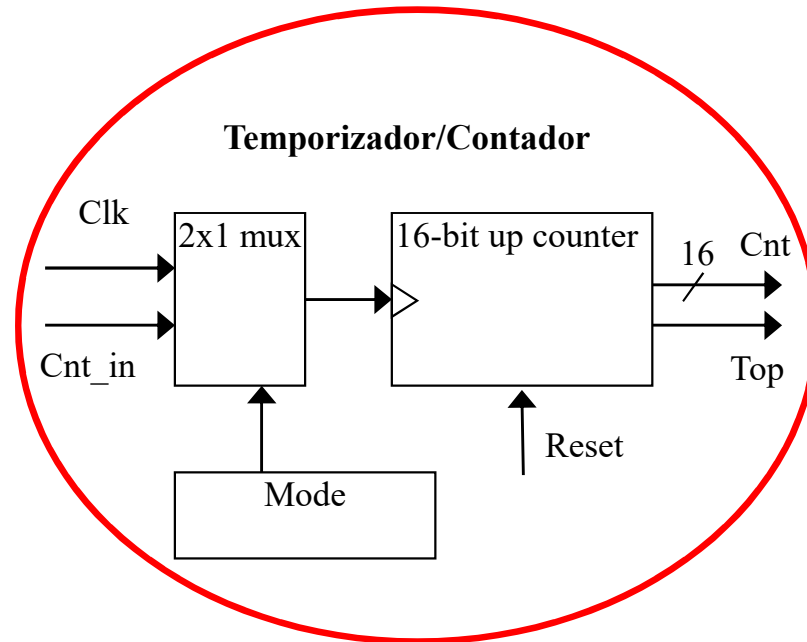
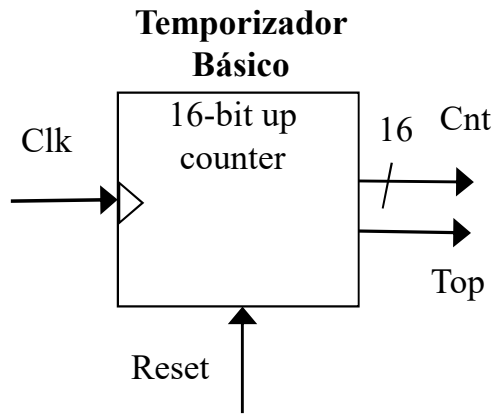


# Temporizador básico

- Temporizador (16-bits)
  - Conta pulsos de relógio
  - Se a frequência do sinal de relógio (Clk) for 100 MHz e contarmos 20.000 pulsos de Clk então, o intervalo de tempo correspondente é de 200  $\mu$ s
- Resolução
  - Mínimo intervalo= 10 ns
- Faixa
  - Máximo intervalo =  $65536 * 10 \text{ ns} = 655,36 \mu\text{s}$
- Saída *Top*: indica que a contagem máxima foi alcançada (overflow),
  - Nesse caso, o contador reinicia no zero (wrap-around)



# Temporizador: Estruturas

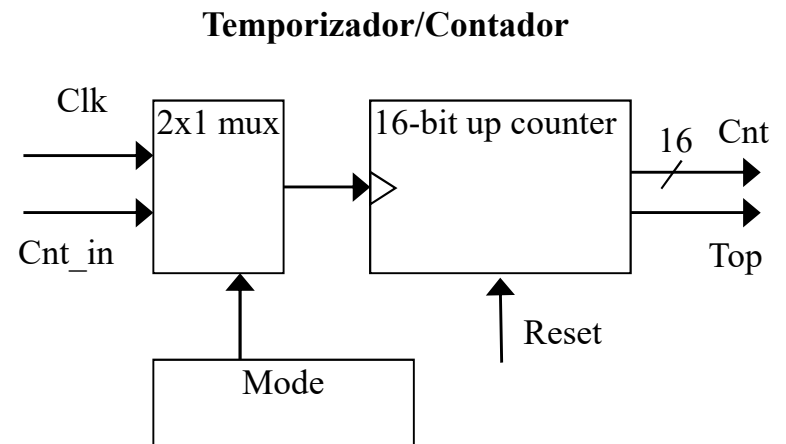


# Temporizador/Contador

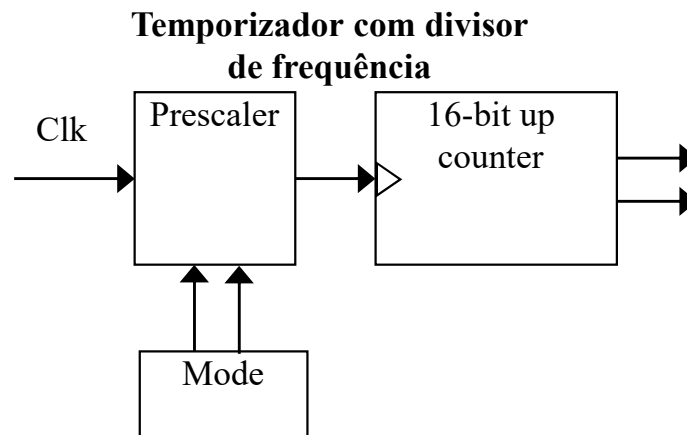
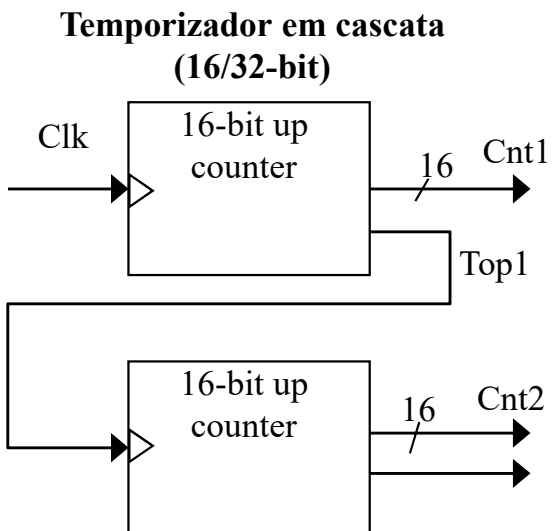
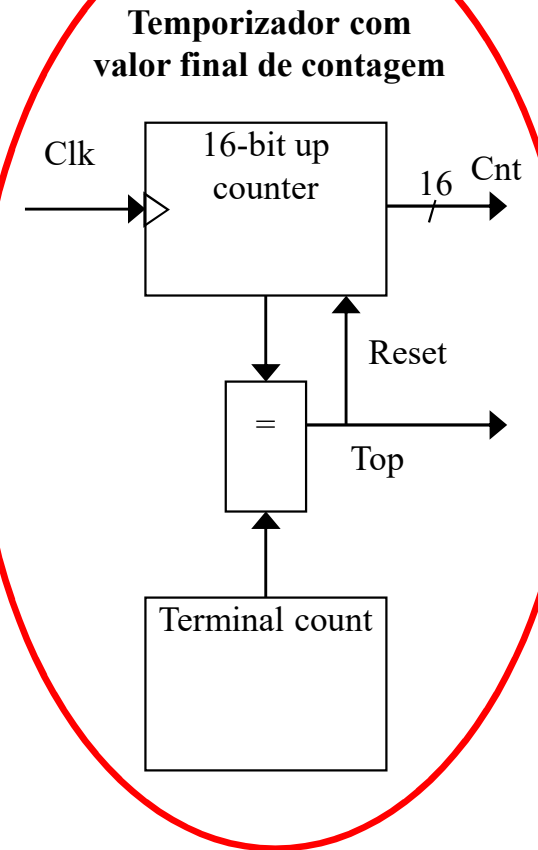
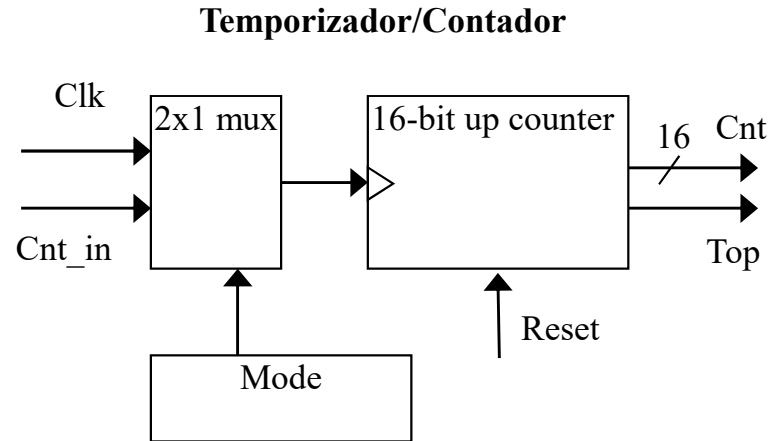
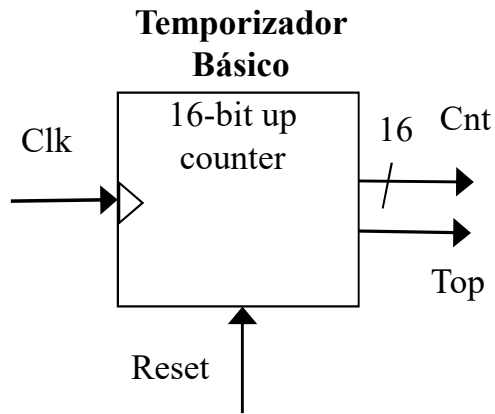
- É uma versão mais genérica do que um temporizador
- Ao invés de contar pulsos de relógio, ele conta pulsos de algum sinal de entrada
  - Ex: Número de pessoas que passa em uma catraca
- Temporizadores e contadores podem ser combinados para medir taxas
  - Ex: Número de voltas que uma roda de carro dá por segundo

# Temporizador/Contador

- Normalmente temos um único dispositivo
- Através de um sinal de modo de operação, configuramos para operar como temporizador ou como contador

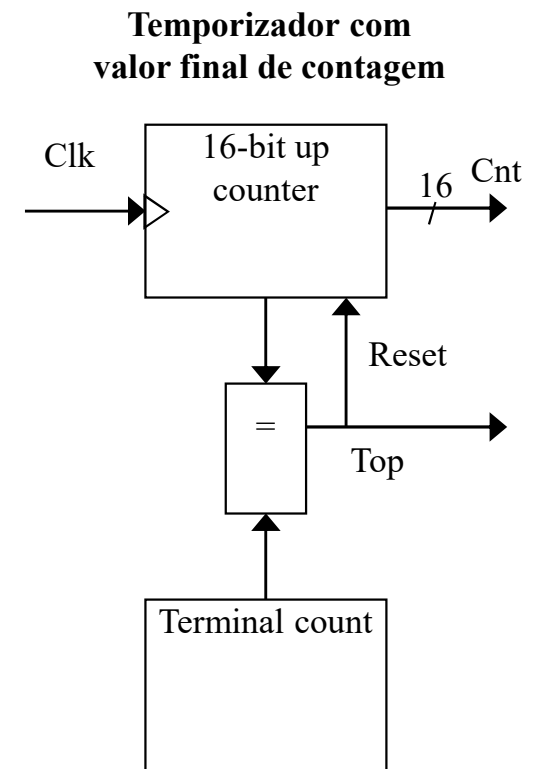


# Temporizador: Estruturas

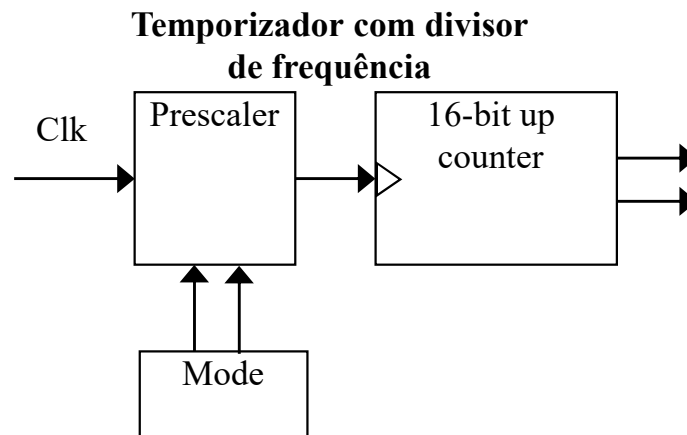
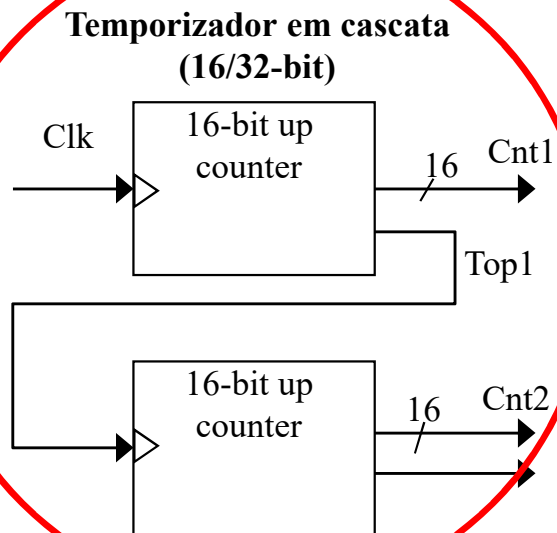
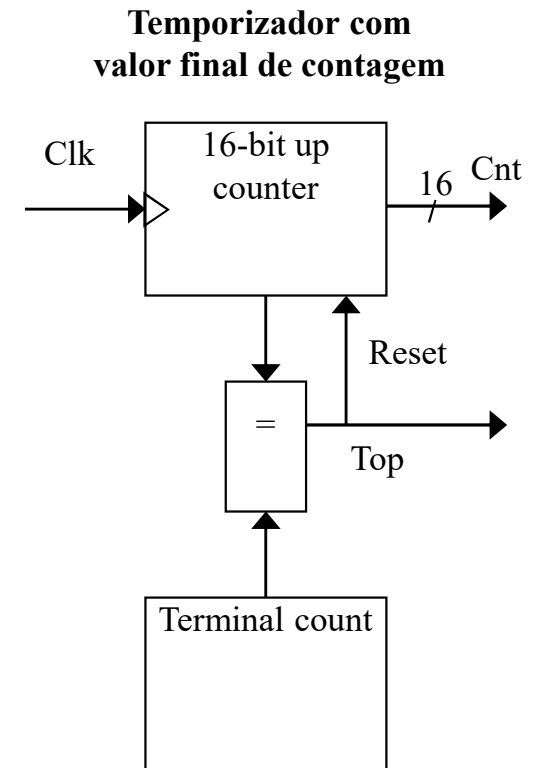
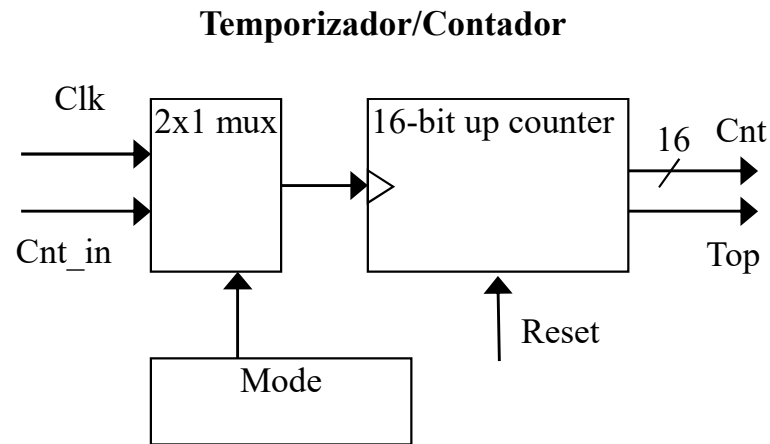
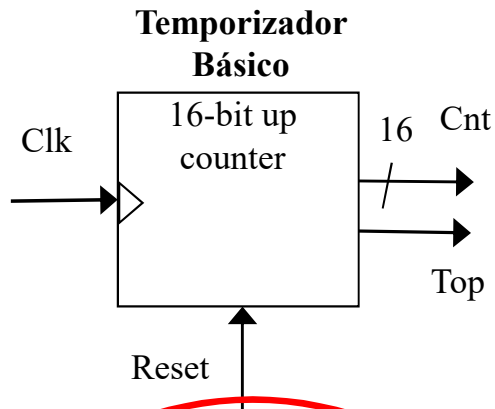


# Temporizador com valor final de contagem

- Temporizador com valor final
  - Conta pulsos de relógio
  - O usuário indica o intervalo desejado e calculamos  $n^{\circ}$  ciclos através da fórmula:  
 $n^{\circ} \text{ ciclos} = \text{intervalo} / \text{período do relógio}$
- Exemplo
  - Intervalo desejado = 3  $\mu\text{s}$
  - Período do relógio = 10 ns
  - $n^{\circ} \text{ ciclos} = 3 \times 10^{-6} / 10 \times 10^{-9} = 300$
- Inclui um comparador: seta a saída *Top* quando o valor desejado foi atingido. Este sinal *Top* gera alguma interrupção e também reseta o temporizador.



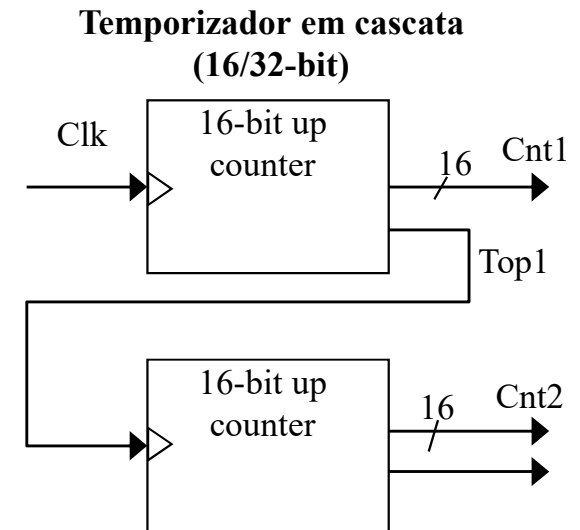
# Temporizador: Estruturas





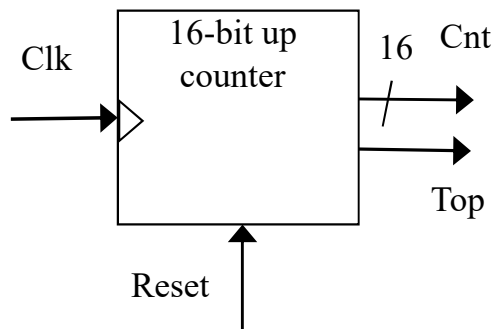
# Temporizador em cascata

- Estende o valor máximo de contagem (faixa do temporizador)
  - A saída *top* do primeiro contador é utilizada como pulso de entrada para o segundo contador
- Exemplo
  - 1 contador 16-bits:
    - faixa =  $65536 = 2^{16}$
  - 2 contadores 16-bits
    - faixa =  $(65536 * 65536) = 2^{32}$

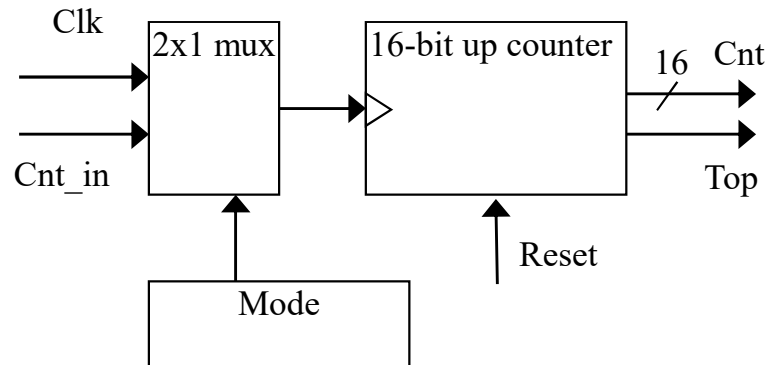


# Temporizador: Estruturas

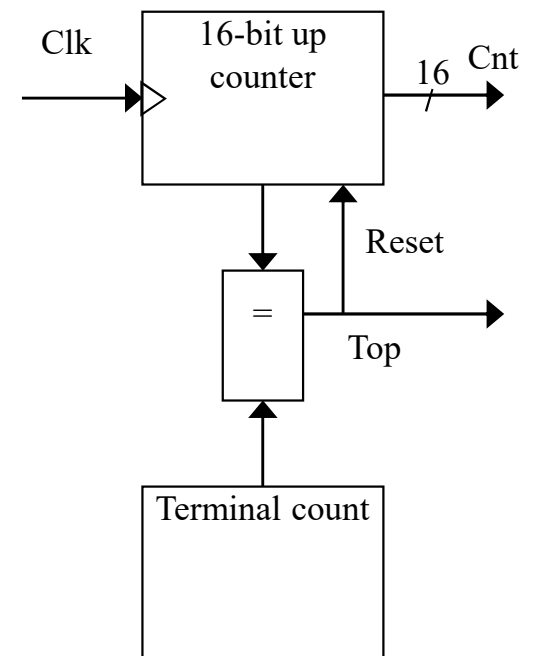
**Temporizador Básico**



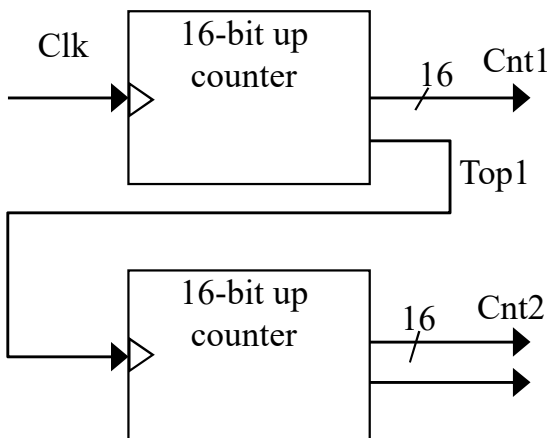
**Temporizador/Contador**



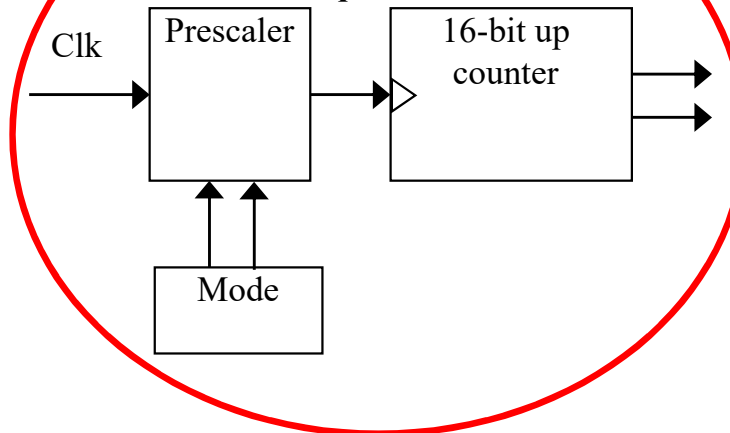
**Temporizador com valor final de contagem**



**Temporizador em cascata (16/32-bit)**



**Temporizador com divisor de frequência**



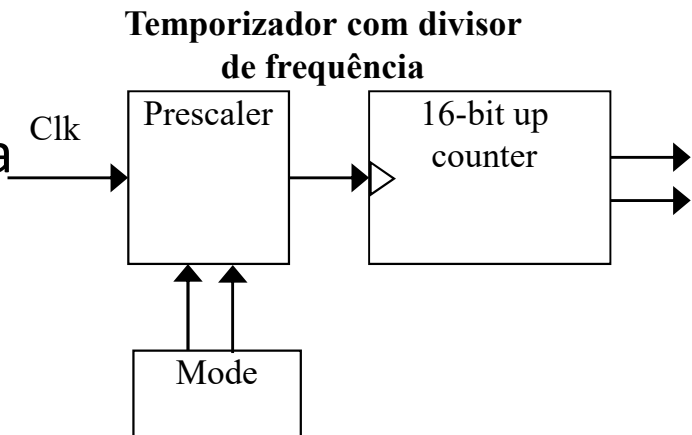
# Temporizador com divisor de frequência

- Estende o valor máximo de contagem (faixa do temporizador)

- A extensão da faixa se dá pela diminuição da resolução
- Os bits setados em *Mode* definem a frequência do sinal de saída

- Exemplo

- Temporizador original
  - Resolução= 10 ns;
  - Faixa =  $65536 \times 10 \text{ ns} = 655,36 \mu\text{s}$
- Divisor = freq./8
  - Nova resolução = 80ns
  - Nova Faixa =  $65536 \times 80 \text{ ns} = 5,25 \text{ ms}$

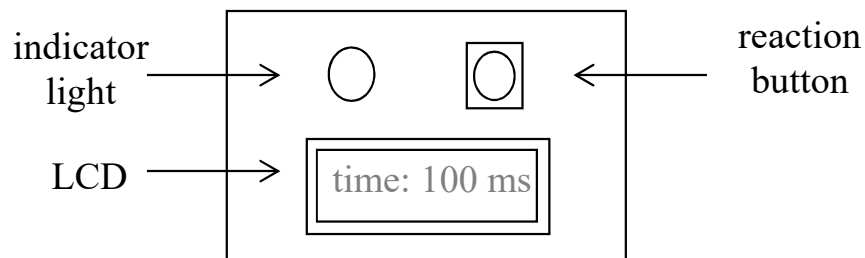


# Temporizadores / Contadores

- É possível utilizar **um processador genérico** para implementar um temporizador:
  - Sabendo o número de ciclos que cada instrução consome, basta escrever um laço (loop) que execute uma determinada sequência de instruções.
  - Quando o loop se encerra, sabe-se que um determinado período de tempo (ciclos) se passou.

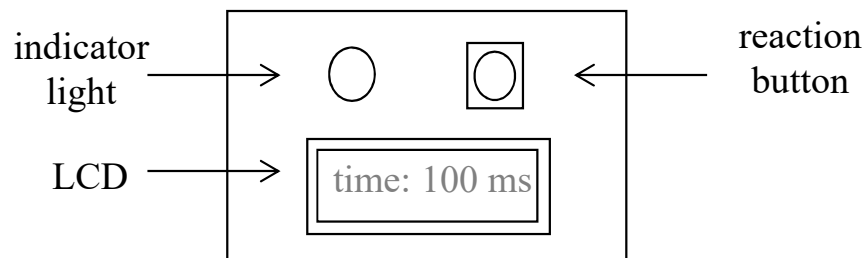
# Exemplo: tempo de reação

- Objetivo: medir o tempo que uma pessoa demora para apertar um botão ao ver um LED se acender. Precisão de ms e tempo esperado da ordem de segundos (s);
- Processador de uso geral com temporizador embutido
- O temporizador será incrementado a cada 1 ciclo de instrução, sendo cada ciclo de instrução igual a 6 pulsos de relógio. O relógio tem uma frequência de 12 Mhz (ou seja  $T=83,33\text{ns}$ ) e o temporizador é de 16-bits



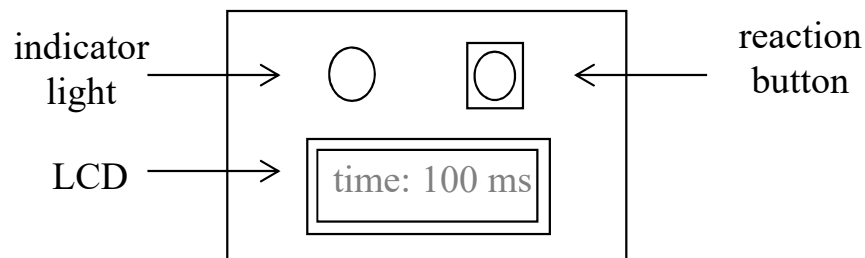
# Exemplo: tempo de reação

- O temporizador será incrementado a cada 1 ciclo de instrução, sendo cada ciclo de instrução igual a 6 pulsos de relógio. O relógio tem uma frequência de 12 Mhz (ou seja  $T=83,33\text{ns}$ )
  - Temporizador de 16-bit, período de relógio de 83.33 ns, contador incrementa a cada 6 ciclos
  - Resolução =  $6 \cdot 83.33 = 0.5 \mu\text{s}$  (atualização)
  - Faixa =  $65536 \cdot 0.5 \mu\text{s} = 32.77 \text{ ms}$



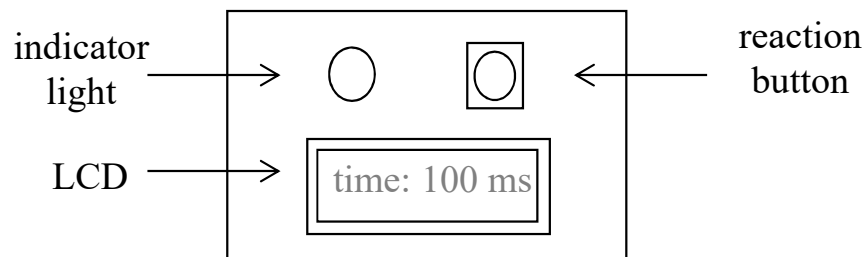
# Exemplo: tempo de reação

- Resolução =  $6 \times 83.33 = 0.5 \mu\text{s}$  (maior do desejado = ms)
- Faixa =  $65536 \times 0.5 \mu\text{s} = 32.77 \text{ ms}$  (menor do que desejado = s)
- Quanto o contador conta em 1 ms ?
  - 1 ciclo de instr  $\Rightarrow 6 \times 83,33\text{ns} = 0,5 \mu\text{s}$
  - x ciclos de instr  $\Rightarrow 1 \text{ ms} \Rightarrow x = 2000$
- Assim inicializaremos o contador com  $65536 - 2000 = 63536$
- Cada vez que Top = 1 terá passado 1 ms e fazemos contador = 63536



# Exemplo: tempo de reação

Toda vez que o sinal  $top = 1$ , o contador (*hardware*) recomeça a contagem do valor `MS_INIT` e incrementamos o contador de milissegundos (*software*).



```
/* main.c */

#define MS_INIT    63536
void main(void){
    int count_milliseconds = 0;

    configure timer mode
    set Cnt to MS_INIT

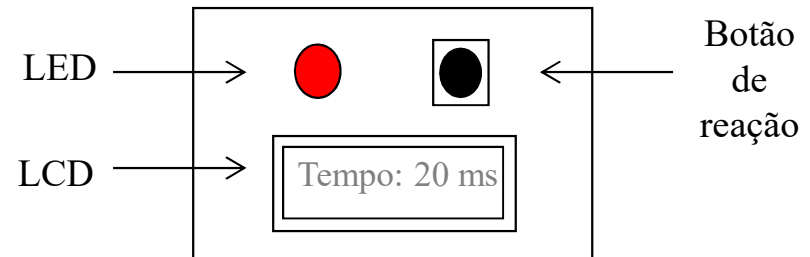
    wait a random amount of time
    turn on indicator light
    start timer

    while (user has not pushed reaction button){
        if(Top) {
            stop timer
            set Cnt to MS_INIT
            start timer
            reset Top
            count_milliseconds++;
        }
    }
    turn light off
    printf("time: %i ms", count_milliseconds);
}
```



# Temporizadores / Contadores

- Exemplo: tempo de reação



- Mede o tempo decorrido entre o acendimento do LED e o acionamento do botão por parte do usuário.
- Processador genérico de 12 MHz com temporizador embutido.
- A cada ciclo de instrução (igual a seis ciclos de relógio), o temporizador é incrementado.
  - A cada  $6 \times 1/12\text{MHz} = 0,5 \mu\text{s}$ , o valor da contagem (16 bits) é incrementado. Esta é a resolução do temporizador.
  - Faixa:  $65535 \times 0,5 \mu\text{s} = 32,77 \text{ ms}$
- Exige-se que o programa determine o tempo de reação com precisão de ms, e espera-se que o tempo de reação seja da ordem de segundos.

# Temporizadores / Contadores

- Exemplo: tempo de reação
  - A faixa do temporizador não é suficiente para medir um tempo em segundos.
  - **Ideia:**
    - Inicializar um valor no contador de 16 bits tal que a contagem final (top = 1) seja atingida após 1 ms.
      - ✓  $c = 1 \text{ ms} / 0,5 \mu\text{s} = 2000$ .
      - ✓ O valor da contagem é, portanto,  $65535 - 2000 = 63535$ .
    - Resta ao programa contar o número de vezes que o sinal top é igual a 1 para determinar o tempo de reação.

# Temporizadores / Contadores

- Exemplo: tempo de reação

```
/* main.c */

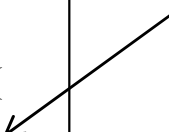
#define MS_INIT    63535
void main(void){
    int count_milliseconds = 0;

    configure timer mode
    set Cnt to MS_INIT

    wait a random amount of time
    turn on indicator light
    start timer

    while (user has not pushed reaction button){
        if(Top) {
            stop timer
            set Cnt to MS_INIT
            start timer
            reset Top
            count_milliseconds++;
        }
    }
    turn light off
    printf("time: %i ms", count_milliseconds);
}
```

Toda vez que `top = 1`, o contador (*hardware*) recomeça a contagem do valor `MS_INIT` e incrementamos o contador de milissegundos (*software*).



# Temporizador Watchdog

- É configurado pelo usuário com um valor  $T$
- Caso o usuário (programa) não gere um sinal a cada intervalo de tempo  $T$  o temporizador sinaliza
- Uma aplicação comum em nosso contexto é permitir que um sistema embarcado se reinicialize em caso de falhas (que seriam detectadas via o não recebimento de um sinal);
  - O sinal do watchdog pode ser conectado ao pino de reset do processador
  - Em casos onde o reset não é viável, o sinal de watchdog pode ser conectado a um pino de interrupção, que irá chamar uma rotina de interrupção e pular para uma parte segura do programa

# Temporizador Watchdog

- Outro uso: controle de *timeouts*
  - Em sistema no qual o usuário deve responder questões em tempo pré-determinado, temporizadores Watchdog evitam o uso de verificações ao longo do programa, permitindo solução mais elegante

# Exemplo: Temporizador *Watchdog*

## Máquina ATM

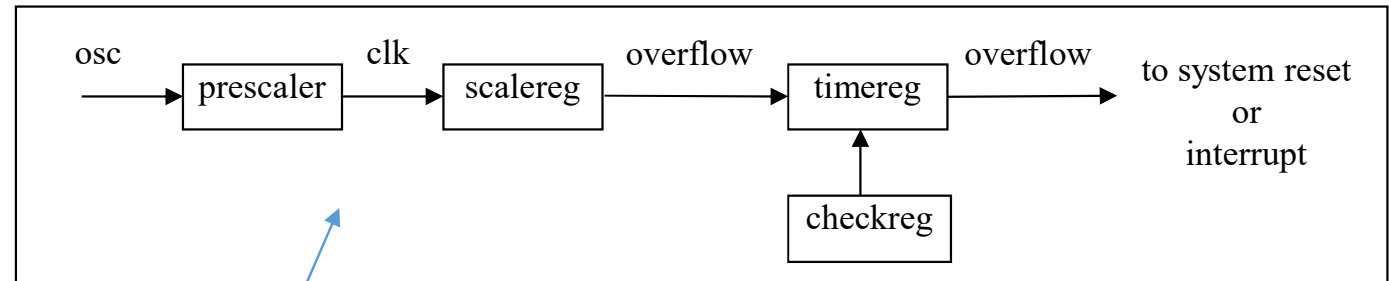
- usuário insere um cartão de banco e entra com seu código
- responde a questões (depósito ou retirada, qual conta envolvida, quanto dinheiro, etc)

Queremos construir uma ATM que termina a sessão se não houver ação em 2 min. Neste caso devolve o cartão e termina.

# Exemplo: Temporizador *Watchdog*

## Máquina ATM

- usuário insere um cartão de banco e entra com seu código
  - responde a questões (depósito ou retirada, qual conta envolvida, quanto dinheiro, etc)
- Queremos construir uma ATM que termina a sessão se não houver ação em 2 min. Neste caso devolve o cartão e termina.



Um oscilador – **osc** – é ligado a um **pre-escalador** que divide a freq. por 12, gerando o relógio – **clk**.

O sinal **clk** é conectado a um contador (up) de 11 bits – **scalereg**.

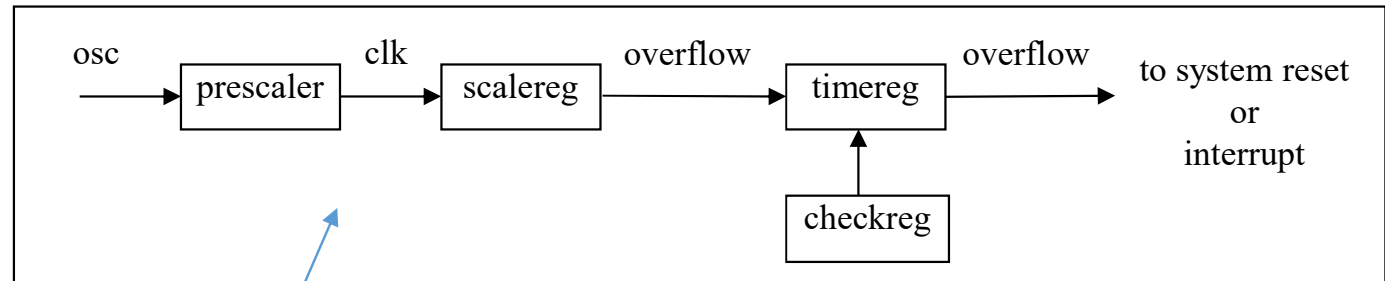
**scalereg** ao atingir overflow zera e gera sinal para o contador de 16-bits **timereg** incrementar.

Quando **timereg** fornece **overflow**, isto dispara **system reset** ou **interrupt**.

# Exemplo: Temporizador *Watchdog*

## Máquina ATM

- usuário insere um cartão de banco e entra com seu código
  - responde a questões (depósito ou retirada, qual conta envolvida, quanto dinheiro, etc)
- Queremos construir uma ATM que termina a sessão se não houver ação em 2 min. Neste caso devolve o cartão e termina.



Um oscilador – **osc** – é ligado a um **pre-escalador** que divide a freq. por 12, gerando o relógio – **clk**.

O sinal **clk** é conectado a um contador (up) de 11 bits – **scalereg**.

**scalereg** ao atingir overflow zera e gera sinal para o contador de 16-bits **timereg** incrementar.

Quando **timereg** fornece **overflow**, isto dispara **system reset** ou **interrupt**.

**checkreg** previne disparo falso de **timereg** (carregamento devido a erro).

$$\text{osc} = 12 \text{ Mhz} \gg \text{clk} = 12\text{Mhz} / 12$$

Assim  $t$  para incremento de **timereg** é obtido por:

$$2^{11} / 10^6 = 0,002048 \text{ seg}$$

Assim temos:

timereg entre 0 e 65535 (16bits)  
ou seja entre 0 e 134.215 ms

Para timeout = 2 min (=120.000 ms)

$$\text{timereg} = (134.215 - 120.000) / 0,002048 = 6941$$



# UART - Transmissão serial

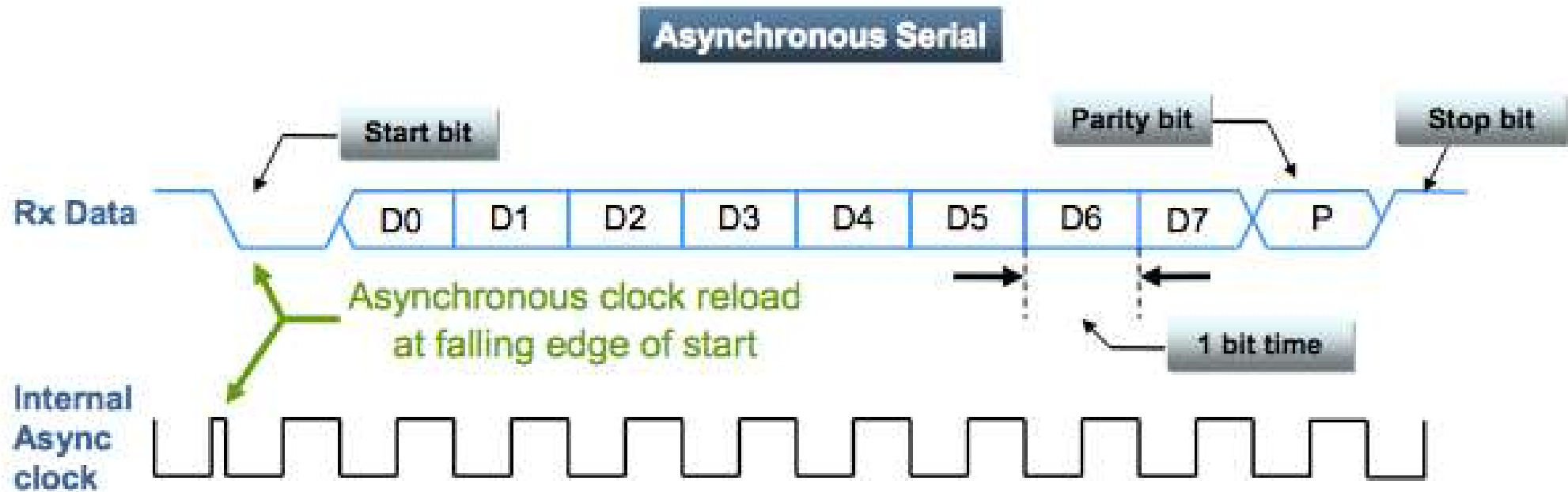
# UART: Transmissão serial

- Receptor/Transmissor universal assíncrono:
  - recebe dados seriais e os armazena como dados paralelos (usualmente um byte);
  - também recebe dados paralelos e os transmite de forma sequencial.
- Útil para comunicação entre dispositivos bastante afastados ou quando há poucos pinos de entrada/saída disponíveis.

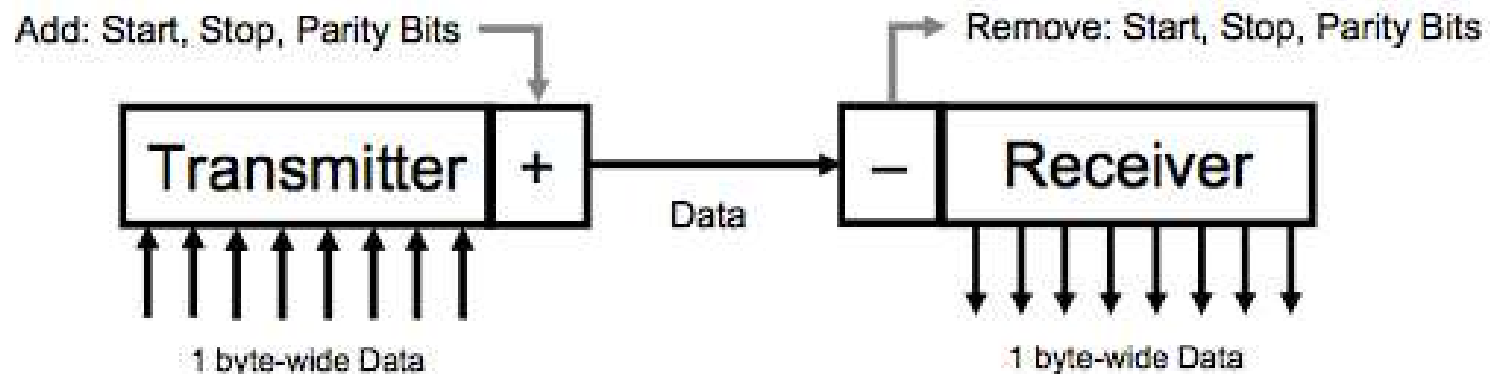
# UART: Transmissão serial

- Protocolo de comunicação:
  - Baud rate = taxa em que bits são enviados;
  - Start/stop bit(s) = número de bits usado para separar duas transmissões consecutivas;
  - Paridade = usado para verificação de erros

# UART: Transmissão serial



# UART: Transmissão serial



Transmitter	Receiver
Shifts the parallel data onto the serial line using its own clock	Extracts the data using its own clock
Also adds the start, stop, and parity check bits	Converts the serial data back to the parallel form after stripping off the start, stop, and parity bits

# UART: Baud rate

- A UART precisa de um relógio para a transmissão e recepção
- As taxas de transmissão UART são geralmente muito menores que o relógio do sistema
- Como o relógio do sistema não pode ser usado, utiliza-se um temporizador (divisor de frequência)
- Exemplo:
  - relógio do sistema = 22.1184 MHz;
  - UART baud rate = 115200

# UART: Start/Stop bits

## Start/Stop bits

- Start bit = geralmente um degrau de descida ( $1 \Rightarrow 0$ )
- Stop bit(s) = um ou mais, importante para dar tempo para o receptor se preparar para uma nova transmissão (=1)

# UART: bit de paridade

Paridade - conta o número de 1s

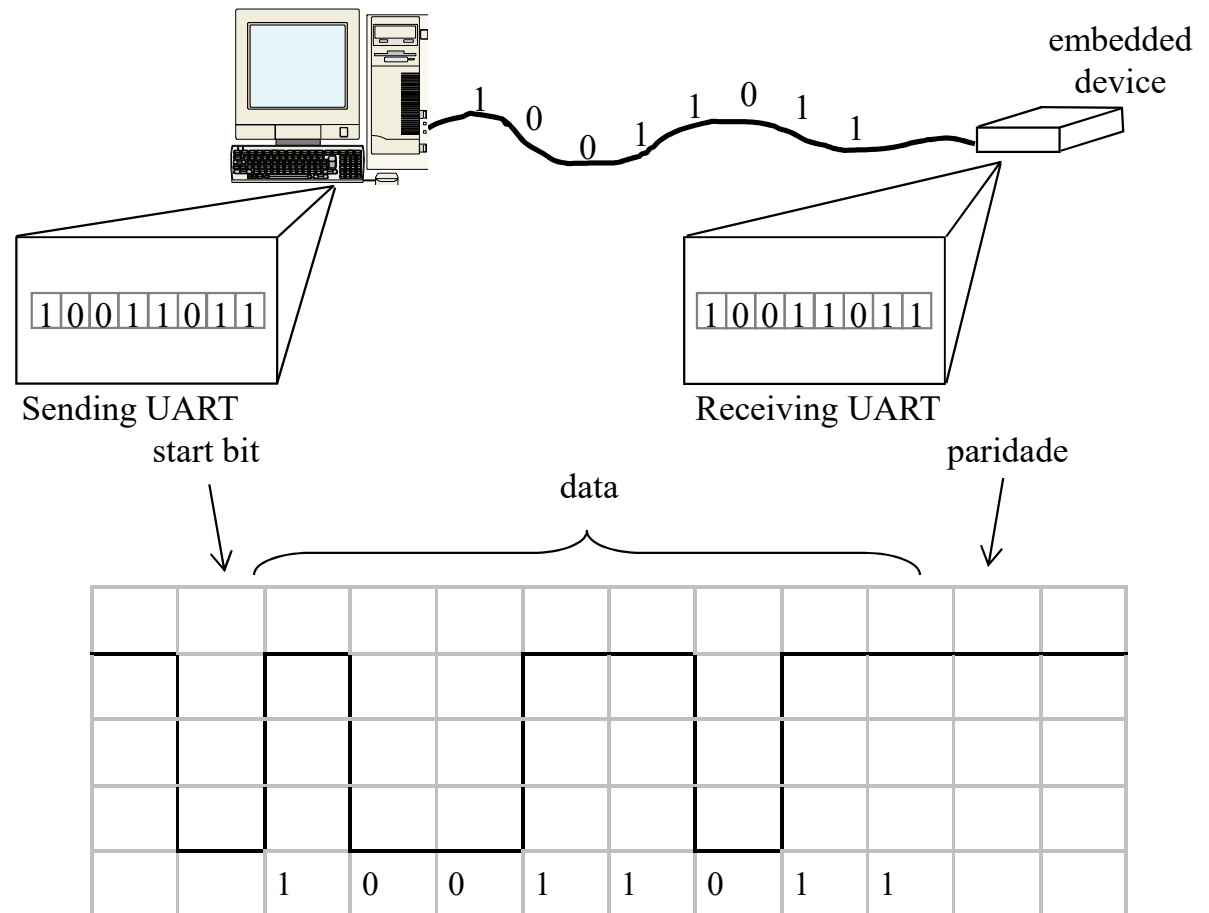
- Odd parity
- Even parity
- Sem paridade

7 bits of data	(count of 1 bits)	8 bits including parity	
		even	odd
0000000	0	00000000	00000001
1010001	3	10100011	10100010
1101001	4	11010010	11010011
1111111	7	11111111	11111110



# UART: Transmissão serial

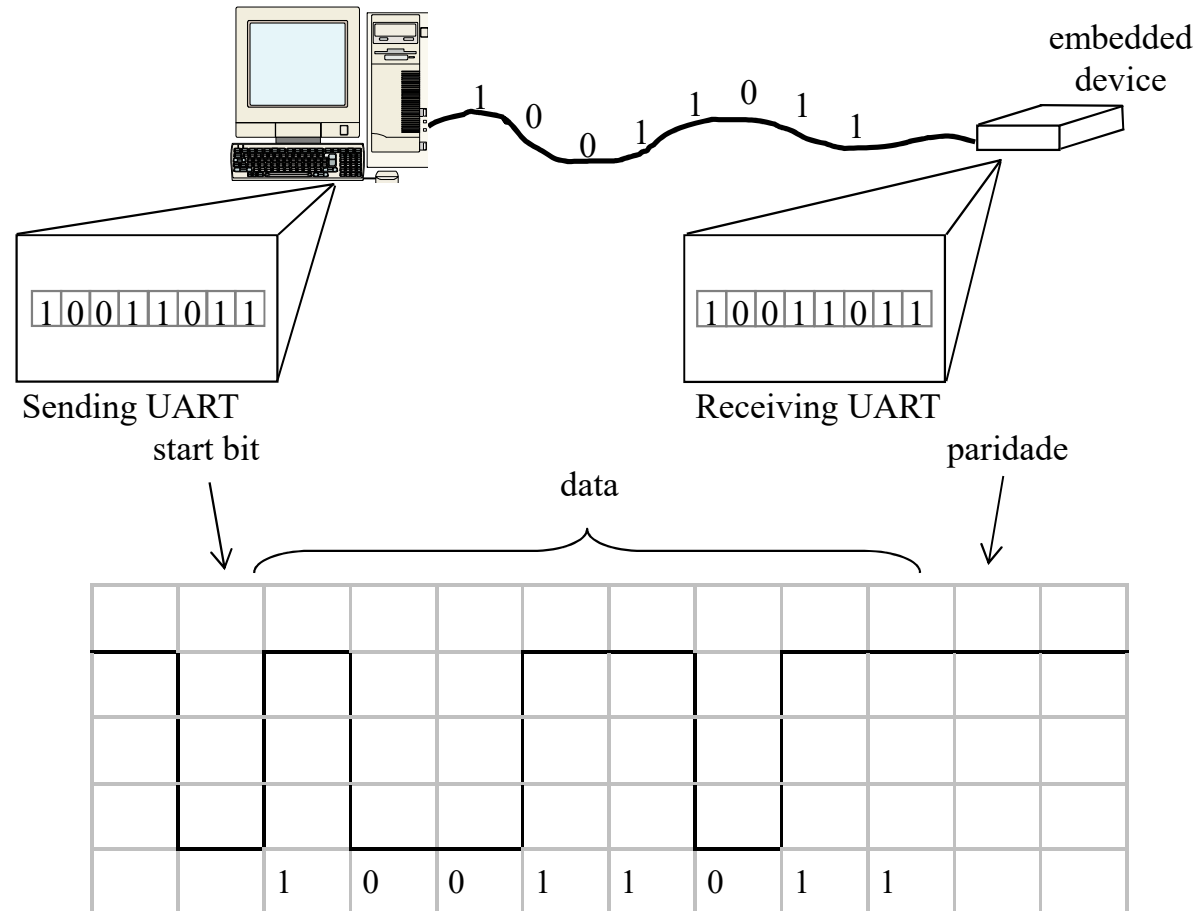
- Buffer de transmissão: registrador de deslocamento
- Transmite 1 bit de cada vez
- Registrador de deslocamento no receptor



# UART: Transmissão serial

## Transmissão

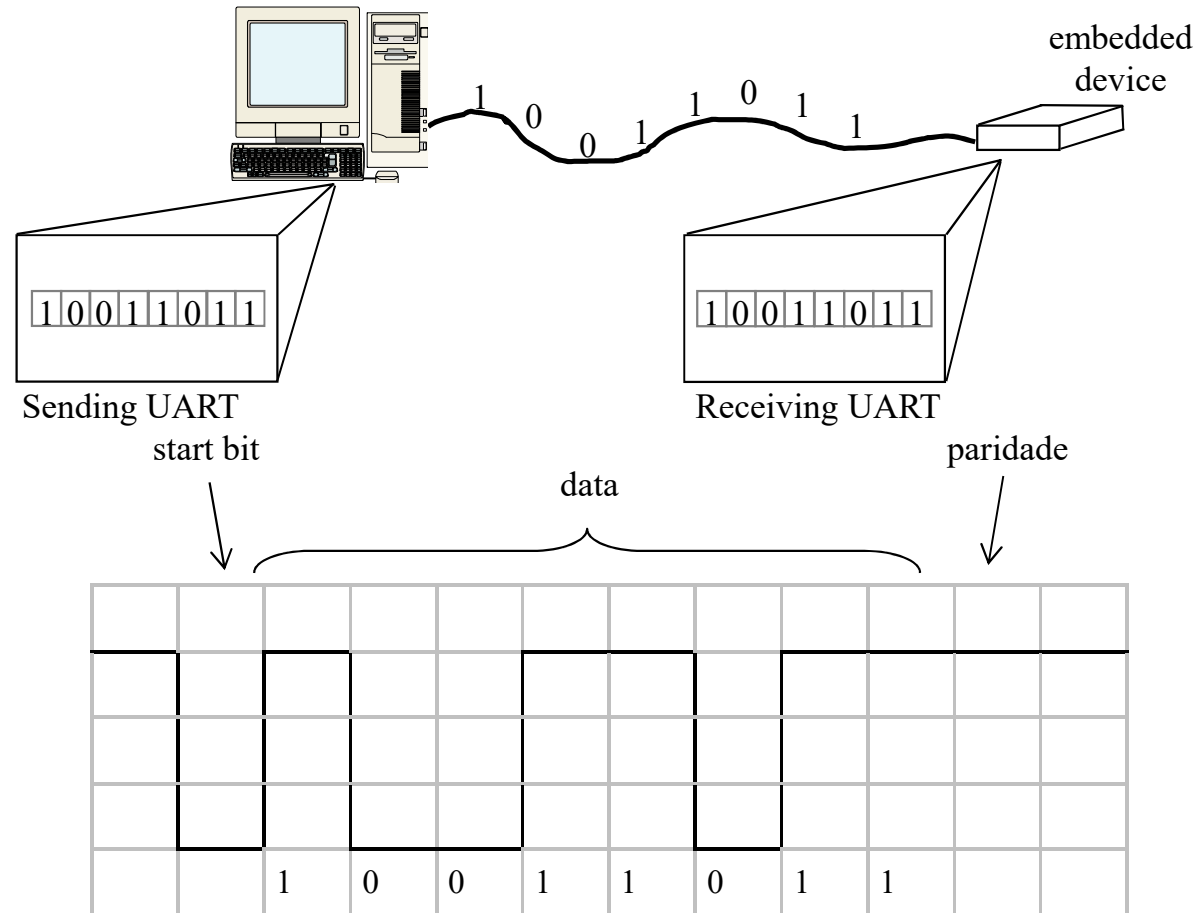
- Envia um *start bit* (1  $\Rightarrow$  0) no pino de transmissão ( $t_x$ )
- Desloca um bit de cada vez através do buffer, para o pino  $t_x$ , a uma determinada taxa
- Pode transmitir também um bit de paridade
- Sinaliza ao processador principal que está pronto para enviar mais dados



# UART: Transmissão serial

## Recepção

- Está constantemente monitorando o pino  $r_x$
- Depois de receber o *start bit*, amostra o sinal em  $r_x$ , à mesma taxa e coloca dados no registrador receptor
- Verifica o bit de paridade
- Sinaliza ao processador principal que está pronto para receber mais dados



# PWM - Pulse Width Modulator

Modulador de largura de pulsos

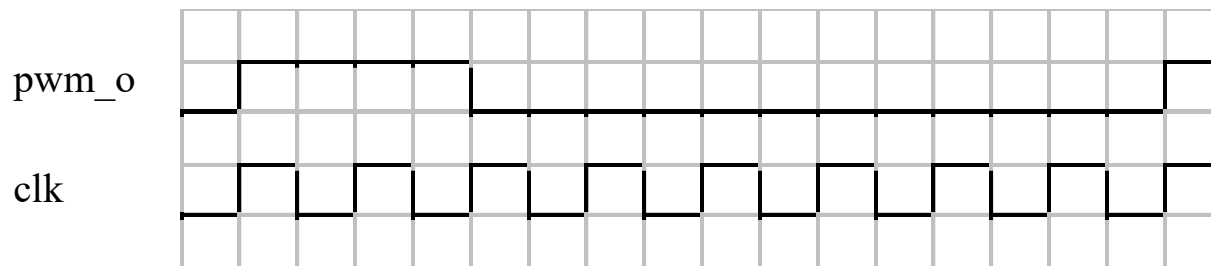
# Pulse Width Modulator (PWM)

- Gera pulsos com intervalos determinados para níveis alto/baixo
- É possível controlar a duração que o pulso permanece com valor alto e com valor baixo, indicando o período desejado e o ciclo ativo
- Ciclo Ativo (*duty cycle*): percentual do período em nível alto
  - Onda quadrada: ciclo ativo é de 50%

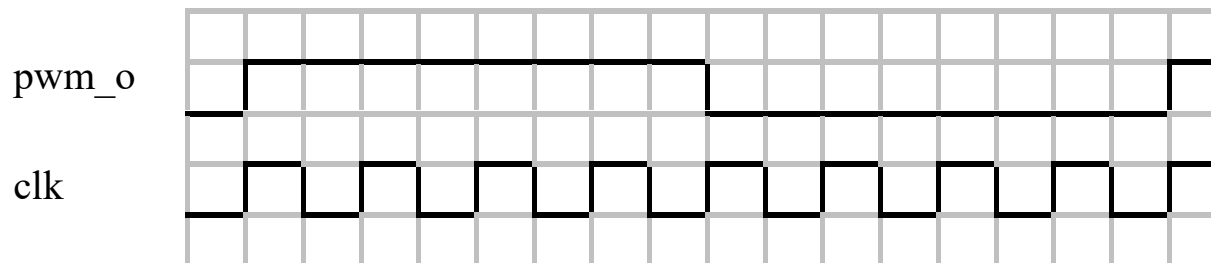
# PWM: Aplicações

- Geração de sinal periódico para outro dispositivo
- Controle da tensão média aplicada a um dispositivo elétrico
- Codifica comandos, permitindo ao receptor utilizar o temporizador para a decodificação
  - Por exemplo, é possível dirigir um carrinho controlado remotamente enviando pulsos de diferentes larguras.
    - Pulso de largura 1 ms = “vire à esquerda”
    - Pulso de largura 4 ms = “vire à direita”
    - Pulso de largura 8 ms = “vá em frente”.
  - Um temporizador acionado quando o pulso começa e desligado quando o pulso termina poderia medir o intervalo de tempo (largura do pulso) e identificar o comando.

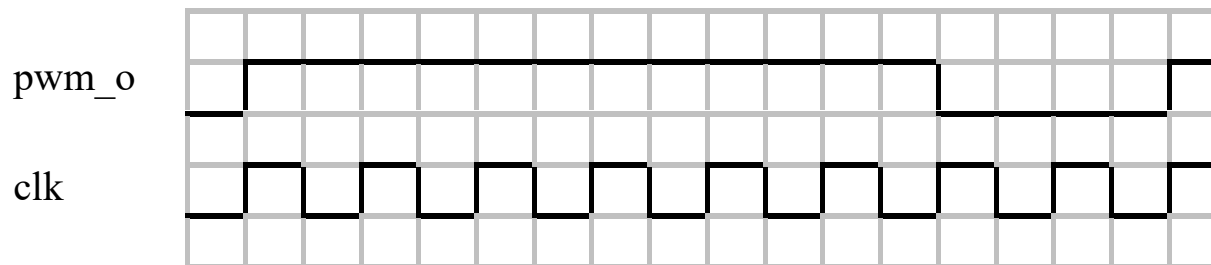
# Pulse Width Modulator (PWM)



25% duty cycle – average `pwm_o` is 1.25V



50% duty cycle – average `pwm_o` is 2.5V.



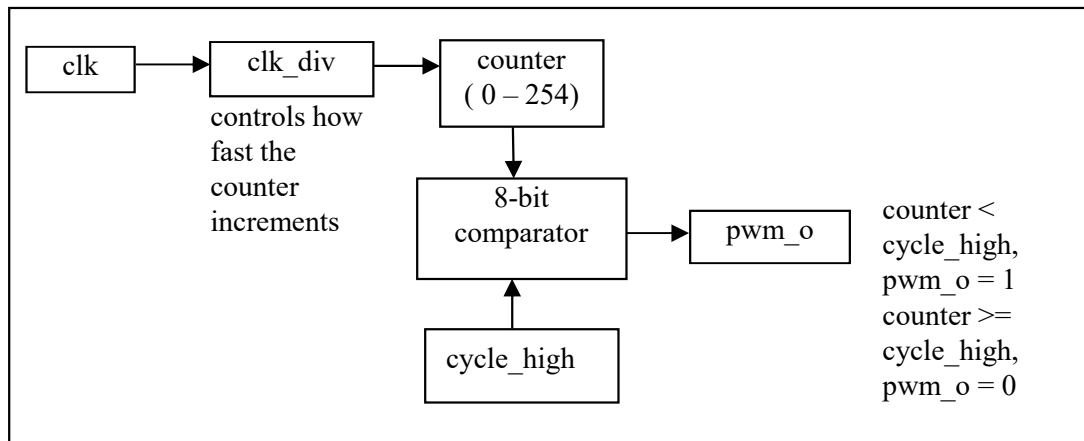
75% duty cycle – average `pwm_o` is 3.75V.

# PWM: controlando um motor DC

Tensão de entrada	Duty cycle (%)	RPM do motor DC
0	0	0
2.5	50	4600
3.75	75	6900
5.0	100	9200

Relação entre a tensão aplicada e a velocidade do motor DC

- Suponha que, dada uma carga constante, a velocidade de um motor DC seja proporcional a tensão aplicada a ele, conforme tabela ao lado
- Precisamos escolher um um “duty cicle” para que a tensão de saída seja a desejada, resultando na velocidade desejada



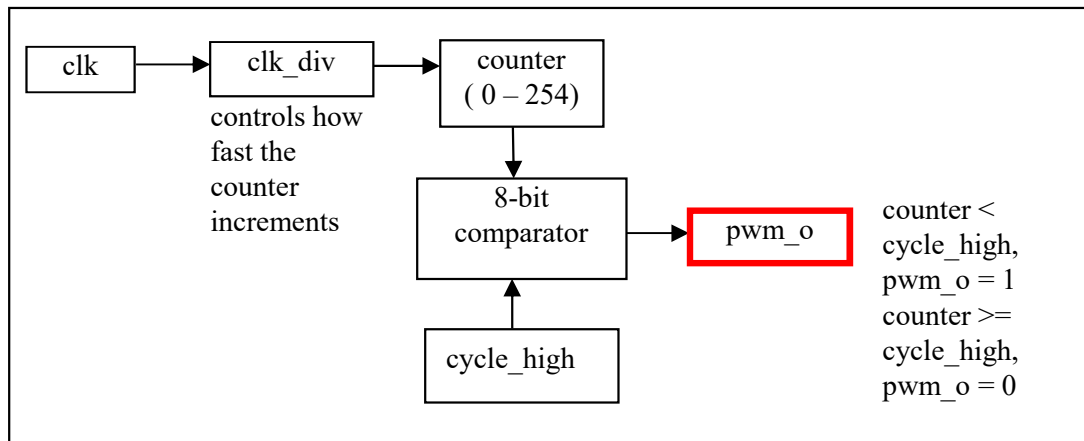
Internal Structure of PWM



# PWM: controlando um motor DC

Tensão de entrada	Duty cycle (%)	RPM do motor DC
0	0	0
2.5	50	4600
3.75	75	6900
5.0	100	9200

Relação entre a tensão aplicada e a velocidade do motor DC



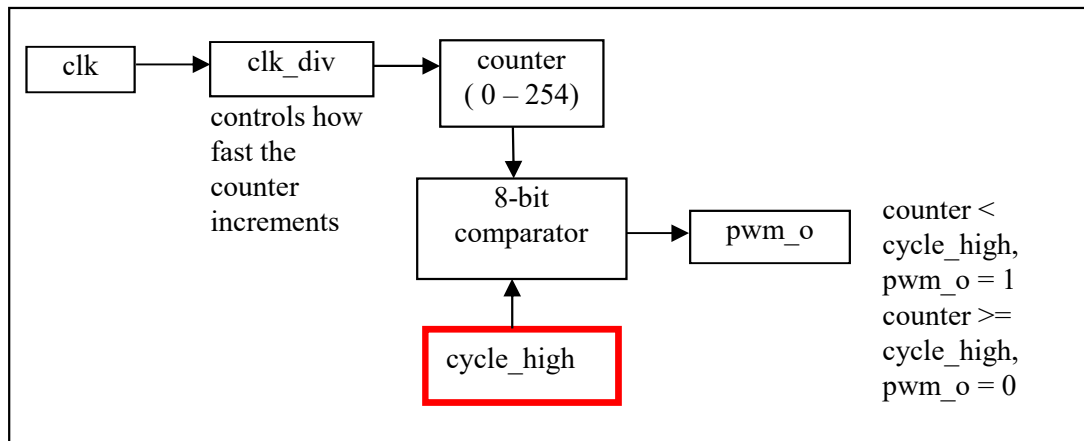
Internal Structure of PWM

- Contador `clk_div` - define o período do PWM
- Depois de um determinado número de pulsos de relógio (`clk`), `clk_div` manda um pulso para o contador de 8-bits
- Contador é incrementado
- Comparador olha os valores do contador e do `cycle_high`
- Registrador `cycle_high` – especifica o duty cycle
  - Se contador  $<$  `cycle_high` => `pwm_o = 1`
  - Se contador  $\geq$  `cycle_high` => `pwm_o = 0`

# PWM: controlando um motor DC

Tensão de entrada	Duty cycle (%)	RPM do motor DC
0	0	0
2.5	50	4600
3.75	75	6900
5.0	100	9200

Relação entre a tensão aplicada e a velocidade do motor DC



Internal Structure of PWM

- Registrador cycle\_high – como escolher o valor?
- Especifica o duty cycle
- Se queremos que o motor funcione a 4600 rpm:
  - Precisamos de um duty cycle de 50%
  - O contador de 8-bits deve ser na metade dos ciclos maior do que cycle\_high e na outra metade dos ciclos, menor do que cycle\_high
  - $\text{Cycle\_high} = 254 \times 0.5 \approx 127$
- Se queremos que o motor funcione a 6900 rpm:
  - Precisamos de um duty cycle de 75%
  - O contador de 8-bits deve ser em 3/4 dos ciclos maior do que cycle\_high e em 1/4 dos ciclos, menor do que cycle\_high
  - $\text{Cycle\_high} = 254 \times 0.75 \approx 191$

# LCD - Liquid Crystal Display

Controlador LCD

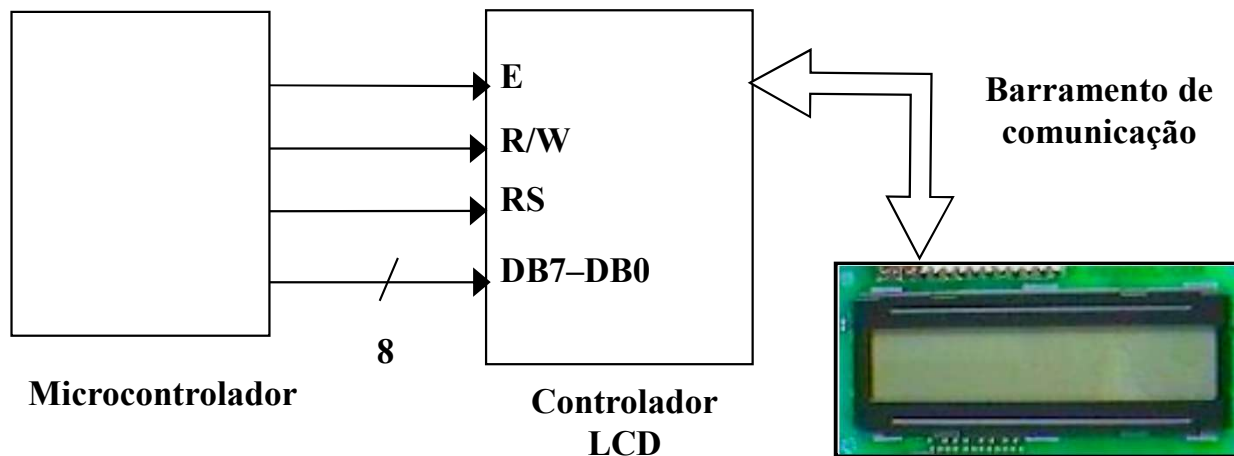
# Controlador LCD

- ***Liquid Crystal Display:***

- Um mostrador de cristal líquido é um dispositivo de baixo custo e baixa potência capaz de exibir texto e imagens.
- LCD refletivo:
  - A luz incidente passa através de uma placa de polarização. Então, a luz polarizada encontra um material cristal líquido que, quando excitado, tem suas moléculas alinhadas, o que faz com que a luz polarizada o atravesse. Senão, a luz não passa.
  - Por fim, a luz que passou atinge uma superfície refletora, de modo que estas regiões excitadas acendem.
- Controlador LCD: provê uma interface simples para o uso de LCDs.

# Controlador LCD

Recebe palavras de controle do microcontrolador e realiza as ações correspondentes no LCD.



RS: quando em nível BAIXO, informa ao controlador que os dados enviados (DB7 – DB0) formam uma palavra de controle.

Toda vez que um dado precisa ser enviado, o bit de *enable* (E) deve ser acionado.

# Controlador LCD

- **Exemplo:** controlador LCD

RS	R/W	DB <sub>7</sub>	DB <sub>6</sub>	DB <sub>5</sub>	DB <sub>4</sub>	DB <sub>3</sub>	DB <sub>2</sub>	DB <sub>1</sub>	DB <sub>0</sub>	Description
0	0	0	0	0	0	0	0	0	1	Clears all display, return cursor home
0	0	0	0	0	0	0	0	1	*	Returns cursor home
0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and/or specifies not to shift display
0	0	0	0	0	0	1	D	C	B	ON/OFF of all display(D), cursor ON/OFF (C), and blink position (B)
0	0	0	0	0	1	S/C	R/L	*	*	Move cursor and shifts display
0	0	0	0	1	DL	N	F	*	*	Sets interface data length, number of display lines, and character font
1	0	WRITE DATA							Writes Data	

CODES	
I/D = 1 cursor moves left	DL = 1 8-bit
I/D = 0 cursor moves right	DL = 0 4-bit
S = 1 with display shift	N = 1 2 rows
S/C = 1 display shift	N = 0 1 row
S/C = 0 cursor movement	F = 1 5x10 dots
R/L = 1 shift to right	F = 0 5x7 dots
R/L = 0 shift to left	

Rotina que envia um caractere ao controlador para escrita no display

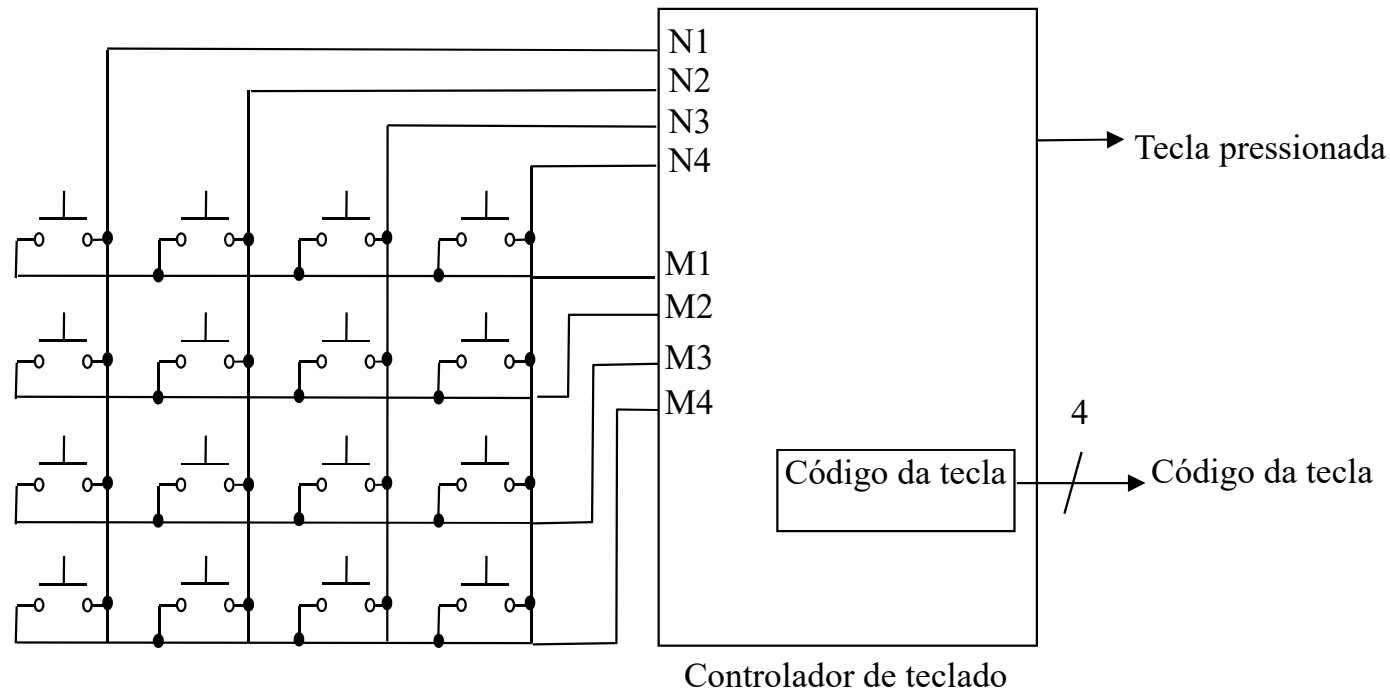
```
void WriteChar(char c){

    RS = 1;                /* indicate data being sent */
    DATA_BUS = c;         /* send data to LCD */
    EnableLCD(45);         /* toggle the LCD with appropriate delay */
}
```

# Controlador de teclado

# Controlador de teclado

- Oferece uma interface simples para leitura de caracteres de entrada em um teclado.
- O sinal “tecla pressionada” pode disparar uma interrupção, de modo que o processador genérico realiza a leitura do código da tecla.

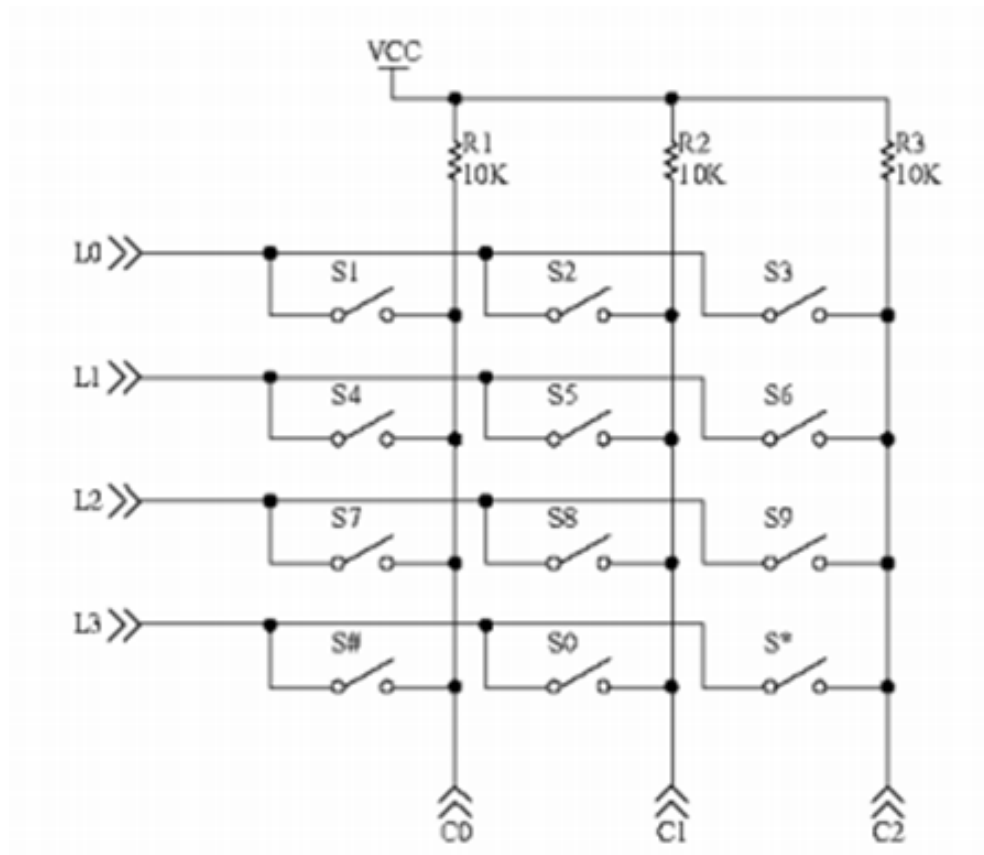


N=4, M=4



# Controlador de teclado

- **Exemplo:**



➤ As colunas (C0-C2) ficam em nível lógico alto enquanto nenhuma tecla é acionada.

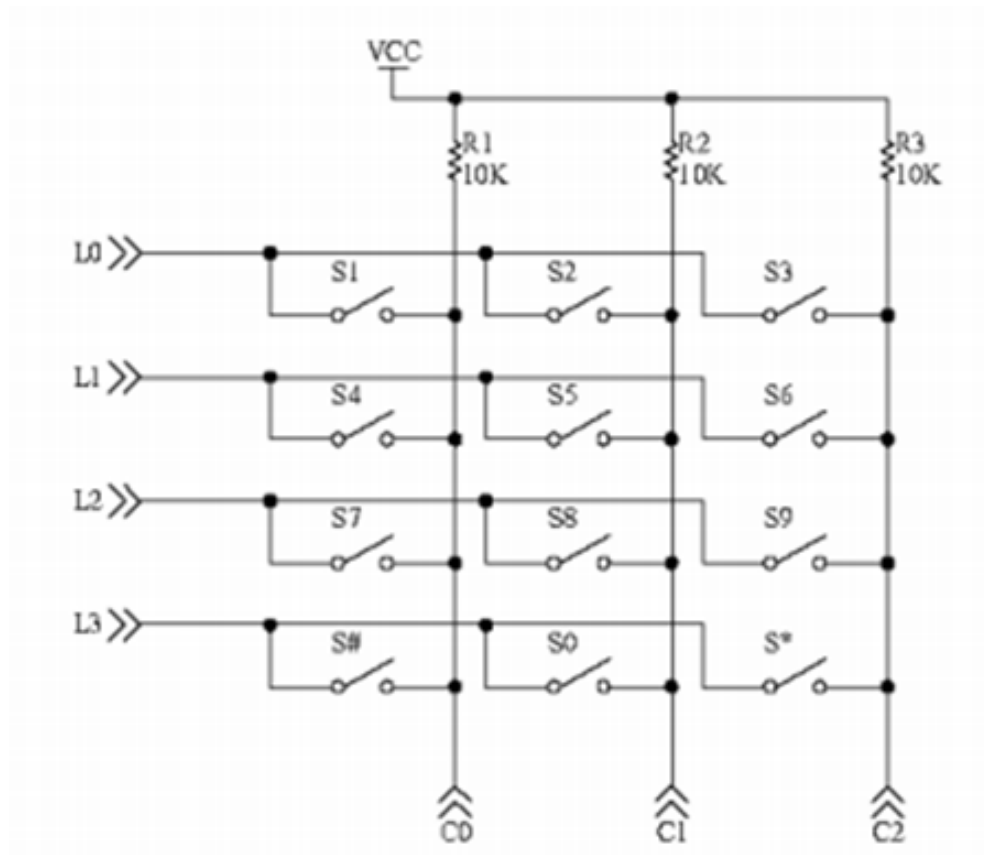
➤ Para identificar uma tecla acionada, faz-se uma varredura da matriz do teclado.

- Primeiramente, coloca-se as linhas em nível lógico baixo e verifica-se qual coluna possui também nível lógico baixo.

- Em seguida, atribui-se nível lógico alto de forma sequencial nas linhas, verificando qual linha produz transição de nível da coluna previamente identificada.

# Controlador de teclado

- Exemplo:



## Registrador (posição de memória):

**Bits:** B7 B6 B5 B4 B3 B2 B1 B0

**Significado:** CI EI X X L3 L2 L1 L0

**Bits:** B7 B6 B5 B4 B3 B2 B1 B0

**Significado:** FI X X X X C2 C1 C0

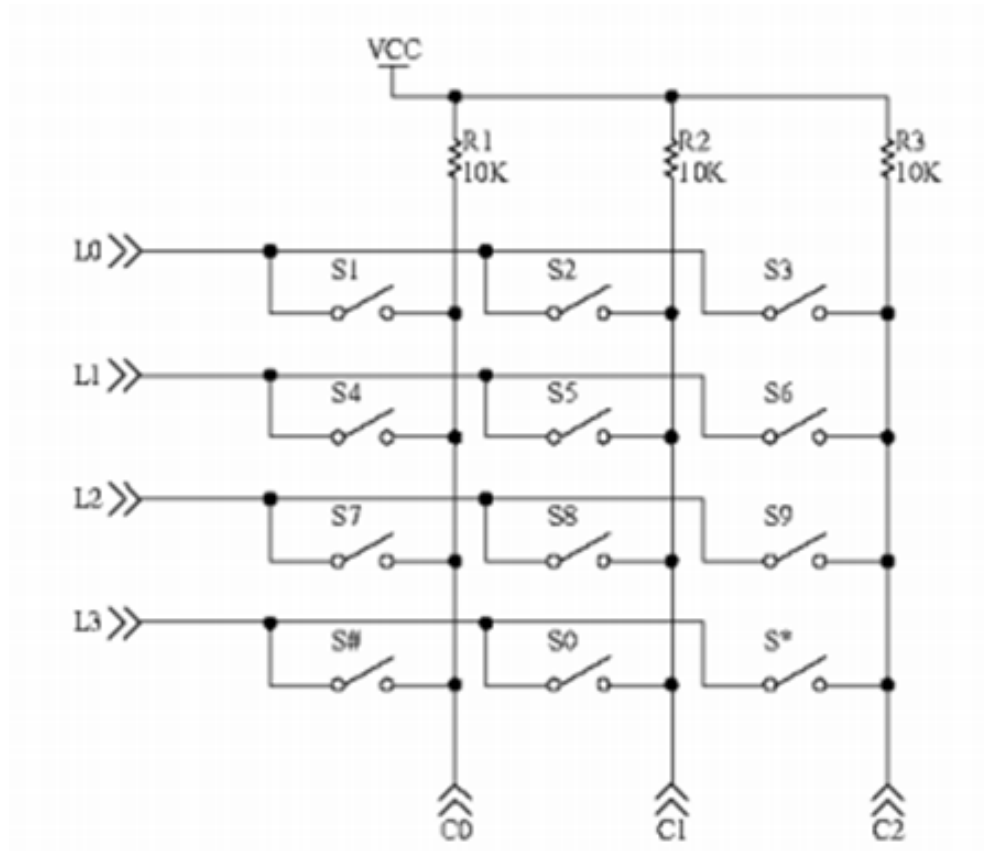
EI – habilita interrupção.

CI – limpa o pedido de interrupção.

FI – indica solicitação de interrupção.

# Controlador de teclado

- **Exemplo:**



- A interface com o teclado pode ser feita mesclando *hardware* e *software*.
- Em hardware,
  - Pode-se eliminar o efeito de bounce usando latches D.
  - Conecta-se a pinos específicos do microprocessador as linhas (entradas do teclado) e colunas (saídas do teclado).
- Em software, faz-se a varredura segundo o procedimento anteriormente mencionado para identificar a tecla pressionada quando uma interrupção do teclado é gerada.

# Controlador de teclado

- Exemplo:

```
/*rotina que identifica qual tecla foi pressionada*/
unsigned char check_key(){

    unsigned char value, tecla;
    value = *key;
    value = value & 0x07;
    if ((value = value & 0x07) != 0x07){ /*se alguma tecla foi apertada*/
        if(value == 0x06){ /*se a tecla esta na 1a coluna*/
            *key = 0x01; /*verifica 1a linha*/
            if((*key & 0x07) == 0x07)
                tecla = '1';
            else{
                *key = 0x02; /*verifica 2a linha*/
                if((*key & 0x07) == 0x07)
                    tecla = '4';
                else{
                    *key = 0x04; /*verifica 3a linha*/
                    if((*key & 0x07) == 0x07)
                        tecla = '7';
                    else { /*4a linha*/
                        tecla = '#';
                    }
                }
            }
        }
    }
}
```

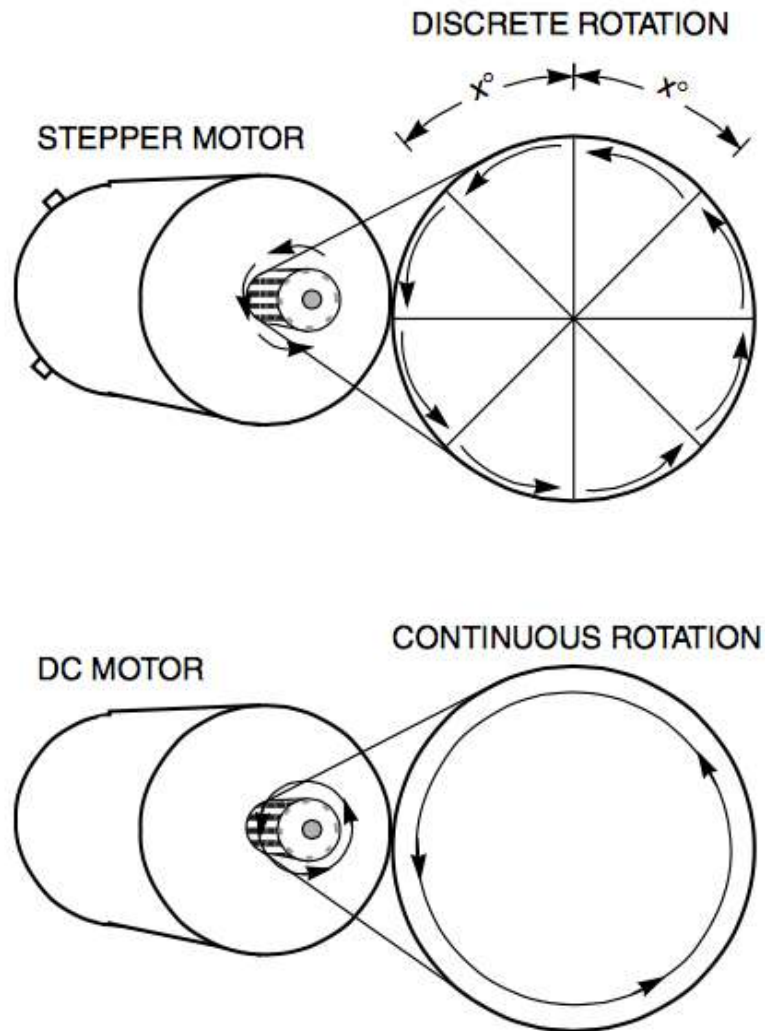
# Stepper Motor

Motor de passos

# Motor de Passo

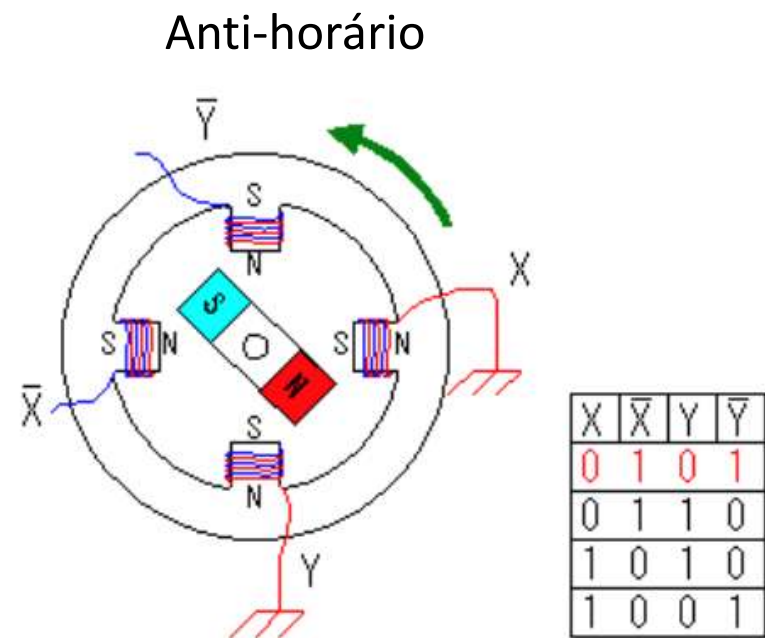
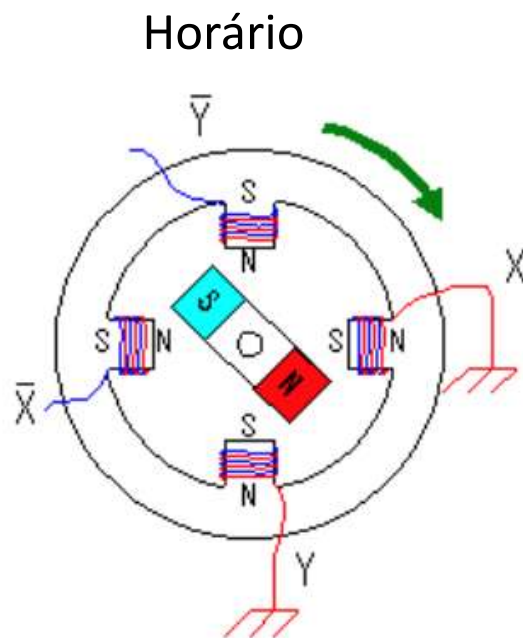
- Motor de Passo: rotaciona um número fixo de graus quando um “sinal de passo” é aplicado
  - Especificado em termos de quantos graus (ex: 1.8) o motor avança a cada passo ou
  - Especificado em termos de quantos passos são necessários para o motor completar 360 graus (1 volta completa)
- Exemplos de uso: discos, impressoras, máquinas de fax e de fotocópia, VCRs, etc.

# Motor de Passo x Motor DC



# Motor de Passo

- A rotação do eixo do motor ocorre quando uma determinada seqüência de tensões é aplicada às bobinas





# Controlando o Motor de Passo

- Tabela que define a sequência de operação para girar o motor de 1 passo (x graus).
- A sequência inteira precisa ser aplicada para que ocorra a rotação do motor (1 passo).
- Para controlar o motor através de software, precisamos manter esta tabela em software
- Além disso, precisamos de uma rotina que aplica valores “1”s às entradas A, B, A’ e B’ baseada nos valores da tabela seguindo a sequência correta;
- O controle é facilitado com o uso de um microcontrolador

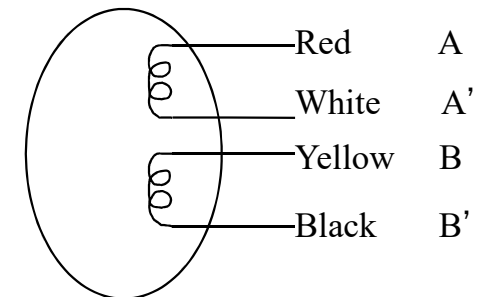
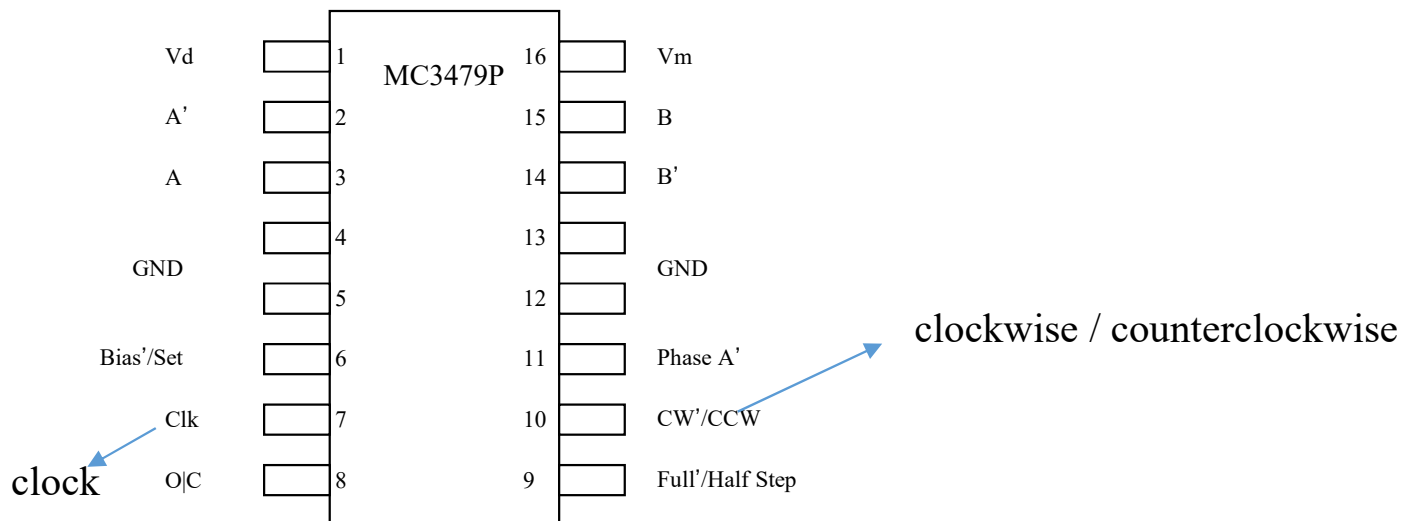
Sequence	A	B	A'	B'
1	+	+	-	-
2	-	+	+	-
3	-	-	+	+
4	+	-	-	+
5	+	+	-	-

*Slide gentilmente cedido pelo Prof. Léo*

# Controlador (*Driver*) de Motor de Passo

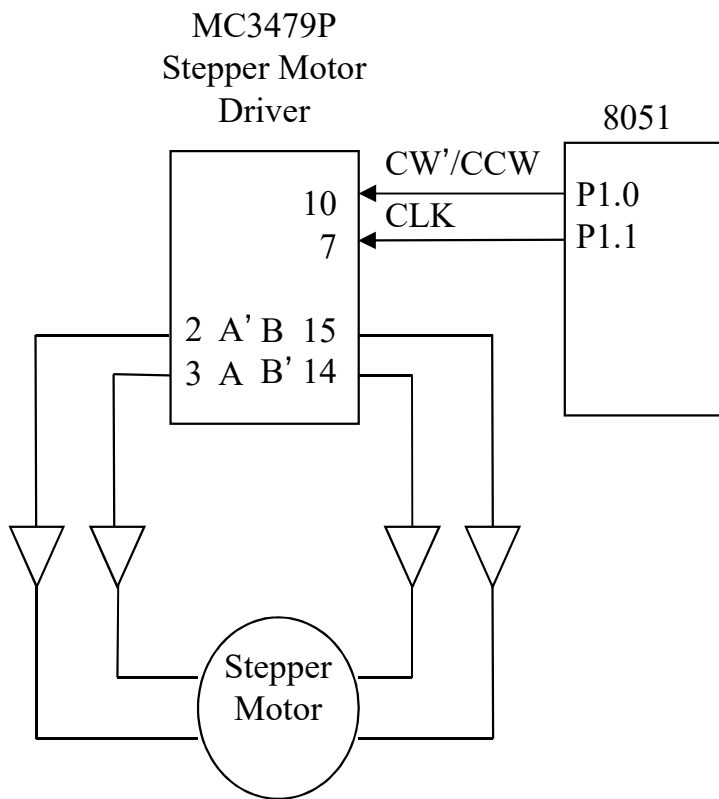
- Neste exemplo temos um motor de 9V, bipolar de 2 fases.
- Tabela que define a sequência de operação para girar o motor de 7,5 graus.
- Para girar no outro sentido, basta aplicar a sequência ao contrário

Sequence	A	B	A'	B'
1	+	+	-	-
2	-	+	+	-
3	-	-	+	+
4	+	-	-	+
5	+	+	-	-



*Slide gentilmente cedido pelo Prof. Léo*

# Motor de Passo **com** Controlador



```
/* main.c */

sbit clk=P1^1; /* ver(*) */
sbit cw=P1^0;

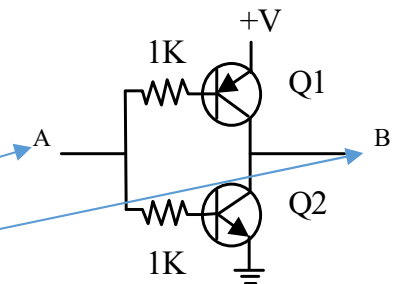
void delay(void){
    int i, j;
    for (i=0; i<1000; i++)
        for ( j=0; j<50; j++)
            i = i + 0;
}
```

```
void main(void){
    /*turn the motor forward */
    cw=0;          /* set direction */
    clk=0;         /* pulse clock */
    delay ( );
    clk=1;

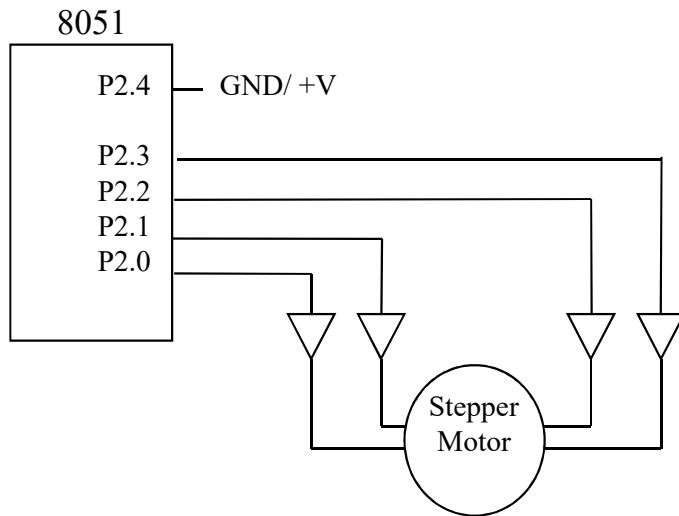
    /*turn the motor backwards */
    cw=1;          /* set direction */
    clk=0;         /* pulse clock */
    delay ( );
    clk=1;
}
```

(\*) compilador C51: [http://www.keil.com/support/man/docs/c51/c51\\_le\\_sbit.htm](http://www.keil.com/support/man/docs/c51/c51_le_sbit.htm)

Os pinos de saída do *driver* do motor de passo não têm a capacidade de gerar a corrente necessária para controlar a rotação do motor. Para amplificar a corrente, um buffer é adicionado ao sistema. Uma possibilidade de implementação do *buffer* é esquematizada na figura ao lado. Q1 é um transistor NPN MJE3055T e Q2 é um transistor PNP MJE2955T. O sinal **A** é conectado ao microcontrolador 8051 e o sinal **B** é conectado ao motor de passo.



# Motor de Passo **sem** Controlador



```
/*main.c*/
sbit notA=P2^0; /* ver(*) tr. 23 */
sbit isA=P2^1;
sbit notB=P2^2;
sbit isB=P2^3;
sbit dir=P2^4;

void delay(){
    int a, b;
    for(a=0; a<5000; a++)
        for(b=0; b<10000; b++)
            a=a+0;
}

void move(int dir, int steps) {
    int y, z;
    /* clockwise movement */
    if(dir == 1){
        for(y=0; y<=steps; y++){
            for(z=0; z<=19; z+=4){
                isA=lookup[z];
                isB=lookup[z+1];
                notA=lookup[z+2];
                notB=lookup[z+3];
                delay();
            }
        }
    }
}
```

```
/* counter clockwise movement */
if(dir==0){
    for(y=0; y<=steps; y++){
        for(z=19; z>=0; z-=4){
            isA=lookup[z];
            isB=lookup[z-1];
            notA=lookup[z-2];
            notB=lookup[z-3];
            delay();
        }
    }
}

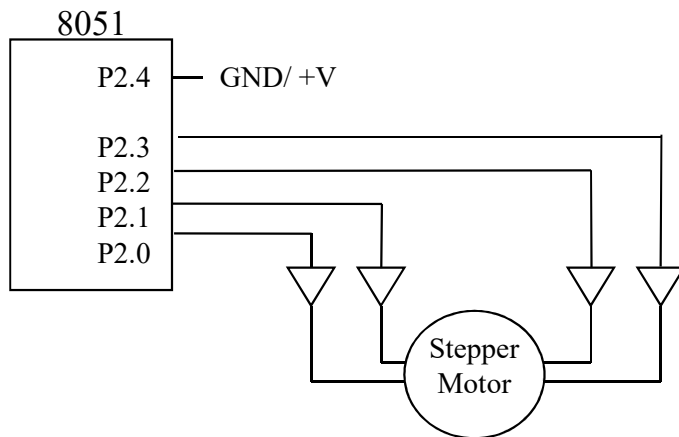
void main() {
    int z;
    int lookup[20] = {
        1, 1, 0, 0,
        0, 1, 1, 0,
        0, 0, 1, 1,
        1, 0, 0, 1,
        1, 1, 0, 0 };
    while(1){
        /*move forward, 15 degrees (2 steps) */
        move(1, 2);
        /* move backwards, 7.5 degrees (1step)*/
        move(0, 1);
    }
}
```

*Slide gentilmente cedido pelo Prof. Léo*

# Motor de passos

## • Exemplo: Sem driver

- A sequência que define um passo tem que ser gerada via software.



```
/*main.c*/
sbit notA=P2^0;
sbit isA=P2^1;
sbit notB=P2^2;
sbit isB=P2^3;
sbit dir=P2^4;

void delay(){
    int a, b;
    for(a=0; a<5000; a++)
        for(b=0; b<10000; b++)
            a=a+0;
}

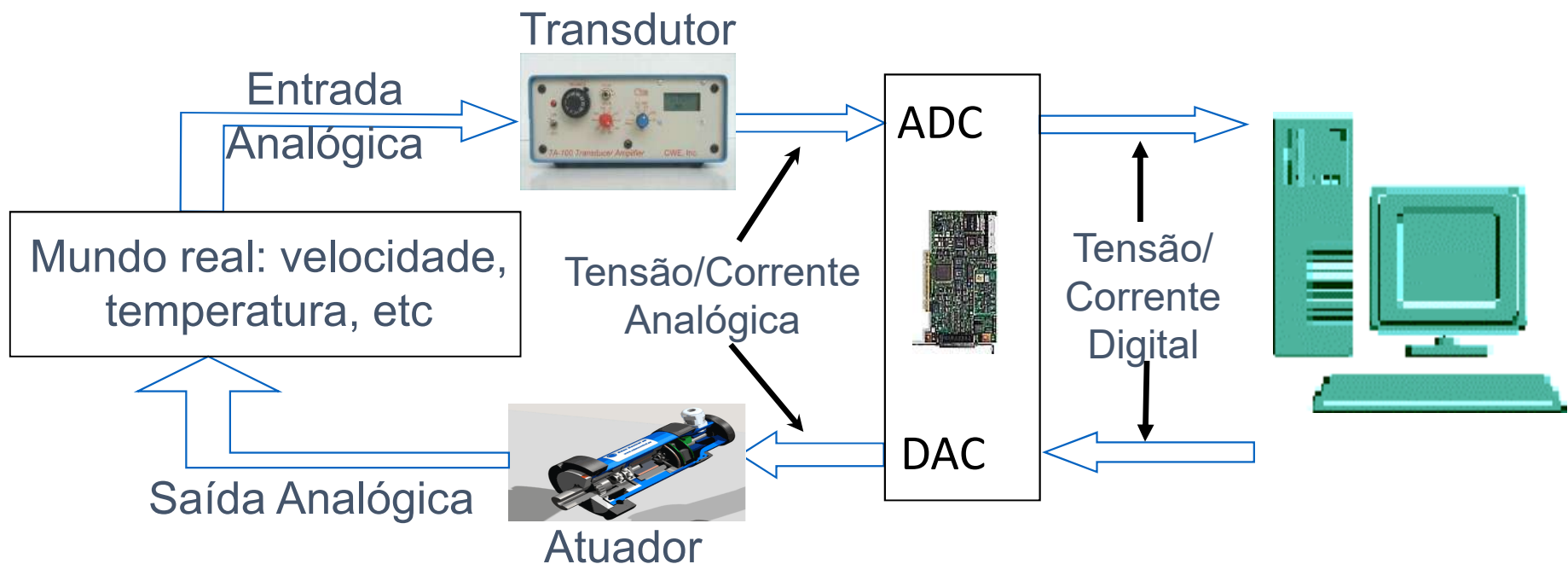
void move(int dir, int steps) {
    int y, z;
    /* clockwise movement */
    if(dir == 1){
        for(y=0; y<=steps; y++){
            for(z=0; z<=19; z+4){
                isA=lookup[z];
                isB=lookup[z+1];
                notA=lookup[z+2];
                notB=lookup[z+3];
                delay();
            }
        }
    }
}
```

```
/* counter clockwise movement */
if(dir==0){
    for(y=0; y<=steps; y++){
        for(z=19; z>=0; z - 4){
            isA=lookup[z];
            isB=lookup[z-1];
            notA=lookup[z -2];
            notB=lookup[z-3];
            delay( );
        }
    }
}

void main() {
    int z;
    int lookup[20] = {
        1, 1, 0, 0,
        0, 1, 1, 0,
        0, 0, 1, 1,
        1, 0, 0, 1,
        1, 1, 0, 0 };
    while(1){
        /*move forward, 15 degrees (2 steps) */
        move(1, 2);
        /* move backwards, 7.5 degrees (1step)*/
        move(0, 1);
    }
}
```

# Conversor A/D e D/A

# Arquitetura: Aquisição de dados



- ❑ **Transdutor: Converte sinais de entrada não-elétricos em sinais elétricos analógicos (proporcionais)**
- ❑ **Atuador: Converte sinais elétricos em quantidades físicas (proporcionais)**

*Slide from Prof. Yann-Hang Lee*

# Arquitetura: Aquisição de dados

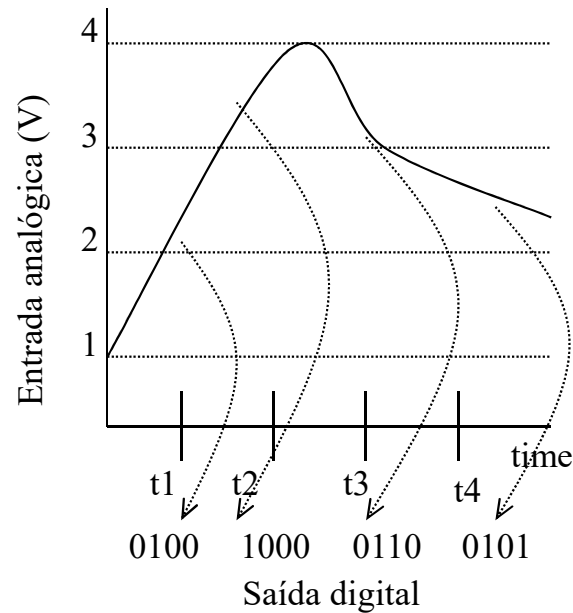
- **Conversor A/D (ADC)** é usado para converter sinais elétricos analógicos em valores digitais proporcionais. Os dados digitais podem ser processados, gravados e visualizados em um computador ou dispositivo digital.
- **Conversor D/A (DAC)** é usado para converter um valor representado em código digital em um sinal analógico de tensão/corrente. Este sinal pode ser então usado para controlar um dispositivo do mundo real através de um atuador.
- Exemplo: Controle de velocidade em um carro
  - Velocímetro => Transdutor => A/D => Microcontrolador
  - Microcontrolador => D/A => Atuador => Acelerador



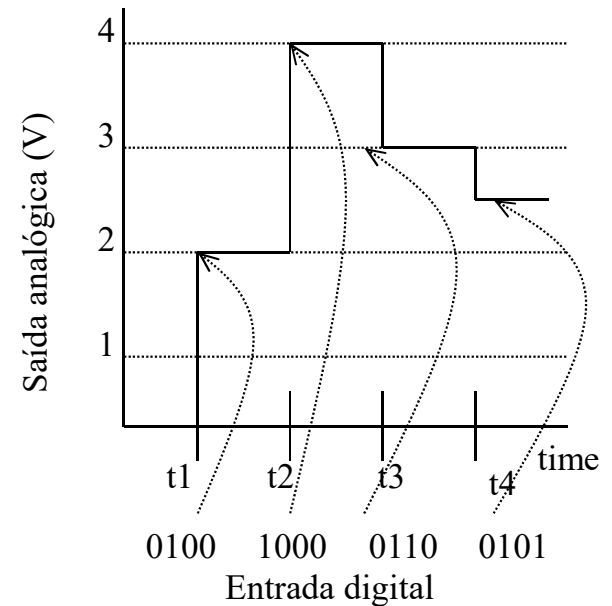
# Conversores A/D e D/A

$V_{\max} = 7.5V$	1111
7.0V	1110
6.5V	1101
6.0V	1100
5.5V	1011
5.0V	1010
4.5V	1001
4.0V	1000
3.5V	0111
3.0V	0110
2.5V	0101
2.0V	0100
1.5V	0011
1.0V	0010
0.5V	0001
0V	0000

**Proporcionalidade**



**Analogico para Digital**



**Digital para analógico**

*Slide gentilmente cedido pelo Prof. Léo*

# Conversores D/A

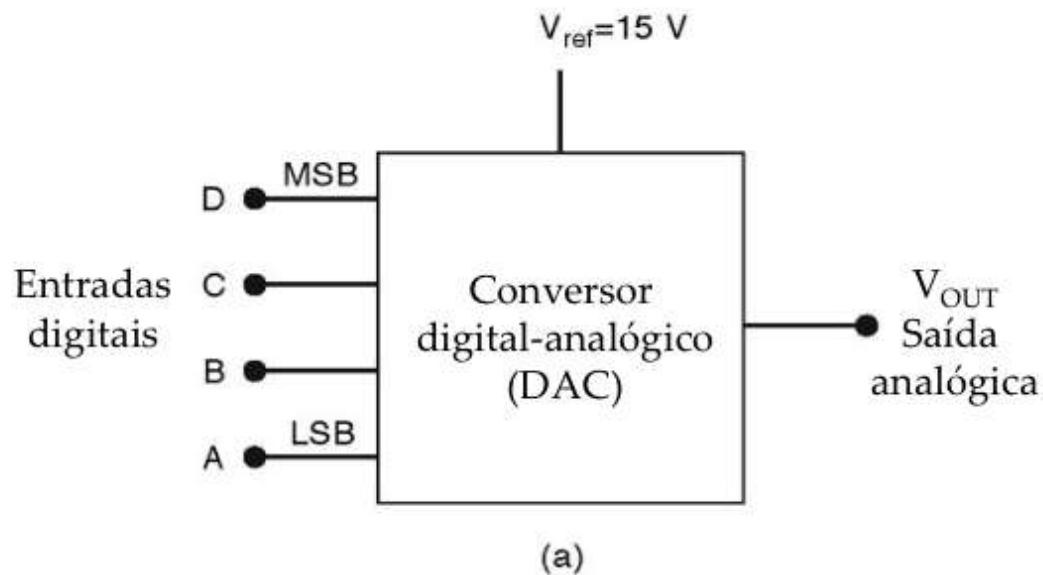
- Converte um sinal digital (código binário) em uma tensão ou corrente proporcional

$$e = k.d$$

Onde

- $e$  – valor analógico correspondente a  $d$
- $d$  – valor digital de entrada
- $k$  – fator de proporcionalidade

# Conversores D/A



D	C	B	A	$V_{OUT}$	
0	0	0	0	0	volts
0	0	0	1	1	↓
0	0	1	0	2	
0	0	1	1	3	
0	1	0	0	4	
0	1	0	1	5	
0	1	1	0	6	
0	1	1	1	7	
1	0	0	0	8	↓
1	0	0	1	9	
1	0	1	0	10	
1	0	1	1	11	
1	1	0	0	12	
1	1	0	1	13	
1	1	1	0	14	
1	1	1	1	15	volts

(b)

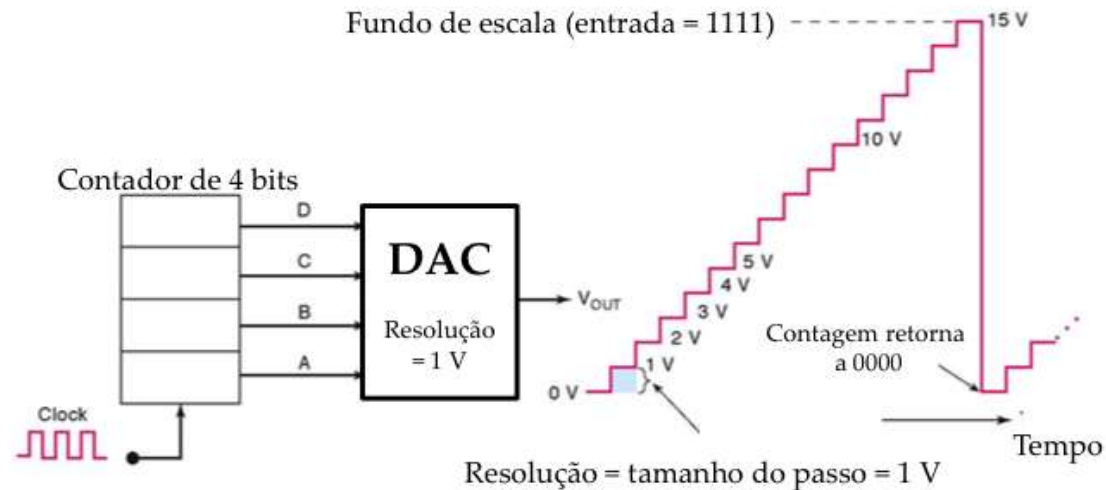
- $K = 1\text{ V}$  e  $V_{OUT} = (1\text{ V}) \times \text{entrada digital}$ .

# Conversores D/A

- Cada bit do sinal digital contribui de forma diferente para o valor analógico
  - o bit menos significativo vale  $k \cdot 2^0$  V
  - o segundo bit menos significativo vale  $k \cdot 2^1$  V
  - o terceiro bit menos significativo vale  $k \cdot 2^2$  V

# Conversor D/A: resolução e faixa

- **Faixa:** A diferença entre a máxima e a mínima tensão (ou corrente) que está sendo convertida.
- **Resolução:** Valor mínimo que pode ser representado pelo conversor. O número de bits do conversor determina a resolução.



# Conversores D/A

$$(e - V_{\min}) / (V_{\max} - V_{\min}) = d / (2^n - 1)$$

Onde

- $e$  – valor analógico
- $V_{\max}$  – valor analógico máximo
- $d$  – valor digital correspondente a  $e$
- $n$  – número de bits
- Resolução =  $(V_{\max} - V_{\min}) / (2^n - 1)$

# Conversor D/A: resolução e faixa

- Para uma faixa de 10 volts:
  - 3-bits => resolução =  $10/(2^3-1)=1.43\text{volts}$
  - 12-bits => resolução =  $10/(2^{12}-1)=2.44\text{ mV}$

# Conversor A/D

- Converte uma tensão analógica em um código binário
- “Rampa digital”, “Aproximação sucessiva” e “Flash” são as arquiteturas mais populares.

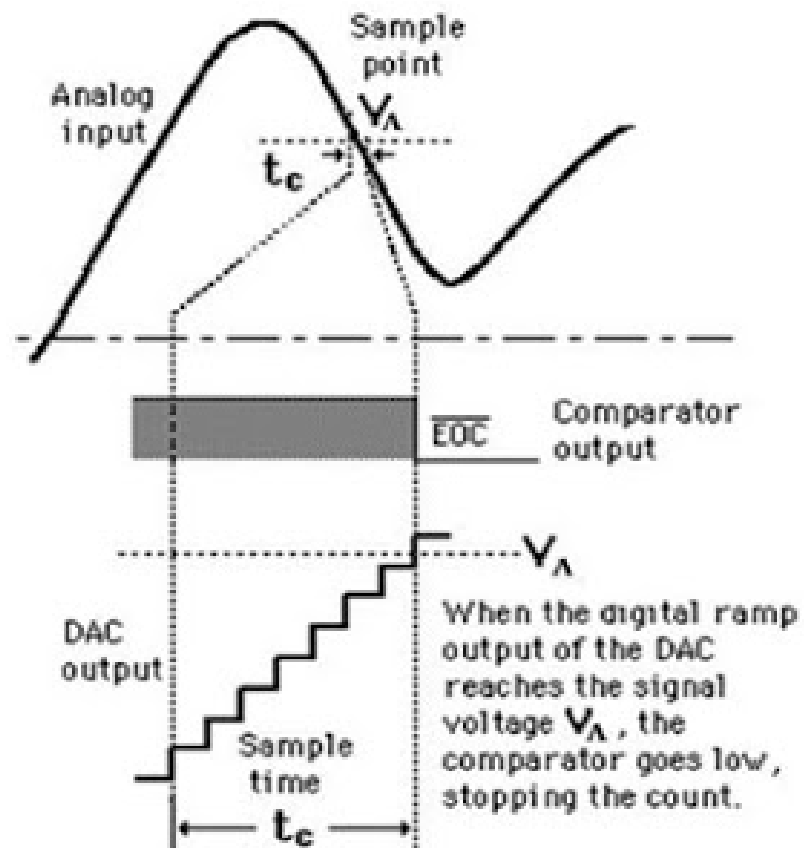
Low Speed, High Accuracy	Medium Speed, Medium Accuracy	High speed, Low-to-medium Accuracy
<i>Integrating Oversampling</i>	<i>Successive Approximation, Algorithmic</i>	<i>Flash, Two-step, Interpolating, Folding, Pipelined, Time-interleaved</i>

*Slide from Prof. Yann-Hang Lee*

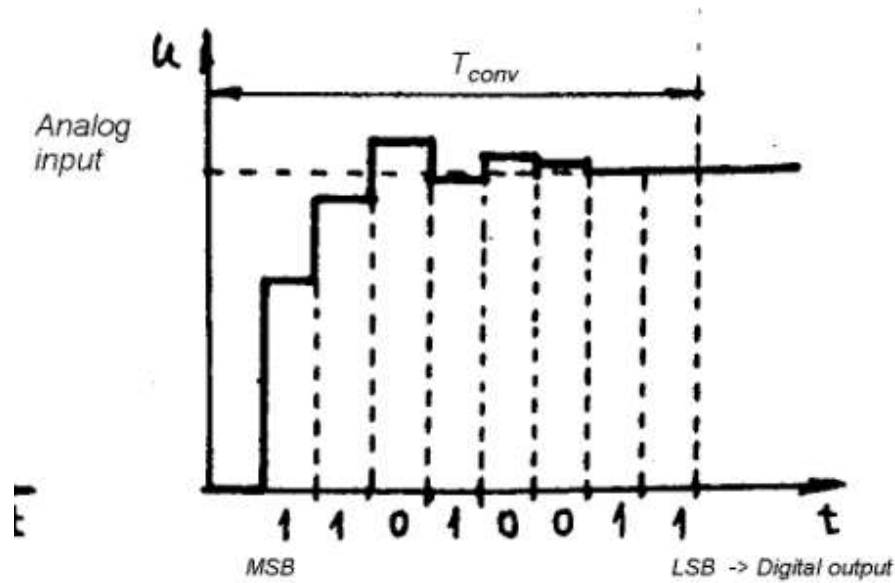


# Conversores A/D: Rampa

- Um contador é incrementado a cada pulso do clock, e o código binário é convertido em uma tensão analógica ( $V_{AX}$ )
- Enquanto  $V_{AX} < V_A$ , o contador continua sendo incrementado



# Aproximação sucessiva



- Começa comparando a entrada analógica com o meio da escala, ou seja, bit mais significativo (MSB) = 1. Se a entrada analógica é maior, MSB permanece intacto e no próximo ciclo o próximo bit é setado. Se a entrada analógica é menor, MSB é trocado por zero e no próximo ciclo o próximo bit é setado.

# Conversão A/D por Aproximações Sucessivas

Dado um sinal de entrada analógico cuja tensão pode variar dentro do intervalo de 0 a 15V e um conversor digital de 8-bit, calcule a correta codificação para um sinal em 5V. Em seguida, aplique o método das aproximações sucessivas para encontrar essa codificação.

$$5/15 = d/(2^8-1)$$
$$d = 85$$

Encoding: 01010101

## *Método de Aproximação Sucessiva*

$$\frac{1}{2}(V_{\max} - V_{\min}) = 7.5 \text{ volts}$$
$$V_{\max} = 7.5 \text{ volts.}$$

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

$$\frac{1}{2}(5.63 + 4.69) = 5.16 \text{ volts}$$
$$V_{\max} = 5.16 \text{ volts.}$$

0	1	0	1	0	0	0	0
---	---	---	---	---	---	---	---

$$\frac{1}{2}(7.5 + 0) = 3.75 \text{ volts}$$
$$V_{\min} = 3.75 \text{ volts.}$$

0	1	0	0	0	0	0	0
---	---	---	---	---	---	---	---

$$\frac{1}{2}(5.16 + 4.69) = 4.93 \text{ volts}$$
$$V_{\min} = 4.93 \text{ volts.}$$

0	1	0	1	0	1	0	0
---	---	---	---	---	---	---	---

$$\frac{1}{2}(7.5 + 3.75) = 5.63 \text{ volts}$$
$$V_{\max} = 5.63 \text{ volts}$$

0	1	0	0	0	0	0	0
---	---	---	---	---	---	---	---

$$\frac{1}{2}(5.16 + 4.93) = 5.05 \text{ volts}$$
$$V_{\max} = 5.05 \text{ volts.}$$

0	1	0	1	0	1	0	0
---	---	---	---	---	---	---	---

$$\frac{1}{2}(5.63 + 3.75) = 4.69 \text{ volts}$$
$$V_{\min} = 4.69 \text{ volts.}$$

0	1	0	1	0	0	0	0
---	---	---	---	---	---	---	---

$$\frac{1}{2}(5.05 + 4.93) = 4.99 \text{ volts}$$

0	1	0	1	0	1	0	1
---	---	---	---	---	---	---	---

*Slide gentilmente cedido pelo Prof. Léo*

# Créditos

## Fontes: Material de oferecimentos anteriores:

- Profa. Leticia
- Prof. Levy
- Prof. Leo Pini
- Profa. Alice
- Prof. Cristiano Araújo, CIN / UFPE
- ...
- Algumas figuras foram extraídas de F. Vahid e T. Givargis, *“Embedded System Design: A Unified Hardware/Software Introduction”*, John Wiley & Sons, 2001.



# BACKUP