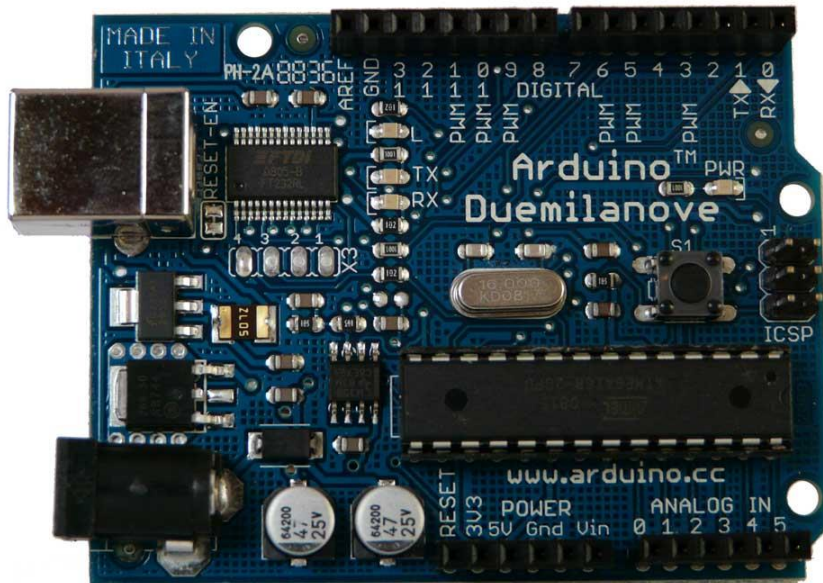




Computação Física



Jorge Pais
2015-2016

Computação Física

1. Introdução

Computação física é uma área que envolve sensores e atuadores controlados por um microprocessador que define o comportamento do sistema por programação permitindo uma interação com o mundo ambiente e com os humanos .

No passado, o desenho de sistemas controlados por microprocessador exigiam um curso intensivo em microprocessadores e sistemas, dado a sua complexidade. Hoje em dia, os microprocessadores além de serem circuitos baratos tornaram-se mais fáceis de utilizar devido ao desenvolvimento de ferramentas melhores e de mais alto nível.

A arquitetura Arduino é um exemplo da facilidade e da acessibilidade a pessoas interessadas na aprendizagem deste tipo de matéria, pois em poucas horas de ensino e aprendizagem torna-se possível o desenvolvimento de sistemas interativos envolvendo a arquitectura.

O desenho de sistemas interativos está presente em muitos objetos que utilizamos no nosso dia a dia, desde o telemóvel, consolas de jogos, computadores, televisão e electrodomésticos inteligentes.

O universo de sistemas interativos é uma experiência em desenvolvimento que permite uma experiência entre o ser humano e a tecnologia muito aliciante e em constante mutação e evolução.

O propósito principal desta disciplina é dotar o aluno de conhecimento e de autonomia para o desenvolvimento de sistemas interativos comportamentais de média e alta complexidade.

Computação Física

Capítulo 1 – Circuitos Combinatórios

1. Base de numeração binária

Uma representação digital de um número é a representação do número em base binária.

A base binária é uma base de numeração com dois algarismos, os algarismos 0 e 1. Estes algarismos são conhecidos como dígitos binários ou algarismos binários, usualmente designados como **bit** que é a abreviatura do termo anglo-saxónico *binary digit*. Em termos físicos faz-se corresponder ao algarismo 0 uma gama de tensão e ao algarismo 1 outra gama distinta de tensão. Por exemplo em tecnologia TTL, o valor 0 corresponde a uma gama de tensões entre [0, 0.8]V e ao valor 1 corresponde uma gama de tensões [2.5, 5]V.

Idealmente e para efeitos de dimensionamento de circuitos, considere o valor binário 0 como um valor de tensão próximo da tensão de referência do circuito que é designada por GND ou 0V e o valor binário 1 como um valor de tensão próximo da tensão de alimentação que é usualmente designada por Vcc.

Um número binário é caracterizado pela quantidade de algarismos binários(bits) que o compõem e pela sua interpretação ser em código absoluto(valores positivos) ou em código de complementos(valores positivos e negativos).

A quantidade de bits dum número define a quantidade de valores diferentes de representação do número. Um número em base 2 e com n bits pode representar 2^n valores diferentes.

A interpretação de um número binário a n bits em código absoluto define a representação da gama de valores $[0, 2^{n-1}]$.

A interpretação em código de complementos ou código binário relativo de um número com n bits define a gama de valores $[-2^{n-1}, +2^{n-1}-1]$. A definição de um valor ser negativo ou positivo é definido pelo algarismo mais significativo do número(MSD –Most Significant Digit), se o MSD for 1 o número é considerado negativo e se for 0 o número é considerado positivo. De notar que na representação digital o valor 0 é considerado um número positivo.

Computação Física

2. Funções lógicas base

As funções definidas na matemática que são a interseção, a união e a negação de conjuntos deram origem às funções lógicas base na área dos sistemas digitais que são o AND (e – em português), o OR (ou – em português) e o NOT (não – em português).

2.1. Função lógica NOT

A função lógica NOT, em português diz-se “não”, é a função lógica negação que só tem uma variável independente de entrada e pode ser definida pela tabela de verdade apresentada na figura 2.1.1. Uma tabela de verdade é constituída por duas partes, uma parte onde se definem as variáveis independentes (a, b, c, ...) que podem ter qualquer combinação de entrada em termos dos seus valores digitais (0 ou 1) e a outra parte que são as funções dependentes definidas como uma função lógica das variáveis independentes, $F = f(a, b, c, d, \dots)$, em que F é a função dependente, $f(\dots)$ é a função lógica e (a, b, c, ...) são as variáveis independentes.

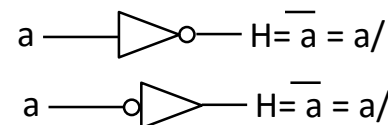
variável independente	função dependente
a	$H = \overline{a} = a/$
0	1
1	0

Figura 2.1.1 – Tabela de verdade da função NOT.

A função lógica NOT tem como operadores os caracteres “tracinho” por cima da variável independente ou o carácter “backslash” a seguir: $\overline{}$ ou $/$

Se quiser escrever “H é igual à função não a” em notação lógica fica $H = \overline{a} = a/$

Os símbolos lógicos da função NOT são:



Definição: A função lógica NOT é 1 ou verdadeira quando a variável independente é 0 ou falsa.

Propriedades: $\overline{\overline{A}} = A$; teorema da involução
 $\overline{\overline{\overline{A}}} = \overline{A}$

Computação Física

2.2. Função lógica AND

A função lógica AND, em português diz-se “e”, a 2 variáveis independentes pode ser definida pela tabela de verdade apresentada na figura 2.2.1.

variáveis independentes		função dependente
a	b	$F = a.b$
0	0	0
0	1	0
1	0	0
1	1	1

A função AND tem como operador o carácter ponto : \cdot .
Se quiser escrever “F é igual à função a e b” em notação lógica é $F = a.b$

Os símbolos lógicos da função AND são:

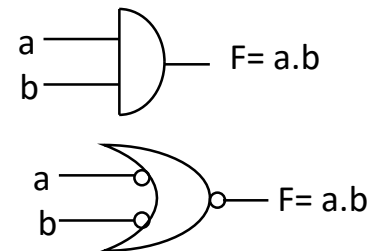


Figura 2.2.1 – Tabela de verdade da função AND.

Definição: A função AND é 1, quando **todas** as variáveis independentes são 1.

Computação Física

Propriedades da função lógica AND:

$a.0 = 0$, o 0 é o elemento absorvente;
$a.1 = a$, o 1 é o elemento neutro;
$a.a = a$, teorema da idempotência,
$a.a/ = 0$, teorema da complementação;
$a.b.c = (a.b).c = a.(b.c)$, propriedade associativa;
$a.b = b.a$, propriedade comutativa;

$(a.b)/ = a/ + b/$, Leis de De Morgan

$a.b = (a.b)// = (a/ + b/)/$

$(a.b.c.d.e....)/ = a/ + b/ + c/ + d/ + e/ +$

Computação Física

2.3. Função lógica OR

A função lógica OR, em português diz-se “ou”, a 2 variáveis independentes pode ser definida pela tabela de verdade apresentada na figura 2.3.1.

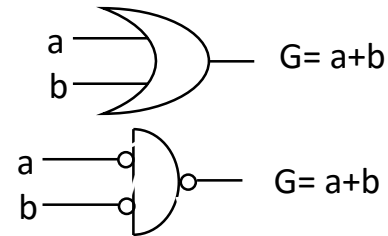
variáveis independentes		função dependente
a	b	$G = a + b$
0	0	0
0	1	1
1	0	1
1	1	1

Figura 2.3.1 – Tabela de verdade da função OR .

A função OR ou “ou” tem como operador o carater “mais” : $+$

Se quiser escrever “G é igual à função a ou b” em notação lógica é $F = a + b$

A representação da porta lógica da função OR é:



Definição: A função lógica OR é 1, quando **pelo menos uma** das variáveis independentes é 1.

Computação Física

Propriedades da função lógica OR:

$a+0 = a$, o 0 é o elemento neutro;
 $a+1 = 1$, o 1 é o elemento absorvente;
 $a+a = a$, teorema da idempotência,
 $a+a/ = 1$, teorema da complementação;
 $a+b+c = (a+b)+c = a+(b+c)$, propriedade associativa;
 $a+b = b+a$, propriedade comutativa;
 $(a+b)/ = a/ \cdot b/$, Leis de De Morgan
 $a+b = (a+b)// = (a/ \cdot b/)/$
 $(a+b+c+d+e+...)/ = a/ \cdot b/ \cdot c/ \cdot d/ \cdot e/ \cdot$

Propriedades da composição de funções AND-OR:

$a+b \cdot c = (a+b) \cdot (a+c)$, propriedade distributiva,
 $a + a \cdot b = a$, teorema da absorção,
 $a + a/ \cdot b = a + b$, teorema da complementação,
 $a \cdot b + a \cdot c = a \cdot (b+c)$, inversa da propriedade distributiva.

Propriedades da composição de funções OR-AND:

$a \cdot (b+c) = a \cdot b + a \cdot c$, propriedade distributiva,
 $a \cdot (a+b) = a$, teorema da absorção,
 $a \cdot (a/+b) = a \cdot b$, teorema da complementação,
 $(a+b) \cdot (a+c) = a + (b \cdot c)$, inversa da propriedade distributiva.

Computação Física

2.4. Mapas de Karnaugh

O mapa de Karnaugh é uma outra técnica de representação de funções lógicas alternativa à tabela de verdade tendo a vantagem de ser um técnica completamente gráfica. O universo é representado por uma forma retangular e este universo é dividido em tantas quadriculas quantas as combinações possíveis das variáveis independentes e que são dadas pela fórmula $N = 2^{\text{número variáveis independentes}}$.

Na representação dos mapas de karnaugh as variáveis devem ser dispostas de forma a que só haja uma quadricula que intersekte todos os conjuntos das variáveis independentes existentes e na representação do universo metade das quadriculas deve pertencer ao conjunto de cada variável independente e a outra metade das quadriculas do universo deve pertencer à variável negada.

. Mapa de karnaugh a 1 variável implica $N=2^1=2$ conjuntos

\overline{a}	
$a/$	a

. Mapa de karnaugh a 2 variáveis implica $N=2^2=4$ conjuntos

\overline{a}	
$a/.b/$	$a.b/$
$b \mid$	$a.b$

Computação Física

. Mapa de karnaugh a 3 variáveis implica $N=2^3=8$ conjuntos

		a	
		<hr/>	
c		a/.b/.c/	a.b/.c/
		a.b.c/	a/.b.c/
c		a/.b/.c	a.b/.c
		a.b.c	a/.b.c
		<hr/>	
		b	

. Mapa de karnaugh a 4 variáveis implica $N=2^4=16$ conjuntos

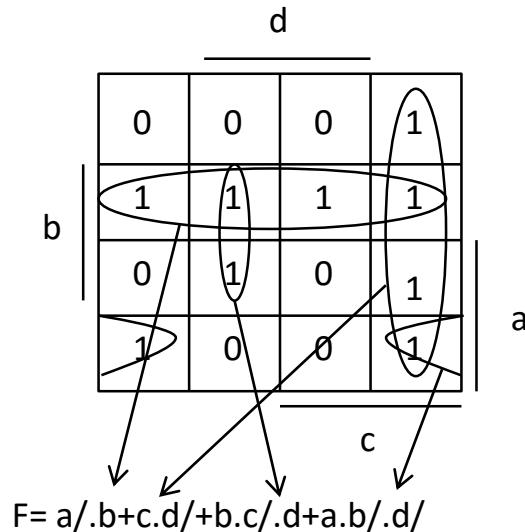
		a	
		<hr/>	
c		a/.b/.c/.d/	a.b/.c/.d/
		a.b.c/.d/	a/.b.c/.d/
c		a/.b/.c.d/	a.b/.c.d/
		a.b.c.d/	a/.b.c.d/
c		a/.b/.c.d	a.b/.c.d
		a.b.c.d	a/.b.c.d
c		a/.b/.c/.d	a.b/.c/.d
		a.b.c/.d	a/.b.c/.d
		<hr/>	
		b	

Computação Física

O objetivo principal dos mapas de karnaugh é permitir tirar uma expressão simplificada de qualquer função dependente desde que se cumpra a seguinte regra na associação de 1's.

Regra: Agrupe todos os 1's duma qualquer função dependente, em que cada grupo deve ter o máximo número de 1's vizinhos numa quantidade igual a uma potência de 2, ou seja, $2^0=1$, $2^1=2$, $2^2=4$, $2^3=8$, $2^4=16$, etc. Minimizando o número de grupos. Os 1's são vizinhos quando estão em linhas e colunas vizinhas. As linhas e as colunas dos extremos do mapa de karnaugh são vizinhas.

Exemplo: Suponha a seguinte tabela de verdade da função $F = f(a,b,c,d)$.
Retire a função lógica simplificada de F.



a	b	c	d	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

Computação Física

Exercícios:

Construa a tabela de verdade e o mapa de karnaugh de cada uma das seguintes funções lógicas e obtenha a expressão simplificada para cada função dada:

1. $F_1 = a.b + a/.b + a/.b/$
2. $F_2 = (a/ + b/ + c/)/$
3. $F_3 = b/ + a.b.c/$
4. $F_4 = (a.b)/ + a.b.c/$
5. $F_5 = a.c/ + b.c/ + a.c + b/.c$
6. $F_6 = x.y.z + x.y.z/ + x.y/.z + x.y/.z/$
7. $F_7 = a.d/ + c.b.a.d/ + c/.b/ + c/$
8. $F_8 = (x.w.y.z + (z/+x/)/)/$

Projete o circuito combinatório para resolver o seguinte problema:

O conselho diretivo de uma escola é formado por 4 membros que têm de votar sobre assuntos de gestão da escola. Pretende-se um circuito combinatório mínimo que acenda um led verde quando a maioria dos votantes votar afirmativamente e um led vermelho quando a maioria votar negativamente a uma determinada decisão.

Computação Física

2.5. Função lógica XOR

A função lógica XOR, também conhecida por exclusive OR ou em português “ou exclusivo”, a 2 variáveis independentes pode ser definida pela tabela de verdade apresentada na figura 2.4.1.

variáveis independentes		função dependente
a	b	$F = a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

A função XOR tem como operador o carácter “mais envolto numa bola” : \oplus

Se quiser escrever “F é igual à função a xor b” em notação lógica é $F = a \oplus b$

A representação da porta lógica da função XOR é:

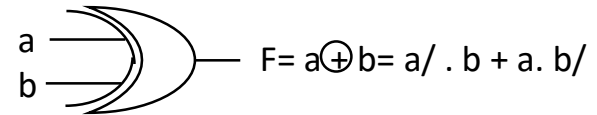


Figura 2.4.1 – Tabela de verdade da função XOR .

Definição: A função lógica XOR é 1, sempre que existe **um número ímpar** de variáveis independentes a 1.

Propriedades da função lógica XOR:

- $a \oplus 0 = a$, o 0 é o elemento neutro;
- $a \oplus 1 = a/$, o 1 é o elemento de complementação;
- $a \oplus a = a/$, teorema da complementação;
- $a \oplus a/ = 1$, tautologia;
- $a \oplus b \oplus c = (a \oplus b) \oplus c$, propriedade associativa;
- $a \oplus b = b \oplus a$, propriedade comutativa;

Computação Física

2.6. Função lógica NAND

A função lógica NAND, também conhecida por NOT AND, a 2 variáveis independentes pode ser definida pela tabela de verdade apresentada na figura 2.6.1.

a	b	$F = a \uparrow b$
0	0	1
0	1	1
1	0	1
1	1	0

A função NAND tem como operador uma seta virada para cima: \uparrow

Se quiser escrever “F é igual à função a NAND b” em notação lógica é $F = a \uparrow b = (a.b)/$

As portas lógicas da função NAND são:

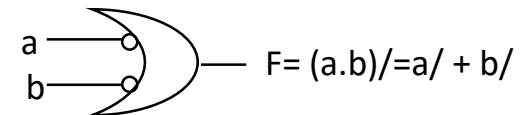
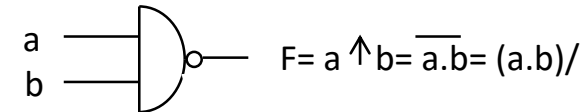


Figura 2.6.1 – Tabela de verdade da função NAND .

Definição: A função lógica NAND é 1, sempre que existe **pelo menos uma** variável independente a 0.

Propriedades da função lógica NAND:

- | | |
|--|---|
| $a \uparrow 0 = 1$ | , o 0 é o elemento universal; |
| $a \uparrow 1 = a/$ | , o 1 é o elemento de complementação; |
| $a \uparrow a = a/$ | , teorema da complementação; |
| $a \uparrow a/ = 1$ | , teorema de verdade; |
| $a \uparrow b \uparrow c \neq (a \uparrow b) \uparrow c$ | , não goza da propriedade associativa; |
| $a \uparrow b = b \uparrow a$ | , propriedade comutativa; |

Analisando as propriedades da função NAND, esta função permite implementar qualquer outra função lógica AND, OR ou NOT. Assim, qualquer circuito pode ser implementado exclusivamente com portas lógicas NAND.

Computação Física

2.7. Função lógica NOR

A função lógica NOR, também conhecida por NOT OR, a 2 variáveis independentes pode ser definida pela tabela de verdade apresentada na figura 2.7.1.

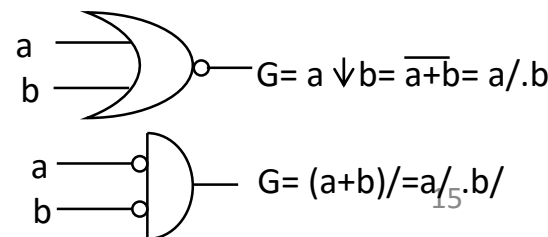
a	b	$G = a \downarrow b$
0	0	1
0	1	0
1	0	0
1	1	0

Figura 2.7.1 – Tabela de verdade da função NOR .

A função NOR tem como operador uma seta virada para cima: \downarrow

Se quiser escrever “G é igual à função a NOR b” em notação lógica é
 $F = a \downarrow b = (a+b)/= a/.b/$

As portas lógicas da função NOR são:



Propriedades da função lógica NOR:

$a \downarrow 0 = a/$, o 0 é o elemento de complementação;
 $a \downarrow 1 = 0$, o 1 é o elemento de absorção;
 $a \downarrow a = a/$, teorema da complementação;
 $a \downarrow a/ = 0$
 $a \downarrow b \downarrow c \neq (a \downarrow b) \downarrow c$, **não goza** da propriedade associativa;
 $a \downarrow b = b \downarrow a$, propriedade comutativa;

Analisando as propriedades da função NOR, esta função permite implementar qualquer outra função lógica AND, OR ou NOT. Assim, qualquer circuito pode ser implementado exclusivamente com portas lógicas NOR.

Computação Física

2.8. Multiplexer

Um multiplexer $2^N \times 1$ é um dispositivo combinatório com 2^N entradas e 1 saída. A saída apresenta o valor lógico da entrada selecionada correspondente ao número binário apresentado nas N entradas de seleção.

Por exemplo, a funcionalidade de um multiplexer 2×1 (2^1 entradas, 1 saída e 1 entrada de seleção) é definida como, quando a entrada de seleção é 0 a saída apresenta o valor lógico da entrada de informação 0 quando a entrada de seleção é 1 a saída apresenta o valor lógico da entrada de informação 1.

A tabela de verdade da figura 2.8.1 descreve a funcionalidade lógica de um multiplexer 2×1 (abreviado por mux2x1) e a figura 2.8.2 descreve o seu símbolo lógico.

Sel	I0	I1	S
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

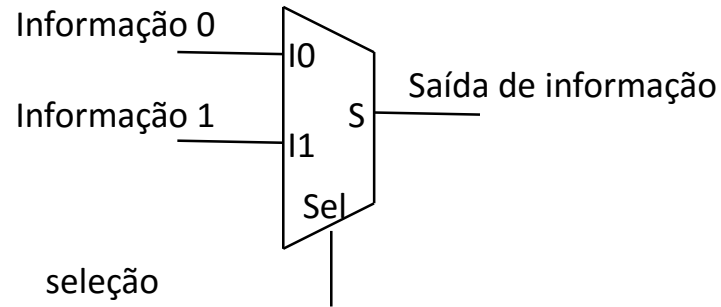


Figura 2.8.2 – Símbolo lógico de um multiplexer 2×1 .

A expressão lógica do multiplexer 2×1 é, $S = \text{Sel} \cdot I1 + \text{Sel}/ \cdot I0$

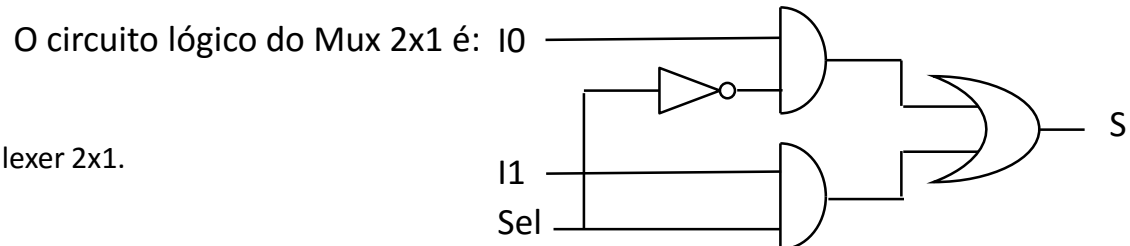


Figura 2.8.1 – Tabela de verdade de um multiplexer 2×1 .

Computação Física

2.9. Demultiplexer

Um demultiplexer 1×2^N é um dispositivo combinatório com 1 entrada de informação e 2^N saídas. A saída selecionada, correspondente ao número binário apresentado nas N entradas de seleção, apresenta a informação lógica presente à entrada.

Por exemplo, a funcionalidade de um demultiplexer 1×2 (1 entrada, 2^1 saídas e 1 entrada de seleção) é definida como, quando a entrada de seleção é 0 a saída $S0$ apresenta o valor lógico da entrada de informação I , quando a entrada de seleção é 1 a saída $S1$ apresenta o valor lógico da entrada de informação I .

A tabela de verdade da figura 2.9.1 descreve a funcionalidade lógica de um demultiplexer 1×2 (abreviado por dmux 1×2), a figura 2.9.2 descreve o seu símbolo lógico e na figura 2.9.3 está o circuito lógico.

Sel	I	S1	S0
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	0

Figura 2.9.1 – Tabela de verdade de um demultiplexer 1×2 .

As expressões lógicas das saídas do demultiplexer 1×2 são:

$$S0 = \text{Sel} / . I$$

$$S1 = \text{Sel} . I$$

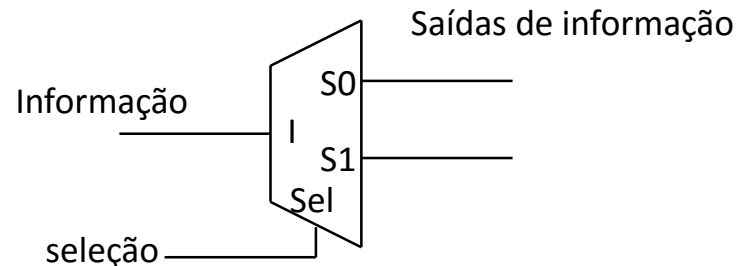


Figura 2.9.2 – Símbolo lógico de um demultiplexer 1×2 .

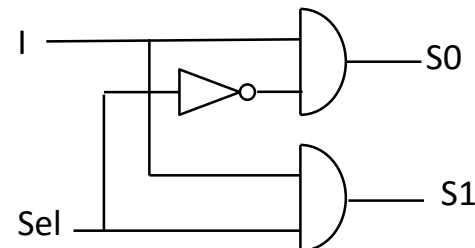


Figura 2.9.3 Circuito lógico de um demultiplexer 1×2 .

Computação Física

Capítulo 2 – Circuitos Sequenciais

1. Células de memória - flip-flop's

Nos circuitos combinatórios o valor duma saída (função dependente) depende apenas dos valores lógicos das entradas (variáveis independentes) nesse instante.

Nos circuitos sequenciais o valor lógico duma saída (função dependente) depende dos valores lógicos das entradas mas também das condições de funcionamento a que o circuito esteve sujeito que se define como estado.

Tal comportamento pressupõe a existência de memória, por forma a guardar a informação acerca de acontecimentos passados.

Uma memória é constituída a partir de células de memória unitária que se designam por flip-flops. Cada flip-flop é capaz de memorizar 1 bit de informação, ou seja, permanecer em 1 de 2 estados possíveis. Designa-se como estado, o valor lógico presente na saída Q de cada célula de memória.

Vamos projetar uma célula de memória.

Por definição, uma célula de memória pode ter 2 entradas D e E e uma saída Q com o seguinte comportamento:

1. Quando $E=1$ então $Q=D$ - atualização do valor lógico da célula em função do valor em D;
2. Quando $E=0$ então $Q=Q^*$ - condição de memorização do valor lógico da célula.

Q^* define o último valor lógico de Q quando $E=1$.

A tabela de verdade deste flip-flop pode ser expressa da seguinte forma:

E	D	Q
0	0	Q^*
0	1	Q^*
1	0	0
1	1	1

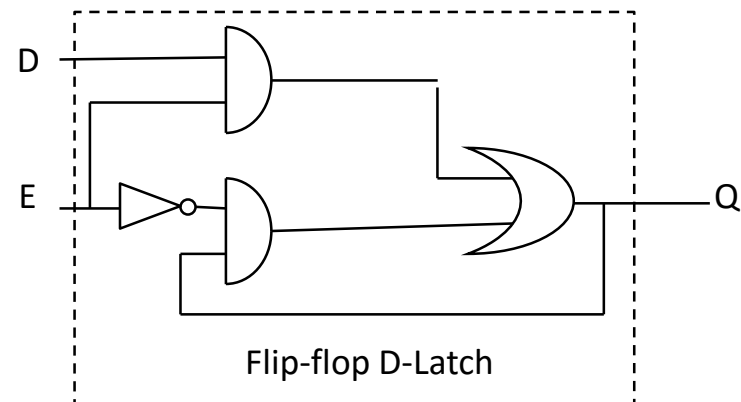
Computação Física

Mas como uma tabela de verdade é definida com 0's e 1's então pode-se reescrever a tabela anterior no formato conhecido, ou seja:

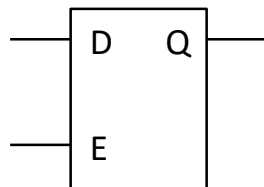
Q*	E	D	Q
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

		D	
Q		0	0
		1	0
Q*		1	0
		1	0
		E	

$$Q = Q^* \cdot E / + D \cdot E$$



Símbolo Flip-Flop D-Latch

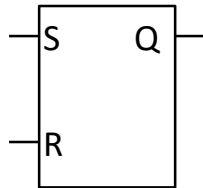


Computação Física

1.1. Flip-flops tipo latch

S	R	Q
0	0	Q^*
0	1	0
1	0	1
1	1	-

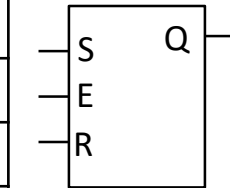
Flip-flop Set-Reset



Símbolo FF SR

E	S	R	Q
0	-	-	Q^*
1	0	0	Q^*
1	0	1	0
1	1	0	1
1	1	1	-

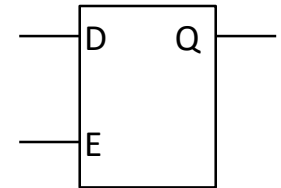
Flip-flop Set-Reset Enable



Símbolo FF SRE

E	D	Q
0	0	Q^*
0	1	Q^*
1	0	0
1	1	1

Flip-flop D-Latch

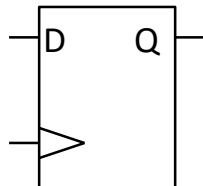


Símbolo FF D-latch

1.2 Flip-flops tipo edge-triggered

clk	D	Q
0,1,↓	0	Q^*
0,1,↓	1	Q^*
↑	0	0
↑	1	1

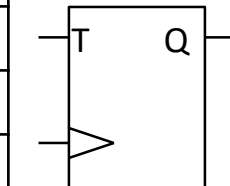
Flip-flop D-edge triggered



Símbolo FF D

clk	T	Q
0,1,↓	0	Q^*
0,1,↓	1	Q^*
↑	0	Q^*
↑	1	$Q^*/$

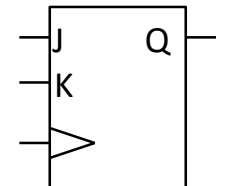
Flip-flop T-edge triggered



Símbolo FF T

clk	J	K	Q
0,1,↓	-	-	Q^*
↑	0	0	Q^*
↑	0	1	0
↑	1	0	1
↑	1	1	$Q^*/$

Flip-flop JK-edge triggered



Símbolo FF JK

Computação Física

2. Projeto de um circuito sequencial

O projeto hardware de um circuito sequencial é baseado em células de memória ou flip-flops e designa-se por máquinas de estado finitas. Os modelos existentes são o modelo de Moore, o modelo de Mealey e o modelo composto Moore-Mealey apresentados respetivamente nas figuras 2.1, 2.2 e 2.3.

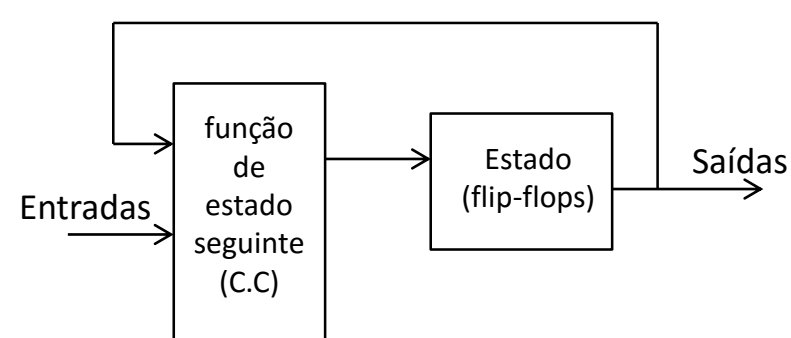


Figura 2.1 - Modelo de Moore.

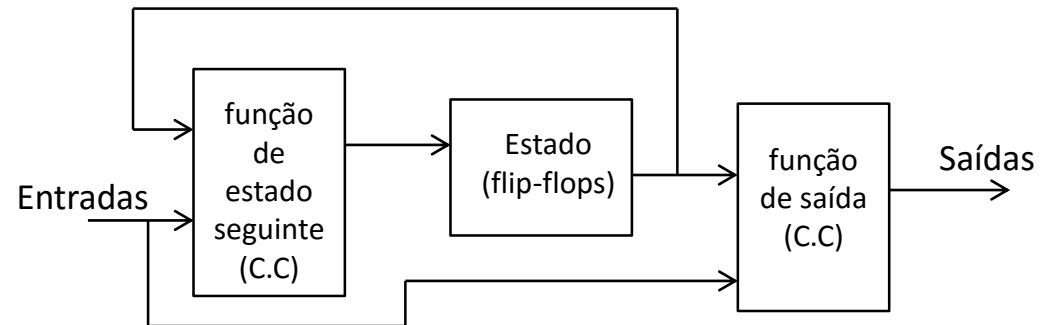


Figura 2.2 - Modelo de Mealey.

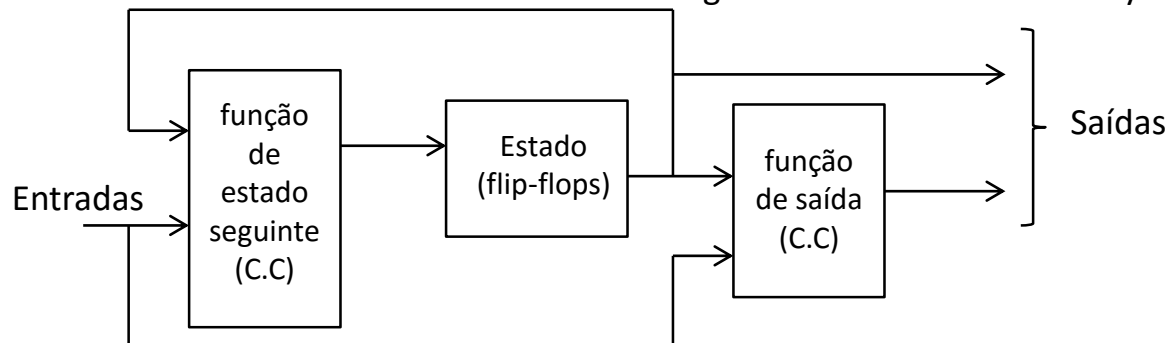


Figura 2.3 - Modelo composto Moore-Mealey.

Computação Física

2.1. Contadores

Um contador é um circuito síncrono com as transições ascendentes ou descendentes da entrada de relógio ou *clock*, definido por duas características:

1. Módulo da contagem;
2. Modo de contagem;

Por exemplo, um contador módulo 4 crescente é um dispositivo que ao ritmo das transições ascendentes do clock faz variar as suas saídas como o descrito na figura 3.1.

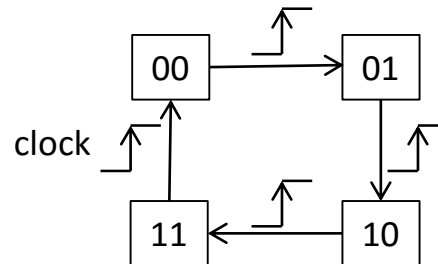


Figura 3.1 - Contador crescente módulo 4.

Um contador módulo 4 decrescente é um dispositivo que ao ritmo das transições ascendentes do clock faz variar as suas saídas como o descrito na figura 3.2.

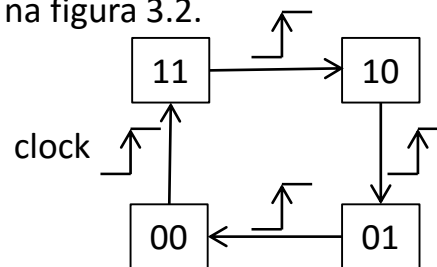


Figura 3.2 - Contador decrescente módulo 4.

Computação Física

3. ASM – Algorithmic State Machine

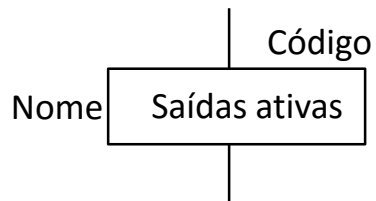
A linguagem gráfica *Algorithmic State Machine* abreviada por ASM serve para representar máquinas de Moore-Mealey na resolução de problemas. Um circuito sequencial também pode ser interpretado como sendo uma máquina de estados. Em que estado define as ações que uma máquina está a fazer num determinado instante temporal, tanto em termos de ativação das saídas como de teste das entradas.

A linguagem ASM pode ser utilizada na implementação circuitos sequenciais síncronos ou assíncronos embora nos iremos focar apenas na implementação de circuitos sequenciais síncronos com um sinal de sincronismo designado por *clock*.

O clock é uma onda digital ciclica com apenas dois níveis de tensão, também conhecida por onda quadrada, com uma determinada frequência de variação.

A linguagem ASM baseia-se nos seguintes 3 símbolos:

i. Estado

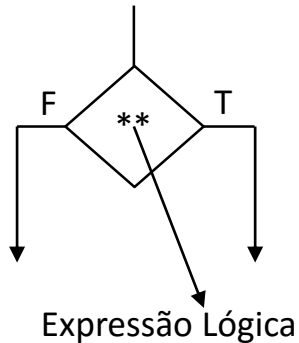


Estado – define o que a máquina está a fazer num determinado ciclo de *clock*. O tempo de permanência mínimo de uma máquina síncrona num estado é de 1 ciclo de *clock*.

Um estado é definido graficamente por um retângulo que contém 3 informações importantes: i. o “Nome” que é um elemento simbólico mas que deve transmitir, tanto quanto o possível, a funcionalidade do estado; ii. O “Código” que é um número binário e que vai identificar fisicamente o estado; iii. As “saídas ativas” que são os nomes de todas as saídas que estão ativas sempre que a máquina de estados

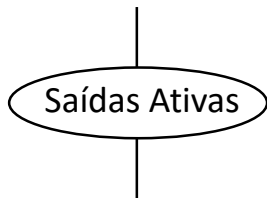
Computação Física

ii. Decisão



Decisão – define dois caminhos da máquina consoante o valor da avaliação da “Expressão Lógica” ser verdadeira (V ou 1) ou falsa (F ou 0).
As decisões estão ligadas à tomadas de decisão de um estado.

iii. Ação condicional



Ação condicional – define quais as saídas que estão ativas “Saídas Ativas” função de estado e de variáveis independentes.

Computação Física

4. Projeto hardware de Um Contador

Em hardware, um contador é um dispositivo que obedece ao modelo de Moore que é implementado à custa de células de memória edge-triggered e em que cada saída do contador é definida à custa de um flip-flop edge-triggered.

As entradas e saídas de um contador módulo 4 crescente que conta 0,1,2 e 3 pode ser descrito através da figura 4.1 e o respetivo ASM através da figura 4.2.

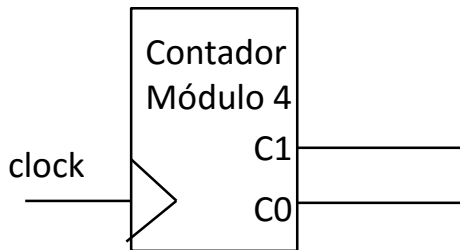


Figura 4.1 – Entradas e Saídas de um contador.

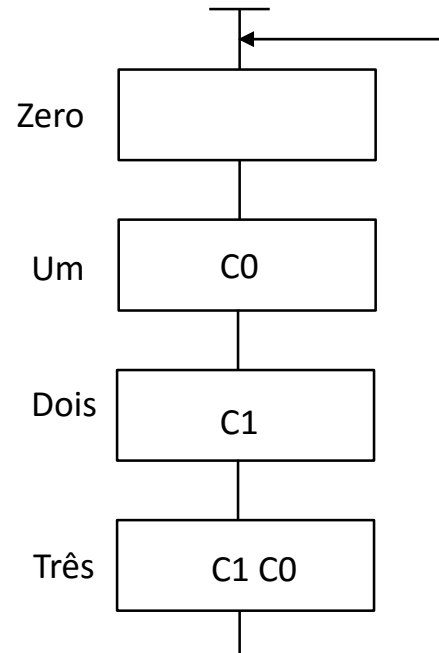


Figura 4.2 – ASM de um contador.

Computação Física

. Continuação do projeto hardware de um Contador

A implementação de um ASM como um circuito digital implica a identificação de cada estado com um código diferente. Como o ASM do contador tem 4 estados diferentes, implica a codificação de cada estado com dois bits que vamos designar como $Q0^*$ e $Q1^*$. Assim, o ASM fica como o da figura 4.3.

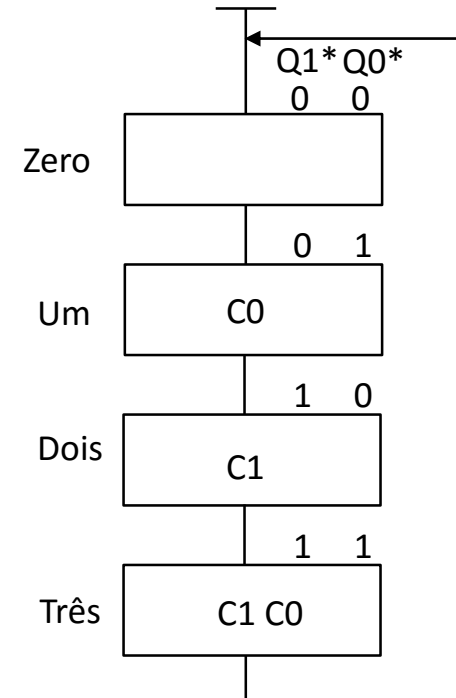


Figura 4.3 – ASM codificado.

O modelo genérico de um circuito sequencial obedece ao modelo Moore-Mealey representado na figura 2.3. Se usarmos flip-flops JK edge-triggered na implementação do contador então teremos o modelo:

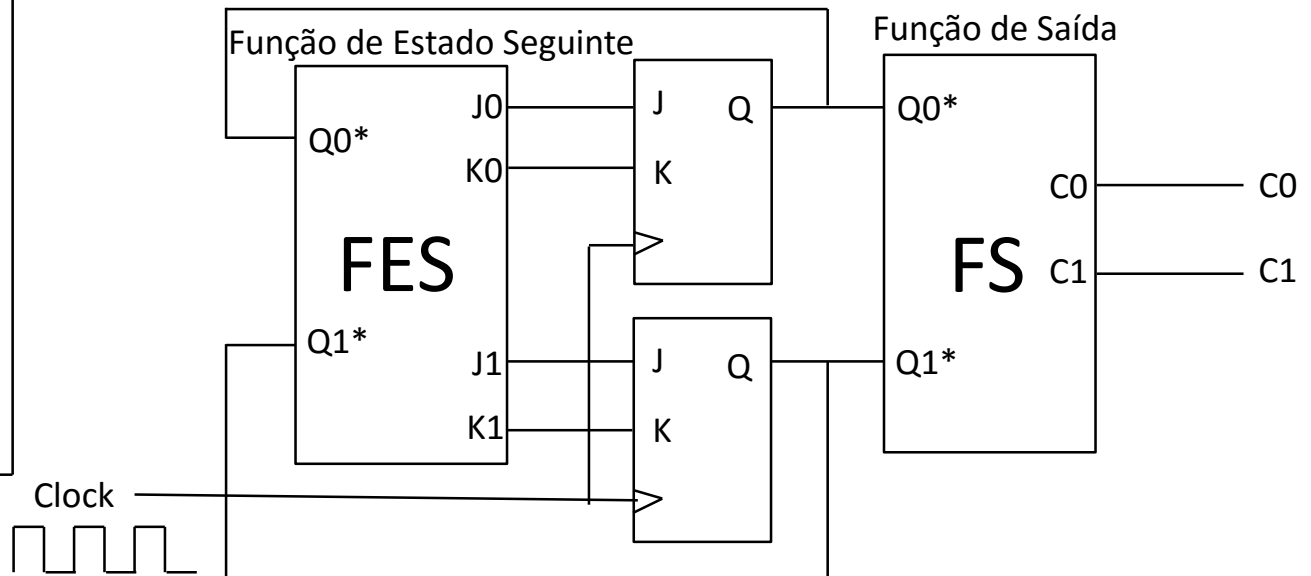


Figura 4.4 - Modelo de Moore-Mealey do contador.

Computação Física

. Continuação do projeto hardware de um Contador

- Implementação da função de estado seguinte (FES)

A FES é uma função combinatória com 2 entradas e 4 saídas que pode ser representada na seguinte tabela de verdade:

Q_1^*	Q_0^*	J_1	K_1	J_0	K_0
0	0	0	x	1	x
0	1	1	x	x	1
1	0	x	0	1	x
1	1	x	1	x	1

Indiferença - uma representação que significa que nesta condição a função tanto pode tomar o valor lógico 0 como 1.

Passando da tabela de verdade para os mapas de karnaugh e tirando as expressões lógicas:

J_1

	Q_0^*
	0
Q_1^*	1
	x

$J_1 = Q_0^*$

K_1

	Q_0^*
	x
Q_1^*	1
	0

$K_1 = Q_0^*$

J_0

	Q_0^*
	1
Q_1^*	x
	1

$J_0 = 1$

K_0

	Q_0^*
	x
Q_1^*	1
	x

$K_0 = 1$

Computação Física

. Continuação do projeto hardware de um Contador

- Implementação da função de saída (FS)

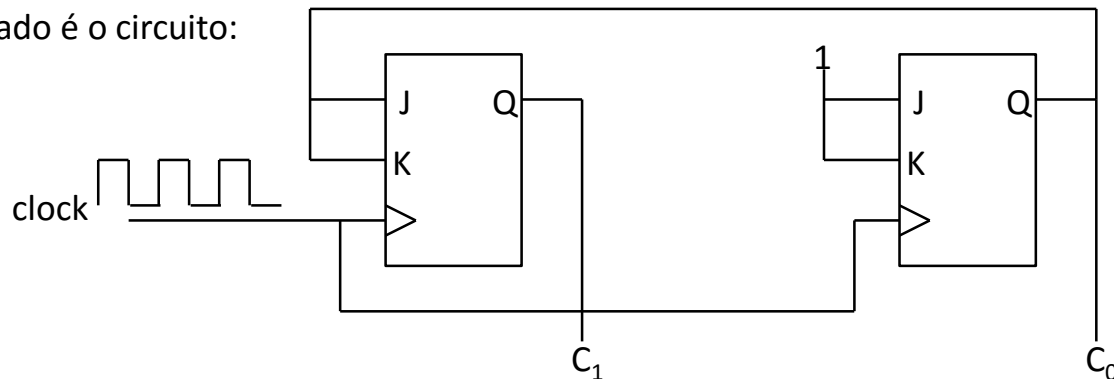
A FS é uma função combinatória com 2 entradas e 2 saídas que pode ser representada na seguinte tabela de verdade:

$Q1^*$	$Q0^*$	$C1$	$C0$
0	0	0	0
0	1	0	1
1	0	1	0
1	1	1	1

Da tabela de verdade tira-se diretamente as seguintes expressões lógicas simplificadas:

$$C1 = Q1^*, C0 = Q0^*.$$

O resultado é o circuito:



Computação Física

5. Registos

Em hardware, um registo é um dispositivo que é implementado à custa de flip-flops tipo D edge-triggered ou D-Latch e que permite registar ou memorizar informação binária.

Exemplos de registos de 8 bits são mostrados nas figuras 5.1 e 5.2.

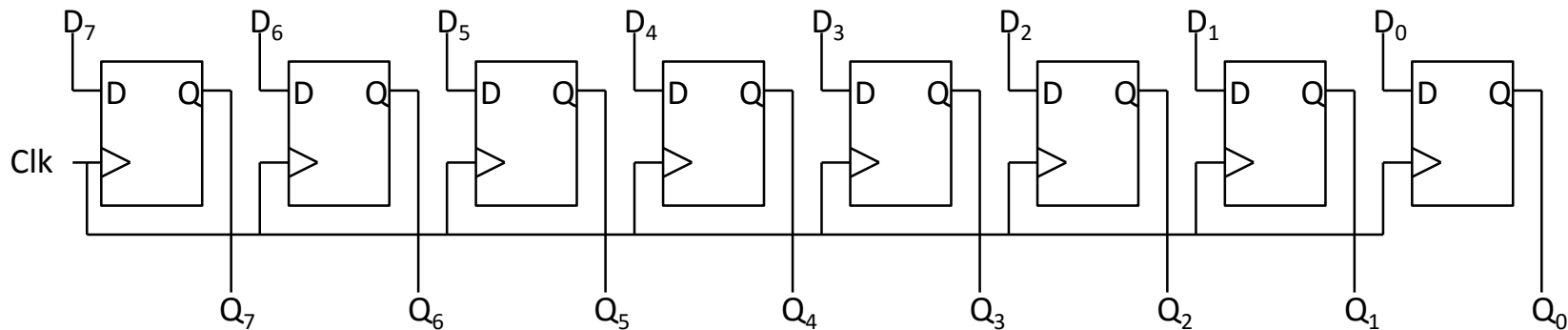


Figura 5.1 - Registo de 8 bits edge-triggered.

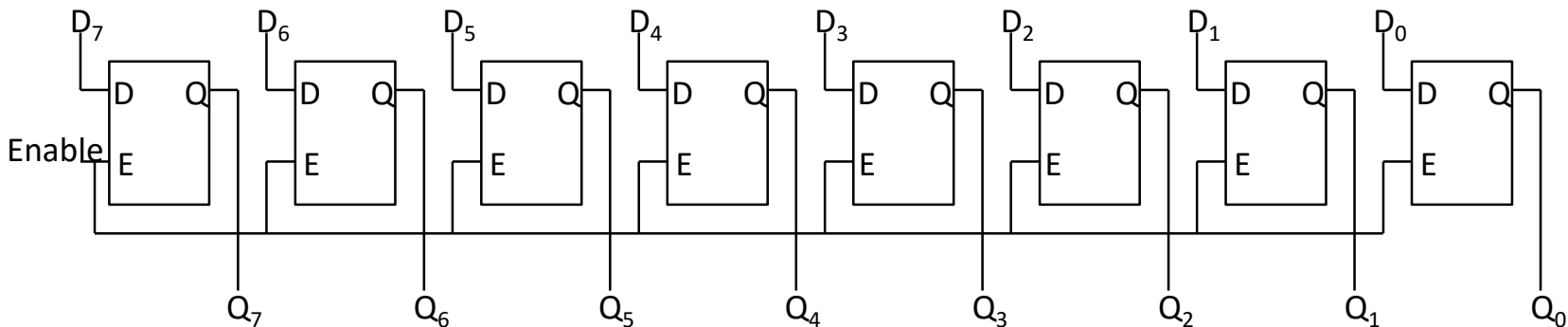


Figura 5.2 - Registo de 8 bits latch.

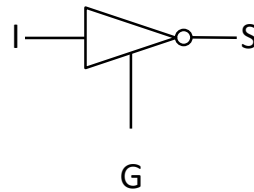
Computação Física

6. Portas Tri-state “três estados”

Uma porta tri-state é uma porta lógica com a capacidade de definir três estados à saída, o 0, o 1 e o Z em que por Z entende-se alta impedância, ou por outras palavras uma resistência infinita equivalente a um circuito aberto.

As tabelas de verdade e os símbolos lógicos de uma porta tri-state estão nas figuras 6.1 e 6.2.

G	I	S
0	0	Z
0	1	Z
1	0	1
1	1	0



G	I	S
0	0	1
0	1	0
1	0	Z
1	1	Z

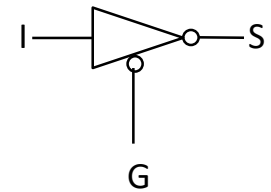
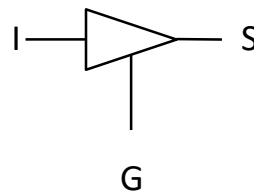


Figura 6.1 - Tabela de verdade e símbolo lógico de portas tri-state inversoras.

G	I	S
0	0	Z
0	1	Z
1	0	0
1	1	1



G	I	S
0	0	0
0	1	1
1	0	Z
1	1	Z

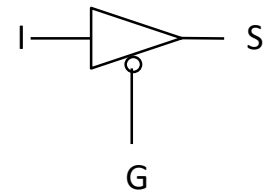


Figura 6.2 - Tabela de verdade e símbolo lógico de portas tri-state não inversoras.

Computação Física

7. Registos bidirecionais

Um registo bidirecional é desenhado a partir de um registo latch e de portas lógicas tri-state, ver secções 5 e 6. Este registo tem três sinais de controlo, um sinal OE/ (Output Enable) que permite por nas saídas do registo o valor memorizado, um sinal WE/ (Write Enable) que permite escrever um novo valor no registo e um sinal CE/ (Chip Enable) que quando ativo indica que o registo está selecionado.

A estrutura interna de um registo bidirecional de 8 bits é mostrado na figura 7.1.

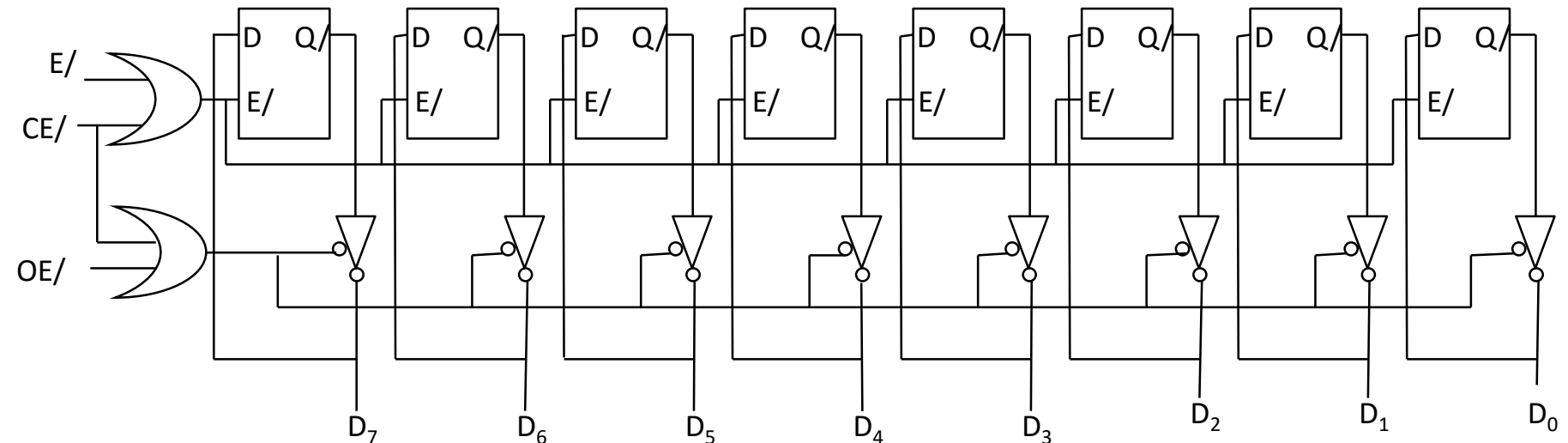


Figura 7.1 - Registo bidirecional de 8 bits.

Computação Física

8. Memória de acesso aleatório - RAM (Random Access Memory)

Uma RAM é constituída por um conjunto de registos bidireccionais de n bits cada sendo cada registo identificado internamente através de um endereço. A quantidade de bits de cada registo define a dimensão do conjunto de sinais por onde flui a informação de e para a RAM, este conjunto de sinais é designado por barramento de dados (*data bus- D_i*). A quantidade de registos internos define o número de bits necessários para o endereço de modo a que cada registo tenha um endereço distinto dos outros. Este conjunto de bits é designado por barramento de endereços (*address bus- A_i*). Existem ainda sinais de controlo que definem o fluxo de informação de e para a RAM que se designam respectivamente por OE/ (*output enable*) ou WE/ (*write enable*), ambos activos a zero. Também deve ser disponibilizado um sinal de controlo CE/ (*chip enable*) que função do seu nível lógico deve permitir realizar ou não operações sobre a RAM consoante esteja ou não activo.

A figura 8.1 ilustra a arquitectura interna de uma RAM 4x8, ou seja, um dispositivo de memória com 4 registos de 8 bits cada.

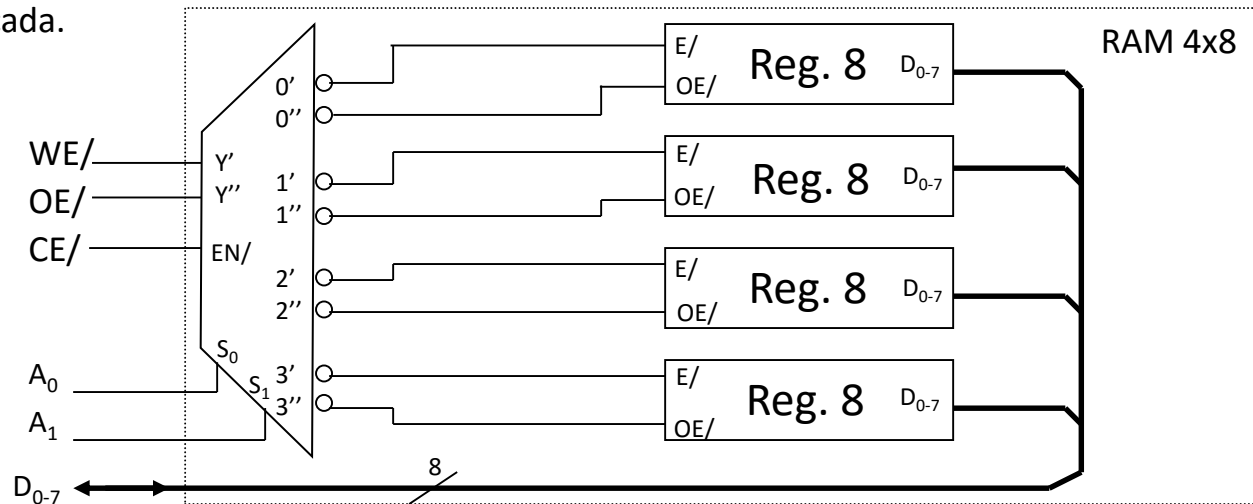


Figura 8.1 - RAM 4x8.

Computação Física

9. Sistema de acesso à memória RAM

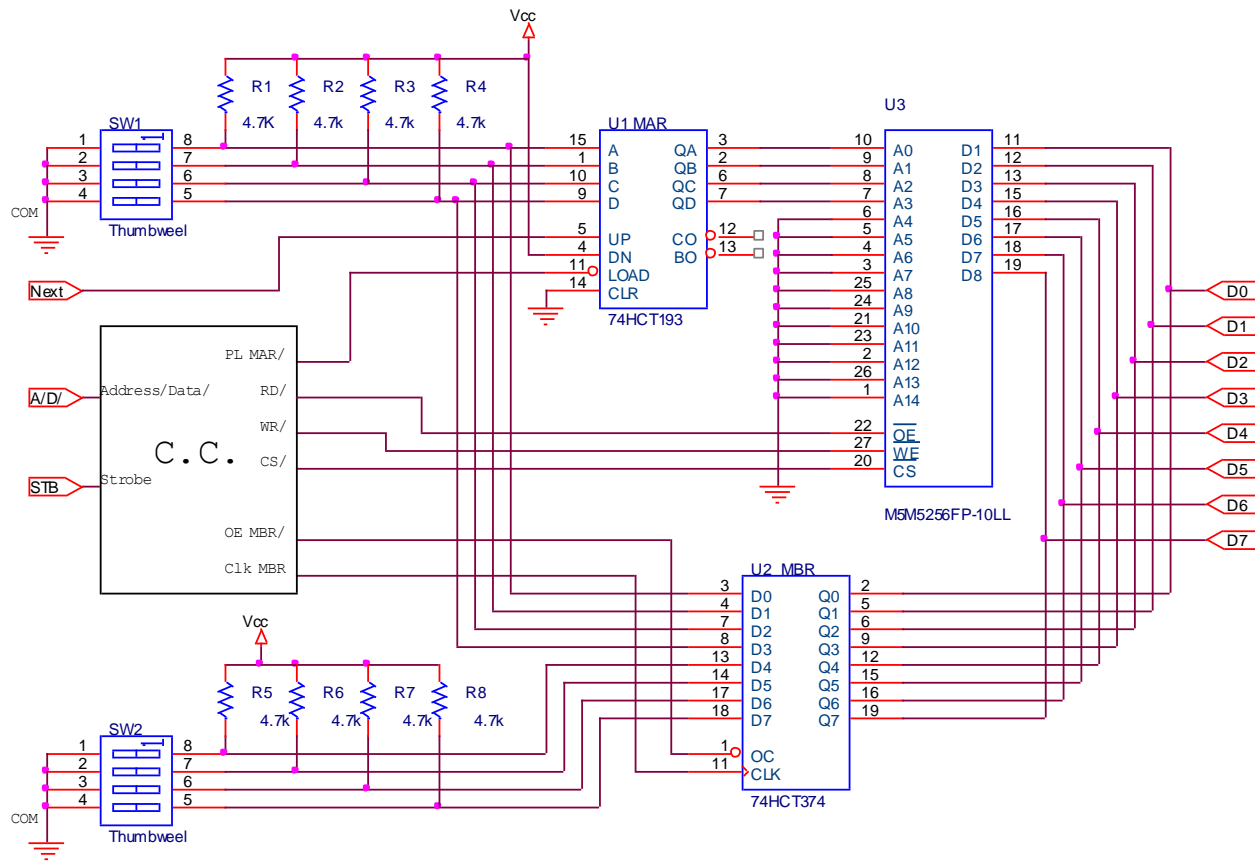
Depois de se compreender o modo de funcionamento e a estrutura interna de uma RAM pode-se então proceder ao projecto de um sistema simples que permita ler e escrever dados de uma RAM 16x8.

No sistema sugerido existem dois registos com os seguintes objectivos:

- *Memory Address Register* (MAR) define um endereço a 4 bits seleccionando a posição de memória a aceder. O seu conteúdo pode ser estabelecido muito simplesmente através de um *thumbwheel switch* hexadecimal quando o sinal *Address/Data/* está a 1 e houver a transição ascendente do sinal *Strobe*, ou então, por acção do sinal *Next* quando se pretenda aceder ao endereço seguinte;
- *Memory Buffer Register* (MBR) contém os 8 bits de dados a escrever na RAM. O seu conteúdo pode ser definido através de dois *thumbwheel switch* quando o sinal *Address/Data/* está a 0 e houver a transição ascendente do sinal *Strobe*.

Computação Física

As ligações sugeridas na figura seguinte ilustram o funcionamento descrito anteriormente.



Exercício: Projecte o circuito combinatório (C.C.) da figura de modo a que o circuito anterior opere correctamente sobre a RAM.

Computação Física

10. ROM (Read Only Memory)

Uma ROM que é abreviatura de Read Only memory é um dispositivo de memória permanente. As ROM têm a característica de memorizarem sempre os mesmos valores, mesmo após serem desligadas da alimentação.

Quando se estabelece a alimentação numa ROM e se endereça a uma determinada posição de memória da ROM, obtém-se sempre o mesmo valor de dados ou de informação.

Existem várias extensões à ROM que são consequência de vários processos físicos de gravação, as PROM (Programmable ROM), EPROM (Erasable PROM), EEPROM (Electrical EPROM) e a FLASHROM. Com excepção das PROM, todos os outros tipo de ROM suportam gravação da informação que contêm através de gravadores apropriados.

As FLASHROM são as memórias não voláteis que são utilizadas nos BIOS dos computadores, USB Disk, cartões MMC, MP3 e discos sólidos. Estas ROM têm um conjunto de sinais idênticos a uma RAM, ver figura 10.1, Address Bus, Data Bus, WE/, OE/ e CE/. No entanto, têm o defeito de permitir um baixo número de regravações, cerca de 10.000 regravações.

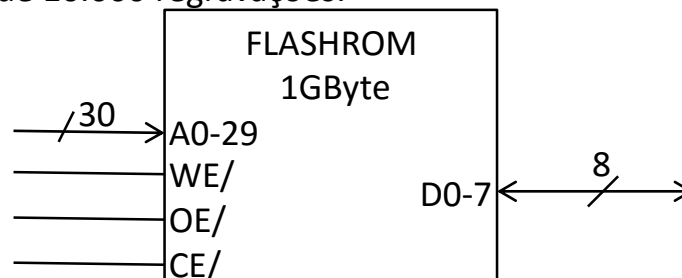


Figura 10.1 - FLASHROM de 1Gbyte.

Computação Física

Capítulo 3 – Projeto de CPU baseado na Representação do Encaminhamento de Dados

Este capítulo descreve o projecto de um microprocessador baseado na técnica de representação de encaminhamento de dados. O processador descrito é uma arquitectura harvard que cumpre uma instrução por cada ciclo de relógio(clock em terminologia anglo-saxónica).

A técnica de representação de dados é inicialmente ilustrada através do desenho de um multiplicador de dois números. Depois, no projecto de um conjunto de 4 registos (register file em terminologia anglo-saxónica). Finalmente no projecto de um microprocessador ou CPU (Central Processing Unit em terminologia anglo-saxónica).

1. Projecto da Estrutura Interna de um CPU e sua Interligação à Memória

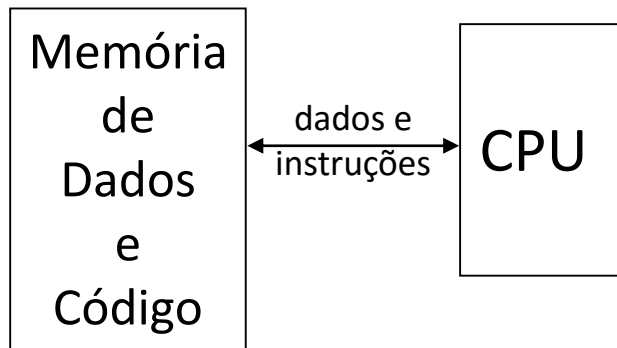
Um microprocessador ou uma unidade de processamento central (CPU – Central Processing Unit em terminologia anglo-saxónica) é uma máquina hardware que cumpre instruções lidas sequencialmente da memória de código. Por instrução entende-se uma acção ou um conjunto de acções que desencadeia uma transferência de informação, uma operação aritmética ou lógica, ou um salto na execução sequenciada de instruções.

Um CPU pode ser desenhado para integrar uma arquitectura Von Newman ou uma arquitectura Harvard. Na arquitectura Von Newman os dados e o código estão misturados na mesma memória física.

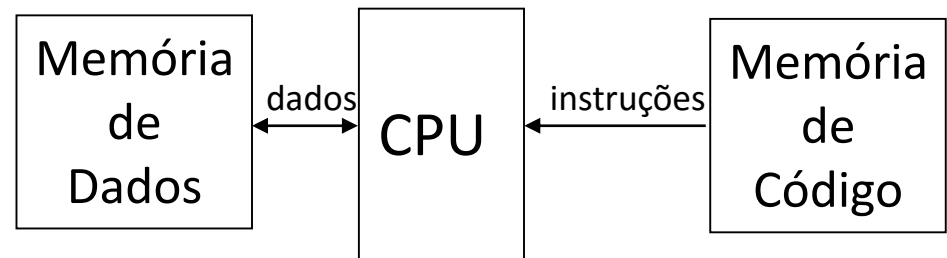
Numa arquitectura Harvard os dados e o código estão em memórias físicas diferentes. As seguintes figuras ilustram a diferença existente entre as duas arquitecturas, quanto ao conteúdo das memórias e barramento de dados.

Computação Física

Arquitetura Von Newman



Arquitetura Harvard



O desenho do CPU depende do tipo de arquitetura onde vai ser integrado. O nosso estudo recairá sobre a arquitetura Harvard que é a mais eficiente porque a informação está distribuída podendo haver paralelismo de ações sobre a memória de código e dados.

Computação Física

2. Projecto *hardware* baseado na Representação de Encaminhamento de dados (*data-path representation* em língua anglo-saxónica)

O projecto *hardware* baseado no encaminhamento de dados consiste na interligação de módulos *hardware*, (registos, conjunto de registos, memórias, unidades aritméticas e lógicas, etc) de forma a conseguir-se encaminhar a informação dos dispositivos fonte para os dispositivos destino. Esta interligação de dispositivos dará origem ao **módulo funcional** onde constam todos os dispositivos *hardware* necessários e respectivas ligações entre dispositivos. Outro problema do encaminhamento de informação é o aspecto da ordenação do encaminhamento da informação. Esta ordenação consegue-se através do desenho do **módulo de controlo**, este módulo garante a activação ordenada dos sinais de controlo dos dispositivos físicos envolvidos no módulo funcional. O módulo de controlo pode ser um circuito combinatório ou uma máquina de estados, dependendo do grau de complexidade envolvido no projecto.

O **módulo funcional** é um diagrama de blocos constituído por todos os dispositivos *hardware* disponibilizado pelos fabricantes, tais como, multiplexers, demultiplexers, comparadores, codificadores, decodificadores, flip-flops, registos, contadores, memórias, etc.

O **módulo de controlo** é um circuito combinatório ou uma máquina de estados que acciona os dispositivos existentes no módulo funcional. Para tal, o módulo de controlo tem como entradas, sinais vindos do módulo funcional, e como saídas, sinais que comandam os dispositivos constituintes do módulo funcional.

Computação Física

- **Módulo funcional**

Os dispositivos base no projecto *hardware* do módulo funcional baseado na representação de encaminhamento de dados são o registo e o multiplexer.

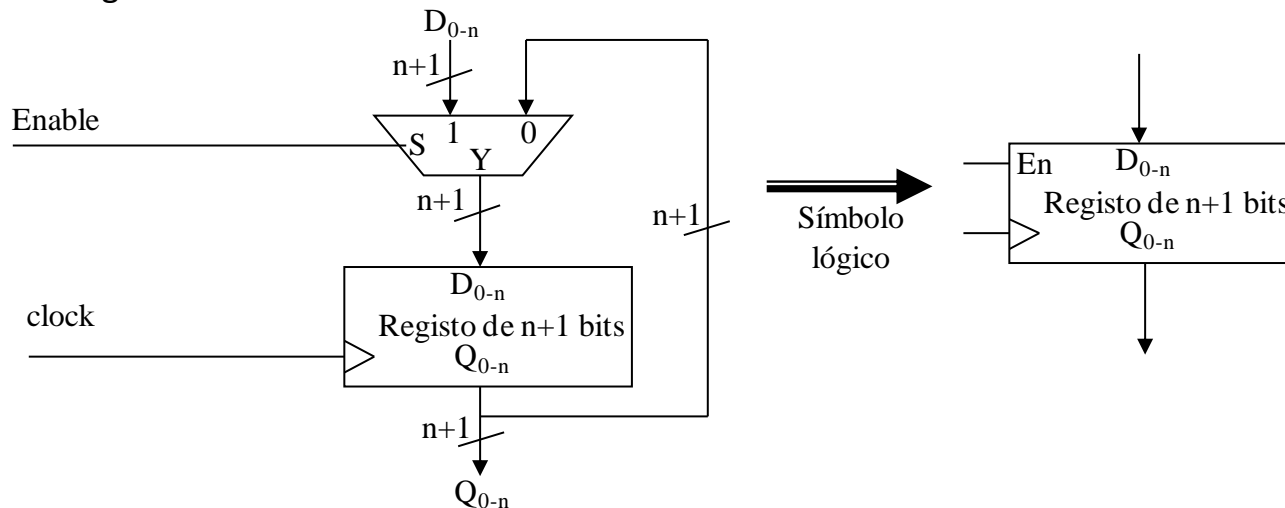
O registo permite o armazenamento de dados para posterior consulta.

O *multiplexer* permite o encaminhamento da informação entre os vários registos ou entre vários módulos *hardware*.

- Registo *edge-triggered* com *enable*

O registo usado na construção do diagrama de blocos do módulo funcional é *edge-triggered* com *enable*. Este registo actualiza a sua informação quando existe uma transição na sua entrada *edge-triggered* de *clock* e a entrada *enable* está activa.

O registo *edge-triggered* com *enable* é desenhado a partir de um registo *edge-triggered* (conjunto de flip-flops tipo D *edge-triggered*) e de um multiplexer, tal como o diagrama de blocos ilustrado na seguinte figura:



Computação Física

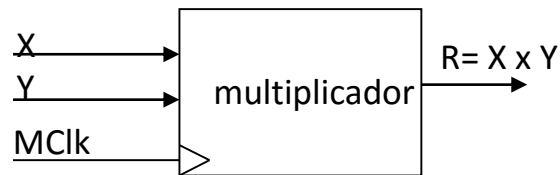
- **Passos para a concretização do projecto *hardware* dum dado problema**
 1. Especificar as entradas e saídas.
 2. Desenhar o algoritmo.
 3. Identificar os tipos *hardware* das entidades envolvidas no algoritmo. O tipo *hardware* duma entidade depende das acções que são feitas sobre a entidade no algoritmo.
 4. Desenhar o diagrama de blocos do módulo funcional baseado no encaminhamento da informação existente no algoritmo.
 5. Especificar as entradas e saídas do módulo de controlo.
 6. Desenhar o módulo de controlo.
 7. Simulação do módulo de controlo e funcional em software para corrigir erros
 8. Implementação do módulo funcional e do módulo de controlo em *hardware*.

Nota: No âmbito da disciplina de Computação Física o ponto 8 não será desenvolvido.

Computação Física

Exercício: Desenhe o diagrama de blocos do módulo funcional dum multiplicador baseado no algoritmo de somas sucessivas e o respectivo módulo de controlo.

1. Entradas e saídas do multiplicador



2. Algoritmo

```
For(R=0, l=Y ; l!=0 ; --l) R+= X;
```

3. Tipos *hardware*

l : registo; // sobre l são feitas as acções de registo ($l = Y$) e de decremento ($--l$)

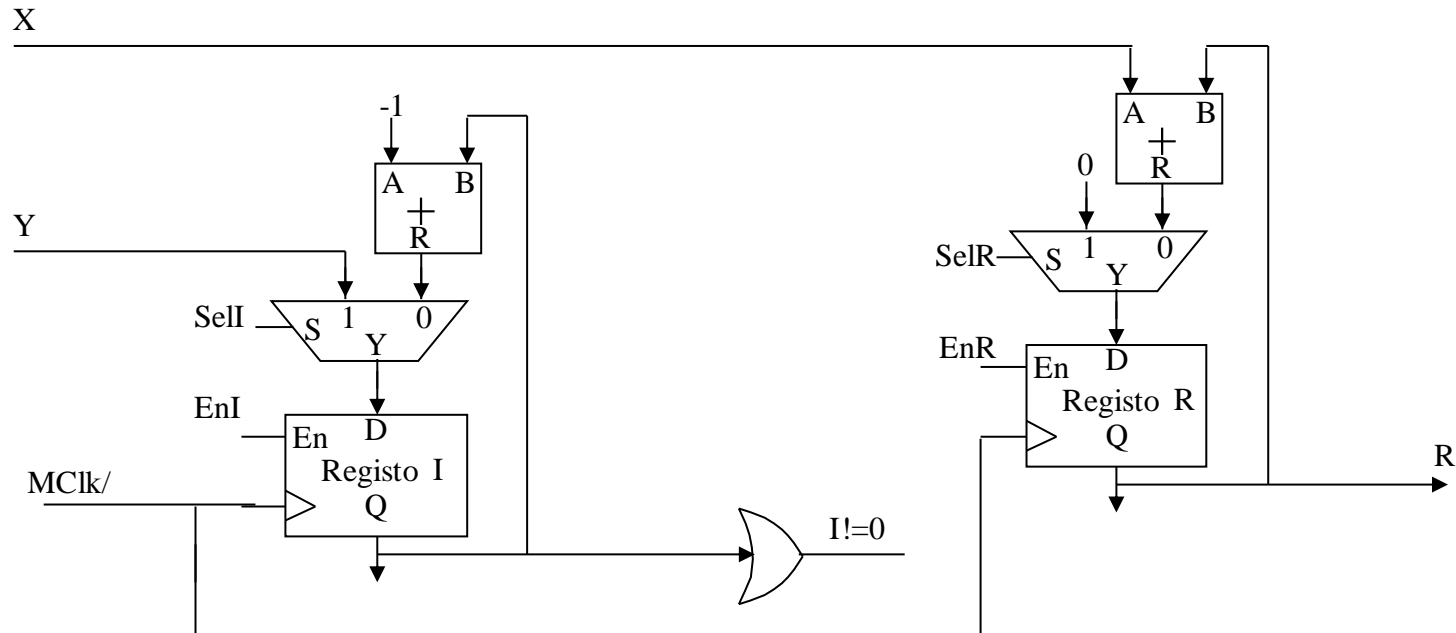
R : registo; // sobre R só são feitas as acções de registo ($R = 0$) e ($R += X$)

No algoritmo também estão envolvidas as entidades X e Y , mas estas são apenas os valores de entrada do multiplicador, portanto pode-se dizer que,

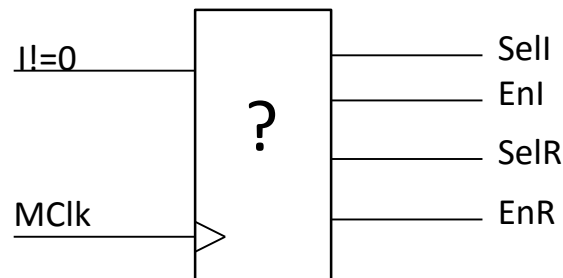
X, Y : entrada;

Computação Física

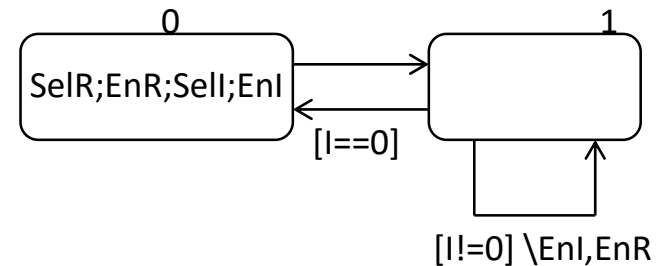
4. Diagrama de blocos do módulo funcional baseado no encaminhamento dos dados



5. Especificar as entradas e saídas do módulo de controle



6. Desenho do módulo de controle

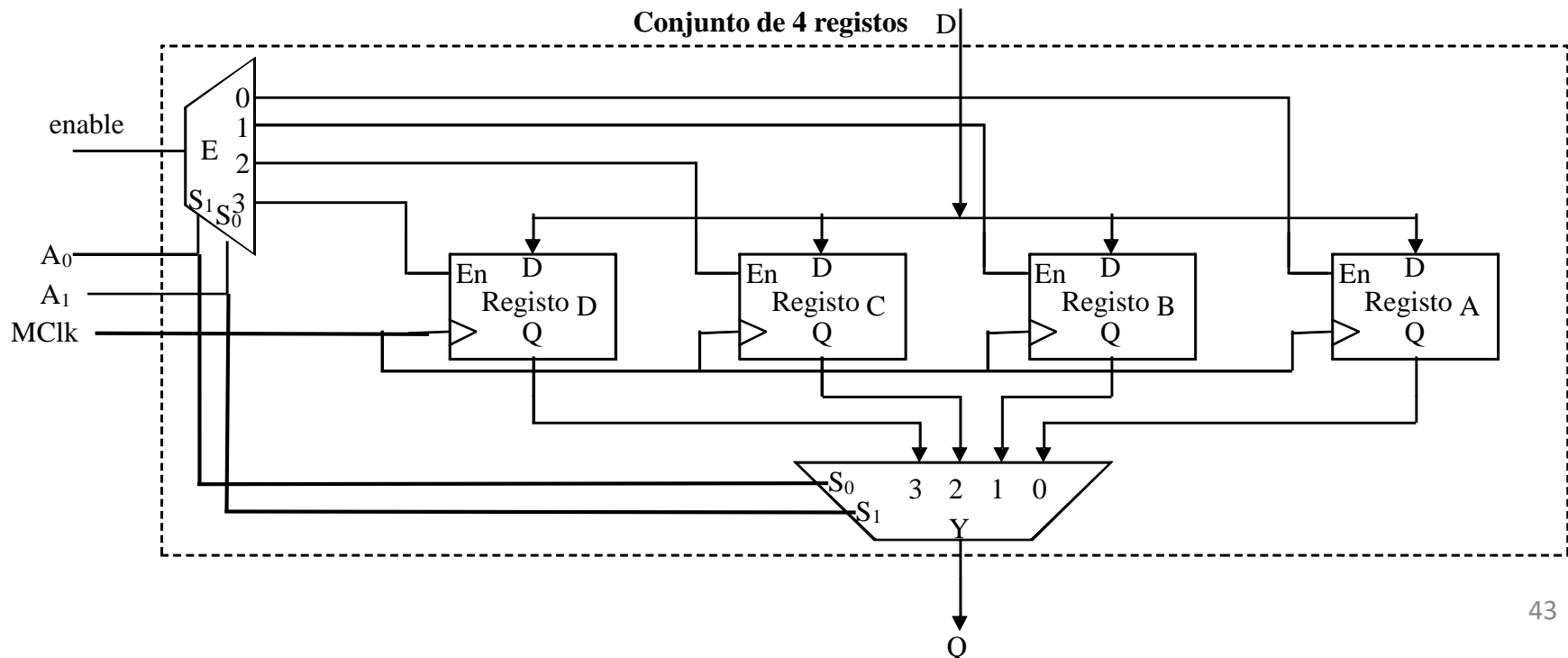


Computação Física

Exercício: Desenhe um módulo hardware com um conjunto de 4 registos (A, B, C, D) em que qualquer registo afecte outro qualquer registo. Por exemplo, o registo A pode ser afectado com o valor de qualquer um dos 4 registos, $A = A$ ou $A = B$ ou $A = C$ ou $A = D$. O mesmo é válido para os registos B, C e D.

Neste exercício o encaminhamento de dados tem que permitir que a saída dum registo possa estar ligada à entrada de qualquer outro registo.

Outra característica acrescentada na solução é que o *enable* de cada registo será obtido a partir dum *enable* geral ao conjunto de registos mais o endereço do registo. Os sinais A_0 e A_1 , designadas por bits de address, definem o registo onde se pretende ler ou escrever.



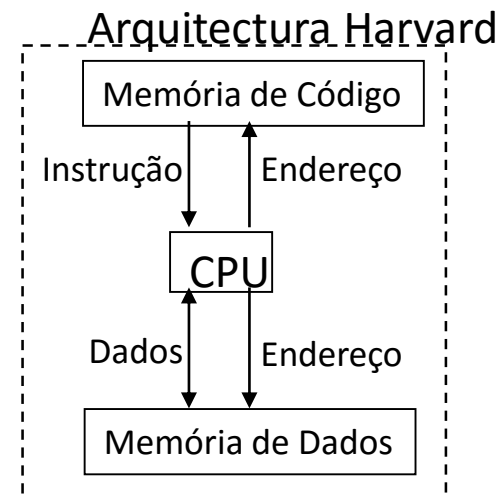
Computação Física

O projecto de um CPU obedece aos mesmos passos do projecto hardware:

1. Especificação dos registos de uso geral e dos barramentos de endereços e dados para as memórias;
 2. Especificação das instruções;
 3. Codificação das instruções;
 4. Desenho do módulo funcional;
 5. Desenho do módulo de controlo;
 6. Simulação da arquitectura para correcção de erros;
 7. Implementação do módulo de controlo e do módulo funcional em dispositivos hardware.
- Equivalente aos 3 primeiros pontos do projecto *hardware*.

Exercício: Desenhe um CPU para uma arquitectura Harvard com o seguinte conjunto de instruções de dimensão unitária:

Instrução	Funcionalidade
MOV A, @Rn	$A = (Rn)$
MOV @Rn, A	$(Rn) = A$
MOV A, Rn	$A = Rn$
MOV Rn, A	$Rn = A$
ADDC A, Rn	$A = A + Rn + C$
SUBB A, Rn	$A = A - Rn - C$
JNC rel6	Se (!Cy) $PC += rel6$
JNZ rel6	Se (!Zero) $PC += rel6$
JMP rel6	$PC += rel6$



Rn – conjunto de 4 registos; rel4 – valor relativo a 4 bits ; rel6 – valor relativo a 6 bits

Computação Física

1. Especificação dos registos de uso geral e dos barramentos de endereço e dados para as memórias

Rn – 4 registos de 8 bits; A – registo de 8 bits; PC – registo de 8 bits; C – flag de Carry;
rel4 – relativo a 4 bits; rel6 – relativo a 6 bits; todos os barramentos do CPU são a 8 bits.

2. Especificação das instruções

Instrução	Funcionalidade
MOV A, @Rn	A= (Rn)
MOV @Rn, A	(Rn)= A
MOV A, Rn	A= Rn
MOV Rn, A	Rn= A
ADDC A, Rn	A= A + Rn + C
SUBB A, Rn	A= A – Rn - C
JNC rel6	Se (!Cy) PC+= rel6
JNZ rel6	Se (!Zero) PC+= rel6
JMP rel6	PC+= rel6

As instruções são em geral agrupadas em classes ou famílias. As classes mais relevantes de instruções são as designadas por *transferência de informação*, *aritméticas*, *lógicas* e de *controlo*.

As instruções de *transferência de informação* são responsáveis pelas acções de leitura ou escrita em registos, contadores e *flags* da estrutura interna do CPU bem como pela alteração ou leitura de valores em dispositivos externos ao CPU mas que partilhem todos os seus barramentos.

Computação Física

As instruções *aritméticas* são aquelas que desempenham operações aritméticas e em geral são realizadas por uma máquina designada por unidade aritmética e lógica (ALU – *arithmetic logic unit*) e que é parte integrante da estrutura interna do CPU.

As instruções *lógicas* implementam as funções lógicas comuns e em geral são realizadas na ALU.

As instruções de *controlo* são todas as que podem alterar, de alguma forma, a execução sequencial de instruções.

3. Codificação das instruções

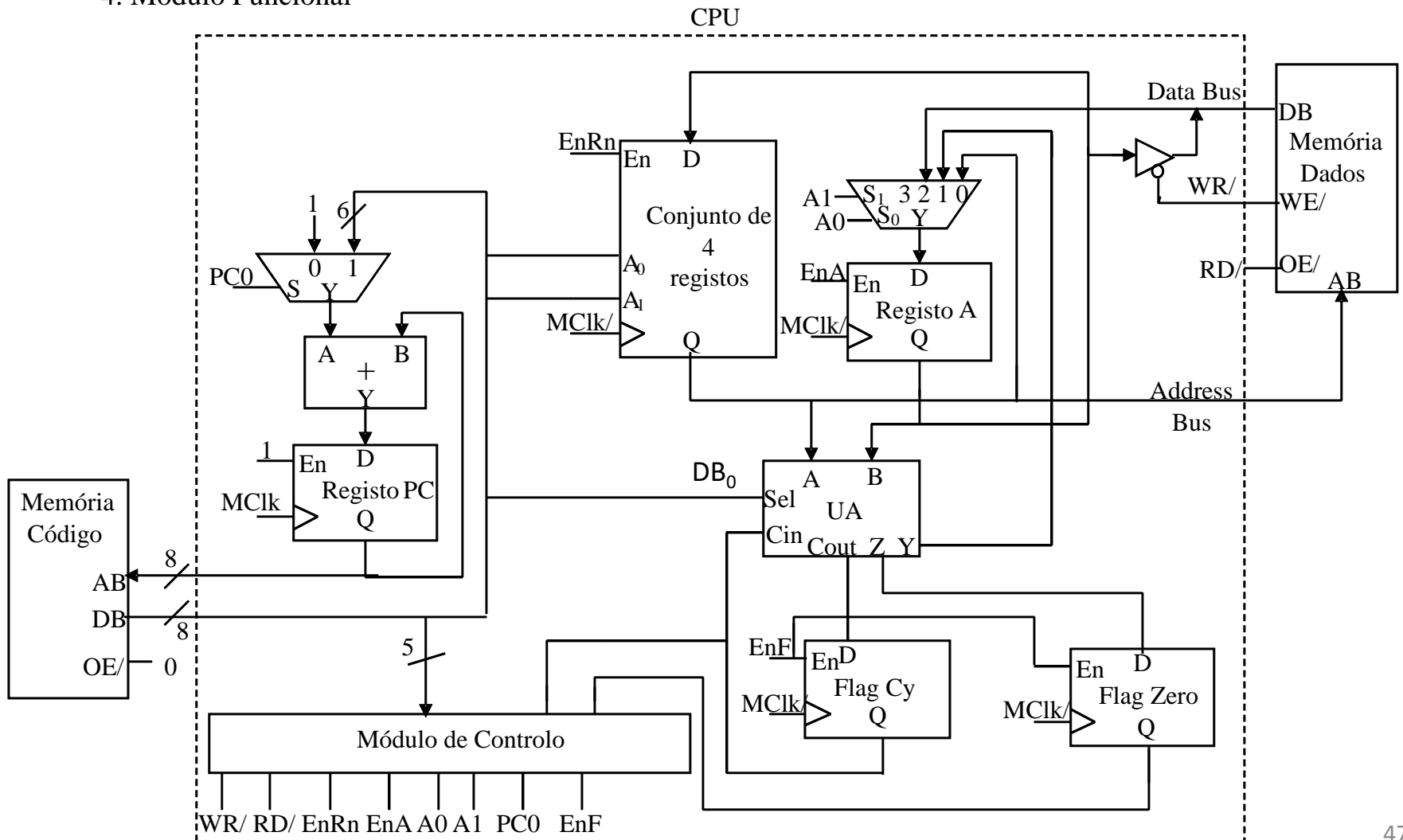
- Critério de codificação

Os bits D7 e D6 servem para distinguir as instruções que têm como único parâmetro Rn das outras instruções. Os bits D2, D1 e D0 servem para distinguir as instruções com Rn como parâmetro.

		codificação a 8 bits							
Instrução	Parâmetros	D7	D6	D5	D4	D3	D2	D1	D0
MOV A, @Rn	Rn	1	1	Rn1	Rn0	0	0	0	0
MOV @Rn, A	Rn	1	1	Rn1	Rn0	0	0	0	1
MOV A, Rn	Rn	1	1	Rn1	Rn0	0	0	1	0
MOV Rn, A	Rn	1	1	Rn1	Rn0	0	0	1	1
ADDC A, Rn	Rn	1	1	Rn1	Rn0	0	1	0	0
SUBB A, Rn	Rn	1	1	Rn1	Rn0	0	1	0	1
JNC rel6	rel6	1	0	r5	r4	r3	r2	r1	r0
JNZ rel6	rel6	0	1	r5	r4	r3	r2	r1	r0
JMP rel6	rel6	0	0	r5	r4	r3	r2	r1	r0

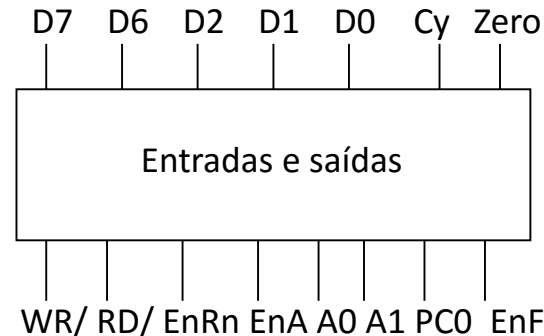
Computação Física

4. Módulo Funcional



Computação Física

5. Módulo de Controlo



Como as instruções do CPU que se está a projectar são simples, o módulo de controlo é um circuito combinatório porque não é necessário encadeamento de acções para se cumprir uma instrução. Assim, pode-se representar a activação dos sinais através da seguinte tabela de verdade.

Instrução	D7	D6	D2	D1	D0	Cy	Z	Sinais Activos
MOV A, @Rn	1	1	0	0	0	-	-	EnA, RD/, A1
MOV @Rn, A	1	1	0	0	1	-	-	WR/
MOV A, Rn	1	1	0	1	0	-	-	EnA
MOV Rn, A	1	1	0	1	1	-	-	EnRn
ADDC A, Rn	1	1	1	0	0	-	-	EnA, A0, EnF
SUBB A, Rn	1	1	1	0	1	-	-	EnA, A0, EnF
JNC rel6	1	0	-	-	-	0	-	PC0
JNC rel6	1	0	-	-	-	1	-	
JNZ rel6	0	1	-	-	-	-	0	PC0
JNZ rel6	0	1	-	-	-	-	1	
JMP rel6	0	0	-	-	-	-	-	PC0

Computação Física

- Tabela de Programação de EPROM para implementação do **Módulo de Controle**

EPROM 128x8

D7	D6	D2	D1	D0	Cy	Z		EnF	WR/	RD/	EnA	A1	A0	PC0	EnRn	
A6	A5	A4	A3	A2	A1	A0	Address	D7	D6	D5	D4	D3	D2	D1	D0	Data
0	0	-	-	-	-	-	[0,1Fh]	0	1	1	0	0	0	1	0	62h
0	1	-	-	-	-	0	[20h,22h,24h,26h,28h,2Ah,2Ch,2Eh,30h,32h,34h,36h,38h,3Ah,3Ch,3Eh]	0	1	1	0	0	0	1	0	62h
0	1	-	-	-	-	1	[21h,23h,25h,27h,29h,2Bh,2Dh,2Fh,31h,33h,35h,37h,39h,3Bh,3Dh,3Fh]	0	1	1	0	0	0	0	0	60h
1	0	-	-	-	0	-	[40h,41h,44h,45h,48h,49h,4Ch,4Dh,50h,51h,54h,55h,58h,59h,5Ch,5Dh]	0	1	1	0	0	0	1	0	62h
1	0	-	-	-	1	-	[42h,43h,46h,47h,4Ah,4Bh,4Eh,4Fh,52h,53h,56h,57h,5Ah,5Bh,5Eh,5Fh]	0	1	1	0	0	0	0	0	60h
1	1	0	0	0	-	-	[60h,63h]	0	1	0	1	1	0	0	0	58h
1	1	0	0	1	-	-	[64h,67h]	0	0	1	0	0	0	0	0	20h
1	1	0	1	0	-	-	[68h,6Bh]	0	1	1	1	0	0	0	0	70h
1	1	0	1	1	-	-	[6Ch,6Fh]	0	1	1	0	0	0	0	1	61h
1	1	1	0	0	-	-	[70h,73h]	1	1	1	1	0	1	0	0	F4h
1	1	1	0	1	-	-	[74h,77h]	1	1	1	1	0	1	0	0	F4h
1	1	1	1	-	-	-	[78h,7Fh]	0	1	1	0	0	0	0	0	60h

Computação Física

Para diminuir a dimensão da EPROM para 32 palavras em vez das 128 palavras necessárias, o sinal PC0 não foi produzido na EPROM gerando-se em vez deste, os sinais JMP, JNZ e JNC que resultam da descodificação do código de cada uma das instruções respetivas.

EPROM 32x10

D7	D6	D2	D1	D0		EnF	WR/	RD/	EnA	A1	A0	JMP	JNZ	JNC	EnRn	
A4	A3	A2	A1	A0	Address	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	Data
0	0	-	-	-	[0,7]	0	1	1	0	0	0	1	0	0	0	188h
0	1	-	-	-	[8,Fh]	0	1	1	0	0	0	0	1	0	0	184h
1	0	-	-	-	[10h,17h]	0	1	1	0	0	0	0	0	1	0	182h
1	1	0	0	0	18h	0	1	0	1	1	0	0	0	0	0	160h
1	1	0	0	1	19h	0	0	1	0	0	0	0	0	0	0	080h
1	1	0	1	0	1Ah	0	1	1	1	0	0	0	0	0	0	1C0h
1	1	0	1	1	1Bh	0	1	1	0	0	0	0	0	0	1	181h
1	1	1	0	0	1Ch	1	1	1	1	0	1	0	0	0	0	3D0h
1	1	1	0	1	1Dh	1	1	1	1	0	1	0	0	0	0	3D0h
1	1	1	1	-	[1Eh,1Fh]	0	1	1	0	0	0	0	0	0	0	180h

O sinal PC0 obtém-se a partir dos sinais gerados na EPROM designados por JMP, JNZ, JNC e das flags de Zero e Cy do CPU através da seguinte expressão lógica:

$$PC0 = JMP + JNZ.\overline{Zero} + JNC.\overline{Cy}$$

Computação Física

4. Portos paralelo de entrada e saída

O CPU ATMEGA em formato DIP disponibiliza 23 sinais digitais de entrada/saída agrupados em 2 portos bidireccionais de 8 bits cada (PB_{0-7} , PD_{0-7}) e um porto bidirecional de 7 bits (PC_{0-6}). Genericamente, cada bit pode consistir num *flip-flop* tipo D, num *driver* de saída e num *buffer tri-state* de entrada, tal como se ilustra na figura 4.1.

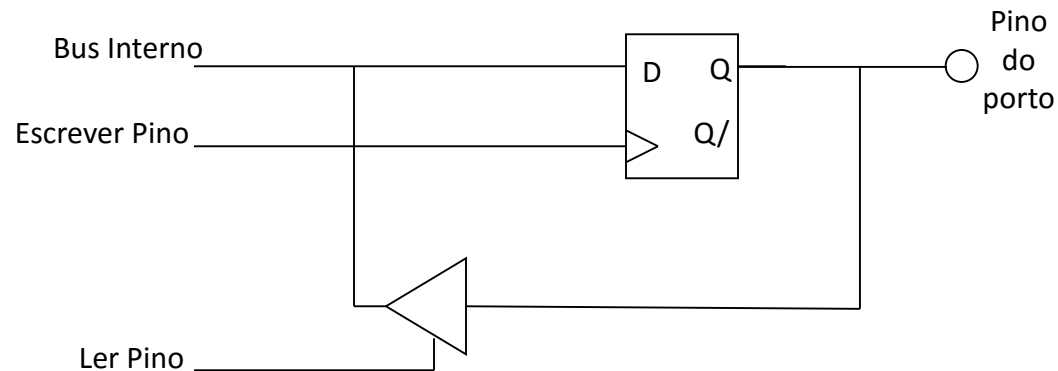


Figura 4.1 - Circuito genérico associado a um bit dos portos de entrada/saída.

Computação Física

3. O CPU ATMEGA 328 em formato DIP

O CPU ATMEGA 328 é o microprocessador base da placa do arduino e tem 32Kbytes de FlashROM, 1 Kbyte de EEPROM e 2Kbytes de RAM.

A FlashROM é onde reside o código das aplicações, a RAM é onde residem as variáveis da aplicação e a EEPROM é onde reside o código de StartUp do microsistema (arduino).

O microprocessador é um RISC(Reduced Instruction Set Computer) de 8 bits com 131 instruções e em que a maioria das instruções é cumprida num clock cycle.

Para um clock de 20MHz o processador cumpre no máximo 20MIPS.

O processador em termos de periféricos tem:

- 2 Contadores/Timers de 8 bit;

- 1 Contador/Timer de 16 bit;

- 1 Contador de tempo real com oscilador separado;

- 6 Canais PWM;

- 6 ADC de 10 bit;

- 1 USART programável;

- 1 Interface SPI Master/Slave;

- 1 Interface I2C;

- 1 Temporizador watchdog com um oscilador separado mas integrado no chip;

Interrupções e modo de acordar do CPU sensíveis a mudanças em pinos externos;

23 pinos programáveis de entrada/saída. (PB0-7, PC0-6 e PD0-7)

Computação Física

Capítulo 4 – O Arduino

1. A placa

A placa Arduino Duemilanove, mostrada na figura 1, é uma arquitetura baseada no microcontrolador ATMEGA 328 P a 16MHz. Esta placa tem 14 sinais digitais(DIGITAL) que podem ser configurados como entradas ou saídas, pinos 0 a 13 localizados na parte superior direita da figura 1. Destes 14 sinais digitais, os pinos 3, 5, 6, 9, 10 e 11 podem ser programados como sinais PWM de saída. A placa tem 6 sinais analógicos (ANALOG IN), os pinos 0 a 5 localizados na parte inferior direita da placa da figura 1. A placa tem 5 sinais (POWER) localizados na parte inferior central. O sinal Vin que permite alimentar o board com 9V. O sinal de referência Gnd que pode ser de entrada e saída. O sinal de saída 5V disponibiliza uma tensão de alimentação para circuitos externos de 5V. O sinal 3V3 que permite a alimentação de circuitos externos com uma tensão de alimentação de 3,3V. E por fim o sinal de Reset que permite a reinicialização do processador. A placa tem um interruptor S1 para fazer também Reset ao microcontrolador.

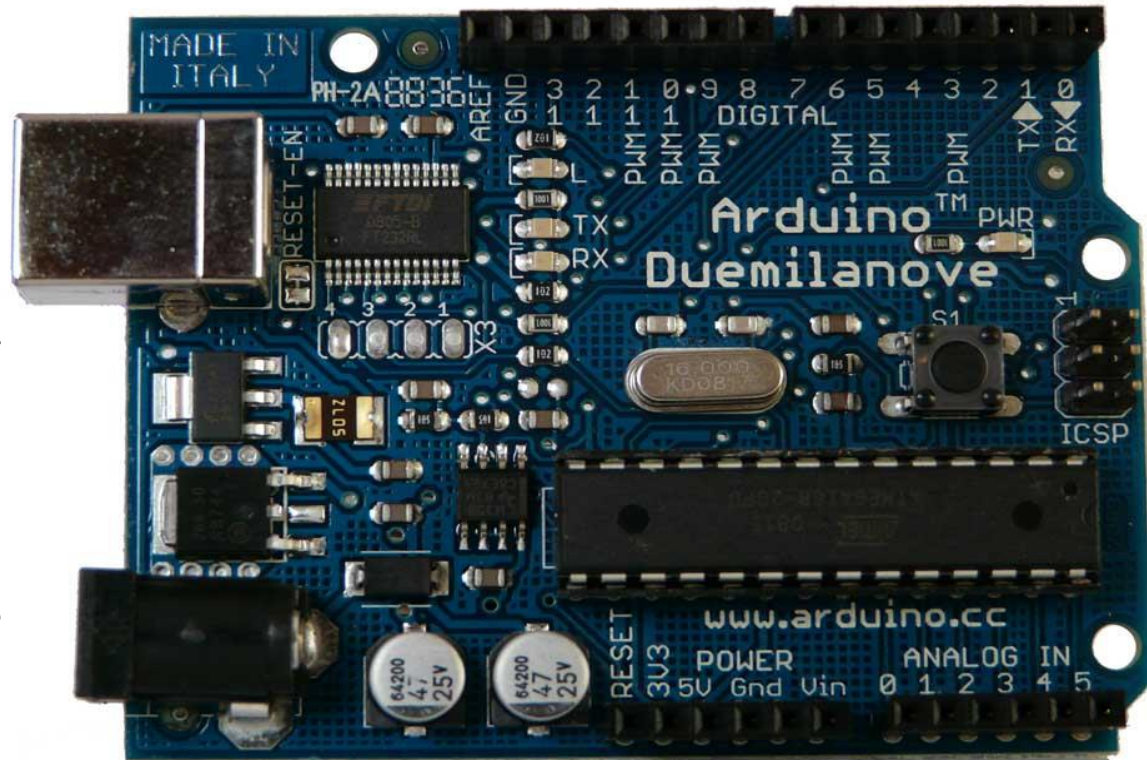


Figura 1.1 – Placa Arduino Duemilanove.

Esta placa pode ser alimentada via usb através da ficha existente na parte superior esquerda da placa ou através dum transformador externo de 9V ligada à ficha existente na parte inferior esquerda da placa.

Computação Física

2. Instalação do ambiente de programação

O ambiente de programação pode ser descarregado do sítio www.arduino.cc/en/Main/Software escolhendo a versão adequada ao sistema operativo do seu computador.

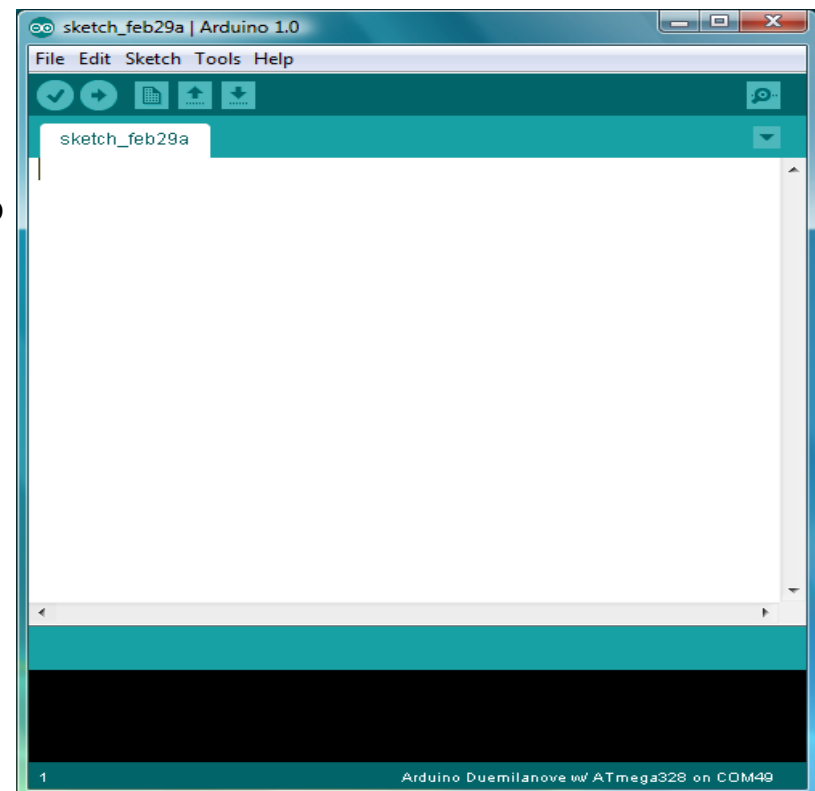
Depois de descarregar o ambiente no seu computador, aparece um ficheiro executável com o nome `arduino` e com o seguinte símbolo:



Fazendo duplo click em cima deste símbolo, executará o ambiente de programação que terá o seguinte aspeto:

Após ter executado com sucesso o ambiente de programação, tem de seleccionar qual a placa arduino com que está a trabalhar.

Depois de configurada a placa, ao ligar o arduino ao PC via cabo USB, é normal que o ambiente de programação não reconheça a placa, pois é necessário configurar o porto de comunicação série para que o ambiente de programação que está a ser executado no PC consiga dialogar com o arduino.

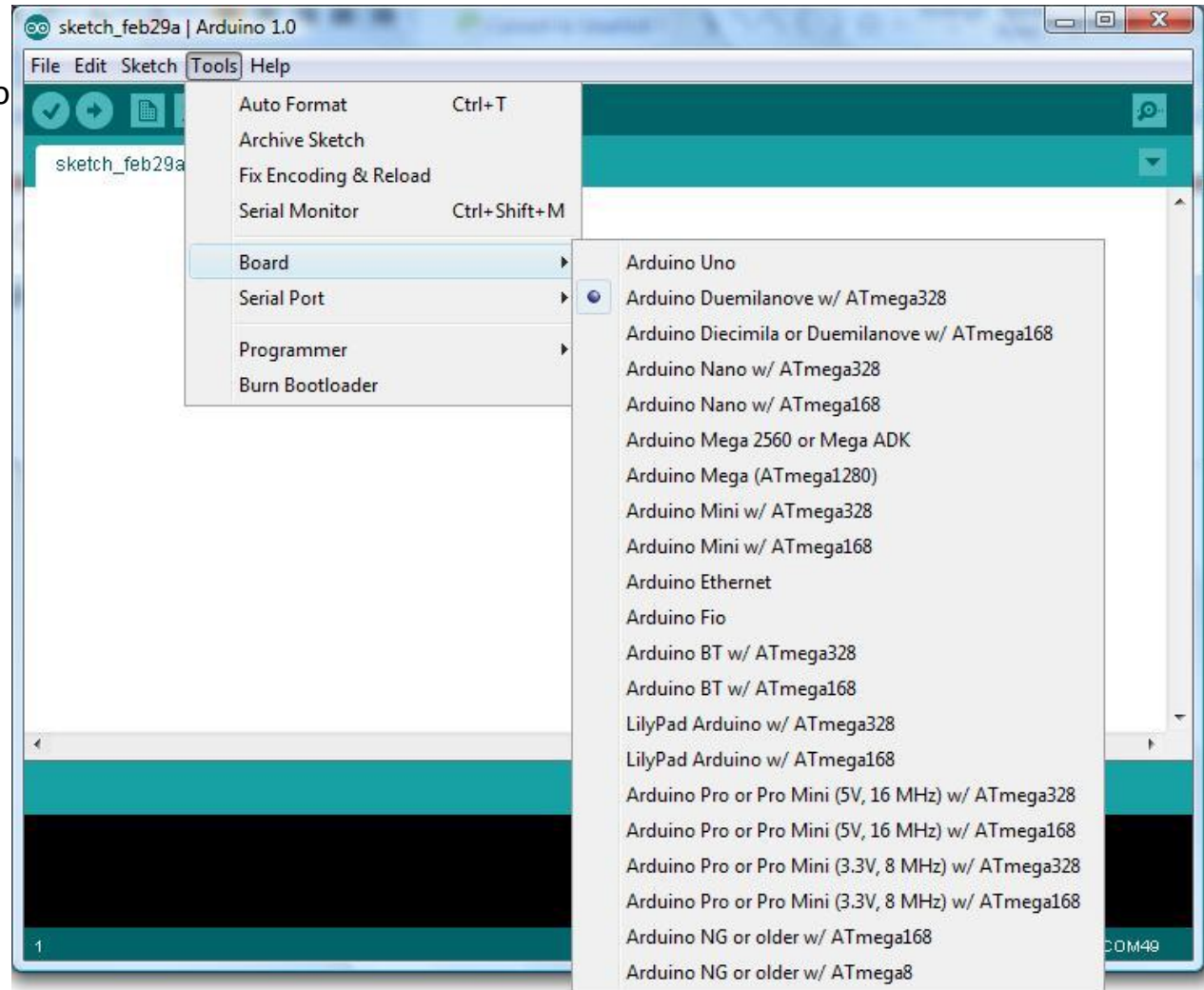


Computação Física

2.1. Configuração da placa

A configuração da placa arduino está ilustrada na figura e é realizada na opção de menu Tools->Board.

Na opção selecionada aparece uma bola azul tal como ilustrado na figura ao lado.



Computação Física

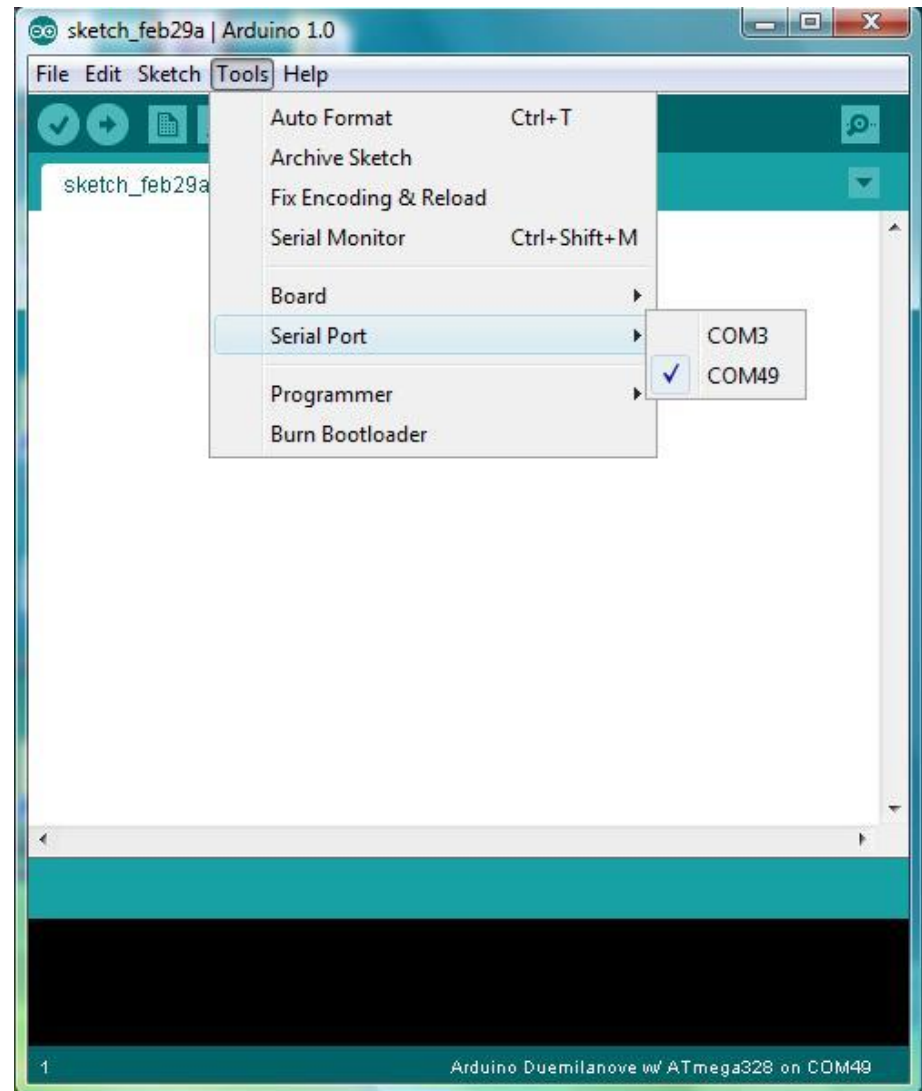
2.2. Configuração do porto série

A configuração do porto série é ilustrada na figura ao lado e é feita através da opção de menu Tools->Serial Port .

Vai aparecer, tal como na figura ao lado, várias portas COM, (COM3, COM49, etc) e vai escolhendo uma COM até ser a adequada.


Como saber qual é a COM adequada?

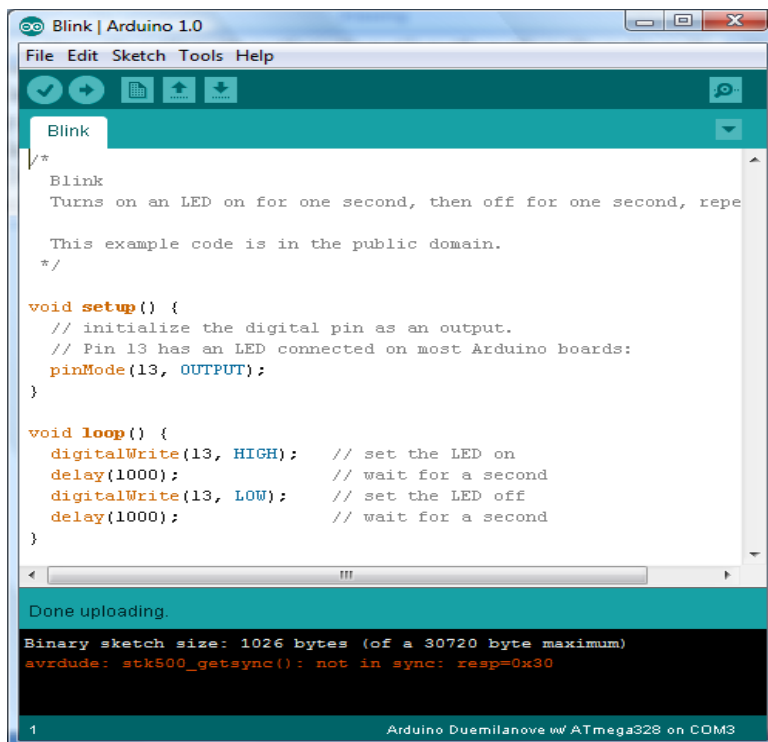
Veja a página seguinte para obter a resposta!



Computação Física

Para saber qual a COM adequada, siga os seguintes passos:

1. Selecione a opção File->Examples->Basics->Blink e clique em cima de Blink. Aparece no editor de texto o código deste programa, tal como na figura ao lado.
2. Carregue no botão  para compilar o programa e descarregar o programa no arduino.
3. Depois de cumprido o ponto 2, tem duas janelas possíveis mostradas abaixo. A janela da esquerda tem na consola uma mensagem de erro, portanto a COM não é a adequada. A janela da direita indica sucesso no descarregar do programa portanto a COM seleccionada é a correcta.



The screenshot shows the Arduino IDE interface with the 'Blink' sketch loaded. The status bar at the bottom indicates 'Arduino Duemilanove w/ ATmega328 on COM3'. The console at the bottom displays an error message in orange text: 'avrduide: stk500_getsync(): not in sync: resp=0x30'. The status bar also shows 'Done uploading.' and 'Binary sketch size: 1026 bytes (of a 30720 byte maximum)'.

```
File Edit Sketch Tools Help
Blink
/*
 * Blink
 * Turns on an LED on for one second, then off for one second, repeatedly.
 *
 * This example code is in the public domain.
 */

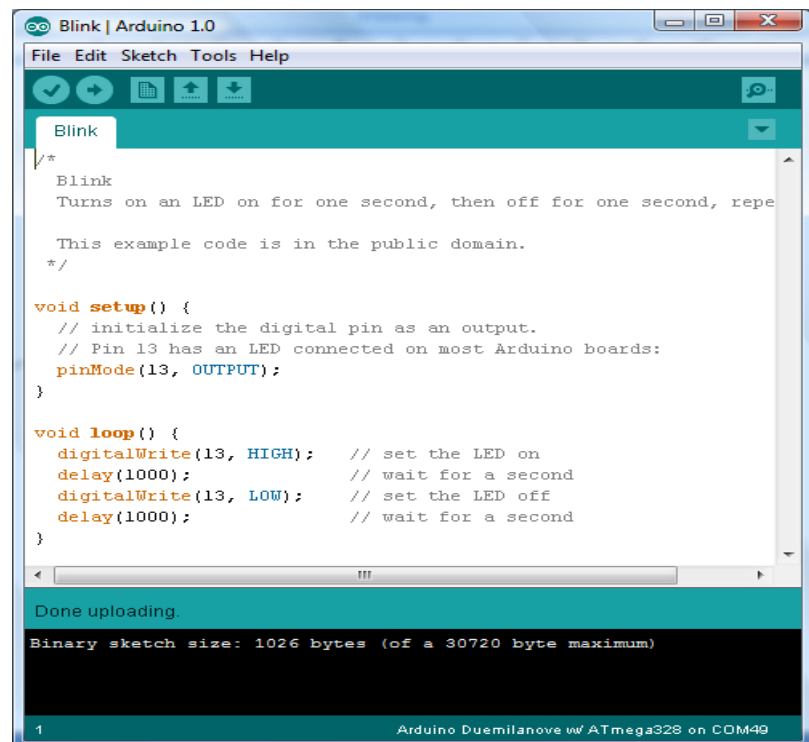
void setup() {
  // initialize the digital pin as an output.
  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH); // set the LED on
  delay(1000);             // wait for a second
  digitalWrite(13, LOW);  // set the LED off
  delay(1000);             // wait for a second
}

Done uploading.
Binary sketch size: 1026 bytes (of a 30720 byte maximum)
avrduide: stk500_getsync(): not in sync: resp=0x30

1 Arduino Duemilanove w/ ATmega328 on COM3
```

COM 3 seleccionada e incorreta.
Com mensagem de erro cor de laranja.



The screenshot shows the Arduino IDE interface with the 'Blink' sketch loaded. The status bar at the bottom indicates 'Arduino Duemilanove w/ ATmega328 on COM49'. The console at the bottom displays the message 'Done uploading.' in green text. The status bar also shows 'Binary sketch size: 1026 bytes (of a 30720 byte maximum)'.

```
File Edit Sketch Tools Help
Blink
/*
 * Blink
 * Turns on an LED on for one second, then off for one second, repeatedly.
 *
 * This example code is in the public domain.
 */

void setup() {
  // initialize the digital pin as an output.
  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH); // set the LED on
  delay(1000);             // wait for a second
  digitalWrite(13, LOW);  // set the LED off
  delay(1000);             // wait for a second
}

Done uploading.
Binary sketch size: 1026 bytes (of a 30720 byte maximum)

1 Arduino Duemilanove w/ ATmega328 on COM49
```

COM 40 seleccionada e correcta.

Computação Física

3. Um programa na linguagem arduino

A estrutura de um programa em arduino baseia-se em dois métodos, o método `setup()` que é onde se faz todas as inicializações da aplicação e o método `loop()` que, tal como o nome indica, é um ciclo infinito onde deve decorrer a execução da aplicação, propriamente dita.

Um programa em arduino tem a seguinte estrutura:

```
// inicializações – construtor da aplicação
setup()
{}
```

```
// programa em execução permanente
loop()
{}
```

No interior de cada um dos métodos anteriores podemos definir constantes, variáveis, estruturas de controlo e evocar funções intrínsecas à linguagem que vão desde funções aritméticas até funções de entrada e saída de informação e uma vasta gama de outras funções disponíveis em bibliotecas já desenvolvidas e colocadas disponíveis de forma grátis na net.

Computação Física

3.1. Constantes

HIGH - Num sinal de entrada digital significa uma tensão de entrada superior a 3V. Num sinal de saída digital significa colocar o pino a 5V.

LOW – Num sinal de entrada digital significa uma tensão de entrada inferior a 2V. Num sinal de saída digital significa colocar o pino a 0V.

INPUT – valor que define um pino como sinal de entrada quando passado como segundo parâmetro à função `pinMode(pino, modo)`.

OUTPUT – valor que define um pino como sinal de saída como segundo parâmetro da função `pinMode(pino, modo)`.

true - qualquer valor diferente de zero.

false - o valor zero.

integer – são os números 0, 1, 2, ..., 10, etc. A base decimal é a base por defeito. Um número em base 2 é precedido pela letra B, por exemplo B1001. Um número em base octal é precedido pelo número 0, por exemplo 0734. Um número em base hexadecimal é precedido pelas letras 0x (zero, x), por exemplo 0xFF.

Computação Física

3.2. Tipos de Variáveis

`void` – é usada para definir um tipo de função que não retorna valor nenhum.

`boolean` – ocupa um byte de memória e pode ter o valor `true` ou `false`.

`char` – ocupa um byte de memória e é definido entre comas, por exemplo `'A'`.

`unsigned char, byte` – ocupa um byte de memória e contém valores entre 0 e 255.

`int` – ocupa 2 bytes de memória e é interpretado em código relativo.

`unsigned int, word` – ocupa 2 bytes de memória e é interpretado em código absoluto.

`long` – ocupa 4 bytes em memória e a gama de valores é de -2147483648 até 2147483647. Um número deste tipo é definido com a letra `L` após definido o número, por exemplo `123L`.

`unsigned long` – ocupa 4 bytes em memória só em código absoluto $[0, 2^{32}-1]$.

`float, double` – ocupam 4 bytes e têm uma gama de valores desde -3.4028235E+38 até 3.4028235E+38.

`string (char array)` – conjunto de caracteres, por exemplo `char msg[] = "ola"` ou `char str[] = {'o', 'l', 'a', '\0'}`

`String (object)` – é uma classe que já tem um conjunto de métodos definidos.

`array` - é um conjunto de tipos simples, por exemplo um array de dois inteiros `int i[2]` ou um array de caracteres `char ola[4] = "ola"`.

Computação Física

3.3. Estruturas de Controlo de Fluxo

A sintaxe das várias estruturas de controlo de fluxo da linguagem são:

- if
if (condição) instrução ou if (condição) { instruções }
- if..else
if (condição) { instruções } else { instruções }
- for
for (inicialização ; condição ; iteração) { instruções }
- switch case
switch (valor) { case valor1: instruções break; case valor2: instruções break; default: instruções }
- while
while (condição) { instruções }
- do... while
do { instruções } while (condição);
- break
break; // é utilizado para interromper um ciclo de repetição while, do while e for.
- continue
continue; // é utilizado num ciclo de repetição para não executar mais nenhuma instrução
- return
return; ou return valor; // permite terminar uma função não retornando ou retornando um valor
- goto
label: instruções goto label; // faz uma quebra de sequencialidade na execução do programa continuando a execução do programa na instrução após a definição da label: .

Computação Física

3.4. Símbolos sintáticos

`;` - fim de uma instrução

`{}` – início de um conjunto com 1 ou mais instruções.

`//` - início de uma linha de comentário

`/* */` - início e fim de um bloco de texto de comentário.

`#define` – define símbolos como constantes.

A sintaxe é:

`#define símbolo valor`

Por exemplo,

`#define pinoLed 3`

`#include` – permite incluir bibliotecas no ficheiro para utilização dos tipos e métodos definidos na biblioteca

A sintaxe é:

`#include <nomeBiblioteca>`

Por exemplo,

`#include <servo>`

Computação Física

3.5. Funções de conversão

char() – aplica-se a qualquer tipo convertendo o tipo para char truncando o seu valor aos 8 bits de menor peso.

byte() – aplica-se a qualquer tipo truncando o valor do tipo a 8 bits sem sinal.

int() – converte qualquer tipo para um valor a 16 bits com sinal.

word() – converte qualquer tipo para um valor a 16 bits sem sinal.

long() – converte qualquer tipo no tipo long.

float() – converte qualquer tipo em float.

3.6. Operadores Aritméticos

= – operador de afetação.

+ – operador de adição.

- – operador de subtração.

* – operador de multiplicação.

/ – operador de divisão.

% -operador de resto .

3.7. Operadores Relacionais

== – operador de igualdade.

!= – operador de diferença.

> – operador de maior.

>= – operador de maior ou igual.

< – operador de menor.

<= – operador de menor ou igual.

Computação Física

3.8. Funções de entrada/saída para aplicar aos sinais digitais, pinos 0 a 13

`pinMode(pino, modo)` – é uma função que define o modo de funcionamento de um sinal digital, o pino é um valor entre 0 e 13 e o modo é INPUT ou OUTPUT.

`digitalWrite(pino, valor)` – é uma função que coloca um valor num sinal digital, o pino é um valor entre 0 e 13 e o valor ou é HIGH ou LOW.

`valor digitalRead(pino)` – é uma função que lê um pino digital de entrada e devolve o valor HIGH ou LOW, respetivamente 1 ou 0.

3.9. Funções entrada/saída para aplicar aos sinais analógicos, pinos 0 a 5 e 9 a 11

`analogReference(tipo)` – o parâmetro tipo define o valor de tensão máximo da conversão que podem ser:

DEFAULT – tensão de alimentação do processador 3,3V (nas placas de 3,3V) ou 5V (nas placas de 5V).

INTERNAL – tensão de 1,1V no atmega328.

EXTERNAL – tensão aplicada no pino AREF. Atenção que existe uma resistência de 32K interna ao arduino.

`int analogRead(pino)` – Existem 6 pinos (A0 a A5) que contêm conversores analógico-digitais de 10 bits, assim a função devolve um valor do tipo int na gama [0, 1023].

`analogWrite(pino, valor)` : PWM - esta função gera um sinal PWM com uma frequência de 490Hz num dos pinos 3, 5, 6, 9, 10 ou 11. O valor define o duty cycle produzido, 0- 0% (off) e 255- 100%(on).

Computação Física

3.10. Outras funções de entrada/saída

`tone(pino, frequência)` – gera no pino especificado uma onda com a frequência especificada em hertz e com 50% de duty cycle. O parâmetro frequência é do tipo `unsigned int`.

`tone(pino, frequência, duração)` – idêntico ao anterior mas com a possibilidade de definir a duração temporal do tom em milissegundos. O terceiro parâmetro da função é do tipo `long`.

`noTone(pino)` – termina a geração de onda no pino especificado.

3.11. Funções de tempo

`unsigned long millis()` – devolve o tempo em milissegundos desde que a placa começou a funcionar. O contador volta a zero ao fim de 50 dias.

`unsigned long micros()` – devolve o tempo em microsegundos desde que a placa começou a funcionar. O contador volta a zero ao fim de 70 minutos.

`delay(unsigned long milissegundos)` – espera a quantidade de milissegundos especificada no parâmetro.

`delayMicroseconds(unsigned long microsegundos)` – espera a quantidade de microsegundos especificado no parâmetro.

Computação Física

3.12. Funções matemáticas

tipo min(tipo x, tipo y) – devolve o mínimo de dois números de qualquer tipo.

tipo max(tipo x, tipo y) – devolve o máximo de dois números de qualquer tipo.

tipo abs(tipo x) – devolve o valor absoluto de um número de qualquer tipo.

tipo constrain(tipo x, tipo a, tipo b) – devolve x se tiver na gama entre a e b. Se x menor que a devolve a. Se x maior que b devolve b.

double pow(float base, float expoente) – devolve a base elevado ao expoente.

double sqrt(tipo x) – devolve um double que é a raiz quadrada dum número x de qualquer tipo.

3.13. Operadores booleanos

&& - and lógico.

|| - or lógico.

! – not lógico.

3.14. Funções trigonométricas

double sin(float radianos) – calcula o seno de um ângulo passado em radianos.

double cos(float radianos) – calcula o coseno de um ângulo passado em radianos.

double tan(float radianos) – calcula a tangente de um ângulo passado em radianos.

Computação Física

3.15. Operadores de bit

`&` – função and entre dois valores bit a bit. Por exemplo, a expressão `(B1010 & B1100)` resultaria em `B1000`.

`|` – função or entre dois valores bit a bit. Por exemplo, a expressão `(B1010 | B1100)` resultaria em `B1110`.

`^` – função xor entre dois valores bit a bit. Por exemplo, a expressão `(B1010 ^ B1100)` resultaria em `B0110`.

`~` – função unária not bit a bit. Por exemplo, a expressão `~B1010` resultaria em `B0101`.

`<<` – função de deslocamento à esquerda(shift left) de um valor. Por exemplo, a expressão `(B1010 << 2)` resultaria em `B101000`.

`>>` – função de deslocamento à direita (shift right) de um valor. Por exemplo, a expressão `(B1010 >> 2)` resultaria em `B10`.

3.16. Funções de bit e byte

`byte lowByte(número)` – função que devolve o byte de menor peso de um número.

`byte highByte(número)` – função que devolve o byte de maior peso de um número

`boolean bitRead(número, bit)` – devolve o valor (0 ou 1) presente no bit de um número . Os valores do parâmetro `bit` são 1 – bit 0, 2- bit 1, 4 bit 2, etc.

`bitWrite(número, bit, valor)` – escreve no bit do número o valor passado no terceiro parâmetro que é 0 ou 1.

`bitSet(número, bit)` – coloca o bit de um número a 1

`bitClear(número, bit)` – coloca o bit de um número a 0.

Computação Física

3.17. Operadores compostos

`++` – incrementa de uma unidade o valor de uma variável.

Por exemplo,

`x++;` // retorna o valor antigo de `x` incrementando-o de 1.

`++x;` // incrementa o valor de `x` de uma unidade devolvendo o novo valor.

`--` – decrementa de uma unidade o valor de uma variável.

Por exemplo,

`y--;` // decrementa o valor de `y` de 1 devolvendo o valor antigo.

`--y;` // decrementa o valor de `y` de 1 devolvendo o novo valor.

`+=` – adição com afetação. Por exemplo, a instrução `y+=2;` é equivalente a `y= y +2;` .

`-=` – subtração com afetação. Por exemplo, a instrução `y+=x;` é equivalente a `y= y +x;` .

`*=` – multiplicação com afetação. Por exemplo, a instrução `y*=y+x;` é equivalente a `y= y*(y +x);` .

`/=` – divisão com afetação. Por exemplo, a instrução `y/=2;` é equivalente a `y= y/2;` .

`&=` – função lógica and bit a bit com afetação.

`|=` – função lógica or bit a bit com afetação.

Computação Física

Destacável Linguagem Arduino

Estrutura		Constantes		Comparação	Tipos		Tempo		Matemática		Analógico I/O	Digital I/O	Serial	
setup()		HIGH	LOW	== != < >	void	int	char	millis()	micros()	min()	max()	analogReference()	pinMode()	begin()
loop()		INPUT	OUTPUT	>= <=	boolean		word	delay()		abs()	map()	analogRead()	digitalWrite()	end()
Controlo		true	false	Op. booleano	long	float		delayMicroseconds()		pow()		analogWrite()	digitalRead()	available()
if	if... else	Op. sintáticos		&& !	Unsigned char		Conversão		constraint()		Avançado I/O	Int.s Externas	read()	
for	while	; {} // /* */		Op. bit	unsigned int		char()	byte()	Bit e Byte		tone()	attachInterrupt()	peek()	
switch case		#define		& ^ ~ >> <<	unsigned long		int()	word()	lowByte()	highByte()	noTone()	detachInterrupt()	flush()	
do... while		#include		Op. composto	double		long()	float()	bitRead()		shiftOut()	Interrupções	print()	
break	continue	Op. aritméticos		++ -- += -=	String	array		Trigonometria		bitWrite()		shiftIn()	interrupts()	println()
return	goto	= + - * / %		*= /= &= =	string		sin()	cos()	tan()	bitSet()	bitClear()	pulseIn()	noInterrupts()	write()

Computação Física

Capítulo 5 – Diagramas de atividade e autómatos

1. Diagramas de atividade da UML para modelar sistemas de computação física

O desenho de sistemas de computação física pode ser decomposto em vários níveis. A um nível funcional ou comportamental, o sistema necessita duma descrição ao longo do tempo, podendo para o efeito utilizar-se os diagramas de estado da UML(Unified Modelling Language), e também precisa duma descrição de sincronização entre os vários comportamentos, podendo para tal, utilizar-se os **diagramas de atividades** da *UML(Unified Modelling Language)*.

As vantagens da utilização duma linguagem gráfica são fundamentalmente, a independência da plataforma, da tecnologia, da linguagem de programação. E principalmente, conseguir reunir um conjunto de investigadores, com formações diversas, para discutir o desenho do modelo duma solução para um determinado sistema de computação física.

Como não existe uma linguagem gráfica standard para modelar sistemas de computação física, vai-se utilizar na disciplina a Unified Modelling Language, e em particular os **diagramas de actividade** para se atingir o objetivo proposto.

1.1. Diagramas de atividade adaptados à computação física

Apesar dos diagramas de atividade terem sido desenhados inicialmente para representar graficamente um fluxo de atividades duma aproximação orientada por objectos. No entanto, é possível a adaptação dos diagramas de atividade a uma linguagem na área da computação física.

Os diagramas de atividade permitem modelar tarefas como um conjunto de atividades e transições entre duas atividades, assim como definir a comunicação entre as tarefas.

Um diagrama de atividade pode representar operações, métodos, processos, paralelismo de processos, sincronização e colaboração entre processos, transferência de objectos entre processos e especificação dos objectos a transferir.

Computação Física

1.2. Estados

Os diagramas de atividade contêm dois tipos de estado, estados de ação e estados de subatividade.

1.2.1. Estados de Ação

Os estados de ação são representados por retângulos com os vértices redondos, em que no interior da caixa está representada a expressão da ação que indica qual a função da ação. A expressão da ação pode ser escrita em português ou em pseudo-código numa linguagem de programação. Se a expressão da ação for descrita em português deve começar por um verbo seguido da função. Exemplos de estados de ação:



Um estado de ação obedece às seguintes características:

- Atómico – não pode ser decomposta noutro conjunto de ações;
- Ininterrompível – não pode ser interrompido desde que começa até que acaba;
- Instantâneo – o trabalho e o tempo de execução são insignificantes.

Cada diagrama de atividade tem dois símbolos especiais – um símbolo inicial que está ligado ao estado que é definido como estado inicial e outro símbolo designado por símbolo final que está ligado ao estado final do diagrama. O símbolo inicial marca o início do fluxo de actividade e o símbolo final representa o fim do fluxo de actividade. A representação destes dois símbolos são os seguintes:



Símbolo inicial



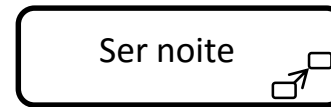
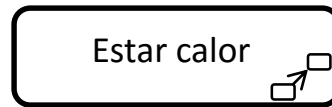
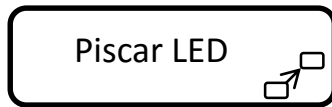
Símbolo final

Computação Física

1.2.2. Estados de Subatividade

Os estados de subatividade são não-atômicos, significando que podem ser decompostos em estados de acção ou em estados de subatividade. Estes estados podem ser interrompidos, e a sua execução deve demorar um tempo finito.

Exemplos de estados de subatividade:

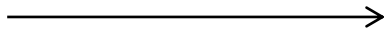


1.3. Transição entre estados

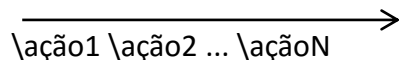
Quando uma acção ou subatividade termina a sua função existe uma transição automática do estado actual para outro estado, representada por uma seta \rightarrow .

Existem quatro tipos de transições distintas e que são representadas do seguinte modo:

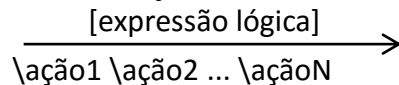
- Transição incondicional sem acções



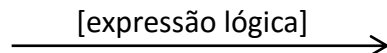
- Transição incondicional com acções



- Transição condicional com acções



- Transição condicional sem acções



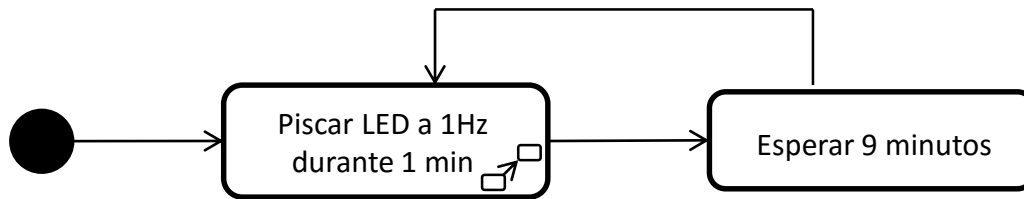
Computação Física

1.4. Aplicação

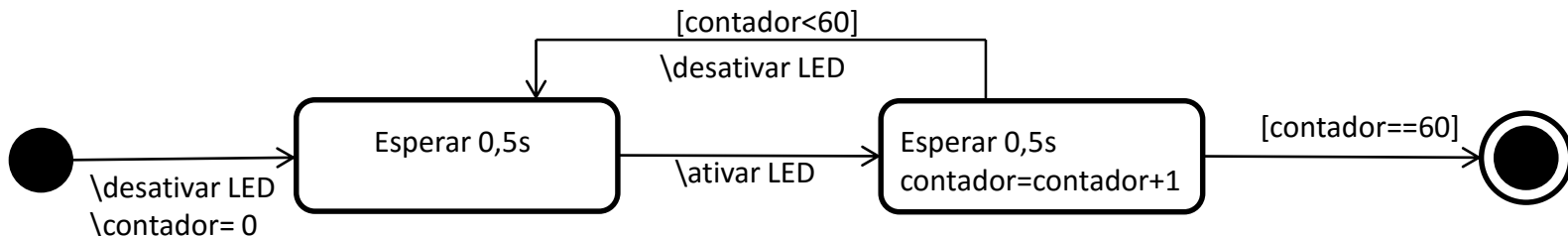
Vamos exemplificar a utilização dos diagramas de atividade na resolução do seguinte problema:

Para garantir a segurança nas suas viagens longas de carro, o Sr. Joaquim pretende instalar no seu carro um sistema inteligente visual antisono nocturno: um LED de alto brilho que pisca a uma frequência de 1 Hz durante 1 minuto de dez em dez minutos só durante a noite.

Um diagrama de atividades para solução do problema será:



Um estado de subatividade pode ser descrito por um ou mais diagramas de atividades até um nível em que todas as atividades sejam atividades de ação atômicas e indevisíveis.



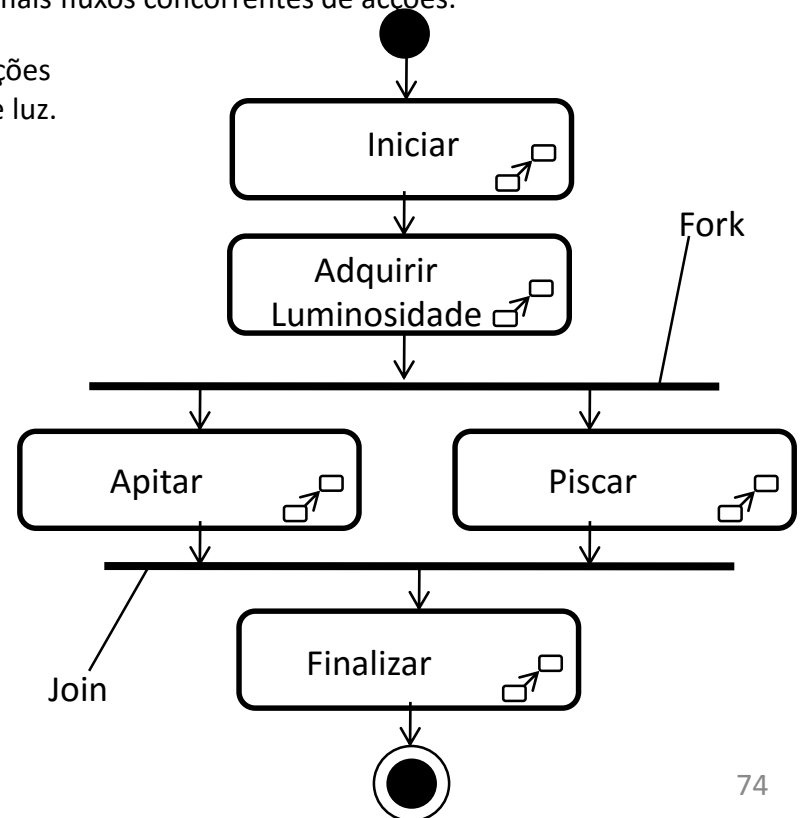
Computação Física

1.5. Fork e Join

Os diagramas de atividade modelam com simplicidade fluxos concorrentes de acções. Pode-se dividir um fluxo de acções em dois ou mais fluxos paralelos de acções utilizando um *fork*, e depois pode-se sincronizar estes fluxos paralelos de acções num *join*. Um *fork* tem exactamente uma transição como entrada e duas ou mais transições como saídas. Um *join* tem duas ou mais transições como entrada e exactamente uma transição como saída. A transição de saída só é cumprida quando todas as transições de entrada estiverem executadas, ou seja, quando todos os fluxos concorrentes tiverem terminado a sua actividade. Portanto, um *join* é um ponto de sincronização entre dois ou mais fluxos concorrentes de acções.

No exemplo seguinte apresenta-se um modelo simples de fluxos de acções concorrentes, que permitem modelar um sistema anti-sono com som e luz.

A aplicação começa por inicializar a infra-estrutura do sistema que são os sensores e atuadores. Depois, fará a aquisição da informação do ambiente circundante para definir se é noite ou de dia. Depois lançará dois automatismos concorrentes, um automato é responsável pela emissão de um apito sonoro e outro automato é responsável por piscar um LED de alto brilho. Quando estes dois automatismos terminarem a execução, a aplicação pára.



Computação Física

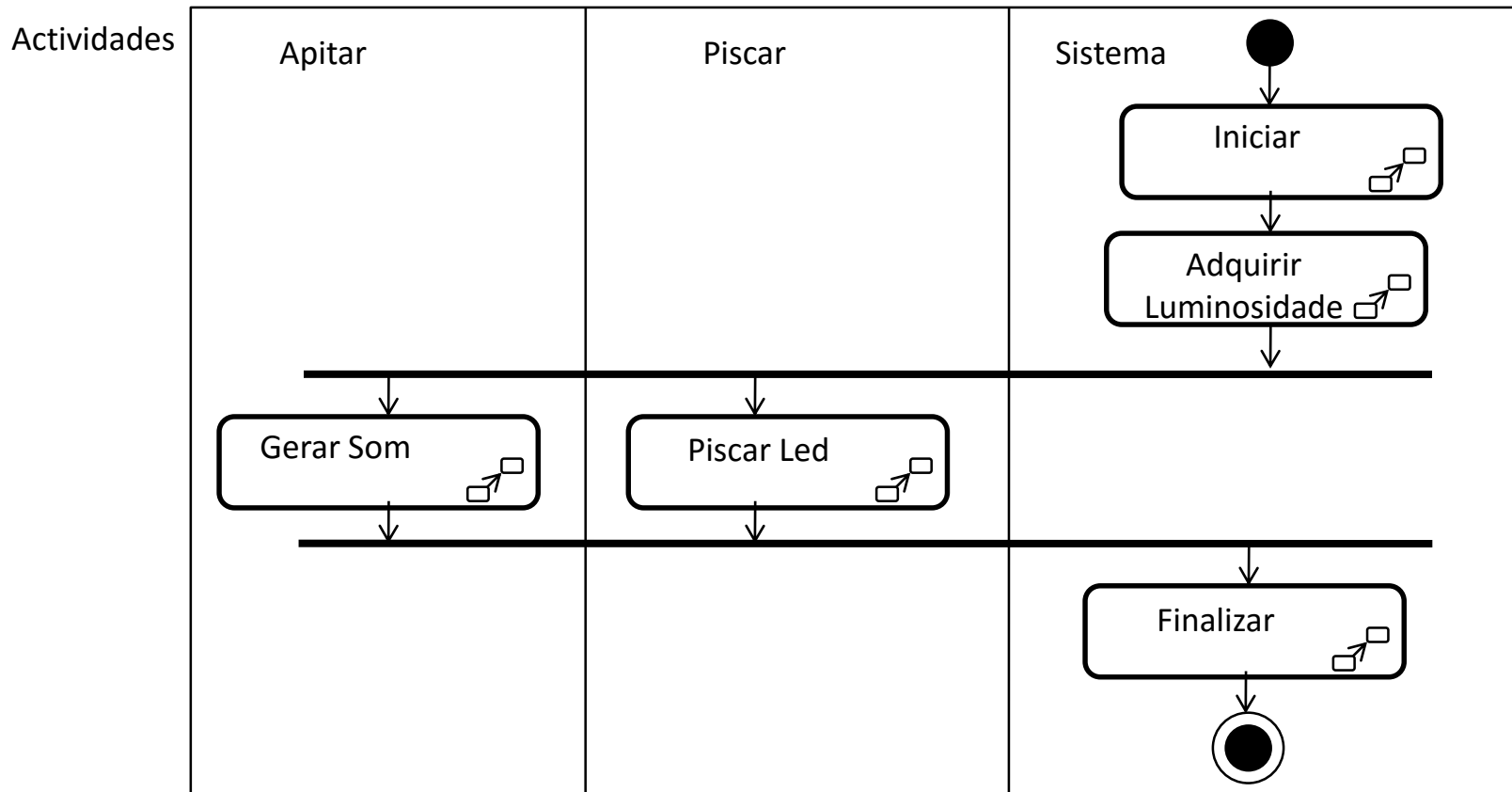
1.6. Swimlanes

As swimlanes servem para fazer a partição dos diagramas de atividade da forma mais adequada! Esta partição definirá as unidades funcionais, designadas como processos que serão implementados com autómatos.

Numa dada aplicação, a partição em swimlanes não é única, podendo haver outras possibilidades de partição.

Em sistemas distribuídos, as swimlanes também são utilizadas para definir o modelo de distribuição de autómatos através de diferentes computadores.

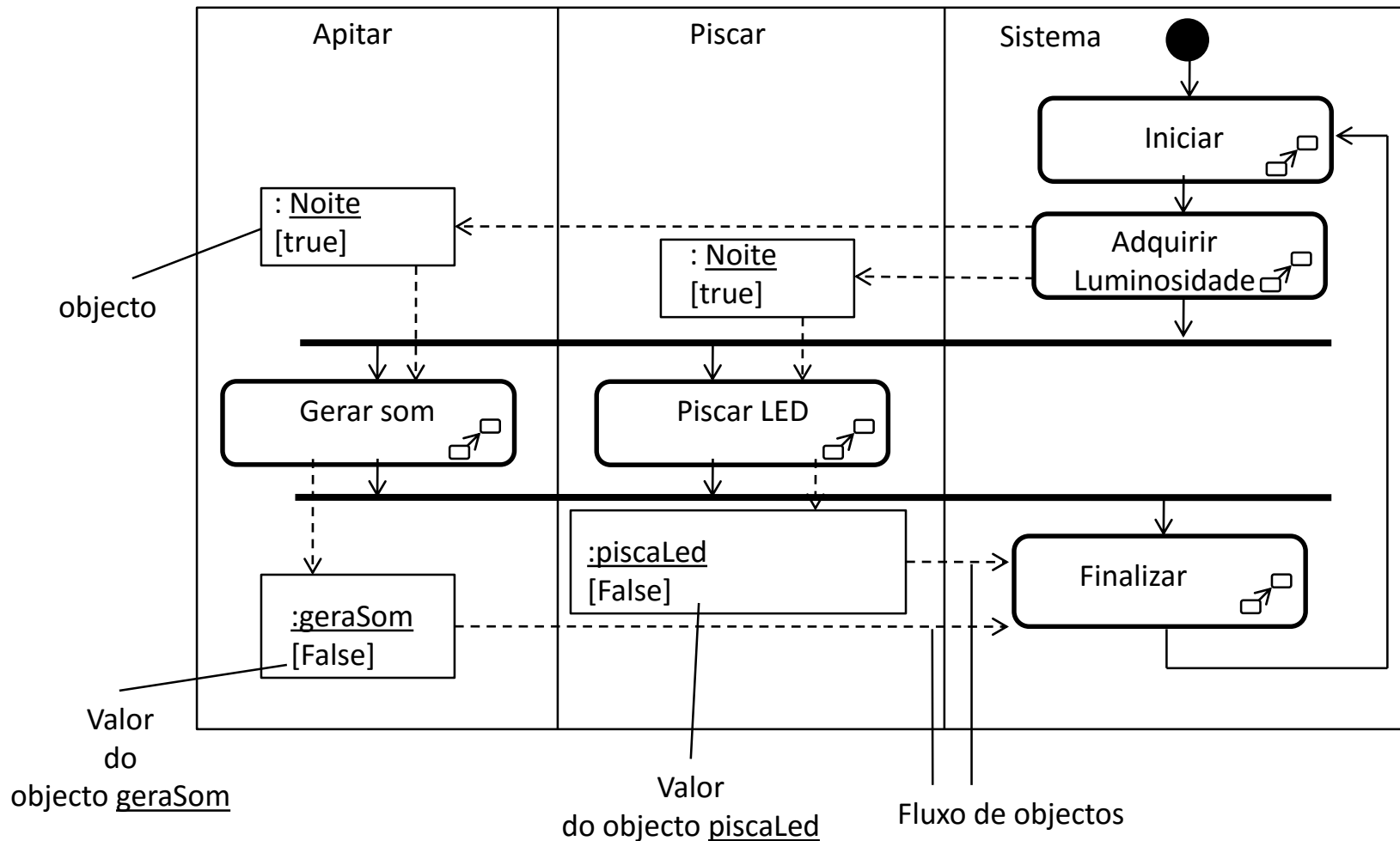
Na seguinte figura, define-se as swimlanes que representam três autómatos a lançar no sistema anti-sono.



Computação Física

1.7. Fluxo de objectos

As atividades podem receber e enviar objectos e até podem modificar o estado dos objectos. Esta transferência de objectos entre as atividades são representadas no seguinte diagrama de atividade.



Computação Física

1.7. Implementação das swimlanes e o fluxo de objectos

Na implementação, uma swimlane é um autómato e um objeto é uma variável visível aos autômatos. O *fork* vai originar um estado adicional em todos os autômatos tal como o grafismo *join* também originará um novo estado, esta é a maneira mais direta de implementar o modelo, mas não é a implementação mais simples.

```
// comunicação de luminosidade
boolean noite;
// variável de estado e valores
int estadoApitar;
int ForkApitar= 0;
int GerarSom= 1;
int JoinApitar= 2;
// comunicação de fim de som
boolean geraSom;
void Apitar() {
    switch (estadoApitar) {
        case ForkApitar:
            if (noite) {
                geraSom= True;
                estadoApitar= GerarSom;
            }
            break;
        case GerarSom:
            if (!gerarSom())
                estadoApitar= JoinApitar;
            break;
        case JoinApitar:
            geraSom=False;
            break;
    }
}
```

```
// variável de estado e valores
int estadoPiscar;
int ForkPiscar= 0;
int Piscar= 1;
int JoinPiscar= 2;
// comunicação fim de piscar
boolean piscaLed;
void PiscarLed() {
    switch (estadoPiscar) {
        case ForkPiscar:
            if (noite) {
                piscaLed= True;
                estadoPiscar= Piscar;
            }
            break;
        case Piscar:
            if (!piscarLed())
                estadoPiscar= JoinPiscar;
            break;
        case JoinPiscar:
            piscaLed= False;
            break;
    }
}
```

```
// variável de estado e valores
int estadoSistema;
int Iniciar= 0;
int AdquirirLuz= 1;
int ForkSistema= 2;
int Finalizar= 3;
void Sistema() {
    switch (estadoSistema) {
        case Iniciar:
            AtividadeIniciar();
            estadoSistema= AdquirirLuz;
            break;
        case AdquirirLuz:
            noite= NaoHaLuz();
            if (noite)
                estadoSistema= Finalizar;
            break;
        case Finalizar:
            if (!piscaLed && !geraSom) {
                estadoSistema= Iniciar;
            }
            break;
    }
}
```

```
void AtividadeIniciar() {
    noite= False;
    estadoApitar= ForkApitar;
    estadoPiscar= ForkPiscar;
}
```

```
int T= 512; // >= 2,5V - noite
boolean NaoHaluz() {
    return digitalRead(A0)>=T;
}
```

```
void setup() {
    noite= False;
    geraSom= False;
    piscaLed= False;
    estadoSistema= Iniciar;
    estadoPiscar= ForkPiscar;
    estadoApitar= ForkApitar;
}

void loop(){
    Apitar();
    PiscarLed();
    Sistema();
}
```

Computação Física

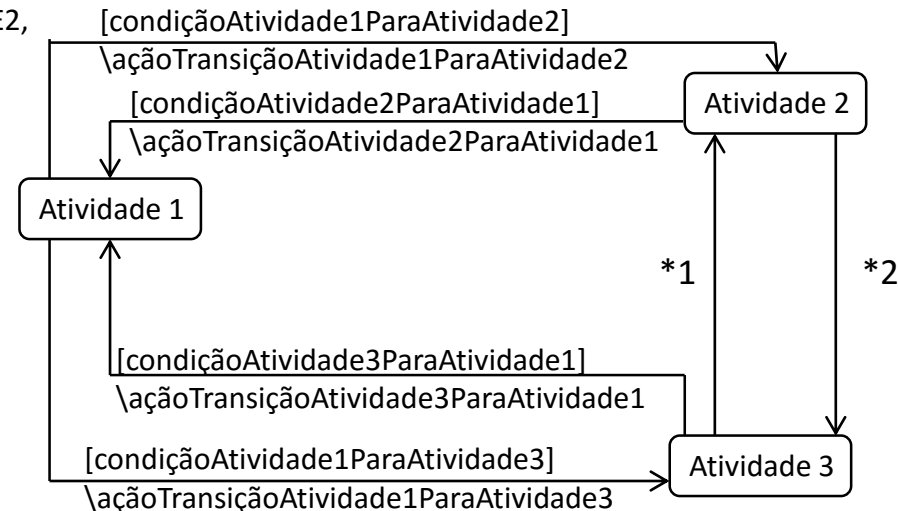
2. Autômato bloqueante

A estrutura genérica de um autômato bloqueante para implementar um diagrama de atividades com 3 atividades {ATIVIDADE1, ATIVIDADE2, ATIVIDADE3} em que todas as atividades estão ligadas a todas as atividades com uma condição é a seguinte:

```
void AutomatoGenérico() {  
    int estado;  
    estado= ATIVIDADE1;  
    while (true) {  
        switch (estado) {  
            case ATIVIDADE1:atividade1(); // ações a efetuar  
                if (condiçãoAtividade1ParaAtividade2){ // condição para a Atividade2  
                    açãoTransiçãoAtividade1ParaAtividade2();  
                    estado=ATIVIDADE2; break;  
                }  
                if (condiçãoAtividade1ParaAtividade3){ // condição para a Atividade3  
                    açãoTransiçãoAtividade1ParaAtividade3();  
                    estado= ATIVIDADE3; break;  
                }  
            }  
        }  
    }
```

// continua na próxima folha

Diagrama de atividades com 3 atividades



*1 [condiçãoAtividade3ParaAtividade2]
\\açãoTransiçãoAtividade3ParaAtividade2

*2 [condiçãoAtividade2ParaAtividade3]
\\açãoTransiçãoAtividade2ParaAtividade3

Computação Física

```
case ATIVIDADE2:atividade2(); // actividades a efetuar no estado 2
    if (condiçãoAtividade2ParaAtividade1){ // inicio da transição –condição para ir para a ATIVIDADE1
        açãoTransiçãoAtividade2ParaAtividade1 ();
        estado= ATIVIDADE1; break;
    }
    if (condiçãoAtividade2ParaAtividade3){ // condição para a ATIVIDADE3
        açãoTransiçãoAtividade2ParaAtividade3 ();
        estado= ATIVIDADE3; break;
    }
case ATIVIDADE3:atividade3(); // actividades a efetuar no estado 3
    if (condiçãoAtividade3ParaAtividade1){ // condição para a ATIVIDADE1
        açãoTransiçãoAtividadeNParaEAtividade1 ();
        estado= ATIVIDADE1; break;
    }
    if (condiçãoAtividade3ParaAtividade2){ // condição para a ATIVIDADE2
        açãoTransiçãoAtividadeNParaAtividade2();
        estado= ATIVIDADE2; break;
    }
} // fecho do switch
} // fecho do forever
} // fecho da função.
```

Computação Física

3. Autômatos não bloqueantes

A implementação de um diagrama de atividades pode ser realizada por um autômato não bloqueante. A característica principal de um autômato não bloqueante é que o tempo de execução é finito, mínimo e limitado à execução das ações de uma atividade. Por exemplo, estes autômatos são adequados à implementação de funções de atendimento de interrupções onde o tempo de execução deve ser o mínimo possível.

3.1. Autômato não bloqueante sem estado final

Alterando a estrutura genérica do autômato bloqueante apresentado na secção 2., a estrutura equivalente de um autômato não bloqueante sem estado final para implementar um diagrama de atividades com 3 atividades em que todas as atividades estão ligadas a todas as outras atividades através de condições de transição, é a seguinte:

```
int estado; // a variável de estado é global
void AutomatoNaoBloqueanteSemEstadoFinal() {
    switch (estado) {
        case ATIVIDADE1:atividade1(); // ações a realizar na atividade 1
            if (condiçãoAtividade1ParaAtividade2){
                açãoTransiçãoAtividade1ParaAtividade2();
                estado= ATIVIDADE2;
                return;
            }
            if (condiçãoAtividade1ParaAtividade3){
                açãoTransiçãoAtividade1ParaAtividade3();
                estado= ATIVIDADE3;
            }
        return;                // continua na próxima folha
```


Computação Física

```
case ATIVIDADE2:atividade2();
    if (condiçãoAtividade2ParaAtividade1){
        açãoTransiçãoAtividade2ParaAtividade1();
        estado= ATIVIDADE1; return;
    }
    if (condiçãoAtividade2ParaAtividade3){
        açãoTransiçãoAtividade2ParaAtividade3 ();
        estado= ATIVIDADE3; return;
    }
    return;
case ATIVIDADE3:atividade3(); // ações a realizar no estado 3
    if (condiçãoAtividade3ParaAtividade1){
        açãoTransiçãoAtividade3ParaAtividade1 ();
        estado= ATIVIDADE1; return;
    }
    if (condiçãoAtividade3ParaAtividade2){
        açãoTransiçãoAtividade3ParaAtividade2();
        estado= ATIVIDADE2; return;
    }
    return;
} // fecho do switch
} // fecho da função.
```

Computação Física

3.2. Autômato não bloqueante com estado final

A estrutura genérica de um autômato não bloqueante com estado terminal de atividade difere do autômato não bloqueante sem estado terminal, porque passa a ser uma função que retorna a informação acerca se terminou ou não a atividade. Por exemplo, considere um diagrama de atividades com 2 atividades completamente ligadas, no qual uma das atividades é final, a estrutura deste autômato é a seguinte:

```
int estado; // a variável de estado é global
```

```
boolean AutomatoNaoBloqueanteComEstadoFinal() {  
    switch (estado) {  
        case ATIVIDADE1:  
            atividade1();  
            if (condiçãoParaAtividadeFinal){ // inicio da transição do Estado1– teste da condição para ir para o Estado2  
                açãoTransiçãoAtividade1ParaAtividadeFinal();  
                estado=ATIVIDADE_FINAL;  
            }  
            return false;  
        case ATIVIDADE_FINAL:  
            atividadeFinal(); // actividades a efetuar no estado final  
            if (condiçãoParaAtividade1){ // inicio da transição do Estado1– teste da condição para ir para o Estado2  
                açãoTransiçãoAtividadeFinalParaAtividade1();  
                estado=ATIVIDADE1;  
                return false;  
            }  
            return true;  
    } // fecho do switch  
} // fecho da função.
```

Computação Física

4. Interrupções no Arduino

O arduino tem duas entradas de interrupção, o pino digital 2 que corresponde à interrupção 0 e o pino digital 3 que corresponde à interrupção 1.

O objetivo da utilização das interrupções é o sistema estar sensível a vários sinais que podem ocorrer ou não em simultâneo, sem perder tempo de processamento a testar estes sinais de interrupção. Para isso, o microcontrolador tem de disponibilizar hardware que está sensível ao nível ou às transições ascendente ou descendente dos sinais de interrupção. Quando se gera um sinal de interrupção deve-se associar uma função de interrupção ao tratamento desse sinal. Por defeito, uma função de interrupção não é interrompível por outro sinal de interrupção.

O arduino disponibiliza um conjunto de métodos de interrupção que têm as seguinte funcionalidade:

`attachInterrupt(interrupção, função, modo)` - esta função ativa a interrupção externa especificada no primeiro parâmetro associando a esta fonte de interrupção a função especificada no segundo parâmetro, no terceiro parâmetro especifica-se o tipo da interrupção externa.

`detachInterrupt(interrupção)` – que desativa a interrupção especificada.

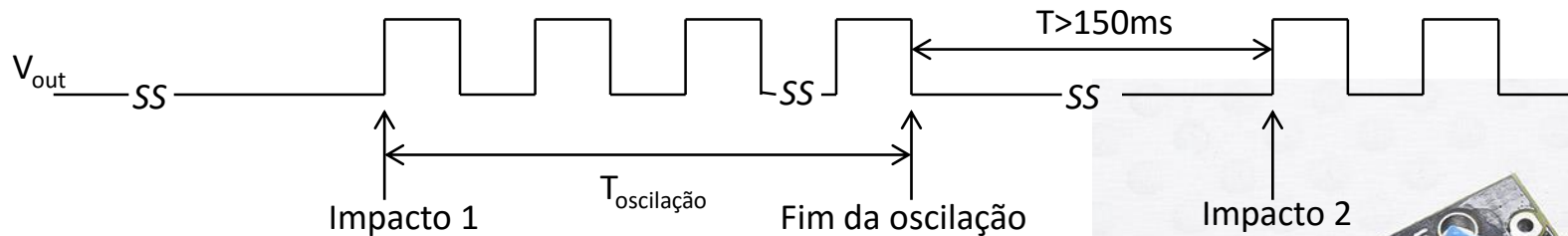
`Interrupts()` – ativa todas as interrupções no arduino.

`noInterrupts()` – desativa todas as interrupções.

Computação Física

5. Sensor de Impacto ou de *Tilt*

Um sensor de impacto ou de *tilt* é um sensor com saída digital, na qual há variações na saída entre 0 e 1 sempre que existe uma aceleração ou desaceleração instantânea, ver figura 5.1 . A tensão de alimentação é entre os 3V e 5V.



- Pino 1 - Vout - Sinal digital entre 0 e 5V;
- Pino 2 - Vcc - Tensão de alimentação entre 3V e 5V;
- Pino 3 - GND - 0V.

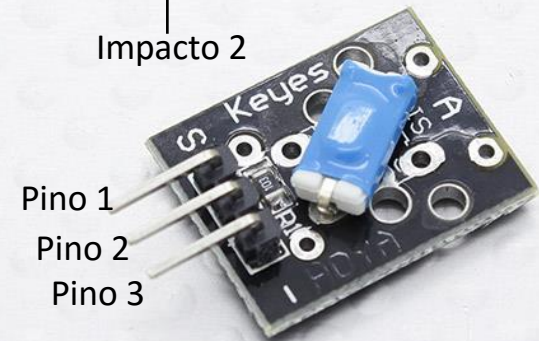


Figura 5.1 - Sensor de impacto ou de *tilt*.

Exercício: Utilizando o sensor de impacto, desenhe e implemente um diagrama de atividades que seja um contador de impactos ao longo do tempo.

Computação Física

6. Sinal PWM – Pulse Width Modulation

Um sinal PWM é um sinal digital com um dado período temporal e com *duty cycle* variável. A figura 6.1 mostra exemplos de um sinal digital PWM que só apresenta ao longo do tempo dois níveis de tensão, 0V correspondente ao nível lógico 0 e V_{cc} correspondente ao nível lógico 1.

O *duty cycle* de um sinal digital com frequência f é definido pela relação entre o tempo em que o sinal está a 1 e o respetivo período de tempo ou seja,

$$duty\ cycle = \frac{T_1}{T} \times 100, \quad T = T_1 + T_0$$

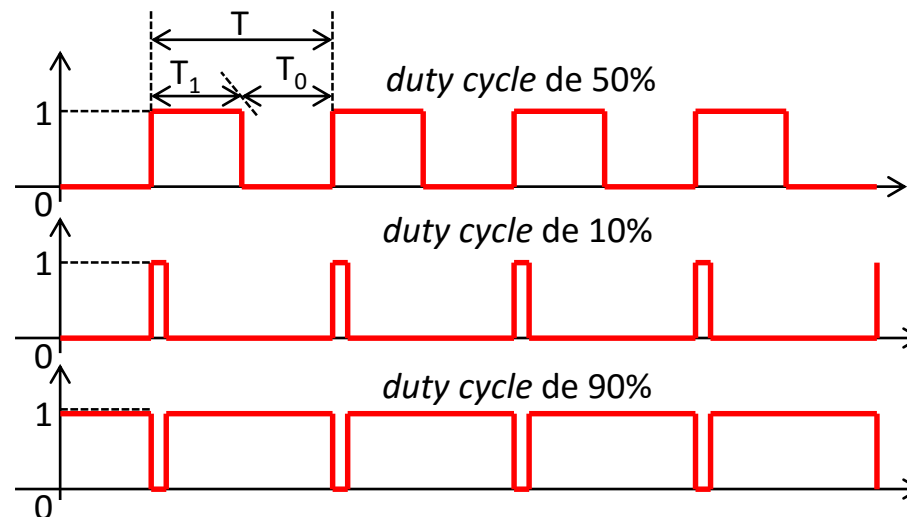
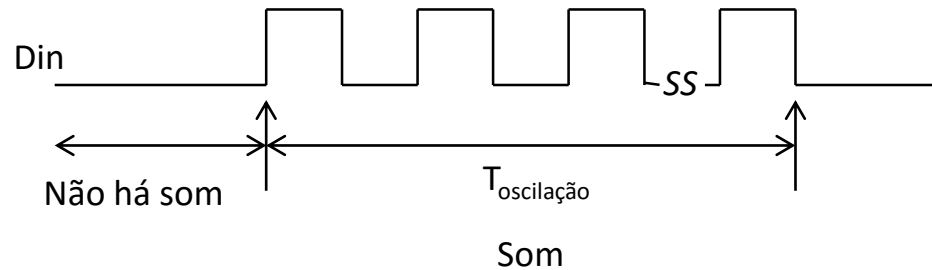


Figura 6.1 – Sinal digital PWM com vários *duty cycles*.

Computação Física

7. Atuador de Som

Um atuador de som é um dispositivo que reproduz um som de acordo com a frequência colocada na sua entrada digital, ver figura 7.1 .



Pino 1 - Din - Sinal PWM digital entre 0 e 5V;

Pino 2 - Não ligado

Pino 3 - GND - 0V.

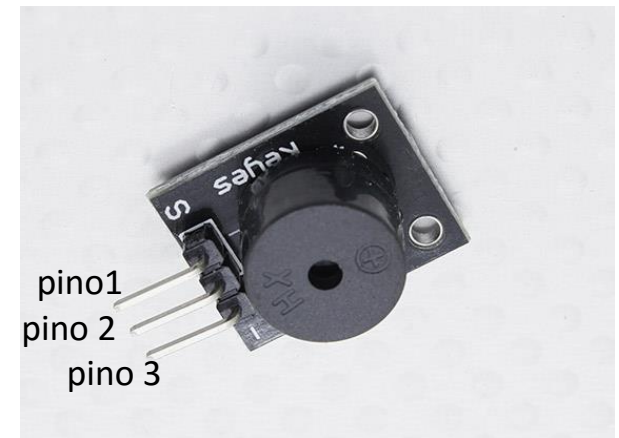


Figura 7.1 - Atuador de som.

Exercício: Utilizando o atuador de som, crie uma melodia.

Computação Física

8. Sensor de distância por ultra-sons

O sensor de distância por ultrasons utiliza a emissão duma onda ultrasónica medindo o tempo que demora a recepção dessa onda e com esta técnica permite medir a distância entre o sensor e um obstáculo existente na direção do sensor.

O sensor de ultrasons a estudar é o HC-SR04 que consiste na placa mostrada na figura 8.1.

O sensor HC-SR04 mede distâncias a um objecto localizado entre 2 cm e 400 cm com uma resolução até 3mm.

O princípio de funcionamento deste sensor consiste em dar um impulso no pino 2 (Trigger) com uma duração de 10us. O sensor após o impulso envia 8 impulsos de 40KHz, ficando à espera do eco do sinal.

Se receber eco, o sensor coloca no pino 3, um sinal PWM com uma duração temporal igual à seguinte fórmula:

$$\text{Distancia [cm]} = T_1[\text{us}]/58$$

A medição de 2 distâncias consecutivas deve de distar no tempo pelo menos 60ms.



Figura 8.1 – Pinos do HC-SR04 Sonar.

Computação Física

9. Protocolo I²C

O protocolo I²C é baseado em dois sinais digitais, o sinal SDA (Serial Data) e o sinal SCL (Serial Clock). A taxa de transferência deste protocolo pode ir até aos 3,4Mbit/seg.

O protocolo tem uma condição de start e uma condição de stop. A condição de start acontece quando há uma transição descendente em SDA e SCL está a 1 e a condição de stop acontece quando há uma transição ascendente em SDA e SCL está a 1, ver figura 9.1.

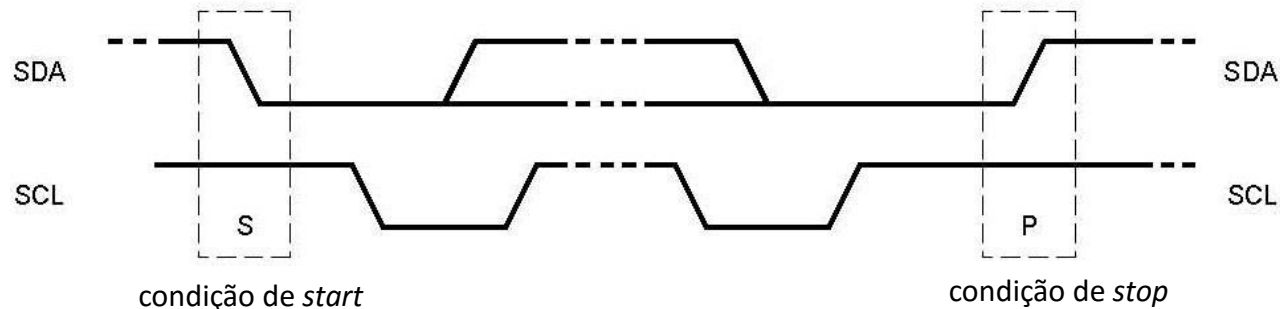


Figura 9.1 – Condições de *start* e *stop* do protocolo I²C.

A validação de um bit de informação em SDA e a mudança de valor lógico estão ilustrados na figura 9.2.

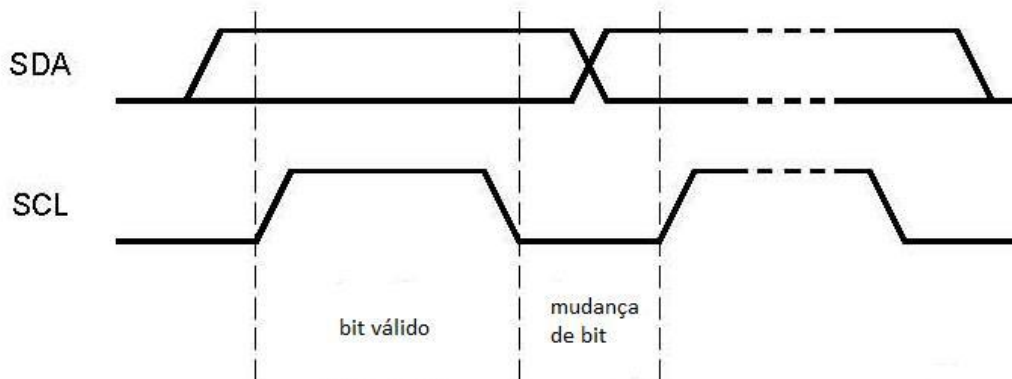


Figura 9.2 – Validação de bit de dados em SDA quando SCL a 1.

Computação Física

A trama completa do protocolo I²C desde a condição de start até à condição de stop é mostrada na figura 9.3.

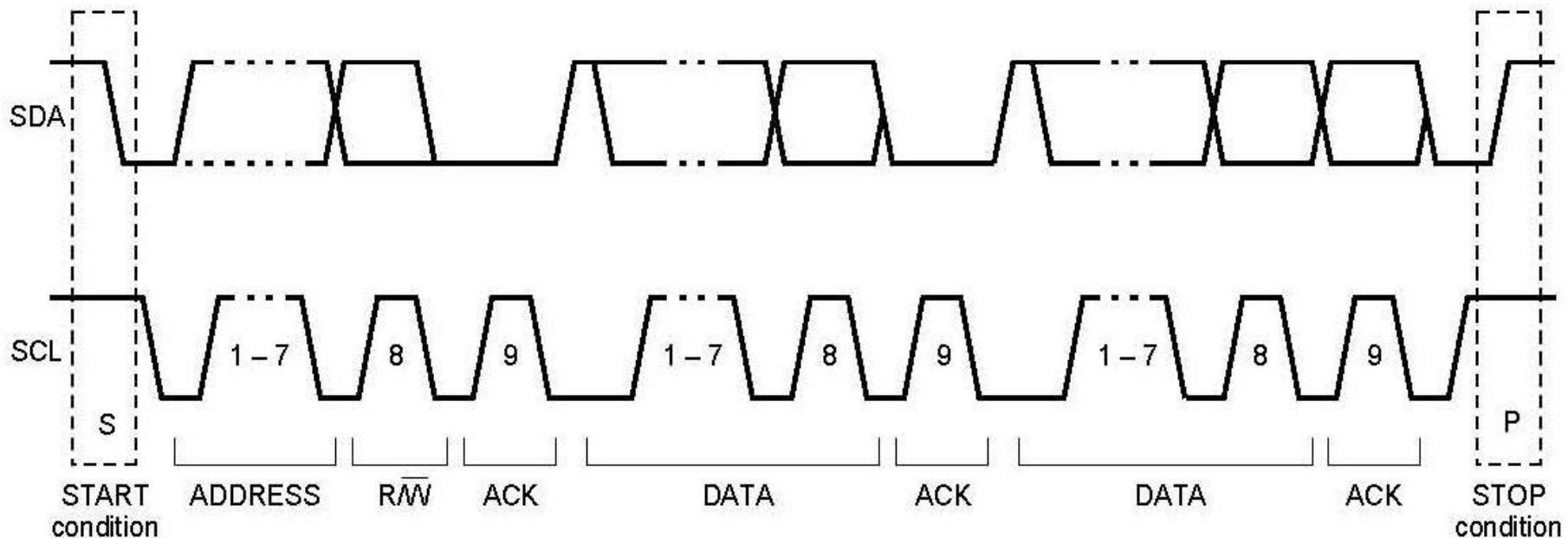


Figura 9.3 - Transferência de informação completa.

A trama começa com a condição de start, depois vêm 7 bits de endereço (address) seguido de um bit que indica se o comando é de leitura ou escrita ($R/W=1$ – leitura; $R/W=0$ – escrita), depois recebe 1 bit de acknowledge. De seguida, recebe ou envia 8 bits de dados seguido de um bit de acknowledge e depois mais 8 bits de dados seguido de um bit de acknowledge, e finalmente a condição de stop. O bit de acknowledge é um bit de validação da informação recebida ou enviada previamente.

Computação Física

9.4. Protocolo I²C no arduino

A arquitetura arduino é baseada no processador AtMega328, e este processador tem dois sinais AIN4(SDA) e AIN5 (SCL) que suportam a implementação dos sinais de protocolo I²C .

A biblioteca do arduino Wire.h manipula os dois sinais, SCL e SDA, implementando um nível de abstração de métodos com as seguintes funcionalidades:

`begin()` – inicializa o arduino como *master* no protocolo I²C.

`begin(endereço)` – inicializa o arduino como *slave* no protocolo I²C, quando especificado um endereço com 7 bits.

`requestFrom(endereço, bytes)` – indica a quantidade de bytes a receber de endereço.

`beginTransaction(endereço)` – produz a condição de *start* na linha, o valor do endereço é 0x77.

`endTransimission()` – produz a condição de *stop* na linha I²C.

`write(byte)` – envia o parâmetro byte.

`byte available()` – devolve a quantidade de *bytes* existente no contentor de recepção.

`byte read()` – recebe o byte mais antigo existente no contentor de recepção.

`onReceive(handler)` – especifica um método em handler que será chamado quando houver recepção de informação como *slave*.

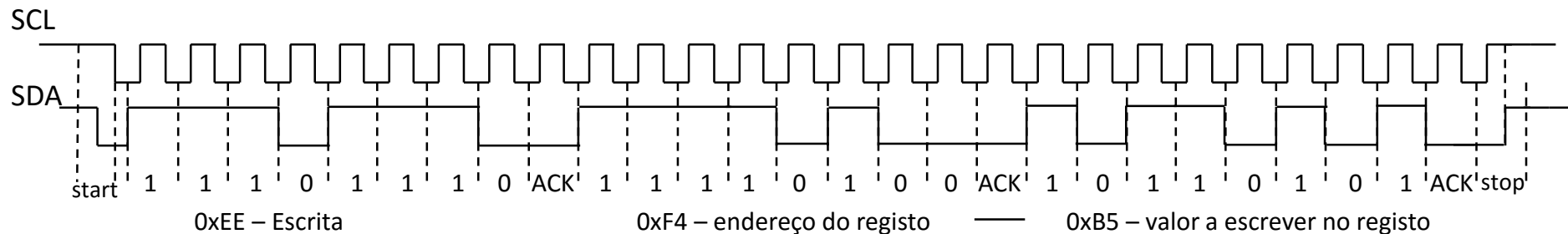
`onRequest(handler)` – especifica um método em handler que será chamado quando um *master* pede informação a um *slave*.

Computação Física

9.5. Geração de uma trama I²C com a biblioteca Wire.h

Dois exemplos da geração de tramas I²C utilizando a biblioteca Wire.h é mostrada a seguir.

Considere que pretende gerar a seguinte trama para escrita de um valor no registo:

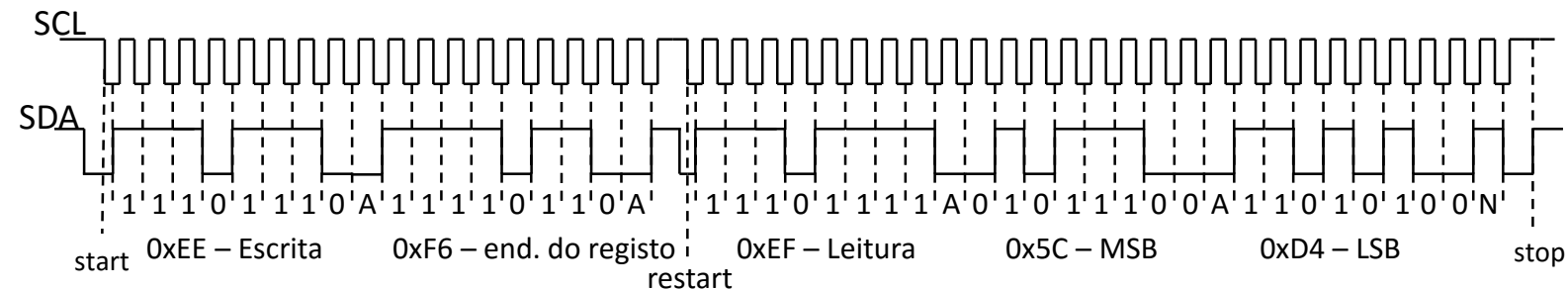


O código correspondente é:

```
Wire.beginTransmission(0x77);  
Wire.write(0xF4);  
Wire.write(0xB5);  
Wire.endTransmission();
```

Computação Física

Considere que pretende gerar a seguinte trama composta por uma escrita num registo seguida de uma leitura de um valor a 16 bits:



o código correspondente é:

```
Wire.beginTransmission(0x77);  
Wire.write(0xF6);  
Wire.endTransmission();
```

```
Wire.requestFrom(0x77, 2);  
while (Wire.available()==0);  
MSB= Wire.read();  
while (Wire.available()==0);  
LSB=Wire.read();  
Wire.endTransmission();
```

Computação Física

10. Display I2C

O display I2C é um de display de 2 linhas com 16 caracteres em cada linha.

O display disponibiliza uma interface I2C para comunicação com um microcontrolador. Por cada byte enviado via I2C, a interface I2C-paralelo define 4 sinais de dados para o display e 4 sinais de controle de acordo com a figura 10.1. O endereço I2C do display é o 27h.

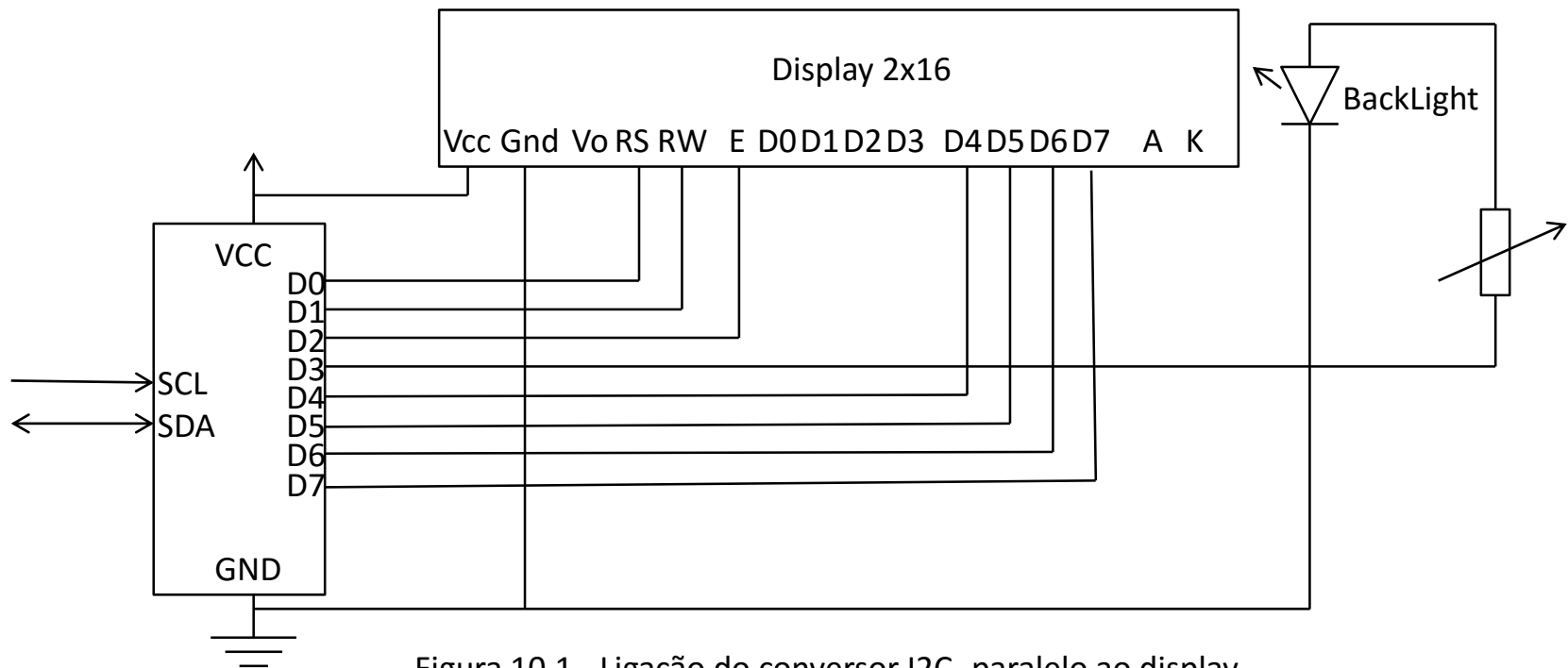


Figura 10.1 - Ligação do conversor I2C- paralelo ao display.

Computação Física

O display tem as seguintes instruções:

Função	Tipo	D7	D6	D5	D4	D3	D2	D1	D0	Tempo
Clear Display	Comando	0	0	0	0	0	0	0	1	5 ms
Cursor Home	Comando	0	0	0	0	0	0	0	0	5 ms
Entry Mode	Comando	0	0	0	0	0	1	I/D	S	120us
Display on/off	Comando	0	0	0	0	1	D	C	B	120 us
Cursor & Display shift	Comando	0	0	0	1	S/C	R/L	x	x	120 us
Function set	Comando	0	0	1	DL	N	F	x	x	120 us
Set Cursor Address	Comando	1	A6	A5	A4	A3	A2	A1	A0	120 us
Write Data to cursor position	Dados	D7	D6	D5	D4	D3	D2	D1	D0	120 us

I/D = 1 - incrementa a posição do cursor ; I/D= 0 - decrementa a posição do cursor.

S= 1 - permite display shift.

D= display, C= cursor, B= blink (1= ON, 0= Off).

S/C= 1 - display shift, S/C= 0 - cursor move.

R/L= 1 - shift right, R/L= 0 - shift left.

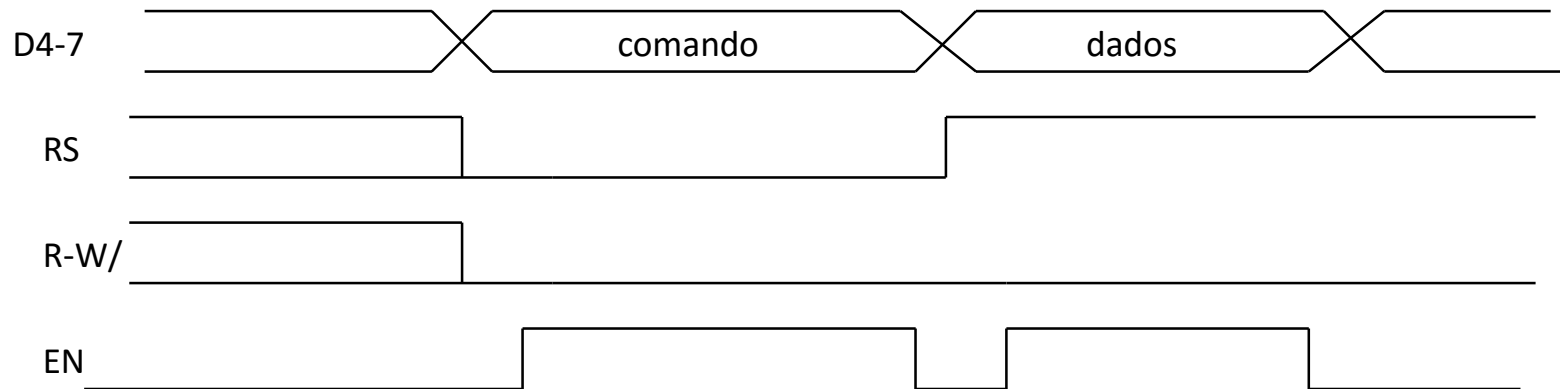
DL= 1 - 8 bits de data, DL= 0 - 4 bits de data.

N= 1 - display 2 linhas, N= 0 - display de 1 linha.

F=1 - (5x10 dots só em 1 linha), F= 0 (5x7 dots em 1 ou 2 linhas).

Computação Física

Para escrever uma 4 bits no display, tem de cumprir o seguinte diagrama temporal para os sinais de controlo:



Exercício 1: Escreva um método para escrever 4 bits de dados no display mantendo a luz do display acesa.

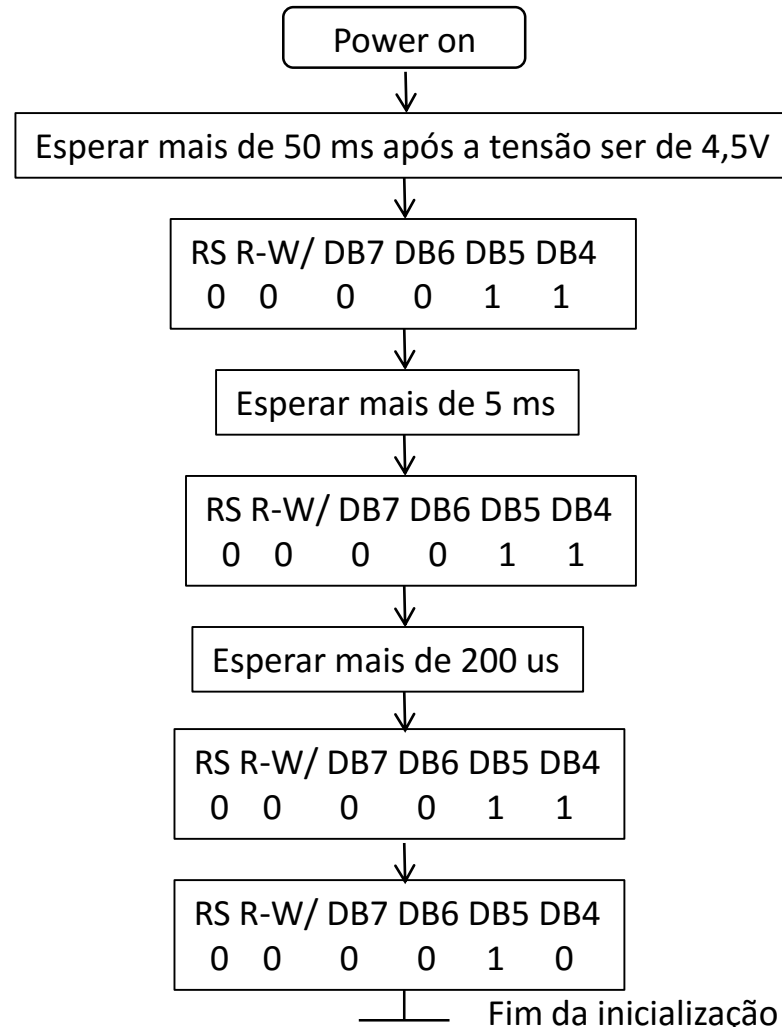
Resolução 1:

Exercício 2: Escreva um método para escrever 4 bits de um comando no display mantendo a luz do display acesa.

```
#define RS 0x01
#define RW 0x02
#define EN 0x04
#define LUZ 0x08
#define ENDERECO 0x27
void escreverDados4(byte quatroBits) {
  Wire.beginTransmission(ENDERECO);
  Wire.write((quatrobits << 4) | LUZ | RS);
  Wire.endTransmission();
  Wire.beginTransmission(ENDERECO);
  Wire.write((quatrobits << 4) | LUZ | RS | EN );
  Wire.endTransmission();
  delayMicroseconds(1); // enable ativo >450ns
  Wire.beginTransmission(ENDERECO);
  Wire.write((quatrobits << 4) | LUZ | RS);
  Wire.endTransmission();
  delayMicroseconds(40); // tempo > 37us para comando fazer efeito}
```

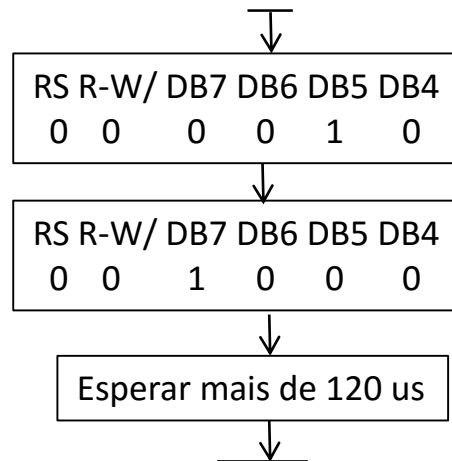
Computação Física

Como a interface I2C faz-se a 4 bits com o display o display tem de ser inicializado para funcionar no modo de 4 bits, que segundo o fabricante consiste no envio da seguinte sequência de instruções:

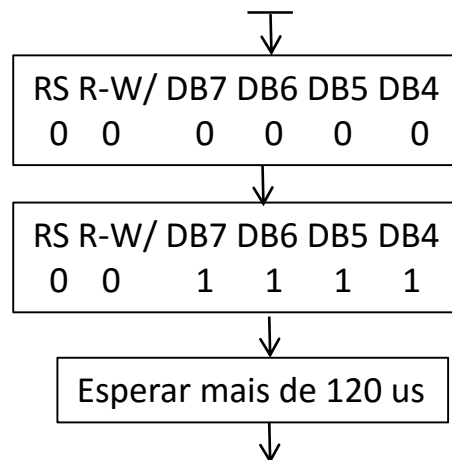


Computação Física

Após cumprir-se a inicialização da interface a 4bits, deve-se definir como o display vai funcionar.
Por exemplo, se quiser usar o display em modo 2 linhas de 16 caracteres, deve enviar o comando Function Set:

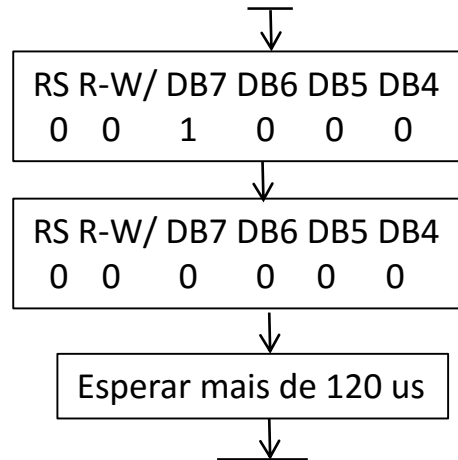


Para colocar o display On, cursor On e blink On deve enviar o comando:



Computação Física

Para colocar o cursor numa determinada posição no display deve enviar a instrução Set Cursor Address, onde na linha 0 cada carater do display vai de 0 a Fh e na linha 1 cada carater vai de 40h a 4Fh.



Computação Física

11. Comunicação Série

A comunicação série consiste no envio e recepção, de forma sincronizada, de 1 bit de informação em cada intervalo temporal designado por tempo de *bit* (*bit time*).

O nível lógico 1 costuma designar-se por “mark” e o nível lógico 0 designa-se por “space”. A nível físico, os dois níveis lógicos correspondem a níveis de tensão diferentes ou à existência ou não de corrente na linha. Por exemplo, apresenta-se de seguida alguns protocolos para comunicação série e os respectivos níveis de tensão:

Protocolo	Mark (1)	Space (0)
TTL	> 2,4 V	< 0,8 V
Loop de Corrente	20 mA	0 mA
RS232	< -3 V	> 3 V

Em termos históricos, o primeiro protocolo adoptado foi o “loop de corrente de 40mA” que se adaptava perfeitamente às características electromecânicas dos dispositivos da época, em que os relés eram directamente excitados pela corrente de 40mA pulsando ao ritmo de mark-space dos sinais enviados.

A conversão de níveis de tensão TTL para RS232 são realizados através de circuitos SSI desenvolvidos para o efeito, caso dos MC1488 e MC1489 da Motorola e os 75188 e 75189 da Texas Instruments.

Usualmente costuma dizer-se que existem dois tipos de comunicação série, a comunicação síncrona e a comunicação assíncrona. Dentro da classe da comunicação síncrona englobam-se os protocolos Bisync, SDLC, HDLC, etc. Na classe da comunicação dita assíncrona temos o protocolo *start-stop* bit. É sobre este protocolo que vamos dirigir o nosso estudo.

A comunicação designa-se assíncrona quando existe unicamente sincronismo ao nível do bit. Enquanto se houver sincronismo ao nível do carácter e da trama(conjunto de caracteres) a comunicação designa-se por síncrona.

Computação Física

11.1. Protocolo *start-stop bit*

A composição deste protocolo é ilustrada na figura 11.1.1 para 5 bits de dados e um bit de paridade (V41P).

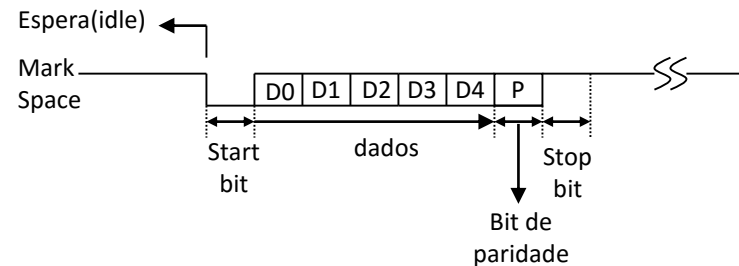


Figura 11.1.1 – Protocolo *start-stop* com 5 bits de dados e 1 bit de paridade.

O bit de paridade pode ou não existir. Caso exista, a paridade designa-se por “par” quando o número de bits de dados mais o bit de paridade têm no conjunto um número par de 1’s. Quando existir um número ímpar de 1’s entre os bits de dados mais o bit de paridade então a paridade designa-se “ímpar”.

Na comunicação série assíncrona o débito da comunicação ou o número de bits por segundo designa-se por *baud rate*. Os valores usuais deste parâmetro são: 110, 300, 600, 1200, 2400, 4800, 9600, 14400, 28800, 57600 e 115200.

Exercício 1: Realize uma rotina para transmissão de dados série através do bit de saída TX do arduino. Esta rotina tem como parâmetro de entrada o carácter a enviar.

Exercício 2: Realize uma rotina para recepção de dados série através do bit de entrada RX do arduino. Esta rotina tem como parâmetro de saída o carácter recebido.

Computação Física

11.2. Comunicação Série no arduino – classe Serial

A comunicação série serve para comunicar entre o arduino e o PC ou entre o arduino e qualquer outro dispositivo externo que suporte comunicação série. O arduino tem dois pinos digitais 0 e 1 que também são designados como RX (pino de recepção) e TX (pino de transmissão) respetivamente.

O ambiente de trabalho tem um monitor que pode servir de interface com o arduino para em fase de teste verificar a aplicação. Para existir comunicação entre o arduino e o monitor tem de se seleccionar o mesmo *baud rate* em ambos. O *baud rate* é a quantidade de bits por segundo que é transferido entre dois equipamentos distintos.

A comunicação série está associada à classe Serial e os métodos são as seguintes:

`begin(long velocidade)` – o parâmetro velocidade pode ter os seguintes valores numéricos: 300, 1200, 4800, 9600, 14400, 19200, 28800, 38400, 57600 ou 115200. Pode especificar outros valores de velocidade. Este método é um método de inicialização portanto deve ser evocado no `setup()` e na forma, `Serial.begin(115200);` .

`end()` – termina a comunicação série nos pinos RX e TX. Para ser reiniciada a comunicação tem de se voltar a chamar o método `begin`.

`int available()` – devolve o número de bytes recebidos e armazenados no contentor de recepção de bytes.

`byte read()` – lê o byte mais antigo recebido, este byte deixa de estar armazenado no contentor.

`byte peek()` - devolve o byte recebido mais antigo do contentor mas este continua armazenado no contentor.

`flush()` – vaza o contentor de recepção de bytes.

`print(tipo)` – envia para o contentor de transmissão o parâmetro tipo passado no método, fazendo a sua conversão para caracteres ASCII. Por exemplo, `Serial.print("Ola");` ou `Serial.print(234);` .

`println(tipo)` – tem um comportamento idêntico ao método `print()` acrescentado um carácter de mudança de linha.

`write(tipo)` – escreve no contentor de transmissão a informação passada no parâmetro sem qualquer conversão.

Computação Física

12. Arduino e Python

A comunicação entre as linguagens arduino e python é suportada pela biblioteca pySerial que é genérica a vários sistemas operativos.

A biblioteca pySerial define a classe serial.Serial que tem a seguinte API:

```
class serial.Serial
```

```
_init_(port=None, baudrate= 9600, bytesize=EIGHTBITS, parity=PARITY_NONE, stopbits=STOPBITS_ONE,  
        timeout= None, xonxoff=False, rtscts=False, writeTimeout=None, dsrdtr=False, interCharTimeout=None)
```

Parâmetros:

port- nome do dispositivo ou número do porto ou None;

baudrate – qualquer valor entre 9600 e 115200;

bytesize- quantidade de bits de dados, os valores possíveis são : FIVEBITS, SIXBITS, SEVENBITS, EIGHTBITS;

parity – bit de paridade com os seguintes possíveis valores: PARITY_NONE, PARITY_EVEN, PARITY_ODD;

stopbits – quantidade de stop bits, os possíveis valores são: STOPBITS_ONE, STOPBITS_ONE_POINT_FIVE, STOPBITS_TWO;

timeout – tempo de espera para leitura;

xonxoff – fluxo controlado por software;

rtscts – fluxo controlado pelo sinal CTS/RTS;

dsrdtr – fluxo controlado pelo sinal DSR/DTR;

writeTimeout – define um tempo de espera para escrita;

interCharTimeout – tempo de espera entre caracteres, o valor None permite a não definição deste tempo.

Retorna:

ValueError – Quando um dos parâmetros está fora da gama de valores permitidos;

SerialException – é gerada uma exceção quando o dispositivo não existe ou não pode ser configurado.

Computação Física

`open()` – abertura do canal de comunicação.

`close()` – fecho do canal de comunicação.

`_del_()` – destrutor da classe, fecha o canal de comunicação.

Os seguintes métodos devolvem `ValueError` quando aplicados a um canal de comunicação fechado:

`read(size=1)`

Parâmetros:

`size` – quantidade de bytes para serem lidos.

Retorno:

Os bytes lidos do canal de comunicação.

`write(dados)`

Parâmetros:

`dados` – bytes para serem enviados.

Retorno:

Quantidade de bytes enviados.

Produz:

`SerialTimeoutException` quando o timeout de escrita é definido e é atingido este timeout.

Computação Física

`inWaiting()` – devolve a quantidade de caracteres no buffer de recepção.

`flushInput()` - esvazia o buffer de entrada.

`flushOutput()` – esvazia o buffer de saída.

`setRTS(level=True)` – define o sinal RTS no nível lógico passado como parâmetro.

`setDTR(level=True)` – define o nível lógico do sinal DTR.

`getCTS()` – devolve o valor lógico do sinal CTS.

`getDSR()` – devolve o valor lógico do sinal DSR.

`getRI()` – devolve o valor do sinal RI.

`getCD()` – devolve o valor do sinal CD.

`linha= readline()` – lê uma linha.

`writeline(linha)` – escreve uma linha.

Computação Física

Exercício:

1. Defina métodos em python que definam os seguintes métodos,
 - a) Inicialização de um canal série.
 - b) Recepção de um byte do arduino.
 - c) Envio de um byte para o arduino.

#Resolução:

```
import serial
```

```
com = 'COM24'  
baudrate = 9600
```

#a)

```
def comInit(com, baudrate):
```

```
try:
```

```
    Serie = serial.Serial(com, baudrate, timeout=0)
```

```
    time.sleep(2) # wait for arduino reset
```

```
    print 'Sucesso na ligação ao arduino.'
```

```
    return Serie
```

```
except:
```

```
    print 'Insucesso na ligação ao arduino.'
```

```
    return None
```

b)

```
def caracterReceive(Serie):
```

```
try:
```

```
    car= Serie.read()
```

```
except:
```

```
    print 'Erro na comunicação.'
```

```
    Serie.close()
```

#c)

```
def caracterSend(Serie, info):
```

```
try:
```

```
    Serie.write(info)
```

```
except:
```

```
    print 'Erro de comunicação.'
```

```
    Serie.close()
```

Computação Física

Bibliografia

Pais J., Computação Física, ISEL, 2012.

Banzi M., "Getting Started with Arduino", O'Reilly, 2009.

Fraden J. , "Handbook of Modern Sensors", Springer-Verlag, 2004.

Haque H., Somlai-Fischer A., "Low Tech Sensors and Actuators", FACT, 2005.

Booch G., Jacobson I., Rumbaugh J., "UML Distilled", Addison-Wesley, 2000.