3. CPLD (*COMPLEX PROGRAMMABLE LOGIC DEVICE*)

O desenvolvimento tecnológico verificado nos últimos anos, permite integrar num único circuito integrado milhões de transístores. Os vários fabricantes desenvolveram vários tipos de estruturas algumas delas programáveis e que denominamos genericamente por CPLD (*Complex Programmable Logic Device*). Os fabricantes de CPLDs desenvolveram o conceito de matriz de portas, onde estabelecem estruturas regulares com possibilidade de estabelecer/programar ligações entre elas.
Entre os CPLDs mais comuns, encontram-se ROMs, PLAs, PALs, FPGAs.

3.1 ROM (*Read Only Memory*)

A ROM é um circuito combinatório que pode ser entendida como a implementação de uma tabela bidimensional de m linhas e n colunas e que pode ser utilizado de duas formas: como um dispositivo de memória que contém m palavras endereçáveis através de ($\log_2$ m) sinais, contendo cada palavra n bits, ou como um sistema gerador de n funções de ($\log_2$ m) variáveis independentes como mostra a Figura 3-1.

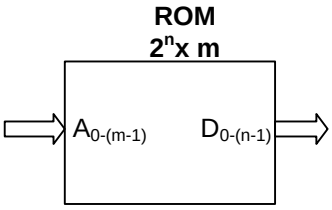

Figura 3-1

A estrutura ROM consiste em duas matrizes: uma AND gerador de todos os termos mínimos de m variáveis como mostra a Figura 3-2 e uma matriz OR, esta programável, e que se constitui pela união de termos produto gerados pela matriz AND como mostra a Figura 3-3.

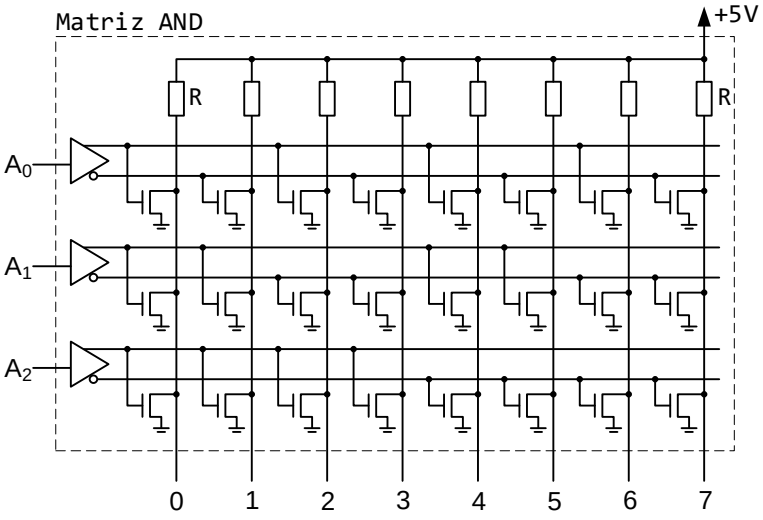

Figura 3-2

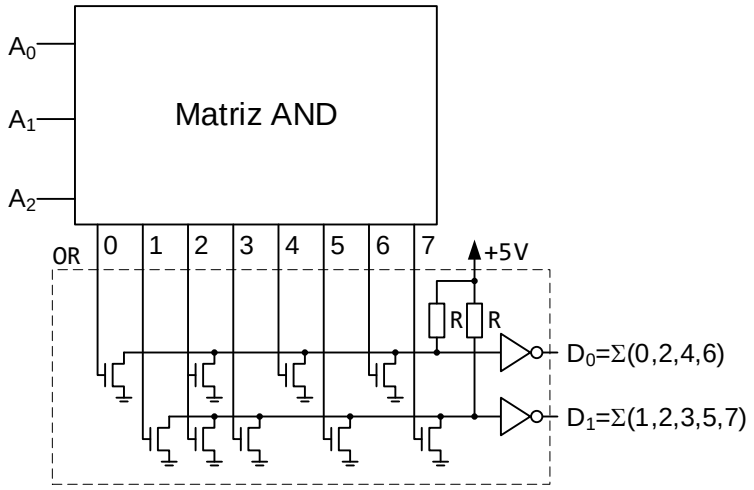

Figura 3-3

Na Figura 3-4 está representada uma estrutura ROM de 8x4 que implementa 4 funções F_0 a F_3 descritas na Tabela 3-1.

F_0 = o número binário presente na entrada é múltiplo de 3.
F_1 = o número binário presente na entrada é divisível por de 3.
F_2 = o número binário presente na entrada é primo.
F_3 = existe um número ímpar de 1s na entrada.

A_2	A_1	A_0	F_3	F_2	F_1	F_0
0	0	0	0	0	0	1
0	0	1	1	1	0	0
0	1	0	1	1	0	0
0	1	1	0	1	1	1
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	1	0	1	1
1	1	1	1	1	0	0

Tabela 3-1

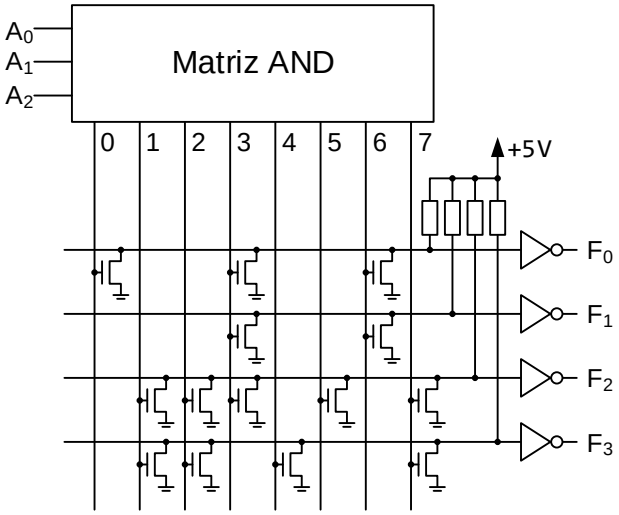

Figura 3-4

Na Figura 3-5 está representada uma estrutura ROM de 8x4 com perspectiva de memória que implementa uma tabela de *look up* ordenada de forma crescente com os números primos entre 0 e 13 descrita na Tabela 3-2.

A_2	A_1	A_0	P_3	P_2	P_1	P_0	
0	0	0	0	0	0	1	1
0	0	1	0	0	1	0	2
0	1	0	0	0	1	1	3
0	1	1	0	1	0	1	5
1	0	0	0	1	1	1	7
1	0	1	1	0	1	1	11
1	1	0	1	1	0	1	13
1	1	1	-	-	-	-	-

Tabela 3-2

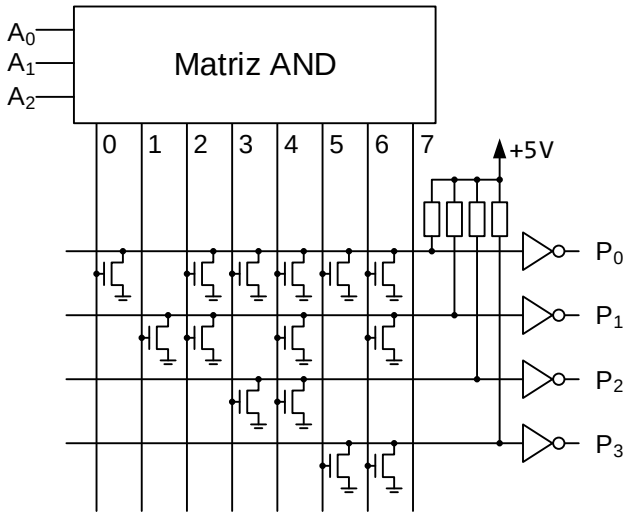

Figura 3-5

3.2 PAL (*Programmable Array Logic*)

Como se pode constatar, na estrutura ROM são gerados todos o mini-termos das variáveis de entrada (endereços), o que representa uma grande ineficiência quando se pretende implementar funções utilizando uma ROM. Isto acontece, porque na generalidade das aplicações, as funções geradas utilizam uma percentagem muito pequena dos mini-termos possíveis das variáveis de entrada. Neste sentido o fabricante AMD, desenvolveu e patenteou uma estrutura programável que denominou por PAL® e que consiste num conjunto de macro células interligáveis através de uma matriz programável. A cada macro célula está associada uma saída do dispositivo e consiste no seguinte: a saída é a união de um número variável de termos produto (8 a 16), e cada um desses mini-termos tem como entrada uma matriz programável onde estão disponíveis todas as variáveis de entrada, as variáveis de saída e o seu complemento. Existe ainda a possibilidade de programar se a saída da união é ou não invertida, no sentido de possibilitar a simplificação por DeMorgan. Na Figura 3-6 está representada a estrutura típica de uma PAL®. Programar consiste em ligar as entradas dos termos produto de cada macro célula às variáveis disponíveis nas colunas.

Na Figura 3-6 podemos ver quais as ligações que seriam necessárias realizar na matriz programável, se pretendermos com ela implementar o segmento **a** e **f** do mostrador de 7 segmentos do exercício 1 do capítulo 2. Consideremos as seguintes atribuições:

I_0=A, I_1=B, I_2=C, I_3=D

X=O_0, a=O_1 f=O_2

$\boldsymbol{X} = D + AC$, $\boldsymbol{a} = X + \overline{A}.\overline{B} + B.\overline{C}$ e $\boldsymbol{f} = X + B.C + \overline{A}.\overline{B}.\overline{C}$.

Uma ligação na matriz é assinalada com um x e corresponde à ligação de uma coluna com uma das entradas da porta AND. As entradas do AND não utilizadas são colocadas ao valor lógico 1 (elemento neutro do produto). No caso da função **X** e da função **a**, o produto que não foi utlizado fica com todas as entradas ao valor lógico zero (elemento absorvente do produto) para implicar elemento neutro dessa mesma união.

Dado que a função **X** tem dois termos produto, caso não fosse utilizada a estrutura multi-nível (*bridging*), tanto o segmento **a** como o segmento **f** não seriam implementáveis nesta PAL pois necessitariam de cinco termos produto, dai também a importância das saídas estarem disponíveis na matriz de ligações.

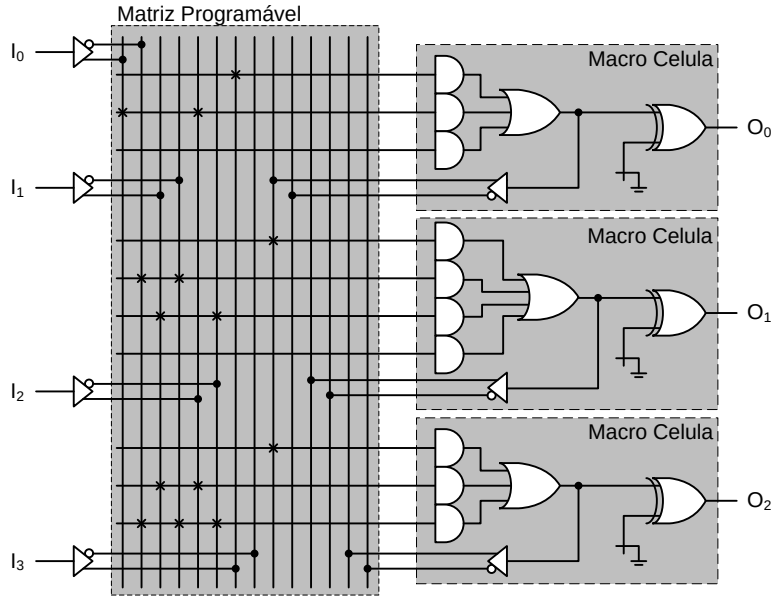

Figura 3-6

Para programar este tipo de dispositivos foram criadas linguagens de descrição de *hardware* que permitem de uma forma simples e estruturada a descrição de um sistema digital. Entre estas linguagens encontra-se o PALASM, o Abel e CUPL, tendo o PALASM sido descontinuado pela AMD. O fabricante ATMEL pôs disponível de forma gratuita uma versão do CUPL para Windows que denominou por WinCUPL.

3.3 CUPL

Este documento não constitui um manual completo de instruções da linguagem CUPL, contém apenas as directivas mais relevantes e que permitem a descrição *hardware* de sistemas digitais de pequena complexidade. Para informação mais detalhada é necessário consultar o manual de utilização "*ATMEL WinCUPL User's Manual*".

O ficheiro contendo a descrição *hardware*, deve ter a extensão PLD e pode ser editado no WinCUPL ou num qualquer editor de texto. Este ficheiro contém as várias instruções que constituem a descrição do sistema digital e que consistem essencialmente em declaração de variáveis e expressões lógicas para cada uma das saídas.

Cada instrução tem que estar contida numa única linha de texto e terminada pelo carácter '**;**'.

3.3.1 Definição de variável

As variáveis são palavras utilizadas para designar os vários sinais ou conjunto de sinais.
O nome de uma variável tem as seguintes regras:

- Podem ser iniciados com uma letra, um dígito ou um sublinhado tendo que conter obrigatoriamente uma letra.
- Podem conter no máximo 31 caracteres, sendo os restantes ignorados.
- Não podem conter espaços, letras acentuadas, palavras reservadas (ver Tabela 3-4) ou símbolos reservados (ver Tabela 3-3). As palavras reservadas não são "*case sensitive*".
- Os nomes atribuídos às variáveis são "*case sensitive*", ou seja, existe distinção entre uma mesma palavra escrita com letras maiúsculas ou minúsculas.

&	#	(	)	-
*	+	[	]	/
:	.	..	/*	*/
;	,	!	'	=
@	$	^		

Tabela 3-3

APPEND	ASSEMBLY	ASSY	COMPANY	CONDITION
DATE	DEFAULT	DESIGNER	DEVICE	ELSE
FIELD	FLD	FORMAT	FUNCTION	FUSE
GROUP	IF	JUMP	LOC	LOCATION
MACRO	MIN	NAME	NODE	OUT
PARTNO	PIN	PINNNODE	PRESENT	REV
REVISION	SEQUENCE	SEQUENCED	SEQUENCEJK	SEQUENCERS
SEQUENCET	TABLE			

Tabela 3-4

Exemplo:

```
Enable
_variavel1
74193_Cy_out
1234  (invalido)
```

3.3.2 Variáveis Indexadas

Quando temos um conjunto de linhas que constituem um grupo ordenado (bus) podemos definir cada uma das linhas como uma variável indexada, ou seja, colocando um número decimal entre 0 e 31 no fim do nome da variável. Por exemplo podemos declarar um grupo de 16 linhas da seguinte forma `[address0..15]` ou de forma discreta `[A0,A1,A5]`.

3.3.3 Valor numérico

Um valor numérico pode ser especificado numa das seguintes bases: binária, octal, decimal ou hexadecimal (ver Tabela 3-5). Para se indicar que um número está numa qualquer base particular precede-se o número do respectivo prefixo (o prefixo não é *case sensitive*).
O número de um pino ou do índice de uma variável é dado em decimal, os restantes, quando não especificado são em base hexadecimal.

Número	Base	Valor Decimal
'b'0	binário	0
'b'1101	binário	13
'o'663	octal	435
'D' 92	decimal	92
'h' BA	hexadecimal	186
'b'[001..100]	binário	Compreendido entre 1 e 4

Tabela 3-5

3.3.4 Ficheiro PLD

O ficheiro PLD é constituído por três segmentos:

- Cabeçalho
- Declaração
- Corpo Principal

3.3.4.1 Cabeçalho

Colocado no início do ficheiro, contém o nome* do projecto, informação genérica e a referência* do dispositivo (`Device`) que vai ser utilizado na implementação.

```
Name     SOMADOR_2_bits ;
PartNo   00 ;
Date     16-03-2008 ;
Revision 01 ;
Designer Jose Paraiso;
Company  ISEL ;
Assembly None ;
Location  ;
Device   p22v10 ;
```

* Preenchimento obrigatório

3.3.4.2 Declaração

Este segmento deve conter a declaração de todas as variáveis utilizadas no programa (Pin, PinNode, Bit Field e MIN).

Definição e atribuição de pinos

Cada pino de entrada ou saída na PAL é considerado como uma variável do nosso sistema, pelo que no segmento de declaração, atribuiremos um nome a cada PIN utilizado. Na atribuição de pinos há que ter o cuidado de manter compatibilidade com o dispositivo que estivermos a utilizar, isto é, não atribuir por exemplo pinos de alimentação ou outros de uso específico. Alguns dispositivos contêm células cuja saída não está disponível num pino do dispositivo, mas que podem ser utilizadas como entrada de outras macro células. Estes elementos podem ser referidos como NODE ou PINNODE. A designação PINNODE é utilizada quando o projectista pretende estabelecer qual dos nós é utilizado.

ex:

```
PIN   2 = sensor_A;    /* comentario */
PIN   3 = !enable;
PIN   [4..7] = [data0..3];
PIN   [14,15] = [LED_Alarme,solenoide];
PINNODE 25 = variavel_x;
```

Variáveis intermédias

Para tornar a leitura mais simples, estruturar o programa e facilitar o *debugging*, podemos utilizar variáveis intermédias atribuindo-lhe uma expressão lógica. Estas variáveis só são utilizadas em tempo de compilação, não alterando por isso a complexidade da expressão lógica final de uma qualquer variável de saída que utilizou esta variável temporária. Estas variáveis não necessitam de ser declaradas.

Bit Field

Permite referir um conjunto de bits por um único nome de variável.

ex:

```
FIELD operando_A = [A0..3];
```

MIN

Permite especificar qual o número de min-termos que se pretende para uma determinada variável. Esta directiva sobrepõe-se ao nível de optimização que estivermos a utilizar no compilador.

ex:

```
MIN var_saida = 4;
```

3.3.4.3 Corpo Principal

Segmento constituído pelas instruções que determinam a lógica do circuito a implementar.

Equações Lógicas

As equações lógicas utilizam os símbolos apresentados na tabela.

Operador	Descrição	Exemplo	Precedência
!	NOT	!A	1
&	AND	A&B	2
#	OR	A#B	3
$	XOR	A$B	4

Tabela 3-6

3.4 Exemplos:

[1] Implementação do descodificador de 7 segmentos utilizando a descrição em tabela de verdade.

```
Name     BCD_7seg;
PartNo   00 ;
Date     08-11-2012 ;
Revision 01 ;
Designer Jose Paraizo;
Company  ISEL ;
Assembly None ;
Location  ;
Device   v750c ;/* PAL ATF750C */

/* *************** INPUT PINS *********************/
PIN [1..4]  = [I0..3]; /* 4 Entradas */

/* *************** OUTPUT PINS *********************/
PIN   [14..20]  = [a,b,c,d,e,f,g] ; /* 7 Segmentos*/

FIELD number = [I0..3];
FIELD segments = ![a,b,c,d,e,f,g]; /* Leds em anodo comum */

/* *************************** BODY *****************************/

/* definem-se as 7 funcoes utilizando uma tabela de verdade */
TABLE number => segments {
0=>'b'1111110;  4=>'b'0110011;  8=>'b'1111111;  C=>'b'xxxxxxx;
1=>'b'0110000;  5=>'b'1011011;  9=>'b'1111011;  D=>'b'xxxxxxx;
2=>'b'1101101;  6=>'b'1011111;  A=>'b'xxxxxxx;  E=>'b'xxxxxxx;
3=>'b'1111001;  7=>'b'1110000;  B=>'b'xxxxxxx;  F=>'b'xxxxxxx;
}
```

[2] Implementação de um circuito com 6 entradas e 6 saídas, em que cada entrada está associada a uma única saída. Quando se activa uma entrada, a respectiva saída é activada se só existir esta entrada activada. Caso exista uma única entrada activada, a activação de mais uma entrada leva a que todas as saídas fiquem desactivas. A implementação recorre a uma estrutura modular como mostra a Figura 3-7. Para a implementação em PAL, são apresentadas quatro descrições CUPL: 1ª explicitando exaustivamente as expressões de cada uma das fatias (*slice*), 2ª utilizando a indexação, 3ª utilizando a directiva REPEAT e por último utilizando a directiva REPEAT associada à directiva FUNCTION. Com a directiva FUNCTION foi implementada uma função geradora das saídas de um *slice*.

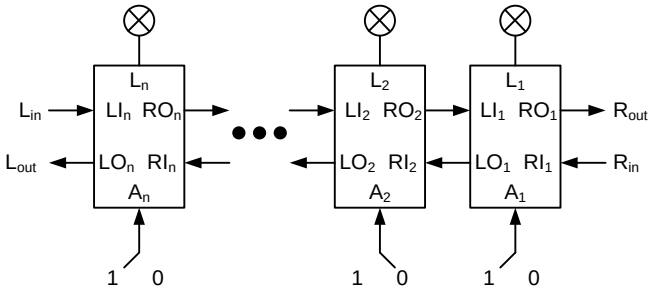

Figura 3-7

1ª Descrição CUPL de forma exaustiva.

```
Name     Fila_LEDs_exp;
PartNo   00 ;
Date     08-11-2012 ;
Revision 01 ;
Designer Jose Paraizo;
Company  ISEL ;
Assembly None ;
Location  ;
Device   v750c ;/* PAL ATF750C */

/* *************** INPUT PINS *********************/
PIN [1..6]  = [A1..6]; /* 6 Entradas */
PIN 7 = Rin; /* informacao vinda da direita */
PIN 8 = Lin; /* informacao vinda da esquerda */

/* *************** OUTPUT PINS *********************/
PIN [16..21]  = [L1..6]; /* 6 Saidas*/
PIN 14 = Rout; /* informacao para a direita */
PIN 15 = Lout; /* informacao para a esquerda */

/* ****************** BODY ************************/
RO7=Rin;
LO0=Lin;
Rout=RO1;
Lout=LO6;

L1 =A1 & !LI1 & !RI1;
RO1=A1 # LI1;
LO1=A1 # RI1;

L2 =A2 & !LI2 & !RI2;
RO2=A2 # LI2;
LO2=A2 # RI2;
```

```
L3 =A3 & !LI3 & !RI3;
RO3=A3 # LI3;
LO3=A3 # RI3;

L4 =A4 & !LI4 & !RI4;
RO4=A4 # LI4;
LO4=A4 # RI4;

L5 =A5 & !LI5 & !RI5;
RO5=A5 # LI5;
LO5=A5 # RI5;

L6 =A6 & !LI6 & !RI6;
RO6=A6 # LI6;
LO6=A6 # RI6;

/* interligacao entre slices */
RI1=LO0;
RI2=LO1;
RI3=LO2;
RI4=LO3;
RI5=LO4;
RI6=LO5;

LI1=RO2;
LI2=RO3;
LI3=RO4;
LI4=RO5;
LI5=RO6;
LI6=RO7;
```

2ª Descrição CUPL utilizando organização em vector

```
/* ****************** BODY ************************/
RO7=Rin;
LO0=Lin;
Rout=RO1;
Lout=LO6;

[L1..6]=[A1..6] & ![LI1..6] & ![RI1..6];
[RO1..6]=[A1..6] # [LI1..6];
[LO1..6]=[A1..6] # [RI1..6];

/* interligacao entre slices */
[RI1..6]=[LO0..5];
[LI1..6]=[RO2..7];
```

3ª Descrição CUPL utilizando a construção REPEAT

```
/* ****************** BODY ************************/
RO7=Rin;
LO0=Lin;
Rout=RO1;
Lout=LO6;

$repeat i = [0,1,2,3,4,5]
/* implementacao de um slice */
  L{i+1}=A{i+1} & !LI{i+1} & !RI{i+1};
  RO{i+1}=A{i+1} # LI{i+1};
  LO{i+1}=A{i+1} # RI{i+1};
/* interligacao entre dois slices */
  RI{i+1}=LO{i};
  LI{i+1}=RO{i+2};
$repend
```

4ª Descrição CUPL utilizando a construção REPEAT e FUNCTION

```
/* ****************** BODY ************************/
RO7=Rin;
LO0=Lin;
Rout=RO1;
Lout=LO6;

function LampSlice(A, LA, RA, RO, LO) {
        RO = A # LA;
        LO = A # RA;
        LampSlice = A & !LA & !RA;
}

$repeat i = [0,1,2,3,4,5]
/* implementacao de um slice */
  L{i+1}=LampSlice(A{i+1},LA{i+1},RA{i+1},RO{i+1},LO{i+1});
/* interligacao entre dois slices */
  RA{i+1}=LO{i};
  LA{i+1}=RO{i+2};
$repend
```

A utilização de variáveis temporárias tem como objectivo melhorar a estruturação e a leitura. Por outro lado facilita o *debugging*, pois caso seja necessário observar o seu valor, basta atribuir-lhe temporariamente um pino de saída que esteja disponível.

A compilação do ficheiro PLD pelo WinCUPL produzirá ficheiros com as seguintes extensões:

DOC contêm informação das expressões finais geradas pelo compilador para cada uma das saídas, quantos min-termos são utilizados em cada saída, qual a matriz de programação produzida.

JED contêm informação de quais as ligações a serem estabelecidas na matriz de programação. O formato desta informação é reconhecido pelos dispositivos de gravação de CPLDs.
Nota importante! O nome deste ficheiro resulta do nome dado no cabeçalho do ficheiro PLD e não do nome do ficheiro de extensão PLD. Para que o nome do ficheiro de extensão JED tenho o nome do ficheiro PLD, é necessário configurar no menu Option->Compiler->General.