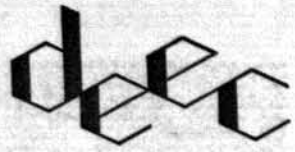


DEPARTAMENTO DE

ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

LÓGICA PROGRAMÁVEL

APLICAÇÃO EM CIRCUITOS DE TELECOMUNICAÇÕES

FACULDADE DE ENGENHARIA

UNIVERSIDADE DO PORTO

Rua dos Bragas, 4099 Porto Codex – PORTUGAL

PEDRO MIGUEL DE VISEU BOTELHO CARDOSO

LÓGICA PROGRAMÁVEL

APLICAÇÃO EM CIRCUITOS DE TELECOMUNICAÇÕES

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Departamento de Engenharia Electrotécnica e de Computadores

JANEIRO 1992

Tese realizada sob a orientação de

Artur Pimenta Alves

Professor Associado do

Departamento de Engenharia Electrotécnica e de Computadores

Faculdade de Engenharia da Universidade do Porto

À Isabel

A meus Pais e Irmãos

Ao meu mestre

Agradecimentos:

Ao Prof. Artur Pimenta Alves, que apesar de assoberbado pelas suas actividades de chefia e coordenação, dispensou algum do seu precioso tempo na discussão e apreciação, sempre pertinentes, do meu trabalho.

Ao Eng. Aníbal Ferreira, responsável pelos módulos de conversão analógico<->digital do SITAD.

Ao departamento de CAD do INESC Porto, na pessoa do José Carlos Azevedo, que com os seus conhecimentos muito contribuiu para a realização dos protótipos SITAD.

A todos os participantes do projecto SIFO, pelo empenho e camaradagem que demonstraram.

Ao Eng. José Manuel Martins Ferreira, pela sua apreciação ao capítulo "Teste - A redefinição de um conceito".

À JNICT, por me ter concedido a bolsa de estudos que possibilitou a execução deste trabalho.

Ao INESC Porto, pelas condições de trabalho que me proporcionou.

INDICE

INDICE DE FIGURAS

1. INTRODUÇÃO

O desenvolvimento de sistemas digitais sofreu, na década de 80, fruto do processo evolutivo tecnológico, profundas alterações na sua filosofia e metodologia de fabrico.
A lógica discreta, baseada na utilização de circuitos integrados de funcionalidade limitada, também designados SSI/MSI (Small/Medium Scale Integration), foi a pouco e pouco substituida pela lógica integrada, com capacidades de integração extremamente elevadas, vulgarmente designada VLSI (Very Large Scale Integration).

O protótipo laboratorial e os instrumentos tradicionais associados de teste, como o osciloscópio e o multímetro, cederam o seu lugar a sofisticados meios computacionais envolvendo capacidades exaustivas de simulação.

O desenho manual da placa de circuito impresso declinou perante o aparecimento de novas ferramentas de desenvolvimento associadas a novas denominações: CAD ; CAE ; EDA.
Englobada nesta míriade de transformações, uma nova tecnologia se destacou - a lógica programável. A explicação do seu estrondoso sucesso assenta em dois factores fundamentais: a pressão crescente do mercado na busca de novos produtos e a consequente necessidade que os fabricantes sentem de recorrer a soluções de baixos custos NRE (Nonrecurring Engineering).

O trabalho apresentado encerra o objetivo claro de demonstrar que este tipo de tecnologia é de importância relevante no espaço industrial e de I&D nacionais, em que os volumes de produção de sistemas nem sempre justificam a aposta em tecnologias do tipo *Gate Array* ou *Standard Cell*.

No capítulo 2 traça-se a evolução histórica dos dispositivos lógicos programáveis, abordam-se problemas tecnológicos prementes como o teste e metastabilidade e referem-se algumas particularidades inerentes à utilização de lógica programável.

O capítulo 3 complementa o estudo efectuado realçando o papel preponderante das ferramentas de desenvolvimento destes dispositivos no sucesso de um projecto. Apresentam-se os produtos existentes, analisam-se as suas virtudes e defeitos. Discutem-se alguns módulos fundamentais destas ferramentas: uma nova linguagem de descrição - VHDL - e o seu suporte - Síntese Lógica. Perspectiva-se o futuro da engenharia de sistemas com a apresentação de um conceito reinventado de pensar e agir em engenharia: Engenharia concorrente.

O capítulo 4 apresenta trabalho desenvolvido na implementação de sistemas electrónicos usando algumas das tecnologias referidas. O sistema escolhido, SITAD, é um sistema de transmissão de audio digital de alta qualidade, do qual foram implementadas três versões distintas, reflexo cronológico da evolução e disponibilidade, de dispositivos lógicos programáveis com capacidades crescentes de integração. É efectuada uma análise comparativa, bloco a bloco, do SITAD, realçando pormenorizadamente algumas descrições específicas do dispositivo utilizado.

No capítulo 5 apresentam-se as conclusões finais, numa análise crítica da evolução tecnológica verificada e discutem-se as técnicas de projecto e teste, actualmente em desenvolvimento, que permite reduzir o consumo e espaço requeridos, aumentar o desempenho, e os níveis de funcionalidade e fiabilidade dos equipamentos.

2. DISPOSITIVOS LÓGICOS PROGRAMÁVEIS

2.1. ARQUITECTURAS FUNDAMENTAIS: PROM, PAL E PLA

Apresentam-se neste capítulo as três arquitecturas fundamentais de lógica combinatória dos circuitos integrados programáveis (PICs) (Sandige, 90): a PROM (Programmable Read Only Memory), a PAL (Programmable Array Logic) e a PLA (Programmable Logic Array). Todas as três resultam de combinações de elementos AND (o plano AND) e elementos OR (o plano OR). Com esta associação é possível a implementação de qualquer equação Booleana, sendo necessária apenas uma transformação algébrica de adaptação à configuração seleccionada.

NOTA: Um PIC (DATA I/O, 90), por definição, é um circuito cuja arquitectura base é estabelecida no momento da sua fabricação, mas cuja configuração, ou seja, a conexão interna dos elementos lógicos que a constituem, é efectuada à posteriori pelo utilizador.

A diferença fundamental entre as arquitecturas reside no grau de liberdade que é concedido ao utilizador para configurar os planos AND e OR.

Na simbologia que utilizamos, um "•" representa uma configuração definida no processo de fabrico do circuito, e um "x" representa uma conexão não definida.

2.1.1. PROM

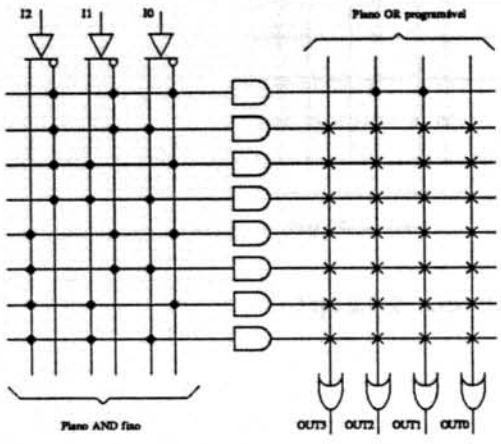
A PROM é a arquitectura menos versátil na implementação de equações Booleanas. A sua estrutura base apresenta o plano AND fixo e o plano OR configurável. As suas aplicações típicas, quase sempre em forma tabular, incluem conversores de códigos (ex. Lei A), armazenamento de tabelas e de programas compilados.

A figura 2.1-1 ilustra a arquitectura interna de uma PROM de 3 entradas e 4 saídas, ou expresso de uma outra forma, de 8 palavras de 4 bits.

Para um número "n" de entradas, obtêm-se 2^n minitermos à saída do plano AND.

Como estas saídas atacam directamente o plano OR, a PROM pode ser usada para a implementação de equações Booleanas utilizando as tabelas de verdade que as representam.

O exemplo da figura 2.1-1 mostra a configuração do plano OR, que permite obter nas saídas o valor desejado (O3 O2 O1 O0 = 0 1 1 0), resultante do valor (I2 I1 I0 = 0 0 0) das entradas.



I2	I1	I0	O3	O2	O1	O0
0	0	0	0	1	1	0
0	0	1	-	-	-	-
0	1	0	-	-	-	-
0	1	1	-	-	-	-
1	0	0	-	-	-	-
1	0	1	-	-	-	-
1	1	0	-	-	-	-
1	1	1	-	-	-	-

Fig. 2.1-1 Arquitectura de uma PROM 8x4.

2.1.2. PAL

A arquitectura PAL (MMI, 87) foi desenvolvida e registada em 1976 por John Birkner. Juntamente com a PLA são utilizadas na implementação de funções sob a forma de equações Booleanas, por oposição à implementação sob a forma tabular da PROM. A grande vantagem destas arquitecturas relativamente à PROM advém da programabilidade do plano AND. Numa PROM, este plano é determinado pelo número de entradas do sistema, havendo uma duplicação do seu tamanho por cada entrada adicional. Há portanto um desperdício grande de lógica interna quando esta é usada na implementação de equações Booleanas de múltiplas entradas.

A arquitectura base de uma PAL apresenta, como já foi referido, o plano AND programável, e o plano OR fixo. A figura 2.1-2 ilustra uma PAL de 3 entradas e 4 saídas. Nesta, o número de entradas no plano AND é igual a 2n em que "n" representa o número de entradas (2n -> n entradas e respectivos complementos). O número de minitermos associados a cada saída, neste caso igual a dois, é determinado no processo de fabrico sendo normalmente igual a 8. A nomenclatura utilizada para designar PALs indica a sua arquitectura fundamental. Assim, por exemplo, a PAL apresentada na figura 2.1-2 seria denominada 3L4, tendo um máximo de 3 entradas, e 4 saídas combinatórias dedicadas.

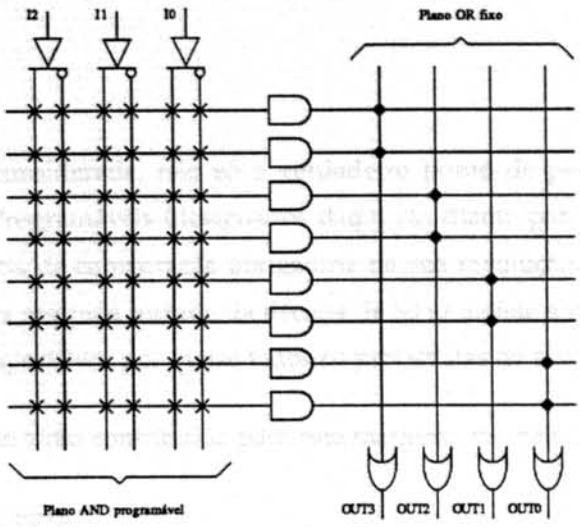


Fig. 2.1-2 Arquitectura de uma PAL 3L4.

2.1.3. PLA

A PLA, terceira arquitectura fundamental, foi introduzida em 1975, sendo de todas a mais flexível. Ambos os planos internos são programáveis, o que comparado com a PAL significa que o número de minitermos disponível para cada saída da PLA, como podemos ver na figura 2.1-3, é configurável pelo utilizador. Esta flexibilidade adicional tem no entanto o seu preço. Dois planos configuráveis significam tempos de propagação internos dos sinais e custos de fabrico mais elevados.

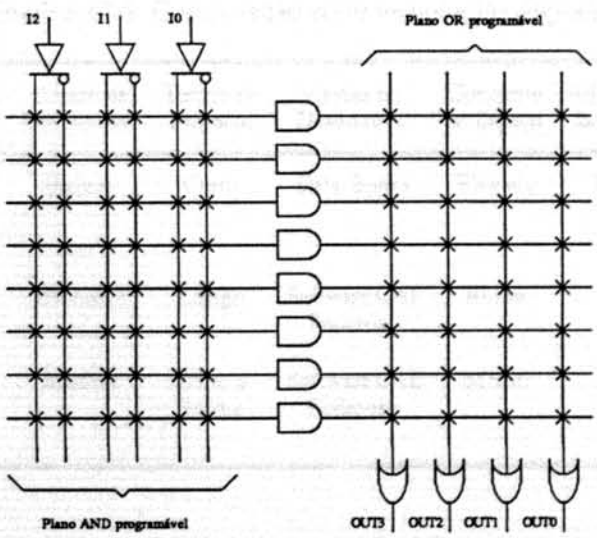


Fig. 2.1-3 Arquitectura de uma PLA.

2.2. EVOLUÇÃO

A PAL deve ser considerada, não só o verdadeiro ponto de partida, na história dos Dispositivos Lógicos Programáveis (designados daqui em diante por PLDs), como, 15 anos volvidos, uma referência de comparação obrigatória na sua revolução.

Somente a partir da segunda metade da década de 80 se assiste a um *boom* generalizado na utilização destes dispositivos por quase todos os projectistas de sistemas.

Quais as razões que terão contribuído para este marasmo de mais de uma década?

- Arquitectura nova
- Desempenho limitado
- Ferramentas de desenvolvimento prácticamente inexistentes

No final dos anos 70, coincidindo com os anos de ouro da família TTL (TI, 84) lançada em meados da década de 60 pela Texas Instruments, a utilização de PALs restringia-se à substituição de pequenos circuitos lógicos contendo elementos combinacionais primários. O domínio desta nova arquitectura, sem a ajuda de ferramentas eficazes na simplificação de lógica Booleana, resultava em implementações de performances reduzidas.

O desenvolvimento tecnológico dos anos 80 abriu finalmente os horizontes aos utilizadores de lógica programável, com o aparecimento no mercado de PLDs com grandes capacidades de integração capazes de implementar funções até aqui limitadas a circuitos integrados de aplicação específica.

Quadro 2.2-1 Comparação entre tecnologias digitais

	Preço por Gate	Custos de Desenvolv.	Tempo de Projecto	Ajudas no Desenvolv.	Consumo de Espaço	Adaptabilidade às Alterações	Segurança
Dispositivos com Funções Específicas	O Mais Elevado	Baixos	Curto	Data Books	Elevado	Razoável	Baixa
Gate Arrays Full Custom	O Menos Elevado	Elevados	Longo	Software CAE Poderoso	Baixo	Fraca	Elevada
Dispositivos Lógicos Programáveis	Médio	Médios	Curto a Médio	Software CAE Poderoso	Médio	Boa	Elevada

As vantagens e desvantagens da lógica programável face aos seus concorrentes directos, lógica discreta e lógica integrada, são resumidas no quadro 2.2-1 (DATA I/O, 86),

comparando alguns dos parâmetros mais relevantes para o projecto de desenvolvimento de sistemas digitais.

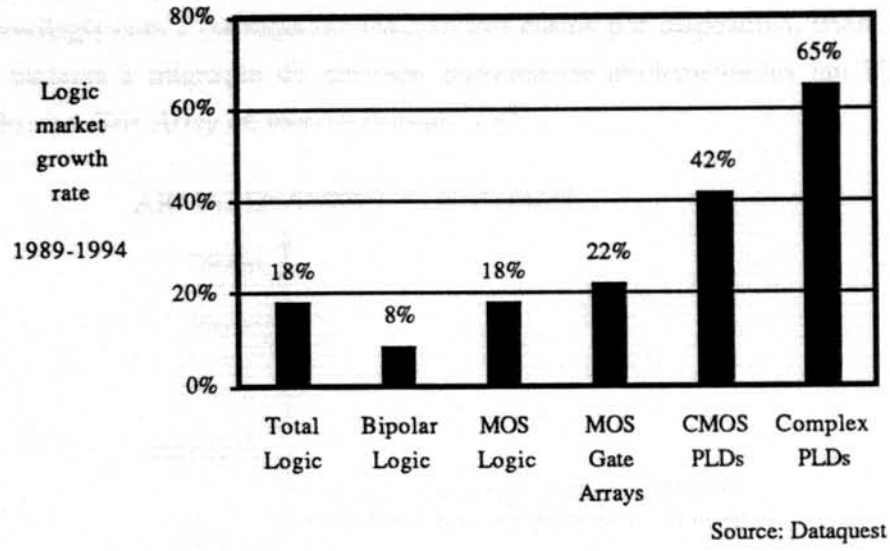


Gráfico 2.2-1 Projecção do consumo de circuitos integrados digitais

O gráfico 2.2-1 mostra-nos uma projecção de mercado efectuada pela Dataquest (Schulze, 91). A evolução prevista para os próximos anos, é a indicação mais segura da popularidade alcançada pela lógica programável na definição dos novos sistemas digitais face aos seus concorrentes directos.

2.2.1. Aplicações e metodologia de desenho

Um inquérito efectuado recentemente nos Estados Unidos (Small, 91b), permite obter conclusões interessantes e importantes acerca das aplicações de PLDs.

A primeira conclusão revela que muito poucos projectistas efectuam a compressão directa de sistemas implementados com lógica convencional. É mais eficiente redesenhar o sistema aproveitando todas as vantagens e especificidades dos PLDs.

A segunda conclusão, e talvez a mais surpreendente mostra, que a utilização de PLDs não termina na realização de protótipos ou em sistemas de pequenas séries. Os sistemas desenvolvidos são a maior parte das vezes projectos finais comercializados com PLDs. Esta conclusão contradiz resultados esperados pelos fabricantes de lógica programável, que estimavam em 30% do seu volume de negócios a transposição de projectos implementados em PLDs para *Gate Arrays*. Esta contradição será talvez explicada pela pressão do mercado,

em termos de tempo de vida útil de novos produtos, que não se compadece com os custos NRE (Nonrecurring Engineering) de execução de um projecto utilizando circuitos integrados do tipo *Gate Array*. Por outro lado, o desenvolvimento tecnológico, com o consequente aumento da quantidade lógica integrada, e a banalização da utilização deste tipo de tecnologia com a consequente redução dos custos por dispositivo, criará barreiras cada vez maiores à migração de circuitos inicialmente implementados em PLDs para soluções do tipo *Gate Array* ou mesmo *Standard Cell*.

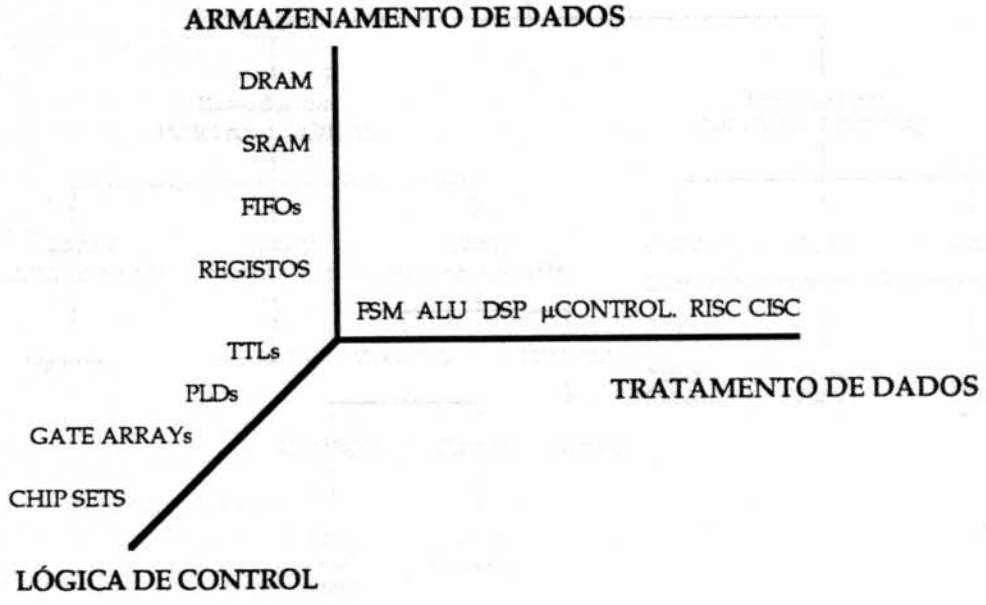


Fig. 2.2-1 Aplicações da lógica programável no presente. No futuro poderá incorporar funções relacionadas com o tratamento e armazenamento de dados.

No que se refere às aplicações de dispositivos do tipo PLD, elas são maioritáriamente funções de controlo (Small, 91a). Existem no entanto algumas aplicações potenciais, ver figura 2.2-1, nos eixos referentes ao armazenamento e tratamento de dados indiciando que, num futuro próximo, a lógica programável poderá incorporar funções até aqui destinadas a circuitos integrados de aplicação específica.

2.2.2. Análise tecnológica da evolução dos PLDs

É difícil, se não impossível, descrever e analisar toda a gama de produtos existentes no mercado. É no entanto possível analisar alguns dos produtos mais representativos, apresentados na figura 2.2-2, de forma a obter uma visão generalizada das múltiplas arquitecturas existentes neste tipo de circuitos.

Uma primeira análise revela-nos a existência de dois grandes grupos (Small, 91a). O

primeiro, baseado em Portas Lógicas, mais não é do que uma evolução lógica da arquitectura PAL. O segundo grupo, baseado em Células lógicas, é uma aproximação rudimentar da estrutura de um *Gate Array*. Existe ainda um terceiro grupo, os chamados PLDs dedicados, que apresentam uma estrutura muito particular e optimizada para a função que se propõem implementar.

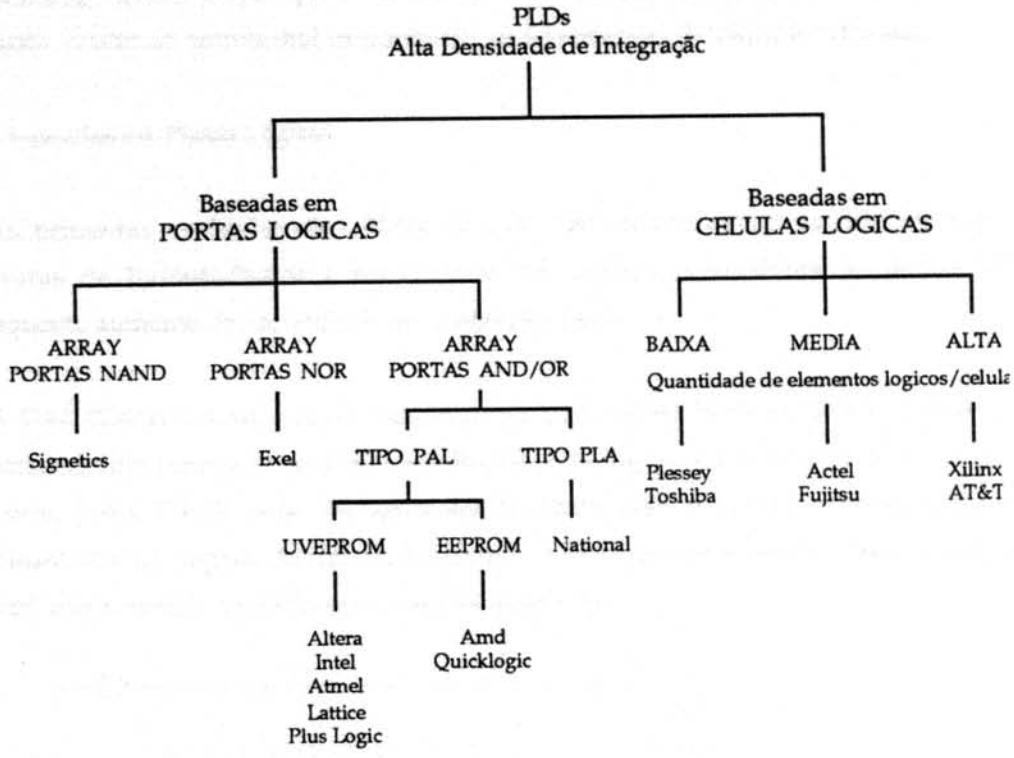


Fig. 2.2-2 Arvore representativa dos PLDs.

A única medida existente para a comparação da capacidade destes PLDs de alta densidade de integração é o chamado número de "portas equivalentes".

Nota: um circuito com um número X de "portas equivalentes", é um circuito que possuiria esse número X de portas se todos os seus elementos lógicos fossem representados por portas AND de duas entradas.

No entanto, e para além do facto de a indústria não ter encontrado um método único para o cálculo do número de "portas equivalentes", este tipo de métrica é muitas vezes desajustado do interesse de engenheiros e de projectistas de circuitos e sistemas.

Uma comparação baseada em aspectos tecnológicos e económicos, é com certeza um método mais de acordo com a realidade da execução de projectos envolvendo PLDs.

Dos aspectos tecnológicos podemos referir como mais importantes a tecnologia de fabrico (Bipolar, CMOS, ECL ou BiCMOS), o tempo de propagação interno, o número de

unidades lógicas básicas que contém, a configuração interna dessas unidades e ainda a configuração de Entrada/Saída de cada circuito.

São no entanto os aspectos económicos que condicionam muitas vezes a escolha de uma ou outra família de PLDs.

Por detrás dos aspectos económicos directos do custo de um determinado PLD com determinadas características que o tornam indicado para a execução de um projecto, existe um factor oculto de primordial importância, as ferramentas de desenvolvimento.

PLDs baseados em Portas Lógicas

As primeiras evoluções dos PLDs do tipo PAL centraram-se na flexibilização das estruturas de Entrada/Saída e no aumento do número equivalente de portas com o consequente aumento da capacidade de integração lógica.

A GAL (Generic Array Logic), desenvolvida pela Lattice Semiconductor (Marshall, 90), cuja macrocélula fundamental pode ser observada na figura 2.2-3, acrescenta ao plano OR fixo uma porta EXOR para configuração da polaridade do sinal resultante do bloco combinatório e um registo do tipo D. No plano AND registe-se a possibilidade de utilização do sinal realimentado da saída ou do seu complementar.

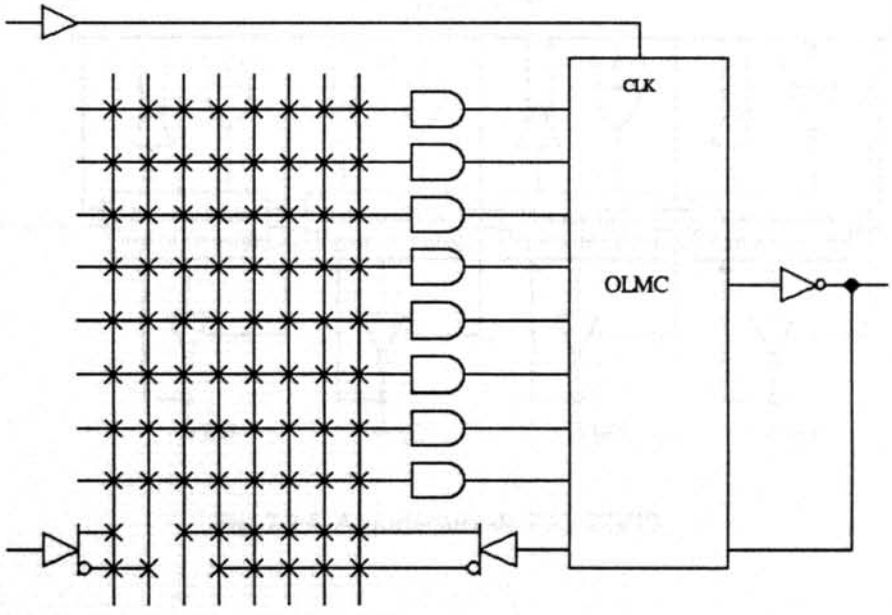


Fig. 2.2-3 Macrocélula de uma GAL.

Com a crescente aceitação pelos utilizadores deste tipo de circuitos, surgiram também novos requisitos ao nível do binómio velocidade/potência. Para satisfazer a necessidade de

PALs com tempos de propagação reduzidos foram introduzidas as primeiras versões ECL e GaAs. Para baixar o consumo de potência recorreu-se à tecnologia de fabrico que revolucionou todo o universo dos PLDs, a tecnologia CMOS.

A implementação mais bem conseguida de uma PAL flexível, foi sem dúvida a 22V10 da AMD (AMD, 90b) esquematizada na figura 2.2-4. Externamente apresenta 11 pinos dedicados de entrada, um pino dedicado de relógio e 10 pinos configuráveis de Entrada/Saída. Para além de uma arquitectura extremamente flexível, com destaque para o número variável de minitermos nas macrocélulas, escalado do número standard de 8 até a um máximo de 16, a PAL 22V10 sofreu uma mutação na tecnologia de fabrico, de bipolar para CMOS com o intuito de diminuir o seu consumo de potência, mantendo o seu nível de desempenho. Também disponível a versão CE22V10, utilizada em desenvolvimento e que pode ser reprogramada graças à utilização de transistores do tipo EPROM na definição da sua configuração interna.

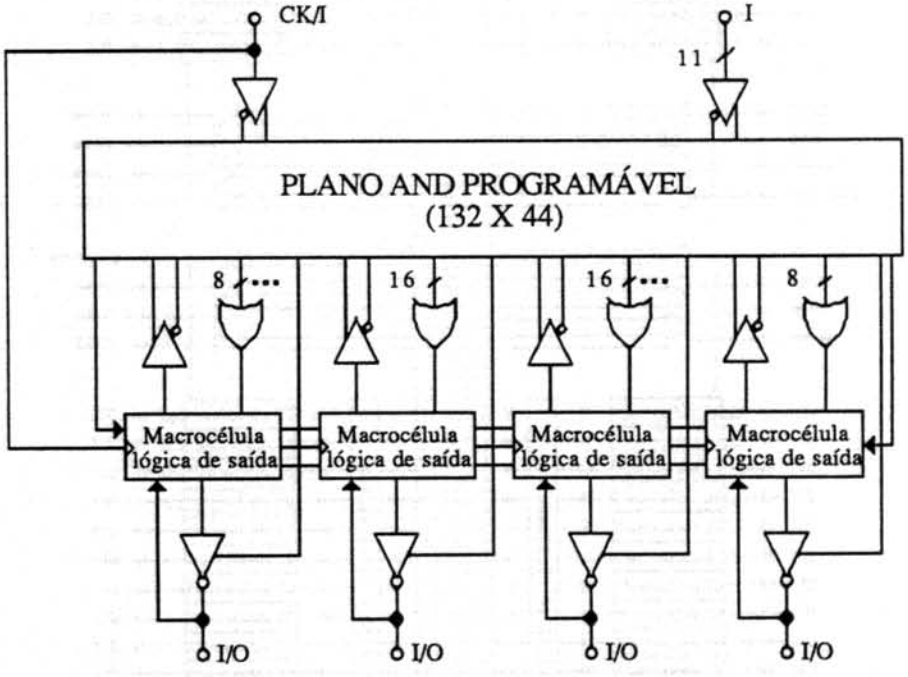


Fig. 2.2-4 Arquitectura da PAL 22V10.

Este tipo de PLDs oferece no entanto uma capacidade de integração relativamente baixa, apresentando como desvantagem fundamental um bloco sequencial reduzido e pouco flexível, que não favorece a integração de blocos lógicos tão populares como os contadores, *shift-registers*, máquinas de estados, comparadores, etc.

O aparecimento no mercado da família EPLD (Erasable Programmable Logic Device),

desenvolvida pela Altera (Intel e Texas Instruments são fontes alternativas), fabricada em tecnologia CMOS de 1.2µm, foi o primeiro passo no desenvolvimento de um novo tipo de PLDs com muito maior capacidade de integração (Altera, 88)(Intel, 87)(TI, 89).

Externamente, o EP1810, elemento da família com maior capacidade de integração lógica e cuja estrutura se mostra na figura 2.2-5, oferece 16 linhas de entrada dedicadas, 4 das quais podem ser utilizadas como entradas de relógio do sistema, uma por quadrante. Possui ainda 48 pinos de Entrada/Saída que podem ser individualmente configurados como pinos de entrada, de saída ou bidireccionais.

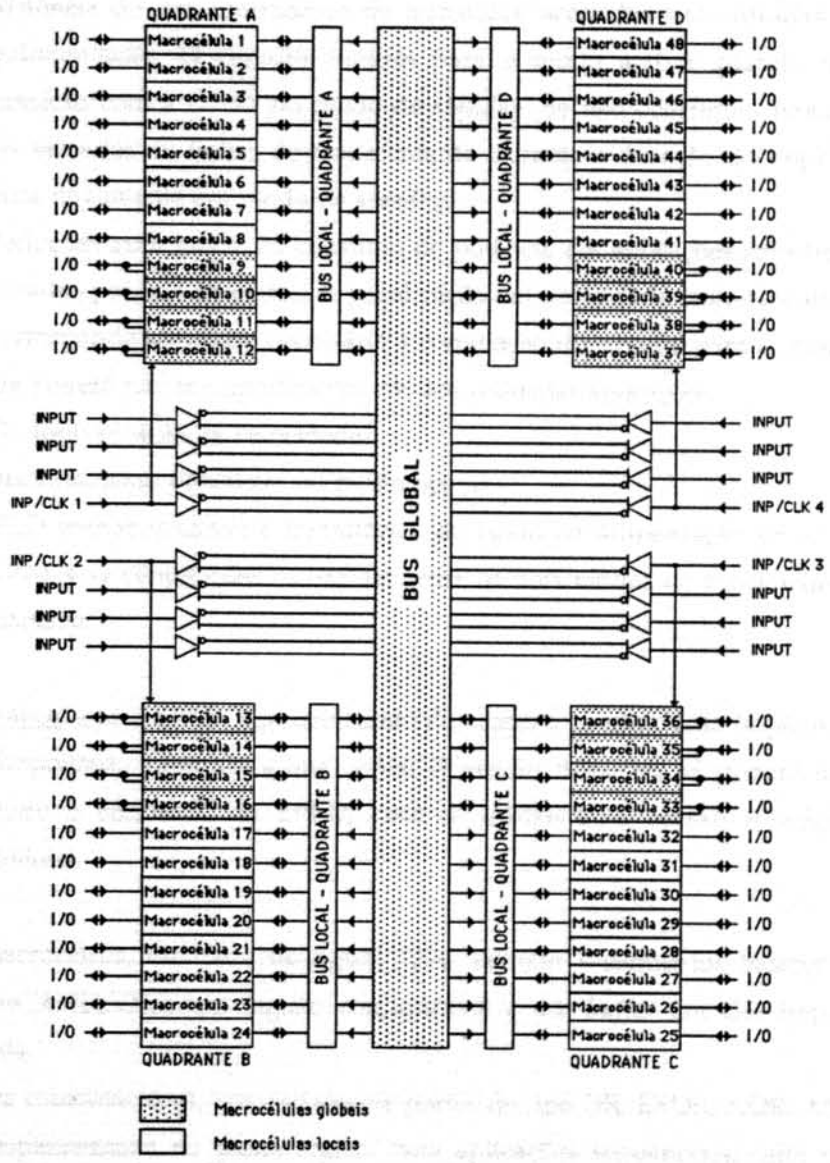


Fig. 2.2-5 Estrutura da EP1800/EP1810/5C180.

Internamente, a arquitectura da EP1810 é composta por uma série de macrocélulas agrupadas em 4 quadrantes. Cada quadrante é constituído por 12 macrocélulas com capacidade de comunicação entre si e com todas as linhas de Entrada/Saída através de um

barramento local. Quatro dessas macrocélulas possuem ainda capacidade de comunicação com os outros 3 quadrantes através de um barramento global.

Possui ainda duas opções de configuração que permitem controlar parâmetros de velocidade/consumo e segurança:

Bit Turbo, que optimiza o funcionamento do dispositivo em termos de velocidade ou consumo de potência.

Turbo Off (baixo consumo de potência)

- existência de um controlador de transições ocorridas nas entradas e linhas de realimentação. O circuito comuta para o modo activo quando detecta uma transição com a EPLD em modo de *standby*. Se não ocorrerem novas transições nas entradas ou linhas de realimentação durante a duração do impulso, a ELPD entra novamente em modo de *standby*.
- Reduções apreciáveis do consumo de potência em aplicações com transições nas entradas pouco frequentes. A penalização em velocidade situa-se entre os 0-30 ns (corresponde ao tempo necessário à comutação do circuito para o estado activo), o que poderá não ser significativo em determinadas aplicações.

Turbo On (optimização de velocidade)

- Funcionamento constante no modo activo.
- EPLD menos sensível a transitórios de ruído na alimentação uma vez que são evitadas as comutações constantes entre os dois modos de funcionamento: activo e *standby*.

Bit de Segurança, que controla a possibilidade de acesso à configuração implementada pelo utilizador, impedindo portanto a sua cópia. O estado deste bit só poderá ser alterado, apagando todo o conteúdo da EPLD, caso se utilize uma versão reprogramável de desenvolvimento.

Cada macrocélula, ilustrada na figura 2.2-6, contém 3 elementos básicos: um plano lógico (plano AND/OR), um registo configurável e um *buffer* em alta impedância de Entrada/Saída.

Toda a lógica combinacional, nomeadamente portas do tipo OR, EXOR, NOR, AND, NAND e NOT, é implementada no plano lógico. Para aplicações sequenciais, cada macrocélula contém um registo configurável num de 4 tipos possíveis: D, T, JK ou SR.

Este tipo de PLDs, apesar de muito mais flexível que as PALs, enferma de um mal comum. A utilização de um registo para implementação de uma qualquer função lógica significa na prática a inutilização de um pino de Entrada/Saída. Na melhor das hipóteses, e no caso de uma EP1810, existem 16 macrocélulas, designadas globais, em que é possível a

realimentação dos sinais registados para dentro da EPLD, mantendo o pino disponível como entrada.

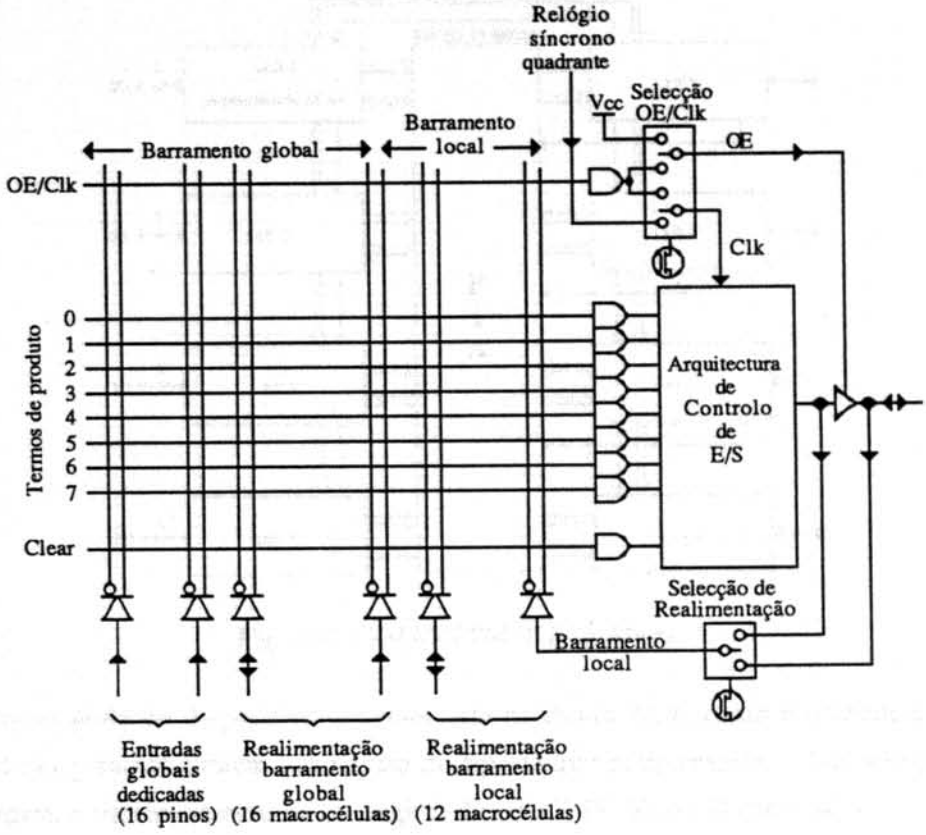


Fig. 2.2-6 Macrocélula local da EP1800/EP1810/5C180.

Esta desvantagem aparece corrigida no patamar seguinte dos PLDs. A família MAX 5000 da Altera (Altera, 90c), apresenta uma estrutura não muito diferente da família EPLD, como se constata na figura 2.2-7, com a particularidade de possuir macrocélulas "enterradas", sem qualquer ligação física a pinos do dispositivo.

As entradas externas da família MAX são ligadas a uma matriz designada PIA (Programmable Interconnect Array). Trata-se de facto, de uma matriz de cruzamento que permite a ligação de qualquer sinal interno ou entrada externa a qualquer pino de saída.

As entradas internas na PIA incluem linhas de realimentação das macrocélulas e de realimentação das linhas de Entrada/Saída. A realimentação da macrocélula permite a utilização do seu pino de E/S como entrada mantendo a macrocélula disponível internamente.

As saídas do PIA são naturalmente encaminhadas para os minitermos AND que alimentam o restante plano lógico e os registos nas macrocélulas do dispositivo.

Alguns dos minitermos AND têm funções específicas de controlo dos registos das macrocélulas, tais como o *output enable, preset, clear* e *clock*.

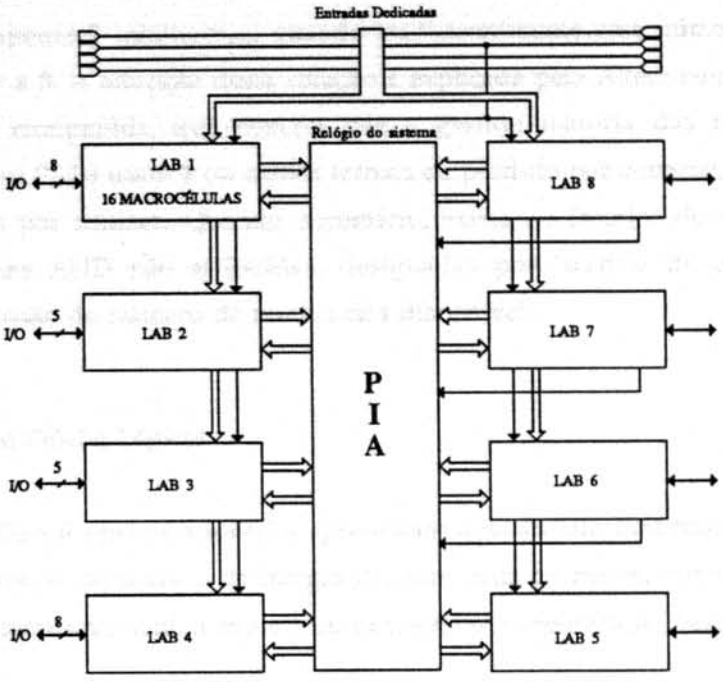


Fig. 2.2-7 O MAX EPM5128 da Altera.

A macrocélula do dispositivo, representada na figura 2.2-8, como é comum em PLDs CMOS desta geração, contém um registo do tipo D que pode emular, utilizando parte do plano lógico, o funcionamento de um registo do tipo T, JK, SR ou D com *enable*.

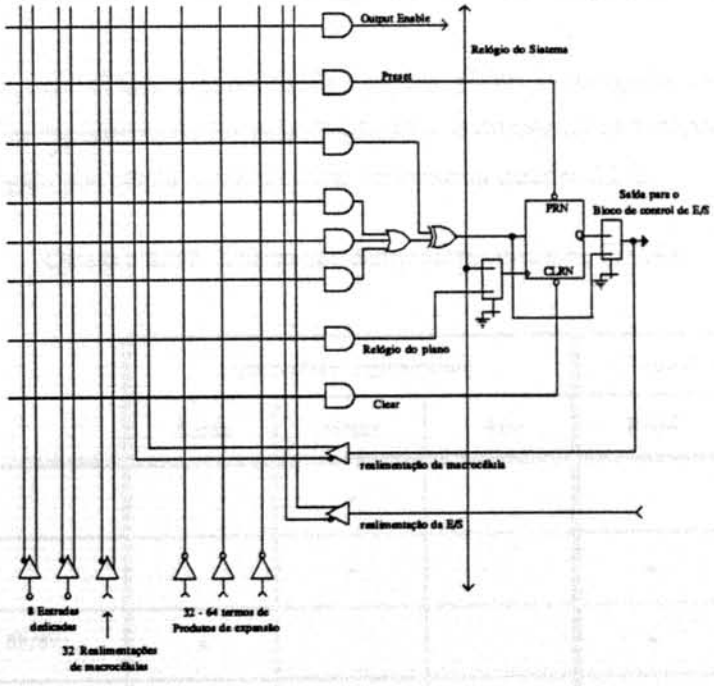


Fig. 2.2-8 Macrocélula de um elemento da família MAX da Altera.

Mas o facto que diferencia a família MAX de outros PLDs é o facto de cada porta OR ser alimentada por apenas 3 minitermos, quando tradicionalmente esse número é nas PALs igual ou superior a 8. A adopção desta solução é explicada pela Altera como resultado da investigação da companhia, que conclui que a grande maioria das funções lógicas implementadas em PALs usam 3 ou menos termos de produto por equação, deixando 5 ou mais minitermos por utilizar. Quando necessário, existe no interior de cada MAX um número de portas AND não atribuídas, designadas por "termos de expansão", que permitem a expansão do número de minitermos disponível.

PLDs baseados em Células Lógicas

Todos os PLDs até aqui mencionados apresentam uma arquitectura baseada numa PAL. Com maior ou menor capacidade de integração, com mais ou menos minitermos, com um registo mais ou menos versátil, a sua estrutura mantém a organização base do decano dos PLDs.

Existe contudo um subgrupo dos PLDs, geralmente designado por FPGAs (Field Programmable Gate Arrays), que compreende dispositivos lógicos programáveis com altíssimas capacidades de integração, e com uma estrutura elementar substancialmente diferente da tradicional. Ao contrário do que o seu nome possa fazer supor, os FPGAs em pouco se assemelham a *Gate Arrays* convencionais. O termo LCA (Logic Cell Array), muito mais apropriado, é no entanto uma marca registada da Xilinx, o pioneiro dos fabricantes de FPGAs.

Internamente, um FPGA é constituído por um plano rectangular de células lógicas, cercado por células específicas de Entrada/Saída. O que diferencia os FPGAs referenciados é a quantidade de lógica por célula como se pode verificar na quadro 2.2-2.

Quadro 2.2-2 Diferenças elementares entre os FPGAs

	Quantidade lógica/célula			Tecnol. de programação	
	Baixa	Média	Alta	RAM	Antifusível
Actel (Actel, 90)		•			•
Fujitsu		•		•	
Plessey (Plessey, 88/89)	•			•	
Toshiba	•			•	
Xilinx (Xilinx, 87)			•	•	

A quantidade de elementos lógicos por célula é um compromisso entre complexidade e *fan-in*. Se essa quantidade for reduzida, então cada bloco lógico será alimentado por um número também reduzido de entradas. O *fan-in* não será problema, mas o dispositivo necessitará de diversos blocos lógicos para realizar uma determinada função.

Se, por outro lado, a quantidade de elementos lógicos por célula for elevada, serão necessários menos blocos para implementar a mesma função. No entanto, um bloco lógico complexo deverá, forçosamente, ser alimentado por um maior número de entradas, criando em determinadas situações, problemas de *fan-in* que poderão exceder a disponibilidade de conexões internas do dispositivo.

O primeiro FPGA a aparecer no mercado foi o LCA da Xilinx. A sua arquitectura base é ilustrada na figura 2.2-9.

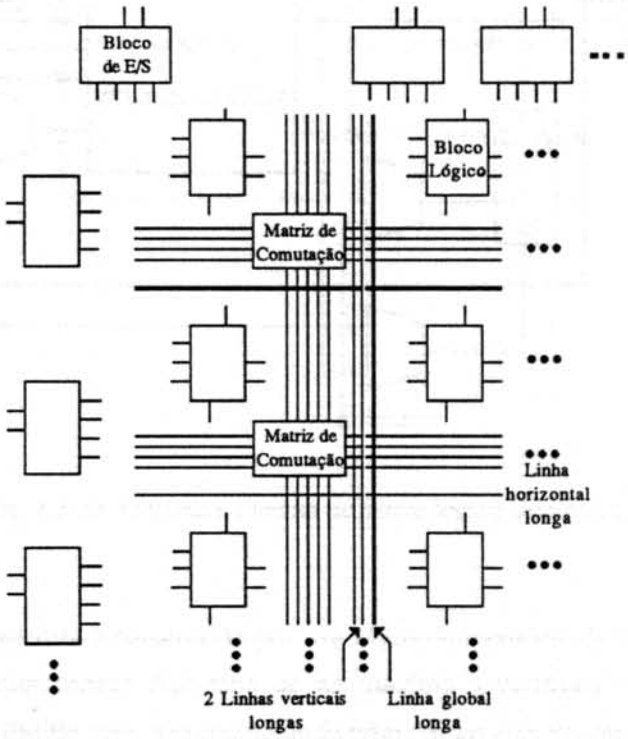


Fig. 2.2-9 Organização interna de um LCA da Xilinx.

Apresenta a estrutura base de um FPGA, com um plano a 2 dimensões de blocos lógicos programáveis, cercado por células de Entrada/Saída que efectuam a interface entre os blocos lógicos e o mundo exterior. O plano de blocos lógicos é atravessado por uma teia de linhas horizontais e verticais constituindo as linhas de interligação entre os blocos de E/S e os blocos lógicos, e destes entre si.

A intersecção das linhas horizontais e verticais é assegurada por matrizes de interligação

do tipo *CrossBar*. A diferença em relação a uma matriz convencional reside no facto de estas serem configuradas uma só vez quando o dispositivo é inicializado.

Possui ainda linhas horizontais e verticais adicionais, as chamadas linhas longas, destinadas aos sinais com tempos de propagação críticos e que percorrem todo o LCA (ex: relógio, sinais de controlo, etc).

O plano de blocos lógicos configuráveis destina-se à implementação da função descrita pelo utilizador. Cada bloco lógico, representado na figura 2.2-10, é constituído por três secções distintas: uma secção combinatória, um registo e uma secção de interligação e controlo. É alimentado por 4 entradas dedicadas e por uma entrada especial de relógio.

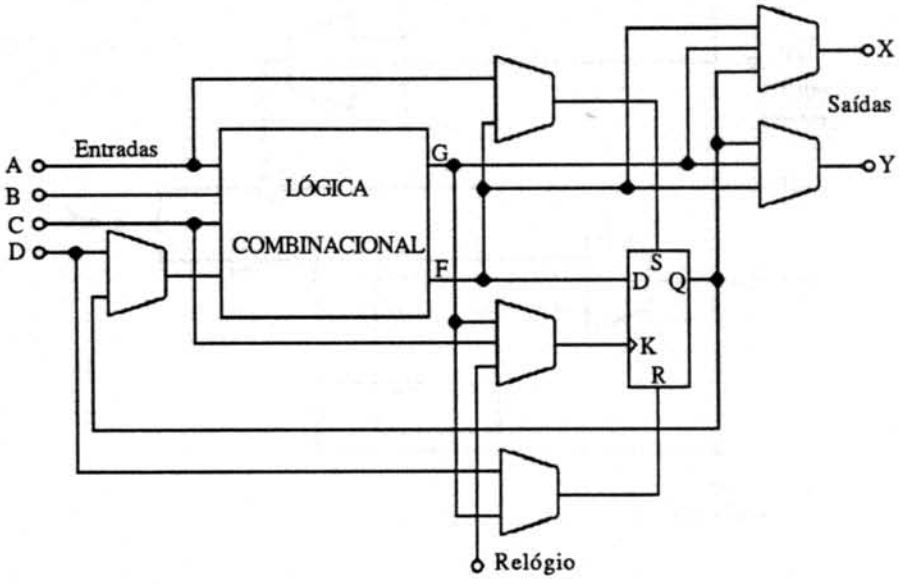


Fig. 2.2-10 Estrutura interna do bloco lógico de um LCA.

A secção combinatória é constituída por um bloco de memória de 16 posições, podendo implementar qualquer função Booleana de no máximo 4 variáveis, segundo o esquema tradicional de uma PROM (ver Arquitecturas Fundamentais dos PLDs). As variáveis podem ser seleccionadas entre as 4 entradas dedicadas e a saída Q do registo. Numa configuração simples de 4 variáveis as saídas F e G são idênticas. Usando a configuração alternativa, que permite descrever duas funções combinatórias de 3 variáveis, F e G serão os resultados lógicos de cada implementação.

Se utilizado, o registo pode ser configurado como tipo D activo à transição, ou tipo D activo ao nível, ambos com controlo assíncrono de *set* e *reset*. A entrada de dados do registo resulta da função lógica presente em F. A entrada de relógio ou *enable* de cada registo pode ser alimentada pela entrada dedicada de relógio, pela entrada C ou ainda pela função lógica presente em G. É possível seleccionar a transição activa do sinal de relógio. As duas saídas

do bloco, podem ser alimentadas quer por uma das funções lógicas presentes em F ou G, quer pela saída Q do registo.

As macrocélulas de Entrada/Saída efectuam, como se ilustra na figura 2.2-11, a interface entre os pinos físicos do LCA e o núcleo lógico interno. Cada macrocélula de E/S inclui um percurso configurável de entrada e um *buffer* programável de saída. Na entrada é possível seleccionar, quer o percurso directo quer o percurso registado, e na saída é possível controlar o sinal de *enable* do *buffer*.

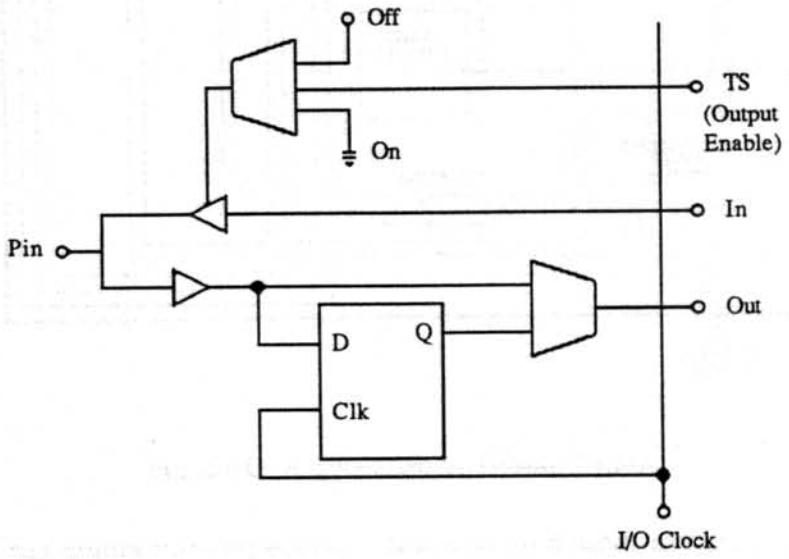


Fig. 2.2-11 Estrutura interna do bloco de E/S de um LCA.

PLDs dedicados

Para além dos PLDs de aplicação genérica, existem ainda os PLDs dedicados, englobando os dispositivos dedicados a aplicações específicas, como por exemplo controlo de barramentos, interface com barramentos ou implementação de máquinas de estados.

A maior parte dos PLDs existentes hoje em dia possui uma capacidade de *drive* variável entre 6 e 24 mA, suficiente para interligações entre circuitos integrados e barramentos internos. Contudo, a conexão de placas de circuito impresso a barramentos, tais como o VME, requere da maior parte dos sinais capacidades de *drive* de 48 mA.

Fabricado para resolver estas situações, o PLX 448 (Bursky, 87), oferece dois conjuntos de 4 macrocélulas de Entrada/Saída flexíveis, com capacidades de *drive* respectivamente de 24 e 48 mA.

O EPS 448 (Altera, 88), esboçado na figura 2.2-12, implementa uma máquina de estados síncrona Moore.

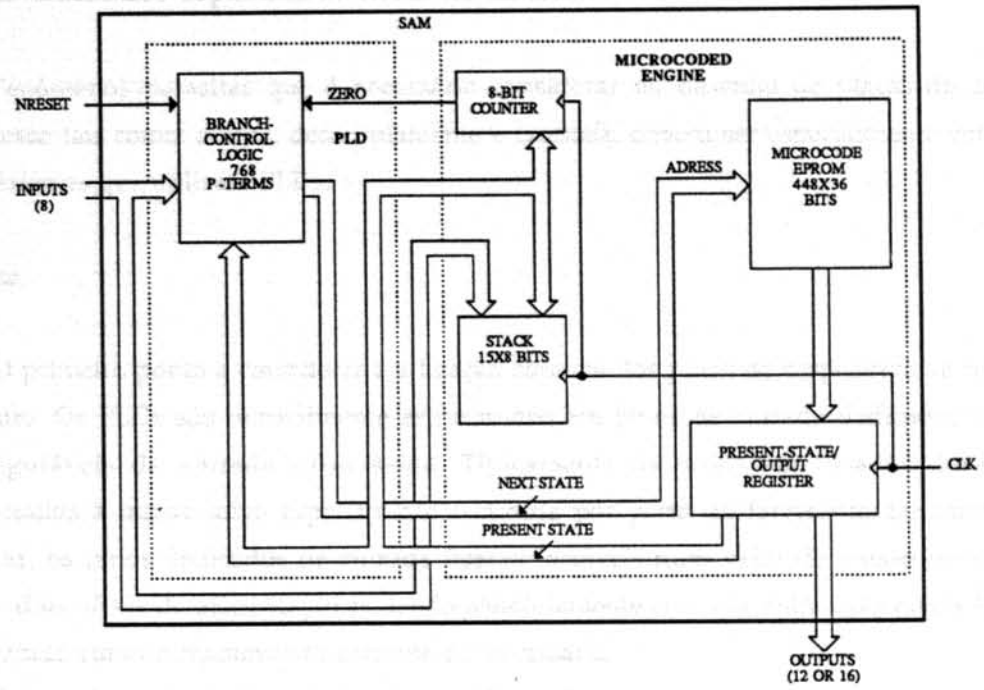


Fig. 2.2-12 Arquitectura do EPS448 (SAM).

Graças à sua arquitectura específica, a descrição de funções utilizando o SAM (Stand Alone Microsequencer) mais se assemelha com a do código de um microprocessador do que propriamente com a descrição de uma máquina de estados:

- o contador e a *stack* podem ser invocados por instruções do tipo: *call, return, push counter*, etc.
- podem ser criadas sub-rotinas encadeadas.
- podem ser executadas contagens em tempo real.

2.3. OS PLDs NOS NOVOS SISTEMAS DIGITAIS

A utilização de lógica programável na descrição de sistemas digitais, acarreta para o engenheiro de projecto cuidados especiais (Tralka, 90). Um PLD deve assim ser considerado, não só como um elemento pertencente a um sistema mas também como um sistema composto de elementos lógicos.

Grandes blocos lógicos são hoje em dia integrados num único PLD, e problemas que parecem estar relacionados com o funcionamento do dispositivo, resultam muitas vezes de

má tradução ou concepção das funções lógicas nele implementadas.

2.3.1. Cuidados especiais ao nível do sistema

Fenómenos parasitas que é necessário considerar no desenho de placas de circuito impresso tais como: cargas, desacoplamento e *crosstalk*, devem ser especialmente cuidados em sistemas que utilizem PLDs.

Cargas

O primeiro ponto a considerar é a ligação correcta dos pinos do dispositivo ao resto do circuito. Os PLDs são normalmente estruturados em pinos de entrada dedicados e pinos configuráveis de entrada e/ou saída. Tipicamente os pinos não usados devem ser conectados à massa salvo especificação contrária por parte do fabricante. Se deixados a flutuar, os pinos dedicados de entrada ficarão instáveis num valor de tensão intermédio entre dois níveis de alimentação podendo aleatóriamente comutar entre dois níveis lógicos, induzindo ruído e consumindo corrente desnecessária.

A consideração seguinte recai sob o problema das cargas resistiva e capacitiva nos pinos de saída dos PLDs. A carga resistiva limita o *fan-out* da saída, podendo ser necessária a utilização de *line-drivers* se o valor de corrente fornecido pelo dispositivo se revelar insuficiente.

Quanto às cargas capacitivas, não afectam o valor do *fan-out*, mas alteram os tempos de subida e descida dos sinais presentes nas saídas dos PLDs (Fenton, 91). Esta situação deriva da equação:

$$I = \frac{dQ}{dT} \quad (Q = CV) \qquad <=> \qquad dT = \frac{d(CV)}{I}$$

I -> corrente fornecida pelo PLD

Q -> carga

T -> tempo

C -> capacidade de carga

V -> variação de tensão na saída

Da equação resulta que, enquanto com o aumento do valor da corrente fornecida pelo PLD nas saídas decrescem os tempos de subida e descida dos sinais respectivos, com o aumento do valor da carga capacitiva esses mesmos tempos aumentam. Como o valor de corrente fornecida pelo dispositivo é intrínseco à sua tecnologia de fabrico, resulta que para

para reduzir os tempos de subida e descida dos sinais à saída dos PLDs, é necessário controlar o valor da carga capacitiva presente nas suas saídas.

Desacoplamento

Outro problema potencial relacionado com fenómenos de corrente é a chamada "flutuação na massa" resultante dos tempos diminutos de comutação entre níveis lógicos nos PLDs actuais. Tenderá a ocorrer quando múltiplas saídas comutarem simultaneamente, induzindo um transitório elevado do fluxo de corrente no caminho de massa do dispositivo. Esta variação do fluxo de corrente, associada à indutância do caminho, traduzir-se-à numa subida do valor da tensão da massa (V=L(dI/dT)) noutros elementos do sistema que partilhem o mesmo trajecto. A subida da sua tensão de referência, pode implicar, a subida do valor de tensão do nível lógico associado, nos sinais presentes nas saídas destes elementos. No pior dos casos, outros elementos do sistema, que não partilhem esse caminho de massa, poderão descodificar erradamente os sinais provenientes dos elementos afectados.

Existem PLDs cujas células de saída são especialmente concebidas para minimizar o problema. Um método eficaz de minimização do problema será o correcto desacoplamento dos pinos de alimentação usando os valores recomendados pelos fabricantes. Estes condensadores, actuarão como reservatórios, isolando os caminhos de massa dos fluxos transitórios de corrente originados nos circuitos integrados.

Crosstalk

O último ponto a considerar relaciona-se com o *routing* das pistas na placa de circuito impresso. O *crosstalk* resultará do acoplamento indutivo e capacitivo entre pistas paralelas.

O acoplamento indutivo ocorre quando o fluxo de corrente numa pista gera um campo magnético que induz um fluxo de corrente parasita na pista paralela adjacente.

O acoplamento capacitivo ocorre quando duas pistas paralelas se comportam como as placas de um condensador tentando manter um valor de tensão constante entre si.

O desenho cuidado da placa de circuito impresso, evitando pistas paralelas demasiado próximas, ajudará a minimizar o problema de *crosstalk*.

2.3.2. Cuidados especiais ao nível do PLD

Sem grande surpresa, muitos dos problemas surgidos em sistemas que utilizam lógica programável, estão relacionados com as descrições lógicas contidas nesses mesmos

dispositivos. Uma simulação, funcional e temporal, cuidada, poderá evitar muitas das falhas existentes na descrição lógica interna do PLD. Atenção especial deverá ser dispensada à geração de relógios e ao processo de minimização lógica.

Geração do relógio

A utilização de lógica sequencial obriga à inclusão de esquemas mais ou menos sofisticados de relógio. Os problemas podem surgir quando os relógios são resultado de funções lógicas combinatórias complexas (Ryherd, 91).

Sinais com transições lógicas mal definidas ou quasi-simultâneas poderão ocasionar *glitches* de tamanho suficiente para activar erradamente os registos que controlam (Jackson, 91). Se é difícil o controlo de relógios gerados por lógica complexa com circuitos integrados discretos, mais difícil se torna com PLDs, onde esta é implementada em níveis múltiplos. A situação torna-se ainda mais complicada, pois poucas vezes é possível exercer um controlo preciso no modo como a ferramenta de desenvolvimento minimiza a descrição lógica.

A sincronização, sempre que possível, de todo o sistema integrado no PLD, por um só relógio síncrono permitirá eliminar este potencial problema.

Por exemplo, é possível converter a saída potencialmente ruidosa de um multiplexador, como o da figura 2.3-1, num sinal fiável, pela simples adição de um registo, eliminando possíveis transições parasitas.

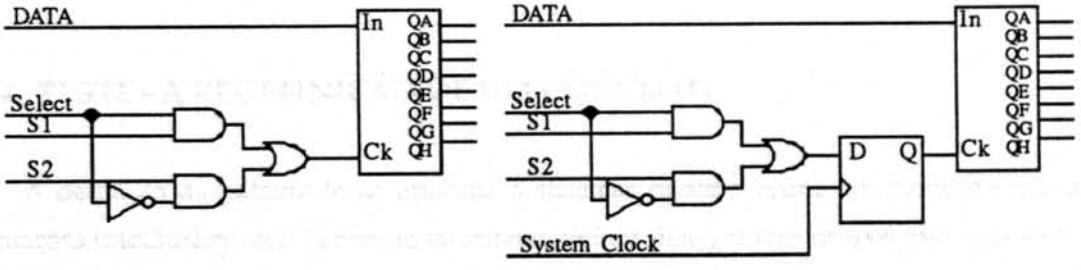


Fig. 2.3-1 Eliminação do ruído potencial na entrada de relógio de um *shift-register*.

Outra prática comum dos projectistas de sistemas digitais, é a descodificação de um dado estado de um contador, sendo este o valor descodificado utilizado para efectuar o *load* assíncrono do mesmo contador. Se não se garantir que todos os registos do contador transitam de estado, simultaneamente, a saída descodificada poderá apresentar *glitches*. Uma saída ruidosa poderá carregar um valor incorrecto no contador.

Uma solução para o problema é obviamente a utilização de contadores com *load* síncrono. Outra forma, esquematizada na figura 2.3-2, consiste em registar a saída,

descodificando-a um estado mais cedo, garantindo um sinal isento de ruído na entrada do contador.

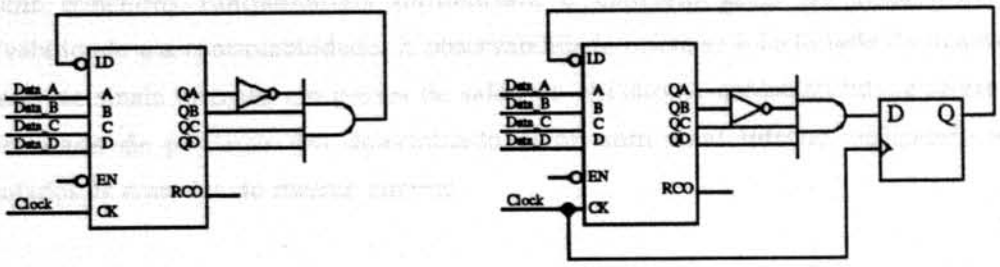


Fig. 2.3-2 Utilização correcta do valor descodificado de um contador para a sua própria reinicialização.

Minimização lógica

Um dos métodos de criar atrasos, muito utilizado em circuitos baseados em lógica discreta, consiste na associação série de portas inversoras, ou outro tipo de portas disponíveis, numa configuração idêntica (ex: NAND com as entradas curto-circuitadas). Esta solução não resulta em PLDs, pois o módulo de minimização da ferramenta de desenvolvimento eliminará toda a cadeia em série. A implementação de atrasos em PLDs deve obedecer a regras particulares consoante o dispositivo seleccionado (ver capítulo 2.6.3).

2.4. TESTE - A REDEFINIÇÃO DE UM CONCEITO

A definição da palavra teste, aplicada a sistemas digitais, reúne um conjunto lato de situações (McCluskey, 86). Podem-se no entanto definir dois grandes grupos que englobem o problema da testabilidade em electrónica:

1 Teste concorrente - engloba o teste com o circuito em funcionamento normal. Códigos de paridade e sinais de indicação do modo de funcionamento são normalmente os mais usados.

2 Teste explícito - engloba os testes efectuados antes do circuito entrar no seu funcionamento normal. Inclui os teste de fabrico dos circuitos integrados, teste do correcto funcionamento da placa de circuito impresso, testes de manutenção e de reparação.

Neste capítulo pretende-se abordar, de uma forma superficial o problema do teste

explícito no âmbito do desenvolvimento tecnológico actual. A referência à palavra teste ou outra consigo relacionada significará pois uma referência ao teste explícito.

Dois conceitos fundamentais influenciam o conceito geral de testabilidade: a observabilidade e a controlabilidade. A observabilidade refere-se à facilidade de determinar o estado de sinais internos em portos de saída do circuito. A controlabilidade refere-se à possibilidade de produzir um determinado valor num sinal interno, aplicando sinais controlados às entradas do mesmo circuito.

Ainda não há muito tempo poucos eram os projectistas de sistemas que consideravam os problemas envolvidos no teste dos seus circuitos. Hoje em dia, o teste tornou-se um factor essencial para o sucesso de um projecto, numa manifestação clara da inversão de valores porque está a passar o desenvolvimento de novos sistemas (Eichelberger, et al. 78). A integração verificada nos novos sistemas, dominados pelos circuitos VLSI (Very Large Scale Integration), pelo recurso a circuitos do tipo SMD (Surface Mount Device), com pinos de geometria reduzida e colocados por vezes em ambos os lados da placa, e pela utilização de placas de circuito impresso multicamada, em que o acesso de pontas de prova a sinais definidos nas camadas internas é extremamente difícil, tornou os métodos até aqui utilizados, com relevo para o *bed of nails*, onerosos e inviáveis no teste de placas de circuito impresso (Matos, et al. 89).

As novas técnicas apontam para o desenvolvimento de circuitos integrados com capacidades intrínsecas de teste. Até à sua utilização plena, no entanto, as capacidades de teste que oferecem terão aplicações restritas, devendo numa primeira fase coexistir com os métodos tradicionais (Donnell, 91).

A adição de capacidade interna de teste será por enquanto uma função da família disponível, e do custo da lógica adicional a implementar. Num desenho simples (1000-2000 portas equivalentes) não será por certo económico a implementação de células de teste. Num patamar mais elevado (20000-40000 portas equivalentes), sem a utilização de técnicas DFT (Design-for-Testability) estruturadas não será possível desenvolver um conjunto de vectores de teste completo, num período aceitável de tempo (Novellino, 91).

A importância das técnicas DFT cresce enormemente com o aumento da complexidade dos novos projectos (DasGupta, et al. 85). As técnicas fundamentais de DFT, com relevo para as baseadas em *scan*, não só permitem o teste de funcionalidade da placa de circuito impresso, como também facilitam a criação automática de vectores de teste e o teste interno ao funcionamento dos blocos lógicos do próprio circuito integrado (Markowitz, 90).

Dos vários métodos de teste baseados em *scan*: LSSD (Level Sensitive Scan Design), Scan Path, Scan Set, Random-Acess Scan, Hybrid Scan, RTS (Register-Transfer Scan) e Boundary Scan, realça-se este último, que pelas suas potencialidades se transformou num standard,

abrindo caminho para que num futuro próximo, se verifique a adopção sistematizada da norma pelos fabricantes de circuitos integrados, permitindo o teste simples e económico de placas de circuito impresso (Parker, 89). E talvez num futuro não muito distante o problema da testabilidade em PLDs, englobe o próprio dispositivo, e a necessidade de teste dos blocos lógicos que o constituem.

No mundo da lógica programável, um primeiro fabricante, a Xilinx, anunciou o lançamento do primeiro FPGA compatível com o standard de Boundary Scan, o XC4005. Aproveitando uma arquitectura interna convidativa, com a separação do núcleo lógico configurável da estrutura de Entrada/Saída, a implementação da lógica adicional de teste de Boundary Scan não representa para os FPGAs um consumo exagerado de recursos (Bursky, 91b).

2.4.1. Boundary-Scan

Os trabalhos de criação do standard Boundary-Scan começaram em Novembro de 1985 com a formação do JTAG (Join Test Action Group) que incluía os gigantes do mundo da electrónica: AT&T, British Telecom, DEC, HP, IBM, Motorola, Philips, Siemens e Texas Instruments (Pradhan, et al. 87).

Em Abril de 1988, surgiu a primeira proposta técnica de criação do Boundary-Scan, sendo o desenvolvimento do standard transferido para a alçada do IEEE.

O standard IEEE 1149.1, designado "Port Acess Test and Boundary Scan Arquithecture", foi aprovado em 15 de Fevereiro de 1990, sendo o documento final publicado em Maio do mesmo ano. O standard define os circuitos de teste a implementar em circuitos integrados e a interface de comunicação entre si (Novellino, 90)(Maunder, et al. 91)(Bennetts, et al. 91).

A estrutura de teste é implementada por 4 pinos adicionais no circuito integrado: TDI (Test Data-in), TDO (Test Data-out), TMS (Test Mode Select) e TCK (Test Clock). Para se conseguir a utilização deste número mínimo de pinos, as instruções e os dados são transmitidos em série.

O barramento de teste permite examinar ou alterar o estado de todos os sinais de Entrada/Saída de um circuito integrado, daí o seu nome, Boundary-Scan. Conectando os sinais TMS e TCK de todos os circuitos integrados em comum, e encadeando o TDO de um com o TDI do seguinte, será possível testar toda uma placa de circuito impresso com este simples barramento.

A figura 2.4-1 ilustra a infra-estrutura BST de um circuito (Quinell, 90b). Os dados de teste passam por três registos, registo de instruções, registo de *bypass* e registo Boundary-

Scan, deslocando-se da entrada (TDI) para a saída (TDO).

O *TAP controller* (Test Acess Point controller), uma máquina de estados que regista e controla os vários registos e multiplexadores, existe em todos os circuitos integrados do tipo Boundary-Scan. O sinal TMS comanda as alterações de estado do *TAP controller*, que ocorrem na transição activa do TCK. Dependendo do seu estado, os circuitos de teste podem ficar inactivos, fazer circular os dados pelos registos, sem afectar o modo de funcionamento normal do circuito integrado, ou ainda executar o comando de teste armazenado no registo de instruções.

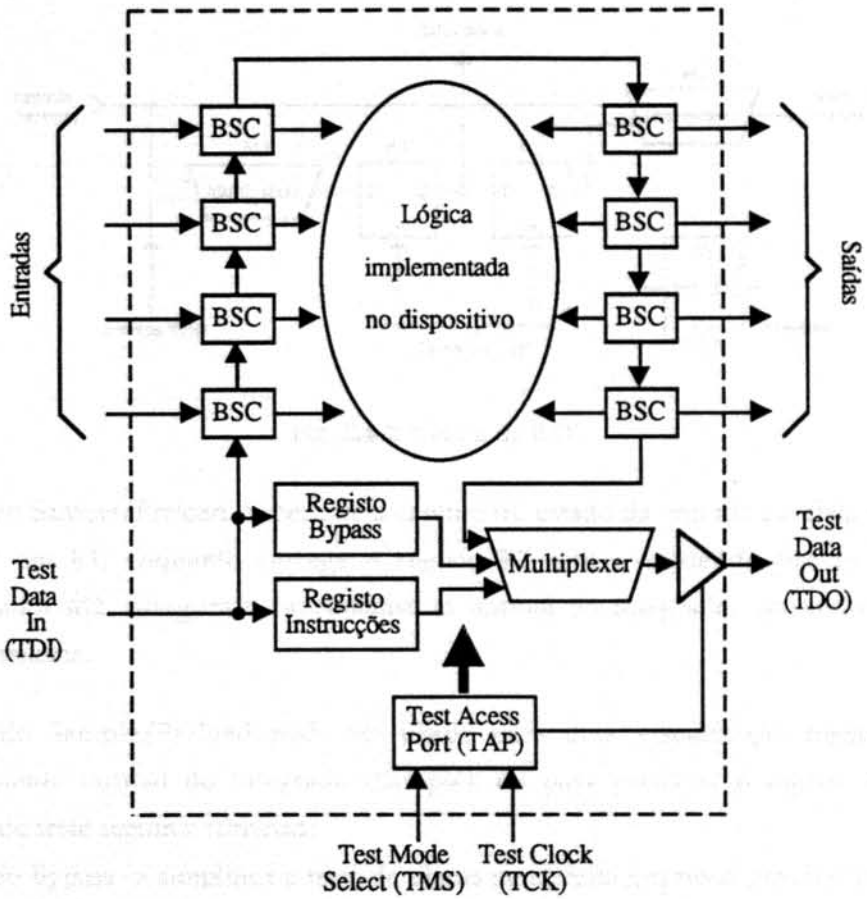


Fig. 2.4-1 Infra-estrutura básica de BST implantada num circuito integrado.

A norma define a existência de três comandos obrigatórios de teste: External test, Sample/Preload e Bypass. Por opção do fabricante, o circuito integrado poderá incluir outros comandos como: Internal test, Self-test, ID test.

Os comandos obrigatórios fornecem os meios para efectuar o teste exaustivo à placa de circuito impresso. Os testes opcionais complementam os obrigatórios, permitindo ainda o teste individual do circuito integrado.

• Comando External test -> controla o comportamento da célula Boundary-Scan, representada na figura 2.4-2, dependendo do tipo de sinal a que esta está conectada.

Um registo conectado a uma saída ou a uma linha de controlo do circuito integrado, usará o valor de R2 para substituir o sinal de saída normal do circuito integrado. Um registo conectado a uma entrada, capturará o estado da linha na transição de TCK a seguir ao comando de External test.

O interesse desta dupla função consiste em permitir impor os níveis lógicos dos sinais de saída de um circuito integrado e verificar a sua recepção noutros elementos, com um só comando. Usando padrões de teste adequados, podem-se isolar curto-circuitos, circuitos abertos ou falhas do tipo *stuck-at*.

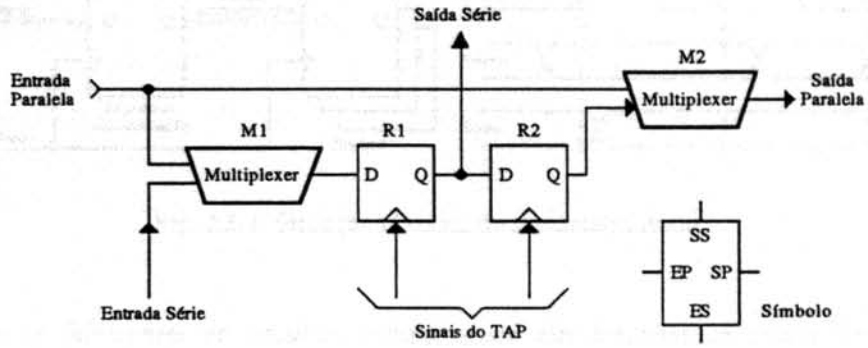


Fig. 2.4-2 Célula de BST.

• Comando Sample/Preload -> permite a captura do estado da entrada paralela do circuito integrado em R1, enquanto carrega o registo R2 com o conteúdo prévio de R1. O multiplexador M2, assegura o funcionamento normal do integrado, deixando passar os dados do sistema.

O comando Sample/Preload pode ser usado para uma visualização instantânea do funcionamento normal do integrado (Sample), ou para preparar o registo R2 para o comando de teste seguinte (Preload).

• Comando Bypass -> simplifica o teste de placas de circuito impresso com muitos circuitos integrados, permitindo testar selectivamente. A utilização do registo de *bypass*, permite remover, a menos do próprio registo, um dispositivo da cadeia de circuitos integrados a testar.

2.5. O PROBLEMA DA METASTABILIDADE EM PLDs

Diversos sistemas digitais apresentam, hoje em dia, algum tipo de interface assíncrona. A amostragem dos sinais provenientes de tal interface, no registo de sincronização, pode incorrer em falhas causadas por um fenómeno, designado de metastabilidade (Marino, 81). Como qualquer registo, é caracterizado pelos tempos de *setup* e *hold*, requerendo que as

transições do sinal de entrada ocorram em pontos determinados relativamente às transições activas do relógio. Contudo, no caso de um registo de sincronização, as transições dos sinais assíncronos à entrada ocorrem aleatóriamente face às transições activas do relógio do sistema, podendo violar as restrições temporais especificadas pelos fabricantes (Bowns, 91).

O estado metastável, de acordo com a figura 2.5-1, é portanto um estado resultante de uma violação temporal, criando na saída do registo de sincronização (algumas vezes referido como árbitro) um sinal não definido e instável quanto ao seu valor lógico.

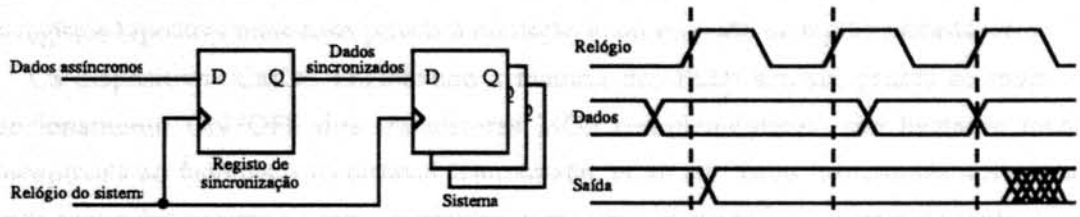


Fig. 2.5-1 Situação clássica de metastabilidade.

Nota: todos os fabricantes de circuitos lógicos, sejam eles discretos ou integrados, incluem especificações dos registos, com realce para os parâmetros de "setup" e "hold". Estes parâmetros indicam que o sinal de entrada não deve variar X nanosegundos antes da transição activa do relógio e que deve permanecer no mesmo estado Y nanosegundos após a mesma transição. Qualquer transição do sinal verificada entre estes dois instantes é considerada inválida.

É possível fazer uma analogia (Masteller, 91) entre o estado metastável e uma bola em equilíbrio precário no topo de um monte (ver figura 2.5-2). Pequenas oscilações da bola no topo do monte não modificarão a sua posição. Qualquer perturbação mais forte, equivalente à presença de ruído num sistema digital, alterará a posição de equilíbrio podendo a bola deslizar para qualquer dos pontos estáveis no sopé do monte. O lado para o qual a bola deslizará é indeterminado.

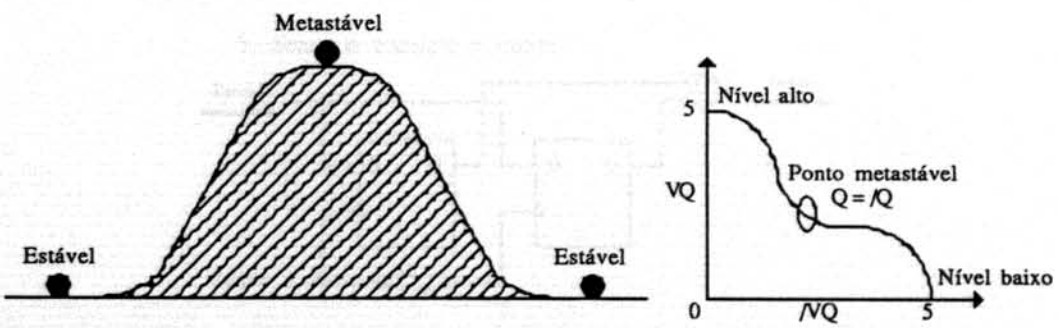


Fig. 2.5-2 Analogia entre uma bola em equilíbrio precário no topo de um monte e o fenómeno da metastabilidade.

2.5.1. Tecnologia

A nova geração de circuitos baseados em tecnologia bipolar, FAST TTL e PLDs, apresenta características de comutação e de frequência máxima de funcionamento substancialmente melhoradas. No entanto, o funcionamento constante dos transistores bipolares na zona linear da sua curva característica e as elevadas capacidades de carga que apresentam, tornam os registos bipolares mais susceptíveis à oscilação e/ou retensão na região metastável.

Os dispositivos CMOS, englobando a maioria dos PLDs actuais, graças ao modo de funcionamento ON/OFF dos transistores MOS complementares, são bastante menos susceptíveis ao fenómeno metastável (Horstmann, et al. 89). Estes transistores apresentam ainda reduzidos valores de carga capacitiva entre nós e de carga nos andares de saída. Todas estas características tornam o tempo de comutação entre os dois níveis lógicos diminutos, aumentando a capacidade de resolução do circuito a uma situação de metastabilidade.

2.5.2. Caracterização

Muitos autores relatam experiências efectuadas em circuitos para prever a probabilidade de metastabilidade em dispositivos (Dingman, 91). As diferentes teorias e circuitos apresentados podem ser sintetizados nos 3 grupos da figura 2.5-3:

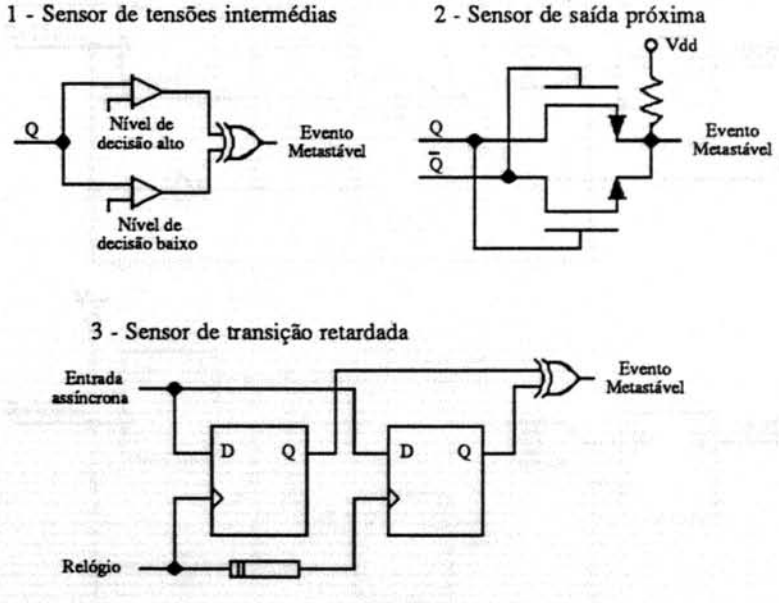


Fig. 2.5-3 Circuitos possíveis para a detecção de eventos metastáveis.

1 - Sensores de tensões intermédias: dois comparadores de tensão são usados para determinar se a tensão de saída do registo permanece entre as duas tensões determinadas, correspondentes aos dois níveis lógicos.

2 - Sensor de saída próxima: determina quando as saídas Q e /Q apresentam valores de tensão idênticos (situação de metastabilidade).

3 - Sensor de transição retardada: se o sensor estiver separado do sinal metastável por uma ou mais portas, como no caso de uma PLD, o sinal metastável propriamente dito não poderá ser detectado. O circuito de teste deverá concluir da ocorrência de metastabilidade por outro meio. Usando um sensor de transição retardada, a entrada é amostrada em dois instantes: t1 e t2. Se estes sinais forem dissemelhantes, significa que o circuito esteve na região de metastabilidade em t1.

O sensor de transição retardada é a única implementação possível para teste da metastabilidade em PLDs (Altera, 90a). O circuito apresentado na figura 2.5-4, permite não só a detecção de falhas por metastabilidade mas também a medição da frequência máxima de funcionamento do registo em teste. Note-se que o circuito apresentado, apesar de capturar todas as falhas, não regista os eventos solucionados antes da transição activa seguinte do relógio a jusante do sincronizador, uma vez que estes eventos não contribuem para erros de funcionamento do sistema.

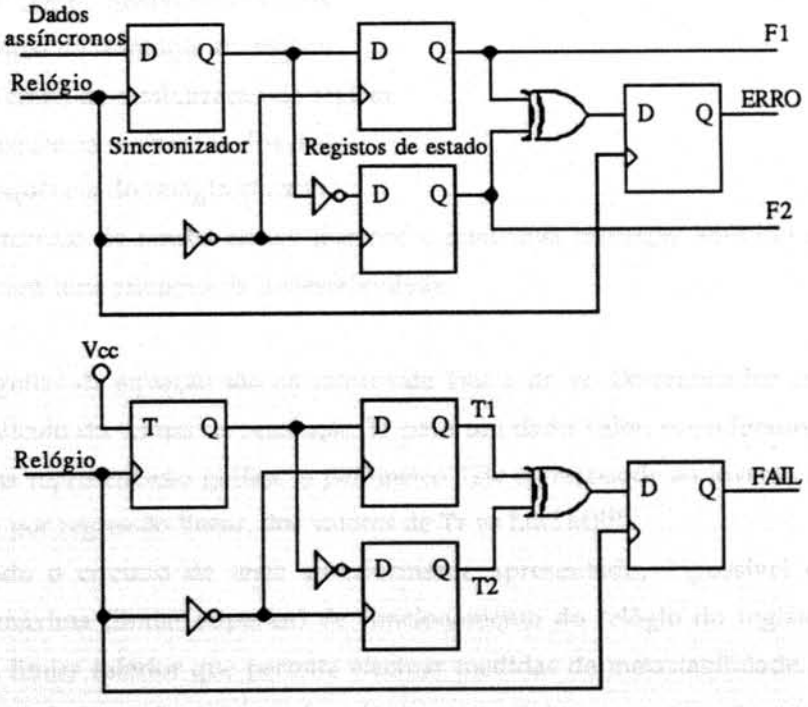


Fig. 2.5-4 Aplicação prática de um circuito do tipo "Sensor de transição retardada".

Uma das particularidades dos circuitos apresentados é a possibilidade de variar o ponto de amostragem dos registos de estado face à transição activa do sincronizador. Modificando o *duty cycle* do relógio do sistema obtemos um atraso controlado entre as 2 transições do relógio, o que nos permite capturar eventos metastáveis com tempos de resolução variáveis.

2.5.3. Cálculo do TMEF (Tempo Médio Entre Falhas)

É possível medir as características de metastabilidade dos registos e calcular o tempo de resolução que deve ser concedido ao sincronizador para obter TMEFs aceitáveis.

É importante frisar que o facto de a saída de um registo alcançar o estado metastável não é por si só importante. A metastabilidade só causará erros de resolução ao sistema se o fenómeno não desaparecer antes da transição activa do relógio do sistema síncrono a jusante do sincronizador.

Torna-se pois importante calcular o tempo de resolução que é necessário conceder ao registo, para um correcto dimensionamento do sistema a jusante do sincronizador.

$$TMEF = \frac{e^{Tr/Tsw}}{FcFdW}$$

TMEF -> Tempo médio entre falhas
Tr -> Tempo de resolução do registo
Tsw -> Tempo de estabilização do registo
Fd -> Frequência assíncrona dos dados
Fc -> Frequência do relógio síncrono
W -> Intervalo de tempo crítico durante o qual uma transição simultânea do relógio provocará uma situação de metastabilidade

As incógnitas da equação são os valores de Tsw e de W. Determinados estes valores é possível o cálculo do tempo de resolução Tr para um dado valor, considerado aceitável, de TMEF. Numa representação gráfica, o parâmetro Tsw corresponde ao inverso do declive da recta, obtida por regressão linear, dos valores de Tr vs Ln(TMEF).

Utilizando o circuito de teste anteriormente apresentado, é possível determinar a frequência máxima (limiar superior) de funcionamento do relógio do registo, bem como calcular um limiar inferior que permita efectuar medidas de metastabilidade. Considere-se como limiar inferior a frequência de relógio que origine uma situação metastável com intervalos de aproximadamente 1 minuto. Notando que Fc deve ser igual a pelo menos 2Fd,

selecciona-se entre estes dois limiares, um valor para o produto FcFd.

Variando Fc, de forma a manter FcFd constante, obtêm-se diversos valores de Tr (Tr=1/Fc - 1/Fmax). Para cada valor obtido, o valor de TMEF será igual ao intervalo de tempo considerado a dividir pelo número de falhas contabilizado. Construindo o gráfico semi-logarítmico é possível obter os valores de Tsw (inverso da inclinação da recta obtida por regressão linear dos pontos calculados), e de W (intersecção da recta com o eixo TMEF dividido por FcFd).

Exemplo:

Nota: os dados aqui apresentados, bem como a equação que permite calcular o TMEF, foram recolhidos de uma nota de aplicação da Cypress Semiconductor.

Consideram-se os seguintes valores para o cálculo de Tr de um PLD do tipo CY7C344-20:

TMEF = 10 anos (315 E6 s)
Fc = 20 MHz
Fd = 10 MHz
W = 0.966 ps
Tsw = 0.233 ns

Tr = Tsw(Ln(TMEF) + Ln(FcFdW))

Tr = 0.233 E-9 (Ln(315 E6) + Ln(20 E6 x 10 E6 x 0.966 E-12) = 5.8 ns

Nos casos em que o valor calculado do TMEF se revelar insuficiente deve considerar-se a utilização de um sincronizador de estágios múltiplos como forma de reduzir as falhas por metastabilidade. Uma simples cascata de registos aumentará a fiabilidade total do sistema.

2.6. IMPLEMENTAÇÕES OPTIMIZADAS PARA PLDs

2.6.1. Método One-Hot

Quase todas, senão todas, as funções de controlo são implementadas hoje em dia sob a forma de máquina de estados. O método tradicional de a descrever, a codificação compacta da sequência de estados, favorece a utilização de circuitos com um número restrito de

registos e um vasto plano de lógica combinatória, como é o caso da PAL. Neste método, uma máquina de 2^n estados necessita de um número de registos igual a "n".

No método OHE (One-Hot Encoding, concebido originalmente pela High-Gate Design) - uma máquina de 2^n estados consome idêntico número de registos, utilizando um registo por estado na sua definição. A sua filosofia de funcionamento é extremamente simples: um só registo activo implica um só estado activo (Knapp, 90).

Para o utilizador conservador, e numa primeira aproximação, o método OHE traduzir-se-ia num desperdício de recursos que implicariam, provavelmente, um PLD de capacidade mais elevada.

A sua utilização, apesar de extensível a todo o universo dos PLDs, PALs inclusivé, é optimizada em estruturas ricas em registos e com blocos lógicos combinatórios reduzidos, como é o caso dos FPGAs. Nestes dispositivos, o método OHE significa geralmente menos recursos e melhor desempenho.

Relembre-se que os FPGAs são PLDs de alta densidade contendo um plano extenso de blocos lógicos configuráveis rodeados por conexões programáveis. Na maioria dos casos um bloco lógico suporta um número reduzido de entradas (3-5). Em contraste, uma macrocélula de uma PAL tem ligações com todas as entradas e realimentações do circuito. Esta diferença fundamental significa que enquanto a PAL pode implementar uma função lógica de várias entradas num só nível de lógica, um FPGA poderá necessitar de utilizar vários blocos lógicos em cascata.

Outra vantagem do método OHE reside na semelhança da sua estrutura com a de um vulgar *shift-register*. A performance da máquina de estados permanecerá aproximadamente constante com o aumento do número de estados, enquanto no método tradicional tal situação implicará uma redução significativa do desempenho devido ao aumento do bloco lógico combinatório de descodificação dos estados.

A descrição da máquina de estados é feita de forma sistemática, e a resolução do problema dos estados ilegais é facilmente solucionada.

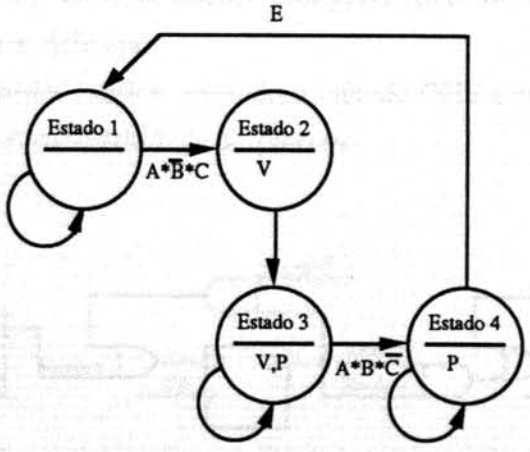


Fig. 2.6-1 Exemplo - diagrama de estados.

Consideremos o exemplo de uma máquina de 4 estados descrita segundo o diagrama tradicional na figura 2.6-1. Dentro de cada "balão", para além da identificação do estado encontram-se as saídas activas correspondentes. As transições de estados incluem as condições lógicas que as geram.

O estado 1 é definido como o estado inicial ou de arranque (power-up). A estrutura particular deste estado, apresentada na figura 2.6-2. A colocação de dois inversores imediatamente antes e depois do registo, deve-se ao facto de, após o arranque, todas as saídas dos registos de um FPGA apresentarem o valor lógico "0". O primeiro inversor garante assim o arranque correcto da máquina de estados, enquanto o segundo repõe a coerência na descrição.

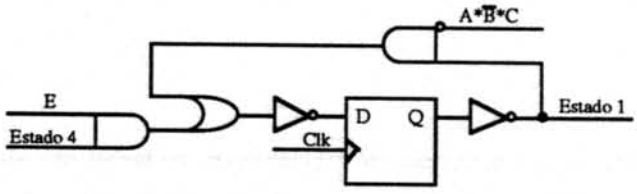


Fig. 2.6-2 Definição do estado de arranque.

Definida a condição inicial, podemos configurar a transição lógica para o estado seguinte. Para tal, contabilizam-se as transições de entrada no estado, adicionando uma unidade se existir a condição de defeito de permanência no mesmo. Define-se então uma porta OR com um número de entradas igual ao número determinado no passo anterior. Para cada entrada da porta OR, constroi-se uma porta AND do estado anterior e respectiva lógica de transição.

No caso de o estado possuir uma condição de defeito, representando a malha de realimentação, constroi-se uma porta AND do estado presente e do inverso de todas as condições de transição para um novo estado. Esta porta AND alimenta a entrada adicional da porta OR anteriormente definida.

A descrição dos restantes estados, segundo o método OHE é efectuada na figura 2.6-3. Realce-se a simplicidade e automaticidade do processo.

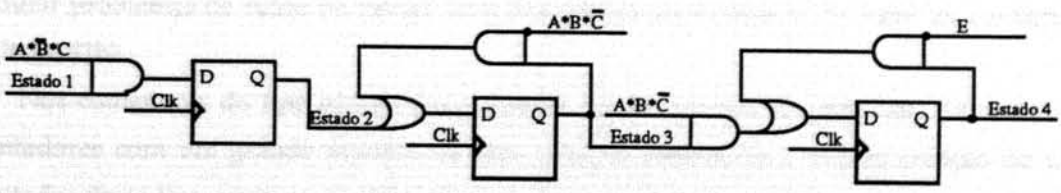


Fig. 2.6-3 Uma máquina de estados segundo o método OHE.

Definição das saídas

Na definição da lógica de saída três situações podem ocorrer:

- saídas activas durante um dos estados
- saídas activas durante estados múltiplos, contíguos
- saídas activas durante estados múltiplos, alternados

A resolução do primeiro caso é directa, com a correspondência da saída ao estado. As restantes possibilidades poderão ser solucionadas com uma simples descodificação, ou, se a situação o justificar, com a utilização de um registo do tipo SR.

Estados ilegais

Um ponto comum em todas as implementações de máquinas de estados é a prevenção da corrupção do funcionamento do sistema por estados ilegais. No caso de máquinas de estados implementadas segundo o método tradicional, o compilador utilizado adiciona lógica para evitar ou recuperar de situações anómalas. Na aproximação OHE, um estado ilegal significa dois registos activos simultaneamente. Por definição tal situação não poderia ocorrer. A sincronização das entradas com o relógio do sistema, que no método tradicional acarretaria custos lógicos consideráveis, evitará situações instáveis de transições das entradas, simultâneas com o relógio síncrono da máquina de estados.

Outra possibilidade será a utilização da entrada de *clear* assíncrono de cada registo. Como só deverá existir um estado activo em cada instante, a saída do seu registo poderá efectuar o *reset* de outros estados.

2.6.2. Contador do tipo código de Gray

Os contadores binários, cuja leitura dos valores de saída é directa, ocupam um lugar de destaque na metodologia de desenho de muitos projectistas. Um dos problemas que poderá ocorrer na sua utilização advém da situação de comutação múltipla das saídas, podendo induzir problemas de ruído ou causar uma descodificação incorrecta do valor de contagem (Altera, 91b).

Nos contadores do tipo código Gray, apenas um bit comuta de cada vez. Contudo em contadores com um grande número de bits, a lógica necessária à implementação de um contador deste tipo cresce para além da capacidade normal de um PLD. A adição de um bit suplementar, *odd bit*, pode reduzir a lógica requerida para implementação de um contador

do tipo código Gray (Beacher, 91).

Uma sequência tradicional de contagem de 4 bits segundo o código será:

0000
0001
0011
0010
0110
0111
0101
0100
1100 ...

O padrão do código não inclui nenhuma sequência repetitiva que favoreça a redução lógica da implementação do contador. A adição do *odd bit* permitirá a criação de uma tal sequência através da comutação simultânea de 2 bits. Esta situação não afectará a robustez tradicional do código uma vez que este bit extra não é descodificado no funcionamento normal do contador.

Contador (MSB -> LSB)	"Odd" bit	Acção
0000	0	transita de estado o bit 0
0001	1	transita de estado o bit 1
0011	0	transita de estado o bit 0
0010	1	transita de estado o bit 2
0110	0	transita de estado o bit 0
0111	1	transita de estado o bit 1
0101	0	transita de estado o bit 0
0100	1	transita de estado o bit 3
1100	0	transita de estado o bit 0
1101	1	transita de estado o bit 1
1111	0	transita de estado o bit 0
1110	1	transita de estado o bit 2
1010	0	transita de estado o bit 0
1011	1	transita de estado o bit 1
1001	0	transita de estado o bit 0
1000	1	transita de estado o bit 5

O novo padrão indica a comutação do bit menos significativo quando o *odd bit* é "0". Na situação contrária, é o bit mais significativo adjacente ao "1" menos significativo que transita. Este esquema reduz o número de termos de produto, necessários à implementação de um contador do tipo código de Gray com um qualquer número de bits, a menos de 4.

A implementação de um *reset* síncrono facilita a programação do valor terminal de contagem. Será necessário actuar apenas nos bits cujo estado lógico seja "1", no momento em que o sinal de *reset* esteja activo.

2.6.3. Atrasos programáveis

O termo atraso programável identifica um elemento digital cujas transições de saída seguem o sinal de entrada com um determinado atraso de propagação que depende de uma ou mais variáveis de controlo.

A grande parte das aplicações de atrasos programáveis pode ser classificada em dois grandes grupos: contínuos e discretos.

No primeiro caso encontramos atrasos programados para cancelar ou compensar a variação de determinados componentes do sistema. São utilizados na distribuição de relógios, interfaces de transmissão, equipamento de teste temporal, etc. A forma da onda de saída é geralmente idêntica à da entrada não sofrendo alteração no elemento de atraso (Feldman, et al. 91).

As duas configurações da figura 2.6-4 podem ser implementadas em PLDs. O "desperdício" de blocos lógicos, não aproveitados ou partilhados por outras funções, torna esta implementação indicada apenas para os dispositivos lógicos programáveis com macrocélulas "enterradas". Os atrasos obtidos dependem fortemente da arquitectura do PLD e dos parâmetros temporais das suas primitivas lógicas.

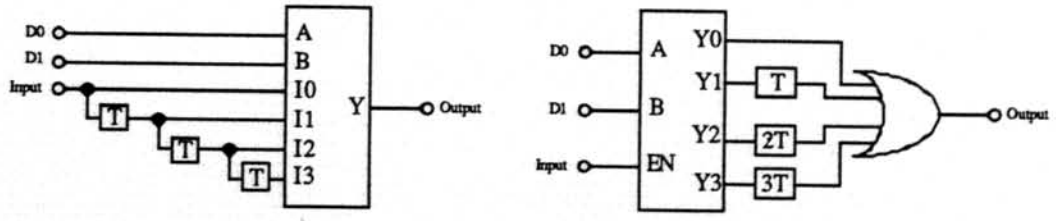


Fig. 2.6-4 Circuitos de atraso programável contínuo, passíveis de integração em PLDs.

As aplicações de atrasos programáveis discretos usam o atraso para iniciar eventos em instantes determinados. Incluem geradores de amostragem, equipamento de teste por estímulos, etc. Muitas vezes, apenas uma transição do sinal de entrada é significante. Neste

caso, o atraso é controlado pela transição de um sinal de *trigger*, fixando os instantes de transição do sinal de saída.

Os exemplos óbvios de atrasos discretos incluem contadores e circuitos organizados em cascata, com atrasos gerados em múltiplos de um relógio ou sinal de referência.

3. FERRAMENTAS DE DESENVOLVIMENTO DE PLDs

3.1. INTRODUÇÃO RETROSPECTIVA

As ferramentas de desenvolvimento de PLDs, que no passado desempenharam um papel preponderante, no desenvolvimento e aceitação da arquitectura PAL, em detrimento da arquitectura PLA, são hoje em dia um suporte indispensável ao utilizador de lógica programável.

A primeira ferramenta comercial, o PALASM (PAL Assembler), foi introduzida no mercado pela MMI (Monolithic Memories Incorporation) no final dos anos 70. Apesar de restrita à síntese de PALs, obteve um enorme sucesso comercial que mantém, contribuindo decisivamente para o sucesso de aceitação dos PLDs (MMI, 87).

Na versão inicial do programa, cujo fluxograma de funcionamento é representado na figura 3.1-1, a descrição da função a implementar na PAL é efectuada por meio de equações Booleanas num editor de texto comum.

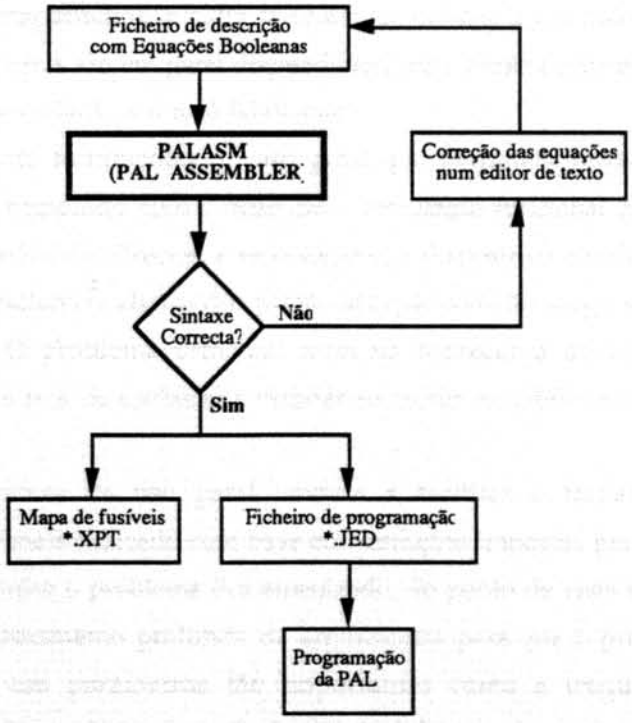


Fig. 3.1-1 Fluxograma de funcionamento do PALASM.

O ficheiro compilado pelo PALASM, após verificação de sintaxe, produz dois novos ficheiros, o ficheiro *.XPT, representando, na estrutura interna da PAL, as conexões definidas pelo utilizador, e o ficheiro *.JED, descrito no formato standard JEDEC, utilizado pelo programador na configuração física do dispositivo.

A evolução das ferramentas de desenvolvimento de lógica programável tem acompanhado a par e passo o desenvolvimento tecnológico dos PLDs. Cientes da sua importante contribuição na exploração exaustiva das novas arquitecturas dos PLDs, os utilizadores reclamam ferramentas não só fáceis de utilizar, mas também fiáveis e eficientes.

3.2. SELECÇÃO DE UMA FERRAMENTA DE DESENVOLVIMENTO

Por detrás dos aspectos económicos directos do custo de um determinado PLD com determinadas características que o tornam indicado para a execução de um projecto, existe um factor oculto de primordial importância, as ferramentas de desenvolvimento (Leibson, 90).

Paralelamente à criação de um dado PLD, ou família de PLDs, cada fabricante produz, um pacote de "software" específico para a descrição, compilação, simulação de projectos e programação de dispositivos, dessa família de PLDs. Como não existe, na esmagadora maioria dos casos, compatibilidade entre ferramentas de desenvolvimento ou ficheiros por estas criados, e como estas são em geral dispendiosas, uma escolha acarreta muitas vezes por si só a dependência em relação a um só fabricante.

Existem no entanto ferramentas de uso geral que permitem projectar com PLDs de fabricantes distintos, decidindo após a descrição e simulação funcional de um projecto, com base em critérios económicos directos, e tecnológicos, o dispositivo a utilizar (Milne, 89).
Convém no entanto esclarecer alguns dos problemas que poderão surgir se se optar por uma solução deste tipo. O problema principal recai na sobrecarga de arquitecturas que o engenheiro projectista terá de conhecer e estudar se quiser rentabilizar convenientemente o seu trabalho.
Este tipo de programas de uso geral tendem a facilitar o trabalho do utilizador, seleccionando o PLD mais indicado com base em restrições impostas pelo projectista. Se do ponto de vista económico o problema fica amenizado, do ponto de vista técnico ele subsiste. É necessário um conhecimento profundo da arquitectura para que o projecto em execução seja bem sucedido em parâmetros tão importantes como a frequência máxima de funcionamento, ou o atraso de propagação de um sinal da entrada para a saída.

Se esta divisão em dois grandes grupos, ferramentas de utilização específica e ferramentas de uso geral, contribuiria por si só para confundir o utilizador menos atento, a utilização de termos idênticos ou similares na descrição de funções dissemelhantes no seu modo de funcionamento e no resultado que apresentam, poderão confundir o mais perspicaz dos utilizadores.

Apresenta-se na figura 3.2-1 uma das ferramentas mais completas de desenvolvimento de PLDs, o MAX+PLUS II da Altera. Baseados na sua estrutura, procuraremos clarificar algumas designações menos objectivas e ajudar a compreender a importância de determinados aspectos fundamentais na selecção consistente de uma ferramenta de desenvolvimento de PLDs (Altera, 91a).

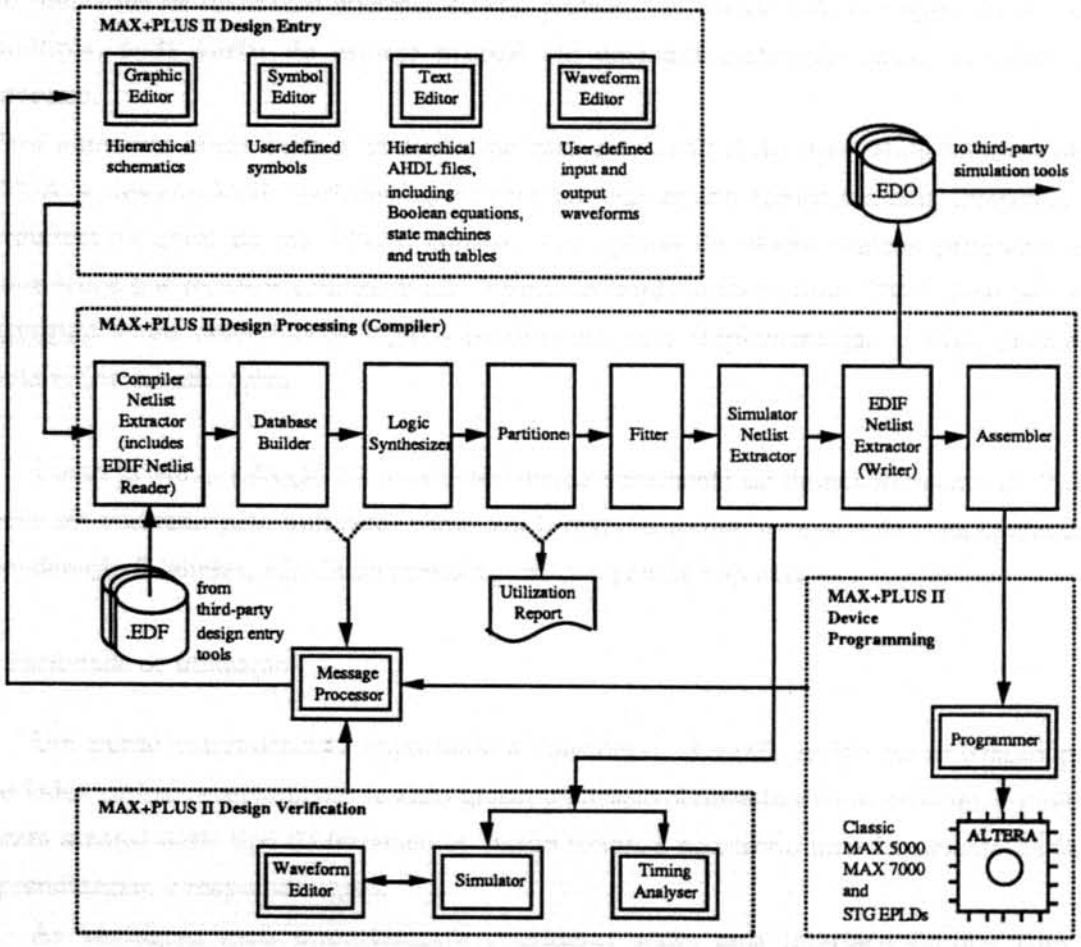


Fig. 3.2-1 Uma ferramenta completa de desenvolvimento de PLDs: MAX+PLUS II.

Como se pode observar, uma ferramenta de desenvolvimento engloba diversos módulos, cada qual com uma função específica, estruturados segundo uma sequência lógica de desenvolvimento do projecto de um sistema ou bloco digital.

O compilador, módulo mais importante de uma ferramenta que efectua a síntese lógica (ver capítulo seguinte) do sistema a implementar numa PLD, suscita muitas vezes dúvidas quanto às suas capacidades efectivas (Schulze, 91).

Por exemplo, a rotina de *partitioning* (capacidade de divisão de um projecto por vários PLDs) é geralmente objecto de especulação por parte das empresas que comercializam ferramentas de desenvolvimento. Uma análise cuidada às capacidades reais desta rotina, revela resultados substancialmente diferentes. Em alguns casos estas capacidades consistem em utilitários rudimentares ou documentação de auxílio na divisão manual do projecto em diversos dispositivos. Noutros casos, a rotina efectua automaticamente a divisão lógica completa em arquitecturas e/ou dispositivos múltiplos.

O ponto principal a reter desta definição ambígua de *partitioning*, é que o esforço pedido ao utilizador na realização eficiente do seu projecto, utilizando esta vantagem de divisão múltipla, pode variar do esforço manual até uma automatização quase completa do processo.

Para aumentar ainda mais a confusão, no caso de um PLD do tipo célula lógica, vulgo FPGA, a designação de *partitioning* encerra um significado completamente diferente. A arquitectura geral de um FPGA, consiste num plano de blocos lógicos programáveis conectados por recursos configuráveis. O projecto implementado num FPGA deve pois ser dividido entre esses blocos lógicos produzindo uma implementação o mais eficiente possível da função lógica.

Como se viu, a selecção de uma determinada ferramenta de desenvolvimento de PLDs deve ser encarada pelo utilizador como um investimento que necessita de uma cuidadosa ponderação (Maliniak, 90). Observemos alguns dos pontos importantes a considerar:

- Facilidade de utilização

Um ponto extremamente importante a considerar. A razão reside na utilização por períodos curtos, relativamente ao ciclo global de desenvolvimento de um produto, e muitas vezes sazonal deste tipo de ferramentas. Assim sendo, é necessário uma ferramenta de fácil aprendizagem e reaprendizagem.

As vantagens mais importantes a considerar serão uma interface simples com o utilizador e a presença de ajuda *on-line*.

- Suporte de simulação

A definição de projectos cada vez mais complexos, e a imposição de restrições temporais cada vez mais apertadas, torna a simulação um factor de primordial importância no sucesso do desenvolvimento do PLD em particular e do produto final em geral.

Uma simulação eficiente, com a geração de vectores de teste que cubram as situações mais delicadas do funcionamento do sistema, diminuirá as possibilidades de falha num estágio mais avançado do ciclo de desenvolvimento, em que a rectificação se torna mais difícil e onerosa.

A possibilidade de interface com ferramentas de âmbito geral, poderá ser importante para o desenvolvimento de produtos que necessitem, para a simulação global do sistema, do modelo funcional da PLD desenvolvida.

• Desempenho

Uma boa capacidade de optimização do projecto e o tempo gasto na geração de uma solução óptima mínima, são dois factores chaves na performance global de uma ferramenta de desenvolvimento.

O primeiro passo da optimização consiste na acção dos algorítmos de redução lógica que se encarregam de eliminar os elementos redundantes preservando a funcionalidade do circuito. O único ponto a realçar, uma vez que esta capacidade está presente em todas as ferramentas, será talvez a possibilidade de não reduzir determinados "nós" especificados do circuito, oferecendo ao utilizador um controlo acrescido sobre o processo de optimização. O segundo, e mais importante passo da optimização, consiste na implementação da função minimizada na arquitectura do dispositivo seleccionado. Em arquitecturas do tipo PLD significa o uso automático do teorema de DeMorgan (McCluskey, 86) para gerar, em cada instante a equação mínima para cada função.

A segunda e última consideração incide sobre o módulo de *fitting*. Quanto mais complexa a arquitectura maior deverá ser a capacidade da ferramenta de implementar, num espaço reduzido de tempo, uma solução óptima mínima no dispositivo seleccionado.

• Especificação do sistema

A disponibilidade de diversos métodos de descrição da função lógica: equações Booleanas, captura esquemática, máquinas de estado, tabelas de verdade, formas de onda e linguagens de descrição de *hardware* (com saliência para o VHDL (ver capítulo VHDL)); e a capacidade acrescida de as misturar numa mesma implementação, resultará numa descrição mais eficiente e compacta do bloco lógico.

• Suporte

Trata-se de um dos pontos menos considerado, e que muitas vezes se torna imprescindível na concretização bem sucedida da implementação de um determinado sistema.

Por um lado é necessário garantir o suporte efectivo por parte do vendedor, da ferramenta de desenvolvimento adquirida. No panorama actual de desenvolvimento tecnológico, um bom contrato de manutenção garantirá ao utilizador a evolução do sistema, a actualização das bibliotecas, a resolução de dificuldades pontuais de utilização da ferramenta e o acesso a cursos de formação.

Pelo outro lado, o suporte de dispositivos não pode ser descurado. A escolha de uma ferramenta universal é por isso, muitas vezes, um pau de dois bicos. Se por um lado permite o desenvolvimento de projectos baseados numa biblioteca completa de várias famílias de vários fabricantes, por outro lado obriga à existência de um suporte múltiplo que torne a utilização da ferramenta efectiva.

A opção por uma ferramenta dedicada poderá pois apresentar vantagens neste aspecto. É no entanto necessário seleccionar uma ferramenta suporte de uma família com capacidades de integração bastante variadas e ao mesmo tempo de um fabricante de créditos firmados que garanta uma suporte continuado, e uma actualização constante dos seus PLDs.

3.2.1. Panorama actual

Sem pretensões de ser exaustivo, apresenta-se um quadro (quadro 3.2-1) o mais completo possível das ferramentas de desenvolvimento de PLDs, universais e dedicadas, que se encontram presentemente disponíveis no mercado extremamente concorrido e volúvel da lógica programável.

Quadro 3.2-1

Company	Product	System	Design Entry	Output	Design Features	Type
Accel	Tango-PLD	D	L,S,B,SM,TT	J	LO,FS	NDS
Actel	Action Logic System	D	L,S,B,SM	N	LO,FS	DS
	Actel Logic Minimizer	D	L,S,B,SM	N	LO	DS
AMD	Palasm 4	D	S,B,SM	J	LO,FS	DS
Altera	Max+Plus II	D,U	L,S,W,B,SM,TT	N,J,O	LO,FS,MDP,ADS	DS
Atmel	Atmel-Abel 4	D	L,S,B,SM,TT	J	LO,FS,ADS	DS
AT&T	FPGA Design Tools	D,U	L,S,B,SM	N,O	LO,FS,MDP	DS
Cadence	Programmable Logic Design System (PLDS)	U	L,S,B,SM	N,J,O	LO,FS,MDP,ADS	NDS
Capilano	MacAbel	M	L,B,SM,TT,O	J,O	LO,FS,ADS	NDS

Company	Product	System	Design Entry	Output	Design Features	Type
Cypress	Warp1	D	L,B,SM,TT	J	LO,FS	DS
	PLD ToolKit	D	L	J	FS	DS
	Max+Plus	D	L,S,B,SM,TT	O	LO,FS,ADS	DS
	Max+Plus II	D	L,S,W,B,SM,TT,O	N,O	LO,FS,MDP,ADS	DS
Data I/O	Abel	D,M,U	L,S,B,SM,TT,O	N,J,O	LO,FS,MDP,ADS	NDS
Dazix	Abel FPGA	U	L,S,B,SM,TT,O	N,J	LO,FS,MDP,ADS	NDS
Exemplar Logic	Synthesis System	D,U	L,S,W,B,SM,TT	N,O	LO	NDS
Plessey	ERA Design Kit	D	S	N	FS	DS
Gould AMI	Place	D	B,SM,O	J	LO,FS	DS
	APEEL	D	B	J	LO,FS	DS
Intel	PLDshell	D	B,SM,TT	J	LO,FS	DS
ISDATA	LOG/ic	D,U	L,S,B,SM,TT	J,O	LO,FS,MDP,ADS	NDS
Lattice	pLSI Development System	D	B	J	LO,FS	DS
	pLSI Development System Plus	D,U	S,B	N,J	LO,FS	DS
Logical Devices	CupL	D,M,U	L,S,B,SM,TT	N,J,O	LO,FS,MDP,ADS	NDS
	CupL Integrated Synthesis Tools	D,U	L,S,W,B,SM,TT	N,J,O	LO,FS,MDP,ADS	NDS
Menthor	PLDSynthesis	U	L,S,B,SM,TT	N,J,O	LO,FS,MDP,ADS	NDS
	AutoLogic FPGA	U	L,S,B,SM,TT,O	N,O	LO,FS	NDS
MINC	PLDesigner	D,U	L,S,W,B,SM,TT,O	N,J,O	LO,FS,MDP,ADS	NDS
	PGADesigner	D,U	L,S,W,B,SM,TT,O	N,J,O	LO,FS,MDP,ADS	NDS
National	Opal	D,U	B,SM,TT	J	LO,FS	DS
	Opal Jr.	D	B	J		DS
Omation	Schema PLD	D	S,B,SM,TT	J	LO	NDS
OrCAD	Programmable Logic Design Tools	D,U	L,S,W,B,SM,TT	N,J,O	LO,FS	NDS
Plus Logic	Plustran	D	S,B	J,O	LO	DS
ProLogic	ProLogic V3.0	D	L,B,SM,TT	J	LO,FS	NDS
QuickLogic	pASIC ToolKit	D	S,B	O	LO,FS	DS

Company	Product	System	Design Entry	Output	Design Features	Type
Signetics	Snap	D	L,S,W,B,SM,TT	N,J	LO,FS	DS
	Slice	D	B,SM	N,J	LO	DS
Teradyne	MultiSim Interactive Design for Programmable Logic	U	L,S,W,B,SM,TT	N,J	LO,FS,MDP,ADS	NDS
Texas	Logic Enhancer/ Synthesizer Tool	D	B,SM,TT	N	LO	DS
	Action Logic System	D	S	N		DS
	ProLogic v2.0	D	L,B,SM,TT	J	LO,FS	DS
Valid	SystemPLD/ SystemPGA	U	L,S,W,B,SM,TT,O	N,J,O	LO,FS,MDP,ADS	NDS
ViewLogic	View PLD	D,U	L,O	N,O	LO,FS,MDP,ADS	NDS
	Retargeter	D,U	L,S,O	N	LO,FS	NDS
Xilinx	XACT Dev. System	D,U	L,S,W,B,SM,TT	N,O	LO,FS,ADS	DS

D = DOS; M = Macintosh; U = Unix
L = Language; S = Schematic; W = Waveform; B = Boolean; SM = State Machine; TT = Truth Table; O = Other
N = Netlist; J = Jedec; O = Other;
LO = Logic optimization; FS = Functional simulation; MDP = Multiple device partioning; ADS = Automatic device sel.
DS = Device specific (specific to the company) ; NDS = Non device specific (universal tools)

3.3. SÍNTESE LÓGICA

Síntese lógica é de acordo com o "Technology Research Group" a nova fronteira da EDA (Electronic Design Automation). Ela modificará e simplificará o modo de descrição de sistemas, acompanhando a par e passo a sua, cada vez maior, complexidade.

Novas ferramentas com capacidade de síntese lógica acrescida, permitirão resolver os problemas crescentes de definição e execução de novos projectos sem um incremento significativo do tempo de execução do processo.

3.3.1. Os Domínios da Representação

É necessário compreender as diversas formas existentes de descrição de um sistema, antes de definir o conceito de síntese lógica. Podemos ver na figura 3.3-1 a estrutura base de um mapa, normalmente referido por mapa "Y" e originalmente desenvolvido por D.

Gajski e R. Kuhn, que sintetiza os três domínios de representação de um sistema: comportamento, estrutura e físico (Markowitz, 89a).

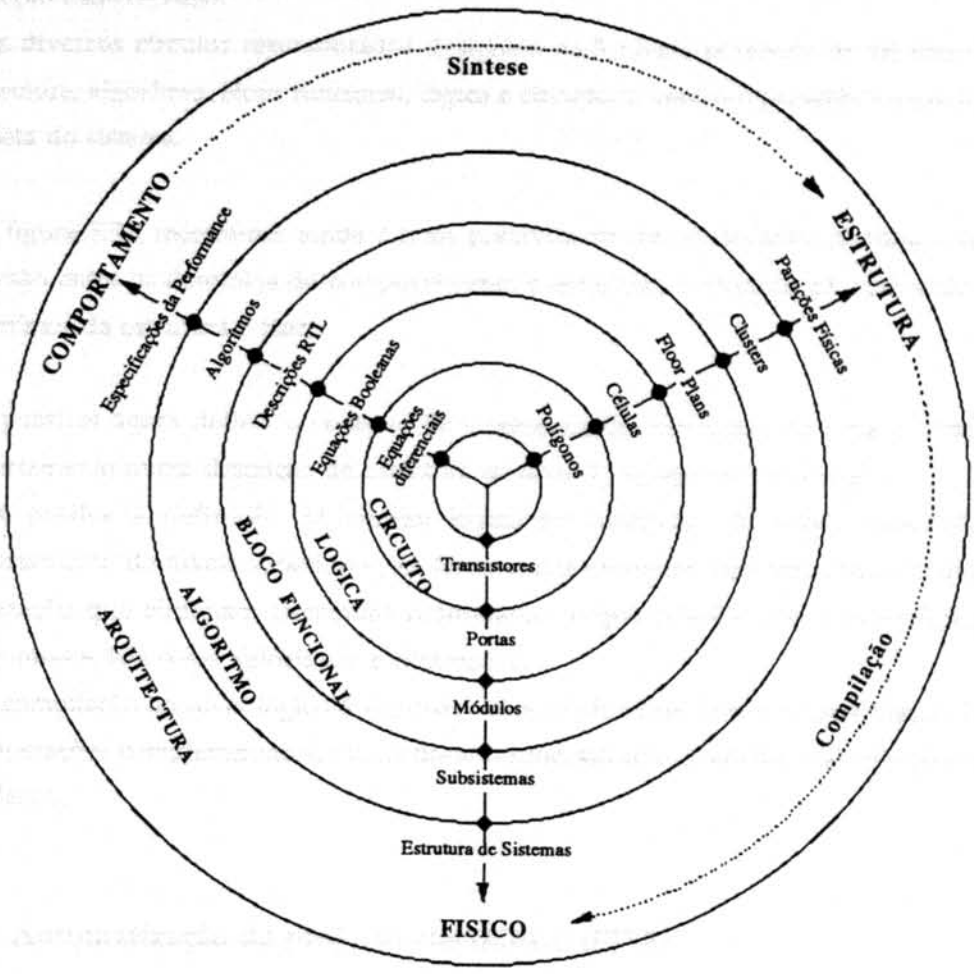


Fig. 3.3-1 Os domínios da representação.

Este modelo pode ser aplicado tanto a um sistema no seu todo, como aos seus elementos constituintes, sejam eles ASICs, PLDs ou circuitos integrados standard.

O domínio do comportamento engloba a funcionalidade do sistema sem qualquer referência quer à estrutura quer aos elementos constituintes. As descrições de comportamento definem apenas as saídas em função das entradas.

O domínio da estrutura traduz a funcionalidade do sistema num conjunto abstracto de estruturas. As descrições estruturais podem ser expressas hierarquicamente, usando diagramas de blocos nos níveis superiores e circuitos integrados com pinos interligados nos níveis inferiores.

O terceiro domínio, o domínio físico, define a representação geométrica do sistema. Especifica o *layout* dos diversos elementos e as suas conexões físicas.

Dentro de cada domínio, um sistema pode ser descrito com maior ou menor grau de especificação. Uma distância crescente do centro do modelo representa um grau de abstracção mais elevado.

Os diversos círculos representados designam os 5 níveis possíveis de representação: arquitectura, algorítmo, bloco funcional, lógico e circuito. O centro representa a especificação completa do sistema.

A figura 3.3-1 mostra-nos ainda 2 tipos possíveis de conversão entre domínios: síntese, conversão entre os domínios do comportamento e estrutura, e compilação, conversão entre os domínios da estrutura e físico.

É possível agora definir o conceito de síntese lógica: conversão de uma descrição de comportamento numa descrição de estrutura ao nível 2, ou seja no nível lógico.

Na prática a definição de síntese lógica foi alargada admitindo descrições de comportamento de níveis superiores. As ferramentas existentes incluem ainda módulos de optimização que eliminam elementos redundantes e que possibilitam a especificação de compromissos tais como velocidade e dimensões.

A compilação ao nível lógico é algumas vezes confundida com a síntese lógica. São de facto operações complementares, iniciando-se o processo com a síntese e terminando com a compilação.

3.3.2. Automatização do projecto electrónico (EDA)

A crescente complexidade dos sistemas digitais, e a necessidade de colocação, mais cedo, dos novos produtos no mercado, tem imposto um automatização cada vez maior de todo o processo de desenvolvimento.

As ferramentas iniciais de CAM (Computer Aided Manufactoring), CAD (Computer Aided Design) e CAE (Computer Aided Engeneering) são hoje em dia substituídas por ferramentas mais latas e flexíveis que englobam todo o processo de desenvolvimento de novos sistemas ou circuitos. É nestas novas ferramentas que o módulo de síntese lógica desempenha um papel de importância crescente (Maliniak, 91c.).

3.3.3. Porquê descrição de comportamento?

Se todos os sistemas fossem descritos no domínio da estrutura, a síntese lógica não seria

necessária.

A razão primeira da sua existência reside na estruturação do próprio pensamento humano. A definição de um sistema, por um engenheiro, começa na definição da função e não da estrutura. A outra razão baseia-se no próprio sistema. Se é verdade que existem circuitos cujo processo de conversão é elementar, noutros o processo é bastante complicado.

Na descrição de uma placa de memória a estrutura resulta naturalmente da função. Considere-se por oposição um simples somador. A sua representação de comportamento será basicamente: SOMA = A+B. Adicionando informação temporal obtemos uma descrição completa do somador. Uma descrição estrutural incluiria uma porta EXOR, uma ou duas portas AND por bit, e as respectivas conexões. Neste caso, trabalhar no domínio da estrutura representaria uma perda de tempo, complementada por um aumento desnecessário da complexidade da descrição da função e/ou sistema.

3.3.4. Vantagens Específicas da síntese lógica

Produtividade acrescida

Os benefícios de produtividade são semelhantes aos que obtêm os utilizadores de linguagens de programação de alto nível. Os seus programas são convertidos automaticamente, compilados, em linguagem-máquina.

Os factores fundamentais de produtividade acrescida são: capacidade de descrever em alto nível, simplicidade inerente dessas descrições.

Descrições qualitativas

Os resultados obtidos com sintetizadores lógicos automáticos superam muitas vezes os obtidos por métodos convencionais. A automatização do processo reduz o esforço e o tempo gasto a efectuar a conversão, deixando ao utilizador tempo adicional para jogar com os parâmetros fundamentais da sua descrição de forma a obter a melhor solução.

Redução dos custos

As funções de optimização incluídas na maioria das ferramentas de síntese lógica permitem produzir melhor, com menores custos, sejam eles económicos ou de superfície.

Redução do número de erros

Todos os sistemas desenvolvidos com a ajuda de ferramentas EDA estão correctos por definição. Os vários módulos por que passa o sistema em desenvolvimento assinalam as possíveis incorrecções, tais como conexões impossíveis, parâmetros mal atribuídos ou simplesmente incompatibilidades na denominação de ligações, evitando que o erro se propague, com custos de correcção acrescidos, no processo de desenvolvimento.

Facilidade de efectuar revisões

As alterações no domínio do comportamento são mais fáceis de efectuar. Quaisquer alterações de funcionalidade ou de parâmetros do sistema são automaticamente processadas.

Independência da tecnologia

Todo o processo pode ser desenvolvido, sem definição da tecnologia a utilizar. As vantagens são evidentes: possibilidade de teste à posteriori das diversas alternativas tecnológicas disponíveis; possibilidade de utilizar mais de uma tecnologia optimizando o modelo por blocos; alteração posterior de uma versão inicial para, por exemplo, uma versão com menores custos (PLD -> *Gate Array*).

3.3.5. Origens da síntese lógica

As primeiras ferramentas de síntese lógica surgiram no início dos anos 50. Tratavam-se de ferramentas básicas de minimização de recursos por eliminação de elementos redundantes; ou de sintetizadores de equações Booleanas. Existiam apenas em organizações de pesquisa e desenvolvimento caso de Universidades ou Laboratórios.

A primeira ferramenta comercial, PALASM, aceitava apenas descrições no nível lógico, e efectuava somente a síntese de PALs. Surgiram posteriormente outras ferramentas, com destaque para o ABEL (DATA I/O, 89), produto da DATA I/O, aceitando descrições de comportamento a um nível superior tais como, diagramas de estados e tabelas de verdade. Estas ferramentas serviam apenas para descrição de PLDs da primeira geração, e ainda não efectuavam os tipos de optimização existentes nas ferramentas actuais.

Foi, o desenvolvimento numa Universidade Americana de dois algoritmos de redução lógica, PRESTO e EXPRESSO, no início dos anos 80, que tornou mais amplo o conceito de

síntese lógica. Ambos os algoritmos são de facto ferramentas de optimização, que trabalhando em conjunto com o módulo de síntese lógica, permitem a minimização da lógica necessária à implementação de determinada função.

As ferramentas actuais, aceitam descrições simultâneas de comportamento e estrutura, oferecendo capacidades de optimização acrescidas, caso da partição automática, podendo ser utilizadas não só em PLDs mas também noutras tecnologias de fabrico de circuitos integrados.

3.3.6. Futuro da síntese lógica

Tal como no passado se assistiu a uma evolução da síntese ao nível lógico, desde as ferramentas primitivas de capacidades limitadas a produtos muito mais poderosos e flexíveis, poder-se-à esperar nesta década, o emergir de ferramentas muito mais sofisticadas que as existentes (Quinnell, 91).

O primeiro passo, já anunciado pelos diversos fabricantes, estenderá o suporte a um leque cada vez mais vasto de tecnologias de fabrico de circuitos integrados. Paralelamente as descrições serão efectuadas em níveis de abstracção superiores (Paradise, 91). Se hoje em dia aceitam descrições no nível lógico, no futuro aceitarão descrições de algoritmos e quem sabe talvez de arquitecturas. Dos métodos a utilizar para este novo conceito de especificação de sistemas, destacam-se as linguagens de descrição textual de "hardware" com realce para o VHDL.

3.4. VHDL

O objectivo principal do governo Americano ao lançar, em 1980, o programa VHSIC (Very High Speed Integrated Circuit), foi de incrementar a evolução nas áreas de desenho e processamento de sistemas e das tecnologias de fabrico (Lipset, et al. 90).

O patrocínio do esforço de desenvolvimento do VHDL (VHSIC Hardware Description Language) como parte do programa global seria uma forma de garantir os meios, para a inserção mais rápida dos novos componentes micro electrónicos em sistemas operacionais e para o desenvolvimento da electrónica comercial com menores custos, tornando mais eficiente a comunicação dentro de, e entre as empresas.

O arranque do esforço VHDL deu-se formalmente em Junho de 1981 num *Workshop* em Woods Hole, Massachusetts.

Em Julho de 1983, a equipa de desenvolvimento da Intermetrics, IBM e Texas Instruments ganhou o contrato de desenvolvimento da nova linguagem e da

implementação do software suporte necessário à utilização da mesma.

A versão final (revista) da mesma, conhecida como VHDL versão 7.2, ficou disponível em Fevereiro de 1987.

Finalmente, a linguagem VHDL foi standartizada pelo IEEE resultando no que hoje é o VHDL IEEE-1076, aprovada em Dezembro de 1987 (IEEE Std 1076, 1987).

3.4.1. Porquê o VHDL?

O VHDL foi desenvolvido para solucionar os problemas existentes no desenvolvimento, na troca de informação e na documentação do hardware digital.

Apesar de existirem diversas linguagens de descrição de hardware, não existia antes do VHDL, nenhum standard aceite na indústria.

A grande maioria das linguagens existentes foram desenvolvidas para servirem os simuladores que as executam, sendo quase sempre propriedade de empresas privadas.

NOTA: A linguagem Verilog HDL (Cadence), principal concorrente do VHDL, foi originalmente desenhada em 1983/84 como um produto de verificação e simulação. Desde essa altura, no entanto, várias ferramentas de análise foram desenvolvidas com base na linguagem, incluindo um simulador de faltas e um analisador temporal. Actualmente o Verilog é uma linguagem aberta a qualquer ferramenta de escrita e leitura (Thomas, et al. 91).

Outras linguagens são orientadas para tecnologias específicas, outras ainda permitem somente a descrição de um projecto num nível baixo (primitiva).

A linguagem VHDL é independente da tecnologia, não está confinada a um simulador em particular, e não obriga à adopção de nenhuma metodologia específica de desenho.

Ninguém pode prever as alterações que ocorrerão no desenvolvimento de hardware digital no futuro. Sendo assim o VHDL oferece uma capacidade de abstracção que facilita a utilização de novas tecnologias em novos circuitos, e na revisão de circuitos já existentes.

3.4.2. Principais vantagens do VHDL

Disponibilidade pública

Sem o estatuto de standard público muitas das suas vantagens não existiriam. A pressão exercida pelo Departamento Americano de Defesa, impondo para todos os contratos

efectuados após 30 de Setembro de 1988, documentação, descrições de comportamento e de estrutura em VHDL, impulsionou ainda mais a utilização deste novo standard.

Metodologia de desenho e suporte de diversas tecnologias de desenho

O VHDL foi desenvolvido para suportar diversas metodologias de desenho (ex: desenho hierárquico *vs* desenho baseado em bibliotecas) e tecnologias de desenho (ex: síncrono *vs* assíncrono, PLA *vs* lógica aleatória). Deste modo a linguagem poderá ser utilizada por organizações que operam de diferentes maneiras e com diferentes especificidades ao nível do projecto.

Independente da tecnologia e do processo

O VHDL permite, no entanto a especificação de tal informação. Na prática, uma descrição funcional (i.e. simulável) de um sistema pode ser desenvolvida a um nível superior ao nível de portas lógicas, e posteriormente decomposto em implementações deste nível dependentes da tecnologia escolhida (ex: CMOS, nMOS, GaAs). As vantagens deste ponto em termos de fontes alternativas (*second sourcing*) são evidentes (Leibson, 89).

Capacidade descritiva estendida

O VHDL suporta a descrição do hardware em termos de comportamento, desde o nível de sistema (i.e. caixa preta) até ao nível detalhado da porta lógica.

Uma das vantagens principais da linguagem reside na capacidade de descrição de um sistema digital, utilizando simultaneamente vários níveis descritivos (é no entanto necessário garantir a coerência de sintaxe e semântica entre eles), e mantendo a capacidade de simulação do sistema no seu todo.

Portabilidade das descrições

VHDL é um standard e como tal os modelos descritos nesta linguagem podem ser executados em qualquer sistema conforme com o standard.

Uma implicação importante desta vantagem reside no facto de equipas distintas poderem independentemente desenvolver descrições de alto nível de subsistemas de um sistema digital, sem afectar a capacidade de simulação global.

3.4.3. Utilização prática

Apesar de o VHDL estar a contribuir, em larga escala, para a transformação da torre de Babel do mercado das ferramentas EDA num catálogo ordenado de produtos compatíveis, muitos problemas existem ainda por resolver antes de a linguagem preencher os requisitos base que levaram à sua criação (Markowitz, 89b).

Por um lado a imaturidade da linguagem. Os problemas mais prementes são o *back annotation*, a criação de uma biblioteca universal de modelos e a resolução de pequenas ambiguidades e pequenas alterações na linguagem que facilitem a modelização de estruturas. Estas alterações estão a ser consideradas na elaboração de uma nova versão da linguagem que se pensa poder estar concluída no fim de 1992.

Por outro lado a formação do engenheiro e/ou projectista de sistemas. Estes estão habituados a utilizar a captura esquemática para a realização de novos circuitos e sistemas, definindo-os com símbolos gráficos representando portas, blocos ou outro tipo de primitivas. E enquanto neste método, erros, como por exemplo conexões não terminadas ou designações incorrectas são detectados por mera observação do esquemático, estes problemas transformam-se em vírgulas e variáveis inconsistentes, para o utilizador do VHDL. É no fundo uma sensação desconfortável de assistir à invasão e conquista, do seu mundo, por um mundo, até então distinto, do programador e das suas linguagens.

3.4.4. Aplicações existentes

Alguns fabricantes de PLDs adoptaram já o VHDL. Outros esperam ainda reacções do mercado para adaptarem as sua ferramentas ao novo standard.

No mercado dos ASICs, como se pode verificar no quadro 3.4-1, várias ferramentas estão já à disposição do utilizador (Markowitz, 89a).

Quadro 3.4-1

Companhia	Produto	PLDs/ASICs	Capacidade
Algorithimic Systems	Ascyn Logic Synthesizer	P,A	1,2,3,4
Altera	Plds-MAX	P	1,2,3,4
Data I/O	Gates	P,A	1,2,3,4
Mentor	PLDSynthesis	P	1,2,3,4

Companhia	Produto	PLDs/ASICs	Capacidade
Minc	PLDesigner	P	1,2,3,4
NCR	Design Synthesis	A	1,2,3,4
Orcad	Orcad/PLD	P	1,2,3,4
Seattle Silicon	Finesse	A	1,2,3
Silc	Silcsyn	A	1,2,3,4
Silicon Compiler Systems	Logic Compiler	A	1,2,3
Synopsis	Design Compiler	P,A	1,2,3,4
VLSI Tech	Logic Synthesizer	P,A	1,2,3

1- Lógica combinatória; 2- Lógica sequencial; 3- Circuitos síncronos; 4- Circuitos assíncronos;

A revolução, lentamente, inicia a sua marcha ...

3.5. ENGENHARIA CONCORRENTE

Engenharia concorrente é uma nova metodologia de desenvolvimento de produtos baseada na interacção de todas as disciplinas de engenharia participantes no processo. A adopção desta nova filosofia, de acordo com um relatório da IDA (Institute for Defense Analysis of EUA), resultará em menores custos de desenvolvimento, ciclos de produção mais curtos e numa maior qualidade final do produto. Não se trata no entanto de uma ferramenta nova que se adquire e utiliza, antes uma nova forma, ou se quisermos um conceito reinventado, de pensar e agir em engenharia.

A engenharia concorrente pode adquirir diferentes significados. Em sentido estrito será a conjunção e/ou sobreposição de tarefas num processo, até aqui sequencial, de desenvolvimento. Num sentido mais lato poderá ser designada de processo paralelo de desenvolvimento com a interacção de áreas tão distintas como o marketing, desenho, teste, produção e distribuição.

De acordo com o esquema apresentado na figura 3.5-1, a grande desvantagem do desenvolvimento sequencial é o fluxo de informação num único sentido. Em engenharia concorrente todos participam com os seus conhecimentos e experiência, num processo de desenvolvimento partilhado, diagnosticando os problemas e prevenindo as situações na fase inicial dos projectos. Como diz Hiroto Kagami, gestor de qualidade da Canon: "*If pure*

water flows from the upper stream, then there is no need to purify it farther downstream" (Markowitz, 91).

Estas equipas pluridisciplinares podem incluir ainda representantes exteriores à organização. A consulta a fornecedores e clientes poderá traduzir-se em benefícios importantes. Os fornecedores poderão aconselhar a utilização de novos produtos das suas representadas que trarão valor acrescentado ao processo em desenvolvimento. O auscultar da opinião dos consumidores resultará na especificação de novos produtos com aceitação garantida no mercado (Decastro, et al. 91).

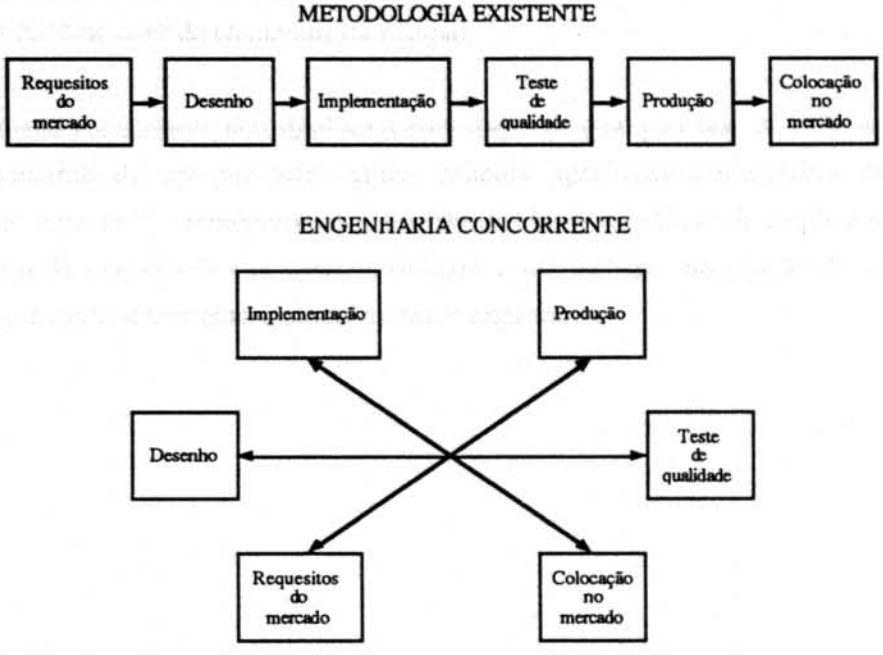


Fig. 3.5-1 Desenvolvimento sequencial *vs* engenharia concorrente

Nem tudo serão rosas, após a adopção da nova filosofia. Os conflitos entre equipas pluridisciplinares serão inevitáveis. Será necessário assumir uma nova posição de liderança, um pouco à semelhança de um misto de treinador e árbitro (Maliniak, 91d). O IDA, no relatório (R-338) acima referido, recomenda uma adopção gradual desta nova metodologia tendo em atenção as seguintes recomendações:

1- Implantação *top-down*. A iniciativa tem de começar de cima.
2- Empenho dos gestores. O empenho deverá incidir nas fases de aprendizagem, compreensão e liderança.
3- Projectos piloto que acelerem a absorção da nova filosofia.
4- Análise de experiências bem sucedidas.

5- Educação e treino.

6- Desenvolvimento e adaptação do método.

7- Identificação e redução de barreiras e inibições.

A utilização de ferramentas de engenharia concorrente será fundamental na implementação com sucesso da metodologia. Elas facilitam a comunicação entre disciplinas. Para além de oferecerem canais de comunicação, elas permitem a partilha de bibliotecas, interfaces e bases de dados. Haverá no entanto, necessidade de estabelecer limites para a informação a partilhar, evitando uma sobrecarga desnecessária. Será necessário reduzir o número de canais de comunicação a um número aceitável. Este, cresce de acordo com a função $(n^2-n)/2$ (n -> nº de elementos da equipa).

Engenharia concorrente não significa a eliminação de qualquer fase do ciclo existente de desenvolvimento de um produto. Antes, procura optimizar a sequência de funções individuais num todo harmonioso. As experiências bem sucedidas de implementação da metodologia da engenharia concorrente, provam a sua validade na criação de uma forma diferente, e significativamente melhor, de fazer engenharia.

4. SISTEMA DE TRANSMISSÃO DE AUDIO DIGITAL (SITAD)

Neste capítulo são apresentados o multiplexador e desmultiplexador do sistema de transmissão de audio digital, destinado à transmissão de programas de som de alta qualidade.

O desenvolvimento do sistema de transmissão de audio de alta fidelidade (SITAD) (Leitão, et al. 89) esteve ligado, desde o seu início, ao projecto SIFO (Sistemas Integrados por Fibra Óptica), um protótipo laboratorial de uma rede digital síncrona de banda larga, com integração de serviços (Alves, 89). O SITAD era não só um desses serviços, quando transmitido isoladamente, mas também parte de um outro serviço quando associado a um ou mais canais de video. Neste contexto, foram desenvolvidas as duas primeiras versões do sistema.

As performances obtidas com a segunda versão, a capacidade de integração alcançada com um PLD de média densidade, o EP1810J da Texas Instruments, e a indisponibilidade no mercado de um dos circuitos integrados dedicados utilizados, aliadas à aquisição de uma nova ferramenta de desenvolvimento de PLDs, especificamente da família 5000 da Altera, motivou o desenvolvimento da versão 3 do referido sistema de transmissão, com os objectivos propostos de integração, desempenho, consumo, fiabilidade e redução dos custos de fabrico a serem completamente alcançados.

Em resumo, a evolução sofrida pelo SITAD ao longo de três versões distintas, reflecte a evolução tecnológica que se tem vindo a sentir no mundo dos dispositivos lógicos programáveis.

O sistema cumpre as normas do CCITT existentes para codificação/descodificação, e transmissão a 2048 KBit/s (CCITT, 1985).

4.1. NORMAS CCITT

4.1.1. Interface a 2048 KBit/s

As características gerais da interface de transmissão a 2048 KBit/s, descrita na recomendação G.703, são as seguintes (Rec G.703, 1985):

Taxa de transmissão: 2048 KBit/s ± 50 ppm
Código: HDB3

Especificações do sinal de saída

Meio de transmissão: Um par coaxial
Impedância de carga de teste: 75 ohms resistiva
Tensão de pico de um impulso: 2.37 V
Tensão de pico de um espaço: 0 ± 0.237 V
Largura nominal do impulso: 244 ns

Especificações do sinal de entrada

O sinal digital presente na entrada do equipamento será definido como o sinal de saída, mas modificado pela característica do meio de transmissão. A atenuação do meio, neste caso um cabo coaxial, assumir-se-à que segue a lei $\sqrt{f}$ e que para uma frequência de 1024 KHz a perda variará entre os 0 e 6 dB.

4.1.2. Código HDB3

Tal como é definido no anexo A da norma G.703 do CCITT, o código HDB3 (High Density Bipolar 3) caracteriza-se por impedir a transmissão de sequências de zeros superiores a três.
A conversão de um sinal NRZ em HDB3 é efectuada de acordo com as seguintes regras:

1) O sinal HDB3 é um sinal pseudo-ternário; os três estados são designados B+, B- e 0.

2) Os zeros do sinal NRZ são codificados como zeros no sinal HDB3. Para sequências de quatro zeros, existem regras de codificações particulares (ver 4).

3) As marcas do sinal NRZ são codificadas alternadamente como B+ e B- no sinal HDB3 (B+ e B- são sinais do tipo AMI). A violação à regra da alternância é introduzida quando se codificam sequências de quatro zeros (ver 4).

4) As sequências de quatro zeros são codificadas de acordo com as seguintes regras:

   a) O primeiro zero da sequência é codificado como um zero se a marca precedente do sinal HDB3 tiver uma polaridade oposta à polaridade da violação precedente e não for ela própria uma violação; é codificado como uma marca, isto é uma não violação (B+ ou B-), se a marca precedente do sinal HDB3 tiver a mesma polaridade da violação precedente ou for ela própria uma violação.

Esta regra garante a alternância de polaridade de violações sucessivas, eliminando uma possível componente contínua do sinal.

b) O segundo e terceiro zeros da sequência são sempre codificados como zeros.

c) O último zero da sequência de quatro é sempre codificado como uma marca, de forma que a sua polaridade viole a regra da alternância de marcas. Estas violações são designadas por V+ ou V- de acordo com a sua polaridade.

Exemplifica-se na figura 4.1-1, a de codificação de um sinal NRZ:

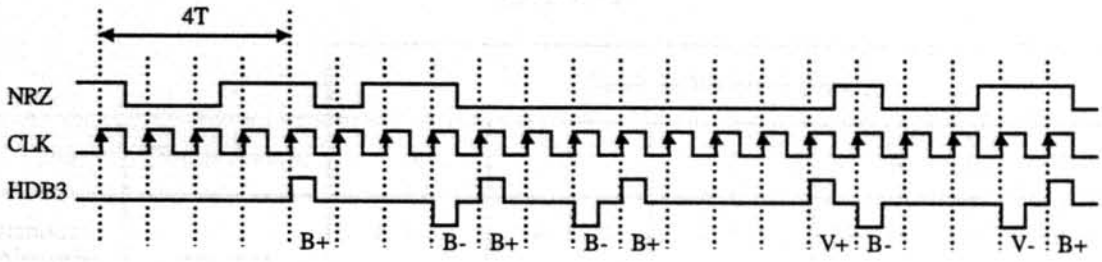


Fig. 4.1-1 Codificação HDB3 de um sinal NRZ.

4.1.3. Estrutura básica da trama a 2048 KBit/s

A recomendação G.704 do CCITT descreve a estrutura básica da trama de 2048 KBit/s (Rec G.704, 1985).

Tabela 4.1-1

	1	2	3	4	5	6	7	8
Trama com SAT	Si Nota 1	0	0	1	1	0	1	1
		Sinal de alinhamento de trama						
Trama sem SAT	Si Nota 1	1 Nota 2	A Nota 3	Sn	Sn	Sn	Sn	Sn
				Nota 4				

Nota 1 — Si — Bits reservados para uso internacional (ver norma G.704 do CCITT). No caso de não serem utilizados, devem ser fixados no nível lógico 1 em linhas digitais internacionais. Se a linha digital não atravessar fronteiras poderão ser utilizados a nível nacional.
Nota 2 — Este bit é fixado no nível lógico 1 para ajudar a evitar simulações do SAT.
Nota 3 — A — Indicação remota de alarme. Em funcionamento normal, nível lógico 0. Sob condição de alarme, nível lógico 1.
Nota 4 — Sn — Bits disponíveis para uso nacional. Quando não utilizados devem estar fixos no nível lógico 1.

A trama tem um comprimento de 256 bits, numerados sequencialmente de 1 a 256. Estes

256 bits são divididos ordenadamente em 32 octetos (64 KBit/s), numerados sequencialmente de 0 a 31. O octeto 0 transporta o SAT (Sinal de Alinhamento de Trama) nas tramas ditas pares, e informação de alinhamento de trama e sinalização nas tramas ditas impares. O octeto 16 poderá transportar informação de sinalização. A alocação dos bits 1 a 8 (octeto 0) é efectuada de acordo com a tabela 4.1-1.

No caso de serem utilizados canais de 384 KBit/s, a norma G.737 do CCITT recomenda a alocação de canais de acordo com a tabela 4.1-2 (Rec G.737, 1985):

Tabela 4.1-2

		Canais de 384 KBit/s (Nota 1)				
País	Tipo de Inserção	A	B	C	D	E
Noruega Alemanha Dinamarca Suiça *Portugal*	síncrona	1 - 2 - 3 17 - 18 - 19	4 - 5 - 6 20 - 21 - 22	7 - 8 - 9 23 - 24 - 25	10 - 11 - 12 26 - 27 - 28	13 - 14 - 15 29 - 30 - 31

Nota 1: Os cinco canais possíveis de 384 KBit/s numa trama de 2048 KBit/s são numerados de A a E. Os pares de canais A-B e C-D devem ser preferentemente usados para transmissão estereofónica.

4.1.4. Sinal de alinhamento de trama (SAT)

A detecção do alinhamento de trama a 2048 KBit/s é regulamentada pelas recomendações G.737, G.738 e G.739 do CCITT (Rec G.737/8/9, 1985).

O alinhamento de trama será considerado perdido quando três ou quatro SATs (Sinal de Alinhamento de Trama) forem recebidos com erro.

Nota 1: de forma a limitar o efeito de imitação de SATs o seguinte procedimento poderá ser usado: o alinhamento de trama será considerado perdido quando o segundo bit do octeto 0, em tramas que não contenham o SAT, for recebido com erro em três ou quatro ocasiões.

O alinhamento de trama será considerado recuperado quando for detectada a seguinte sequência:

- inicialmente, a presença de um SAT correcto;
- a ausência do SAT na trama seguinte, pela confirmação de que o segundo bit do octeto 0 tem o valor lógico 0;
- pela segunda vez, a presença de um SAT correcto na trama seguinte.

Nota 2: para evitar a possibilidade de um estado de não recuperação do alinhamento de trama devido à presença de um SAT imitado o seguinte procedimento poderá ser usado: quando um SAT válido for detectado na trama n, deve ser confirmada a sua ausência na trama n+1 e a sua presença na trama n+2. O não cumprimento de uma ou de ambas as condições deve resultar numa repetição do processo a partir da trama n+2.

O diagrama de estados que representa a sequência de operações de perda e recuperação do alinhamento de trama de acordo com as recomendações é esquematizado na figura 4.1-2.

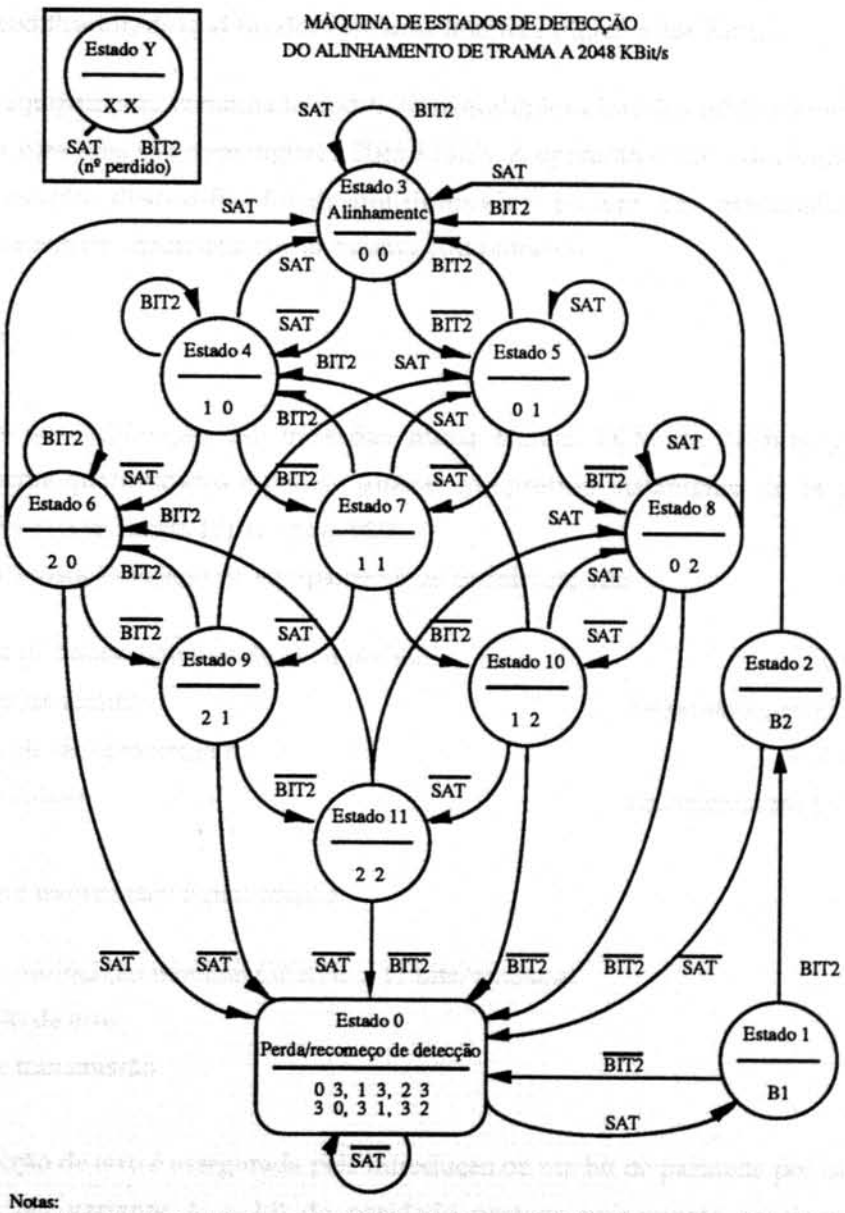


Fig. 4.1-2 Diagrama de estados possível para a detecção do alinhamento de trama.

4.1.5. Codificação de sinais audio para transmissão em canais de 384 KBit/s

A recomendação J.41 fornece as características do equipamento para a codificação de sinais analógicos monofónicos de 15 KHz num sinal digital de 384 KBit/s. Para a transmissão estereofónica devem ser utilizados dois canais monofónicos (Rec J.41, 1985).

O equipamento para a codificação de sinais analógicos tal como é especificado na recomendação pode ser:

a) um codificador/descodificador com uma interface digital a 384 KBit/s.

b) um equipamento combinado codificador-multiplexador/descodificador-desmultiplexador com uma interface digital a 2048 KBit/s. A operação codificador-multiplexador e a operação descodificador-desmultiplexador podem ser efectuadas em dois equipamentos separados ou no mesmo equipamento.

Codificação

As leis de codificação são baseadas numa técnica PCM de 14 bits por amostra uniformemente quantificados e podem utilizar compressão instantânea de 14 para 11 bits segundo a lei A (ver Tabela 1/J.41 - pag. 163).

As características básicas do equipamento de codificação são:

Largura de banda nominal do sinal audio:	0.04 a 15 KHz
Interface de audio:	Recomendação J.21 do CCITT
Frequência de amostragem:	32 (1 ± 5E-5) KHz
Pre/De-ênfase:	Recomendação J.17 do CCITT

A taxa de transmissão é pois igual a:

Taxa de codificação nominal (32 KHz x 11 bits/amostra)	352 KBit/s
Protecção de erro	32 KBit/s
Taxa de transmissão	384 KBit/s

A protecção de erro é assegurada pela introdução de um bit de paridade por cada amostra de 11 bits. Na variante A, o bit de paridade protege unicamente os cinco bits mais significativos de cada amostra. No conversor da parte de transmissão o bit de paridade é adicionado como sendo o bit nº 12 de cada palavra codificada. O seu valor é fixado de forma que o bloco de paridade contenha sempre um número impar de "1s". De forma que

situações de erro pares possam também resultar em violações de paridade, os bits protegidos e não protegidos são entrelaçados numa sequência ascendente e descendente respectivamente, de acordo com a figura 4.1-3.

Nota: uma situação de erro par significa a ocorrência de erros num número par de bits da palavra codificada. Se essa ocorrência se verificasse nos bits protegidos a situação de erro duplo não seria detectada. O facto de os bits protegidos serem entrelaçados com os não protegidos diminui a probabilidade da ocorrência acima citada, uma vez que a probabilidade de erro em dois bits afastados é manifestamente menor que a probabilidade de erro em dois bits consecutivos.

Nota: Variante A — 1 -> bit de sinal; 2, 3, 4 -> bits de segmento; 5 a 11 -> bits de valor; 12 -> bit de paridade

Fig. 4.1-3 Disposição dos diferentes bits de uma amostra codificada, nos octetos correspondentes de uma trama de 2048 KBit/s.

4.1.6. Sinalização

De acordo com a recomendação G.735 do CCITT o equipamento deverá sinalizar determinadas situações que indiciem o seu funcionamento anormal. Destas situações salientam-se as fundamentais (Rec G.735, 1985):

- Falha da fonte de alimentação (multiplexador e desmultiplexador)
- Ausência de sinal dos canais de 384 KBit/s (multiplexador)
- Ausência de sinal de entrada a 2048 KBit/s (desmultiplexador)
- Perda do alinhamento de trama (desmultiplexador)

4.1.7. Controlo do *jitter* na hierarquia dos 2048 KBit/s

A recomendação G.823 do CCITT aponta algumas directrizes a serem seguidas no projecto do equipamento, de forma a controlar o *jitter* em redes digitais baseadas na hierarquia dos 2048 KBit/s, considerando que (Rec G.823, 1985):

a) o *jitter*, definido como uma variação de curta duração dos instantes significativos de um sinal digital em relação à sua posição ideal no tempo, pode ocorrer em redes digitais.

b) se não for exercido um controlo suficiente, então em determinadas circunstâncias, pode verificar-se uma acumulação excessiva de *jitter* ao ponto de poderem ocorrer as seguintes situações:

   i) um aumento da probabilidade de ocorrência de erros em regeneradores de sinais digitais, como resultado do deslocamento da informação temporal destes sinais da sua posição óptima no tempo;

   ii) a introdução de deslizamentos não controlados em sinais digitais ocorridos em determinados tipo de equipamentos terminais que incluam *buffers* ou comparadores de fase e em determinados equipamentos de multiplexagem digital.

   iii) a degradação da codificação digital de informação analógica como resultado da modulação de fase das amostras reconstruídas no conversor digital/analógico no termo da conexão;

c) o *jitter* pode ser reduzido em magnitude utilizando esquemas adequados.

4.2. O SITAD E A EVOLUÇÃO TECNOLÓGICA DOS PLDs

O sistema SITAD permite suportar a transmissão de um máximo de 5 canais monofónicos de alta qualidade, os quais podem ser agrupados constituindo pares estereofónicos. A transmissão é feita sobre canais de alto débito.

Para evitar uma descrição repetitiva, optamos por efectuar uma abordagem geral do sistema, e das normas CCITT que lhe deram origem, particularizando por blocos os aspectos específicos de cada uma das versões.

A arquitectura fundamental do sistema de transmissão de audio digital, encontra-se descrita no diagrama de blocos da figura 4.2-1. De referir que a versão inicial do sistema apenas permitia a codificação/descodificação de 4 canais monofónicos de 384 KBit/s, agrupados de acordo com a recomendação do CCITT em dois pares estéreo, A+B e C+D (Cabral, 90).

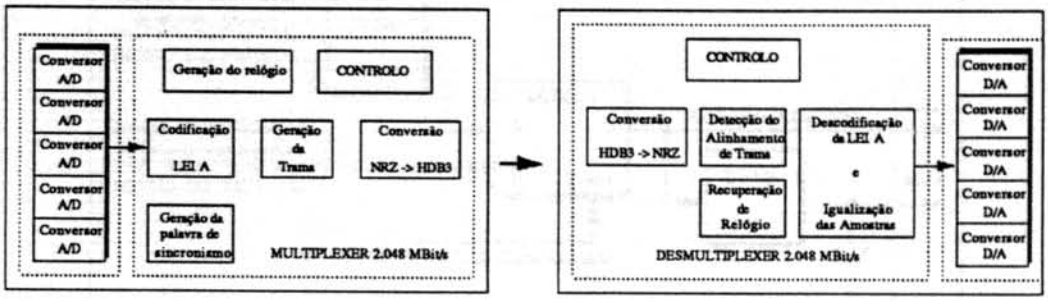


Fig. 4.2-1 Diagrama de blocos do SITAD.

Nas duas versões subsequentes foi prevista a codificação do 5º canal, o canal E, e para além deste facto o multiplexador sofreu uma alteração na sua estrutura básica (ver figura 4.2-2) de forma a possibilitar a multiplexagem dos canais de audio activos numa trama em trânsito pelo sistema (Cardoso, 90).

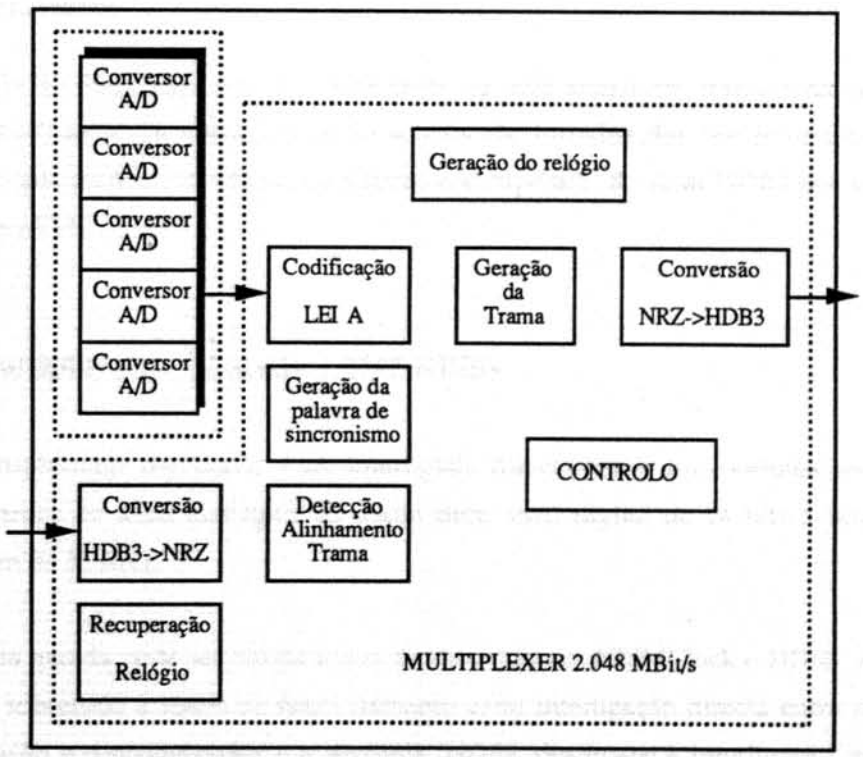


Fig. 4.2-2 Flexibilização do multiplexador nas versões 2 e 3 do SITAD.

Tal flexibilização deu origem à definição, ilustrada na figura 4.2-3, nas duas últimas versões, de dois modos distintos de funcionamento, consoante se encontrasse ou não presente na entrada do multiplexador a referida trama em trânsito. A selecção do modo de funcionamento é efectuada pelo sinal de alarme que detecta a presença, ou ausência, de um sinal HDB3 na entrada do multiplexador.

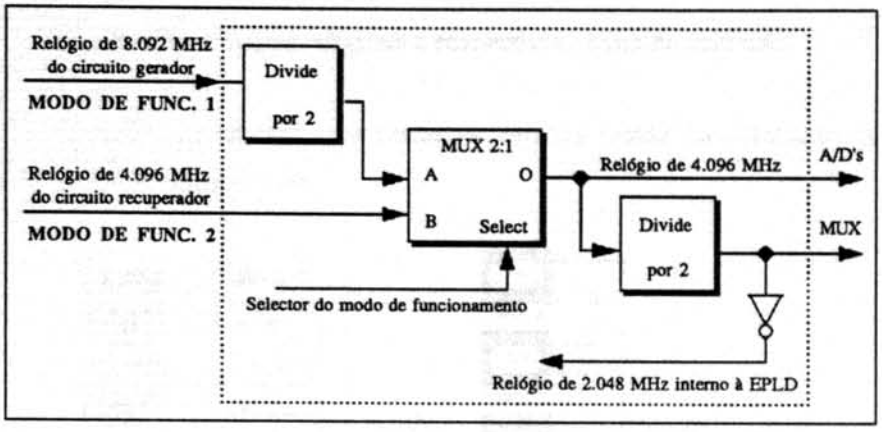


Fig. 4.2-3 Circuito de selecção do modo de funcionamento do multiplexador nas versões 2 e 3.

- Modo de funcionamento 1 -> Trama de 2048 KBit/s e relógio de 2048 KHz gerados internamente

- Modo de funcionamento 2 -> Existência de uma trama em trânsito na entrada do multiplexador. Os canais de audio activos são introduzidos nos intervalo de tempo correspondentes, e o relógio do sistema é recuperado do sinal HDB3 por um circuito do tipo PLL.

4.2.1. Multiplexador de audio a 2048 KBit/s

O multiplexador destina-se a ser interligado directamente aos módulos responsáveis pela conversão do sinal analógico de audio num sinal digital de 14 bits a uma taxa de amostragem de 32 KHz.

A trama gerada pode ser obtida numa de duas formas: NRZ/Clock e HDB3. A primeira destina-se sobretudo a testes de funcionamento e/ou interligação directa entre os sistemas de codificação e descodificação, e a segunda, HDB3, destina-se à interligação de sistemas através de linhas de transmissão nas diversas hierarquias digitais.

Sinais de interface com o sistema

Sinais recebidos pelo Multiplexador

- Dados dos conversores analógico-digitais e respectivos sinais de controlo

A diferença entre a primeira e as restantes versões reside na colocação dos *buffers*, conforme se mostra na figura 4.2-4.

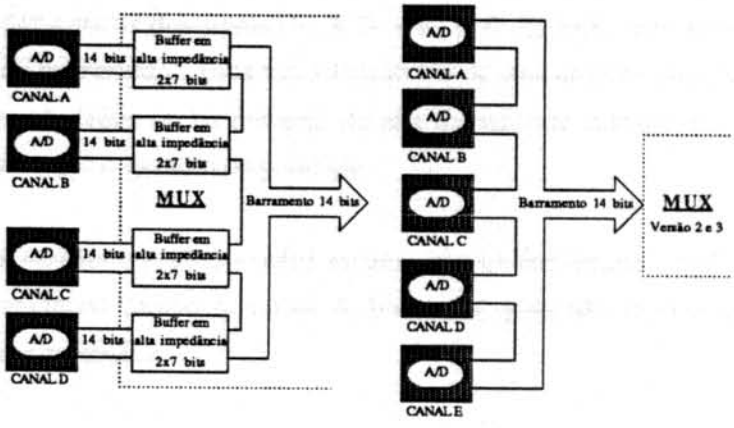


Fig. 4.2-4 Colocação dos *buffers* de armazenamento das amostras codificadas.

Na versão inicial estes estão colocados no multiplexador, um par por cada canal monofónico, ligando-se assim directamente cada conversor A/D a um barramento isolado de 14 bits existente no multiplexador (Inocêncio, 88).

A frequência de amostragem e o instante exacto do inicio da conversão são controlados pelo multiplexador a partir dos sinais conv/AB e conv/CD que actuam simultaneamente no início da conversão dos A/Ds dos canais A+B ou C+D, impondo a cada par de conversores a frequência de amostragem pretendida (32 KHz).

Tal como se pode constatar da figura 4.2-4, todos os sinais de entrada provenientes dos conversores passam por um andar de armazenamento com saída em alta impedância para permitir a sua interligação a um único barramento de 14 bits. Sendo assim os conversores analógico/digitais são directamente acoplados ao multiplexador sem a necessidade de qualquer interface intermédia.

Cada *buffer* é controlado por dois sinais:

- Um sinal de *enable* (EN/X) que activa as saídas dos *buffers* correspondentes a um dos canais, A,B,C ou D, colocando a amostra desse canal disponível para a transmissão.

- Um sinal de armazenamento (CK/X) que faz a captura dos dados dos canais A,B,C ou D, permitindo armazenar a próxima amostra, ou seja libertar o conversor A/D para efectuar a próxima conversão.

Nas versões 2 e 3, e aproveitando a existência de conversores série/paralelo do tipo 74LS595 (TI, 84) nos conversores A/D (o resultado da conversão dos A/Ds nestas versões é apresentado sob a forma série (Ferreira, 90)), utilizaram-se estes circuitos como *buffers*, interligando os 5 conversores num único barramento de alta impedância controlado pelo multiplexador.

A frequência de amostragem e o instante exacto do início da conversão são controlados pelo multiplexador a partir dos sinais CK_X (X = A, B, C, D ou E), que actuam no início da conversão dos A/Ds impondo a cada um a frequência de amostragem pretendida (32 KHz), e EN_X que coloca os dados no barramento de alta impedância disponíveis para o bloco de compressão, codificação e geração de paridade.

Em todas as versões o multiplexador assume que os conversores A/D são capazes de realizar a conversão no tempo que lhes é destinado, pois não realiza qualquer teste à validade dos dados recebidos.

• Trama em trânsito a 2048 KBit/s codificada em HDB3

Caso se encontre presente na entrada do multiplexador uma trama em trânsito, são nela inseridos os canais de audio activos. Os intervalos de tempo correspondentes a canais inactivos mantêm o seu conteúdo original. No caso de não existir trama à entrada do multiplexador, esta é gerada internamente de acordo com as normas.

Sinais gerados pelo multiplexador.

• NRZ

O sinal NRZ presente na saída do multiplexador, será a trama de 2048 KBit/s gerada ou a trama em trânsito na qual foram carregados os canais de audio activos, conforme a versão e/ou modo de funcionamento do sistema. O sinal NRZ está sincronizado com a transição ascendente do relógio a 2048 KHz, possuindo no entanto, como se verifica na figura 4.2-5, um atraso em relação a este devido ao tempo de propagação do bloco final de conversão paralelo/série.

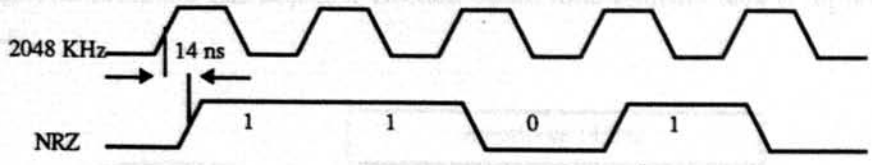


Fig. 4.2-5 Relação entre o sinal NRZ à saída do multiplexador e o relógio do sistema.

• HDB3

O sinal HDB3 é obtido a partir do sinal NRZ após a necessária codificação. Em virtude das características dessa mesma codificação, o sinal de HDB3 possui um atraso temporal em relação ao sinal NRZ de cerca de 4 bits.

Descrição do multiplexador por blocos

Como já foi anteriormente referido, os multiplexadores das versões 2 e 3 apresentam um diagrama de blocos substancialmente diferente da versão inicial. Os blocos adicionais que possuem são no entanto em tudo semelhantes aos blocos do mesmo nome existentes no desmultiplexador das respectivas versões. Assim sendo, evitando uma descrição repetitiva que não resulte contudo, em perda de pormenor na apresentação dos diferentes blocos constituintes do multiplexador, optou-se por fazer a sua referência no capítulo referente ao desmultiplexador. No entanto, e sempre que relevantes para a explicação de funcionamento de um outro bloco, serão oportunamente referidos.

Compressão, codificação e geração de paridade

Tal como é definido pela norma J.41 do CCITT, a transmissão de audio de alta qualidade em canais de 384 KBit/s deve ser afectada de uma compressão de 14 para 11 bits por amostra. Para realizar essa compressão, foi adoptado o método da compressão instantânea (Tabela 1 da norma J.41) na sua variante A (transmissão a 2048 KBit/s). Para além da compressão propriamente dita (14 para 11 bits), existe ainda definida na citada norma uma codificação das posições dos diversos bits da amostra e a adição de um bit de paridade com vista a proteger a integridade dos dados transmitidos. Assim sendo os 5 bits mais significativos (dos 11 obtidos após compressão) são protegidos por um bit de paridade, sendo este bloco de 6 bits intercalado nos outros 6 bits para diminuir a probabilidade de não detecção de um erro duplo na transmissão.

O diagrama completo das acções a efectuar sobre uma amostra será o representado na figura 4.2-6:

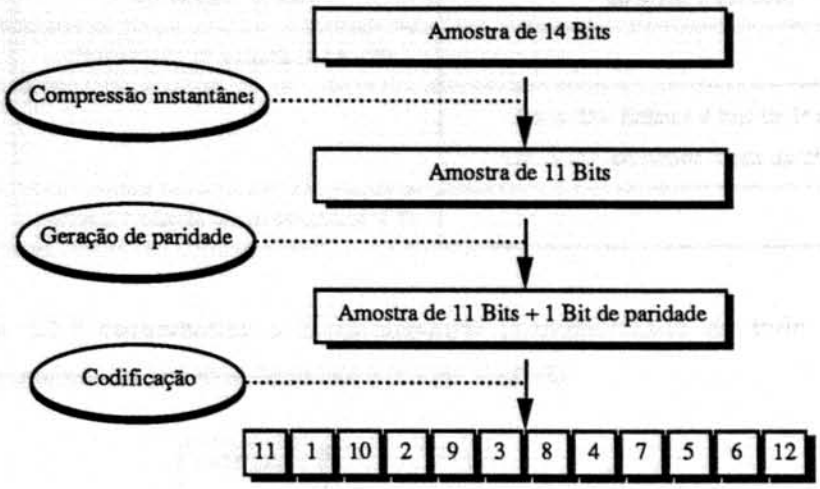


Fig. 4.2-6 Fluxograma das operações necessárias à codificação e compressão das amostras segundo a lei A.

Toda esta multiplicidade de codificações foi implementada numa EPROM, sendo o seu código gerado por uma rotina descrita em Pascal (Anexo).

Realce-se o carácter peculiar da disposição das sucessivas amostras de um canal ao longo da trama. Efectivamente a sucessão de amostras na trama não corresponde à sua sucessão no tempo, isto é, embora por cada trama transmitida sejam enviadas 4 amostras de um determinado canal, estas são enviadas duas a duas o que implica que o seu espaçamento não é constante. O facto de uma EPROM comercial apresentar apenas um barramento de dados de 8 bits, aliado à particularidade acima referida, permite concluir que para transmitir as amostras i e i+1 do canal X a alocação dos octetos será a seguinte:

- octeto n - bits 1, 2, 3, 4, 8, 9, 10 e 11 da amostra i
- octeto n+1 - bits 5, 6, 7 e 12 da amostra i
  - bits 1, 2, 10 e 11 da amostra i+1
- octeto n+2 - bits 3, 4, 5, 6, 7, 8, 9 e 12 da amostra i+1

Versão 1

Para implementar a codificação da lei A são utilizadas nesta versão, não uma, mas sim duas EPROMs. No quadro 4.2-1 sintetizam-se as intervenções de cada EPROM na codificação de duas amostras consecutivas de um mesmo canal.

Quadro 4.2-1

	EPROM 1 (27256)	EPROM 2 (27128)
octeto n	1ºs 8 bits da 1ª amostra (A14 = 0)	—
octeto n+1	—	D0 -> D3: últimos 4 bits da 1ª amostra D4 -> D7: primeiros 4 bits da 2ª amostra
octeto n+2	últimos 8 bits da 2ª amostra (A14 = 1)	—

A figura 4.2-7 esquematiza o funcionamento pormenorizado de todo o bloco de codificação, comandado por três sinais distintos de controlo.

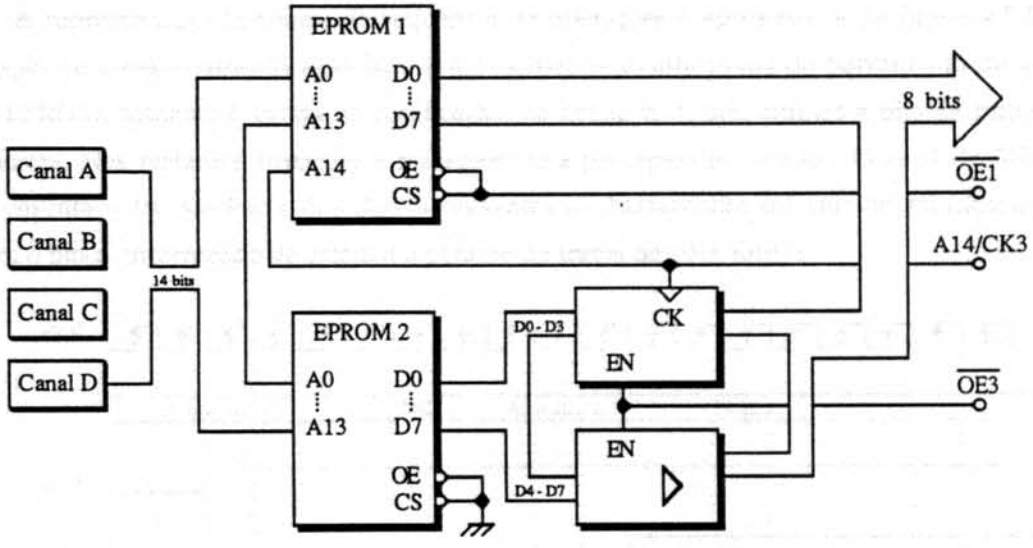


Fig. 4.2-7 Codificação e compressão na versão 1 do multiplexador.

Os dois blocos associados à EPROM 2 encarregam-se de armazenar e isolar do barramento interno, o resultado da codificação efectuada sobre os quatro últimos bits da 1ª amostra e os primeiros quatro da amostra seguinte, respectivamente.

Versão 2 e 3

Uma só EPROM é utilizada nestas duas versões, como se pode observar na figura 4.2-8. Para possibilitar a codificação em 3 tempos (um por cada octeto), utilizou-se uma EPROM do tipo 27512 (14 linhas de endereço para os bits da amostra + 2 linhas de endereço para seleccionar a tabela activa em cada instante.

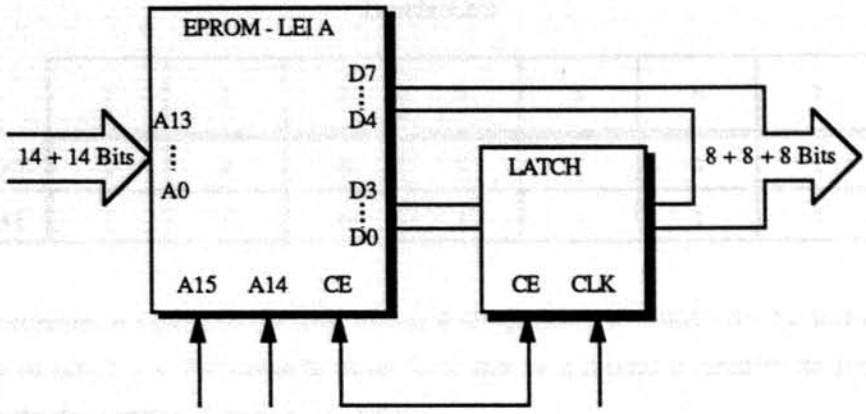


Fig. 4.2-8 Codificação e compressão nas versões 2 e 3 do multiplexador.

A representação temporal da sequência de operações é apresentada na figura 4.2-9. A função de armazenamento associada aos bits menos significativos do barramento de saída da EPROM, somente é válida na codificação, no octeto n+1, dos últimos 4 bits da primeira amostra. Nas restantes situações é transparente à passagem dos dados. O sinal de SH/LD representa a transferência dos dados presentes no barramento interno do multiplexador para o bloco encarregado de efectuar a geração da trama de 2048 KBit/s.

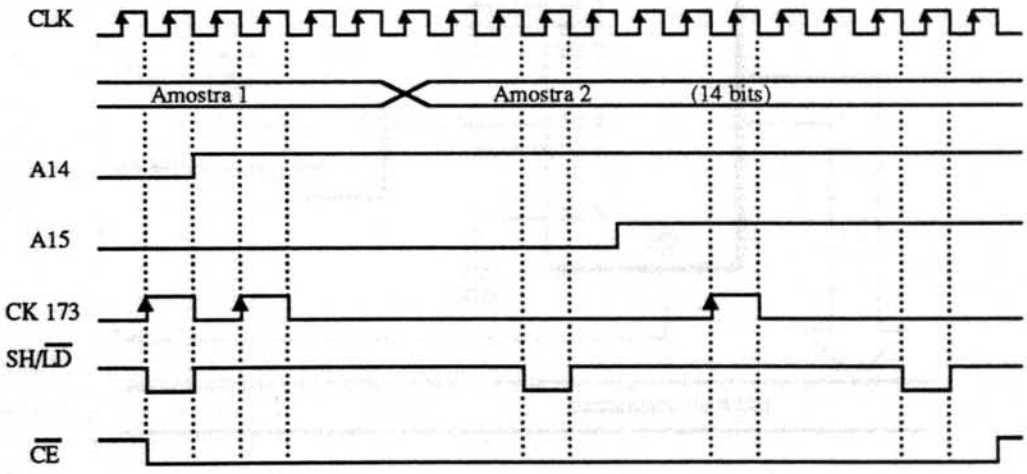


Fig. 4.2-9 Diagrama temporal da sequência de operações de codificação e compressão: versão 2 e 3

Inserção do sinal de alinhamento de trama

O sinal de alinhamento de trama (SAT), tal como é recomendado nas normas, é introduzido no octeto 0 e em tramas alternadas. Nas tramas sem SAT (tramas impares) o octeto 0 é preenchido com um sinal, que como se pode constatar no quadro 4.2-2, lhe é muito semelhante.

Tabela 4.2-2

	1	2	3	4	5	6	7	8
Trama com SAT	1	0	0	1	1	0	1	1
Trama sem SAT	1	1	0	1	1	1	1	1

Efectivamente, o octeto Ø de uma trama é simplesmente obtido do da trama anterior invertendo os bits 2 e 6. Foi baseado neste facto que se adoptou o circuito da figura 4.2-10 para a geração dos octetos Ø de todas as tramas.

Como se pode verificar os bits 2 e 6 são obtidos através de um registo que comuta de estado a cada transição ascendente do relógio de sincronismo de trama (8 KHz). Os restantes bits são pré-programados já que o seu valor é inalterável. Assim, de cada vez que o sinal de controlo é activado pela máquina de estados, os bits relativos ao octeto zero são transferidos para o barramento interno do multiplexador.

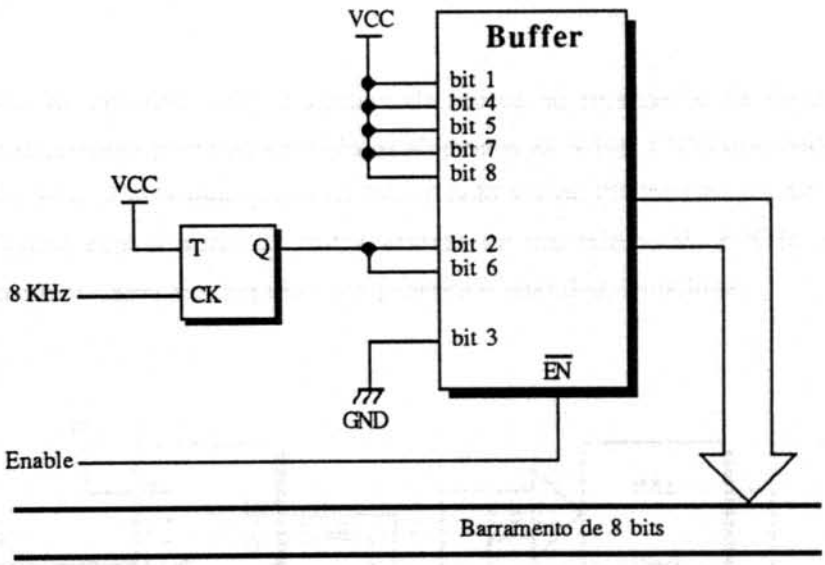


Fig. 4.2-10 Geração da palavra de alinhamento de trama.

Nas versões 2 e 3, no caso de existir uma trama de 2048 KBit/s presente na entrada do multiplexador, é utilizado o SAT que esta transporta de forma a não alterar a informação de sinalização e alarme que este possa conter.

Geração da trama NRZ

A geração do sinal NRZ a partir dos dados existentes no barramento interno de 8 bits consiste apenas em efectuar a serialização ordenada desses mesmos dados.

A conversão paralelo/série está a cargo de um *shift-register* cujo sinal de relógio é o próprio sincronismo de bit (2048 KHz). Os dados a serem serializados são transferidos do barramento interno de alta impedância de 8 bits, por activação dos diversos sinais de controlo. A unidade de controlo activa um destes sinais após o envio do último bit do octeto anterior, pelo que na transição para o primeiro bit do octeto seguinte os 8 bits existentes no barramento são armazenados conversor série/paralelo.

Unidade de controlo - Máquina de estados e sua temporização

É a unidade central do multiplexador, e baseia-se numa máquina de 256 ou 128 estados de decisão conforme as versões, controlada a partir do relógio de 2048 KHz e sincronizada pelo relógio de 8 KHz ou 16 KHz respectivamente. Tal máquina é capaz de gerar os vários sinais de controlo por forma a comandar sincronamente todos os blocos do multiplexador.

É neste bloco que se registam as maiores diferenças entre as versões desenvolvidas, e será por isso objecto de uma análise mais detalhada e particularizada.

Versão 1

A unidade de controlo, cujo diagrama de blocos se representa na figura 4.2-11, é constituída basicamente por dois contadores síncronos de 4 bits ('161) que endereçam um bloco combinacional constituído por uma PAL (16L8) e uma PROM (28L22). Os contadores são sincronizados com o inicio da trama através de um relógio de 8 KHz que, na sua transição ascendente, gera um impulso que provoca o *reset* dos contadores.

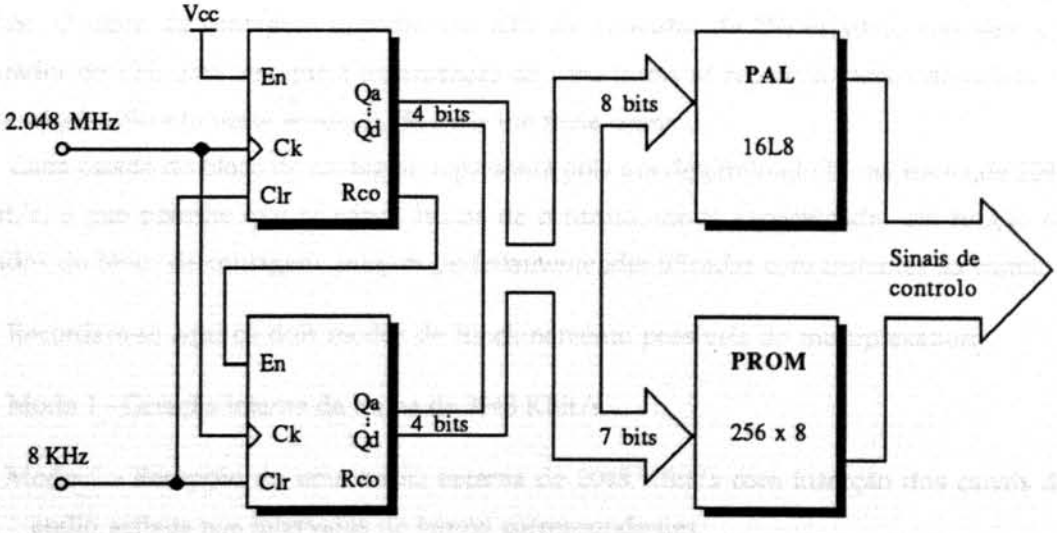


Fig. 4.2-11 Diagrama de blocos da unidade de controlo do multiplexador (versão 1).

Convém agora explicar com um pouco de detalhe o porquê da escolha de uma PROM e de uma PAL na geração dos sinais de controlo. A geração de um certo número de sinais, nomeadamente o OE1 e OE3, exige um número de somas de produtos que excede a capacidade de uma PAL de 'tamanho' aceitável, o que impossibilita logo à partida a sua utilização na geração de todos os sinais de controlo. A escolha de uma PROM para a sua geração também não era aconselhável dado que alguns sinais necessitam de 9 variáveis de entrada o que implicaria o uso de uma PROM de 512x8. Tal facto, embora não limitativo, traduzir-se-ia num grande desperdício da sua capacidade e do espaço disponível na placa de circuito impresso.

Optou-se por isso por uma solução mista. Uma PROM de 256x8 suportaria os sinais de geração mais complexa, mas que possuíssem um menor número de variáveis de entrada, enquanto que a PAL suportava os sinais de mais fácil geração (equações menores) que exigissem no entanto um maior número de variáveis de entrada.

Chegou-se assim à solução adoptada, que aproveita ao máximo os recursos disponíveis sem qualquer contrapartida em termos de eficiência.

Versão 2

Nesta versão a unidade central do multiplexador baseia-se numa máquina de 128 estados de decisão, controlada a partir do relógio de 2048 KHz e sincronizada pelo relógio de 16 KHz. Esta máquina de estados, assim como alguns blocos anteriormente descritos foram implementados numa EPLD do tipo EP1810JC da Texas Instruments.

Trata-se de um esquema tradicional no qual foi efectuada uma pequena alteração de maneira a optimizar a utilização da EPLD.

Uma trama de 2048 KBit/s está dividida em 32 octetos perfazendo um total de 256 bits (32x8). O bloco de contagem implementa, não um contador de 256 estados, mas sim um contador de 128, uma vez que a organização de uma trama se repete, a menos do octeto 16 (que não é utilizado neste sistema), de meia em meia trama.

Cada estado do bloco de contagem representa pois um determinado bit na trama de 2048 KBit/s, o que permite que as várias linhas de controlo, sendo especificadas em função de estados do bloco de contagem, estejam perfeitamente identificadas com instantes da trama.

Recordam-se aqui os dois modos de funcionamento possíveis do multiplexador:

Modo 1 - Geração interna da trama de 2048 KBit/s.

Modo 2 - Recepção de uma trama externa de 2048 KBit/s com inserção dos canais de audio activos nos intervalos de tempo correspondentes.

Analisando o funcionamento dos dois modos verifica-se que a maior dificuldade reside,

quando o circuito se encontra em funcionamento no modo 2, na sincronização do estado Ø do bloco de contagem com o bit Ø da trama externa de forma que os sinais de controlo estejam sincronizados com os instantes requeridos da trama.

O circuito de detecção do alinhamento de trama utilizado, PEB 2030 da Siemens, permite a sincronização dos dados série ou paralelo por um sinal de 4 KHz externo (SCT-Station Counter Trigger pulse).

Como a frequência mínima que é possível obter com um contador de 128 estados e com um relógio de 2048 KHz é de 2048/128 = 16 KHz, foi necessário implementar um circuito divisor adicional, na EPLD, para obter o sinal desejado de 4 KHz, com a temporização ilustrada na figura 4.2-12, de sincronização do bit Ø da trama de 2048 KBit/s com o estado Ø do bloco de contagem.

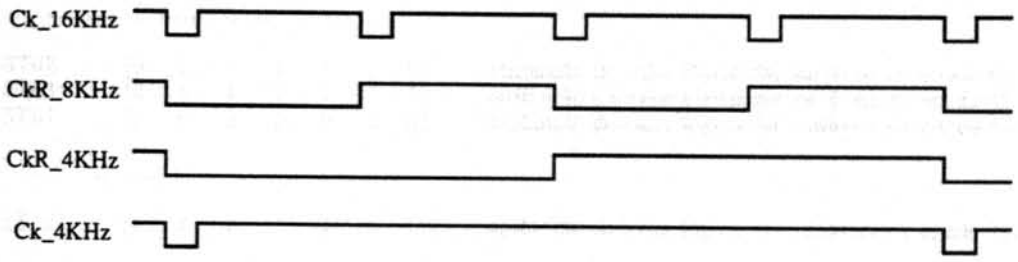


Fig. 4.2-12 Diagrama temporal de geração do relógio de 4 KHz.

Para a implementação de todo o bloco de controlo na EPLD, efectuada sob a forma de uma máquina de estados, foi utilizada a linguagem de descrição de hardware de alto nível especificada na ferramenta de desenvolvimento utilizada, Aplus (TI, 88). O diagrama de estados apresentado na figura 4.2-13, sugere uma organização sequencial com transições unívocas entre estados.

Fig. 4.2-13 Máquina de estados da unidade de controlo do multiplexador (versão 2).

Apresenta-se parte da descrição efectuada, realçando a facilidade com que é possível descrever uma máquina de estados utilizando este tipo de linguagem. Em anexo apresenta-se a descrição completa da unidade de controlo do multiplexador.

```
MUX - MAQ. ESTADOS     PEDRO CARDOSO/INESC NORTE     EP1810J

INPUTS:                                          ; uma máq. de estados sequencial não necessita de entradas

OUTPUTS: c0, c1, c2, c3, c4, c5, c6              ; uma máquina de 128 estados necessita de 7 variáveis

MACHINE: MUX                                     ; nome atribuído à máquina de estados

    CLOCK: clk          CLEAR: resn              ; sinais de CLOCK e CLEAR da máquina de estados

    STATES:  [c6  c5  c4  c3  c2  c1  c0]
    ST0      [0   0   0   0   0   0   0]   ; atribuição do valor lógico das variáveis no estado 0
    ST1      [0   0   0   0   0   0   1]   ; atribuição do valor lógico das variáveis no estado 1

     ..................................................

    ST62     [0   1   1   1   1   1   0]   ; atribuição do valor lógico das variáveis no estado 62
    ST63     [0   1   1   1   1   1   1]   ; atribuição do valor lógico das variáveis no estado 63
    ST64     [1   0   0   0   0   0   0]   ; atribuição do valor lógico das variáveis no estado 64

     ..................................................

    ST127    [1   1   1   1   1   1   1]   ; atribuição do valor lógico das variáveis no estado 127

                                             ; palavra reservada que indica o início da descrição
BEGIN

ST0: ST1                                     ; transição incondicional do estado 0 para o estado 1
     OUTPUTS: Sckbyte                        ; sinal Sckbyte activo durante o estado 0

ST1: ST2                                     ; transição incondicional do estado 1 para o estado 2
     OUTPUTS: Sce541 Scepeb                  ; sinais Sce541 e Scepeb activos durante o estado 1

ST2: ST3                                     ; transição incondicional do estado 2 para o estado 3

     ..................................................

ST61: ST62                                   ; transição incondicional do estado 61 para o estado 62

ST62: ST63                                   ; transição incondicional do estado 62 para o estado 63
     OUTPUTS: RenC SckB                      ; sinais RenC e SckB activos durante o estado 62

ST63: ST64                                   ; transição incondicional do estado 63 para o estado 64
     OUTPUTS: Rckbyte                        ; sinal Rckbyte activo durante o estado 63
          IF select THEN Rcepeb              ; se select = 1 então Rcepeb activo durante o estado 63

ST64: ST65                                   ; transição incondicional do estado 64 para o estado 65
     OUTPUTS: SenC Sckbyte Sa15              ; sinais SenC,Sckbyte e Sa15 activos durante o estado 64

     ..................................................

ST126: ST127                                 ; transição incondicional do estado 126 para o estado 127
     OUTPUTS: Rck16 SckB                     ; sinais Rck16 e SckB activos durante o estado 126
          IF !select THEN Rcepeb             ; se select = 0 então Rcepeb activo durante o estado 126

ST127: ST0                                   ; transição incondicional do estado 127 para o estado 0
     OUTPUTS: Rckbyte Sck16                  ; sinais Rckbyte e Sck16 activos durante o estado 127
          IF select THEN Rce541              ; se select = 1 então Rce541 activo durante o estado 127

END$                                         ; palavra reservada que indica o fim da descrição
```

O multiplexador recebe de cada um dos conversores A/D um sinal canX (X=A, B, C, D ou E), que lhe indica a presença de amostras válidas de 384 KBit/s no barramento. Estes 5 sinais juntamente com um sinal que indica a presença de uma trama codificada em HDB3 na entrada do multiplexador (Lip - Loss of Input), são multiplexados num único sinal (select).

O sinal *select* indica à unidade de controlo a presença de uma trama de 2048 KBit/s em trânsito pelo sistema, para que este possa preservar o SAT que esta contém, e quais os canais activos, para que apenas estes sejam incluídos na trama de saída, seja esta gerada, ou não, internamente.

Por uma questão de simplificação na descrição da máquina de estados, e de optimização dos recursos existentes na EPLD, nomeadamente a utilização dos registos existentes em cada macrocélula (utilizados na configuração SR (Set/Reset)), decompôs-se cada sinal de controlo em dois sinais que reflectissem apenas as transições do sinal original.

O sinal *ckbyte* (ver figura 4.2-14), por exemplo, é decomposto em dois sinais que reflectem respectivamente as suas transições ascendentes (Sckbyte) e transições descendentes (Rckbyte), o primeiro aplicado no pino de *set* do registo e o segundo no pino de *reset*. Sincronizando o funcionamento do SR com o relógio invertido de 2048 KHz, obtem-se na saída o sinal de controlo desejado.

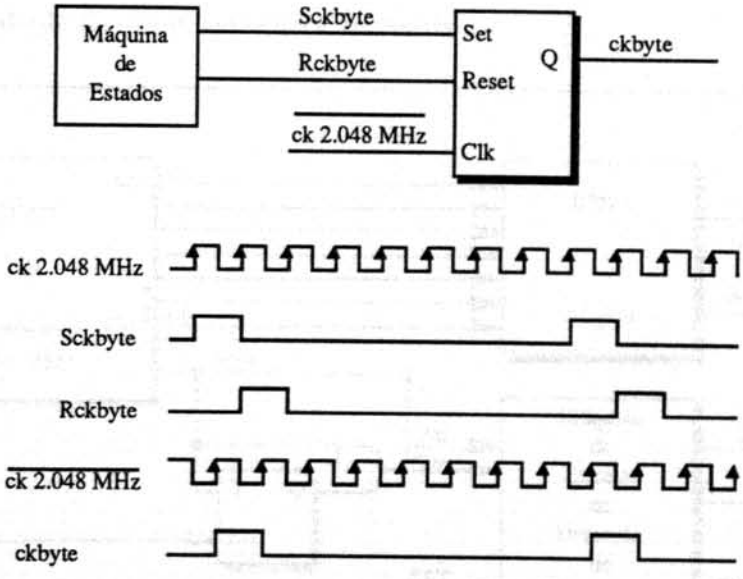


Fig. 4.2-14 Aproveitamento dos recursos disponíveis na EP1810J para a geração simplificada dos diversos sinais de controlo.

No caso de um sinal com um número de transições de um dos tipos, ascendente ou descendente, superior a 8 pode verificar-se a situação de o algoritmo de minimização do

Aplus, não conseguir reduzir a um valor igual ou inferior a 8 o número de soma de produtos que representam equação Booleana do sinal. Nesta situação é necessário a utilização de células NOCF (Non Output Combinatorial Feedback) que efectuam a realimentação do sinal para uma nova macrocélula, desaproveitando assim recursos da EPLD (um pino de saída por cada realimentação efectuada), e provocando um atraso adicional numa parte dos sinal decompostos, mas permitindo implementar funções em que o número de termos de produto ultrapasse o valor 8. Este tipo de atrasos é insignificante neste caso, uma vez que o período de relógio é de 488 ns, valor muito superior ao tempo de realimentação, podendo tornar-se crítico em implementações com períodos de relógios bastante menores (T≈50ns => f≈20 MHz).

Versão 3

A implementação da unidade de controle na versão 3, ao contrário de outros blocos similares aos da versão 2, e devido à integração de todo o bloco lógico do multiplexador num único dispositivo lógico programável, com excepção da tabela-lei A, é efectuada de forma particular, como se pode observar na figura 4.2-15. Assim, e um pouco à semelhança da versão 1, é utilizado um bloco de contagem cujas saídas endereçam o bloco lógico que define os diversos sinais de controle. A adopção de um contador resulta da necessidade de rentabilizar a lógica existente. A integração, no PLD, do bloco de detecção do alinhamento de trama, e a existência neste bloco de um contador de estados, permite a sua partilha e a consequente redução da quantidade lógica necessária.

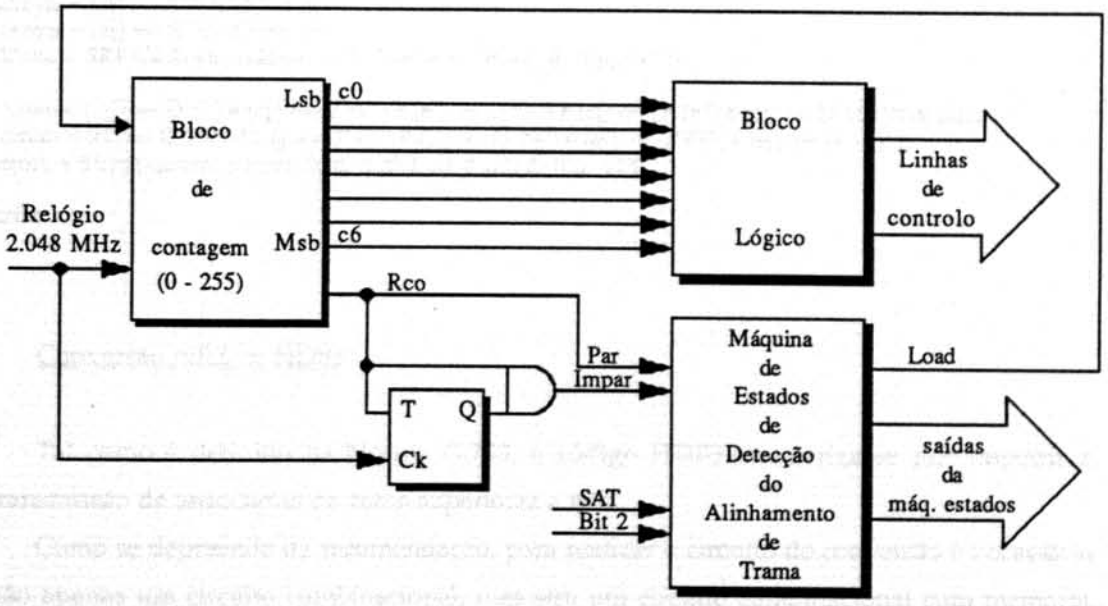


Fig. 4.2-15 Diagrama de blocos da unidade de controlo do multiplexador (versão 3).

A sincronização do estado Ø do bloco de contagem com o bit Ø da trama em trânsito, é

efectuada pela máquina de estados que implementa a detecção da alinhamento de trama, reinicializando sempre que necessário o contador referido.

A descrição da unidade de controle (Anexos), bem como de toda a implementação no PLD da versão 3, foi efectuada utilizando a linguagem AHDL (Altera Hardware Description Language). Trata-se de uma linguagem de descrição alto nível, com uma sintaxe um pouco à semelhança de algumas linguagens de programação, casos do Pascal e C. É possível a inclusão, por referência, de outros blocos descritos numa das múltiplas opções oferecidas pela ferramenta Max+Plus utilizada, facilitando desta maneira a descrição hierárquica de um sistema digital (Altera, 90b).

Apresenta-se uma parte da descrição efectuada para que seja possível a comparação com a efectuada relativamente à versão 2.

```
TITLE "Máq. de estados de controlo do Mux de 2.048 Mbits";     DESIGN IS "CONTROL" DEVICE IS "AUTO";

FUNCTION SRFF(s, r, clk, clrn, prn) RETURNS (q);

SUBDESIGN CONTROL     (clk, lip, st[2..0], c[6..0], canA, canB, canC, canD, canE :INPUT;
                       ckbyte, cerom, select :OUTPUT;)

VARIABLE              Rckbyte, Sckbyte, Rcerom, Scerom :NODE;

BEGIN

select = MCELL (   (c[]  >   D"4")  &  (c[] <  D"27")  &  !canA
                 #  (c[]  >  D"28")  &  (c[] <  D"51")  &  !canB
                 #  (c[]  >  D"52")  &  (c[] <  D"75")  &  !canC
                 #  (c[]  >  D"76")  &  (c[] <  D"99")  &  !canD
                 #  (c[]  > D"100")  &  (c[] < D"124")  &  !canE
                 #  ((c[] > D"124")  #  (c[] <   D"4"))  &  lip   );


rckbyte = (c[] == B"XXXX111");
sckbyte = (c[] == B"XXXX000");
ckbyte = SRFF(sckbyte, rckbyte, !clk, !(st0 # st1 # st2) & !lip, VCC);

rcerom = ((c[] == D"6") # (c[] == D"30") # (c[] == D"54") # (c[] == D"78") # (c[] == D"102")) & select;
scerom = (c[] == D"25") # (c[] == D"49") # (c[] == D"73") # (c[] == D"97") # (c[] == D"121");
cerom = SRFF(scerom, rcerom, !clk, !(st0 # st1 # st2) & !lip, VCC);

END;
```

Conversão NRZ -> HDB3

Tal como é definido na Norma G.703, o código HDB3 caracteriza-se por impedir a transmissão de sequências de zeros superiores a três.

Como se depreende da recomendação, para realizar o circuito de conversão é necessário, não apenas um circuito combinacional, mas sim um circuito combinacional com memória, mais precisamente com seis bits de memória. São usados 4 bits para armazenar o sinal NRZ e assim permitir a análise de 4 bits consecutivos, 1 bit para para guardar a polaridade do último '1' transmitido e 1 bit para guardar a polaridade da última violação transmitida.

Para implementar o conversor na versão 1 do multiplexador, optou-se pois por uma PAL 16R6, o que permite a realização do conversor utilizando apenas um circuito integrado de reduzidas dimensões.

Na versão 2, devido à existência de conversão bidireccional, e por falta de recursos de integração que permitissem incluir na EPLD os blocos de conversão NRZ->HDB3 e HDB3->NRZ, optou-se pela utilização de um circuito integrado específico, o MV1441 da Motorola.

Versão 3

A capacidade de integração lógica disponível permitiu a implementação dos dois blocos de conversão no PLD utilizado, com as vantagens inerentes de compactação e eliminação de um circuito integrado.

Apresenta-se a descrição AHDL do conversor NRZ->HDB3.

```
TITLE "CONVERSOR NRZ->HDB3";          DESIGN IS "NRZ" DEVICE IS "AUTO";
SUBDESIGN NRZ          (clk, nrz :INPUT; hmais, hmenos :OUTPUT;)
VARIABLE               spvi, spum, flop[3..0] :DFF;
BEGIN
  (flop[], spvi, spum).clk = clk;
  spvi.d = spvi XOR (!flop3 & !flop2 & !flop1 & !flop0)
  spum.d = !( spum &  flop0
        # !spum & !flop0 &  flop1
        # !spum & !flop0 &  flop2
        # !spum & !flop0 &  flop3
        # !spvi & !flop0 & !flop1 & !flop2 & !flop3);
  flop3.d = nrz;
  flop2.d = flop3 XOR (!flop3 & !flop2 & !flop1 & !flop0);
  flop1.d = flop2;
  flop0.d = flop1;
  hmais = !( flop0 &  spum & !clk # spvi & spum & !flop3 & !flop2 & !flop1 & !flop0 & !clk);
  hmenos = !( flop0 & !spum & !clk # !spvi & !spum & !flop3 & !flop2 & !flop1 & !flop0 & !clk);
END;
```

4.2.2. Desmultiplexador de audio a 2048 KBit/s

Nesta secção descreve-se o sistema de desmultiplexagem a 2048 KBit/s, com capacidade

para quatro canais monofónicos (5 nas versões 2 e 3 do referido sistema). O sistema efectua a interface directa com um bloco de conversores D/A possuindo todas as compatibilidades por estes exigidas.

As diferenças de funcionalidade entre as diversas versões são aqui muito menos acentuadas, com excepção para o nível de integração obtido, e para a inclusão de um ou outro bloco ou de uma ou outra flexibilização no desempenho global do desmultiplexador.

A particularidade da disposição das amostras de cada canal na trama (duas amostras consecutivas em cada meia trama), obriga à existência de um sub-bloco suplementar de igualização do espaçamento das amostras de cada canal, de forma que os conversores D/A recebam um fluxo paralelo de dados igualmente espaçados à cadência recomendada de 32 KHz.

Sinais de interface com o sistema

Sinais recebidos pelo desmultiplexador

- Trama série a 2048 KBit/s codificada em HDB3

De acordo com as recomendações CCITT, o desmultiplexador recebe uma trama série codificada em HDB3, com os diversos canais de audio multiplexados. Apenas a versão 3 do sistema permite efectuar teste globais de funcionalidade ao sistema, com a inclusão de um modo de teste, seleccionável internamente por um interruptor, admitindo um sinal NRZ externo de 2048 KBit/s e o respectivo relógio.

Sinais gerados pelo desmultiplexador

- Dados para os conversores D/A e respectivos sinais de controlo

O problema de colocação dos *buffers* repete-se no desmultiplexador. Na primeira versão estes estão colocados no desmultiplexador. Nas versões 2 e 3 aproveitam-se os conversores paralelo/série do tipo 74LS597 existentes nos conversores D/A para essa função (novamente se realça o facto de os conversores D/A utilizados requererem um sinal de entrada série (Ferreira, 90)).

Na versão 1, a utilização de um barramento multiplexado, canais A e C de um lado e canais B e D do outro, permite uma maior rentabilização do hardware utilizado, sem contudo comprometer as especificações requeridas para o audio de alta qualidade. Assim aparecem ligados a este barramento multiplexado, dois pares de sinais de conversão (um

para os 7 bits mais significativos e um outro para os 7 bits menos significativos), um ligado ao canal estereofónico A+B e um outro ao canal estereofónico C+D, que permitem a recolha das amostras pelo conversor D/A, à cadência especificada (32 KHz).

Nas versões seguintes, em que já é possível a desmultiplexagem dos 5 canais especificados nas normas, o barramento de interface é único pelos motivos acima apontados. É gerado um par de sinais de controlo afectos a cada canal, um dos quais controla os 8 bits mais significativos enquanto o segundo se encarrega dos 6 bits menos significativos (simplifica-se assim a geração do fluxo série nos conversores D/A).

Descrição do desmultiplexador por blocos

Conversão HDB3->NRZ e Detecção na entrada de presença de um sinal HDB3

A conversão HDB3->NRZ, nas versões 1 e 2 do desmultiplexador é efectuada pelo circuito dedicado MV1441 da Motorola. Se na primeira versão a utilização deste circuito se deveu a uma necessidade de contenção de recursos, aproveitando a sua particularidade de recuperação de relógio, na segunda versão, em que a recuperação de relógio é efectuada com o auxílio de uma PLL (Phased Locked Loop), a sua utilização deveu-se à falta de recursos de integração disponíveis na EP1810J e à urgência da inclusão do SITAD na demonstração do projecto global SIFO, que obstou ao desenvolvimento de um bloco de conversão passível de integração num PLD do tipo PAL.

A detecção na entrada de presença de um sinal HDB3 é sinalizada para o exterior através de um LED. Nestas duas versões o circuito de detecção é parte integrante da funcionalidade do circuito dedicado utilizado, MV1441. Este sinal é de extrema importância no multiplexador sendo o selector do seu modo de funcionamento.

Versão 3

A utilização de um PLD com capacidade de integração substancialmente superior, o EPM5128J da Altera, permitiu, como já foi referido, a integração do bloco de conversão HDB3->NRZ no dispositivo utilizado.

```
TITLE "Conversor HDB3->NRZ";    DESIGN IS "HDB3" DEVICE IS "AUTO";
SUBDESIGN HDB3          (clk, hmais, hmenos :INPUT; nrz, pll_sum  :OUTPUT;)
VARIABLE                in1,in2,marca,viola,  flop[4..1] :DFF;
BEGIN
  pll_sum = !hmais # !hmenos;
```

```
%Inicio da descricao do bloco de conversao HDB3-NRZ%

  in1.d = VCC;              in2.d = VCC;
  in1.clk = hmais;          in2.clk = hmenos;
  in1.clrn = hmenos;        in2.clrn = hmais;

  (flop[], viola, marca).clk = clk;

  marca.d = !(hmais & hmenos);      viola.d = !hmais & in1 # !hmenos & in2;

  flop1.d = ! (marca & !viola # marca & flop2 # marca & !flop1);

  flop2.d = !flop1;

  flop3.d = flop2;

  flop4.d = ! ((viola & flop1 & !flop2) # !flop3);

  nrz = flop4;

END;
```

A detecção da presença de um sinal HDB3 na entrada foi também integrado no PLD. A sinalização de falta ocorre quando é detectada na entrada uma sequência de "Øs" superior a 11. O mesmo bloco implementa ainda a validação do sinal recebido do canal de transmissão. Alternâncias simultâneas (erro_V) ou 3 alternâncias consecutivas com a mesma polaridade (erro) são consideradas violações do código. Foi utilizado o método OHE (ver capítulo 2.6.1) na descrição do diagrama de estados da figura 4.2-16.

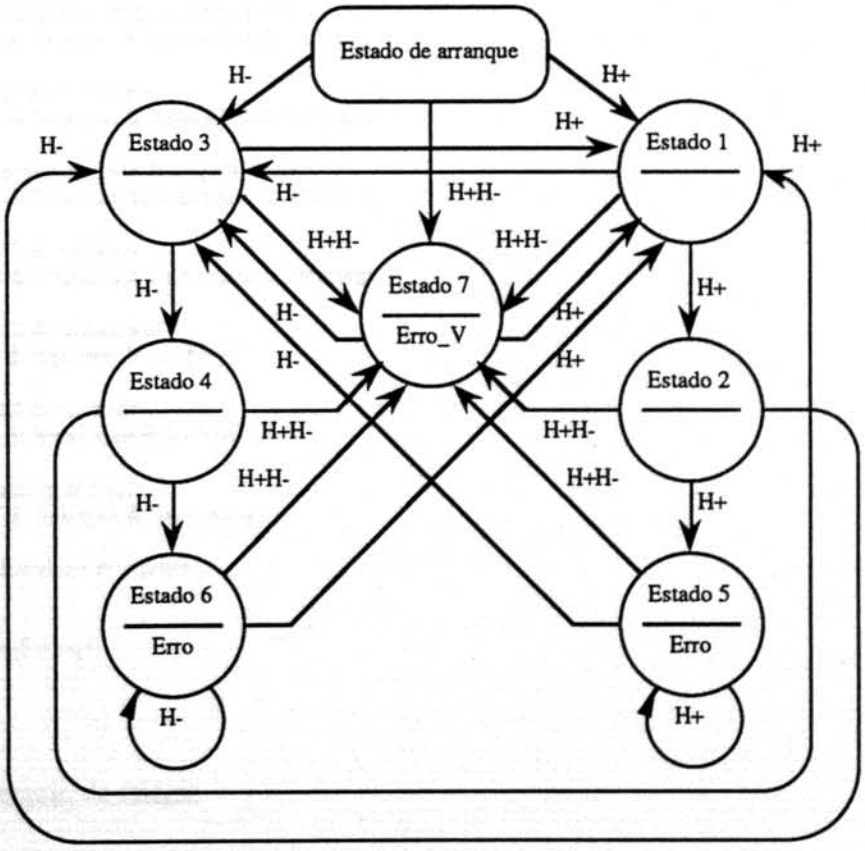


Fig. 4.2-16 Digrama de estados do circuito detector de erro (no sinal HDB3 recebido)

implementado na versão 3 do desmultiplexador.

Utilizando a linguagem AHDL apresenta-se a descrição dos módulos referidos.

```
TITLE "Detector do sinal HDB3";    DESIGN IS "ERRO" DEVICE IS "AUTO";

INCLUDE "74163";

SUBDESIGN ERRO          (clk, reset, hmais, hmenos :INPUT; erro, erro_v, lip  :OUTPUT;)

VARIABLE                stq[7..1] :DFF; count :74163; maispos, menospos, doisneg :NODE;

BEGIN

%Inicio da descrição do gerador do sinal de falta de HDB3%

   count.(clk, ldn, clrn, enp, ent, d, c, b, a) = (clk, hmais & hmenos, VCC, VCC, VCC, GND, VCC, GND, GND);

   lip = SRFF(count.rco, GND, !clk, !flop4, VCC);

%Inicio da descrição do bloco de erro%

   stq[].clk  = clk;
   stq[].clrn  = !reset;

%Definição dos instantes de transição%

   maispos = hmais & !hmenos;
   menospos = !hmais & hmenos;
   doisneg = !hmais & !hmenos;

%Descodificação dos estados da máquina%

   stq1.d = menospos & !stq1 & !stq2 & !stq5
        # stq1 & !maispos & !menospos & !doisneg;

   stq2.d = stq1 &  menospos
        # stq2 & !maispos & !menospos & !doisneg;

   stq3.d = maispos & !stq3 & !stq4 & !stq6
        # stq3 & !maispos & !menospos & !doisneg;

   stq4.d = stq3 &  maispos
        # stq4 & !maispos & !menospos & !doisneg;

   stq5.d = stq2 &  menospos
        # stq5 & !maispos & !doisneg;

   stq6.d = stq4 &  maispos
        # stq6 & !menospos & !doisneg;

   stq7.d = doisneg & !stq7
        # stq7 & !maispos & !menospos;

%Definição dos sinais de saída%

   erro = stq7;
   erro_v = stq5 # stq6;

END;
```

Recuperação de relógio

A recuperação rudimentar de relógio efectuada na versão inicial aproveita as facilidades

do circuito dedicado de conversão HDB3<->NRZ, MV1441 da Motorola. Utiliza-se um cristal com uma frequência de oscilação 8 vezes superior (16384 KHz) à frequência do relógio de bit 2048 KHz. O preço a pagar, tal como se pode observar na figura 4.2-17, é um sinal com *jitter* apreciável em relação ao relógio do codificador. O sinal usado como referência é o relógio de 4096 KHz fornecido pelo multiplexador ao bloco de conversores A/D.

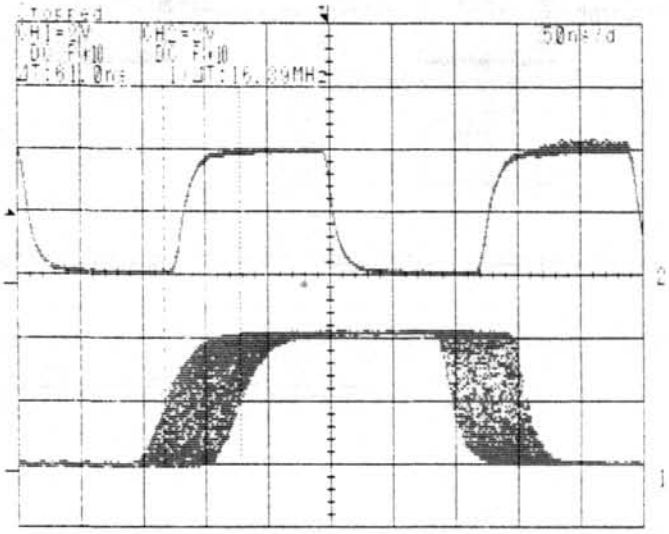


Fig. 4.2-17 *Jitter* no relógio recuperado (versão 1).

Recordemos uma das alíneas da recomendação G.823 do CCITT a respeito do controlo do *jitter* na hierarquia dos 2048 KBit/s: "*...se não for exercido um controlo suficiente, então em determinadas circunstâncias, pode verificar-se uma acumulação excessiva de jitter ao ponto de poderem ocorrer as seguintes situações: ... a degradação da codificação digital da informação analógica como resultado da modulação de fase das amostras recebidas no conversor D/A no termo da conexão*".

Cientes da importância da recomendação e das consequências que uma acumulação excessiva de *jitter* poderia causar do desempenho global do sistema, procurou-se, nas versões 2 e 3, melhorar este aspecto específico com a introdução de um circuito recuperador de relógio baseado numa PLL (NE 564 (Signetics, 84)). O bloco implementado, esquematizado na figura 4.2-18, inclui na malha de realimentação dois registos em cascata que permitem, não só a extracção do relógio sem perturbar o desempenho do bloco como, no caso do multiplexador, a extracção do relógio de 4096 KHz requerido pelos conversores A/D.

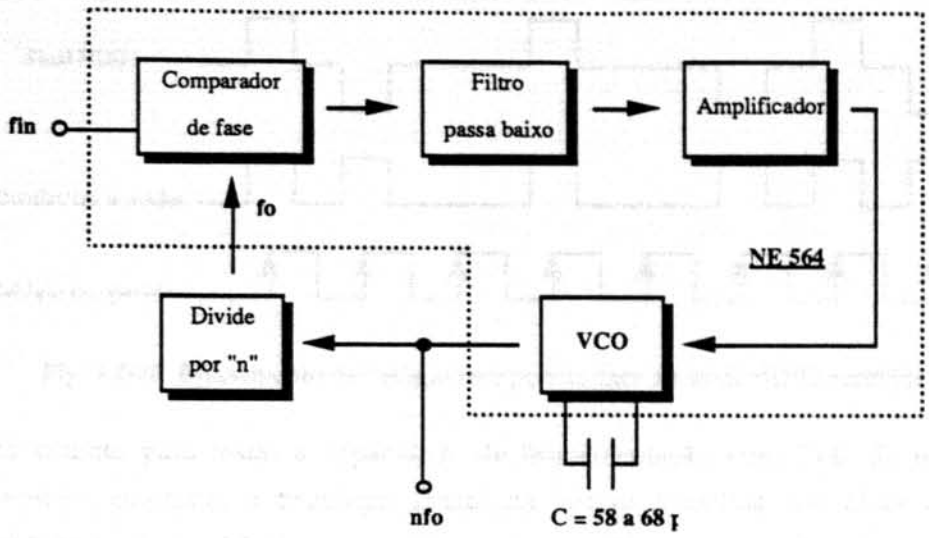


Fig. 4.2-18 Funcionamento elementar de um circuito recuperador de relógio com PLL.

Apresenta-se na figura 4.2-19, a medições de *jitter* observadas no sinal recuperado de 2048 KHz, em condições semelhantes às anteriores.

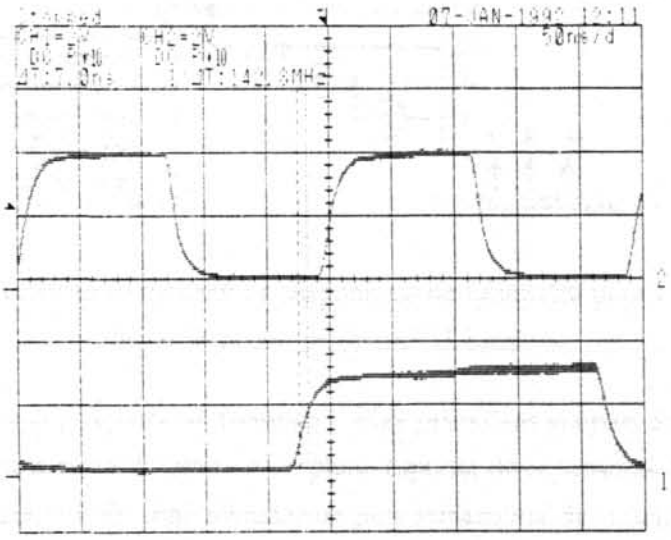


Fig. 4.2-19 *Jitter* no relógio recuperado (versão 2 e 3).

O problema da sincronização do relógio recuperado com os dados codificados em HDB3 não existe, uma vez que o sinal na saída da PLL aparece desfasado de 45° em relação ao sinal de entrada (ver figura 4.2-20), que mais não é do que a soma das duas alternâncias do sinal HDB3 original.

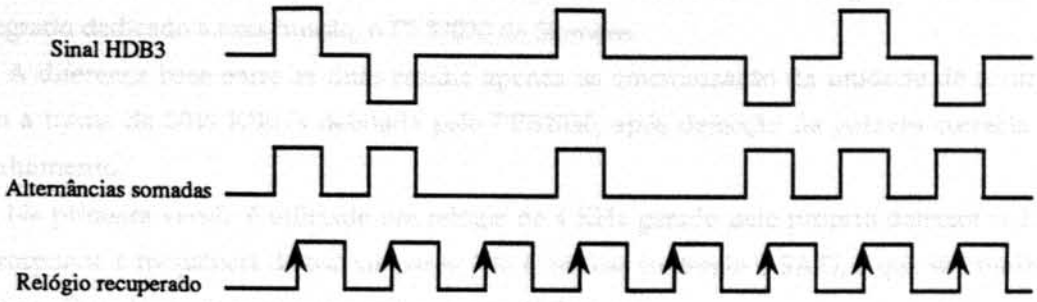


Fig. 4.2-20 Esfasamento do relógio recuperado face ao sinal HDB3 recebido.

No entanto para testar a capacidade de implementação num PLD de um atraso programável contínuo, o protótipo inicial da versão 3 incluía um bloco adicional apresentado na figura 4.2-21, que permitia atrasar o relógio recuperado relativamente ao sinal codificado em HDB3 em intervalos de aproximadamente 45 ns.

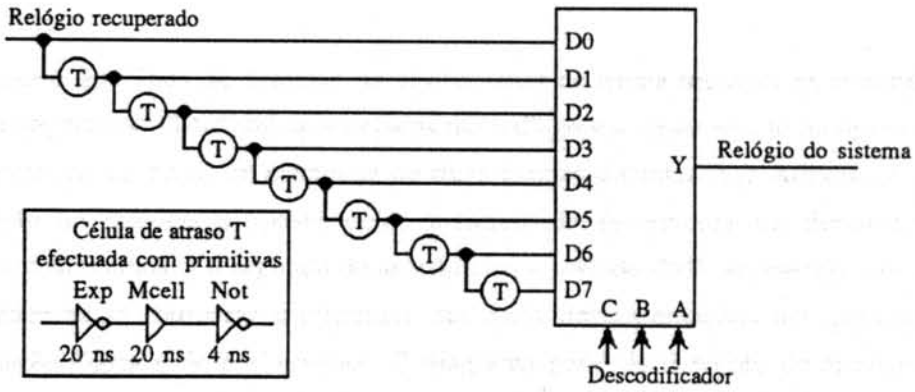


Fig. 4.2-21 Aproveitamento de recursos disponíveis no EPM5128J para a construção de um bloco de atraso programável contínuo.

O bloco de atraso programável contínuo utiliza primitivas simples do PLD. A primitiva "Exp" representa uma soma de produto do plano especial de expansores que a família MAX contém. A primitiva "Mcell", não sintetizável pela ferramenta de desenvolvimento, impõe uma realimentação da macrocélula para o plano de interligação programável (PIA), fixando o atraso igual ao tempo de realimentação de macrocélula do dispositivo. O inversor final, cujo atraso introduzido é pouco significativo, destina-se apenas a compensar a inversão efectuada no sinal pela primitiva "Exp".

Detecção do alinhamento de trama a 2048 KBit/s

A detecção da palavra correcta de alinhamento de trama (SAT) é sinalizada para o exterior através de um LED.

Nas versões 1 e 2, a detecção do alinhamento de trama ficou a cargo de um circuito integrado dedicado a essa função, o PEB2030 da Siemens.

A diferença base entre as duas residia apenas na sincronização da unidade de controle com a trama de 2048 KBit/s debitada pelo PEB2030, após detecção da palavra correcta de alinhamento.

Na primeira versão é utilizado um relógio de 4 KHz gerado pelo próprio detector (4 KHz corresponde à frequência de tramas pares, isto é, tramas contendo o SAT), e que sincroniza o instante Ø do bloco de contagem com o estado Ø da trama.

Na versão seguinte foi adoptada a solução inversa, tirando vantagens da capacidade de armazenamento interno do circuito detector e da possibilidade de sincronização externa do instante inicial de colocação no exterior dos dados alinhados, simplificando, no caso do multiplexador funcionando no modo 2, a sincronização da geração de trama interna com a detecção do alinhamento da trama em trânsito.

Versão 3

Define-se um bloco de detecção de alinhamento de trama segundo as recomendações CCITT, integrado no EPM5128J, que implementa o diagrama representado na figura 4.2-22.

A descrição do bloco foi efectuada de duas formas distintas (ver Anexos), a primeira recorrendo ao método proposto pelos manuais da ferramenta de desenvolvimento utilizada, o Max+Plus, e a segunda de acordo com o método OHE apresentado no capítulo 2.6.1. Procurou-se comparar a eficiência das duas implementações nos parâmetros de compactação lógica e de performance. O diagrama geral de detecção do alinhamento de trama é idêntico para os dois métodos, a menos do bloco "Máquina de Estados", que será designado por MAX_alitra e OHE_alitra respectivamente.

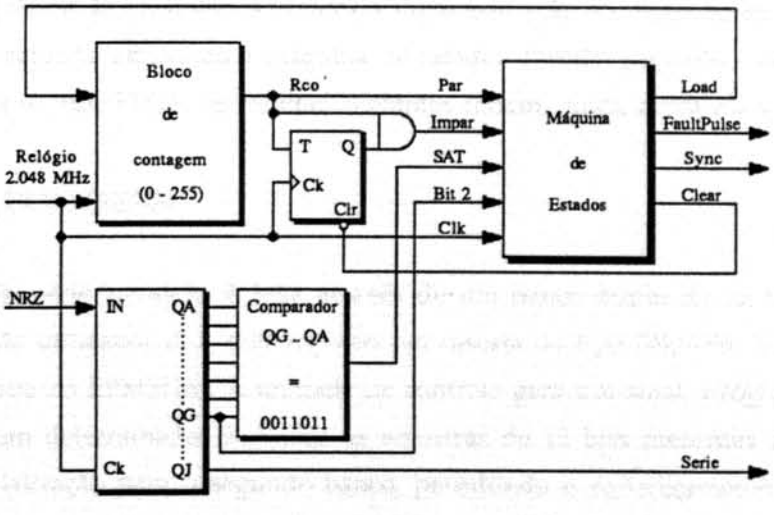


Fig. 4.2-22 Diagrama de blocos do detector de alinhamento de trama (versão 3).

Os resultados obtidos são apresentados na tabela 4.2-3:

Tabela 4.2-3

	Totais da lógica necessária à implementação de cada bloco				
	Pinos de entrada	Pinos de saída	Macrocélulas	Termos de expansão	Frequência Máx.
MAX_alitra	5	4	8	54	28.57 MHz
OHE_alitra	5	4	15	17	38.46 MHz
Alitra_v. MAX	2	3	58	116	7.63 MHz
Alitra_v. OHE	2	3	36	48	9.09 MHz

Uma análise cuidada da tabela, permite-nos obter duas conclusões importantes:

- A implementação da máquina de estados propriamente dita resulta, como seria de esperar, num consumo superior de lógica combinatória no caso MAX_alitra e de lógica sequencial no caso OHE_alitra. A performance inferior do método MAX resulta da utilização intensiva de termos de expansão na implementação da lógica combinatória de descodificação dos estados, com o consequente atraso adicional.

- A integração da máquina de estados no bloco global de detecção do alinhamento de trama, e o aumento consequente do bloco lógico combinatório, satura rapidamente o número de termos de expansão disponíveis na versão MAX, incrementando as realimentações de macrocélula, e resultando num consumo excessivo de recursos lógicos e degradação de performance quando comparada com a versão OHE.

Em consequência dos resultados obtidos a implementação da detecção do alinhamento de trama foi efectuada utilizando a máquina de estados descrita segundo o método OHE. Numa estrutura do tipo FPGA, os resultados obtidos seriam, quiçá, ainda mais conclusivos.

Conversão Série/Paralelo

A conversão série/paralelo, é feita através de um banco duplo de 12 registos. Nas versões 1 e 2 são utilizados dois *shift-registers* em cascata do tipo 74LS595. Na versão 3 o banco é integrado no EPM5128J. A unidade de controlo gera um sinal, relógio de captura, que transfere em determinados instantes as amostras de 12 bits presentes do banco de registos de serialização para o segundo banco, permitindo o endereçamento paralelo do bloco seguinte de descodificação e descompressão.

Verificação da paridade - versão 3

Aproveitando os recursos de integração disponíveis, foi implementado nesta versão o detector de paridade ilustrado na figura 4.2-23.

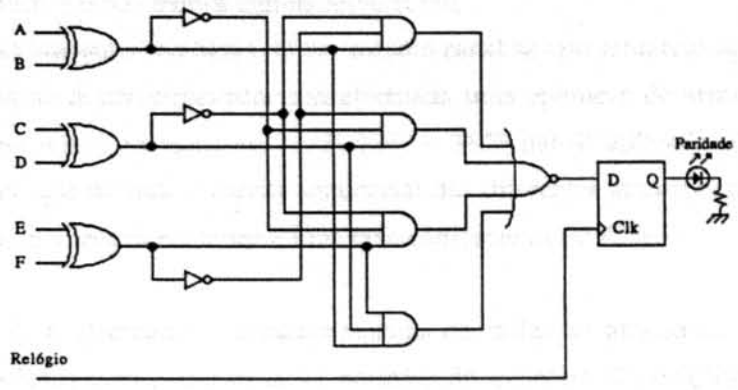


Fig. 4.2-23 Circuito para detecção de erros de paridade segundo a lei A.

O funcionamento do circuito é elementar. A ocorrência de um número par de "1s" nos seis bits verificados traduzir-se-à na sinalização de uma situação ilegal. Lembre-se que, de acordo com as recomendações, é efectuada uma codificação de paridade impar.

Os sinais verificados são recolhidos após a conversão série/paralelo anterior garantindo a sua estabilidade durante o tempo equivalente à duração de 12 bits. O registo, comandado por um sinal de relógio destina-se fundamentalmente a eliminar possíveis descodificações ruidosas.

Descodificação e Descompressão da lei A

As amostras paralelas que endereçam o bloco permanecem válidas na entrada durante um intervalo de tempo equivalente à duração de 12 bits (12 X 488 ns = 5.856 µs). Este facto permite a utilização de uma EPROM para efectuar as operações de descodificação e descompressão.

Devido ao barramento de dados da EPROM ser de 8 bits e as amostras serem de 14 bits, foi necessário dividir a capacidade da EPROM em 2 partes: LSB (bits menos significativos) e MSB (bits mais significativos). Assim para se obter uma amostra são necessárias duas leituras sucessivas da EPROM.

A EPROM apresenta-se assim apenas como um bloco transparente onde, e de acordo com a norma J.41 do CCITT são implementadas as operações inversas às efectuadas no multiplexador.

Para gerar a tabela correspondentes foi mais uma vez realizado um programa em Pascal (ver anexo com a respectiva listagem).

Armazenamento e igualização do espaçamento entre as amostras

Na saída da EPROM temos as amostras de 14 bits divididas em dois blocos de 7, na versão 1, e 8+6 nas restantes versões, designados respectivamente de bloco mais significativo (MSB) e bloco menos significativo (LSB).

O facto de as amostras recebidas de um mesmo canal se apresentarem agrupadas duas a duas, leva a que no desmultiplexador seja efectuada uma operação de armazenamento que permita igualizar o seu espaçamento (1/32 KHz = 31.25 μs). É utilizada uma RAM (Read Access Memory), que permite a escrita sequencial das diferentes amostras descodificadas e descomprimidas, e a leitura posterior a uma taxa constante de 32 KHz.

Na versão 1, é efectuado o armazenamento de todas as amostras, sendo por isso necessário uma RAM com pelo menos 16 posições de memória (2 posições por amostra, 2 amostras por canal, 4 canais).

Nas versões subsequentes, apenas são armazenadas as segundas amostras de cada canal, sendo as primeiras recolhidas imediatamente, após a descodificação e descompressão da lei A, pelos *buffers* existentes no conversor D/A respectivo. O processo de recolha é comandado pela unidade de controlo do desmultiplexador, que gera para cada canal dois sinais, lsbX e msbX (X = A, B, C, D, E).

Lembrando que nestas duas últimas versões é efectuada a desmultiplexagem de 5 canais, a distribuição das posições das amostras na RAM é efectuada de acordo com a tabela 4.2-4:

Tabela 4.2-4

	VERSÃO 1		VERSÕES 2 e 3		
Posição	Canal/Amostra	Palavra	Canal/Amostra	Palavra	Valor em Hexadecimal
0000	A1	LSB	A2	LSB	0
0001	A2	LSB	A2	MSB	1
0010	B1	LSB	B2	LSB	2
0011	B2	LSB	B2	MSB	3
0100	C1	LSB	C2	LSB	4
0101	C2	LSB	C2	MSB	5
0110	D1	LSB	D2	LSB	6
0111	D2	LSB	D2	MSB	7
1000	A1	MSB	E2	LSB	8
1001	A2	MSB	E2	MSB	9
1010	B1	MSB	—	—	A
1011	B2	MSB	—	—	B
1100	C1	MSB	—	—	C
1101	C2	MSB	—	—	D
1110	D1	MSB	—	—	E
1111	D2	MSB	—	—	F

Os endereços, sinal de *enable* e sinal de leitura/escrita na memória são gerados na respectiva unidade de controlo do sistema.

Interface com os conversores D/A - versão 1

Apenas na versão inicial é efectuada, no desmultiplexador, a interface de comunicação com os conversores D/A. Nas restantes versões, o armazenamento é efectuado a jusante, nos *buffers* de cada conversor, existindo apenas um barramento de comunicação controlado pelo desmultiplexador.

Este último bloco tem como finalidade preparar as amostras dos diferentes canais para serem entregues aos conversores D/A.

Como já foi anteriormente referido usou-se um barramento multiplexado respectivamente para os canais A+C e B+D.

As amostras vão sendo armazenadas em registos, gerando a unidade de controlo 4 sinais de captura das amostras (dois para o par A C e outros dois para o par B D), e outros dois sinais (convAB para o par A+B e convCD para o par C+D) com a duração correspondente a um bit, indicando ao conversor D/A os instantes de início de conversão, e garantindo ao mesmo tempo um espaçamento constante, entre amostras do mesmo canal, a uma taxa de 32 KHz.

A figura 4.2-24 ilustra a técnica utilizada:

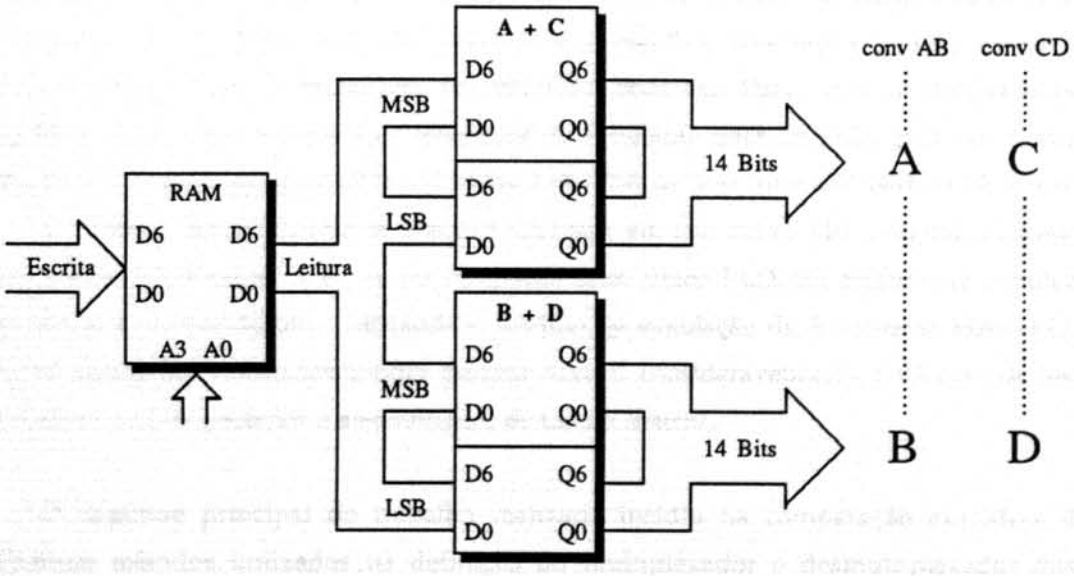


Fig. 4.2-24 Interface de comunicação com os conversores D/A (versão 1).

Unidade de controlo

As unidades de controlo das diferentes versões foram implementadas de maneira idêntica às suas homónimas do multiplexador (ver Anexos). Evitamos portanto repetir as descrições já efectuadas, reportando para o capítulo seguinte as comparações que necessariamente terão de ser efectuadas, e a especificação dos diferentes sinais de controlo por elas gerados.

4.3. ANÁLISE ESTATÍSTICA DOS RESULTADOS OBTIDOS

A funcionalidade global do sistema foi verificada com o auxílio de instrumentação de teste e medida, nomeadamente osciloscópio, analisador lógico, analisador de espectros e analisador de transmissão PCM.

O teste explícito dos diferentes protótipos reflecte o índice tecnológico das diferentes versões.

Na versão inicial, dominada pela utilização de lógica discreta, realizou-se inicialmente uma placa de teste, em Wire-Wrap. Este método facilitou a introdução de alterações revelando-se, no entanto, extremamente moroso e de fiabilidade duvidosa.

Na versão 2, a ferramenta de desenvolvimento do PLD utilizado, APLUS, permitiu efectuar a simulação funcional de todo o bloco lógico integrado. A geração de vectores de teste é extremamente simples e a visualização dos resultados em forma tabular, ou utilizando o editor de formas de onda, permite obter resultados extremamente fiáveis. A integração do PLD no resto do sistema, e respectiva simulação global, apesar da disponibilidade de uma ferramenta de CAD/CAE poderosa, Daisy, não foi efectivada pela ausência de modelos funcionais e temporais de simulação quer do PLD, quer dos circuitos integrados de aplicação específica utilizados, PEB 2030 da Siemens e MV 1441 da Motorola.

A versão 3, tecnológicamente a mais evoluida, e em que todo o bloco digital, à excepção dos blocos respeitantes à lei A, foi integrado num único PLD, foi totalmente simulada, funcional e temporalmente, utilizando o módulo de simulação da ferramenta MAX+PLUS. A fiabilidade dos resultados obtidos permite reduzir consideravelmente o número de testes a realizar na fase posterior à assemblagem de todo o sistema.

O objectivo principal do trabalho realizado incidiu na comparação exaustiva dos diversos métodos utilizados na definição do multiplexador e desmultiplexador das 3 versões operacionais do SITAD. Terminada a descrição das diferentes versões resta-nos compilar e resumir os resultados obtidos.

4.3.1. Recursos necessários à implementação das diferentes versões

Multiplexador

Quadro 4.3-1

Multiplexador	Versão 1	Versão 2	Versão 3
Conversão HDB3->NRZ	—	MV1441	EPM5128J
Recuperação de relógio	—	NE564 74F74	NE564 74F74
Detecção do alinhamento de trama	—	PEB 2030	EPM5128J
Codificação Lei A	8 x 74LS374 EPROM 27128 EPROM 27256 74LS173 74LS368	EPROM 27512 74LS173	EPROM 27512 EPM5128J
Geração da palavra de sincronismo	74LS244 74LS107	74LS541 EP1810J	EPM5128J
Geração de relógio	Relógio externo	CRISTAL 8.192MHZ 74LS04 EP1810J	CRISTAL 8.192MHZ 74LS04 EPM5128J
Geração da trama	74LS166	EP1810J	EPM5128J
Unidade de controlo	2 x 74LS161 PAL 16L8 PROM 28L22	EP1810J	EPM5128J
Conversão NRZ->HDB3	PAL 16R6	MV1441	EPM5128J
Nº Total de CIs	20	10	6
Consumo (mA)	720	360	380
Placa de circuito impresso	3U - Eurocard 2 camadas de sinal Furo não metalizado	3U - Eurocard 2 camadas c/ plano de massa Furo metalizado	3U - Eurocard 1 camada sinal + 1 camada plano de massa Furo metalizado

Desmultiplexador

Quadro 4.3-2

Desmultiplexador	Versão 1	Versão 2	Versão 3
Conversão HDB3->NRZ	MV1441	MV1441	EPM5128J
Recuperação de relógio	Cristal 16.384 MHz	NE564 74F74	NE564 74F74
Detecção do alinhamento de trama	PEB2030	PEB2030	EPM5128J
Descodificação da lei A	2 x 74LS595 EPROM 27128	2 x 74LS595 EPROM 2764	EPM5128J EPROM 2764
Verificação da paridade	—	—	EPM5128J
Igualização das amostras	RAM 6116 8 X 74LS374	RAM 6116	RAM 6116
Unidade de controlo	74LS00 2 X 74LS169 PAL 16R8 PROM 28L22	EP1810J	EPM5128J
Nº Total de CIs	20	9	5
Consumo (mA)	590	360	210
Placa de circuito impresso	3U - Eurocard 2 camadas de sinal Furo não metalizado	3U - Eurocard 2 camadas c/ plano de massa Furo metalizado	3U - Eurocard 1 camada sinal + 1 camada plano de massa Furo metalizado

4.3.2. Sinais gerados por cada Unidade de controlo

Multiplexador

Quadro 4.3-3

	Sinais gerados pelas diferentes Unidades de controlo do Multiplexador	Nº de sinais
Versão 1	ckA, ckB, ckC, ckD, convA/B, convC/D, enEX, enTR, enA, enB, enC, enD, OE1, OE2, A14/ck3, par_AD	16
Versão 2	ckA, ckB, ckC, ckD, ckE, ck173, ck256K, ck8K, ck4K, enA, enB, enC, enD, enE, cePEB, ceROM, en541	17
Versão 3	ckA, ckB, ckC, ckD, ckE, enA, enB, enC, enD, enE, ceROM	11

Desmultiplexador

Quadro 4.3-4

	Sinais gerados pelas diferentes Unidades de controlo do Desmultiplexador	Nº de sinais
Versão 1	lsbAC, lsbBD, msbAC, msbBD, convA/B, convC/D, ram0, ram1, ram2, ram3, R/W, CE, open, A13	14
Versão 2	lsbA, lsbB, lsbC, lsD, lsE, msbA, msbB, msbC, msbD, msbE, ram0, ram1, ram2, ram3, R/W, CEram, CErom, open, a12, ck4K	20
Versão 3	lsbA, lsbB, lsbC, lsD, lsE, msbA, msbB, msbC, msbD, msbE, ram0, ram1, ram2, ram3, R/W, CEram, CErom, a12	18

4.4. FOTOGRAFIAS DOS PROTOTIPOS

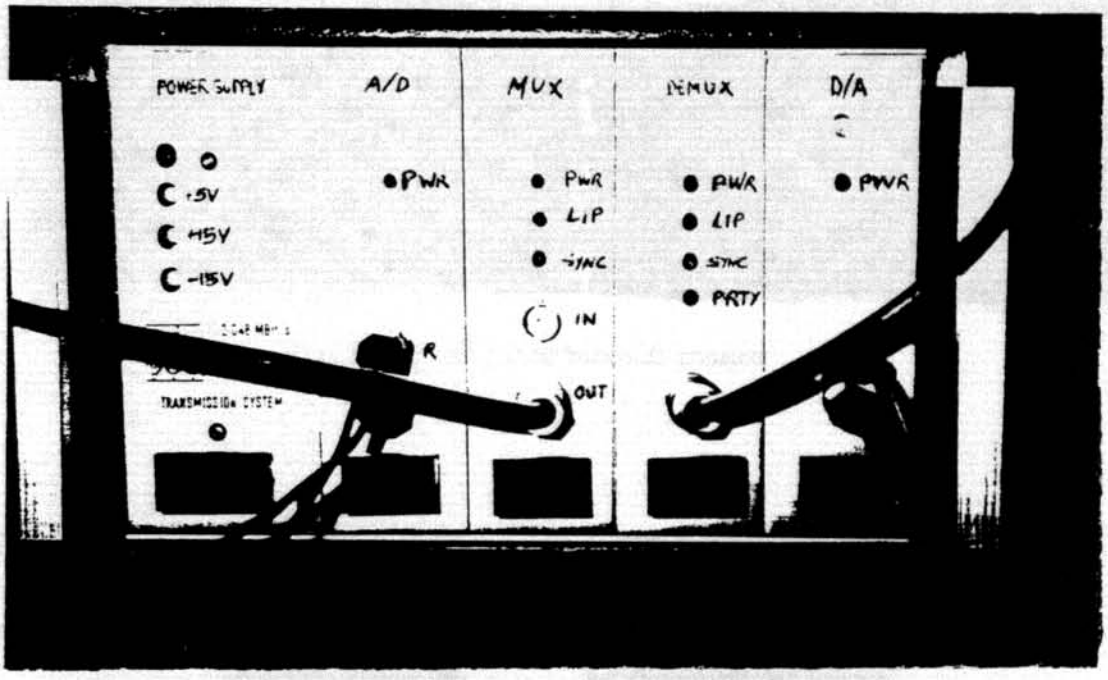


Fig. 4.4-1 SITAD.

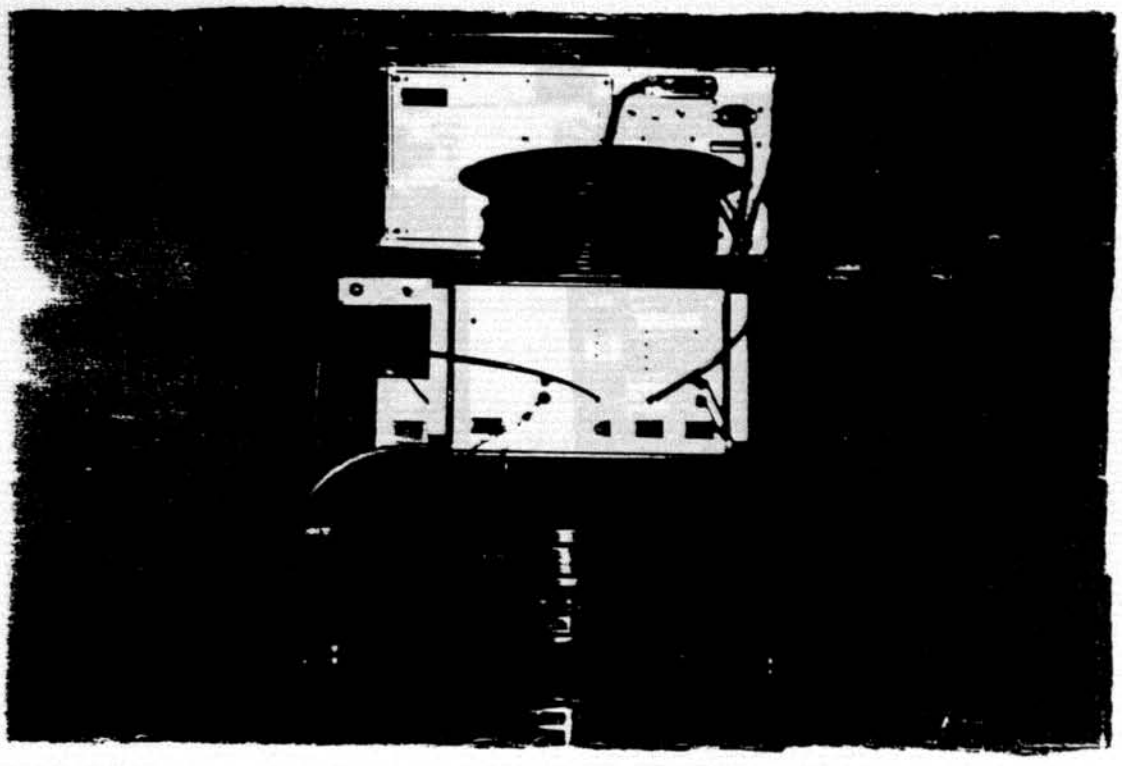


Fig. 4.4-2 O SITAD no banco de ensaios.

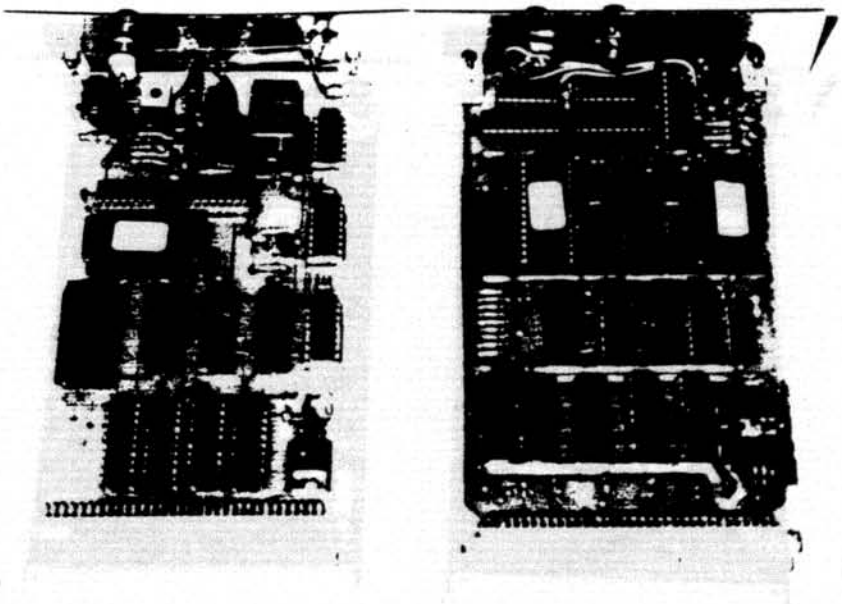


Fig. 4.4-3 Multiplexador e desmultiplexador (versão 1).

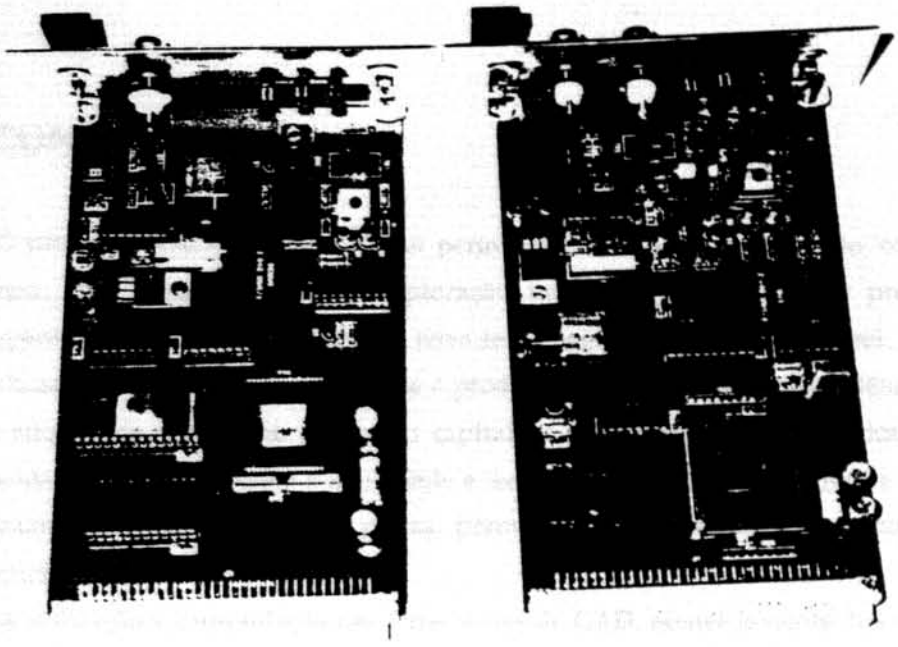


Fig. 4.4-4 Multiplexador e desmultiplexador (versão 2).

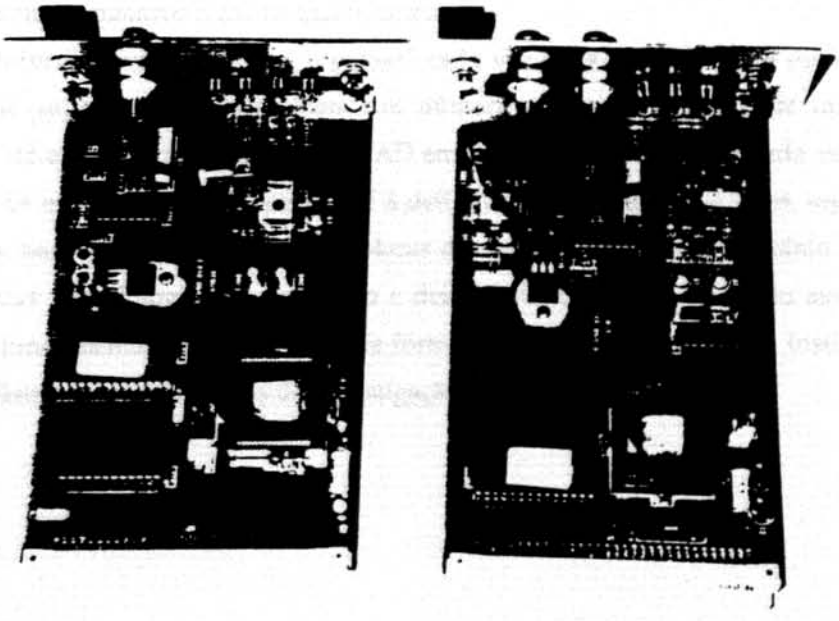


Fig. 4.4-5 Multiplexador e desmultiplexador (versão 3).

5. CONCLUSÕES

O projecto SITAD, para além de ter permitido desenvolver um produto, concluido com sucesso, com possibilidades de exploração industrial, foi ainda um projecto piloto indispensável para o domínio de uma nova tecnologia - a lógica programável.

Pensamos que mais importante que o produto em si, tenha sido a experiência e o *Know-how* adquiridos. Conforme se viu no capítulo 4 a utilização destas técnicas possibilita aumentar considerávelmente a fiabilidade e testabilidade dos produtos, reduz o número de componentes e a dimensão das placas, permitindo em simultâneo a melhoria das suas características técnicas.

A utilização e consolidação das ferramentas de CAD, nomeadamente dos seus módulos de simulação, reduz a necessidade de implementação de protótipos intermédios, facilitando a fase de projecto.

Para além deste, vários outros projectos têm sido desenvolvidos, usufruindo numa escala crescente da experiência e Know-how do departamento de CAD do INESC Porto, em que o autor se encontra integrado. Parece assim indispensável continuar a investir nestes domínios e a promover a transferência de tecnologia para a indústria, através de projectos e da formação de quadros técnicos qualificados.

No futuro, a evolução passa, e passará cada vez mais, pelo domínio de ferramentas de CAD que suportem e automatizem um número crescente de fases de um projecto. O espectro de aplicação das técnicas de CAD em Electrónica cobre áreas cada vez mais vastas que vão da especificação do sistema, até à definição de circuitos integrados, sejam eles ASICs ou PLDs, passando pelo desenho das placas de circuito impresso. O domínio de técnicas de CAD e das metodologias de utilização e desenvolvimento que lhe estão associadas é um aspecto fundamental que condiciona de forma importante o sucesso das instituições, sejam elas indústria ou organizações de investigação e desenvolvimento.

GLOSSÁRIO

Ajuda on-line - Refere-se a uma situação de ajuda constante na utilização, neste caso, de uma ferramenta computacional de desenvolvimento. O utilizador deverá ter acesso, sempre que necessário, a menus explicativos das diversas formas de proceder, e a avisos de situações ilegais que tenha provocado, e que impedirão o desenrolar correcto de acções posteriores.

Back annotation - Retorno actualizado de informação num processo interactivo e complementar de desenvolvimento.

Bed of nails - Matriz de pernos, que permite aceder a um grande número de nós, e excitar/observar o seu comportamento.

Buffer - Uma estrutura electrónica destinada ao armazenamento intermédio de dados. Um dispositivo para isolamento de sinais em/entre sistemas.

CAD - Projecto assistido por computador. Refere-se a todas as ferramentas computacionais auxiliares de projecto.

CAE - Engenharia assistida por computador. Refere-se a todas as ferramentas computacionais de auxílio ao desenvolvimento de projectos de engenharia.

Combinacional - Uma operação lógica cuja saída é função directa de um conjunto de varáveis de entrada, i. e., não dependente de uma referência temporal.

CrossBar - Matriz de cruzamentos utilizada em telecomunicações. O seu funcionamento baseia-se na interligação configurável de linhas horizontais e verticais.

Crosstalk - Interferência na recepção causada por sinais indesejados.

Custos NRE - São custos não contabilizados de desenvolvimento de um determinado projecto. A razão reside na distribuição dos custos das diferentes tarefas a executar, pelas

múltiplas entidades envolvidas na concretização do projecto.
Inclui também, no caso da definição de um circuito integrado do tipo Gate array ou Standard cell, o tempo (tempo é dinheiro), pós desenvolvimento, consumido na fabricação do circuito integrado e o consequente atraso na colocação do produto nos circuitos comerciais, com os respectivos custos.

Driver - Um dispositivo para aumentar o fan-out dos sinais que o alimentam.

Duty cycle - Valor percentual que traduz a relação entre a duração temporal dos níveis lógicos de uma onda periódica.

EDA - Automatização do projecto electrónico. Refere as ferramentas computacionais de auxílio ao desenvolvimento de projectos na área da electrónica.

Fan-in - valor mínimo de corrente requerido na entrada de um dispositivo, para que um determinado sinal seja considerado válido.

Fan-out - valor máximo de corrente que um determinado dispositivo é capaz de fornecer na sua saída.

Gate Arrays - São circuitos integrados digitais semi-definidos durante o processo fabrico e configurados à posteriori pelo utilizador que define, numa ou duas camadas de metal, as interligações entre os diversos elementos lógicos (transistores). O termo semi-definido significa que o mesmo circuito integrado pode ser configurado para realizar diferentes funções.

Glitch - Perturbação de curta duração, no nível lógico de um sinal digital.

Layout - Distribuição física dos elementos lógicos constituintes de um circuito digital, seja ele uma placa de circuito impresso ou um circuito integrado. No primeiro caso referir-se-à à distribuição dos circuitos integrados na placa, enquanto no segundo focará a distribuição de portas lógicas, primitivas ou macros no circuito integrado.

Load - Acção de recolha de um sinal, ou conjunto de sinais, colocados na entrada de um sistema ou de um circuito integrado. A recolha pode ser síncrona, i. e., accionada pela transição activa de um sinal de relógio, ou assíncrona, i. e., efectuada quando o sinal encarregado de autorizar a recolha estiver num determinado nível lógico.

Macro - São padrões pré-definidos de interligações entre elementos lógicos que realizam determinda função lógica, podendo ser repetidamente utilizados.

Minitermo - Termos de produto fundamentais, i. e. , a função mais elementar de saída, resultado da combinação de todas as variáveis de entrada.

Routing - Encaminhamento físico de ligações eléctricas.

Scan - Varrer em sequência.

Standard Cell - A abordagem standard cell na definição de circuitos integrados involve a interligação de macros armezenados numa base de dados para a realização de um circuito integrado de aplicação específica. A diferença fundamental relativamente aos Gate arrays reside na necessidade de definição, por parte do utilizador, das diversas máscaras indispensáveis ao fabrico de um circuito integrado.

Stuck-at - Situação em que um determinado sinal digital fica fixo num determinado valor lógico. Pode ocorrer por má concepção do sistema, ou mais vulgarmente pela existência de situações anómalas que só o teste exaustivo ajudará a soluccionar.

Termo de produto - Resultado do AND lógico de duas ou mais variáveis, i. e., entradas lógicas.

Trigger - Refere a sincronização de determinado sinal por um outro sinal designado de referência. O sinal afectado poderá ser modificado em relação à sua forma original.

REFERÊNCIAS

Actel, 1990. ACT™ 1 - Field Programmable Gate Arrays. Actel Corporation, 1990.

Altera, 1988. Altera Data Book. Altera Corporation, September 1988.

Altera, 1990a. Altera Data Book. Altera Corporation, October 1990.

Altera, 1990b. MAX+PLUS Development System. Altera Corporation, April 1990.

Altera, 1990c. The MAXimalist Handbook. Altera Corporation, January 1990.

Altera, 1991a. Altera Data Book. Altera Corporation, September 1991.

Altera, 1991b. Applications Engineering News and Views - A newsletter for Altera Customers. Altera Corporation, April 1991.

Alves, A. Pimenta 1989. O Projecto SIFO: Situação Actual e Perspectivas Futuras. Actas do ENDIEL 89 (ST3), Abril de 1989, pp. 1-11.

AMD, 1990a. MACH™ Devices - High Density EE CMOS Programmable Logic Data Book. Advanced Micro Devices, Inc., February 1990.

AMD, 1990b. PAL® Device Data Book - CMOS. Advanced Micro Devices, Inc., January 1990.

AMD, 1991. MACH™ Technical Briefs. Advanced Micro Devices, Inc., September 1991.

Beacher, R. 1991. Hardware description language and PLDs generate waveforms. EDN, May 23, 1991, pp. 131-142.

Beenker, F. 1985. Systematic and Structured Methods for Digital Board Testing. IEEE International Test Conference Proceedings, 1985, pp. 380-385.

Bennetts, R. G. and Osseyran, A. 1991. IEEE Standard 1149.1-1990 on Boundary-Scan: History, Literature Survey, and Current Status. Journal of Electronic Testing: Theory and Applications. March 1991, Vol.2, Nº1, pp. 11-25.

Bowns, T. 1991. Control Metastability in high-speed CMOS circuits. Electronic Design, September 26, 1991, pp. 74-80.

Bursky, D. 1987. Faster, more complex PLDs arrive with better programming tools. Electronic Design, April 1987, pp. 13-25.

Bursky, D. 1991a. Combination RAM/PLD opens new application options. Electronic Design, May 23, 1991, pp. 138-140.

Bursky, D. 1991b. High-density programmable logic takes on Gate Arrays. Electronic Design, March 14, 1991, pp. 45-57.

Cabral, J. M. 1990. Projecto SIFO: Subsistema de Audio Digital - Relatório Técnico Final. Inesc Norte, Maio de 1990.

Cardoso, P. B. 1990. Sistemas Multiplexador e Desmultiplexador para Áudio Digital. Comunicação privada (Inesc Porto), 1990.

CCITT, 1985. Red Book - Conjunto das Normas das séries G e J. CCITT, Geneve, 1985.

DasGupta, S., Graf, M. C., Rasmussen, R. A., Walther, RG. and Williams, TW. 1984. Chip Partioning Aid: A Design Technique for Partitionability and Testability in VLSI. Design Automation Conference Proceedings, 1984, pp. 203-208.

DATA I/O, 1986. Programmable Logic - A Basic Guide for the Designer. DATA I/O Corporation, 1986.

DATA I/O, 1989. ABEL™ 3.1. DATA I/O Corporation, January 1989.

DATA I/O, 1990. PIC Technologies and Tools - A guide to design, verification, programming, and testing tools for programmable integrated circuits. DATA I/O Corporation, 1990.

Decastro, J., Hoogerhuis, P. 1991. Concurrent Engineering meets design automation. Electronic Design, January 10, 1991, pp. 73-82.

Dingman, S. 1991. Determine PLD metastability to derive ample MTBFs. EDN, August 5, 1991, pp. 147-157.

Donnell, J. 1991. Boundary Scan Puts Tomorrow´s Devices To Test. Electronic Design, June 27, 1991, pp. 75-86.

Eichelberger, E. and Williams, T. 1978. A Logic Design Structure for LSI Testability. Journal of Design Automation and Fault Tolerant Computing, Vol. 2, Nº2, May 1978, pp. 165-178.

Feldman, R., Rosky, D. 1991. A step-by-step guide to Programmable Delays. EDN, June 13, 1991, pp. 97-105.

Fenton, J. 1991. Don't take your ASIC prototypes for granted. Electronic Design, August 8, 1991, pp. 75-78.

Ferreira, Aníbal. J. S. 1990. Codificador e Descodificador para Áudio de Alta Qualidade. Comunicação privada (Inesc Porto), Ri-10-90, 1990.

Gallant, J. 1991. Design demands technology tradeoffs. EDN, August 19, 1991, pp. 77-88.

Gosh, J. 1991. BiCMOS Array speeds communications design. Electronic Design, August 8, 1991, pp. 117-120.

Horstmann, J. U., Eichel, H. W., Coates, R. L. 1989. Metastability Behavior of CMOS ASIC Flip-Flops in Theory and Test. IEEE Journal of Solid-State Circuits, February 1989, Vol. 24, Nº 1, pp. 146-157.

IEEE Std 1076. 1987. IEEE Standard VHDL LAnguage Reference Manual. IEEE Standards Board, March 1988.

Inocêncio, E. M., Castro. R. M. 1988. Multiplexador de audio digital a 2048 KBit/s. Comunicação privada (Inesc Norte), Ri-126-88, 1988.

Intel, 1987. Programmable Logic Handbook. Intel Corporation, 1987. ISBN 1-55512-028-8.

Jackson, R. M. 1991. Patience and reason solve digital-system debugging problems. EDN, May 9, 1991, pp.135-140.

Jensen, Q. C. 1991. Build asynchronous state-machines that capitalize on PLDs. EDN, May 23, 1991, pp. 121-124.

Knapp, S. K. 1990, Accelarate FPGA Macros with ONE-HOT approach. Electronic Design, September 13, 1990, pp. 65-71.

Kopec, S. 1988a. Asynchronous state machines challenge digital designers. EDN, June 9, 1988, pp. 179-186.

Kopec, S. 1988b. State machines solve control-sequence problems. EDN, May 26, 1988, pp. 177-188.

Leibson, S. H. 1989. EDN Special Report. EDN, March 16, 1989, pp. 111-124.

Leibson, S. H. 1990. PLD development software. EDN, August 2, 1990, pp. 100-116.

Leitão, M. J., Teixeira, J. M. F., Cabral, J. M., Cardoso, P. B. 1989. Transmissão de audio digital na rede pública. Actas do ENDIEL 89 (ST3), Abril de 1989, pp. 317-320.

Lipset, R., Schaefer, C. F., Ussery, C. 1990. VHDL: Hardware Description and Design. Kluwer Academic Publishers. ISBN 0-7923-9030-X

Maliniak, L. 1990. PLD Design Tools Mature. Electronic Design, October 11, 1990, pp. 55-66.

Maliniak, L. 1991a. Design automation takes over more tasks early on. Electronic Design, June 13, 1991, pp. 47-63.

Maliniak, L. 1991b. Optimize and Retarget existing logic designs. Electronic Design, May 23, 1991, pp. 135-136.

Maliniak, L. 1991c. Synthesis tools complete front-to-back EDA system. Electronic Design, February 28, 1991, pp. 91-93.

Maliniak, L. 1991d. Teamwork is the key to Concurrent Engineering. Electronic Design, January 10, 1991, pp. 37-50.

Marino, L. R. 1981. General Theory of Metastable Operation. IEEE Transactions on Computer, February 1981, Vol. C-30, Nº2, pp. 107-115.

Markowitz, M. C. 1989a. Logic synthesis prepares for VHDL. EDN, March 30, 1989, pp. 51-62.

Markowitz, M. C. 1989b. VHDL tools survive reality check. EDN, December 20, 1990, pp. 54-62.

Markowitz, M. C. 1990. Software adds logic to make designs testable. EDN, October 11, 1990, pp. 59-70.

Markowitz, M. C. 1991. Concurrent Engineering journey starts with the first step. EDN, July 18, 1991, pp. 110-114.

Marshall, T. 1990. Creating Custom chips. BYTE, January 1990, pp 271-283.

Masteller, S. R. 1991. Design a digital synchronizer with a low metastability-failure rate. EDN, April 25, 1991, pp. 169-174.

Matos, J. S., Ferreira, J. M. 1989. SEMINÁRIO - O Teste de Sistemas Electrónicos: Introdução a Metodologias de Projecto Testável. Faculdade de Engenharia, 24 de Outubro de 1989.

Maunder, C. and Tuloss, R. E. 1991. An Introduction to the Boundary Scan Standard: ANSI/IEEE Std 1149.1. Journal of Electronic Testing: Theory and Applications. March 1991, Vol.2, Nº1, pp. 27-42.

McCluskey, E. J. 1986. Logic Design Principles: with emphasis on testable semicustom circuits. Prentice-Hall International. ISBN 0-13-539768-5.

Milne, B. 1989. Robust Tools tackle latest PLDs. Electronic Design - International, May 1989, pp. 55-60.

MMI, 1987. PAL® Device Handbook. MMI - A wholly Owned Subsidiary of Advanced Micro Devices, Inc., 1987.

Novellino, J. 1990. Boundary Scan Holds Court at ITC'90 - Technical Papers On IEEE-1149,1 And A Meeting With Its Authors Highlight The Show. Electronic Design, August 23, 1990, pp. 34-37.

Novellino, J. 1991. Design Teams evolve to face Testability - More teamwork and new tools will help meet the challenge presented by complex systems. Electronic Design, January 10, 1991, pp.52-70.

Paradise, J. P. 1991. Logic-synthesis tools take the tedium out of logic design, EDN, September 2, 1991, pp.147-154.

Parker, K. 1989. The Impact of Boundary Scan on Board Test. IEEE Design and Test of Computers, August. 1989, pp. 18-30.

Plessey, 1988. Product brief - ERA - Electrically Reconfigurable Arrays. Plessey Semiconductors, November 1988.

Plessey, 1989. ERA 60100, Electrically Reconfigurable Arrays - ERA. Plessey Semiconductors, November 1988.

Pradhan, M. M., Tulloss, R. E., Bleeker, H. and Beenker, F. P. 1987. Developing a Standard for Boundary Scan Implementation. IEEE International Conference on Computer Design: VLSI in Computers and Processors, 1987, pp. 462-466.

Quinell, R. A. 1990a. FPGA family offers speed, density, on-chip RAM, and wide-decode logic. EDN, December 6, 1990, pp. 62-64.

Quinell, R. A. 1990b. JTAG Boundary-Scan Test - Adding testability also aids debugging. EDN, August 2, 1990, pp. 67-74.

Quinnell, R. A. 1991. Synthesis tools speed PLD design efforts. EDN, April 11, 1991, pp. 73-80.

Rabinovich, R. 1991. State-machine design curbs illegal states and transitions. EDN, February 4, 1991, pp. 95-100.

Rec G.703, 1985. Red Book. CCITT, Vol. III - Fasc. III.3, Geneve 1985.

Rec G.704, 1985. Red Book. CCITT, Vol. III - Fasc. III.3, Geneve 1985.

Rec G.735, 1985. Red Book. CCITT, Vol. III - Fasc. III.3, Geneve 1985.

Rec G.736, 1985. Red Book. CCITT, Vol. III - Fasc. III.3, Geneve 1985.

Rec G.737, 1985. Red Book. CCITT, Vol. III - Fasc. III.3, Geneve 1985.

Rec G.823, 1985. Red Book. CCITT, Vol. III - Fasc. III.3, Geneve 1985.

Rec J.41, 1985. Red Book. CCITT, Vol. III - Fasc. III.3, Geneve 1985.

Rec J.42, 1985. Red Book. CCITT, Vol. III - Fasc. III.3, Geneve 1985.

Ryherd, E. 1991. Don't get skewed on your next ASIC design. EDN, September 2, 1991, pp. 139-144.

Sandige, R. S. 1990. Modern Digital Design. McGraw-Hill. ISBN 0-07-100933-7.

Schulze, B. 1991. Avoid pitfalls in selecting the right programmable-logic design tools. Electronic Design, November 7, 1991, pp. 71-81.

SIFO, 1989. Projecto SIFO: Demonstração Laboratorial - Relatório Técnico. Inesc Norte, 28 de Abril de 1989.

Signetics, 1984. Linear LSI Data and Applications Manual 1985. Signetics Corporation, 1984.

Small, C. H. 1987. Programmable -logic devices. EDN, February 5, 1987, pp. 112-133.

Small, C. H. 1988. Programmable Logic Devices. EDN, November 10, 1988, pp 142-156.

Small, C. H. 1991a. Family tree sorts out high-density PLDs. EDN, September 16, 1991, pp. 75-80.

Small, C. H. 1991b. FPGA Design methods. EDN, August 5, 1991, pp. 114-122.

Smith, D. 1986. Special Report - CMOS PLDs. EDN, May 15, 1986, pp. 95-109.

Thomas, D. E., Moorby, P., 1991. The Verilog® Hardware Description Language. Kluwer Academic Publishers. ISBN 0-7923-9126-8

TI, 1984. The TTL Data Book, Texas Instruments, Inc., Vol. 3, 1984.

TI, 1988. Programmable Logic Development System - A+Plus. Texas Instruments, Inc., October 1988.

TI, 1989. EP1810 High-performance 48-macrocell Erasable Programmable Logic Device (EPLD). Texas Instruments, Inc., February 1989.

Tralka, C. 1990. Avoid design pitfalls when working with PLDs. Electronic Design, October 11, 1990, pp. 75-78.

Xilinx, 1987. The Programmable Gate Array Design Handbook. Xilinx, inc., 1987.

ANEXOS

I. VERSÃO 2

Unidade de controlo do multiplexador

PEDRO CARDOSO - INESC NORTE - DEZEMBRO, 1990 MUX - MAQ. ESTADOS

PART: EP1810J OPTIONS: TURBO = OFF, SECURITY = ON

INPUTS:
OUTPUTS:S0@10, S1@11, S2@12, S3@13, S4@24, S5@25, S6@26
NETWORK:

MACHINE: MUX CLOCK: CLK CLEAR: RESN

STATES:	[s6	s5	s4	s3	s2	s1	s0]
ST0	[0	0	0	0	0	0	0]
ST1	[0	0	0	0	0	0	1]

.....

ST62	[0	1	1	1	1	1	0]
ST63	[0	1	1	1	1	1	1]
ST64	[1	0	0	0	0	0	0]

.....

ST127	[1	1	1	1	1	1	1]

ST0: ST1	OUTPUTS: SCK256	
ST1: ST2	OUTPUTS: SCE244 SCEPEB	
ST2: ST3		
ST3: ST4		
ST4: ST5	OUTPUTS: RCKE	
ST5: ST6	OUTPUTS: RENA	
ST6: ST7	OUTPUTS: SCKE RA14 RA15	IF !SELECT THEN RCEROM
ST7: ST8	OUTPUTS: RCK256 S1CK173	IF SELECT THEN RCEPEB
ST8: ST9	OUTPUTS: R1CK173 SCK256 SA14	
ST9: ST10	OUTPUTS: S1CK173 SCEPEB	
ST10: ST11	OUTPUTS: R1CK173 SENA	
ST11: ST12		
ST12: ST13		
ST13: ST14		
ST14: ST15	OUTPUTS: RENA	
ST15: ST16	OUTPUTS: RCK256	IF SELECT THEN RCEPEB
ST16: ST17	OUTPUTS: SENA SCK256 SA15	
ST17: ST18	OUTPUTS: SCEPEB	
ST18: ST19		
ST19: ST20	OUTPUTS: S1CK173	
ST20: ST21	OUTPUTS: RCKC R1CK173	
ST21: ST22		
ST22: ST23	OUTPUTS: RENA SCKC	
ST23: ST24	OUTPUTS: RCK256	IF SELECT THEN RCEPEB
ST24: ST25	OUTPUTS: SENA SCK256	
ST25: ST26	OUTPUTS: SCEPEB SCEROM	
ST26: ST27		
ST27: ST28		
ST28: ST29		
ST29: ST30	OUTPUTS: RENB	
ST30: ST31	OUTPUTS: RA14 RA15	IF !SELECT THEN RCEROM
ST31: ST32	OUTPUTS: RCK256 S1CK173	IF SELECT THEN RCEPEB
ST32: ST33	OUTPUTS: R1CK173 SCK256 SA14	
ST33: ST34	OUTPUTS: S1CK173 SCEPEB	

```
ST34: ST35  OUTPUTS: R1CK173 SENB
ST35: ST36
ST36: ST37  OUTPUTS: RCKA
ST37: ST38
ST38: ST39  OUTPUTS: RENB SCKA
ST39: ST40  OUTPUTS: RCK256                          IF SELECT THEN RCEPEB
ST40: ST41  OUTPUTS: SENB SCK256 SA15
ST41: ST42  OUTPUTS: SCEPEB
ST42: ST43
ST43: ST44  OUTPUTS: S1CK173
ST44: ST45  OUTPUTS: RCKD R1CK173
ST45: ST46
ST46: ST47  OUTPUTS: RENB SCKD
ST47: ST48  OUTPUTS: RCK256                          IF SELECT THEN RCEPEB
ST48: ST49  OUTPUTS: SENB SCK256
ST49: ST50  OUTPUTS: SCEPEB  SCEROM
ST50: ST51
ST51: ST52
ST52: ST53
ST53: ST54  OUTPUTS: RENC
ST54: ST55  OUTPUTS: RA14 RA15                       IF !SELECT THEN RCEROM
ST55: ST56  OUTPUTS: RCK256 S2CK173                  IF SELECT THEN RCEPEB
ST56: ST57  OUTPUTS: R2CK173 SCK256 SA14
ST57: ST58  OUTPUTS: S2CK173 SCEPEB
ST58: ST59  OUTPUTS: R2CK173 SENC
ST59: ST60
ST60: ST61  OUTPUTS: RCKB
ST61: ST62
ST62: ST63  OUTPUTS: RENC SCKB
ST63: ST64  OUTPUTS: RCK256                          IF SELECT THEN RCEPEB
ST64: ST65  OUTPUTS: SENC SCK256 SA15
ST65: ST66  OUTPUTS: SCEPEB
ST66: ST67
ST67: ST68  OUTPUTS: S2CK173
ST68: ST69  OUTPUTS: RCKE R2CK173
ST69: ST70
ST70: ST71  OUTPUTS: RENC SCKE
ST71: ST72  OUTPUTS: RCK256                          IF SELECT THEN RCEPEB
ST72: ST73  OUTPUTS: SENC SCK256
ST73: ST74  OUTPUTS: SCEPEB SCEROM
ST74: ST75
ST75: ST76
ST76: ST77
ST77: ST78  OUTPUTS: REND
ST78: ST79  OUTPUTS: RA14 RA15                       IF !SELECT THEN RCEROM
ST79: ST80  OUTPUTS: RCK256 S2CK173                  IF SELECT THEN RCEPEB
ST80: ST81  OUTPUTS: R2CK173 SCK256 SA14
ST81: ST82  OUTPUTS: S2CK173 SCEPEB
ST82: ST83  OUTPUTS: R2CK173 SEND
ST83: ST84
ST84: ST85  OUTPUTS: RCKC
ST85: ST86
ST86: ST87  OUTPUTS: REND SCKC
ST87: ST88  OUTPUTS: RCK256                          IF SELECT THEN RCEPEB
ST88: ST89  OUTPUTS: SEND SCK256 SA15
ST89: ST90  OUTPUTS: SCEPEB
ST90: ST91
ST91: ST92  OUTPUTS: S2CK173
ST92: ST93  OUTPUTS: R2CK173
ST93: ST94
ST94: ST95  OUTPUTS: REND
ST95: ST96  OUTPUTS: RCK256                          IF SELECT THEN RCEPEB
ST96: ST97  OUTPUTS: SEND SCK256
ST97: ST98  OUTPUTS: SCEPEB SCEROM
ST98: ST99
ST99: ST100
ST100: ST101        OUTPUTS: RCKA
ST101: ST102        OUTPUTS: RENE
ST102: ST103        OUTPUTS: SCKA RA14 RA15          IF !SELECT THEN RCEROM
ST103: ST104        OUTPUTS: RCK256 S2CK173          IF SELECT THEN RCEPEB
```

```
ST104: ST105        OUTPUTS: R2CK173 SCK256 SA14
ST105: ST106        OUTPUTS: S2CK173 SCEPEB
ST106: ST107        OUTPUTS: R2CK173 SENE
ST107: ST108
ST108: ST109        OUTPUTS: RCKD
ST109: ST110
ST110: ST111        OUTPUTS: RENE SCKD
ST111: ST112        OUTPUTS: RCK256                    IF SELECT THEN RCEPEB
ST112: ST113        OUTPUTS: SENE SCK256 SA15
ST113: ST114        OUTPUTS: SCEPEB
ST114: ST115
ST115: ST116        OUTPUTS: S1CK173
ST116: ST117        OUTPUTS: R1CK173
ST117: ST118
ST118: ST119        OUTPUTS: RENE
ST119: ST120        OUTPUTS: RCK256                    IF SELECT THEN RCEPEB
ST120: ST121        OUTPUTS: SENE SCK256
ST121: ST122        OUTPUTS: SCEPEB SCEROM
ST122: ST123
ST123: ST124
ST124: ST125        OUTPUTS: RCKB
ST125: ST126
ST126: ST127        OUTPUTS: RCK16 SCKB                IF !SELECT THEN RCEPEB
ST127: ST0  OUTPUTS: RCK256 SCK16                  IF SELECT THEN RCE244

END$
```

Unidade de controlo do desmultiplexador

PEDRO CARDOSO - INESC NORTE - DEZEMBRO, 1990 DEMUX - MAQ. ESTADOS

PART: EP1810J OPTIONS: TURBO = OFF, SECURITY = ON

INPUTS:
OUTPUTS:S0@10, S1@11, S2@12, S3@13, S4@24, S5@25, S6@26
NETWORK:

MACHINE: MUX CLOCK: CLK CLEAR: RESN

```
STATES:  [s6  s5  s4  s3  s2  s1  s0]
ST0      [0   0   0   0   0   0   0]
ST1      [0   0   0   0   0   0   1]

 .....

ST62     [0   1   1   1   1   1   0]
ST63     [0   1   1   1   1   1   1]
ST64     [1   0   0   0   0   0   0]

 .....

ST127    [1   1   1   1   1   1   1]
```

```
ST0: ST1     OUTPUTS: SOPEN1 SCK16K
ST1: ST2     OUTPUTS: RRW1
ST2: ST3     OUTPUTS: SRW1
ST3: ST4     OUTPUTS: SA01 S1A12
ST4: ST5     OUTPUTS: RRW1
ST5: ST6     OUTPUTS: SRW1
ST6: ST7     OUTPUTS: RA3 SA2 RA01
ST7: ST8     OUTPUTS: RlsbC
ST8: ST9     OUTPUTS: SlsbC
ST9: ST10    OUTPUTS: SA01
ST10: ST11   OUTPUTS: RmsbC
ST11: ST12   OUTPUTS: SmsbC
ST12: ST13
ST13: ST14   OUTPUTS: RA12
ST14: ST15
ST15: ST16
ST16: ST17
ST17: ST18
ST18: ST19
ST19: ST20   OUTPUTS: ROPEN1
ST20: ST21   OUTPUTS: SOPEN1 RlsbA
ST21: ST22
ST22: ST23
ST23: ST24   OUTPUTS: SCERAM1
ST24: ST25   OUTPUTS: SlsbA
ST25: ST26   OUTPUTS: S1A12
ST26: ST27   OUTPUTS: RmsbA
ST27: ST28   OUTPUTS: SmsbA
ST28: ST29   OUTPUTS: RCERAM
ST29: ST30   OUTPUTS: RlsbD RA01 SA1 RA12
ST30: ST31   OUTPUTS: SlsbD
ST31: ST32   OUTPUTS: RmsbD SA01 ROPEN1
ST32: ST33   OUTPUTS: SmsbD SOPEN1
ST33: ST34   OUTPUTS: RRW1 RA2 RA1 RA01
```

ST34: ST35 OUTPUTS: SRW1
ST35: ST36 OUTPUTS: SA01 S1A12
ST36: ST37 OUTPUTS: RRW1
ST37: ST38 OUTPUTS: SRW1
ST38: ST39
ST39: ST40
ST40: ST41
ST41: ST42
ST42: ST43
ST43: ST44 OUTPUTS: ROPEN1
ST44: ST45 OUTPUTS: SOPEN1 RlsbB
ST45: ST46 OUTPUTS: RA12
ST46: ST47
ST47: ST48 OUTPUTS: SCERAM1
ST48: ST49 OUTPUTS: SlsbB
ST49: ST50 OUTPUTS: S1A12
ST50: ST51 OUTPUTS: RmsbB
ST51: ST52 OUTPUTS: SmsbB
ST52: ST53 OUTPUTS: RCERAM
ST53: ST54 OUTPUTS: RlsbE SA3 RA01
ST54: ST55 OUTPUTS: SlsbE
ST55: ST56 OUTPUTS: RmsbE SA01 ROPEN1 RA12
ST56: ST57 OUTPUTS: SmsbE SOPEN1
ST57: ST58 OUTPUTS: RRW1 RA3 SA1 RA01
ST58: ST59 OUTPUTS: SRW1
ST59: ST60 OUTPUTS: SA01 S1A12
ST60: ST61 OUTPUTS: RRW1
ST61: ST62 OUTPUTS: SRW1
ST62: ST63
ST63: ST64 OUTPUTS: RA12
ST64: ST65
ST65: ST66
ST66: ST67
ST67: ST68 OUTPUTS: ROPEN1
ST68: ST69 OUTPUTS: SOPEN1 RlsbC
ST69: ST70
ST70: ST71
ST71: ST72 OUTPUTS: SCERAM1
ST72: ST73 OUTPUTS: SlsbC
ST73: ST74 OUTPUTS: S1A12
ST74: ST75 OUTPUTS: RmsbC
ST75: ST76 OUTPUTS: SmsbC
ST76: ST77 OUTPUTS: RCERAM
ST77: ST78
ST78: ST79
ST79: ST80 OUTPUTS: ROPEN1 RA12
ST80: ST81 OUTPUTS: SA2 RA1 RA01 SOPEN1
ST81: ST82 OUTPUTS: RRW1
ST82: ST83 OUTPUTS: SRW1
ST83: ST84 OUTPUTS: SA01 S1A12
ST84: ST85 OUTPUTS: RRW1
ST85: ST86 OUTPUTS: SRW1
ST86: ST87 OUTPUTS: RA2 RA01
ST87: ST88 OUTPUTS: RlsbA
ST88: ST89 OUTPUTS: SlsbA
ST89: ST90 OUTPUTS: SA01
ST90: ST91 OUTPUTS: RmsbA RlsbD
ST91: ST92 OUTPUTS: SmsbA ROPEN1 RA12
ST92: ST93 OUTPUTS: SOPEN1
ST93: ST94 OUTPUTS: SCERAM1
ST94: ST95 OUTPUTS: SlsbD
ST95: ST96 OUTPUTS: S1A12
ST96: ST97 OUTPUTS: RmsbD
ST97: ST98 OUTPUTS: SmsbD
ST98: ST99 OUTPUTS: RCERAM
ST99: ST100
ST100: ST101
ST101: ST102
ST102: ST103 OUTPUTS: RA12
ST103: ST104 OUTPUTS: ROPEN2

```
ST104: ST105      OUTPUTS: SA2 SA1 RA01 SOPEN2
ST105: ST106      OUTPUTS: RRW1
ST106: ST107      OUTPUTS: SRW1
ST107: ST108      OUTPUTS: SA02 S1A12
ST108: ST109      OUTPUTS: RRW1
ST109: ST110      OUTPUTS: SRW1
ST110: ST111      OUTPUTS: RA2 RA02
ST111: ST112      OUTPUTS: RlsbB RA12
ST112: ST113      OUTPUTS: SlsbB
ST113: ST114      OUTPUTS: SA02
ST114: ST115      OUTPUTS: RmsbB RlsbE
ST115: ST116      OUTPUTS: SmsbB ROPEN2
ST116: ST117      OUTPUTS: SOPEN2
ST117: ST118      OUTPUTS: SCERAM1
ST118: ST119      OUTPUTS: SlsbE
ST119: ST120      OUTPUTS: S2A12
ST120: ST121      OUTPUTS: RmsbE
ST121: ST122      OUTPUTS: SmsbE
ST122: ST123      OUTPUTS: RCERAM
ST123: ST124      OUTPUTS: RA12
ST124: ST125
ST125: ST126
ST126: ST127
ST127: ST0        OUTPUTS: SA3 RA1 RA02 ROPEN1 RCK16K

END$
```

II. VERSÃO 3

Multiplexador de audio a 2048 KBit/s

```
TITLE "Multiplexer de 2.048 Mbit/s";

INCLUDE "HDB3";
INCLUDE "ALITRA";
INCLUDE "CONTROL";
INCLUDE "NRZ";

DESIGN IS "MUX" DEVICE IS "EPM5128J";

SUBDESIGN MUX

(hmais_in, hmenos_in, reset, in[7..0], canA,canB,canC,canD,canE, clk_ger, clk_rec4, clk_rec2, mode:INPUT;
erro, erro_v, lip, pll_sum, fp_out, sync_out, serie_al, cerom, select, a14, a15,
ckA, enbA, ckB, enbB, ckC, enbC, ckD, enbD, ckE, enbE, hmais_out, hmenos_out, clk_out :OUTPUT;)

VARIABLE

        BLhdb3          :HDB3;
        BLalitra        :ALITRA;
        BLcontrol       :CONTROL;
        BLnrz           :NRZ;

clk_mux, clk                                                    :NODE;

ger                                                             :TFF;

BEGIN

%XXXX Multiplexer de seleção do relógio do circuito e que é imposto pelo modo de funcionamento.XXXXX%

ger = TFF(VCC, TFF(VCC, clk_ger, VCC, VCC), VCC, VCC);

clk_mux = (BLhdb3.lip $ mode) & ger
        # !(BLhdb3.lip $ mode) & clk_rec2;

clk  = MCELL(!clk_mux);

clk_out =  (BLhdb3.lip $ mode) & ger1
        # !(BLhdb3.lip $ mode) & clk_rec4;

%XXXXXXXXXXXXXXXX Definição das ligações de entrada e saída do bloco HDB3.XXXXXXXXXXXXXXXXX%

BLhdb3.clk = clk;
BLhdb3.hmais = hmais_in;
BLhdb3.hmenos = hmenos_in;
BLhdb3.reset = reset;

lip = BLhdb3.lip;
erro = BLhdb3.erro;
erro_v = BLhdb3.erro_v;
pll_sum = TRI(BLhdb3.pll_sum, !mode);
```

```
%XXXXXXXXXXXXXXX Definição das ligações de entrada e saída do bloco ALITRA.XXXXXXXXXXXXXXXX%

BLalitra.clk = !clk;
BLalitra.reset = reset;
BLalitra.lip = BLhdb3.lip;
BLalitra.nrz_hdb3 = BLhdb3.nrz;

serie_al = BLalitra.serie_out;
sync_out = BLalitra.sync_out;
fp_out = BLalitra.fp_out;

%XXXXXXXXXXXXXXX Definição das ligações de entrada e saída do bloco CONTROL.XXXXXXXXXXXXXXX%

prclk =  mode & BLcontrol.open8
      # !mode & BLcontrol.open12;

BLcontrol.clk = clk;
BLcontrol.st[2..0] = BLalitra.st[2..0];
BLcontrol.c[6..0] = BLalitra.cont[6..0];
BLcontrol.in[] = in[];
BLcontrol.tr[7..0] = BLalitra.tr[7..0];
BLcontrol.canA = canA;
BLcontrol.canB = canB;
BLcontrol.canC = canC;
BLcontrol.canD = canD;
BLcontrol.canE = canE;
BLcontrol.lip = BLhdb3.lip;

select = BLcontrol.select;
cerom = BLcontrol.cerom;
enbA  = BLcontrol.enbA;
enbB  = BLcontrol.enbB;
enbC  = BLcontrol.enbC;
enbD  = BLcontrol.enbD;
enbE  = BLcontrol.enbE;
a14   = BLcontrol.a14;
a15   = BLcontrol.a15;
ckA   = BLcontrol.ckA;
ckB   = BLcontrol.ckB;
ckC   = BLcontrol.ckC;
ckD   = BLcontrol.ckD;
ckE   = BLcontrol.ckE;

%XXXXXXXXXXXXXXXXX Definição das ligações de entrada e saída do bloco NRZ.XXXXXXXXXXXXXXXX%

BLnrz.clk  = clk;
BLnrz.nrz  = BLcontrol.serie;
hmais_out  = BLnrz.hmais;
hmenos_out = BLnrz.hmenos;

END;
```

Desmultiplexador de audio a 2048 KBit/s

```
TITLE "Desmultiplexer de 2.048 Mbit/s";

INCLUDE "HDB3";
INCLUDE "ALITRA";
INCLUDE "CONTROL";
INCLUDE "PARITY";

DESIGN IS "DEMUX" DEVICE IS "EPM5128J"

SUBDESIGN DEMUX
(hmais, hmenos, nrz_in, mode, enable, select, reset, clk_nrz, clk_in                :INPUT;
erro, erro_v, lip, pll_sum, out[12..1], prty, fp_out, sync_out, serie_out, ceram, cerom,
rw, a[3..0], a12, msbA, lsbA, msbB, lsbB, clk_out                                   :OUTPUT;)

VARIABLE

        BLhdb3          :HDB3;
        BLalitra        :ALITRA;
        BLcontrol       :CONTROL;
        BLparity        :PARITY;

clk_mux, clk, clkn, prclk                                                           :NODE;

tr[12..1]                                                                           TRI;

BEGIN

%XXXX Multiplexer de seleção do relógio do circuito e que é imposto pelo modo de funcionamento.XXXX%

clk_mux =  select & clk_in
        # !select & clk_nrz;

%XXXXXXXXXXXXXX Relógio e relógio invertido de funcionamento interno do Demux.XXXXXXXXXXXXX%

clk  = !clk_mux;
clkn =  clk_mux;
clk_out  = !clk_mux;

%XXXXXXXXXXXXXXXXX Definição das ligações de entrada e saída do bloco HDB3.XXXXXXXXXXXXXXXXX%

BLhdb3.clk = clk;
BLhdb3.hmais = hmais;
BLhdb3.hmenos = hmenos;
BLhdb3.reset = reset;

lip = BLhdb3.lip;
erro = BLhdb3.erro;
erro_v = BLhdb3.erro_v;
pll_sum = BLhdb3.pll_sum;
```

```
%XXXXXXXXXXXXXXXX Definição das ligações de entrada e saída do bloco ALITRA.XXXXXXXXXXXXXXX%

BLalitra.clk = clkn;
BLalitra.prclk = prclk;
BLalitra.reset = reset;
BLalitra.prclr = !mode;
BLalitra.select = select;
BLalitra.nrz_in = nrz_in;
BLalitra.nrz_hdb3 = BLhdb3.nrz;

tr[12..1].in = BLalitra.pr[12..1];
tr[12..1].oe = enable;
out[12..1]   = tr[12..1].out;

serie_out = BLalitra.serie_out;
sync_out = BLalitra.sync_out;
fp_out = BLalitra.fp_out;

%XXXXXXXXXXXXXXX Definição das ligações de entrada e saída do bloco CONTROL.XXXXXXXXXXXXXX%

prclk =  mode & BLcontrol.open8
      # !mode & BLcontrol.open12;

BLcontrol.clk = clk;
BLcontrol.st[2..0] = BLalitra.st[2..0];
BLcontrol.cont[6..0] = BLalitra.cont[6..0];

ceram = BLcontrol.ceram;
cerom = BLcontrol.cerom;
a[3..0] = BLcontrol.a[3..0];
a12 = BLcontrol.a12;
rw = BLcontrol.rw;
lsbA = BLcontrol.lsbA;
msbA = BLcontrol.msbA;
lsbB = BLcontrol.lsbB;
msbB = BLcontrol.msbB;

%XXXXXXXXXXXXXXXX Definição das ligações de entrada e saída do bloco PARITY.XXXXXXXXXXXXXXX%

BLparity.clk = clkn;
BLparity.pr[12..1] = BLalitra.pr[12..1];
prty = BLparity.parity;

END;
```

Detector do alinhamento de trama

```
TITLE "Bloco de alinhamento de trama";      DESIGN IS "ALITRA" DEVICE IS "AUTO";

SUBDESIGN ALITRA
(nrz_hdb3, clk, lip, reset :INPUT;   st[2..0], cont[6..0], tr[7..0], fp_out, sync_out, serie_out :OUTPUT;)

VARIABLE

in[7..0], ldn, state0, par, impar, fas, bit4, pr_[4..1]   :NODE;

q[7..0], fp_reg, sync_reg, stq[11..0], srq[12..1]         :DFF;

BEGIN

    srq[].clk = clk;                        %Definicao do shift register Serie-Paralelo de 12 bits%
    srq[].clrn = !reset;
    srq[12].d = nrz_hdb3;
    srq[11..1].d = srq[12..2];

    fas = (srq[10..4] == B"1101100");       %Definicao do comparador que encontra a FAS%

    tr[7..0] = srq[8..1];                   %Definicao dos diferentes sinais de saida do SPR12%
    serie_out = srq1;
    bit4 = srq4;

%XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX CONTADOR XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX%

    q[7..0].clk = clk;
    q[7..0].clrn = !reset;

    in[7..0] = B"11111111";

    q0.d = (q0 & ldn) $ (in0 & !ldn # ldn);

    q1.d = (q1 & ldn) $ (in1 & !ldn # q0 & ldn);

    q2.d = (q2 & ldn) $ (in2 & !ldn # q0 & q1 & ldn);

    q3.d = (q3 & ldn) $ (in3 & !ldn # q0 & q1 & q2 & ldn);

    q4.d = (q4 & ldn) $ (in4 & !ldn # q0 & q1& q2& q3 & ldn);

    q5.d = (q5 & ldn) $ (in5 & !ldn # q0 & q1 & q2 & q3 & q4 & ldn);

    q6.d = (q6 & ldn) $ (in6 & !ldn # q0 & q1 & q2 & q3 & q4 & q5 & ldn);

    q7.d = (q7 & ldn) $ (in7 & !ldn # q0 & q1 & q2 & q3 & q4 & q5 & q6 & ldn);

    cont[6..0] = q[6..0];

    par_t = TFF(q[7..0] == B"11111110", clk, !state0, VCC);

    par   = (q[7..0] == B"11111110") & !par_t;
    impar = (q[7..0] == B"11111110") &  par_t;
```

```
%XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX ONE_HOT XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX%

    (stq[], fp_reg, sync_reg).clk  = clk;
    stq[].clrn  = reset !& lip;
    (fp_reg, sync_reg).prn  = !reset;

    pr_1 = impar & !fas;                    %Definição dos instantes de transição%
    pr_2 = par   & !bit4;
    pr_3 = par   & bit4;
    pr_4 = impar &  fas;

%Descodificação dos estados da máquina%

    stq0.d = !( stq1   & pr_2
              # stq2   & pr_1
              # stq6   & pr_1
              # stq8   & pr_2
              # stq9   & pr_1
              # stq10  & pr_2
              # stq11  & (pr_1 # pr_2)
              # state0 & !fas);

    state0 = EXP(stq0);

    stq1.d = state0 &  fas
              # stq1   & !par;

    stq2.d = stq1 &  pr_3
              # stq2 & !impar;

    stq3.d = stq2  &  pr_4
              # stq4  &  pr_4
              # stq5  &  pr_3
              # stq6  &  pr_4
              # stq8  &  pr_3
              # stq3  & !pr_1 & !pr_2;

    stq4.d = stq3  &  pr_1
              # stq7  &  pr_3
              # stq10 &  pr_3
              # stq4  & !impar & !pr_2;

    stq5.d = stq3  &  pr_2
              # stq7  &  pr_4
              # stq10 &  pr_4
              # stq5  & !pr_1 & !par;

    stq6.d = stq4  &  pr_1
              # stq9  &  pr_3
              # stq11 &  pr_3
              # stq6  & !impar & !pr_2;

    stq7.d = stq4  & pr_2
              # stq5  & pr_1
              # stq7  & !par & !impar;
```

```
        stq8.d = stq5  &  pr_2
                 # stq10 &  pr_4
                 # stq11 &  pr_4
                 # stq8  & !pr_1 & !par;

        stq9.d = stq6  &  pr_2
                 # stq7  &  pr_1
                 # stq9  & !par & !impar;

        stq10.d = stq7  &  pr_2
                 # stq8  &  pr_1
                 # stq10 & !par & !impar;

        stq11.d = stq9  &  pr_2
                 # stq10 &  pr_1
                 # stq11 & !par & !impar;


fp_reg.d = !(stq3 & (pr_3 # pr_4 # !par # !impar));      %Definicao dos sinais de saida%
fp_out   = fp_reg;

    IF !lip THEN
                 ldn = !((state0 & fas)  #  (q[7..0] == B"11111110"));
    ELSE
                 ldn = VCC;
    END IF;

sync_reg.d = !( pr_1 & !state0 & !stq1 & !stq2 & !stq6 & !stq9 & !stq11
                 # pr_2 & !state0 & !stq1 & !stq2 & !stq9 & !stq10 & !stq11
                 # pr_3 & !state0 & !stq1 & !stq2
                 # pr_4 & !state0 & !stq1);
sync_out   = sync_reg;

st[2..0]  =  (stq[2..1],state0);




%XXXXXXXXXXXXXXXXXXXXXXXXXX MÉTODO TRADICCIONAL XXXXXXXXXXXXXXXXXXXXXXXX%

VARIABLE

    clk,sync,fp,clear        :NODE;

    ss : MACHINE OF BITS (q3,q2,q1,q0)
                 WITH STATES(  st0=B"000" ,
                                          st1=B"0001" ,
                                          st2=B"0010" ,
                                          st3=B"0011" ,
                                          st4=B"0100" ,
                                          st5=B"0101" ,
                                          st6=B"0110" ,
                                          st7=B"0111" ,
                                          st8=B"1000",
                                          st9=B"1001",
                                        st10=B"1010",
                                        st11=B"1011");
```

```
BEGIN

    clk = clk_in;

    ss.clk = clk;

    TABLE
        ss,   par,  impar,  bit2,  fas  =>  load,  sync,  fp,  clear,  ss;
        st0,  x,    x,      x,     1    =>  0,     1,     1,   1,      st1;
        st0,  x,    x,      x,     0    =>  1,     1,     1,   0,      st0;

        st1,  1,    0,      0,     x    =>  1,     1,     1,   1,      st0;
        st1,  1,    0,      1,     x    =>  0,     1,     1,   1,      st2;
        st1,  0,    x,      x,     x    =>  1,     1,     1,   1,      st1;
        st1,  1,    1,      x,     x    =>  1,     1,     1,   1,      st1;

        st2,  1,    1,      x,     0    =>  1,     1,     1,   1,      st0;
        st2,  1,    1,      x,     1    =>  0,     0,     1,   1,      st3;
        st2,  0,    x,      x,     x    =>  1,     1,     1,   1,      st2;
        st2,  1,    0,      x,     x    =>  1,     1,     1,   1,      st2;

        st3,  1,    0,      0,     x    =>  0,     0,     1,   1,      st5;
        st3,  1,    0,      1,     x    =>  0,     0,     0,   1,      st3;
        st3,  1,    1,      x,     0    =>  0,     0,     1,   1,      st4;
        st3,  1,    1,      x,     1    =>  0,     0,     0,   1,      st3;
        st3,  0,    x,      x,     x    =>  1,     1,     0,   1,      st3;

        st4,  1,    0,      0,     x    =>  0,     0,     1,   1,      st7;
        st4,  1,    0,      1,     x    =>  0,     0,     1,   1,      st4;
        st4,  1,    1,      x,     0    =>  0,     0,     1,   1,      st6;
        st4,  1,    1,      x,     1    =>  0,     0,     1,   1,      st3;
        st4,  0,    x,      x,     x    =>  1,     1,     1,   1,      st4;

        st5,  1,    0,      0,     x    =>  0,     0,     1,   1,      st8;
        st5,  1,    0,      1,     x    =>  0,     1,     1,   1,      st3;
        st5,  1,    1,      x,     0    =>  0,     0,     1,   1,      st7;
        st5,  1,    1,      x,     1    =>  0,     0,     1,   1,      st5;
        st5,  0,    x,      x,     x    =>  1,     1,     1,   1,      st5;

        st6,  1,    0,      0,     x    =>  0,     0,     1,   1,      st9;
        st6,  1,    0,      1,     x    =>  0,     0,     1,   1,      st6;
        st6,  1,    1,      x,     0    =>  1,     1,     1,   1,      st0;
        st6,  1,    1,      x,     1    =>  0,     0,     1,   1,      st3;
        st6,  0,    x,      x,     x    =>  1,     1,     1,   1,      st6;

        st7,  1,    0,      0,     x    =>  0,     0,     1,   1,      st10;
        st7,  1,    0,      1,     x    =>  0,     0,     1,   1,      st4;
        st7,  1,    1,      x,     0    =>  0,     0,     1,   1,      st9;
        st7,  1,    1,      x,     1    =>  0,     0,     1,   1,      st5;
        st7,  0,    x,      x,     x    =>  1,     1,     1,   1,      st7;

        st8,  1,    0,      0,     x    =>  1,     1,     1,   1,      st0;
        st8,  1,    0,      1,     x    =>  0,     1,     1,   1,      st3;
        st8,  1,    1,      x,     0    =>  0,     0,     1,   1,      st10;
        st8,  1,    1,      x,     1    =>  0,     0,     1,   1,      st8;
        st8,  0,    x,      x,     x    =>  1,     1,     1,   1,      st8;
```

```
        st9,  1,  0,  0,  x  =>  0,  0,  1,  1,  st11;
        st9,  1,  0,  1,  x  =>  0,  0,  1,  1,  st6;
        st9,  1,  1,  x,  0  =>  1,  1,  1,  1,  st0;
        st9,  1,  1,  x,  1  =>  0,  0,  1,  1,  st5;
        st9,  0,  x,  x,  x  =>  1,  1,  1,  1,  st9;

        st10, 1,  0,  0,  x  =>  1,  1,  1,  1,  st0;
        st10, 1,  0,  1,  x  =>  0,  0,  1,  1,  st4;
        st10, 1,  1,  x,  0  =>  0,  0,  1,  1,  st11;
        st10, 1,  1,  x,  1  =>  0,  0,  1,  1,  st8;
        st10, 0,  x,  x,  x  =>  1,  1,  1,  1,  st10;

        st11, 1,  0,  0,  x  =>  1,  1,  1,  1,  st0;
        st11, 1,  0,  1,  x  =>  0,  0,  1,  1,  st6;
        st11, 1,  1,  x,  0  =>  1,  1,  1,  1,  st0;
        st11, 1,  1,  x,  1  =>  0,  0,  1,  1,  st8;
        st11, 0,  x,  x,  x  =>  1,  1,  1,  1,  st11;
        END TABLE;

fp_out  = DFF(fp, clk, VCC, VCC);
sync_out  = DFF(sync, clk, VCC, VCC);
clear_out  = DFF(clear, clk, VCC, VCC);

END;
```

Unidade de controlo do multiplexador

```
TITLE "Maquina de estados de control do mux de 2048 KBit/s";   DESIGN IS "CONTROL" DEVICE IS "AUTO";

SUBDESIGN CONTROL
(clk, lip, st[2..0], in[7..0], tr[7..0], canA, canB, canC, canD, canE, c[6..0]          :INPUT;
serie, cerom, select, a14, a15, ckE, ckD, ckC,c kB, ckA, enbE, enbD, enbC,e nbB, enbA   :OUTPUT;)

VARIABLE

Rckbyte,Sckbyte,        Rckbreak,Sckbreak,      Rhalframe,Shalframe,
Rsatger,Ssatger,        Rcerom,Scerom,          Ra14,Sa14,Ra15,Sa15,
RckE,SckE,RenE,SenE,    RckD,SckD,RenD,SenD,    RckC,SckC,RenC,SenC,
RckB,SckB,RenB,SenB,    RckA,SckA,RenA,SenA                                     :NODE;

ckbyte, ckbreak, halframe, satrec, satger, invert, bus[7..0]                    :NODE;

piso[7..0], d[3..0]                                                             :DFF;

BEGIN

select = MCELL (  (c[]  >   D"4")  &  (c[] <  D"27")  &  !canA
                #  (c[]  >  D"28")  &  (c[] <  D"51")  &  !canB
                #  (c[]  >  D"52")  &  (c[] <  D"75")  &  !canC
                #  (c[]  >  D"76")  &  (c[] <  D"99")  &  !canD
                #  (c[]  > D"100")  &  (c[] < D"124")  &  !canE
                #  ((c[] > D"124")  #  (c[] <   D"4"))  &  lip       );


Rhalframe = (c[] == D"126");                    Shalframe = (c[] == D"127");
halframe = SRFF(Shalframe, Rhalframe, !clk, !(st0 # st1 # st2) & !lip, VCC);

Rckbyte  = (c[] == B"XXXX111");                 Sckbyte  = (c[] == B"XXXX000");
ckbyte = SRFF(Sckbyte, Rckbyte, !clk, !(st0 # st1 # st2) & !lip, VCC);

Rsatger  = (c[] == D"125") & select;            Ssatger  = (c[] == D"3");
satger = SRFF(Ssatger, Rsatger, !clk, !(st0 # st1 # st2) & !lip, VCC);

RckE = (c[] == B"X000100");                     SckE = (c[] == B"X000110");
ckE = SRFF(SckE, RckE, !clk, !(st0 # st1 # st2) & !lip, VCC);

RckD = (c[] == B"X101100");                     SckD = (c[] == B"X101110");
ckD = SRFF(SckD, RckD, !clk, !(st0 # st1 # st2) & !lip, VCC);

RckC = (c[] == B"X010100");                     SckC = (c[] == B"X010110");
ckC = SRFF(SckC, RckC, !clk, !(st0 # st1 # st2) & !lip, VCC);

RckB = (c[] == B"X111100");                     SckB = (c[] == B"X111110");
ckB = SRFF(SckB, RckB, !clk, !(st0 # st1 # st2) & !lip, VCC);

RckA = (c[] == B"X100100");                     SckA = (c[] == B"X100110");
ckA = SRFF(SckA, RckA, !clk, !(st0 # st1 # st2) & !lip, VCC);

RenA = (c[] == B"0000101");                     SenA = (c[] == B"0011001");
enA = SRFF(SenA, RenA, !clk, !(st0 # st1 # st2) & !lip, VCC);
```

```
RenB = (c[] == B"0011101");                    SenB = (c[] == B"0110001");
enB = SRFF(SenB, RenB, !clk, !(st0 # st1 # st2) & !lip, VCC);

RenC = (c[] == B"0110101");                    SenC = (c[] == B"1001001");
enC = SRFF(SenC, RenC, !clk, !(st0 # st1 # st2) & !lip, VCC);

RenD = (c[] == B"1001101");                    SenD = (c[] == B"1100001");
enD = SRFF(SenD, RenD, !clk, !(st0 # st1 # st2) & !lip, VCC);

RenE  = (c[] == B"1100101");                   SenE = (c[] == B"1111001");
enE = SRFF(SenE, RenE, !clk, !(st0 # st1 # st2) & !lip, VCC);

Rckbreak = (c[] == D"8") # (c[] == D"10") # (c[] == D"20") # (c[] == D"32") # (c[] == D"34") # (c[] == D"44")
         # (c[] == D"56") # (c[] == D"58") # (c[] == D"68") # (c[] == D"80") # (c[] == D"82") # (c[] == D"92")
         # (c[] == D"104") # (c[] == D"106") # (c[] == B"116");
Sckbreak = (c[] == D"7") # (c[] == D"9") # (c[] == D"19") # (c[] == D"31") # (c[] == D"33") # (c[] == D"43")
         # (c[] == D"55") # (c[] == D"57") # (c[] == D"67") # (c[] == D"79") # (c[] == D"81") # (c[] == D"91")
         # (c[] == D"103") # (c[] == D"105") # (c[] == D"115");
ckbreak = SRFF(Sckbreak, Rckbreak, !clk, !(st0 # st1 # st2) & !lip, VCC);


Rcerom = ((c[] == D"6") # (c[] == D"30") # (c[] == D"54") # (c[] == D"78") # (c[] == D"102")) & select;

Scerom = (c[] == D"25") # (c[] == D"49") # (c[] == D"73") # (c[] == D"97") # (c[] == D"121");
cerom = SRFF(Scerom, Rcerom, !clk, !(st0 # st1 # st2) & !lip, VCC);

Ra14 = (c[] == D"6") # (c[] == D"30") # (c[] == D"54") # (c[] == D"78") # (c[] == D"102");
Sa14 = (c[] == D"16") # (c[] == D"32") # (c[] == D"56") # (c[] == B"80") # (c[] == B"104");
a14 = SRFF(Sa14, Ra14, !clk, !(st0 # st1 # st2) & !lip, VCC);

Ra15 = (c[] == D"6") # (c[] == D"30") # (c[] == D"54") # (c[] == D"78") # (c[] == D"102");
Sa15 = (c[] == D"16") # (c[] == D"24") # (c[] == D"64") # (c[] == D"88") # (c[] == D"96");
a15 = SRFF(Sa15, Ra15, !clk, !(st0 # st1 # st2) & !lip, VCC);

satrec = cerom & satger;

invert = (VCC, TFF (VCC, halframe, VCC, VCC), VCC, VCC);

d[].clk  = ckbreak;
d[3..0].d = in[3..0];

bus[7..0] = (VCC,VCC,invert,VCC,VCC,GND,invert,VCC) & !satger
         # (in[7..4],d[3..0]) &  satger & select
         # tr[7..0] & satrec & !lip
         # (VCC, VCC, VCC, VCC, VCC, VCC, VCC, VCC) & satrec & lip;

piso[].clk = clk;
piso[].d   = bus[7..0] & !ckbyte
         # (GND,piso[7..1]) & ckbyte;

serie = piso0;

END;
```

Unidade de controlo do desmultiplexador

```
TITLE "Maquina de estados de control do demux de 2.048 Mbits"; DESIGN IS "Control" DEVICE IS "AUTO";

SUBDESIGN Control
(clk, st[2..0], cont[6..0]                                                    :INPUT;
ceram, cerom, a0, a1, a2, a3, a12, rw, open12, open8, lsbA, msbA, lsbB, msbB  :OUTPUT;)

VARIABLE

Sceram, Rceram, Sa0, Ra0, Sa1, Ra1, Sa2, Ra2, Sa3, Ra3, Sa12, Ra12,
Srw, Rrw, Sopen12, Ropen12, Sopen8, Ropen8,
SlsbA, RlsbA, SmsbA, RmsbA, SlsbB, RlsbB, SmsbB, RmsbB                        :NODE;

BEGIN

Srw = (c[] == D"22") # (c[] == D"25") # (c[] == D"46") # (c[] == D"49") # (c[] == D"70") # (c[] == D"73")
        # (c[] == D"94") # (c[] == D"97") # (c[] == D"118") # (c[] == D"121");
Rrw = (c[] == D"21") # (c[] == D"24") # (c[] == D"45") # (c[] == D"48") # (c[] == D"69") # (c[] == D"72")
        # (c[] == D"93") # (c[] == D"96") # (c[] == D"117") # (c[] == D"120");
rw = SRFF(Srw, Rrw, clk, !(st0 # st1 # st2) & !lip, VCC);

Sceram  = (c[] == D"11") # (c[] == D"35") # (c[] == D"59") # (c[] == D"81") # (c[] == D"105");
Rceram  = (c[] == D"16") # (c[] == D"40") # (c[] == D"64") # (c[] == D"86") # (c[] == D"109");
ceram = SRFF(Sceram, Rceram, clk, !(st0 # st1 # st2) & !lip, VCC);

Sa12 = (c[] == D"13") # (c[] == D"15") # (c[] == D"37") # (c[] == D"55") # (c[] == D"61")
        # (c[] == D"71") # (c[] == D"83") # (c[] == D"95") # (c[] == D"107") # (c[] == D"119");
Ra12 = (c[] == D"1") # (c[] == D"17") # (c[] == D"33") # (c[] == D"39") # (c[] == D"51")
        # (c[] == D"67") # (c[] == D"79") # (c[] == D"90") # (c[] == D"99") # (c[] == D"111");
a12 = SRFF(Sa12, Ra12, clk, !(st0 # st1 # st2) & !lip, VCC);

Sa3 =Sa12 = (c[] == D"41") # (c[] == D"115");
Ra3 = (c[] == D"45") # (c[] == D"122");
a3 = SRFF(Sa3, Ra3, clk, !(st0 # st1 # st2) & !lip, VCC);

Sa2 = (c[] == D"68") # (c[] == D"92") # (c[] == D"122");
Ra2 = (c[] == D"21") # (c[] == D"74") # (c[] == D"98");
a2 = SRFF(Sa2, Ra2, clk, !(st0 # st1 # st2) & !lip, VCC);

Sa1 = (c[] == D"17") # (c[] == D"45") # (c[] == D"92");
Ra1 = (c[] == D"21") # (c[] == D"68") # (c[] == D"115");
a1 = SRFF(Sa1, Ra1, clk, !(st0 # st1 # st2) & !lip, VCC);

Sa0 = (c[] == D"19") # (c[] == D"23") # (c[] == D"43") # (c[] == D"47") # (c[] == D"71")
        # (c[] == D"77") # (c[] == D"95") # (c[] == D"101") # (c[] == D119") # (c[] == D"125");
Ra0 = (c[] == D"17") # (c[] == D"21") # (c[] == D"41") # (c[] == D"45") # (c[] == D"68")
        # (c[] == D"74") # (c[] == D"92") # (c[] == D"98") # (c[] == D115") # (c[] == D"122");
a0 = SRFF(Sa0, Ra0, clk, !(st0 # st1 # st2) & !lip, VCC);

Sopen12 = (c[] == D"8") # (c[] == D"20") # (c[] == D"32") # (c[] == D"44") # (c[] == D"56")
        # (c[] == D"68") # (c[] == D"80") # (c[] == D"92") # (c[] == D104") # (c[] == D"116");
Ropen12 = (c[] == D"7") # (c[] == D"19") # (c[] == D"31") # (c[] == D"43") # (c[] == D"55")
        # (c[] == D"67") # (c[] == D"79") # (c[] == D"91") # (c[] == D103") # (c[] == D"115");
```

```
open12 = SRFF(Sopen12, Ropen12, clk, !(st0 # st1 # st2) & !lip, VCC);

Sopen8 = (c[] == B"XXXX000");                Ropen8 = (c[] == B"XXXX111");
open8 = SRFF(Sopen8, Ropen8, clk, !(st0 # st1 # st2) & !lip, VCC);

SmsbB = (c[] == B"X100111");                 RmsbB = (c[] == B"X100110");
msbB= SRFF(SmsbB, RmsbB, clk, !(st0 # st1 # st2) & !lip, VCC);

SlsbB = (c[] == B"X100101");                 RlsbB = (c[] == D"32") # (c[] == D"99");
lsbB= SRFF(SlsbB, RlsbB, clk, !(st0 # st1 # st2) & !lip, VCC);

SmsbA = (c[] == B"X001111");                 RmsbA = (c[] == B"X001110");
msbA= SRFF(SmsbA, RmsbA, clk, !(st0 # st1 # st2) & !lip, VCC);

SlsbA = (c[] == B"X001100");                 RlsbA = (c[] == D"8") # (c[] == D"75");
lsbA = SRFF(SlsbA, RlsbA, clk, !(st0 # st1 # st2) & !lip, VCC);

cerom = ceram $ rw;  %XOR%

END;
```

Codificação e compressão da lei A

```
PROGRAM CODIFICA(INPUT,OUTPUT);

VAR     A,B                                          : ARRAY [1..16] OF INTEGER;
        SLOT                                         : ARRAY [1..8] OF INTEGER;
        i, a14a15, cont, sinal, segmento, factor, valor, epmsb, eplsb   :INTEGER;
        hexamsb,hexalsb                              :CHAR;

        MAT1                                         : ARRAY [1..1025,1..44] OF CHAR;
        endr,somA2,x,j                               :INTEGER;

        i1,i2                                        :INTEGER;
        EP1FIC                                       :FILE OF CHAR;

{***************************** Rotina de cálculo de potências ******************************}

        FUNCTION power(base,expoente:INTEGER):INTEGER;

        VAR aux,j:INTEGER;

        BEGIN
                aux:=1;
                IF expoente <> 0 THEN FOR j:=1 TO expoente DO aux:=aux*base;
                power:=aux;
        END;

{************************ Rotina de conversão decimal/hexadecimal ************************}

        FUNCTION hexa(dec:INTEGER):CHAR;

        BEGIN
                CASE dec OF
                        0 : hexa:='0';
                        1 : hexa:='1';
                        2 : hexa:='2';
                        3 : hexa:='3';
                        4 : hexa:='4';
                        5 : hexa:='5';
                        6 : hexa:='6';
                        7 : hexa:='7';
                        8 : hexa:='8';
                        9 : hexa:='9';
                        10 : hexa:='A';
                        11 : hexa:='B';
                        12 : hexa:='C';
                        13 : hexa:='D';
                        14 : hexa:='E';
                        15 : hexa:='F';
                END;
        END;
```

```
{************************** Rotina de conversão decimal/binário ***************************}

        PROCEDURE converte(num,K:INTEGER);

        BEGIN
              FOR i:=K DOWNTO 1 DO
                        BEGIN
                              B[i]:=num MOD 2;
                              num:=num DIV 2;
                        END;
        END;

{********************** Rotina de criação de linhas em formato TEK HEX ************************}

        PROCEDURE formata;

        VAR     somA1,at2,at3,at4,lsa1,msa1,lsa2,msa2  :INTEGER;

        BEGIN
              IF (cont MOD 16) <> 15 THEN
                        BEGIN
                              somA2:=somA2+epmsb+eplsb;
                              MAT1[x,10+2*j]:=hexamsb;
                              MAT1[x,11+2*j]:=hexalsb;
                              j:=j+1;
                        END
              ELSE BEGIN
                        at2:=(endr DIV 16) MOD 16;
                        at3:=(endr DIV 256) MOD 16;
                        at4:=endr DIV 4096 + a14a15*4;
                        somA1:=at2+at3+at4+1;
                        lsa1:=somA1 MOD 16;
                        msa1:=somA1 DIV 16;
                        somA2:=(somA2+epmsb+eplsb) MOD 1024;
                        lsa2:=somA2 MOD 16;
                        msa2:=somA2 DIV 16;
                        MAT1[x,1]:='/';
                        MAT1[x,2]:=hexa(at4);
                        MAT1[x,3]:=hexa(at3);
                        MAT1[x,4]:=hexa(at2);
                        MAT1[x,5]:='0';
                        MAT1[x,6]:='1';
                        MAT1[x,7]:='0';
                        MAT1[x,8]:=hexa(msa1);
                        MAT1[x,9]:=hexa(lsa1);
                        MAT1[x,40]:=hexamsb;
                        MAT1[x,41]:=hexalsb;
                        MAT1[x,42]:=hexa(msa2);
                        MAT1[x,43]:=hexa(lsa2);
                        MAT1[x,44]:=CHR(10);
                        x:=x+1;
                        endr:=endr+16;
                        somA2:=0;
                        j:=0;
              END;
        END;
```

```
{****************************** PROGRAMA PRINCIPAL ******************************}

BEGIN
    FOR a14a15:=0 TO 2 DO
        BEGIN
            somA2:=0;
            x:=1;
            j:=0;
            endr:=0;
            FOR cont:=0 TO 16383 DO
                BEGIN
                    IF cont <= 8191 THEN sinal:=0 ELSE sinal:=1;
                    IF cont <= 127 THEN
                        BEGIN
                            segmento:=0;
                            factor:=1;
                        END
                    ELSE IF cont <= 255  THEN
                        BEGIN
                            segmento:=2;
                            factor:=1;
                        END
                    ELSE IF cont <= 511  THEN
                        BEGIN
                            segmento:=3;
                            factor:=2;
                        END
                    ELSE IF cont <= 1023  THEN
                        BEGIN
                            segmento:=4;
                            factor:=4;
                        END
                    ELSE IF cont <= 2047  THEN
                        BEGIN
                            segmento:=5;
                            factor:=8;
                        END
                    ELSE IF cont <= 4095  THEN
                        BEGIN
                            segmento:=6;
                            factor:=16;
                        END
                    ELSE IF cont <= 8191  THEN
                        BEGIN
                            segmento:=7;
                            factor:=32;
                        END
                    ELSE IF cont <= 12287 THEN
                        BEGIN
                            segmento:=7;
                            factor:=32;
                        END
                    ELSE IF cont <= 14335 THEN
                        BEGIN
                            segmento:=6;
                            factor:=16;
                        END
```

```
ELSE IF cont <= 15359 THEN
        BEGIN
                segmento:=5;
                factor:=8;
        END
ELSE IF cont <= 15871 THEN
        BEGIN
                segmento:=4;
                factor:=4;
        END
ELSE IF cont <= 16127 THEN
        BEGIN
                segmento:=3;
                factor:=2;
        END
ELSE IF cont <= 16255 THEN
        BEGIN
                segmento:=2;
                factor:=1;
        END
ELSE IF cont <= 16383 THEN
        BEGIN
                segmento:=0;
                factor:=1;
        END;

IF sinal=0 THEN valor := (cont DIV factor) MOD 128
ELSE valor := 127 - ((cont DIV factor) MOD 128);

A[1]:=sinal;

converte(segmento,3);

A[2]:=B[1];
A[3]:=B[2];
A[4]:=B[3];

converte(valor,7);

A[5]:=B[1];
A[6]:=B[2];
A[7]:=B[3];
A[8]:=B[4];
A[9]:=B[5];
A[10]:=B[6];
A[11]:=B[7];

IF segmento=0 THEN
        BEGIN
                FOR i:=4 TO 10 DO A[i]:=A[i+1];
                A[11]:=0;
        END;

FOR i:=1 TO 5 DO  A[i] := (A[i]+1) MOD 2;

A[12]:=(A[1] + A[2] + A[3] + A[4]+ A[5] + 1) MOD 2;
```

```
                IF a14a15=0 THEN
                        BEGIN
                                SLOT[1]:=A[11];
                                SLOT[2]:=A[1] ;
                                SLOT[3]:=A[10];
                                SLOT[4]:=A[2] ;
                                SLOT[5]:=A[9] ;
                                SLOT[6]:=A[3] ;
                                SLOT[7]:=A[8] ;
                                SLOT[8]:=A[4] ;
                        END
                ELSE IF a14a15=1 THEN
                        BEGIN
                                SLOT[1]:=A[7] ;
                                SLOT[2]:=A[5] ;
                                SLOT[3]:=A[6] ;
                                SLOT[4]:=A[12];
                                SLOT[5]:=A[11];
                                SLOT[6]:=A[1] ;
                                SLOT[7]:=A[10];
                                SLOT[8]:=A[2] ;
                        END
                ELSE IF a14a15=2 THEN
                        BEGIN
                                SLOT[1]:=A[9] ;
                                SLOT[2]:=A[3] ;
                                SLOT[3]:=A[8] ;
                                SLOT[4]:=A[4] ;
                                SLOT[5]:=A[7] ;
                                SLOT[6]:=A[5] ;
                                SLOT[7]:=A[6] ;
                                SLOT[8]:=A[12];
                        END;
                epmsb:=0;
                eplsb:=0;

                FOR i:=4 DOWNTO 1 DO
                        BEGIN
                         epmsb := epmsb + power(2,i-1)*SLOT[i+4];
                         eplsb := eplsb + power(2,i-1)*SLOT[i];
                        END;

                hexamsb:=hexa(epmsb);
                hexalsb:=hexa(eplsb);

                formata;
        END;

MAT1[1025,1]:='/';
MAT1[1025,2]:='0';
MAT1[1025,3]:='0';
MAT1[1025,4]:='1';
MAT1[1025,5]:='3';
MAT1[1025,6]:='0';
MAT1[1025,7]:='0';
MAT1[1025,8]:='0';
MAT1[1025,9]:='4';
```

```
                    FOR I2:=10 TO 44 DO MAT1[1025,I2]:=' ';

                    IF a14a15=0 THEN
                            BEGIN
                                ASSIGN(EP1FIC,'MUX-A0.DAT');
                                REWRITE(EP1FIC);
                                FOR I1:=1 TO 1025 DO
                                        BEGIN
                                            FOR I2:=1 TO 44 DO
                                              BEGIN
                                                WRITE(EP1FIC,MAT1[I1,I2]);
                                              END;
                                        END;
                                CLOSE(EP1FIC);
                            END
                    ELSE IF a14a15=1 THEN
                            BEGIN
                                ASSIGN(EP1FIC,'MUX-A1.DAT');
                                REWRITE(EP1FIC);
                                FOR I1:=1 TO 1025 DO
                                        BEGIN
                                            FOR I2:=1 TO 44 DO
                                              BEGIN
                                                WRITE(EP1FIC,MAT1[I1,I2]);
                                              END;
                                        END;
                                CLOSE(EP1FIC);
                            END
                      ELSE IF a14a15=2 THEN
                            BEGIN
                                ASSIGN(EP1FIC,'MUX-A2.DAT');
                                REWRITE(EP1FIC);
                                FOR I1:=1 TO 1025 DO
                                        BEGIN
                                            FOR I2:=1 TO 44 DO
                                              BEGIN
                                                WRITE(EP1FIC,MAT1[I1,I2]);
                                              END;
                                        END;
                                CLOSE(EP1FIC);
                            END;
                END;
END.
```

Descodificação e descompressão da lei A

```
PROGRAM DESCOD(INPUT,OUTPUT);

VAR     A,B                                     : ARRAY [1..12] OF INTEGER;
        C                                       : ARRAY [1..14] OF INTEGER;
        i, cont,segmento,z,factor,valor,result, MS1,LS1,MS2,LS2,cons     :INTEGER;
        EP1,EP2                                 :ARRAY [1..8] OF INTEGER;
        MSO1,LSO1,MSO2,LSO2                     :CHAR;
        MS1,LS1,MS2,LS2,cons:INTEGER;

        somA2,somB2,x,j,endr                    :INTEGER;
        MAT1,MAT2                               : ARRAY [1..256,1..44] OF CHAR;

        i1,i2                                   :INTEGER;
        EP1FIC,EP2FIC                           :FILE OF CHAR;

{***************************** Rotina de cálculo de potências ******************************}

        FUNCTION power(base,expoente:INTEGER):INTEGER;

        VAR aux,j:INTEGER;

        BEGIN
                aux:=1;
                IF expoente <>0 THEN FOR j:=1 TO expoente DO aux:=aux*base;
                power:=aux;
        END;

{************************ Rotina de conversão decimal/hexadecimal ************************}

        FUNCTION hexa(dec:INTEGER):CHAR;

        BEGIN
                CASE dec OF
                        0 : hexa:='0';
                        1 : hexa:='1';
                        2 : hexa:='2';
                        3 : hexa:='3';
                        4 : hexa:='4';
                        5 : hexa:='5';
                        6 : hexa:='6';
                        7 : hexa:='7';
                        8 : hexa:='8';
                        9 : hexa:='9';
                        10 : hexa:='A';
                        11 : hexa:='B';
                        12 : hexa:='C';
                        13 : hexa:='D';
                        14 : hexa:='E';
                        15 : hexa:='F';
                END;
        END;
```

```
{************************** Rotina de conversão decimal/binário ***************************}

        PROCEDURE converte(num,k:INTEGER);

        BEGIN
              FOR i:=K DOWNTO 1 DO
                        BEGIN
                              A[i]:=num MOD 2;
                              num:=num DIV 2;
                        END;
        END;

{********************** Rotina de criação de linhas em formato TEK HEX ************************}

        PROCEDURE formata;

        VAR     somA1,at2,at3,lsa1,msa1,lsa2,msa2:INTEGER;
                somB1,bt2,bt3,lsb1,msb1,lsb2,msb2:INTEGER;

        BEGIN
              IF (cont MOD 16) <> 15 THEN
                        BEGIN
                              somA2:=somA2+LS1+MS1; somB2:=somB2+LS2+MS2;
                              MAT1[x,10+2*j]:=MSO1; MAT2[x,10+2*j]:=MSO2;
                              MAT1[x,11+2*j]:=LSO1; MAT2[x,11+2*j]:=LSO2;
                              j:=j+1;
                        END
              ELSE BEGIN
                        at2:=(endr DIV 16) MOD 16; bt2:=(endr DIV 16) MOD 16;
                        at3:=endr DIV 256; bt3:=endr DIV 256;
                        somA1:=at2+at3+1; somB1:=bt2+bt3+1;
                        lsa1:=somA1 MOD 16; lsb1:=somB1 MOD 16;
                        msa1:=somA1 DIV 16; msb1:=somB1 DIV 16;
                        somA2:=(somA2+MS1+LS1) MOD 256; somB2:=( somB2+MS2+LS2) MOD 256;
                        lsa2:=somA2 MOD 16; lsb2:=somB2 MOD 16;
                        msa2:=somA2 DIV 16; msb2:=somB2 DIV 16;
                        MAT1[x,1]:='/'; MAT2[x,1]:='/';
                        MAT1[x,2]:='0'; MAT2[x,2]:='0';
                        MAT1[x,3]:=hexa(at3); MAT2[x,3]:=hexa(bt3);
                        MAT1[x,4]:=hexa(at2); MAT2[x,4]:=hexa(bt2);
                        MAT1[x,5]:='0'; MAT2[x,5]:='0';
                        MAT1[x,6]:='1'; MAT2[x,6]:='1';
                        MAT1[x,7]:='0'; MAT2[x,7]:='0';
                        MAT1[x,8]:=hexa(msa1); MAT2[x,8]:=hexa(msb1);
                        MAT1[x,9]:=hexa(lsa1); MAT2[x,9]:=hexa(lsb1);
                        MAT1[x,40]:=MSO1; MAT2[x,40]:=MSO2;
                        MAT1[x,41]:=LSO1; MAT2[x,41]:=LSO2;
                        MAT1[x,42]:=hexa(+msa2); MAT2[x,42]:=hexa(msb2);
                        MAT1[x,43]:=hexa(lsa2); MAT2[x,43]:=hexa(lsb2);
                        MAT1[x,44]:=CHR(10); MAT2[x,44]:=CHR(10);
                        x:=x+1;
                        endr:=endr+16;
                        somA2:=0; somB2:=0;
                        j:=0;
              END;
        END;
```

```
{***************************** PROGRAMA PRINCIPAL *******************************}

BEGIN
        somA2:=0; somB2:=0;
        x:=1;
        j:=0;
        endr:=0;
        FOR cont:=0 TO 4095 DO
                BEGIN
                        converte(cont,12);
                        B[1]:=A[2];
                        B[2]:=A[4];
                        B[3]:=A[6];
                        B[4]:=A[8];
                        B[5]:=A[10];
                        B[6]:=A[11];
                        B[7]:=A[9];
                        B[8]:=A[7];
                        B[9]:=A[5];
                        B[10]:=A[3];
                        B[11]:=A[1];
                        B[12]:=A[12];

                        FOR i:=1 TO 5 DO B[i] := (B[i]+1) MOD 2;

                        segmento:=0;
                        C[1]:=B[1];

                        FOR i:=2 TO 4 DO segmento := segmento + B[i]*power(2,4-i);

                        IF (segmento=0) OR (segmento=1) THEN
                                FOR i:=10 DOWNTO 4 DO B[i+1]:=B[i];
                        IF C[1]=0 THEN
                                BEGIN
                                        CASE segmento OF
                                                0 : z:=0;
                                                1 : z:=0;
                                                2 : z:=128;
                                                3 : z:=256;
                                                4 : z:=512;
                                                5 : z:=1024;
                                                6 : z:=2048;
                                                7 : z:=4096;
                                        END;
                                END
                        ELSE IF C[1]=1 THEN
                                BEGIN
                                        CASE segmento OF
                                                0 : z:=16256;
                                                1 : z:=16256;
                                                2 : z:=16128;
                                                3 : z:=15872;
                                                4 : z:=15360;
                                                5 : z:=14336;
                                                6 : z:=12288;
                                                7 : z:=8192;
                                        END;
                                END;
```

```
        CASE segmento OF
                0 : factor:=1;
                1 : factor:=1;
                2 : factor:=1;
                3 : factor:=2;
                4 : factor:=4;
                5 : factor:=8;
                6 : factor:=16;
                7 : factor:=32;
        END;
        valor:=0;
        result:=0;
        FOR i:=5 TO 11 DO  valor := valor + B[i]*power(2,11-i);

        IF C[1]=0 THEN
                BEGIN
                        result := z + factor*valor + (factor DIV 2);
                END
        ELSE IF C[1]=1 THEN
                BEGIN
                        valor := 127 - valor;
                        result := z + factor*valor - (factor DIV 2);
                END;

        FOR i:=14 DOWNTO 2 DO
                BEGIN
                        C[i]:=result MOD 2;
                        result:=result DIV 2;
                END;

        FOR i:=1 TO 8 DO EP1[i]:=C[i];     { engloba c(1) a c(8)  }
        FOR i:=1 TO 6 DO EP2[i]:=C[i+8];   { engloba c(9) a c(14) }

        EP2[7]:=0;
        EP2[8]:=0;

        LS1:=0;
        MS1:=0;
        LS2:=0;
        MS2:=0;

        FOR i:=1 TO 4 DO
                BEGIN
                        cons:=power(2,4-i);
                        LS1:=LS1+EP1[i+4]*cons;  { engloba c(4) a c(7)   }
                        MS1:=MS1+EP1[i]*cons;    { engloba c(1) a c(3)   }
                        LS2:=LS2+EP2[i+4]*cons;  { engloba c(11) a c(14) }
                        MS2:=MS2+EP2[i]*cons;    { engloba c(8) a c(10)  }
                END;

        LSO1:=hexa(LS1);
        MSO1:=hexa(MS1);
        LSO2:=hexa(LS2);
        MSO2:=hexa(MS2);

        formata;
END;
```

```
                ASSIGN(EP1FIC,'MSB2.DAT');
                REWRITE(EP1FIC);
                FOR i1:=1 TO 256 DO
                        FOR i2:=1 TO 44 DO
                                BEGIN
                                        WRITE(EP1FIC,MAT1[i1,i2]);
                                END;
                CLOSE(EP1FIC);

                ASSIGN(EP2FIC,'LSB2.DAT');
                REWRITE(EP2FIC);
                FOR i1:=1 TO 256 DO
                        FOR i2:=1 TO 44 DO
                                BEGIN
                                        WRITE(EP2FIC,MAT2[i1,i2]);
                                END;
                CLOSE(EP2FIC);
END.
```