Desenvolvimento do Circuito Integrado *TB47* (Tratador de Interface de Linha *PCM-30*) Utilizando a Metodologia de Projeto *Top Down*

Janete Mouallem

Campinas, 16 de maio de 1996.

Orientador:

Prof. Dr. **José Antônio Siqueira Dias**

Tese apresentada à Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas UNICAMP, como requisito parcial para obtenção do título de **Doutor em Engenharia Elétrica**.

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

`M86d

Mouallem, Janete
Desenvolvimento do circuito integrado TB47 (Tratador de Interface de Linha PCM-30) utilizando a metodologia de projeto top down / Janete Mouallem.--Campinas, SP: [s.n.], 1996.

Orientador: José Antônio Siqueira Dias.
Tese (doutorado) - Universidade Estadual de Campinas Faculdade de Engenharia Elétrica.

1. Circuitos integrados digitais. 2. Hardware - Linguagens descritivas. 3. Telecomunicações. I. Dias, José Antônio Siqueira. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica. III. Título.

Resumo

O objetivo deste trabalho é apresentar o circuito integrado para o "Tratamento da Interface de Linha" (*TB47*) e sua implementação em *FPGAs Xilinx*, através de uma metodologia de projeto *Top Down*.

Como o *TB47* foi desenvolvido para utilização em placa do sistema *CLAD* (Concentrador de Linhas de Assinantes Distribuido) em desenvolvimento no Centro de Pesquisa e Desenvolvimento da Telebrás, inicialmente será descrito este sistema e como o *TB47* se encaixa no mesmo.

Em seguida, será feita uma descrição do funcionamento do *TB47* e apresentada a metodologia de projeto utilizada para seu desenvolvimento.

Finalmente, será apresentada a sua implementação em dois componentes *FPGAs* (*Field Programmable Gate Arrays*) *Xilinx 4008PQ208*, para teste do sistema. Para isto utilizou-se o *software Xilinx Automatic CAE Tools* (*XACT*).

Agradecimentos

Na realização deste trabalho muitas pessoas ajudaram direta ou indiretamente.

Ao Edson, pela participação e apoio necessários durante o período de elaboração desta tese.

Aos colegas do CPqD-Telebrás pelas diversas discussões técnicas, que muito contribuiram para o conteúdo desta tese. Em especial Rosana, Mendonça, Dei Santi e Ronaldo, também envolvidos diretamente neste trabalho.

Aos meus chefes no CPqD-Telebrás, José Roberto e Favoreto, que me propiciaram o tempo necessário para a conclusão deste trabalho.

Em especial, gostaria de apresentar meu agradecimento ao Prof. Dr. José Antônio Siqueira Dias, orientador deste trabalho, pelas discussões técnicas e inúmeras sugestões que permitiram que o mesmo fosse realizado a contento.

ÍNDICE

CAPÍTULO I

Introdução

Este trabalho tem por objetivo apresentar o projeto do circuito integrado para o "Tratamento da Interface de Linha" (*TB47*) e sua implementação em *FPGA* (*Field Programmable Gate Array*) do fabricante *Xilinx* [1], através de uma metodologia de projeto *Top Down*. Este circuito foi implementado para uso no sistema Concentrador de Linhas de Assinantes Distribuido (*CLAD*). *Xilinx* é uma empresa americana atuante na área de dispositivos lógicos programáveis.

Trata-se de um trabalho desenvolvido por engenheiros do Centro de Pesquisa e Desenvolvimento da Telebrás. O Sistema *CLAD* é de responsabilidade do Departamento de Desenvolvimento de Sistemas DDS e o desenvolvimento do projeto *TB47* é de responsabilidade de engenheiros do Departamento de Tecnologias Básicas DTB.

O sistema *CLAD* tem aplicação em locais distantes da central telefônica e não assistidos por rede básica, em localidades de baixa concentração demográfica, prédios, etc. Este sistema tem como grande vantagem a economia na rede, possibilitando o aumento do número de assinantes sem a contrapartida do investimento em ampliação da rede, ou seja, melhora a relação custo/benefício da rede já implantada.

O circuito *TB47* foi desenvolvido utilizando uma metodologia de projeto *Top Down*, tendo como plataforma de trabalho o *software Mentor* versão 8.2_10, instalado em Estações de Trabalho *SUN Sparc Station 10*. A *Mentor Graphics Corporation* é uma empresa americana que desenvolve e comercializa software para automação de projeto.

Inicialmente, para teste do sistema, que consiste em uma configuração inovadora, este circuito foi implementado, como protótipo, em duas *FPGAs Xilinx 4008PQ208-5*. Para isto utilizou-se o *software Xilinx Automatic CAE Tools* (*XACT*), técnica na qual nos concentraremos neste trabalho. Pois, as razões de se ter um protótipo são antecipar a depuração do sistema, possibilitando testes com alto grau de controlabilidade e observabilidade, que viabilizem detectar e corrigir erros de projeto ou de especificação em fases preliminares do ciclo de projeto [2].

Para facilitar o entendimento da função do circuito, inicialmente, será descrito o sistema *CLAD* e como o circuito *TB47* se encaixa neste sistema. Em seguida, será feita uma descrição do funcionamento do circuito e apresentada a metodologia de projeto utilizada para seu desenvolvimento. Finalmente, será apresentada a sua implementação em dois componentes *FPGA Xilinx 4008PQ208*.

CAPÍTULO II

Sistema *CLAD* (Concentrador de Linhas de Assinantes Distribuido)

Aqui será definido o sistema *CLAD*, apresentada uma comparação com a topologia da rede atual e algumas possíveis aplicações [3].

1 Características Gerais

O Sistema Concentrador de Linhas de Assinantes Distribuido (*CLAD*) foi concebido para ser interligado a centrais telefônicas da família TRÓPICO-RA através de troncos *PCM-30*.

Sua arquitetura básica pode ser vista na figura 1. A partir da central TRÓPICO-RA dois enlaces *PCM-30* transitam em sentidos opostos, interligando todos os nós do Sistema, chamados de Unidades de Derivação de Linha (*UDLs*).

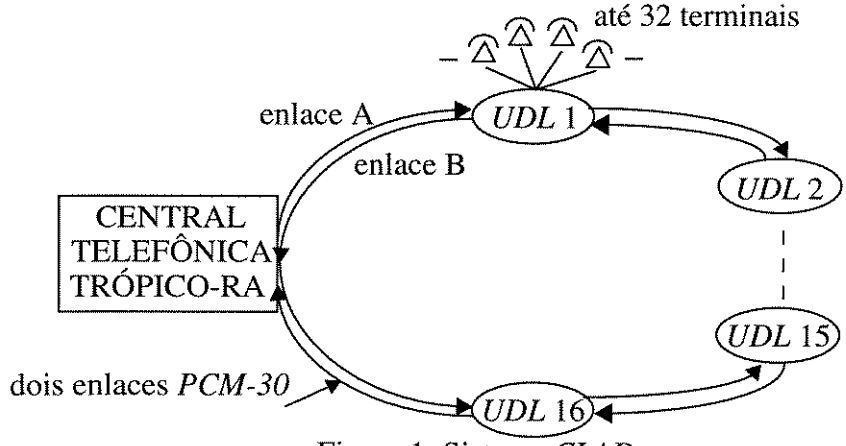


Figura 1: Sistema *CLAD*.

Este sistema pode ser interligado por uma rede de cobre ou fibra óptica.

As *UDLs* seriam instaladas em postes próximos do usuário e distantes da central telefônica, otimizando o uso da rede. Cada *UDL* é responsável pela interface entre os enlaces *PCM* A e B com até 32 terminais.

Os objetivos principais deste sistema, através da otimização do uso da rede instalada e da aproximação da eletrônica ao usuário, são reduzir sensivelmente o custo do terminal integrado e ganhar pares na rede de assinantes.

Os benefícios introduzidos por este sistema são:

- possibilidade de digitalização da rede até a casa do assinante;
- comunicação de dados a taxas de 16, 32 e 64kbits/s;

- diminuição do custo do terminal integrado, devido a redução dos custos da rede telefônica;
- economia de fios na rede de assinantes com consequente diminuição dos custos de implantação;
- viabilização econômica do atendimento às áreas rurais;
- maior eficiência na utilização dos recursos da rede, devido ao elevado grau de integração entre transmissão e comutação.

2 Situação Atual

A rede de assinantes atual possui uma topologia em forma de estrela evidenciando a exclusividade do par de fios (enlaces A e B - figura 1) desde o Distribuidor Geral da central telefônica até a casa do usuário, como pode ser visto na figura 2. Aqui, tanto a distância da central trânsito até as centrais locais, quanto a distância da central local até os usuários, são longas.

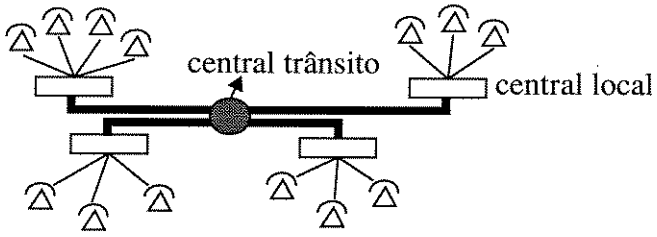


Figura 2: Topologia em Forma de Estrela.

3 Topologia com *CLAD*

A topologia de rede com o *CLAD* passa da forma de estrela para a forma de anel, com a origem e o término do anel na central, como pode ser visto na figura 1.

As *UDLs* (Unidades de Derivação de Linha) levam a eletrônica de linha de assinante para a proximidade do usuário, otimizando desta forma a rede.

Nesta topologia, para atender 512 assinantes são necessários 4 pares de fios de cobre ou 4 fibras ópticas ao invés de 512 pares de fios de cobre atuais.

4 Arquitetura *CLAD*

A arquitetura do sistema *CLAD* possui as seguintes características:

- topologia em anel (constituída por dois enlaces digitais contradirecionais);
- redundância ativa de enlaces com partição de carga;
- unidade de derivação com circuitos de linha no poste;
- transmissão padrão *PCM-30* (G-732 do CCITT);
- sinalização entre *UDL* e a central de comutação via canal 16 (64kbits/s);
- *UDLs* microprocessadas;

- protocolo de sinalização do anel: ponto multi-ponto, com a central sendo mestre do anel;
- número de assinantes/*CLAD* = 512;
- número de *UDLs*/*CLAD* = até 16;
- número de assinantes/*UDL* = até 32;
- alimentação local;
- razão Média de Tráfego = 0,087 Erlang/assinante.

A figura 3 mostra o exemplo de um usuário neste sistema, ligando a *UDL* a uma residência através de dois fios de cobre ou uma fibra óptica.

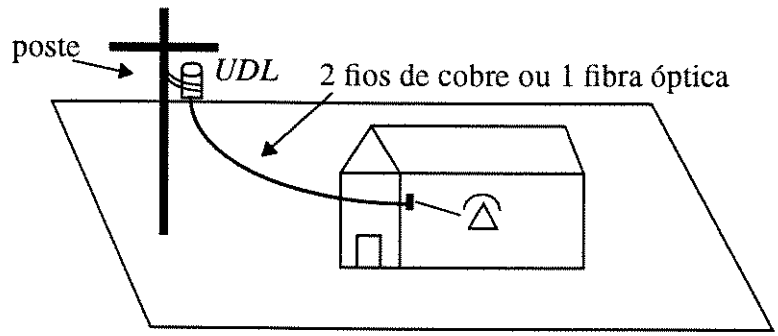


Figura 3: Ligação da *UDL* a uma Residência.

5 Aplicações

As principais aplicações seriam em:

- áreas urbanas de baixa concentração demográfica;
- pequenas localidades;
- áreas rurais;
- áreas urbanas de baixo poder aquisitivo.

Algumas possíveis aplicações específicas estão ilustradas a seguir.

5.1 A *UDL* como Estação Rádio Base

Pode-se associar a uma *UDL* uma estação rádio base (*ERB*) para telefonia celular fixa, atendendo usuários num raio de 15 a 20 km (rural), como pode ser visto na figura 4.

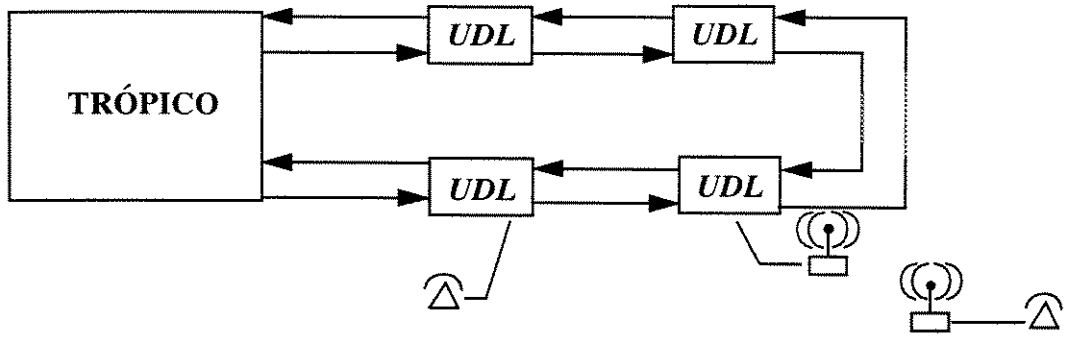


Figura 4: A *UDL* como Estação Rádio Base.

5.2 Utilização do *CLAD* na Zona Rural

Na figura 5, o *CLAD* óptico atende, através de uma *UDL*, uma vila com fios de 300m; outra *UDL* atende uma cooperativa a 3km e duas outras *UDLs* atendem a usuários rurais distantes de 15 a 20 km em torno da *UDL*.

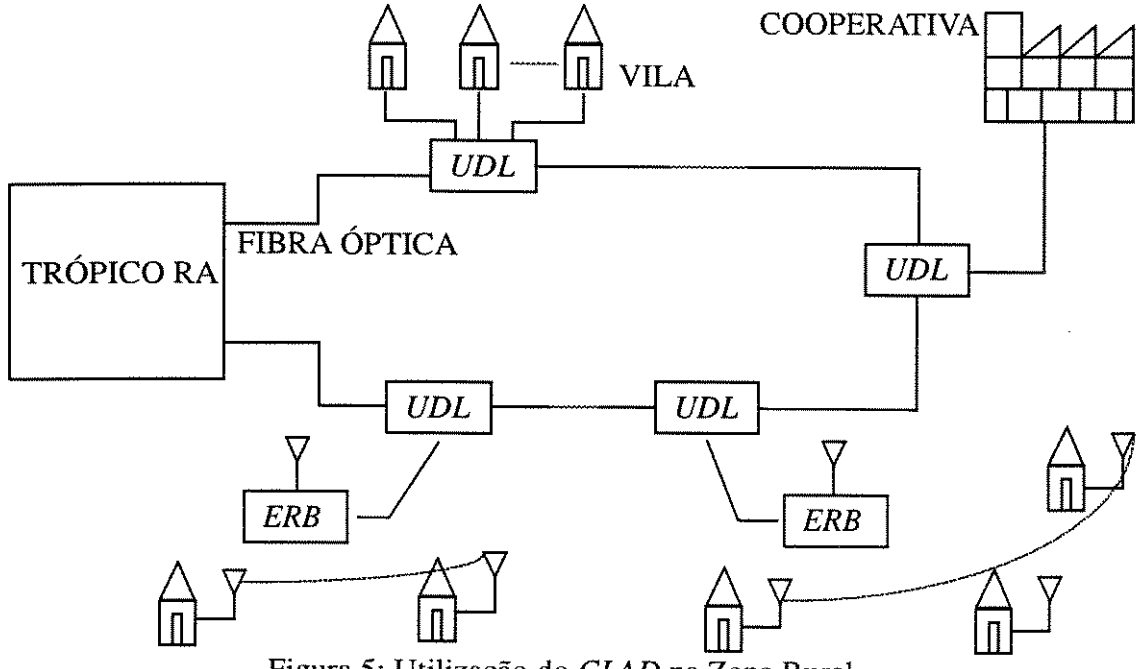


Figura 5: Utilização do *CLAD* na Zona Rural.

CAPÍTULO III

Posicionamento do Circuito Integrado *TB47* no Sistema *CLAD*

Esta seção mostra como o circuito integrado *TB47* se posiciona no sistema *CLAD* e suas vantagens para este sistema.

1 Informações de Uso no Sistema

Cada *UDL*, representada na figura 1 (capítulo II), faz a interface entre os enlaces *PCM* (A e B) e até 32 terminais e pode ser representada por um diagrama em blocos como mostrado na figura 6 [4].

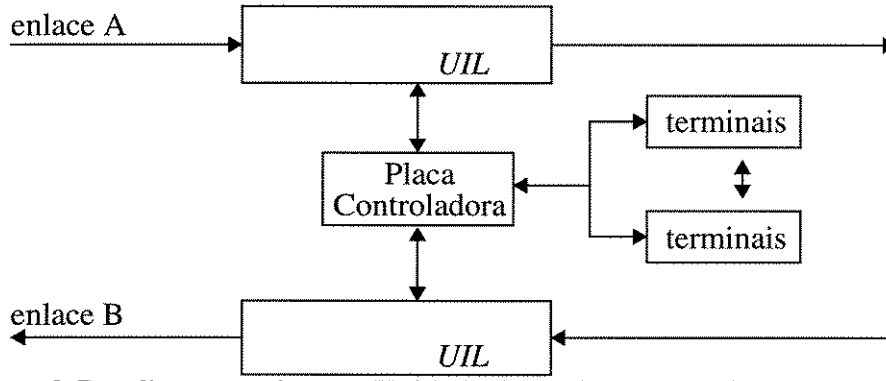


Figura 6: Detalhamento de uma Unidade de Derivação de Linha (*UDL*).

Cada Unidade de Interface de Linha (*UIL*) trata um dos enlaces *PCM*, fazendo a interface entre este e a placa controladora da *UDL*. Esta placa controla a comutação dos canais de voz alocados a esta *UDL*, para seus respectivos terminais.

A *UDL* deve prever, para o caso de falha em um dos enlaces, a facilidade de comutação dos enlaces, controlada pela placa controladora.

A placa *UIL* é composta de interfaces analógicas, na entrada e saída do enlace, e o circuito *TB47*. Como pode ser visto na figura 7, as interfaces analógicas são constituídas de um circuito regenerador de linha, na recepção do enlace, e de *buffers* de linha, na transmissão do enlace. As funções do circuito regenerador de linha são recuperação do relógio de linha e do sinal de linha.

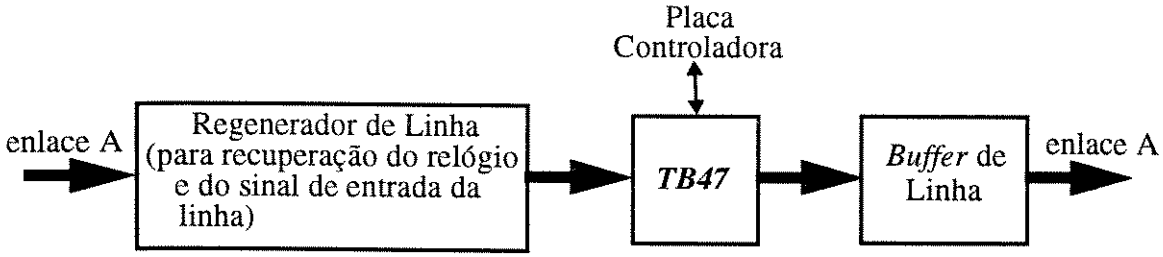


Figura 7: Detalhamento de uma *UIL*.

2 Aplicações

Concebido para o Sistema *CLAD*, o *TB47* pode ser empregado em qualquer outro sistema que necessite de interface com equipamentos de transmissão digital (família de multiplexadores) e centrais telefônicas digitais, que empreguem o *PCM-30*/CRC-4.

3 Características Elétricas

As principais características elétricas do circuito *TB47* são:

- compatível com *TTL*;
- tensão de alimentação: 5V ± 5%;
- consumo máximo: 120mW;
- frequência máxima de operação: 4 MHz;
- temperatura de operação: 0 a 70°C;
- capacidade de carga das saídas: 2 *TTL-LS*.

4 Vantagens para o Sistema

O uso do *TB47*, além de viabilizar o Sistema como um todo trouxe ganhos nos seguintes pontos:

- placa com menos componentes e, por consequência mais confiável, e com menor custo de produção;
- a introdução de facilidades de teste *on the fly* para o sistema permite que a *UDL*, instalada a distância da central telefônica, possa ser monitorada sem a necessidade de deslocamento de pessoal;
- a *UDL* do *CLAD* foi projetada para ser instalada ao ar livre, estando sujeita às variações climáticas comuns no Brasil. Portanto, necessita ser fabricada em uma tecnologia de baixo consumo, o que é satisfeito por circuitos dedicados, que dissipam muito menos potência do que uma implementação das mesmas funções com circuitos integrados comerciais.

CAPÍTULO IV

Descrição Funcional do Circuito Integrado *TB47* (Tratador de Interface de Linha *PCM-30*)

Esta seção descreve todas as funções do *TB47*. Estas funções podem ser divididas em 5 blocos básicos, que são: tratamento do enlace recebido (recepção do enlace), tratamento do enlace a ser transmitido (transmissão do enlace), interface com a placa controladora [5], facilidades para teste e *loop* de reconfiguração do enlace.

A figura 8 mostra o diagrama em blocos básico do *TB47*.

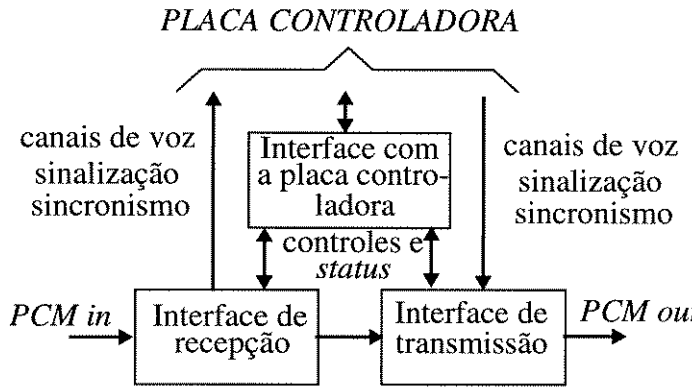


Figura 8: Diagrama em Blocos Básico do *TB47*.

As figuras 9, 10 e 11 expandem o diagrama anterior.

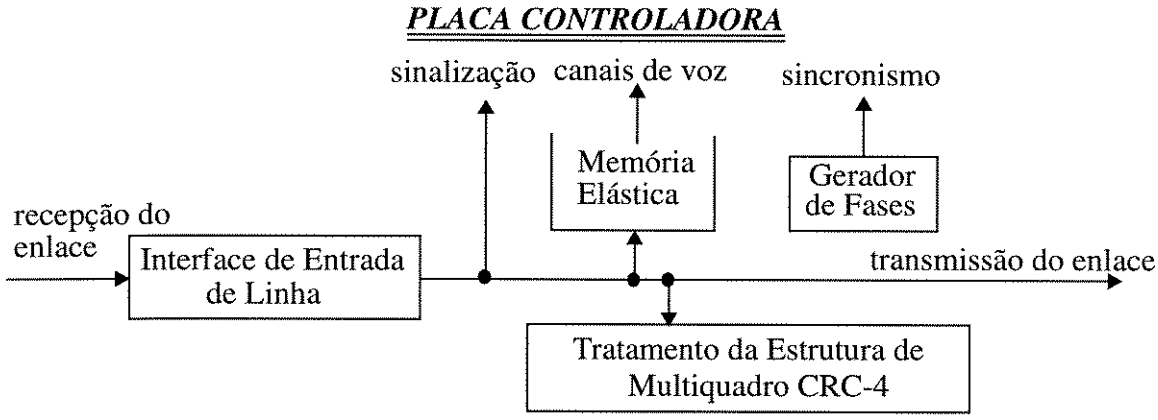


Figura 9: Diagrama em Blocos da Recepção.

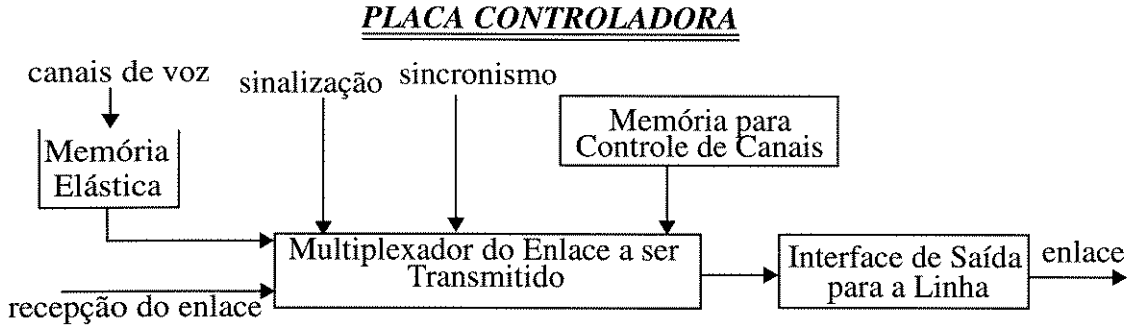


Figura 10: Diagrama em Blocos da Transmissão.

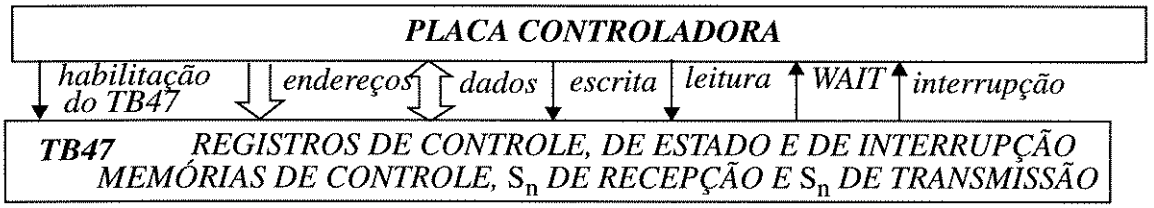


Figura 11: Interface com a Placa Controladora.

A figura 12 mostra um diagrama em blocos para reconfiguração do enlace (seção 5).

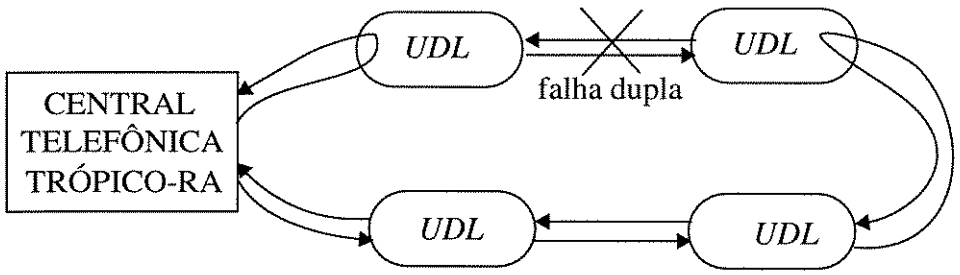


Figura 12: Sistema *CLAD* com Falha Dupla - *Loop* de Reconfiguração do Enlace.

As observações abaixo são importantes para o melhor entendimento do circuito:

- os *bits* dos registros de interrupção, estado e controle, são designados no texto como X.Y, onde X é o nome do registrador e Y é o nome do *bit* do registrador em questão, como por exemplo *ctrl.rdis* (seção 3.1);
- a palavra "*local*" se refere à Interface com a Placa Controladora e aos circuitos cuja temporização seja estabelecida pelos sinais dos pinos de entrada *SCLK_4M* e *SYPQ*;
- a palavra "*linha*" se refere à interface de linha e aos circuitos síncronos com os sinais dos pinos de entrada *RRCLK_2M* e *RRCLK_4M*.

A seguir descreveremos cada um destes blocos.

1 Recepção do Enlace

É responsável pelas seguintes funções:

1.1 Interface de Entrada de Linha

Este bloco deve tratar os sinais provenientes da linha *PCM*, via os pinos de entrada *RDIP*, *RDIM* (interface elétrica) e *ROID* (interface óptica). Deve também detectar a presença ou não de sinal nestes pinos de entrada. Para tanto deve ser composto de 3 sub-blocos (Tratamento da Interface Elétrica, Tratamento da Interface Óptica e Detector de Falta de Sinal na Linha), definidos abaixo.

Este circuito, deve ainda:

- ser programável para aceitar os sinais de entrada tanto com polaridade positiva, quanto negativa. A programação é feita pela placa controladora através do *bit* de controle *ctrl.rdis* (seção 3.1.3);
- externar o alarme de falta do sinal de entrada proveniente da linha (*bit* de estado *intl.nos*, seção 3.1.1).

Os sinais de saída dos blocos de tratamento das interfaces elétrica e óptica devem ser multiplexados, sob controle do pino de entrada *POE1* (ver figura 18, página 49), gerando um único sinal interno, no formato unipolar, a ser usado pelos demais blocos.

1.1.1 Tratamento da Interface Elétrica

Este bloco deve executar a decodificação do código de transmissão ternário HDB3 (*High Density Bipolar 3*), de acordo com a recomendação G-703 do CCITT [6]. A decodificação do código HDB3 feita a partir dos pinos de entrada *RDIP* e *RDIM* no formato unipolar RZ, gera um sinal binário unipolar NRZ como saída.

1.1.2 Tratamento da Interface Óptica

Este bloco trata o pino de entrada *ROID*, que pode ser RZ ou NRZ, da seguinte forma:

- caso o sinal recebido neste pino tenha sido embaralhado na sua origem, ele precisa ser desembaralhado. O controle para o bloco executar a função de desembaralhador é dado pelo pino *POE2*='1' (tabela 11). Neste caso o sinal deve ser RZ. O polinômio utilizado no circuito desembaralhador é x^7+x^6+1, conforme definido na Recomendação G-709, do CCITT [6];
- caso *POE2*='0', o circuito desembaralhador não é usado, devendo o sinal na entrada *ROID* ser NRZ (figura 21).

A função do embaralhador/desembaralhador é garantir transições na linha, mesmo que o sinal original contenha grandes sequências de zeros ou uns [7]. Evitando, desta forma, que sinais vindos pela linha de recepção e que tenham sequências grandes de zeros ou uns, levem o PLL do recuperador de relógio da linha a fugir da frequência a ser recuperada.

1.1.3 Detector de Falta de Sinal na Linha

A condição de falta de sinal na linha, é detectada quando são recebidos na linha 3 ou menos 1's em um intervalo de tempo de 250 μs, equivalente a 512 *bits*. Sempre que for detectada esta condição na recepção, deverá ser ativado o *bit* de estado *intl.nos*='1' e gerado pedido de interrupção (*bit intl.inte*='1' e pino de saída *AINT*='0') para o processador, caso o *bit masl.mnos* do registro de máscara não esteja ativo (seção 3.1.1).

A condição de detecção da falta de sinal descrita acima implica em resultados diferentes para as interfaces ópticas e elétricas. A detecção de 3 ou menos 1's na interface óptica implica na detecção de 3 ou menos *bits* em '1'. Já a detecção feita na interface elétrica irá sinalizar a existência de impulsos na linha, que não necessariamente correspondem a um *bit* em '1'.

1.2 Memória Elástica de Recepção

Como o sinal *PCM* de entrada estará síncrono com o relógio recuperado da linha e o circuito de controle (placa controladora) estará funcionando com outro relógio, é necessário prever um circuito que permita o acoplamento entre ambos, de forma que as amostras de voz recebidas com o relógio de linha possam passar a ser tratadas com o relógio local.

Tal circuito, memória elástica, deve ser implementado com capacidade para armazenar dois quadros completos do *PCM* (total de 64 canais). Sendo que cada amostra de voz em um canal tem 8 *bits*. Pode-se considerar que a memória esteja dividida em duas porções, cada uma com 32 palavras correspondendo aos 32 canais de um quadro *PCM 30*. As porções serão designadas: quadros *A* e *B*.

Para proporcionar a detecção de possíveis erros na própria memória, deve ser implementada a função de geração e verificação da paridade, para cada um dos *bytes*. Assim, ao ser escrito um *byte*, deve ser gerada a sua respectiva paridade que deve ser armazenada juntamente com o *byte*, formando uma palavra de 9 *bits*. Após cada leitura de uma destas palavras deve ser verificado se a paridade dos 8 *bits* que compõem o *byte* lido coincide com o valor do *bit* restante. Em caso negativo deve ser ativado o *bit int2.epm* do registro de interrupção (seção 3.1.1).

A função de verificação da paridade deve ser inibida quando ocorrer a perda de sincronismo de multiquadro e permanecer inibida até que se tenha a garantia de que não ocorram possíveis violações de *timing* da memória provocando erros de paridade.

No caso de perda de sincronismo de quadro, a escrita na memória elástica deve continuar na base de tempo anterior. Somente após a recuperação do alinhamento de multiquadro (que pode estar fora de fase com a base de tempo anterior), a operação de escrita deve ser ressincronizada.

A leitura dos canais é feita com base no relógio e sincronismo de quadro local (pinos *SCLK_4M* e *SYPQ*). Como a frequência deste relógio e a frequência do relógio recuperado da linha poderão apresentar variações em instantes significativos em relação as suas posições de referência no tempo (diferenças de fase, *jitter, wander)*, poderá ocorrer que os instantes de leitura e escrita se aproximem. A sobreposição destes acessos à memória deve ser evitada uma vez que poderá implicar em algum tipo de violação no *timing* da memória. Para evitar que os instantes cheguem a coincidir deve ser prevista uma guarda de 0,5 pulso de relógio para cada lado dos pulsos de leitura e de escrita.

Uma condição de *slip* é detectada quando os ponteiros de escrita (W) e de leitura (R) da memória estão próximos (sobreposição de guardas), isto é, o ponteiro de escrita está dentro dos limites do *slip* (S_+, S_-), confome figura 13. Deve ser implementado um circuito que sempre que detectar uma condição de *slip*, execute as seguintes funções:

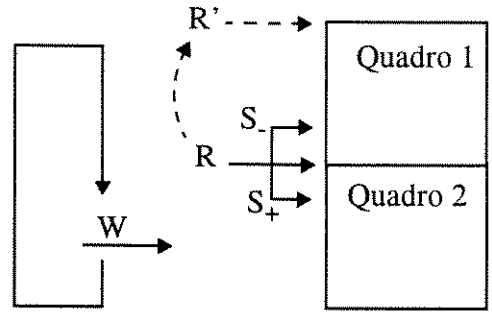


Figura 13: Condição de *Slip*.

- *slip* negativo (o ponteiro W se aproxima do ponteiro R, ou seja, os instantes de leitura e escrita estão próximos, no mesmo canal e quadro): o próximo quadro recebido é ignorado.
- *slip* positivo (o ponteiro R se aproxima do ponteiro W, ou seja, os instantes de leitura e escrita estão próximos, no mesmo canal e quadro): o quadro anterior é lido novamente.
- *slip* de canal (os ponteiros W e R estão se aproximando, porém estão acessando o mesmo canal em quadros diferentes): deve ser provocada a comutação do instante de leitura da memória entre os intervalos de tempo dos *bits* 2 e 6 de cada canal.

Cada ocorrência de *slip* deve ativar o *bit* de estado *int2.slp* e cada ocorrência de erro de paridade na memória elástica deve ativar o *bit* de estado *int2.epm* (seção 3.1.1). Não será feita a contagem de ocorrências de *slip*.

Deve ser detectada a diferença de fase entre os relógios de escrita e leitura desta memória. Com este valor o processador pode determinar a diferença de fase entre os relógios de linha *RRCLK_4M* e local *SCLK_4M*, para uso no PLL. O valor desta diferença de fase estará disponível no registro de estado *DFI* (seção 3.1.4). O conteúdo deste registro deve ser:

bit 7: *bit* menos significativo do contador de quadro de linha,

bits 6 a 2: contador de canal de linha e

bits 1 e 0: os dois *bits* mais significativos do contador de *bits* de linha.

Desta forma, a diferença de fase entre os contadores é fornecida com resolução de 2 *bits*.

1.3 Tratador de CRC de Recepção

Para prover uma proteção adicional quanto ao falso alinhamento de quadro e uma monitoração da taxa de erro no enlace, o *TB47* deve ser capaz de tratar a estrutura de multiquadro CRC-4 definida na recomendação G-704 do CCITT [6].

A tabela 1 mostra a alocação dos *bits* 1 a 8 dos intervalos de tempo de canal 0 na estrutura de multiquadro CRC-4. A palavra de CRC a ser transmitida deverá ser calculada a cada submultiquadro e transferida para o circuito tratador de CRC de transmissão (seção 2.6) para ser inserida nos *bits* C_1 a C_4 do submultiquadro subsequente, sendo C_1 o *bit* mais significativo. A palavra de alinhamento de multiquadro CRC é representada pela sequência binária "001011", transmitida no *bit* 1 dos canais 0 dos quadros ímpares (de 1 a 11), como mostra a tabela 1.

submultiquadro	quadro	1	2	3	4	5	6	7	8
I	0	C_1	0	0	1	1	0	1	1
	1	0	1	A	S_{n1}	S_{n2}	S_{n3}	S_{n4}	S_{n5}
	2	C_2	0	0	1	1	0	1	1
	3	0	1	A	S_{n1}	S_{n2}	S_{n3}	S_{n4}	S_{n5}
	4	C_3	0	0	1	1	0	1	1
	5	1	1	A	S_{n1}	S_{n2}	S_{n3}	S_{n4}	S_{n5}
	6	C_4	0	0	1	1	0	1	1
	7	0	1	A	S_{n1}	S_{n2}	S_{n3}	S_{n4}	S_{n5}
II	8	C_1	0	0	1	1	0	1	1
	9	1	1	A	S_{n1}	S_{n2}	S_{n3}	S_{n4}	S_{n5}
	10	C_2	0	0	1	1	0	1	1
	11	1	1	A	S_{n1}	S_{n2}	S_{n3}	S_{n4}	S_{n5}
	12	C_3	0	0	1	1	0	1	1
	13	E	1	A	S_{n1}	S_{n2}	S_{n3}	S_{n4}	S_{n5}
	14	C_4	0	0	1	1	0	1	1
	15	E	1	A	S_{n1}	S_{n2}	S_{n3}	S_{n4}	S_{n5}

Tabela 1: Estrutura do Multiquadro CRC-4.

Os *bits E* (quadros 13 e 15) são usados para indicar erro no CRC dos submultiquadros recebidos. Para cada submultiquadro recebido com erro, um destes *bits* (na transmissão) deve ser levado de '1' para '0'. O atraso entre a detecção do erro em um submultiquadro e sua sinalização em um dos *bits* deve ser feita em menos de 1 segundo.

Nota: Os bits E devem sempre ser levados em conta, mesmo que o submultiquadro que os contém apresente erro de CRC, pois a probabilidade de estarem errados é pequena.

A palavra de CRC, localizada no submultiquadro ***n***, é o resto da multiplicação por x^4 seguida de divisão (módulo 2) pelo polinômio x^4+x+1, da representação polinomial do submultiquadro ***n-1***. Para efeito da geração da palavra de CRC os *bits* 1 dos quadros 0 e 8 devem ser tomados como os mais significativos de seus respectivos submultiquadros.

O procedimento de decodificação deve seguir as seguintes etapas:

- o submultiquadro recebido passa pelo processo de multiplicação/divisão definido acima, após ter seus *bits* de CRC extraídos e substituídos por '0';
- o resto deste processo é armazenado e posteriormente comparado *bit* a *bit* com os *bits* C_1 a C_4 recebidos no próximo submultiquadro;
- se o resto assim calculado corresponder exatamente à palavra CRC recebida, o submultiquadro é considerado livre de erros.

Quando um dos *bits E* do enlace de recepção for recebido ativo (igual a '0'), o circuito deve se manter transparente a eles, de forma a passar as informações recebidas para frente.

Caso não estejam ativos, o circuito deve inserir nas posições *E* do enlace de saída, o resultado da avaliação do CRC calculado. Em caso de perda de alinhamento de quadro e de multiquadro CRC, deverá ser inibido o cálculo dos *bits E* e a contagem de erros de CRC, e inseridos os *bits E* iguais a '0' no enlace de transmissão.

A cada erro de CRC detectado na recepção devem ser incrementados os registros contadores de erros de CRC (*CECH* e *CECL*, seção 3.1.4). Estes registros formam um contador de 10 *bits* usados para avaliação da taxa de erro na linha e para o fluxograma de acompanhamento do sincronismo de quadro (figura 15). O contador (2 registros) pode ser programado através do *bit* de controle *ctr2.r915*, para funcionar com uma base de tempo fixa de 1s, síncrona com a estrutura de multiquadro CRC (modo automático), ou sob controle da placa controladora (seção 3.1.3).

No primeiro caso o contador é zerado a cada 1000 submultiquadros CRC (1s) e verifica automaticamente dois limiares de contagem de erros de CRC:

- caso a contagem atinja 915: é imediatamente reiniciado o processo de busca de sincronismo de quadro (seção 1.4.2), que implica na ativação de pedido de interrupção via *bit intl.los* (seção 3.1.1);
- caso a contagem atinja o valor programado no registro *LTE*: ao final do período de amostragem é ativado o *bit* de estado de interrupção *intl.ste* (seção 3.1.1).

No segundo caso, a base de tempo é definida pelo processador e cada registro (*CECH* e *CECL*) é zerado ao final do pulso de leitura e, imediatamente, está liberado para iniciar novo período de amostragem. Neste modo o contador, caso atinja 3FFH, deve ser travado até que haja leitura pelo processador.

A figura 14 mostra uma proposta (anexo da Recomendação G-704 do CCITT) de implementação do circuito de CRC usando registro de deslocamento. A entrada I deve receber o submultiquadro serialmente começando pelo *bit* 1 do quadro 0 (ou *bit* 1 do quadro 8) (tabela 1). Os *bits* C_1 a C_4 devem ser forçados para '0'. Quando o último *bit* do submultiquadro (*bit* 256 do quadro 7, ou do quadro 15) entrar no registro, os *bits* C_1 a C_4 estarão disponíveis nas saídas do circuito. Esta palavra de CRC é transmitida no submultiquadro seguinte. As saídas do registro de deslocamento devem ser levadas a '0' ao fim de cada submultiquadro.

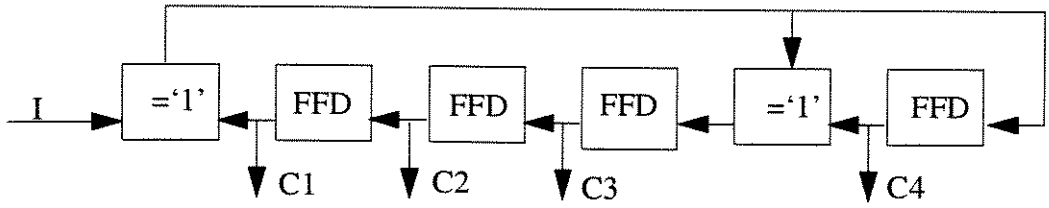


Figura 14: Proposta de Implementação do CRC.

1.4 Gerador de Fases, Sincronismos e Supervisão do Enlace

1.4.1 Gerador de Fases

Este circuito é responsável por gerar os sinais, necessários para a sincronização dos circuitos que dependem da temporização da linha ou da temporização local. Todos os sinais são derivados dos relógios de linha e local (pinos *RRCLK_4M*, *RRCLK_2M* e *SCLK_4M*) e do

sinal de sincronismo de quadro local (pino *SYPQ*).

- Base de tempo de 1s: deve ser gerado sinal com período de 1s, síncrono com a estrutura de multiquadro CRC, para o Circuito Tratador do CRC de Recepção (seção 1.3).
- Base de tempo de 8ms: deve ser gerado sinal com período de 8ms, síncrono com a estrutura de multiquadro CRC, para o circuito Recuperador de Sincronismo (seção 1.4.2).
- Temporização de linha: deve gerar os sinais necessários ao endereçamento das memórias elásticas; recuperação e inserção de *bits* de serviço e CRC do canal 0; e sincronismo do canal 16.
- Temporização local: deve gerar os sinais necessários ao endereçamento das memórias elásticas.
- Falha no relógio de linha de 2 MHz (pino *RRCLK_2M*): uma falha no relógio recebido ocorre quando este permanece estável, por um intervalo de tempo de 4 ou mais períodos de relógio local (*SCLK_4M*). Quando for detectada esta condição, deverá ser ativado o *bit* de estado de interrupção *int1.nck* (seção 3.1.1). O pino *AINT* será ativado caso o *bit mas1.mnck* do registro de máscara não esteja ativo.

1.4.2 Recuperador de Sincronismo

Este bloco é responsável por executar os algoritmos e demais funções definidas na recomendação G-706 do CCITT [6] para determinação do alinhamento de quadro e multiquadro CRC-4.

A palavra de alinhamento de quadro é representada pela sequência binária "0011011", transmitida nos *bits* 2 a 8 do canal 0 dos quadros pares (tabela 1). O alinhamento do quadro deverá ser considerado perdido quando forem recebidas 3 palavras de alinhamento consecutivas erradas.

O alinhamento de quadro deverá ser considerado recuperado quando for detectada a seguinte sequência:

- a presença da palavra de alinhamento de quadro é detectada pela primeira vez (quadro n);
- no quadro seguinte (n + 1), a ausência da palavra de alinhamento de quadro detectada através da verificação de que o *bit* 2 do intervalo de tempo de canal 0 tem valor binário '1';
- no quadro seguinte (n + 2), a presença da palavra de alinhamento de quadro correta pela segunda vez.

No caso de qualquer falha na ocorrência de algum destes requisitos o procedimento de recuperação do alinhamento de quadro deverá ser reiniciado a partir do *bit* imediato.

O alinhamento de quadro, também deverá ser considerado perdido quando não for possível estabelecer o alinhamento de multiquadro CRC, ou quando for excedida a contagem de 915 em 1000 amostras de CRC com erro, como definido a seguir.

O alinhamento de multiquadro CRC será considerado estabelecido se, após atendida a

condição de alinhamento de quadro, no mínimo dois sinais de alinhamento de multiquadro CRC puderem ser localizados dentro de um intervalo de tempo de 8ms. Se o alinhamento de multiquadro não puder ser estabelecido em até 8ms, deverá ser assumido que o alinhamento de quadro foi estabelecido a partir de um sinal de alinhamento de quadro simulado. Neste caso, deverá ser reiniciada a procura do alinhamento de quadro. A figura 15 mostra o procedimento para passar da busca de alinhamento de quadro para o estado de monitoração da estrutura de multiquadro CRC-4.

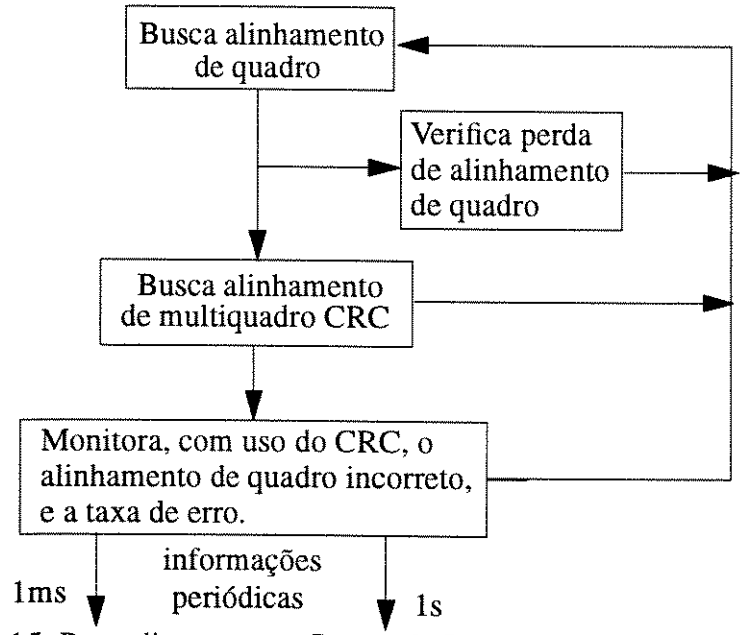


Figura 15: Procedimento para Passar da Busca pelo Alinhamento de Quadro para a Monitoração de Erro com Uso do CRC.

Notas:

O reinício do alinhamento de multiquadro deverá começar imediatamente após o ponto onde o alinhamento de quadro havia sido assumido, prevenindo-se desta maneira contra o realinhamento sobre o sinal de alinhamento simulado;

Uma vez restabelecido o alinhamento de quadro, não mais deverão ser tomadas as ações definidas na tabela 2.

Para efeito de proteção contra falso alinhamento de quadro deve ser monitorado o CRC através do limiar de contagem de 915 blocos CRC errados em 1000, sendo que uma contagem maior ou igual a 915 blocos CRC errados indica um falso alinhamento de quadro, devendo-se então reiniciar a procura do alinhamento de quadro.

A seguir estão resumidas as condições nas quais o circuito deve considerar que não está sincronizado a nível de quadro com o enlace recebido:

- três palavras de alinhamento de quadro, consecutivas, erradas;
- quando em sincronismo de quadro, não for possível alinhar multiquadro;
- 915, ou mais, em 1000 amostras de CRC com erro.

De acordo com o *bit* de controle *ctr2.r915*, o processador habilita ou não a verificação automática das condições de 915/1000 e taxa excessiva de erro (*TEE*), de acordo com o descrito a seguir:

- se *ctr2.r915*='0' o processador habilita a verificação automática das condições de 915/1000 e taxa excessiva de erro (*TEE*).
  A verificação da *TEE*, neste caso, deve ser feita pela comparação do conteúdo do registro *CEC* (contador de erros de CRC) com o registro *LTE* (limiar da taxa excessiva de erro). O registro *LTE* deve ser programado pelo processador. Caso a contagem feita em *CEC*, dentro de uma base de tempo com período de 1s, atinja o valor programado em *LTE*, deve ser gerada interrupção para o processador (*bit int1.ste*, seção 3.1.1).
  A verificação da condição de 915 amostras de CRC erradas em 1000 deve ser feita automaticamente e quando excedida deve provocar a busca de alinhamento de quadro (figura 15). O processador tomará conhecimento da situação através do *bit* de interrupção *int1.los* e do *bit* de estado *int2.cal* (seção 3.1.1);
- se *ctr2.r915*='1', tanto a verificação de 915/1000 quanto da taxa excessiva de erro ficam a cargo do processador. A base de tempo na qual será feita a verificação é definida pelo intervalo entre os acessos de leitura feitos aos registros *CECL* e *CECH* (seção 3.1.4). Neste modo de funcionamento o processador controla a máquina de sincronismo de quadro e multiquadro através do *bit ctr5.frs* (seção 3.1.3).

Obs.: O cálculo de CRC do enlace de entrada e a consequente contagem de erros de CRC só deve ser feita enquanto o circuito estiver em alinhamento de multiquadro.

1.4.3 Supervisão do Enlace Recebido

Alguns blocos do circuito executam funções destinadas à detecção de condições de falha no enlace *PCM* recebido. São as seguintes as condições a serem verificadas:

- perda de sinal na entrada, mesmo que isto não acarrete perda de alinhamento de quadro (*PSE*, seção 1.1.3);
- perda de alinhamento de quadro (*PAQ*, seção 1.4.2);
- identificação do sinal indicativo de alarme remoto (seção 2.3 e tabela 1, onde é tratado como *bit* **A**);
- identificação da indicação de erro de CRC remoto através dos *bits E* (tabela 1 e seção 1.3);
- erro no CRC do enlace recebido (seções 1.3 e 2.6).

Cada uma dessas condições, quando detectada, implica em ações consequentes que estão listadas na tabela 2.

Tipo de Falhas	**Ações tomadas**				
	envia *bit A*	**envia *bits E***	**ativa registro de estado**	**incrementa contador**	**ativa interrupção**
PSE	sim(4)		sim		sim
PAQ	sim(4)		sim		sim
bit A ativo	sim(5)		sim		sim
bits E ativos		sim (1)			

Tabela 2: Ações Tomadas para Falhas no Enlace Recebido.

Tipo de Falhas	Ações tomadas				
	envia *bit A*	envia *bits E*	ativa registro de estado	incrementa contador	ativa interrupção
erro CRC local		sim (2)		sim	sim(3)

Tabela 2: Ações Tomadas para Falhas no Enlace Recebido.

(1) Caso os *bits E* sejam recebidos ativos (iguais a '0'), o *TB47* deve ser transparente, enviando no enlace de transmissão o mesmo valor.

(2) Caso os *bits E* sejam recebidos não ativos (iguais a '1'), o *TB47* deve inserir no enlace de transmissão o resultado da avaliação do CRC relativo ao submultiquadro.

(3) A interrupção é gerada apenas no modo automático (*ctr2.r915*='0'), quando ocorrer:

- saturação do limiar programado pelo processador (registros de controle *LTEL* e *LTEH*), ativando o *bit* de estado *int1.ste* ou,
- detecção de 915 em 1000 amostras de CRC com erro, ativando o *bit* de estado *int1.los*.

(4) Neste caso, o envio do bit *A* tem que ser comandado pelo processador.

(5) Caso o *bit A* do enlace de recepção esteja ativo, o *hardware* é transparente ao mesmo.

Mesmo quando forem detectados erros de CRC em submultiquadros que contém quadros com o *bit A* ativo, a verificação do CRC deve ser executada.

Notas:

A verificação automática da taxa de erro, medida através dos erros de CRC local, deve ser feita pelo processador via programação do bit de controle ctr2.r915='0' e do registro LTE (seção 3.1.3) que estabelece o limiar de saturação do contador de erros de CRC que irá gerar interrupção.

É necessário que seja feita uma persistência local, pelo processador, dos alarmes gerados pela UDL, antes da informação ser enviada para a central. Este tempo deve ser definido a nível de sistema.

Quando as condições de defeito desaparecerem, ou quando o sinal de indicação de falha não for mais recebido, as indicações da falha nos registros de estado devem ser retiradas de imediato (bits ATUAIS, seção 3.1.1).

2 Transmissão do Enlace

É responsável pelas seguintes funções:

2.1 Memória Elástica de Transmissão

O comportamento da memória elástica de transmissão é o mesmo da memória elástica de recepção discutida na seção 1.2, exceto que:

- os dados serão escritos na memória com o relógio local e serão lidos da memória com o relógio recuperado da linha;
- no caso de perda de sincronismo de quadro, a leitura na memória elástica deve continuar na base de tempo anterior. Somente após a recuperação do alinhamento de multiquadro (que pode estar fora de fase com a base de tempo anterior), a operação de

leitura deve ser ressincronizada;

- não será detectada a diferença de fase à nível de *bit* entre os relógios de escrita e leitura desta memória.

2.2 Memória para Controle de Canais

Esta memória contém 32 palavras de 7 *bits* cada (*MC6* a *MC0*). As palavras de 0 a 31 se referem aos canais 0 a 31, respectivamente, do enlace de transmissão. Esta memória é acessada (lida) pelo *hardware*, no começo de cada canal, com o relógio de linha. Quando *ctr2.lre*='1' (enlace de reconfiguração) esta memória não é utilizada. A escrita na memória só é feita pelo processador, que também pode ler seu conteúdo.

Durante a leitura pelo *hardware*, se o processador estiver fazendo acesso à memória, deve-se gerar um sinal de *WAIT* para o processador tão pequeno quanto possível, via pino de saída *WAIT*.

Os *bits MC3* a *MC0* endereçam os *bits B8* a *B1* (*B1* é o *bit* mais significativo) de cada canal do enlace de transmissão, definindo a localização efetiva do canal dentro dos 8 *bits*, uma vez que o *TB47* admite trabalhar com canais a 64, 32 ou 16Kbits/s. Os *bits* podem ser selecionados 2 a 2 possibilitando todas as combinações de alocação dos canais de baixas taxas.

As funções dos *bits MC6* a *MC0* estão apresentadas na tabela 3.

bit da memória de controle	valor do *bit*	
	='0'	='1'
MC6 (*D6*)	Operação normal.	Envia o conteúdo do registro de controle *IDLE* para o enlace de transmissão (64Kbits/s).
MC5 (*D5*)	Operação normal.	*Loop back* interno para efeito de teste. Aconselha-se que este teste seja feito apenas em canais não alocados.
MC4 (*D4*) (só tem sentido se *MC5*='1')	*Loop back* interno do conteúdo do registro *IDLE*.	*Loop back* interno do enlace serial vindo da interface de sistema via pino *XDI*, após passar pela memória elástica de transmissão.
MC3 (*D3*)	*Bits B2* e *B1* não alocados.	*Bits B2* e *B1* alocados.
MC2 (*D2*)	*Bits B4* e *B3* não alocados.	*Bits B4* e *B3* alocados.
MC1 (*D1*)	*Bits B6* e *B5* não alocados.	*Bits B6* e *B5* alocados.
MC0 (*D0*)	*Bits B8* e *B7* não alocados.	*Bits B8* e *B7* alocados.

Tabela 3: Funções dos *Bits* da Memória para Controle de Canais.

2.3 Montagem do Canal 0

A inserção do canal 0 no enlace transmitido deve ser feita a todo intervalo de tempo de canal 0 independente de a *UIL* estar ou não em sincronismo de quadro com o enlace recebido.

A tabela 4 mostra os padrões a serem inseridos nos intervalos de canal 0 pares e ímpares de um submultiquadro CRC.

Canal 0	**Conteúdo**							
par	C_n	0	0	1	1	0	1	1
ímpar	S_m	1	A	S_{n1}	S_{n2}	S_{n3}	S_{n4}	S_{n5}

Tabela 4: Conteúdo dos Canais 0.

Sendo:

C_n: cada um dos 4 *bits* que compõem a palavra de CRC (seções 1.3 e 2.6).

A: indicação de alarme remoto. Em operação normal é '0' e em condição de alarme é '1'. Quando o *bit A* do canal 0 do enlace de recepção já chegar ativo, a *UDL* deve ser transparente a ele, fazendo com que no enlace de saída o *bit A* seja transmitido ativo, independente das condições locais de alarme. Caso contrário, o *bit A* a ser transmitido será inserido sob comando do processador através dos *bits* de controle *ctr2.exra* e *ctr2.vxra* (seção 3.1.3).

S_m: cada um dos *bits* que compõem a palavra de alinhamento de multiquadro CRC-4 ("001011"), ou um dos *bits E* de indicação de erro de CRC (seção 1.3). Quando os *bits E* recebidos (erro de CRC remoto) estiverem ativos ('0'), a *UDL* deve ser transparente a eles. Caso contrário ('1'), serão inseridos os *bits* resultantes da verificação de erro de CRC local.

S_{n1} a S_{n5}: *bits* destinados a serviço. Para eles devem ser previstas memórias distintas para o armazenamento na recepção e para a transmissão (seção 3.1.2). Eles devem ser tratados em conjuntos de 40 *bits* alocados na estrutura de multiquadro CRC (5 *bits* por canal, 8 canais por multiquadro).

2.4 Multiplexador do Enlace de Transmissão

Este circuito é responsável por montar o enlace *PCM* de transmissão. Sua função é dependente do *bit* de controle *ctr2.lre* (seção 3.1.3). Quando for '1', enlace de reconfiguração, o *TB47* deve enviar para a linha o enlace de saída da memória elástica de transmissão, conforme descrito na seção 5 e representado na figura 19.

Quando *ctr3.lbme*='1' (*loop* de memória elástica), o *TB47* também deve enviar para a linha o enlace de saída da memória elástica de transmissão, porém, o canal 0 deve continuar sendo inserido como em operação normal, conforme descrito na seção 2.3.

O canal 16 a ser transmitido (pino de entrada *CSTR*) tem um atraso de um período de 2MHz em relação aos demais canais. Para possibilitar o acerto da temporização com os demais canais durante a montagem do enlace, o *TB47* deve compensar internamente este atraso.

No modo normal de funcionamento, *ctr2.lre*='0', devem ser multiplexados diversos sinais que juntos formam o enlace a ser transmitido, isto deve ser feito por partes:

- CANAL 0: esperar o intervalo de tempo de canal 0 e inserí-lo conforme descrito na seção 2.3.
- CANAIS ALOCADOS: esperar intervalo de tempo (do relógio de linha) de canais alocados e inserir os dados lidos da memória elástica de transmissão.
- CANAIS NÃO ALOCADOS: esperar intervalo de tempo (do relógio de linha) de canais não alocados e inserir os dados vindos do enlace de recepção, no formato unipolar (seção 1.1).

Obs.: A informação sobre canais alocados e não alocados está na memória de controle de canais, seção 2.2. Os canais alocados podem estar na taxa de 64, 32 ou 16Kbits/s. Portanto, pode-se alocar apenas alguns bits de um canal, sendo que os demais bits serão tratados como não alocados.

- INSERÇÃO DE CONTEÚDO PROGRAMÁVEL: no intervalo de tempo de qualquer canal que esteja programado (memória de controle) para inserção de conteúdo programável (seção 2.5), deve ser inserido o padrão definido no registro *IDLE*. É possível inserir este padrão até mesmo nos canais 0 e 16. Para este último é preciso programar também os *bits ctr3.c16b* e *ctr3.c16a* (seção 3.1.3).
- CANAL 16: esperar intervalo de tempo (do relógio de linha) de canal 16, e inserir o canal 16 vindo da interface de sistema via pino *CSTR* (seção 7), com programação dos *bits ctr3.c16b* e *ctr3.c16a*, do registro de controle (seção 3.1.3).

A saída desta primeira parte do multiplexador será então o resultado da multiplexação dos canais alocados, não alocados, canal 0 (atrasados de um período do relógio de linha) e do canal 16. Todos com a possibilidade de terem o conteúdo do registro *IDLE* inserido. Vide o sinal *canal16_voz*, na figura 19.

A esta altura o enlace está praticamente montado, faltando passar pelo cálculo e inserção dos *bits* de CRC (seção 2.6 - vide o sinal *reg_CRC_tx0*, na figura 19).

2.5 Inserção de Conteúdo Programável

O *TB47* deve possibilitar a inserção de um *byte* programável em qualquer canal. Os canais nos quais devem ser inseridos estarão definidos na memória de controle de canais, conforme descrito na seção 2.2 e representado nas figuras 18 e 19.

A inserção pode ser:

- para dentro da *UDL* (em caso de teste de continuidade interna dos canais da *UDL* - *loop back* interno, seção 4.4),
- para o enlace de transmissão (seção 2.4), ou
- tanto para dentro da *UDL* quanto para o enlace de transmissão.

O conteúdo do *byte* a ser inserido no enlace de transmissão é definido no registro de controle *IDLE*. No caso de *loop back* interno, o padrão a ser inserido em cada canal selecionado será o definido ou pelo registro de controle *IDLE* ou pelo enlace serial vindo da interface de sistema via pino *XDI*, após ter passado pela memória elástica de transmissão.

A programação de um canal implica no estabelecimento de prioridade para a inserção

do conteúdo selecionado sobre qualquer outro conteúdo.

2.6 Tratador de CRC de Transmissão

A estrutura de multiquadro CRC na transmissão é idêntica à definida para a recepção na seção 1.3. A sequência de procedimentos para a codificação e inserção da palavra de CRC no enlace é:

- forçar '0' nas posições relativas aos *bits* C_1 a C_4 do submultiquadro;
- executar o processo de multiplicação/divisão definido na seção 1.3;
- armazenar o resto obtido deste processo para posterior inserção nas respectivas posições dos *bits* C_1 a C_4 do submultiquadro seguinte.

2.7 Interface de Saída para a Linha

É composta de dois sub-blocos, descritos a seguir, e quatro pinos: *XDOP* e *XDOM*, para o enlace codificado em HDB3; *XOID*, para o enlace em formato RZ ou NRZ, embaralhado ou não, respectivamente; e o pino *XULR* para o enlace em formato NRZ, não embaralhado.

Este último pino é necessário pois a definição da funcionalidade de *XOID* é dada pelo *hardware* da placa através do pino *POE3* e, para o caso de *loop* de reconfiguração, independente desta programação, é necessário que a *UIL* A passe o enlace a ser transmitido para a *UIL* B, sem estar embaralhado.

Os sinais de saída do *TB47* derivados deste bloco, são síncronos com o relógio de linha (*RRCLK_4M*) quando a *UIL* está no modo funcionamento normal, e síncronos com o relógio local (*SCLK_4M*) quando a *UIL* está no modo *loop* de reconfiguração (seção 5).

2.7.1 Codificador HDB3

Este bloco deve executar a codificação em código de transmissão ternário HDB3 (*High Density Bipolar 3*), de acordo com a Recomendação G-703 do CCITT [6]. A codificação HDB3 feita a partir de um sinal binário de entrada no formato unipolar NRZ, gera dois sinais binários no formato unipolar RZ como saída (pinos *XDOP* e *XDOM*).

2.7.2 Embaralhador

O pino de saída deste bloco, *XOID*, apresenta o enlace a ser transmitido para a linha no formato RZ ou NRZ, dependendo do pino *POE3* (figura 21):

- caso o pino *POE3*='1', o sinal de saída passa pelo embaralhador e é apresentado em formato RZ;
- caso o pino *POE3*='0', o sinal de saída é apresentado em formato NRZ, sem passar pelo embaralhador.

Este bloco deve adotar o circuito embaralhador cujo polinômio gerador é x^7+x^6+1, definido na Recomendação G-709 do CCITT [6].

2.8 Inserção de *AIS - Alarm Indication Signal* (tudo '1')

A penúltima etapa pela qual passa o enlace a ser transmitido, antes de ir para o

codificador HDB3 (e para o pino *XOID*, saída óptica) é a possibilidade de inserção, ou não, de tudo '1', sob controle do *bit ctrl.xais* (seção 3.1.3). Caso o *bit* seja igual a '1' todo o enlace montado deve ser ignorado e a linha deve receber '1', caso o *bit* seja igual a '0', transmite-se o enlace montado.

O enlace após passar por este estágio ficará disponível no formato RZ (embaralhado) ou NRZ (não embaralhado) via pino de saída *XOID*, e codificado no formato HDB3 (RZ) nos pinos de saída *XDOP* e *XDOM*.

2.9 Controle do *Bypass*

Através do registro de controle (*ctrl.byp*) existe a possibilidade de tornar a *UIL* transparente *para o enlace (bypass* na *UIL*). Esta função serve para, por exemplo, prevenir a interrupção de todo um enlace em caso de falha interna na *UDL* (figura 19).

Quando em *bypass* o *TB47* deve apenas prosseguir fazendo uso das interfaces de entrada e saída selecionadas pelos pinos *POE1* a *POE3*. Todas as demais funções do circuito devem ser contornadas pelo enlace em trânsito.

No funcionamento normal do *TB47* certamente será imposto ao enlace um atraso de alguns *bits*, em função dos circuitos das interfaces de entrada e saída e do circuito de *delay* para compensar o atraso do canal 16 no sistema. Com o objetivo de manter o enlace de transmissão quando em *bypass*, síncrono com o enlace de transmissão quando em funcionamento normal, deve-se prever no caminho do *bypass* o mesmo atraso em períodos de 2MHz.

O *bypass* também deverá ser ativado quando o pino de entrada *SIPA*='1', que indica a ausência do processador.

O *bypass*, comandado tanto pelo *bit* de controle *ctrl.byp* quanto pelo pino *SIPA*, tem prioridade sobre qualquer outro *bit* de controle que esteja ativo e sempre trabalha com o relógio de linha.

Devido à possibilidade do uso de interface de entrada de tipo diferente da interface de saída (por exemplo: entrada via pino *ROID* e saída via pinos *XDOP* e *XDOM*) o atraso provocado no enlace quando em modo *bypass* é variado e dependente das interfaces em uso. Estas são definidas pelos pinos *POE1* a *POE3*. Em qualquer caso, porém, o atraso provocado quando em *bypass* deve ser exatamente o mesmo observado no uso das interfaces selecionadas.

3 Interface com a Placa Controladora

É responsável pelas seguintes funções:

3.1 Registros para o Processador

A seleção de registros do *TB47* pelo processador que controla o sistema é feita pelos pinos de endereço, dados, escrita, leitura e seleção. Através deles o processador define a transação e endereça o registro a ser acessado para escrita ou leitura. No barramento de dados, nesta hora, transita o conteúdo a ser escrito no (ou que foi lido do) registro endereçado.

Os registros do *TB47* são divididos em duas categorias: os de controle (seções 3.1.2 e

3.1.3) e os de estado (seções 3.1.1 e 3.1.4). Através de escritas nos registros de controle o processador programa o *TB47*. Pela leitura dos registros de estado o processador toma conhecimento da situação do circuito. Todos os registros de controle podem ter seu conteúdo lido pelo processador com o mesmo endereço de escrita.

Após o *reset* ser ativado, o estado de cada *bit* dos registros para o processador é '0'. Aqueles que fogem a esta regra têm seu valor inicial anotado nas seções onde são definidos.

A tabela 5 mostra todos os registros necessários para este circuito, com seus endereços.

Endereços (*A6* a *A0*)	Mnem.	escr/ leit	leit s/ *reset*	escr p/ *reset*	leit c/ *reset*	Comentários
0000000		X				*Byte* 0 da Memória de Controle
0000001		X				*Byte* 1 da Memória de Controle
0000010		X				*Byte* 2 da Memória de Controle
...		X				*Byte* ... da Memória de Controle
0011111		X				*Byte* 32 da Memória de Controle
0100000	*SNR1*	X				Registro 1 S_n de Recepção
0100001	*SNR2*	X				Registro 2 S_n de Recepção
0100010	*SNR3*	X				Registro 3 S_n de Recepção
0100011	*SNR4*	X				Registro 4 S_n de Recepção
0100100	*SNR5*	X				Registro 5 S_n de Recepção
0100101	*SNR6*	X				Registro 6 S_n de Recepção
0100110	*SNR7*	X				Registro 7 S_n de Recepção
0100111	*SNR8*	X				Registro 8 S_n de Recepção
0101000	*SNT1*	X				Registro 1 S_n de Transmissão
0101001	*SNT2*	X				Registro 2 S_n de Transmissão
0101010	*SNT3*	X				Registro 3 S_n de Transmissão
0101011	*SNT4*	X				Registro 4 S_n de Transmissão
0101100	*SNT5*	X				Registro 5 S_n de Transmissão
0101101	*SNT6*	X				Registro 6 S_n de Transmissão
0101110	*SNT7*	X				Registro 7 S_n de Transmissão
0101111	*SNT8*	X				Registro 8 S_n de Transmissão
0110000	*CTR1*	X				Registro de Controle 1

Tabela 5: Endereços de Acesso do Microprocessador.

Endereços (*A6* a *A0*)	Mnem.	escr/ leit	leit s/ *reset*	escr p/ *reset*	leit c/ *reset*	Comentários
0110001	*CTR2*	X				Registro de Controle 2
0110010	*CTR3*	X				Registro de Controle 3
0110011	*MAS1*	X				Registro Máscara das Fontes de Interrupção
0110100	*LTEL*	X				Reg. Limiar Taxa Excessiva de Erro (baixa)
0110101	*LTEH*	X				Reg. Limiar Taxa Excessiva de Erro (alta)
0110110	*IDLE*	X				Registro com o Código de Canal *Idle*
0110111	*DFI*		X			Diferença de fase entre relógios linha e local
0111000	*int1.snv*			X		*Bit* semáforo.
0111001	*CECL*		X			*Bits* Menos Significativos Cont. Erros CRC
0111010	*INT1*				X	Registro de Interrupção 1
0111011	*CECH*		X			*Bits* Mais Significativos Cont. Erros CRC
0111100	*INT2*				X	Registro de Interrupção 2
0111101	*INT2*		X			Registro de Interrupção 2
0111110	*INT1*		X			Registro de Interrupção 1
0111111	*PROG*		X			Pinos de controle e presença da placa
1000000	*CTR4*					Registro de Controle 4
1000001	*CTR5*					Registro de Controle 5

Tabela 5: Endereços de Acesso do Microprocessador.

Nas seções a seguir estão definidos cada *bit* dos registros.

3.1.1 Registros de Estado de Interrupção

A figura 16 mostra um fluxograma para o tratamento de interrupções.

A tabela 5 lista os endereços dos registros de tratamento de interrupção e o *bit int1.snv*.

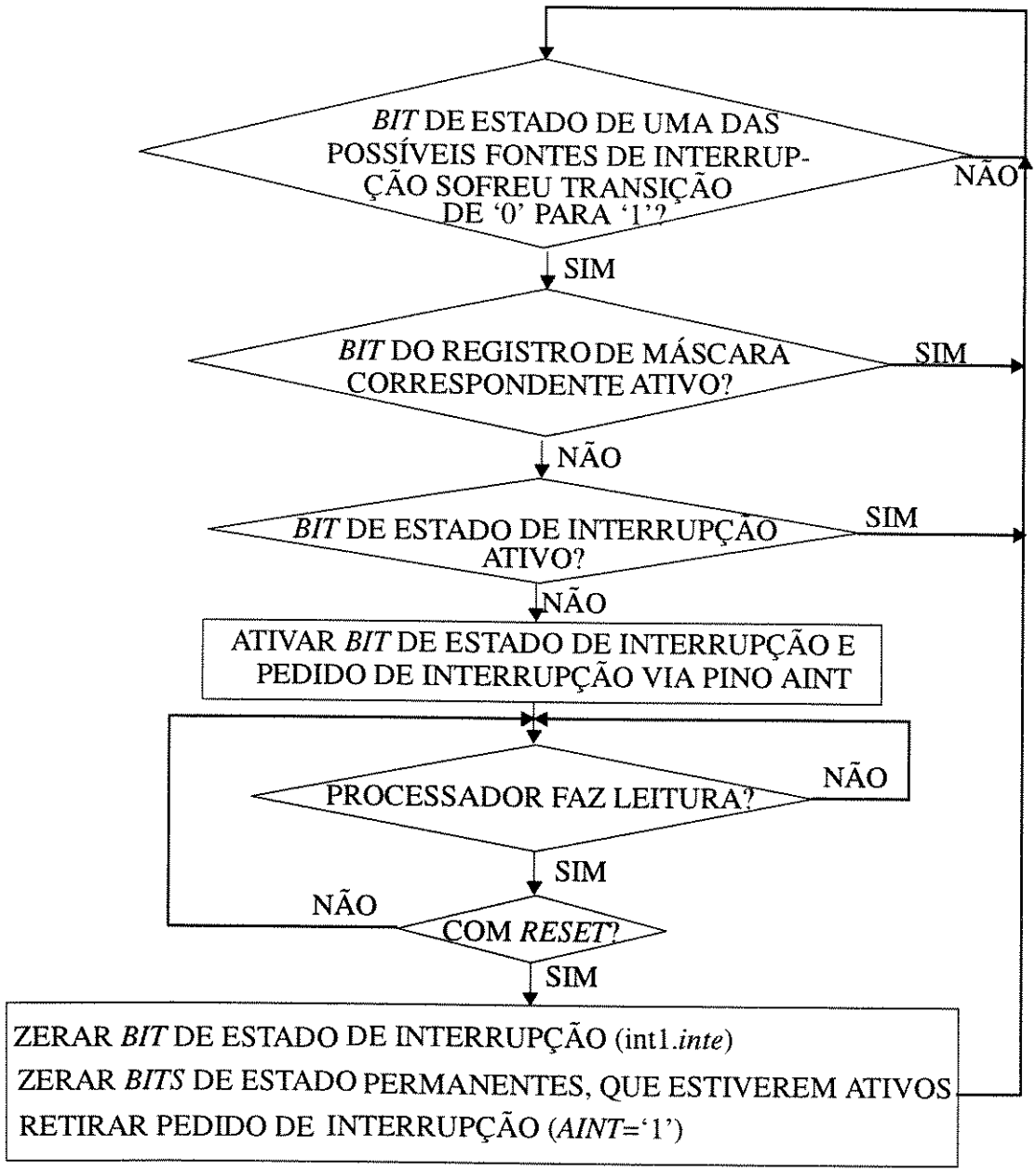


Figura 16: Fluxograma para o Tratamento de Interrupções.

Existem três tipos de *bit* de estado de interrupção:

- *BITS* ATUAIS: são os *bits* que sempre refletem a situação atual do *hardware*, portanto só são atualizados pelo *hardware*, sem interferência do processador. Estes *bits* são *next*, *los*, *nos*, *rra* e *nck*.
- *BIT* SEMÁFORO: sua função está descrita no registro *int1.snv*.
- *BITS* PERMANENTES: são os *bits* que ao serem ativados devem gerar um pedido de interrupção (desde que não estejam mascarados) e permanecer no mesmo valor até que o processador execute uma leitura com *reset* quando, no final do pulso de leitura, este *bit* é zerado pelo *hardware*.

Estes *bits* são *inte*, *epm*, *ste* e *slp*.

Para os *bits* permanentes também deve ser prevista a condição de o processador fazer leitura, fora da rotina de tratamento de interrupção. Neste caso não deve ser alterado o conteúdo do *bit*, ou seja, deve ser feita uma leitura sem *reset*.

Para verificação da integridade, durante o funcionamento normal do *TB47*, do registro de interrupção, está prevista a função simulação de alarmes (seção 4.8).

Todos os *bits* de estado que constituem fontes de interrupção devem poder ser mascarados via registro de controle *MAS1* (seção 3.1.3).

O *bit int1.next* representa um OU lógico dos *bits* de estado 7 e 6 do registro *INT2*, caso não estejam mascarados. Através do *bit int1.next* o *software*, ao ler o registro *INT1*, fica sabendo se há *bits* ativos no registro *INT2*. O *bit int1.inte* representa um OU das transições ('0' para '1' ou '1' para '0', de acordo com o *bit* de controle *ctr2.cbi* - seção 3.1.3) dos *bits* de estado 6 a 0 do registro *INT1*, ou seja, armazena o resultado de um OU lógico sempre que ocorrer uma transição de algum dos *bits* de estado que não estejam mascarados. Estes dois *bits* (*int1.next* e *int1.inte*) não constituem fontes de interrupção.

O *bit int2.cal* também não constitue fonte de interrupção.

Os demais *bits* destes dois registros representam todas as possíveis fontes de interrupção.

Sempre que o processador atender um pedido de interrupção (leitura com *reset*), o *hardware* automaticamente no final do pulso de leitura zera o *bit* int1.*inte* e retira o pedido de interrupção via pino de interrupção *AINT*.

Na leitura sem *reset*, o *bit int1.inte* e o pedido de interrupção (pino *AINT*) não podem ser retirados.

A tabela 6 lista todos os registros de interrupção do *TB47*.

Mnemônico	D7	D6	D5	D4	D3	D2	D1	D0
INT1	*inte*	*ste*	*snv*	*rra*	*los*	*nos*	*nck*	*next*
INT2	*slp*	*epm*	-	-	-	-	-	*cal*

Tabela 6: Registros de Interrupção.

A seguir estão descritos cada *bit* dos registros de interrupção *INT1* e *INT2*.

1. *INT1.INTE*: Interrupção

Este *bit* representa:

- OU das transições positivas de qualquer um dos *bits* permanentes dos registros de interrupção, *INT1* e *INT2* que não estejam mascarados, com exceção do próprio *int1.inte*.
- OU das transições dos *bits* atuais que não estejam mascarados, com exceção do *int1.next*, de acordo com o *bit* de controle *ctr2.cbi*, conforme descrito abaixo:
  - caso *ctr2.cbi*='0', deve representar um OU das transições positivas;
  - caso *ctr2.cbi*='1', deve representar um OU de qualquer transição.

Sempre que for para '1' vai permanecer neste valor até que o *hardware* o leve para '0'

imediatamente após um pulso de leitura com *reset*. Nas leituras sem *reset* seu valor não é alterado.

2. *INT1.STE*: Saturação do Contador de Erros de CRC por Taxa Excessiva de Erro

Só tem sentido quando *ctr2.r915*='0'. Será feito igual a:

0 ... pelo *hardware* imediatamente após um pulso de leitura com *reset*.

1 ... quando for atingido o limiar de contagem definido no registro de controle *LTE*, se *ctr2.r915*='0' (modo automático).

3. *INT1.SNV*: Conjunto de Registros S_n Válido

Será feito igual a:

0 ... pelo processador, através de escrita no endereço definido na tabela 5 (o barramento de dados é ignorado); ou automaticamente pelo *hardware* no começo do quadro 15 de cada multiquadro CRC recebido, se *ctr1.rsn*='0' (modo livre).

1 ... pelo *hardware* quando os 40 *bits* S_n recebidos forem copiados para uma memória interna e estiver válida para leitura pelo processador, conforme descrito na seção 3.1.2.

4. *INT1.RRA*: Alarme Remoto Recebido

0 ... se o *bit* 3 da palavra de serviço recebida estiver em '0'.

1 ... se o *bit* 3 da palavra de serviço recebida estiver em '1' (alarme ativo).

5. *INT1.LOS*: Perda de Sincronismo de Quadro

0 ... imediatamente após a recuperação do sincronismo de quadro.

1 ... enquanto estiver na condição de busca de sincronismo de quadro, conforme as condições descritas na seção 1.4.2.

6. *INT1.NOS*: Indicação de Perda do Sinal de Entrada

0 ... enquanto não for detectada a condição de alarme.

1 ... enquanto for detectada perda do sinal de entrada, conforme descrito na seção 1.1.3.

Após a ressincronização ser recuperada (*int1.los*='0'), *int1.nos* deve ser ignorado por 250μs.

7. *INT1.NCK*: Falta do Relógio de Linha

0 ...enquanto não for detectada a condição de alarme.

1 ... enquanto for detectada perda de relógio, conforme descrito na seção 1.4.1.

8. *INT1.NEXT*: Registro *INT2* Ativo

Este *bit* representa um OU lógico dos *bits* de estado das fontes de interrupção, não mascaradas, do registro *INT2*. Este *bit* será automaticamente atualizado pelo *hardware*.

9. *INT2.SLP*: Ocorrência de *Slip* na Memória Elástica de Recepção ou de Transmissão

Será feito igual a:

0 ... pelo *hardware* imediatamente após um pulso de leitura com *reset*.

1 ... quando a diferença entre as frequências do relógio de linha *RRCLK_4M* e o relógio

local *SCLK_4M* resultarem em uma repetição ou eliminação de um quadro recebido, conforme descrito na seção 1.2.

10. *INT2.EPM*: Erro de Paridade na Memória Elástica de Recepção ou de Transmissão

Será feito igual a:

0 ... pelo *hardware* imediatamente após um pulso de leitura com *reset*.

1 ... a cada canal, quando na leitura for detectado um erro de paridade na memória elástica de recepção ou de transmissão, conforme descrito nas seções 1.2 e 2.1.

A perda de sincronismo de multiquadro bloqueia a ativação deste sinal.

11. *INT2.CAL*: Perda de Alinhamento de Multiquadro CRC-4

0 ... imediatamente após a recuperação do sincronismo de multiquadro.

1 ... enquanto o circuito estiver na condição de busca de alinhamento de quadro ou busca de alinhamento de multiquadro, conforme descrito na seção 1.4.2.

3.1.2 Memórias para *Bits* S_n

São *bits* destinados a serviço que trafegam nos canais 0 ímpares, nos *bits* 4 a 8. Para eles devem ser previstos conjuntos de registros distintos para o armazenamento na recepção e para a transmissão. Eles devem ser tratados em conjuntos de 40 *bits* alocados na estrutura de multiquadro CRC-4 (5 *bits* por canal, 8 canais por multiquadro) conforme mostra a tabela 1.

A organização de cada conjunto de registros é mostrada na tabela 7. Do ponto de vista do processador, cada conjunto é organizado em 8 *bytes*, nos quais os 3 *bits* mais significativos não são utilizados. Ao conjunto de recepção, no modo funcional, só é permitido o acesso do processador para leituras (exceto se ctr3.esnr='1'), já o conjunto de registros de transmissão pode ser escrito ou lido pelo processador.

Bits do barram de dados	escrita/leitura referente ao quadro #/palavra # de acesso pelo processador							
	q1/p1	q2/p2	q3/p3	q4/p4	q5/p5	q6/p6	q7/p7	q8/p8
0	S_{n1}	S_{n1}	S_{n1}	S_{n1}	S_{n1}	S_{n1}	S_{n1}	S_{n1}
1	S_{n2}	S_{n2}	S_{n2}	S_{n2}	S_{n2}	S_{n2}	S_{n2}	S_{n2}
2	S_{n3}	S_{n3}	S_{n3}	S_{n3}	S_{n3}	S_{n3}	S_{n3}	S_{n3}
3	S_{n4}	S_{n4}	S_{n4}	S_{n4}	S_{n4}	S_{n4}	S_{n4}	S_{n4}
4	S_{n5}	S_{n5}	S_{n5}	S_{n5}	S_{n5}	S_{n5}	S_{n5}	S_{n5}

Tabela 7: Organização dos *Bits* S_n nos Conjuntos de Recepção/Transmissão.

Do ponto de vista dos circuitos de recepção e transmissão os conjuntos são organizados como 8 registros de deslocamento com 5 *bits* cada. Na recepção o conjunto é preenchido serialmente e na transmissão o registro é enviado serialmente, ambos em fase com a temporização de linha e durante um multiquadro CRC-4. Note que os *bits* são escritos/lidos pelos circuitos de recepção/transmissão serialmente na direção das colunas da tabela 7 e a leitura/escrita pelo processador é feita *byte* a *byte*, no mesmo sentido. Na recepção estes registros são de estado e na transmissão são de controle.

Os endereços pelos quais o processador acessa cada uma das palavras estão definidos na tabela 5.

RECEPÇÃO (registros de estado):

Na recepção devem ser previstos dois modos de funcionamento, selecionáveis pelo *bit* de controle *ctr1.rsn* (seção 3.1.3):

- **livre ('0')**: o processador é interrompido ao final de cada multiquadro (*bit int1.snv*='1') e tem até 2ms para atender ao pedido de interrupção, lendo os 8 *bytes* e a seguir fazendo, via escrita no endereço "111000", o *bit int1.snv*='0'. Tendo o processador lido ou não os *bytes* S_n, a interrupção é retirada automaticamente pelo circuito (*bit int1.snv*='0') no início do quadro 15 de todo multiquadro.

  Obs.: A retirada pelo TB47 da sinalização feita através do bit int1.snv='0', não implica na retirada do pedido de interrupção que permanecerá ativo até esta ser atendida (via uma "leitura com reset"). A leitura do registro deve ser evitada caso int1.snv='0', pois seu conteúdo será alterado no início de todo multiquadro.

- **congelado ('1')**: caso o *bit int1.snv*='0', o circuito aguarda o início do próximo multiquadro e passa a preencher os registros com os *bits* S_n recebidos. Ao final deste multiquadro o processador é interrompido (*bit int1.snv*='1') e o conteúdo dos registros fica congelado até que o processador autorize novo prenchimento fazendo o *bit int1.snv*='0'.

  Obs.: Se o bit de interrupção int1.snv estiver mascarado e o processador for habilitá-lo, ele primeiro deve forçar int1.snv='0' para que o conteúdo dos registros, na primeira interrupção gerada, corresponda ao valor mais recente dos bits S_n recebidos. E, caso o processador não deseje trabalhar com interrupção, pode deixar o bit int1.snv mascarado (só a interrupção não é gerada) e fazer a monitoração dos registros através de polling neste bit.

Para efeito de teste do registro é prevista a escrita pelo processador nas memórias S_n de recepção. A escrita é habilitada pelo próprio processador ativando o *bit* de controle *ctr3.esnr* (seção 3.1.3).

TRANSMISSÃO (registros de controle):

Na transmissão o circuito tem dois modos de operação controlados pelo *bit ctr1.esn* (seção 3.1.3):

- **transparente ('0')**: deve inserir nas respectivas posições do enlace de transmissão os *bits* S_n recebidos pelo enlace de recepção.
- **controlado ('1')**: são inseridos os *bits* dos registros S_n de transmissão, conforme definido na tabela 7. Ver *bit ctr4.txsn* na seção 3.1.3.

3.1.3 Registros de Controle

A tabela 8 lista todos os registros de controle do *TB47* (figuras 18 e 19). Todos os registros de controle podem ser lidos pelo processador no mesmo endereço de escrita.

Mnemônico	D7	D6	D5	D4	D3	D2	D1	D0
CTR1	*rsn*	*esn*	-	*byp*	*xais*	*sais*	*xdos*	*rdis*
CTR2	*cbi*	*fepr*	*fept*	*lre*	*r915*	-	*exra*	*vxra*
CTR3	*c16b*	*c16a*	*emsn*	*esnr*	*lbcs*	*lbme*	*sim*	*crci*
MAS1	*mste*	*msnv*	*mrra*	*mlos*	*mnos*	*mnck*	*mslp*	*mepm*
IDLE	*idl7*	*idl6*	*idl5*	*idl4*	*idl3*	*idl2*	*idl1*	*idl0*
LTEL	*lte7*	*lte6*	*lte5*	*lte4*	*lte3*	*lte2*	*lte1*	*lte0*
LTEH	-	-	-	-	-	-	*lte9*	*lte8*
CTR4	-	-	-	-	-	-	-	*txsn*
CTR5	-	-	-	-	-	-	-	*frs*

Tabela 8: Registros de Controle.

A seguir estão descritos cada um dos registros de controle.

1. *CTR1.RSN*: Modo de Funcionamento das Memórias S_n de Recepção

Vide descrição dos modos na seção 3.1.2.

0 ... modo livre.

1 ... modo congelado.

2. *CTR1.ESN*: Modo de Funcionamento das Memórias S_n de Transmissão

Vide descrição dos modos na seção 3.1.2.

0 ... modo transparente.

1 ... modo controlado.

3. *CTR1.BYP*: *Bypass* do Circuito

0 ... operação normal.

1 ... habilita *bypass* do circuito conforme descrito na seção 2.9 (**ativo logo após o *reset***).

4. *CTR1.XAIS*: Transmissão de *AIS* no Enlace de Transmissão

0 ... operação normal (seção 2.4).

1 ...insere *AIS* para o enlace de transmissão, na entrada do circuito Interface de Saída para a Linha (seção 2.7).

5. *CTR1.SAIS*: Envia *AIS* em Direção à Interface de Sistema

0 ... operação normal (seção 3.3).

1 ... envia *AIS* na interface de sistema via o pino de saída *RDO* (**ativo logo após o *reset***).

6. *CTR1.XDOS*: Polaridade do Dado de Saída a ser Transmitido

0 ... os pinos de saída *XDOP* e *XDOM* são ativos baixo. O pino de saída *XOID* é ativo alto.

1 ... os pinos de saída *XDOP* e *XDOM* são ativos alto. O pino de saída *XOID* é ativo baixo.

7. *CTR1.RDIS*: Polaridade do Dado de Entrada Recebido

0 ... pinos de entrada *RDIP* e *RDIM* ativos baixo. Pino de entrada *ROID* ativo alto.

1 ... pinos de entrada *RDIP* e *RDIM* ativos alto. Pino de entrada *ROID* ativo baixo.

8. *CTR2.CBI*: Controle da Borda Geradora de Interrupção

0 ... os sinais *AINT* e *ctr1.inte*, de interrupção, são acionados a partir das bordas positivas dos sinais geradores da interrupção.

1 ... os sinais *AINT* e *ctr1.inte*, de interrupção, são acionados a partir das bordas positivas dos *bits* permanentes dos registros de interrupção, além das bordas positivas e negativas dos *bits* atuais dos registros de interrupção.

9. *CTR2.FEPR*: Força Erro de Paridade na Memória Elástica de Recepção

0 ... o sinal de saída do bloco gerador de paridade da recepção, é escrito no *bit* 9 da memória elástica de recepção.

1 ... o sinal de saída do bloco gerador de paridade da recepção, é invertido antes de ser escrito no *bit* 9 da memória elástica de recepção.

10. *CTR2.FEPT*: Força Erro de Paridade na Memória Elástica de Transmissão

0 ... o sinal de saída do bloco gerador de paridade da transmissão, é escrito no *bit* 9 da memória elástica de transmissão.

1 ... o sinal de saída do bloco gerador de paridade da transmissão, é invertido antes de ser escrito no *bit* 9 da memória elástica de transmissão.

11. *CTR2.LRE*: *Loop* de Reconfiguração do Enlace (seção 5, figura 17).

0 ... operação normal (*UIL* A).

1 ... modo *loop* de reconfiguração (*UIL* B).

Obs.: Este modo só tem sentido, dentro de uma UDL que faz o loop de reconfiguração do enlace, com uma UIL configurada no modo normal ctr2.lre='0' e outra com ctr2.lre='1'.

12. *CTR2.R915*: Ressincronização 915 em 1000 Automática

0 ... habilita a base de tempo interna de 1s (seção 1.4.2).

1 ... a base de tempo é controlada pelo processador.

13. *CTR2.EXRA*: Habilitação para Transmissão do Alarme Remoto

0 ... o *hardware* é transparente para o alarme remoto recebido.

1 ... envia o valor programado no *bit ctr2.vxra*, no *bit* 3 do canal 0 dos quadros ímpares, até que seja reprogramado para '0'.

14. *CTR2.VXRA*: Valor a ser Transmitido para o Alarme Remoto

Só tem sentido se *ctr2.exra*='1'.

0 ... envia '0' no *bit* 3 de todas as palavras de serviço.

1 ... envia '1' no *bit* 3 de todas as palavras de serviço.

15. *CTR3.C16B* e *CTR3.C16A*: Taxa de Transmissão do Canal 16

Estes dois *bits* de controle definem o que deve ser transmitido no intervalo de tempo do canal 16, assim como permitem que os intervalos de tempo dos canais 17 e 18 sejam usados, da mesma forma que o canal 16, para sinalização entre processadores. Devem ser codificados como definido na tabela 9.

c16b	*c16a*	Modo de Funcionamento
0	0	no intervalo de tempo de canal 16, insere o enlace montado internamente
0	1	no intervalo de tempo de canal 16, insere o sinal vindo do pino *CSTR*
1	0	no intervalo de tempo dos canais 16 e 17, insere o sinal vindo do pino *CSTR*
1	1	no intervalo de tempo dos canais 16, 17 e 18, insere o sinal vindo do pino *CSTR*

Tabela 9: Modo de Funcionamento dos *Bits* de Controle *ctr3.c16b* e *ctr3.c16a.*

16. *CTR3.EMSN*: Endereçamento das Memórias S_n de Recepção

Só tem sentido se *ctr3.esnr*='1'.

0 ... endereça a memória 1 para o processador escrever.

1 ... endereça a memória 2 para o processador escrever.

17. *CTR3.ESNR*: Escrita nas Memórias S_n de Recepção (para Teste pelo Processador)

0 ... operação normal.

1 ... habilita a escrita pelo processador nas memórias S_n de recepção.

18. *CTR3.LBCS*: *Loop Back* do Canal de Sinalização (16)

0 ... operação normal.

1 ... habilita *loop back* interno do canal 16 (seção 4.3).

19. *CTR3.LBME*: *Loop* de Memória Elástica

0 ... operação normal.

1 ... habilita *loop* de memória elástica de recepção para memória elástica de transmissão (seção 4.2).

20. *CTR3.SIM*: Simulação de Alarmes

0 ... operação normal.

1 ... habilita a simulação de alarmes (seção 4.8).

21. *CTR3.CRCI*: Inserção do CRC de Transmissão Invertido no Enlace de Transmissão

0 ... operação normal.

1 ... insere o CRC de transmissão invertido no enlace de transmissão (seção 4.5).

Obs.: ctr3.crci permite realizar testes dos circuitos detectores de erro de CRC no sistema em funcionamento.

22. *CTR4.TXSN*: Transmissão do Conjunto de Registros S_n

Bit semáforo. Só tem sentido se *ctr1.esn*='1'.

0 ... (ativado pelo *hardware*) o conjunto de registros S_n de transmissão já foi transmitido, o que significa que pode voltar a ser preenchido pelo processador, conforme descrito na seção 3.1.2. Esta condição é forçada pelo *hardware* após ter transmitido todo o conjunto de registros S_n.

1 ... (ativado pelo processador) o conjunto de registros S_n está válido para transmissão.

Obs.: Imediatamente após uma mudança de modo de funcionamento de transparente para controlado, enquanto o bit ctr4.txsn, que sinaliza que o registro está válido para transmissão, permanecer em '0' o TB47 deve prosseguir funcionando como no modo transparente. Após a primeira ida de ctr4.txsn para '1' (pelo processador), este bit é levado para '0' pelo TB47 e o conteúdo do registro é transmitido repetidamente até que o processador novamente sinalize um conteúdo válido no registro (fazendo ctr4.txsn='1'). O TB47 deve fazer a comutação entre o conteúdo anterior e o novo a ser transmitido, em sincronismo com a estrutura de multiquadro do enlace. O processador deve fazer ctr4.txsn='1' após ter programado nos registros S_n de transmissão o conjunto de bits a ser enviado.

23. *CTR5.FRS*: Força Ressincronização

Este *bit* só tem sentido quando *ctr2.r915*='1'. É feito igual a:

0 ... pelo *hardware*, assim que entrar em busca de sincronismo de quadro (seção 1.4.2).

1 ... pelo processador, para forçar a entrada em busca de sincronismo de quadro.

24. *MAS1*: Registro de Máscara das Fontes de Interrupção

Cada um dos *bits* tem por finalidade mascarar ('0') ou não ('1') o correspondente *bit* gerador de interrupção. O *bit* de interrupção quando mascarado não provoca a ativação do sinal no pino *AINT*, mas pode ser verificado através de leitura do registro de interrupção. Todos os *bits* deste registro são **ativos após o *reset***.

a. *mste* ... Máscara da Saturação do Contador de Erros de CRC por Taxa Excessiva de Erro

b. *msnv* ... Máscara do Conjunto de Registros S_n Válido

c. *mrra* ... Máscara do Alarme Remoto Recebido

d. *mlos* ... Máscara da Perda de Sincronismo de Quadro

e. *mnos* ... Máscara da Perda do Sinal de Entrada

f. *mnck* ... Máscara da Falta do Relógio de Linha

g. *mslp* ... Máscara da Ocorrência de *Slip* na Memória Elástica de Recepção ou de Transmissão

h. *mepm* ... Máscara do Erro de Paridade na Memória Elástica de Recepção ou de Transmissão

25. *IDLE*: Registro com o Código de Canal *IDLE*

a. *idl7* a *idl0*: Código de Canal *IDLE*

Padrão a ser inserido em todos os canais selecionados pela memória para controle de canais (seção 2.2). É inicializado, **após *reset*, com padrão de silêncio (54H)**.

26. *LTEH/LTEL*: Registro com o Limiar da Taxa Excessiva de Erro

a. *lte9* a *lte0*: Limiar da Taxa Excessiva de Erro

Composto de 2 registros que totalizam 10 *bits*. Permitem ao processador definir o limiar de erros de CRC que implica na interrupção por taxa excessiva de erro (seção 1.4.2). A interrupção é gerada após a comparação do valor programado em *LTE* com o valor acumulado no contador de erros de CRC (*CEC*), caso *CEC* seja maior ou igual a *LTE*.

3.1.4 Registros de Estado

A tabela 10 lista os registros de estado do *TB47*.

Mnemônico	*D7*	*D6*	*D5*	*D4*	*D3*	*D2*	*D1*	*D0*
DFI	*df7*	*df6*	*df5*	*df4*	*df3*	*df2*	*df1*	*df0*
PROG	-	-	*pres*	*tp2*	*tp1*	*poe3*	*poe2*	*poe1*
CECL	*ce7*	*ce6*	*ce5*	*ce4*	*ce3*	*ce2*	*ce1*	*ce0*
CECH	-	-	-	-	-	-	*ce9*	*ce8*

Tabela 10: Registros de Estado.

A seguir estão descritos cada um dos registros de estado.

1. *DFI*: Diferença de Fase entre Relógios de Linha e Local

a. *df7* a *df0*: diferença de fase entre o relógio de linha e o relógio local, para a memória elástica de recepção. O valor apresentado neste registro (*df7* é o *bit* mais significativo) indica o número de 2 períodos de 2MHz de linha (2 *bits*) decorridos entre o sincronismo de quadro de linha e o sincronismo de quadro local.

Quando o *hardware* for alterar o conteúdo deste registro e o processador também estiver endereçando este registro, o *hardware* deve gerar um sinal de *WAIT* para o processador, tão pequeno quanto possível, via pino de saída *WAIT*. Isto deve ser feito para evitar uma leitura do registro pelo processador durante a transição do mesmo.

2. *PROG.PRES*: Presença da Placa no Sistema

0 ... placa presente. No *TB47* este sinal deve estar sempre ligado em '0'.

3. *PROG.TP2* e *PROG.TP1*: Tipo da Placa

Sinal vindo direto dos pinos *TP2* e *TP1*. O *TB47* não toma conhecimento do valor (nível lógico) destes pinos. Simplesmente repassa-os para o processador.

4. *PROG.POE3*: Saída Óptica Embaralhada ou Não

Este sinal, vindo do pino *POE3* é repassado para o processador e usado para a definição da saída óptica:

0 ... sinal na saída óptica (pino *XOID*) **NÃO** é embaralhado.

1 ... sinal na saída óptica é embaralhado.

5. *PROG.POE2*: Entrada do Sinal Óptico Embaralhada

Sinal vindo do pino *POE2* e repassado para o processador para definir a entrada óptica:

0 ... sinal na entrada óptica (pino *ROID*) **NÃO** passa pelo desembaralhador.

1 ... sinal na entrada óptica é desembaralhado.

6. *PROG.POE1*: Interface Óptica em Uso

Este sinal, vindo do pino *POE1* é repassado para o processador e usado para a definição do tipo da interface de entrada e saída:

0 ... o *TB47* usa a interface HDB3.

1 ... o *TB47* usa a interface óptica.

7. *CECL*: Registro com os *Bits* Menos Significativos do Contador de Erros CRC

a. *ce7* a *ce0*: Contador de Erros de CRC

Bits menos significativos do contador de erros de CRC (seção 1.4.2).

Quando o *hardware* for alterar o conteúdo deste registro e o processador também estiver endereçando o mesmo, o *hardware* deve gerar um sinal de *WAIT* para o processador, tão pequeno quanto possível, via pino de saída *WAIT*. Isto deve ser feito para evitar uma leitura do registro pelo processador durante a transição do mesmo.

8. *CECH*: Registro com os *Bits* Mais Significativos do Contador de Erros de CRC

a. *ce9* e *ce8*: Contador de Erros de CRC

Bits mais significativos do contador de erros de CRC (seção 1.4.2).

Quando o *hardware* for alterar o conteúdo deste registro e o processador também estiver endereçando o mesmo, o *hardware* deve gerar um sinal de *WAIT* para o processador, tão pequeno quanto possível, via pino de saída *WAIT*. Isto deve ser feito para evitar uma leitura do registro pelo processador durante a transição do mesmo.

3.2 Decodificação do Barramento de Endereços da Interface de Sistema

Este bloco é responsável por gerar os endereços necessários para os registros de interrupção (seção 3.1.1), memórias para *bits* S_n (seção 3.1.2), registros de controle (seção 3.1.3), registros de estado (seção 3.1.4) e memória para controle de canais (seção 2.2), através dos pinos de entrada *A0* a *A6*. O pino de entrada *CEQ* habilita um acesso de escrita/leitura nos registros internos e os pinos de entrada *WRQ* e *RDQ* definem se a operação será de escrita ou de leitura, respectivamente.

Portanto, se os pinos *CEQ* e *WRQ* estiverem ativos, a informação de controle contida no barramento de dados via pinos de entrada/saída *D0* a *D7* pode ser escrita no *TB47*, na posição endereçada por *A0* a *A6*. Se os pinos *CEQ* e *RDQ* estiverem ativos, a informaçao de estado da posição endereçada por *A0* a *A6* poderá ser lida do *TB47* via barramento de dados (pinos de entrada/saída *D0* a *D7*).

3.3 Interface de Sistema

Nesta seção é definida a forma e os sinais de comunicação entre o *TB47* e a placa

controladora do sistema (seção 7).

- sinal de saída do circuito de recepção (pino *RDO*). Este pino pode conter os seguintes dados:
  - em operação normal, contém o sinal de saída da memória elástica de recepção;
  - quando *ctr1.sais*='1', contém *AIS* (tudo '1').
- sinal de entrada para a transmissão (pino *XDI*): vindo da placa controladora com o enlace a ser transmitido.
- sinais associados ao enlace de recepção para extração do canal 16:
  - enlace *PCM* unipolar de recepção (pino *ERUN*), que contém o sinal unipolar vindo do pino *ROID* (*POE1*='1') ou o sinal de saída do bloco decodificador HDB3 (*POE1*='0'). Caso *POE1*='1', o enlace de entrada pode ter passado (*POE2*='0'), ou não (*POE2*='1') pelo bloco desembaralhador;
  - intervalo de tempo de canal 16 (pino *RSIGM*), que é derivado do relógio recuperado da linha (figura 21).
- sinal de relógio de 4MHz local (pino *SCLK_4M*).
- sincronismo de quadro local (pino *SYPQ*).
- barramento de endereços (pinos *A0* a *A6*).
- barramento de dados bidirecional (pinos *D0* a *D7*).
- *read* (pino *RDQ*).
- *write* (pino *WRQ*).
- *chip enable* (pino *CEQ*).
- sinal para controle de *bypass* (pode ser usado para indicar se o processador está presente ou não) (pino *SIPA*) (seção 2.9).
- canal 16 para transmissão (pino *CSTR*).
- interrupção para o processador (pino *AINT*).
- *WAIT* para o processador (pino *WAIT*).

4 Implementação de Facilidades para Teste

É responsável pelas seguintes funções:

4.1 Inserção de Erros no Enlace Transmitido

Para permitir que em bancada seja possível inserir controladamente erros no enlace de saída, será necessária a inclusão deste circuito. Trata-se de um XOR que tem em uma das entradas o enlace já completamente montado, pronto para ser enviado à interface de saída, ou seja, esta entrada deve estar conectada à saída do circuito de *bypass* (seção 2.9). A outra entrada deverá estar conectada ao pino *INSERR*. A saída deste circuito vai direto à interface de saída (seção 2.7).

4.2 Conexão das Memórias Elásticas

Deve ser prevista uma conexão da memória elástica de recepção com a memória elástica de transmissão, para Testes de Bancada. Esta função é habilitada pelo registro de controle *ctr3.lbme* (seção 3.1.3 e figura 19) e permite que um sinal *PCM* de entrada passe pelas memórias elásticas e saia do *TB47* pelo enlace de transmissão. Nesta situação o canal 0 deve continuar sendo inserido como em modo normal de funcionamento (seção 2.4).

4.3 *Loop Back* do Canal 16

Para o caso de teste de continuidade interna da *UDL*, deve ser previsto o *loop back* do canal 16, que liga o sinal de canal 16 vindo da interface de sistema para a transmissão (pino *CSTR*) no sinal de canal 16 que vai para a interface de sistema na recepção (pino *RSIGM*). Esta função é habilitada pelo registro de controle *ctr3.lbcs* (seção 3.1.3 e figura 18).

4.4 *Loop Back* para Teste Interno da *UDL*

Para o caso de teste de continuidade interna dos canais da *UDL* deve ser previsto o *loop back* de cada um dos canais (do 0 ao 31) (ver seção 3.1.3 e figura 18). Neste teste, será feito um *loop back* da memória elástica de transmissão para a memória elástica de recepção, função habilitada pela memória para controle de canais (seção 2.2).

O padrão a ser inserido em todos os canais selecionados será o definido no registro de controle *IDLE* (seção 3.1.3) ou o enlace serial vindo da interface de sistema via pino *XDI*, após ter passado pela memória elástica de transmissão.

> *Obs.: Esta função tem por finalidade testar o caminho: enlace de transmissão que sai do sistema (pino de entrada XDI); memória elástica de transmissão; memória elástica de recepção; enlace de recepção do TB47 para o sistema (pino de saída RDO). Durante o funcionamento normal da UDL, este teste pode ser aplicado a canais que não estejam alocados. Este teste autônomo da UDL não vai testar o codificador/decodificador HDB3. Este caminho pode ser testado pela central.*

4.5 Simulação de Erro de CRC

Sob controle do *bit ctr3.crci* (seção 3.1.3) deve-se poder inverter os *bits* do CRC calculado na transmissão, antes de inserí-los no enlace (via canal 0). A *UDL* seguinte deverá detectar este erro, constatando o funcionamento correto do circuito detector de erros de CRC.

4.6 Forçar Erro de Paridade nas Memórias Elásticas

Existe a possibilidade de inversão do sinal de saída do bloco gerador de paridade, antes de ser escrito no *bit* 9 das memórias elásticas, com o objetivo de forçar a ocorrência de erro de paridade. Para a memória elástica de recepção o *bit* de controle é *ctr2.fepr* e para a memória elástica de transmissão é *ctr2.fept* (seção 3.1.3).

4.7 Escrita nas Memórias S_n de Recepção

Esta memória, no modo funcional, não pode ser escrita pelo processador, o que impede seu teste. Para permitir este tipo de acesso, foram definidos os *bits* de controle *ctr3.esnr* e *ctr3.emsn* (seção 3.1.3). Controlando-os, o processador se permite fazer escritas nesta memória para testá-la. Durante esta fase de teste, ela não estará disponível funcionalmente.

4.8 Simulação de Alarmes

A simulação de alarmes não afeta a operação normal do dispositivo, todos os canais permanecem disponíveis para comunicação. Entretanto, possíveis condições reais de alarme não serão reportadas para o processador quando o dispositivo estiver neste modo de teste.

Neste caso, as máscaras dos *bits* de interrupção, representadas pelos *bits* do registro de controle *MAS1* (seção 3.1.3), mantêm-se em funcionamento, bloqueando ou não a ativação da interrupção, via pino de saída *AINT* (seção 3.1.1).

A simulação de alarmes é iniciada com o processador fazendo o *bit ctr3.sim*='1'. Após ele ser zerado, também pelo processador, todas as condições de erro simuladas irão desaparecer e o circuito volta ao funcionamento normal.

Quando em simulação de alarme, os *bits* de interrupção (registros *INT1* e *INT2*) são ativados e o contador de erros de CRC (registros *CECH* e *CECL*) é incrementado a cada 1ms.

As seguintes condições de alarme são simuladas (seção 3.1.1):

- saturação da taxa de erro excessiva (*int1.ste*),
- conjunto de registros S_n válido (*int1.snv*),
- alarme remoto recebido (*int1.rra*),
- perda de sincronismo de quadro (*int1.los*),
- perda do sinal de entrada (*int1.nos*),
- perda de relógio de linha (*int1.nck*),
- ocorrência de *slip* na memória elástica de recepção ou de transmissão (*int2.slp*),
- erro de paridade na memória elástica de recepção ou de transmissão (*int2.epm*),
- o contador de erros de CRC é incrementado a cada 1ms.

4.9 Testabilidade

A implementação prevista para o circuito é em *FPGA* na fase de protótipo e em *Standard Cells* na fase de produção. Embora seja suficiente a avaliação funcional para o teste de *FPGA*, pois estruturalmente é garantido pelo fabricante, deverá ser prevista a implementação de testabilidade para teste estrutural, por ser necessária na fase de produção.

5 *Loop* de Reconfiguração do Enlace

Para o caso de falha nos enlaces de transmissão e de recepção em um dos pontos do anel (falha dupla), torna-se necessária a reconfiguração do anel, denominada *loop* de reconfiguração do enlace, e habilitada pelo *bit* de controle *ctr2.lre* (seção 3.1.3). Para possibilitar esta reconfiguração uma *UIL* (tratada por A) prossegue executando suas funções normalmente. A outra *UIL* (B), tem o *bit ctr2.lre* programado para '1' e passa a funcionar de forma específica, como mostrado na figura 17 e descrito a seguir:

- na recepção, passa a simplesmente monitorar o sinal recebido atualizando os alarmes *int1.nos*, *int1.rra* e *int1.los* (seção 3.1.1). O processador, para ficar monitorando o estado da linha desativada, deve varrer periodicamente o registro de interrupção;

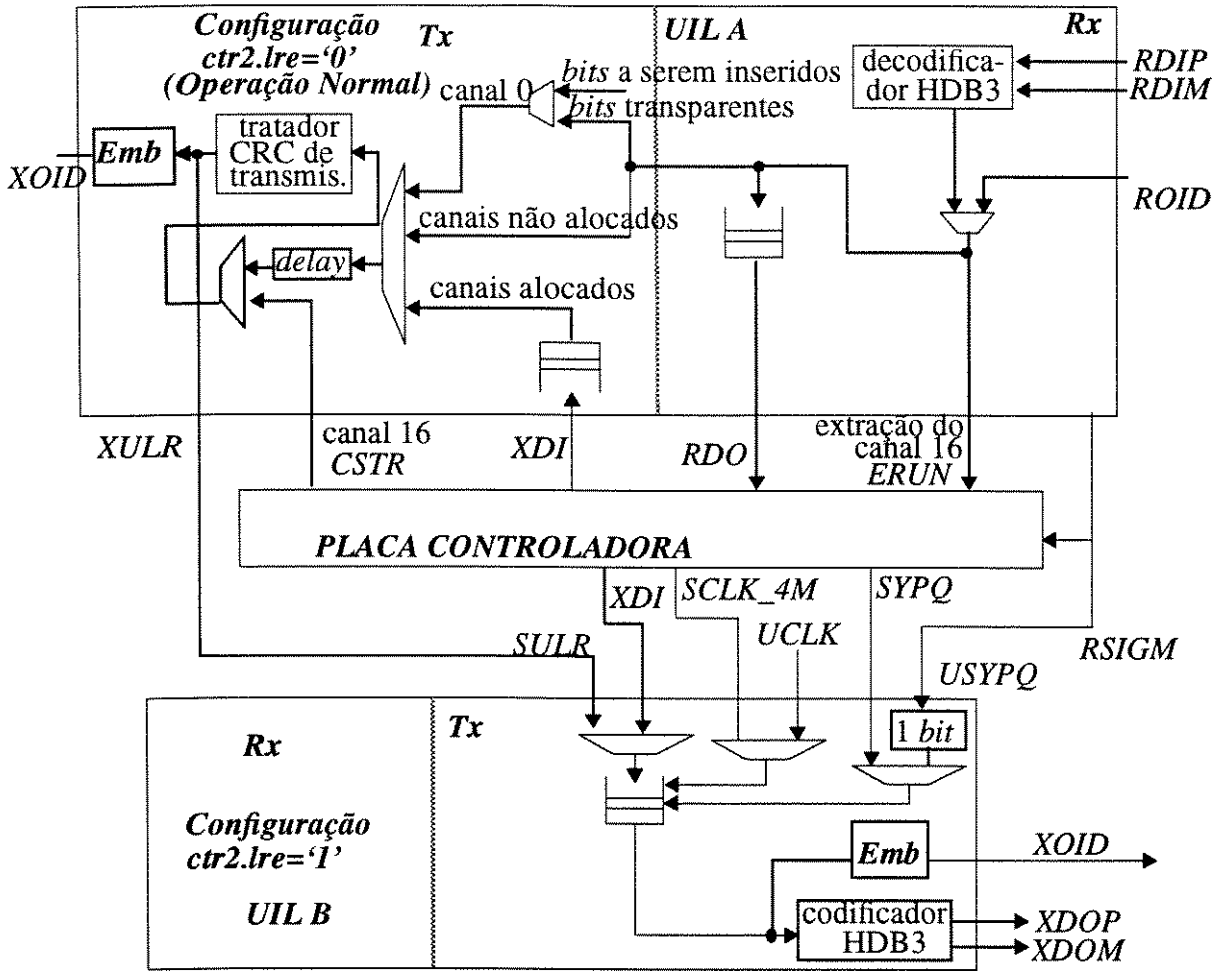


Figura 17: *Loop* de Reconfiguração do Enlace.

- na transmissão, o enlace a ser transmitido, que no modo normal é montado internamente dentro da própria *UIL* (seção 2.6), no modo reconfiguração, é substituído pelo enlace montado na outra *UIL*, vindo do pino *SULR*, após ter passado pela memória elástica de transmissão. A seguir o sinal segue o mesmo caminho tanto para a *UIL* A como para a *UIL* B;
- a memória elástica de transmissão, no caso da *UIL* B, deve ter sua temporização alterada de forma que a sua recepção fique síncrona com a temporização da *UIL* A, e sua saída fique síncrona com o relógio de sistema (local). Isto deve ser feito para reduzir o *jitter* no enlace sendo transmitido.

6 Fluxo dos Sinais de Controle para os Enlaces de Recepção e de Transmissão

As figuras 18 e 19 representam o fluxo dos sinais de controle para os enlaces de recepção e de transmissão.

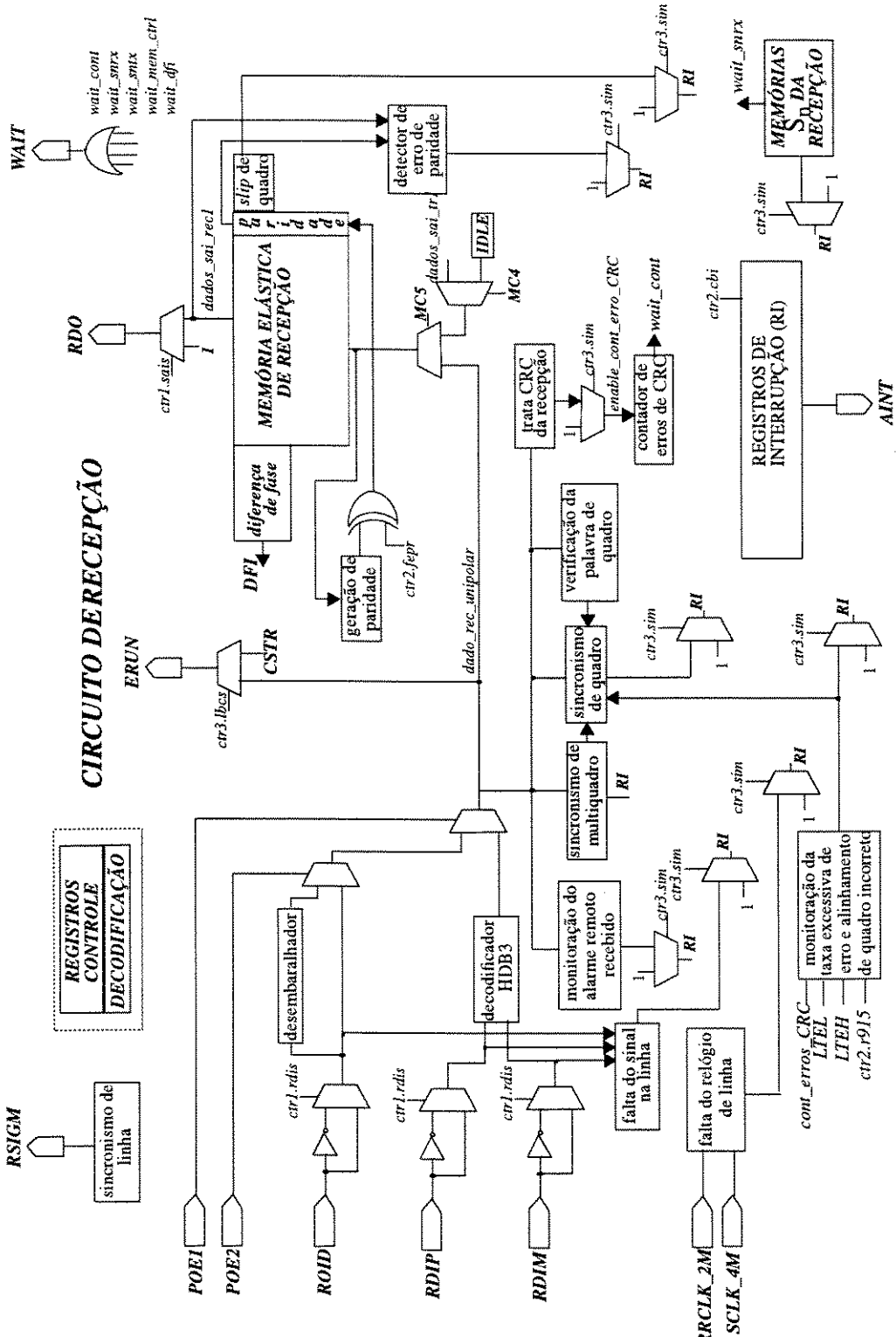


Figura 18: Sinais de Controle para o Enlace de Recepção.

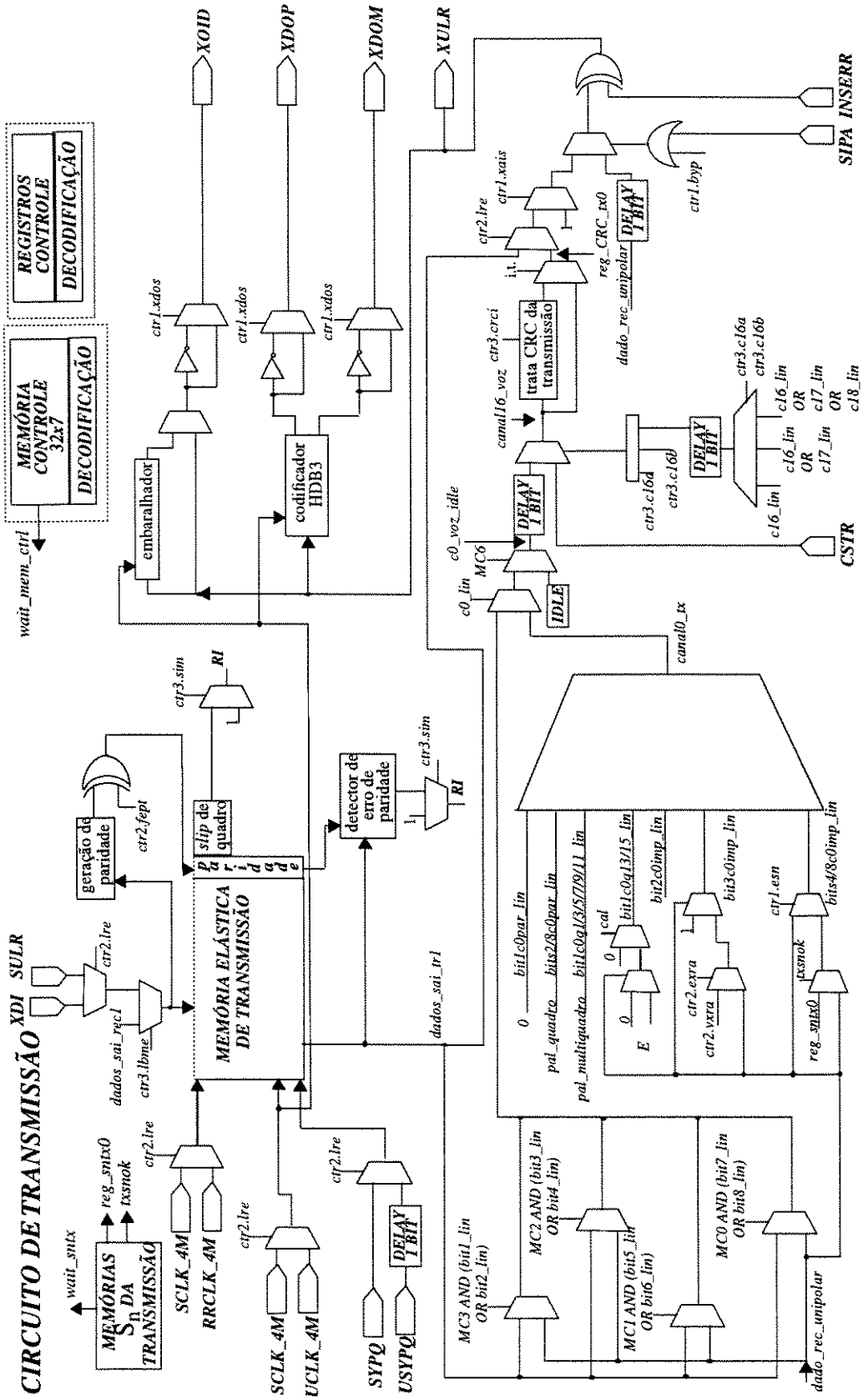


Figura 19: Sinais de Controle para o Enlace de Transmissão.

7 Pinagem

7.1 Descrição dos Pinos

A figura 20 mostra os pinos do circuito integrado *TB47* e na tabela 11 tem-se uma descrição funcional dos mesmos.

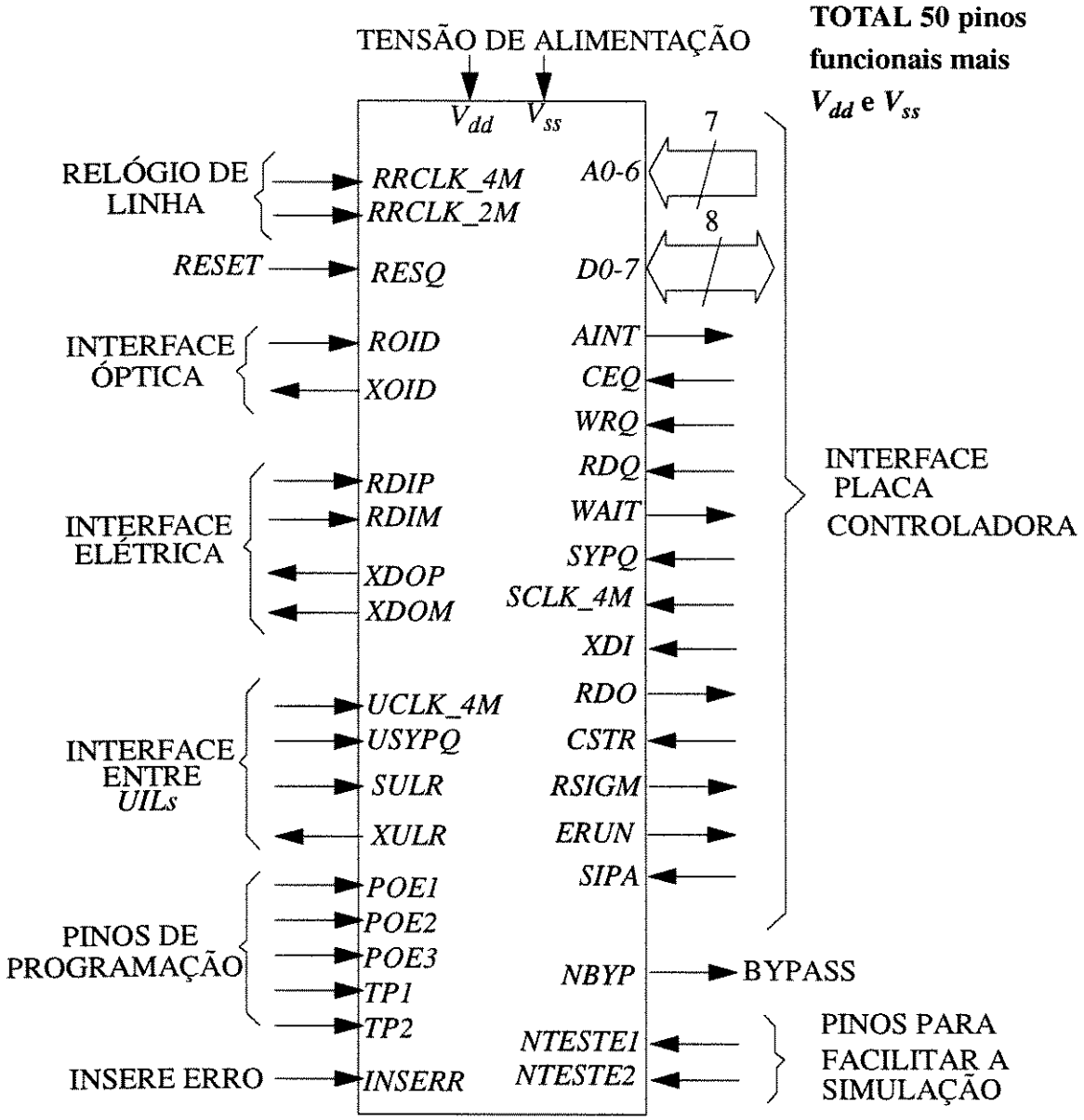


Figura 20: Pinagem.

Símbolo	I/O	Função
ROID	I	**Dado da Interface Óptica de Recepção** Dado unipolar NRZ ou RZ, na frequência de 2048kHz.

Tabela 11: Pinagem do *TB47*.

Símbolo	I/O	Função
RESQ	I	***Reset*** (ativo baixo) Um sinal ativo baixo zera todos *flip-flops* internos. Os *bits* dos registros de controle são inicializados com os valores definidos na seção 3.1.3.
RRCLK_4M	I	**Relógio de Linha Recebido a 4096kHz** Relógio do enlace de entrada recuperado na placa.
RRCLK_2M	I	**Relógio de Linha Recebido a 2048kHz** Relógio do enlace de entrada recuperado na placa.
RDIP *RDIM*	I I	**Dados do Enlace *PCM* de Recepção Positivo e Negativo** Sinais de entrada no formato HDB3 unipolar RZ em 2048kHz.
A0 a *A6*	I	**Barramento de Endereços** Sete linhas do barramento de endereço do processador para selecionar registros internos.
D0 a *D7*	I/O	**Barramento de Dados** Oito linhas de dados *tri-state* bidirecional que fazem interface com o barramento de dados do processador.
CEQ	I	***Chip Enable*** (ativo baixo) Habilita o acesso de escrita/leitura nos registros internos.
WRQ	I	**Habilitação de Escrita** (ativo baixo) Este sinal ativa uma operação de escrita dos dados *D0* a *D7* em uma posição interna do *TB47* endereçada pelos sinais *A0* a *A6*, caso *CEQ* também esteja ativo.
RDQ	I	**Habilitação de Leitura** (ativo baixo) Este sinal ativa uma operação de leitura, através dos sinais *D0* a *D7*, de uma posição interna do *TB47* endereçada pelos sinais *A0* a *A6*, caso *CEQ* também esteja ativo.
SYPQ	I	**Pulso de Sincronismo de Sistema** Define o começo do intervalo de tempo de canal 0 da interface de sistema. O ciclo do pulso é um múltiplo inteiro de 125μs.
SCLK_4M	I	**Relógio de Sistema** Relógio local com uma frequência de 4096kHz.
XDI	I	**Enlace da Interface de Sistema a ser Transmitido** Enlace recebido da interface de sistema na frequência de 2048kHz.
CSTR	I	**Canal 16 a ser Transmitido** Enlace contendo o canal 16 a ser inserido no enlace de transmissão. Este sinal deve ser síncrono com o relógio de linha, *RRCLK_2M*.
UCLK_4M	I	**Relógio da *UIL* A para *Loop* de Reconfiguração** Relógio para uso pela *UIL* B quando no modo *loop* de reconfiguração.

Tabela 11: Pinagem do *TB47*.

Símbolo	I/O	Função
USYPQ	I	**Sincronismo da *UIL* A para *Loop* de Reconfiguração** Define o começo do intervalo de tempo de canal 0 da *UIL* A. O ciclo do pulso é um múltiplo inteiro de 125µs.
SULR	I	**Sinal Unipolar para *Loop* de Reconfiguração** Enlace *PCM* vindo da *UIL* A para, no modo *loop* de reconfiguração, ser transmitido.
SIPA	I	**Sinal Indicativo de Processador Ausente** 0 ... indica processador presente. 1 ... indica processador ausente.
POE1 a *POE3*	I	**Programação das Interfaces de Linha** Permitem a definição sobre o uso de interface HDB3 ou unipolar, assim como do embaralhador/desembaralhador.
TP1 e *TP2*	I	**Tipo de Placa** Informam ao processador, via registro de estado do *TB47*, o tipo de placa presente.
INSERR	I	**Controle da Inserção de Erro** Sinal para permitir inserção controlada de erros no enlace transmitido.
V_{dd}	I	**Alimentação** Tensão de alimentação de +5V.
V_{ss}	I	**Terra** 0V.
RSIGM	O	**Sincronismo para Extração do Canal 16 Recebido** Sinal de sincronismo para possibilitar a localização do canal 16 no enlace do pino *ERUN*.
ERUN	O	**Enlace *PCM* Recebido no Formato Unipolar** O enlace *PCM* recebido, é repassado para a interface de sistema para possibilitar a extração do canal 16.
XOID	O	**Dado da Interface Óptica de Transmissão** Dado unipolar NRZ ou RZ, na frequência de 2048kHz.
XDOP *XDOM*	O O	**Dados do Enlace *PCM* de Transmissão Positivo e Negativo** Sinais de saída no formato HDB3 unipolar RZ na frequência de 2048kHz.
AINT	O	**Interrupção** (*open drain*, ativo baixo) É ativada pelas fontes de alarme não mascaradas.
WAIT	O	**Sinal de Espera para o Processador** (ativo alto) Ativo sempre que o processador ao tentar escrever ou ler registros possa provocar conflitos entre este seu acesso e um eventual acesso do próprio *TB47* ao mesmo registro.

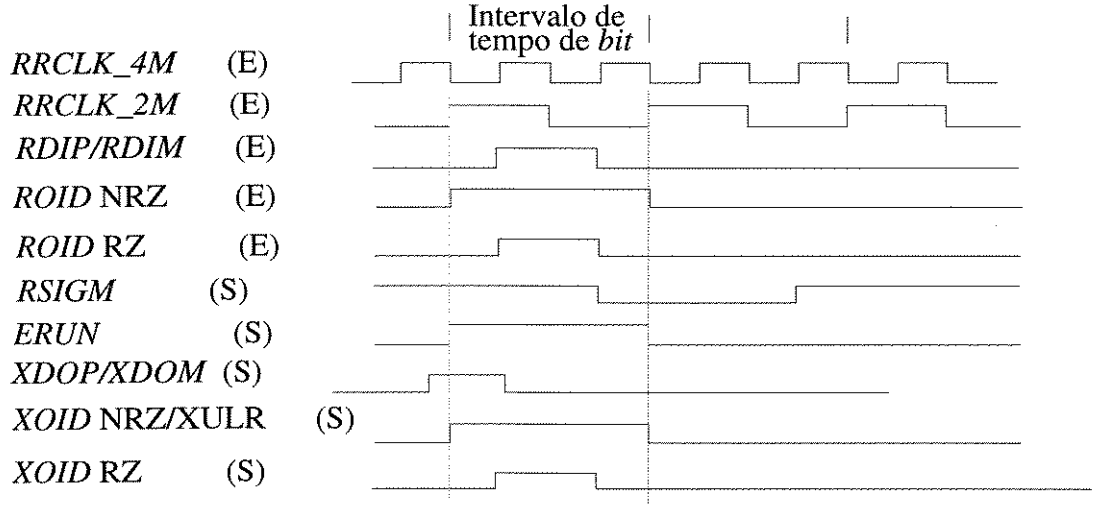
Tabela 11: Pinagem do *TB47*.

Símbolo	I/O	Função
RDO	O	**Enlace Recebido tendo Passado pela Memória Elástica** Dado de saída da memória elástica de recepção enviado para a interface de sistema na frequência de 2048kHz.
XULR	O	**Enlace de Saída da *UIL* A** Enlace *PCM* de saída da *UIL* A para, no modo *loop* de reconfiguração, ser transmitido pela *UIL* B (figura 17).
NBYP	O	**Sinal Indicativo de Bypass** Sinal ativo em '0' que indica UIL em bypass.
NTESTE1 *NTESTE2*	I	**Pinos para Facilitar a Simulação** Como existiam pinos sobrando no componente, foram incluídos estes dois pinos para diminuir o tempo de simulação e teste de algumas funções do componente. Se "11" o componente está fora do modo teste. Se "00", o componente está no modo teste e o circuito contador de linha passa somente pelos canais 0 e 31.

Tabela 11: Pinagem do *TB47*.

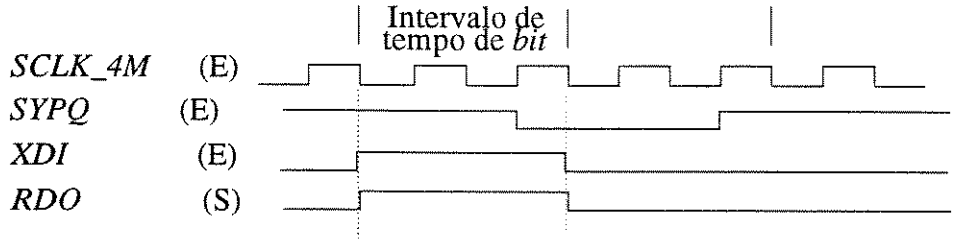
7.2 Diagrama de Tempos dos Pinos de Entrada/Saída

As figuras 21, 22 e 23 apresentam a relação de fase dos sinais de entrada/saída do *TB47*.



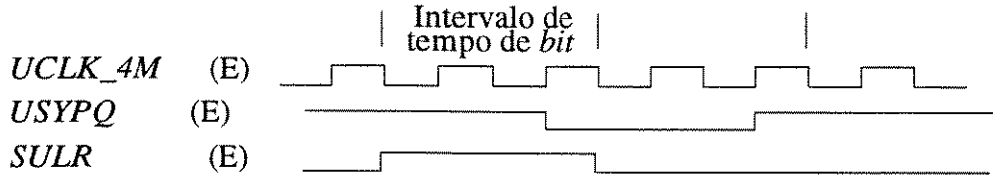
Com exceção do sinal *RSIGM*, todos são cíclicos a nível de intervalo de tempo de *bit*. *RSIGM* se repete a cada quadro (256 *bits*), com início no final do *bit* 8 do canal 31. A primeira borda de descida do relógio *RRCLK_4M* com *RSIGM* em zero, define o início do canal 0.

Figura 21: Relação de Fase dos Sinais Síncronos com a Linha.



A 1ª borda de descida do relógio *SCLK_4M* com *SYPQ* em zero, define o início do canal 0.

Figura 22: Relação de Fase dos Sinais Síncronos com o Relógio Local.



A 1ª borda de descida do relógio *UCLK_4M* com *USYPQ* em zero, define o início do canal 0.

Figura 23: Relação de Fase dos Sinais de Entrada Necessários para *Loop* de Reconfiguração.

CAPÍTULO V

Metodologia de Projeto

A metodologia de projeto aqui utilizada é a denominada *Top Down*. Portanto, inicialmente será apresentada uma teoria sobre Metodologia *Top Down*. Será apresentado, também, sobre *Programmable Gate Arrays* para facilitar a posterior explicação da implementação final do circuito.

Finalmente, tem-se a Descrição da Sequência de Etapas de Projeto utilizada com um detalhamento de cada etapa.

Devido a alta complexidade do circuito e devido ao fato de ser um trabalho em grupo, somente serão descritos os blocos desenvolvidos pelo autor deste trabalho.

1 Metodologia *Top Down*

1.1 Introdução

Até agora, os projetistas eram obrigados a usar a tradicional metodologia de projeto baseada em esquemático a nível de portas (*Bottom Up*). Na década de 90, a complexidade dos circuitos aumentou, com *ASICs* (*Application Specific Integrated Circuits*) excedendo 100.000 portas, junto com grandes exigências do mercado. As metodologias de projeto *Bottom Up* da década de 80 não são mais soluções viáveis, porque são incômodas, monótonas, lentas e propensas a erros.

Um nível mais alto, a síntese *Top Down* é essencial para satisfazer as necessidades atuais de projeto [8, 9]. Ou seja, conduzidos por uma feroz competição global e uma diminuição das janelas de mercado, os projetistas de sistemas eletrônicos estão mudando sua metodologia de projeto para suportar projetos *Top Down*. O objetivo desta metodologia é reduzir drasticamente o tempo para disponibilidade no mercado, enquanto aumenta a funcionalidade, desempenho e confiabilidade do produto [10].

O projeto *Top Down* necessita de um meio comum, o que é satisfeito pela linguagem *VHDL* (*VHSIC* (*Very High Speed Integrated Circuits*) *Hardware Description Language*), que é uma linguagem potente para descrição e modelamento de circuitos e sistemas [11, 12, 13, 14, 15, 16, 17]. A nível mundial, esta linguagem para modelamento de *hardware VHDL* está sendo adotada por uma grande parcela da comunidade científica na área de projeto de sistemas e circuitos integrados [13]. Um projeto pode ser rapidamente definido usando *VHDL* e verificado através de simulação. Com a funcionalidade verificada, a síntese pode transformar, rapidamente, o projeto para o nível de portas, automatizando uma tarefa que anteriormente era tediosa e propensa a erros. Usando a síntese *VHDL Top Down*, os projetistas que produzem 50 *gates* por dia usando os métodos a nível de portas, podem tipicamente produzir 200 ou mais *gates* por dia. A síntese faz para os projetistas de *hardware*, o que o ALGOL (e seus derivados C e Pascal) fazem pelos engenheiros de *software*, livrando-os do trabalho penoso de tarefas

repetitivas de projeto [8]. Existem vários sistemas de síntese que aceitam o *VHDL*, ou melhor, um subconjunto do mesmo como linguagem de entrada [18, 19, 20].

A síntese *VHDL Top Down* permite que se concentre na funcionalidade do projeto ao invés de se preocupar com o projeto *Bottom Up* a nível de portas [21]. A combinação de linguagens de descrição de *hardware* com técnicas de síntese lógica aumenta a produtividade do projeto, permitindo que se explore novas alternativas de implementação [22].

Uma das vantagens da síntese *VHDL* é que o processo de projeto é independente da tecnologia. O projetista faz o projeto funcional e adia os detalhes dependentes de tecnologia. As relações de projeto e arquitetura podem ser feitas sem a necessidade de se considerar a tecnologia destino. O poder da síntese consiste em permitir que um único projeto *VHDL* seja automaticamente sintetizado em uma variedade de tecnologias, tais como, *Application Specific Integrated Circuits* (*ASICs*), *Field Programmable Gate Arrays* (*FPGAs*) e *Programmable Logic Devices* (*PLDs*) (seção 2.2).

Outra vantagem do projeto *Top Down* é a facilidade com que a funcionalidade do projeto pode ser modificada. Uma mudança na especificação funcional durante o projeto usando um método de projeto a nível de portas (*Bottom Up*) pode resultar em um grande atraso. Com um método de projeto *Top Down* empregando síntese *VHDL*, mudanças funcionais podem ser feitas rapidamente e verificadas através de simulação. Devido a geração automática do esquemático, o tempo de produção só é afetado pelo pequeno tempo de re-implementação do modelo *VHDL*.

O método de projeto baseado em linguagem fornece uma documentação inerente. Devido ao fato do *VHDL* ser padronizado mundialmente, os projetos são portáveis de, e para, ferramentas de vários vendedores.

Considerando-se as restrições de projeto durante a síntese de alto nível, a síntese *VHDL* produz arquiteturas de forma a obedecer estas restrições de forma mais efetiva, tais como caminhos críticos.

Como o processo de projeto *ASIC* tem se tornado muito mais complexo com um aumento da complexidade dos projetos e menores ciclos de projeto, a síntese e a otimização lógica se tornaram significativamente mais importantes no processo de projeto. Estas ferramentas oferecem um método confiável de manter-se em dia com as demandas de mercado, diminuindo o tempo para disponibilidade dos seus projetos. O fluxograma da figura 24 mostra onde a síntese e a otimização se encaixam no processo de projeto.

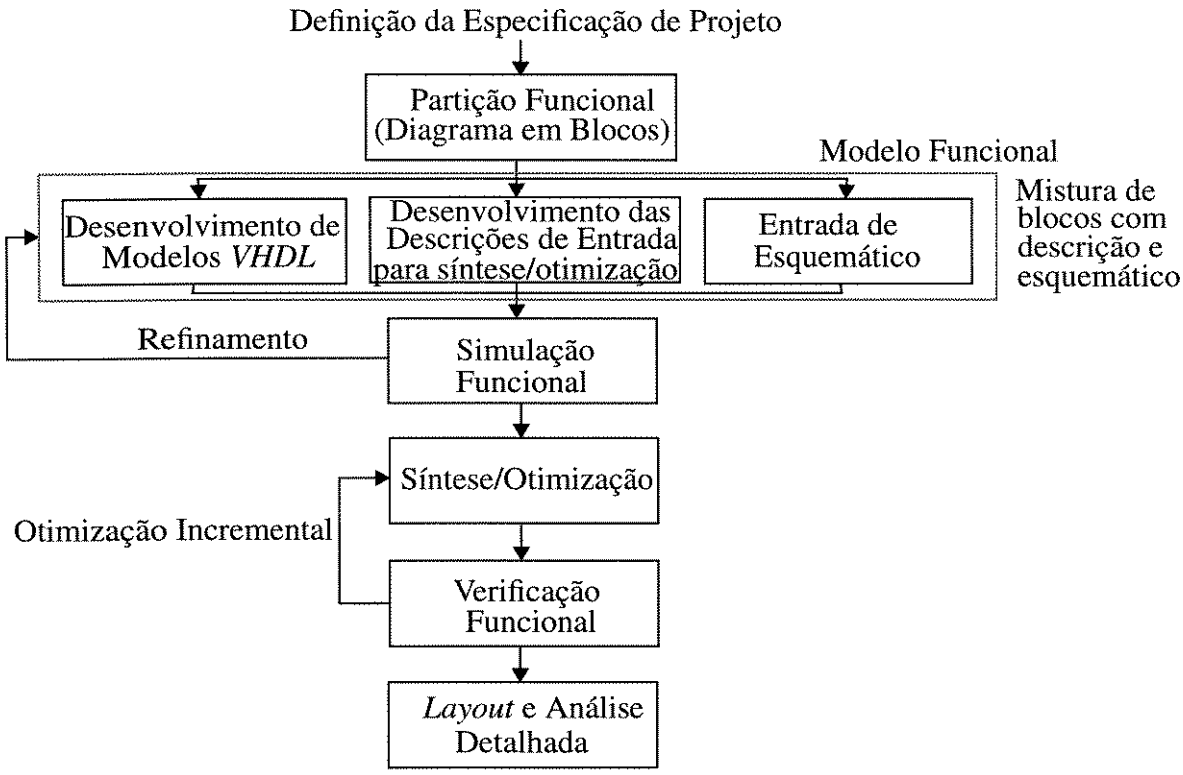


Figura 24: Processo de Projeto *Top Down* com Síntese Lógica.

Portanto, as seis principais vantagens do projeto *Top Down*, são: [23]

- aumenta a produtividade do projeto: aumento de 3 a 4 vezes no número de *gates* verificados por semana,
- diminui o ciclo de desenvolvimento do produto: começa o desenvolvimento do *software* mais cedo e em paralelo com o desenvolvimento do *hardware*,
- aumenta a competitividade: produz projetos em menos tempo com mais confiabilidade e com mais funções,
- aumenta a probabilidade de sucesso na primeira vez: simula seu circuito no contexto do sistema,
- reduz os custos de desenvolvimento: detecta erros funcionais mais cedo no ciclo de projeto, e
- aumenta o re-uso do projeto.

1.2 O que é Projeto *Top Down*

De uma forma geral, o projeto *Top Down* pode ser descrito como o seguinte fluxo, aplicado repetidamente sobre o projeto até se chegar a níveis apropriados de abstração (ilustrados na figura 25):

Projeto menos detalhado -> Refinamento -> Projeto mais detalhado [24].

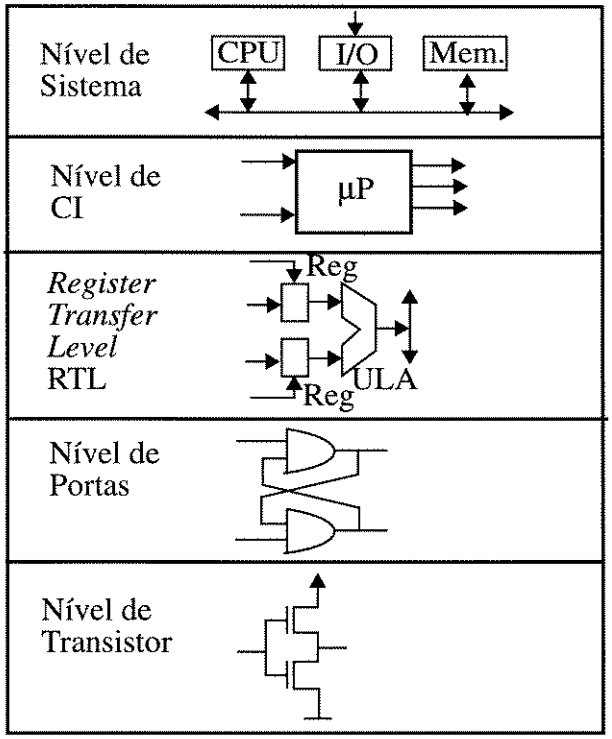


Figura 25: Níveis de Abstração de um Projeto.

Usando-se esta definição bem geral, as pessoas já tem usado os métodos de projeto *Top Down* há vários anos. O projeto *Top Down* é um processo e não um conjunto de ferramentas. Os grupos de projeto já desenvolveram modelos de simulação para partes de sistemas, implementando-os na mão, antes das ferramentas de síntese estarem disponíveis. Hoje, as ferramentas de síntese e otimização ajudam a automatizar muitos dos passos de refinamento que anteriormente eram feitos manualmente.

Hoje, o termo Projeto *Top Down* geralmente se refere à prática do uso de uma Linguagem de Descrição de *Hardware* (*HDL*) para capturar a funcionalidade de um projeto em um certo nível de abstração. Simuladores, ferramentas de síntese e otimização são usadas para transformar a abstrata descrição *HDL* em um projeto físico especificamente direcionado para uma tecnologia *ASIC*, *FPGA* ou *PLD*.

1.3 Fluxo Geral de Projeto

O fluxo de um projeto *Top Down* pode ser resumido nos seguintes passos e tarefas associadas da tabela 12 [24]. Cada etapa pode ser feita por um ou vários indivíduos trabalhando no grupo de projeto [25].

Passo	Tarefa
Especificação do produto	Escrever os documentos de especificação
Desenvolvimento do algorítmo	Modelamento comportamental e simulação
Decomposição do projeto	Diagramas em bloco, modelamento a nível *RTL* (*Register Transfer Level*) [26]

Tabela 12: Fluxo Geral de Projeto *Top Down*.

Passo	Tarefa
Verificação funcional	Simulação funcional
Implementação do módulo	Captura de esquemático, síntese e otimização
Layout	Base de dados *netlist* para um vendedor *ASIC*
Verificação à nível de portas pós-*layout*	Simulação a nível de portas e análise de *timing pós-layout*

Tabela 12: Fluxo Geral de Projeto *Top Down.*

Cada etapa normalmente terá uma ferramenta específica ou um conjunto de ferramentas que são usadas para ajudar a capturar ou automatizar o processo da figura 26.

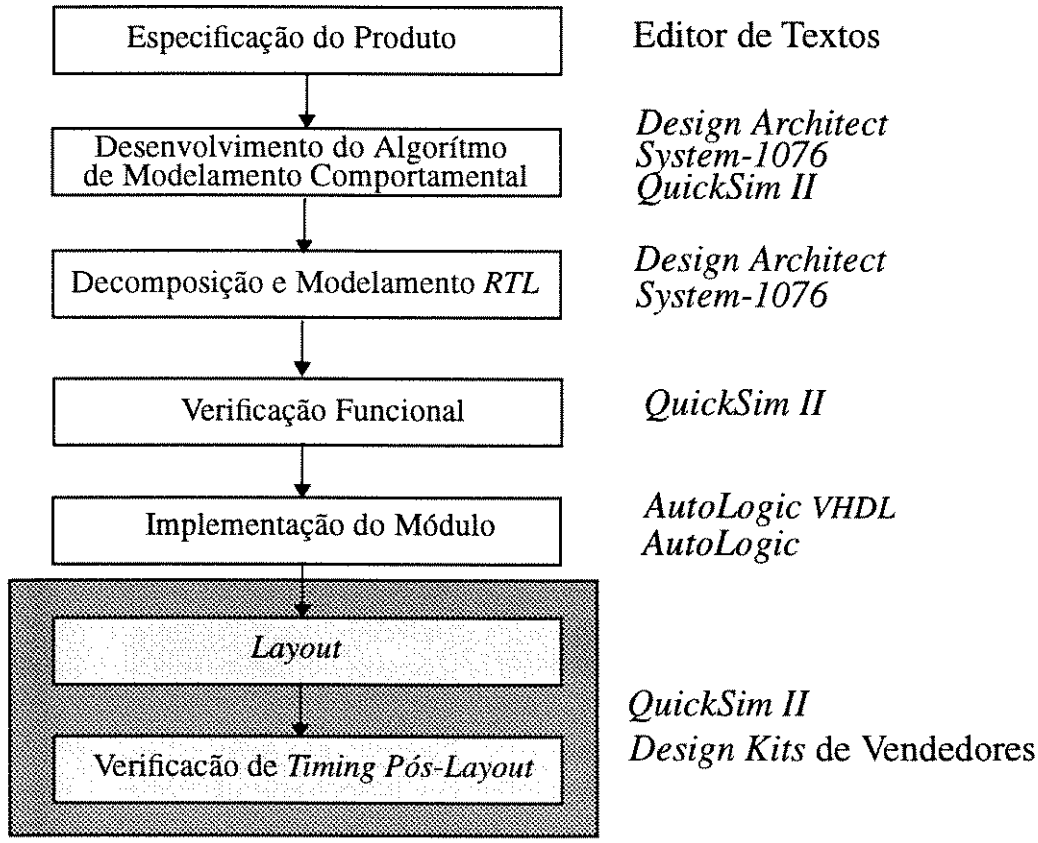


Figura 26: Fluxo Geral de Projeto *Top Down.*

Cada etapa no processo pode ser muito complexa. Por exemplo, o desenvolvimento de um único módulo *VHDL* pode conter as seguintes etapas:

1. Criação de um diagrama em blocos do módulo;
2. Escrita do código *VHDL;*
3. Compilação do código *VHDL* e correção de todos possíveis erros;
4. Desenvolvimento do *Test Bench* [2] para o módulo;
5. Simulação do módulo, verificação da funcionalidade, interação até estar correto;
6. Avaliação para garantir a síntese do módulo;

7. Criação do símbolo para o componente; e,

8. Cópia do módulo para o diretório final.

Cada etapa no processo pode ser repetida várias vezes até que os resultados desejados sejam obtidos. Então, o processo é repetido no próximo módulo na hierarquia até que o projeto esteja completo.

As ferramentas *Mentor Graphics* tipicamente usadas na implementação de uma estratégia *Top Down*, são:

Design Architect - edição *VHDL* e esquemático [27].

System-1076 - compilação *VHDL* e simulação [28].

QuickSim II - para simulação *VHDL* e a nível de portas [29].

AutoLogic VHDL - síntese do *VHDL* para portas genéricas, independente de tecnologia [30].

AutoLogic - otimização lógica e mapeamento de tecnologia [31].

Juntamente com as ferramentas da *Mentor*, será necessário um *Design Kit* do vendedor de *ASIC* ou *FPGA*. Este *Kit* inclui programas específicos que fazem a interface com os vendedores de sistemas *CAD (Computer-Aided Design)*, incluindo [23]:

- modelos de biblioteca para simulação e síntese,
- integração *framework*, incluindo a interface do usuário, regras de projeto e biblioteca de símbolos,
- interfaces que incluem *netlisters* para extração da descrição do circuito a nível de portas da base de dados da *Mentor*, *back annotator*, formatadores de vetores de teste e verificador de regras de projeto,
- aplicações específicas do vendedor, tais como ferramentas de *layout* (no caso de vendedores de *FPGA*), e
- documentação.

Obs.:

Netlist: um arquivo que lista e descreve todos os componentes em um projeto e mostra como eles estão interconectados; geralmente usado como entrada para simuladores ou programa roteadores de fios [32].

Netlister: um programa que produz um arquivo (netlist) que descreve um projeto [32].

Back annotator: um programa que produz um arquivo back annotation, que contém informações dos parâmetros extraídos do respectivo layout.

2 Programmable Gate Arrays

2.1 Introdução

Constantes avanços no nível de integração de circuitos eletrônicos têm melhorado muitas características dos equipamentos, reduzindo custos, consumo de potência e tamanho do sistema, enquanto aumenta o desempenho e a confiabilidade [33]. O aumento dos níveis de integração são mais evidentes em circuitos integrados de microprocessadores e memória.

Com cada geração de processo, a diferença tecnológica entre estes circuitos *VLSI* e outros circuitos integrados lógicos padronizados tem se ampliado. Para conseguir densidades comparáveis para suas funções lógicas proprietárias, os projetistas de equipamento digital tem sido forçados a considerar os *ASICs* dedicados e semi-dedicados programados na fábrica.

FPGAs são *ASICs* de alta densidade que podem ser configurados pelo usuário. Eles combinam os benefícios de integração de lógica de *VLSI* dedicado com as vantagens de projeto, produção e exigências de mercado de produtos padrão. Os projetistas definem as funções lógicas do circuito e podem fazer alterações nestas funções quando necessário. Assim *FPGAs* podem ser projetadas e verificadas em poucos dias, em oposição a várias semanas para *gate arrays* dedicados; mudanças no projeto *FPGA* podem requerer poucas horas, comparado a várias semanas para um *array* dedicado. Isto resulta em significante economia pela redução dos riscos de mudanças de projeto, replanejamento, e eliminação de custos *non-recurring engineering* (*NRE* - Engenharia Não Recorrente).

2.2 Alternativas *ASIC*

ASICs são a melhor solução para a maioria das funções lógicas. A melhor solução *ASIC* depende dos requisitos de densidade e volume de produção. A tabela 13 mostra uma comparação entre os diversos dispositivos [34].

método	custo de desenvolvimento/chip (US$)	tempo para recebimento dos protótipos	% pré-processada do *wafer*	metodologia de projeto
full custom	50k a 250k	6 a 18 meses	0	*custom*
standard cells	25k a 80k	2 a 6 meses	0	*custom*
gate array	5k a 40k	2 semanas a 3 meses	80 a 90	*semicustom*
programável em campo	0	*off-the-shelf*	100	(padrão)

Tabela 13: Comparação entre Alternativas *ASIC*.

Na técnica *full custom* todo o circuito tem que ser projetado e são usadas máscaras exclusivas para todas as camadas utilizadas na fabricação. Este método resulta no mais alto custo de desenvolvimento e um maior tempo de desenvolvimento. Portanto, é ideal para circuitos de alto volume de produção e longo tempo de vida.

O método *standard cells* (projeto baseado em células) também é dedicado (requer um conjunto exclusivo de máscaras), mas utiliza funções básicas já projetadas, que formam uma biblioteca de células.

As células desta biblioteca *standard cells* são normalmente retangulares e a posição das conexões de entrada e saída são padronizadas, o que automatiza a alocação das células e roteamento das linhas de interconexão entre as células para desempenhar a função específica. Com esta automatização, o tempo de desenvolvimento e o custo é reduzido de 50 a 60% em relação ao *full custom*. Entretanto, tem-se um aumento de 20 a 40% na área do *die* em relação ao *full custom* e o desempenho do circuito também é levemente menor.

Os *gate arrays* permitem programação pela interconexão de transistores, para composição de funções mais complexas, durante os últimos estágios do processo de fabricação. Uma vez que estes *arrays* já estão definidos em termos de posicionamento e tamanho, os fabricantes podem processar os *wafers* com as camadas já definidas e deixá-los prontos para padronização futura. Portanto, possui um menor custo e menores tempos de produção.

Os componentes programados em campo (por exemplo, *FPGA*) embora sejam dispositivos programados pelo usuário, do ponto de vista dos fabricantes são *CIs* de prateleira (*off-the-shelf*), ou seja, já são 100% pré-processados. Normalmente são usados para prototipagem ou pequeno volume de produção.

Obs.:

Custom IC: são circuitos integrados projetados para uma aplicação de uso específico que requer um conjunto completo de máscaras para implementar sua função (full custom e standard cells).

Semicustom IC: qualquer dispositivo que requer um pequeno número de máscaras exclusivas (normalmente para metalização) para implementação de uma determinada função (gate array). Normalmente, os dispositivos programados em campo estão nesta classificação embora não se encaixem na definição tradicional.

2.3 Lógicas Programáveis "*versus*" *ASICs*

Abaixo são relacionadas algumas vantagens de *FPGAs* (arquitetura *Gate Array* programável) em relação aos *ASICs* [35].

1. Projeto e validação mais rápidos: Pode-se projetar e validar as *FPGAs* em poucos dias ao contrário de várias semanas para *ASICs*. Não existem custos *de non-recurring engineering* (*NRE*), e nenhum protótipo por esperar.
2. Mudança no projeto sem penalidades: Devido ao fato dos dispositivos *FPGAs* serem programados por *software* via programação instantânea, as modificações possuem muito menos riscos e podem ser feitas em poucas horas, ao contrário de algumas semanas para o caso de *ASICs*.
3. Menor tempo para disponibilidade no mercado: Com as lógicas programáveis, o tempo de disponibilidade no mercado varia de alguns dias a poucas semanas. Para os *ASICs* este tempo é medido em meses (ver figura 27).

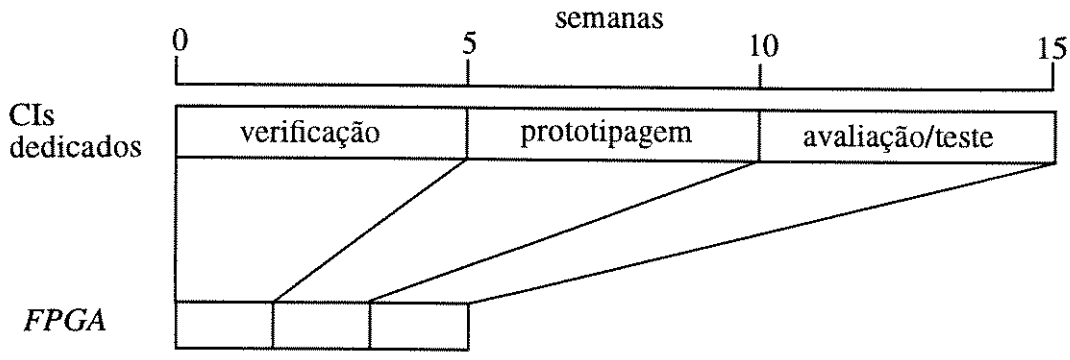


Figura 27: Tempo para Disponibilidade no Mercado.

4. Testes feitos pelos fabricantes: Ao contrário dos *ASICs*, os dispositivos programáveis são totalmente testados pelos fabricantes com programas de teste exaustivos.

Como conclusão, pode-se dizer que para protótipos e produção em pequena escala de componentes não muito densos, compensa o uso de *FPGAs*. Porém, para produções maiores o custo inviabiliza o uso de *FPGAs*, e os *ASICs* se tornam a melhor opção.

2.4 Arquitetura de *Programmable Gate Arrays*

A arquiterura *LCA* (*Logic Cell Array*) é similar a de *Gate Arrays*, com uma matriz interior de blocos lógicos e uma envoltória de blocos de interface de entrada/saída. Os recursos de interconexão ocupam os espaços entre as linhas e colunas de blocos lógicos, e entre os blocos lógicos e os blocos de entrada/saída.

Como um microprocessador, o dispositivo *LCA* é um dispositivo de lógica programável. As funções dos blocos lógicos configuráveis *LCA* e blocos de entrada/saída, e sua interconexão, são controlados por um programa de configuração armazenado em uma memória. O programa de configuração é carregado automaticamente de uma memória externa quando em *power-up* ou através de um comando, ou é programado por um microprocessador como parte da inicialização do sistema [33].

2.4.1 Bloco Lógico Configurável

O centro do dispositivo *LCA* é uma matriz de *CLBs* (*Configurable Logic Blocks*) idênticos. Cada *CLB* contém uma lógica combinacional programável e um registrador. A seção lógica combinacional do bloco é capaz de implementar qualquer função *booleana* de suas entradas. A entrada dos registradores pode vir da lógica combinacional ou diretamente de uma entrada *CLB*. As saídas dos registradores podem alimentar a lógica combinacional via um caminho de realimentação interno.

2.4.2 Bloco de Entrada/Saída

A periferia do dispositivo *LCA* é feita de blocos de entrada/saída (*IOBs*) programáveis pelo usuário. Cada bloco pode ser programado independente de ser uma entrada, uma saída com controle *tri-state* ou um pino bidirecional. As entradas podem ser programadas para reconhecer limiar *TTL* ou *CMOS*. Cada *IOB* também inclui *flip-flops* que podem ser usados como *buffers* de entrada e saída.

2.4.3 Interconexões

A flexibilidade do dispositivo *LCA* é devida aos recursos programáveis que controlam a interconexão de quaisquer dois pontos no *chip*. Como outros *Gate Arrays*, os recursos de interconexão *LCA* incluem duas camadas de metal em uma rede de linhas e colunas entre as *CLBs*. As chaves programáveis conectam as entradas e saídas de *IOBs* e *CLBs* perto das linhas de metal.

Chaves nos pontos de cruzamento e permuta nas intersecções de linhas e colunas podem chavear o sinal de um caminho para outro. *Long lines* passam por todo o comprimento e largura do *chip*, passando por cima de intersecções para fornecer a distribuição de sinais críticos com mínimo atraso de *skew*.

3 Descrição da Sequência Utilizada

3.1 Introdução

Por se tratar de um sistema novo, em que a especificação precisa ser testada, optou-se por uma implementação inicial em *FPGA*, que será utilizada para testes de bancada. Porém, como esta implementação implica em um custo elevado e alto consumo de potência, o que seria inviável para este sistema, a implementação final deste circuito será em técnica *Standard Cells* com tecnologia *CMOS* de 0.7μm. Neste trabalho, nos limitaremos a descrição da implementação em *FPGA*.

Neste projeto foi essencial a utilização da metodologia de projeto *Top Down*, devido à complexidade do circuito; necessidade de um curto ciclo de projeto; previsão de mudanças na especifição durante o projeto, pois o sistema ainda se encontra em fase de testes; e, necessidade de um projeto independente de tecnologia, já que a implementação do protótipo é em *FPGA* e do produto final em *Standard Cells*, como dito anteriormente.

A sequência de projeto utilizada nesta metodologia está ilustrada na figura 28. Utilizou-se ferramentas da *Mentor Graphics Corporation* e *Design Kit* da *Xilinx*.

Inicialmente foi feita uma especificação visando o uso do circuito *TB47* no sistema *CLAD* e, em seguida, uma divisão funcional do circuito em blocos, para facilitar a sua validação inicial e, também, a sua visualização como um todo, devido à complexidade do mesmo.

Partiu-se, então, para a descrição *VHDL*, seguida de compilação de cada um dos blocos e, interativamente, uma redefinição destes blocos de acordo com a conveniência.

Com esta partição fechada, foi feita a simulação *VHDL* de cada bloco para validação da sua funcionalidade, utilizando a ferramenta *Quicksim II*.

Com todos os blocos validados, a etapa seguinte consiste em validar a interface entre os mesmos. Para isto, foi feita uma união dos blocos em *VHDL*, seguida de compilação. Possíveis problemas nesta fase, podem implicar na alteração das descrições *VHDL* dos blocos.

Com a geração do modelo *VHDL* completa e validada funcionalmente por meio de simulações, o projeto tem que ser implementado em um nível menor de abstração (Nível de Blocos Funcionais ou Nível de Transferência de Registros), utilizando-se ferramentas de síntese automática [36].

Portanto, com o circuito validado funcionalmente (descrição *VHDL*), passa-se para a fase de síntese e otimização lógica, utilizando-se a ferramenta *Autologic*. Para isto, necessitamos de um *Design Kit* do fabricante (no caso *Xilinx*), que contém uma biblioteca de células. O resultado desta etapa é um esquemático.

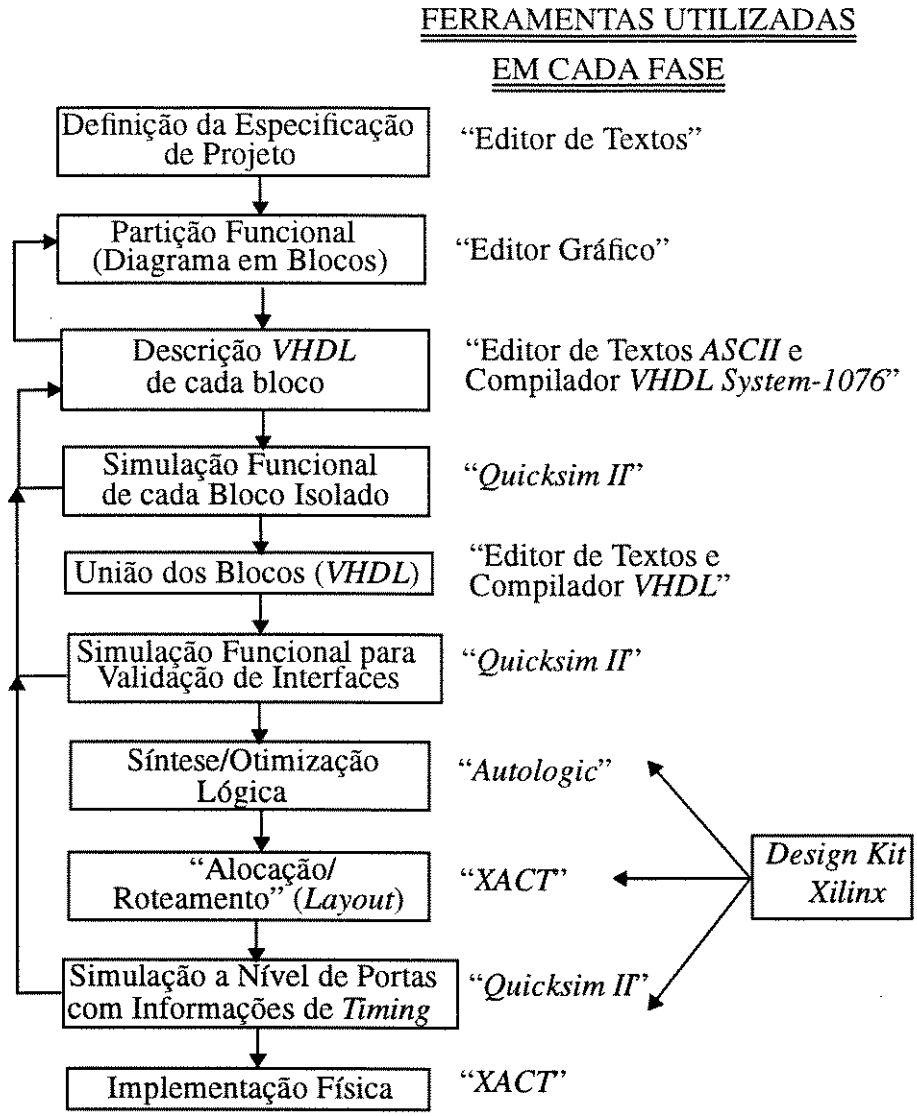


Figura 28: Metodologia de Projeto.

Em uma metodologia de projeto *Top Down*, utilizando-se *VHDL*, é necessária a simulação do circuito em vários níveis de abstração [37, 38]. Portanto, a partir deste esquemático, necessita-se fazer uma simulação com informações de *timing*, utilizando-se o *Quicksim II* e o *Design Kit* da *Xilinx*. Para isto, existe um passo intermediário denominado alocação/roteamento, que consiste em uma distribuição do circuito em cima da estrutura do componente *FPGA Xilinx (layout)*, o que é feito pela ferramenta denominada *PPR*, que faz parte do Sistema de Desenvolvimento *XACT*. Com isto feito, é gerado um arquivo com informações de *timing* dos sinais.

Devido ao fato de que a etapa de alocação/roteamento é aleatória, os atrasos dos sinais gerados nesta fase podem ser muito grandes, tornando-se um problema para o circuito. Portanto, pode ser necessário a repetição desta etapa até se conseguir atrasos satisfatórios.

Finalmente, passou-se para a fase de implementação em dois componentes *Xilinx FPGA (Field Programmable Gate Arrays) 4008PQ208*.

As *FPGAs* são, como o próprio nome diz, programáveis em campo. Portanto, necessitam de uma *EPROM*, onde são gravadas as informações do circuito, e que será conectada com as *FPGAs*. Desta forma, ao ligar o circuito, as *FPGAs* são automaticamente programadas pela *EPROM*, entrando em funcionamento normal, ou seja, prontas para serem utilizadas. Portanto, a implementação consiste na utilização do *software XACT* que gera um arquivo para programação a ser utilizado em um gravador de *EPROM*.

3.2 Descrição das Etapas de Projeto

Aqui serão detalhadas cada uma das etapas mencionadas na figura 28.

3.2.1 Definição da Especificação de Projeto

Esta etapa define a descrição funcional detalhada do circuito, tendo como produto um *Documento de Especificação Técnica*. Este documento é definido juntamente com todas as pessoas envolvidas no sistema e, é utilizado como guia para o projeto.

Detalhes suficientes devem ser incluídos na especificação do *hardware*, tal que a função do dispositivo fique muito bem entendida. Trabalhando com uma boa especificação pode-se diminuir muito o número de interações necessárias para se conseguir um funcionamento correto [24].

A ferramenta utilizada nesta etapa foi um *Editor de Textos*.

3.2.2 Partição Funcional (Diagrama em Blocos)

Antes da codificação em *VHDL*, deve-se fazer uma partição funcional do circuito [39]. O diagrama em blocos pode ser usado como um guia na escrita do código *VHDL* sintetizável. O diagrama em blocos também vai ajudar ao projetista primeiro pensar no *hardware* para depois pensar no *software* (ou *VHDL*) [24].

A figura 29 apresenta o diagrama em blocos do *TB47*.

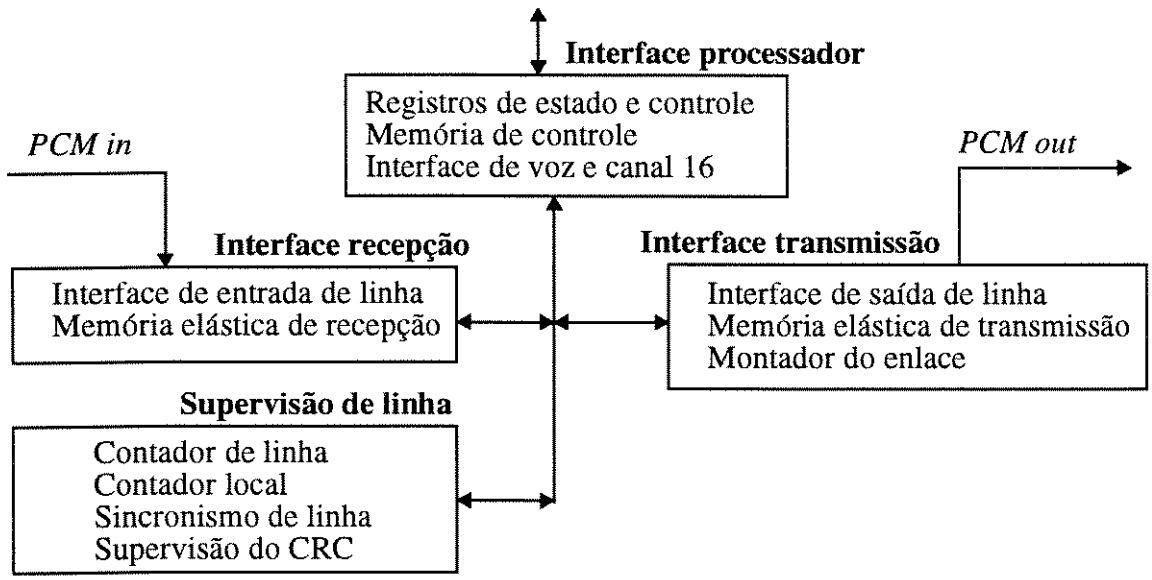


Figura 29: Diagrama em Blocos do *TB47*.

Nele pode-se ver as interfaces do circuito integrado com o sistema controlador; com o *PCM* de entrada; com o *PCM* de saída; e o circuito supervisor da linha. Aqui, utilizou-se um editor com recursos gráficos.

Cada bloco é dividido nos sub-blocos listados no interior de sua respectiva caixa. Nesta seção serão detalhados cada um dos blocos desenvolvidos pelo autor.

A. Sincronismo de Quadro e Multiquadro do Enlace de Recepção

O circuito da figura 30 apresenta um diagrama em blocos para a função de busca de alinhamento de quadro e multiquadro definido na figura 15 e seção 1.4 do capítulo IV. As principais funções consistem da busca de alinhamento de quadro e de multiquadro. Porém, após a entrada em alinhamento de quadro deve monitorar a possível perda de alinhamento de quadro. Para isto, temos os seguintes blocos funcionais:

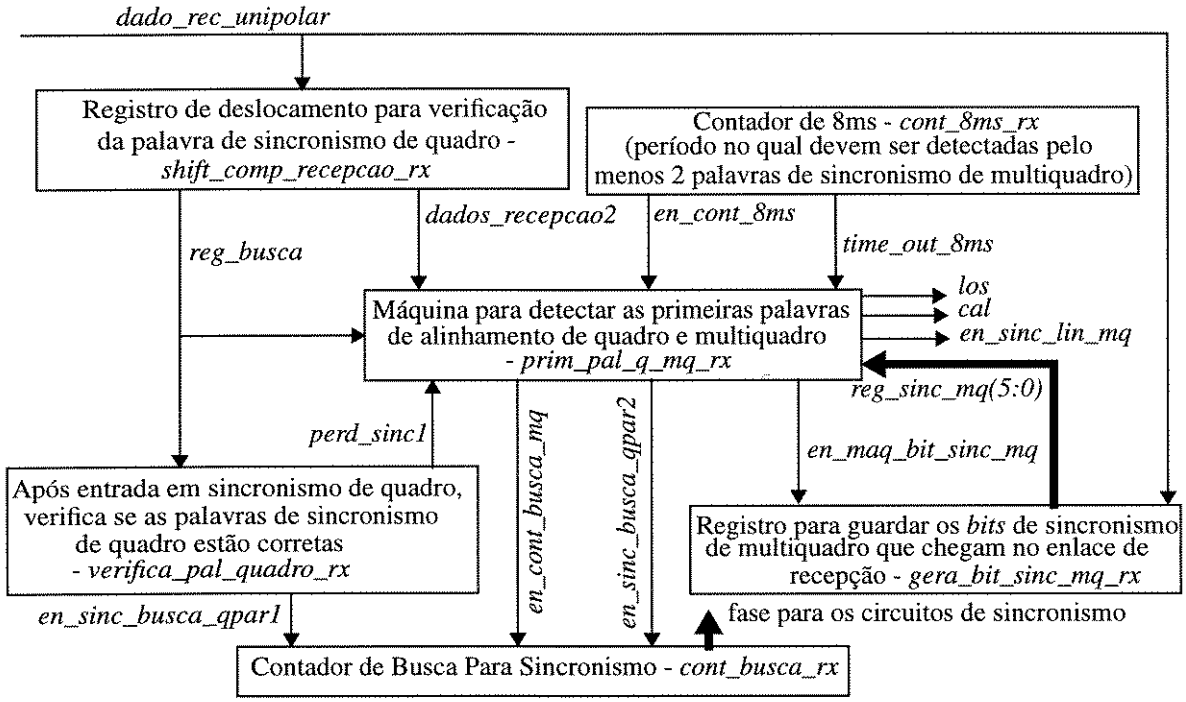


Figura 30: Sincronismo de Quadro e Multiquadro do Enlace de Recepção.

- A busca de alinhamento de quadro e multiquadro utiliza o circuito *prim_pal_q_mq_rx* (Máquina para Alinhamento) para detectar as primeiras palavras de alinhamento de quadro e multiquadro. Para isto, necessita dos seguintes blocos auxiliares:
  - *shift_comp_recepcao_rx* (Verifica Palavra de Alinhamento de Quadro): registro de deslocamento para verificação da palavra de sincronismo de quadro, que se for igual a palavra esperada ativa um sinal que será analisado em *prim_pal_q_mq_rx*.
  - *gera_bit_sinc_mq_rx* (Registro para Geração da Palavra de Alinhamento de Multiquadro): registro de deslocamento para guardar a palavra de multiquadro recebida da linha e que será comparada em *prim_pal_q_mq_rx*.
  - *cont_8ms_rx* (Contador de 8ms): responsável pela monitoração do período máximo de 8ms no qual devem ser detectadas, pelo menos, 2 palavras de sincronismo de multiquadro.

- O monitoramento da possível perda de alinhamento de quadro utiliza o circuito *verifica_pal_quadro_rx* (Máquina Verifica Alinhamento de Quadro), que após entrada em sincronismo de quadro, verifica se as palavras de sincronismo estão corretas. Caso haja 3 falhas consecutivas, deve sair de sincronismo de quadro e forçar o circuito *prim_pal_q_mq_rx* a reiniciar o processo de busca de alinhamento de quadro e multiquadro.
- O circuito *cont_busca_rx* (Contador de Busca para Sincronismo) é um contador de *bit*, canal e quadro, síncrono com o relógio de linha, usado para temporização de toda a lógica descrita acima.

A.1. Máquina para Alinhamento (*prim_pal_q_mq_rx*)

Para detectar as primeiras palavras de alinhamento de quadro e multiquadro foi implementada uma máquina de estado, cujo funcionamento está descrito a seguir.

- *Reset*: como pode ser visto na figura 15 (capítulo IV), esta máquina deve ser iniciada sempre que forem detectadas uma das seguintes condições:
  - *reset* externo assíncrono (pino de entrada *RESQ*);
  - na falta de três palavras de quadro consecutivas corretas (sinal *perd_sinc1* vindo do circuito *verifica_pal_quadro_rx*);
  - não conseguiu alinhar multiquadro em 8ms (sinal *time_out_8ms* vindo do circuito *cont_8ms_rx*);
  - se atingir 915 erros de CRC em 1000 (sinal *perd_sinc2* vindo do circuito *verifica_CRC_rx* - item B.2);
  - sempre que a verificação de taxa de erro for feita pelo processador (*ctr2.r915*='1'), verifica se o processador força ressincronização (*ctr5.frs*='1') (seção 3.1.3 do capítulo IV).
- Estado A: procura primeira palavra de quadro. Verifica o resultado da comparação dos *bits* relativos a palavra de sincronismo de quadro extraídos do enlace de recepção com a palavra de sincronismo de quadro, vindo do circuito *shift_comp_recepcao_rx*. Durante o intervalo de tempo do *bit* no qual foi detectada a palavra de sincronismo de quadro, é forçada a sincronização do contador de busca.
- Estado B: verifica se o *bit* 2 do canal seguinte é igual a '1'.
- Estado C: verifica a segunda palavra de alinhamento de quadro. Aguarda o intervalo de tempo de canal 0 e verifica se a comparação com a palavra de sincronismo de quadro foi correta. Durante o intervalo de tempo do *bit* no qual foi detectada a palavra de sincronismo, é forçada a sincronização do contador de busca.
- Estado D: entra em sinronismo de quadro (indicado pelo sinal de saída *los*='0') e passa a procurar a primeira palavra de multiquadro. Libera o registro do circuito *gera_bit_sinc_mq_rx* que gera a sequência dos *bits* da palavra de sincronismo de multiquadro. Compara, *bit* a *bit*, os *bits* 1 dos canais 0 ímpares com a palavra de sincronismo de multiquadro. Caso todos os *bits* coincidam na sequência correta, passa para E. O contador de 8ms (*cont_8ms_rx)* é liberado para contagem.
- Estado E: primeira palavra de sincronismo de multiquadro foi detectada corretamente.

Este estado corresponde ao intervalo de tempo do quadro 11. Os estados E, G e H aguardam a passagem dos útimos canais deste multiquadro. Passa para G no intervalo de tempo do quadro 13. As saídas nos estados E e G são idênticas.

- Estado G: no intervalo de tempo do quadro 15 passa para H.
- Estado H: início do canal 0, do quadro 0 e do multiquadro seguinte. Durante intervalo de tempo do *bit* 1, sincroniza o contador de busca, a nível de multiquadro.
- Estado I: procura a segunda palavra de sincronismo de multiquadro. Procura, *bit* a *bit*, comparando o *bit* 1 dos canais 0 ímpares com a palavra de multiquadro. Em caso de sucesso vai para J. Caso não sejam verificadas duas amostras corretas em 8ms, volta ao estado A, via sinal *time_out_8ms*, que força esta máquina a voltar ao estado inicial.
- Estado J: foram encontradas duas amostras boas em menos de 8ms. Aguarda final do multiquadro e vai para L. Durante intervalo de tempo do *bit* 8, do canal 31 e do quadro 15 sincroniza o contador de linha, a nível de multiquadro (via sinal de saída *en_sinc_lin_mq*).
- Estado L: entrou em sincronismo de multiquadro. Só sai deste estado para o A se forem detectadas algumas das possíveis condições de *reset*, definidas anteriormente. O sinal *int2.cal*='0' indica que está em sincronismo de multiquadro. Enquanto em sincronismo de multiquadro, libera a máquina *verifica_CRC_rx* - item B.2.

A.2. Verifica Palavra de Alinhamento de Quadro (*shift_comp_recepcao_rx*)

Registro de deslocamento para verificação da palavra de sincronismo de quadro, cujo funcionamento está descrito a seguir.

- Se o pulso de *reset* (pino *RESQ*) estiver ativo, o registro que contém os dados de recepção é zerado. Caso contrário, a cada pulso de relógio é deslocado o dado de recepção unipolar.
- Sempre que o registro que contém os dados de recepção coincidir com a palavra de alinhamento de quadro ("1101100"), o sinal *reg_busca* é ativado (este sinal será utilizado no circuito *prim_pal_q_mq_rx*).

A.3. Registro para Geração da Palavra de Alinhamento de Multiquadro (*gera_bit_sinc_mq_rx*)

Registro de deslocamento da palavra de alinhamento de multiquadro, cujo funcionamento está descrito a seguir.

- Registro para guardar os *bits* de sincronismo de multiquadro que chegam no enlace de recepção. Quando do *reset* externo (*RESQ*) ou enquanto fora de sincronismo de quadro (*en_maq_bit_sinc_mq*='0', vindo do circuito *prim_pal_q_mq_rx*), o registro fica zerado. Fora destas condições, é deslocado todo *bit* 1 de canal 0 ímpar do contador de busca.

A.4. Contador de 8ms (*cont_8ms_rx*)

Contador para detecção do tempo máximo de 8ms, cujo funcionamento está descrito a

seguir.

- *Reset*: externo, via pino *RESQ* (assíncrono) ou enquanto o contador de 8ms não estiver habilitado (*en_cont_8ms*='0' - sinal síncrono vindo do circuito *prim_pal_q_mq_rx*). Se vier um pulso de *reset*, o *cont_8ms_rx* é zerado. Caso contrário, é incrementado a cada final de canal 0 de todo quadro.
- Após o final da contagem de 8ms e quando chegar no *bit* 8, canal 0 par do contador de busca, é gerado o *time out* do contador de 8ms (sinal *time_out_8ms*), que será usado no circuito *prim_pal_q_mq_rx*.

A.5. Verifica Palavra de Alinhamento de Quadro (*verifica_pal_quadro_rx*)

Para a monitoração do alinhamento de quadro, foi implementada uma máquina de estado, cujo funcionamento está descrito a seguir.

- *Reset*: apenas o *reset* externo assíncrono (*RESQ*).
- Estado A: aguarda *los*='0', ou seja, entrada em sincronismo de quadro.
- Estado B: caso *los*='1' (fora de sincronismo), por qualquer motivo determinado na máquina *prim_pal_q_mq_rx*, volta ao estado A. Caso contrário, se a palavra de sincronismo recebida estiver correta, permanece em B. No caso de receber palavra incorreta vai para C. A cada palavra de sincronismo de quadro correta no *bit* 8 canal 0 par do contador de busca, é gerado o sinal *en_sinc_busca_qpar1*, que irá ressincronizar o contador de busca.
- Estado C: caso *los*='1', por qualquer motivo determinado na máquina *prim_pal_q_mq_rx*, volta ao estado A. Caso contrário, se a palavra de sincronismo recebida estiver correta, volta para B. No caso de receber a segunda palavra incorreta vai para D. Os sinais gerados são idênticos aos do estado B.
- Estado D: caso *los*='1', por qualquer motivo determinado na máquina *prim_pal_q_mq_rx*, volta ao estado A. Caso contrário, se a palavra de sincronismo recebida estiver correta, volta para B. No caso de receber a terceira palavra incorreta vai para E. Os sinais gerados são idênticos aos do estado B.
- Estado E: perdeu três palavras de quadro consecutivas, logo, sai de sincronismo de quadro. Gera *perd_sinc1*='1' que irá fazer a máquina *prim_pal_q_mq_rx* se reinicializar, gerando *los*='1', que faz esta máquina (*verifica_pal_quadro_rx*) também se reinicializar.

A.6. Contador de Busca Para Sincronismo (*cont_busca_rx*)

Contador utilizado na busca de sincronismo de quadro, cujo funcionamento está descrito a seguir.

- Se o sinal de *reset* do pino de entrada *RESQ* estiver ativo, o contador de busca é zerado. Caso contrário, se um dos sinais que habilitam o sincronismo de quadro par (sinais *en_sinc_busca_qpar1* e *en_sinc_busca_qpar2*, dos circuitos *verifica_pal_quadro_rx* e *prim_pal_q_mq_rx*, respectivamente) estiver ativo, o contador será sincronizado a nível de *bit*, canal e *cont_busca_quadro(0)*. Se não, o contador de *bit* é incrementado

em todo pulso de relógio, o contador de canal é incrementado a cada 8 *bits*. Para o contador de quadro, existe o sinal de habilitação *en_cont_busca_mq* vindo do circuito *prim_pal_q_mq_rx*, que se for igual a '1', a cada 32 canais o contador de quadro é incrementado; se '0', o contador só diz se é quadro par ou ímpar (*bit cont_busca_quadro(0)*) a cada 32 canais, os outros *bits* do contador de quadro permanecem em '0'.

- São decodificados vários intervalos de tempo do contador de busca, para uso nas outras lógicas do circuito.

B. Circuitos de CRC da Recepção

O circuito da figura 31 apresenta um diagrama em blocos para a função de tratamento de CRC de recepção, conforme definido na seção 1.3 do capítulo IV. As funções principais consistem da extração dos *bits* C_n do enlace de recepção, geração do CRC referente aos submultiquadros recebidos, comparação do CRC gerado com os *bits* C_n recebidos, verificação do contador de erros de CRC para avaliação da taxa de erro na linha e para o fluxograma de acompanhamento do sincronismo de quadro (figura 15 do capítulo IV). Para isto, temos os seguintes blocos funcionais:

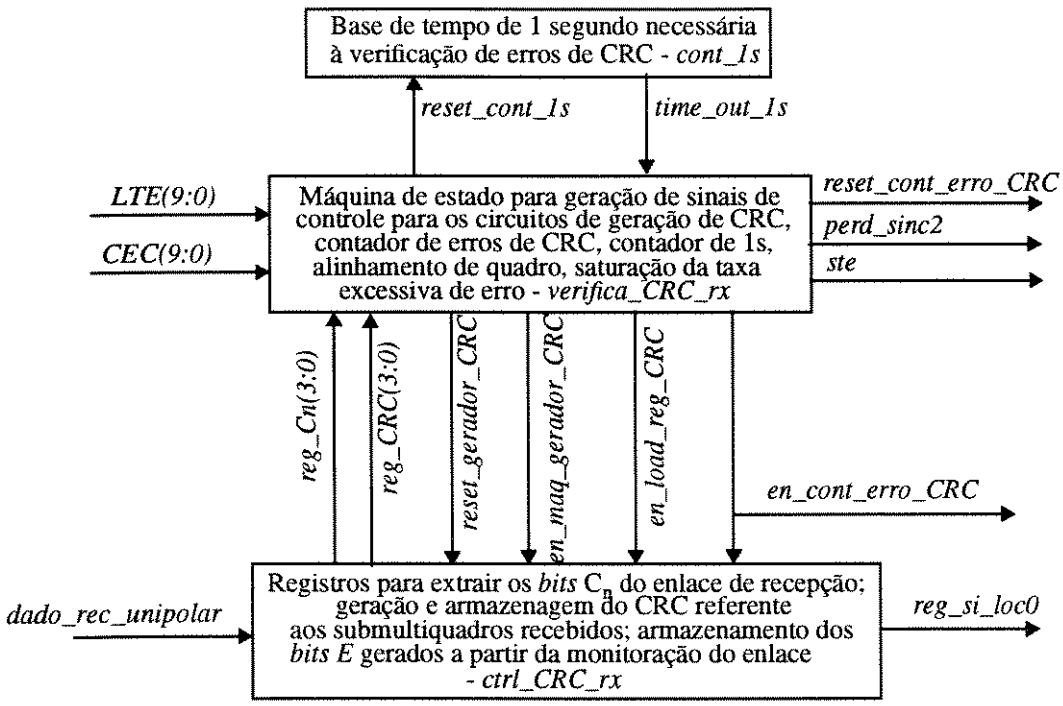


Figura 31: Circuitos de CRC da Recepção.

- *ctrl_CRC_rx* (Circuitos de Controle do CRC na Recepção): contém contadores e registros necessários à verificação do CRC na recepção, ou seja, extrai os *bits* C_n do enlace de recepção, gera e armazena o CRC referente aos submultiquadros recebidos e armazena o resultado da comparação do CRC gerado com o recebido, para posterior inserção nos *bits E* do enlace a ser transmitido.
- *verifica_CRC_rx* (Verifica CRC na Recepção): faz a comparação do CRC gerado com o recebido, gera sinais de controle para os circuitos *ctrl_CRC_rx*, *cont_1s*,

prim_pal_q_mq_rx, habilitação para o contador de erros de CRC e define a saturação da taxa excessiva de erro.

- *cont_1s* (Contador de 1s): define uma base de tempo de 1s para verificação de erros de CRC.

B.1. Circuitos de Controle do CRC (*ctrl_CRC_rx*)

Define contadores e registros necessários à verificação do CRC na recepção. O seu funcionamento está descrito a seguir.

- Armazena Registro C_n: se vier um pulso de *reset* externo assíncrono (*RESQ*), o registro C_n é zerado. Caso contrário, em todo *bit* 1, canal 0 par do contador de busca é deslocado o dado de recepção no formato unipolar.
- Armazena Registro CRC: se vier um pulso de *reset* externo assíncrono (*RESQ*), o registro de CRC é zerado. Caso contrário, sempre que for habilitado pelo sinal *en_load_reg_CRC* do circuito *verifica_CRC_rx* , ele é carregado com o CRC calculado.
- Armazena *bit E* Local: registro de deslocamento para armazenamento dos *bits E* gerados a partir da monitoração do enlace. O sinal *en_cont_erro_CRC* do circuito *verifica_CRC_rx* indica a ocorrência ou não de erros de CRC na recepção.
- Gerador de CRC: prepara o sinal para entrar no gerador de CRC. Deve ser zerado no intervalo de tempo de *bit* 1, canal 0 par do contador de busca. Se vier um pulso de *reset* (*RESQ* ou o sinal *reset_gerador_CRC* do circuito *verifica_CRC_rx*), o CRC calculado é zerado. Caso contrário, sempre que for habilitada a geração do CRC (sinal *en_maq_gerador_CRC* do circuito *verifica_CRC_rx*) o mesmo é calculado.

B.2. Verifica CRC na Recepção (*verifica_CRC_rx*)

Foi implementada uma máquina de estado, cujo funcionamento está descrito a seguir.

- *Reset*: sinal *cal* (alinhamento de multiquadro) vindo da máquina *prim_pal_q_mq_rx* ou sinal de *reset* externo assíncrono (*RESQ*).
- Estado A: aguarda início do primeiro submultiquadro 2 e passa para B.
- Estado B: fica neste estado por todo submultiquadro 2. Libera o circuito gerador de CRC.
- Estado C: chega a este estado no início do multiquadro, habilitando o circuito *ctrl_CRC_rx* a ter uma amostra válida em *reg_CRC* (registro com o CRC calculado), iniciando assim, o procedimento cíclico de geração e armazenagem (em *reg_CRC*) do CRC dos submultiquadros seguintes; armazenagem dos *bits* C_n recebidos (em *reg_Cn*) para comparação com *reg_CRC* e armazenagem dos *bits E* calculados. Habilita também a contagem dos eventuais erros de CRC detectados. Todas estas atividades são feitas em paralelo e, estão descritas a seguir.
  - Caso a base de tempo esteja sob controle do *TB47* (*ctr2.r915*='0' - seção 3.1.3 do capítulo IV), se vencer o período de 1 segundo *time_out_1s*='1' (sinal vindo do circuito *cont_1s*) o sinal *ste* (saturação da taxa excessiva de erro) é feito igual a zero. Caso contrário, a cada fim de submultiquadro é verificado se o contador de

erros de CRC atingiu o valor programado no registro *LTE* (limiar para alarme da taxa excessiva de erro), caso positivo, faz *ste*='1' para sinalizar o alarme.

- A cada final do último canal 0 de cada submultiquadro, caso haja diferença entre o CRC armazenado em *reg_CRC* e os *bits* recuperados em *reg_Cn* é gerado habilitação para o contador de erros de CRC (sinal de saída *en_cont_erro_CRC*).
- É avaliado o número de erros de CRC. Caso seja igual a 915 deve ser forçada a perda de sincronismo de quadro, feita pelo sinal de saída *perd_sinc2* para a máquina *prim_pal_q_mq_rx*. Porém, se estiver no modo teste o valor limite do contador é menor, para facilidade de simulação.
- Verifica se a base de tempo para a monitoração da taxa de erro está sob controle do processador (*ctr2.r915*='1'), ou se deve ser feita com base no *cont_1s*. O sinal *ctr2.r915*='1', trava *cont_1s* (via sinal de saída *reset_cont_1s*) fazendo com que o sinal *time_out_1s* nunca seja igual a '1'.
- A cada começo de submultiquadro é gerado habilitação para carregar o CRC calculado no *reg_CRC* (sinal de saída *en_load_reg_CRC*); o gerador de CRC deve ser zerado para poder ser reiniciado o processo de geração (sinal de saída *reset_gerador_CRC*).
- Quando vence o período de 1 segundo o contador de erros de CRC deve ser zerado para iniciar novamente o período de amostragem (sinal de saída *reset_cont_erro_CRC*).
- A geração cíclica do CRC dos submultiquadros seguintes está sempre habilitada.

B.3. Contador de 1s (*cont_1s*)

Define o contador de 1s necessário à verificação de erros de CRC. Base de tempo de 1 segundo. Conta 500 multiquadros (2ms cada). O seu funcionamento está descrito a seguir.

- Se vier um pulso de *reset* (assíncrono *RESQ* ou o sinal síncrono *reset_cont_1s* do circuito *verifica_CRC_rx*), o *cont_1s* é zerado. Caso contrário, é incrementado a cada começo de multiquadro (2ms), até atingir 1s.
- No final da contagem de 1s e no começo de um multiquadro, é gerado o *time out* do contador de 1s (sinal *time_out_1s*).

C. Contadores Necessários ao Circuito

Aqui é definida toda a temporização do *TB47* (figura 32), onde o contador de linha define a temporização necessária aos circuitos síncronos com a linha e o contador local define a temporização necessária aos circuitos síncronos com a placa controladora. O contador de linha é sincronizado pelo contador de busca (seção A.6), após a entrada em sincronismo.

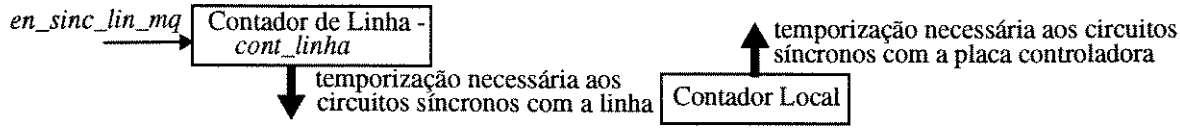


Figura 32: Contadores Necessários ao Circuito.

No intervalo de tempo de canal N do contador de linha, o enlace recebido e o enlace a ser transmitido devem conter este mesmo canal N. Portanto, para satisfazer aos requisitos do *CLAD*, usando duas memórias elásticas, os circuitos de temporização de linha e local devem respeitar as relações de fase mostradas na figura 33.

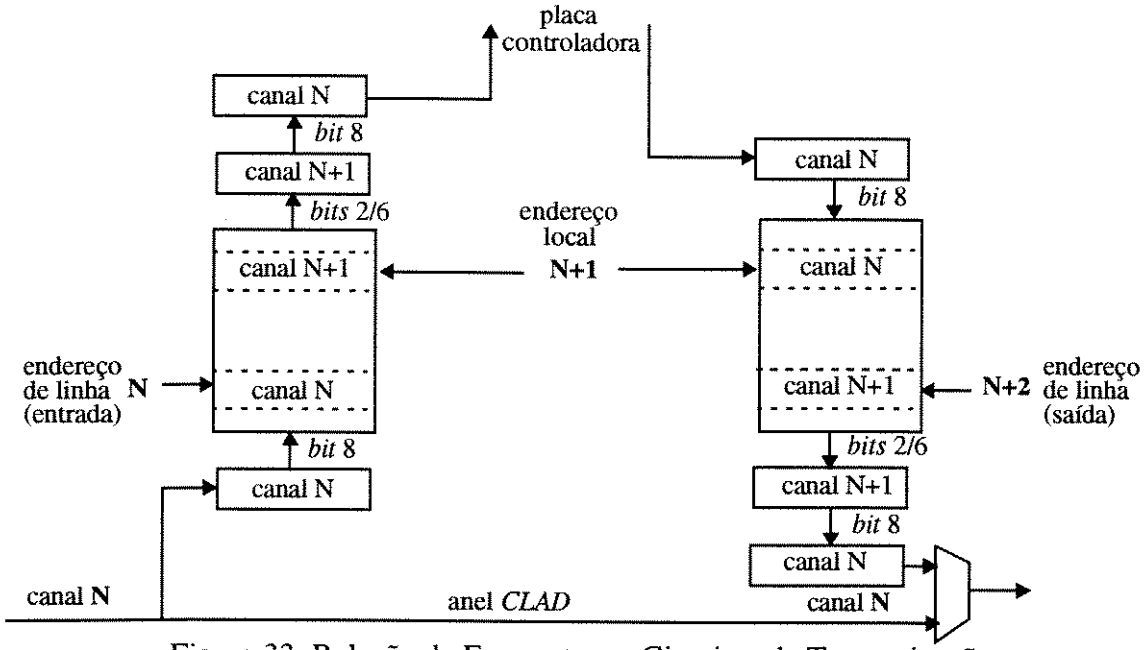


Figura 33: Relação de Fase entre os Circuitos de Temporização.

Deste modo, as memórias elásticas necessitam de três temporizações distintas:

- Temporização de linha na recepção: contador exatamente sincronizado a nível de canal com o enlace de recepção.
- Temporização local: contador adiantado de um canal com relação ao sincronismo local (*SYPQ* ou *USYPQ*).
- Temporização de linha de transmissão: Contador adiantado em dois canais com relação ao sincronismo de linha (*RSIGM*).

A temporização local foi implementada no contador local e as temporizações de linha para a recepção e a transmissão foram implementadas no contador de linha. A implementação do contador local não será apresentada aqui.

C.1. Contador de Linha (*cont_linha*)

Contador de *bit*, canal e quadro responsável por, no modo de funcionamento normal, gerar todos os sinais com base na temporização de linha (*RRCLK_4M* e *en_sinc_lin_mq* de *prim_pal_q_mq_rx* - seção A.1) para todo o *TB47*. No modo de funcionamento em *loop* de reconfiguração, sua temporização é comutada para SCLK e *SYPQ*. Seu funcionamento está descrito a seguir.

- Controle do *Loop* de Reconfiguração: multiplexação, controlada pelo sinal *ctr2.lre* (*loop* de reconfiguração), dos sinais de *reset* e relógio a serem usados pelo próprio contador de linha.
- Os sinais de sincronismo de quadro (*en_sinc_lin_mq* e *SYPQ*) são usados como *reset*

síncronos do contador de forma a garantir que, a cada quadro, este contador esteja em fase com a temporização de linha, ou local (se em *loop* de reconfiguração).

- Se o sinal de *reset* estiver ativo (assíncrono ou síncrono), o contador de linha é zerado. Caso contrário, a cada pulso de relógio, o contador de *bit* é incrementado; a cada 8 *bits*, o contador de canal é incrementado; e a cada 32 canais o contador de quadro é incrementado.
- Decodificação dos intervalos de tempo necessários, para os demais circuitos do *TB47*.
- O contador de transmissão (*cont_tr*) tem uma defasagem de 2 canais em relação a recepção.

D. Montagem do Enlace de Transmissão

O circuito da figura 34 apresenta um diagrama em blocos para a função de montagem do enlace a ser transmitido. As funções principais consistem em montagem do canal 0, canal 16 e canais de voz. Deve prever, também, as funções de inserção de um padrão definido em qualquer canal, tratamento do CRC para transmissão, inserção de *AIS*, *loop* de reconfiguração e *bypass*. Para isto, temos os seguintes blocos funcionais:

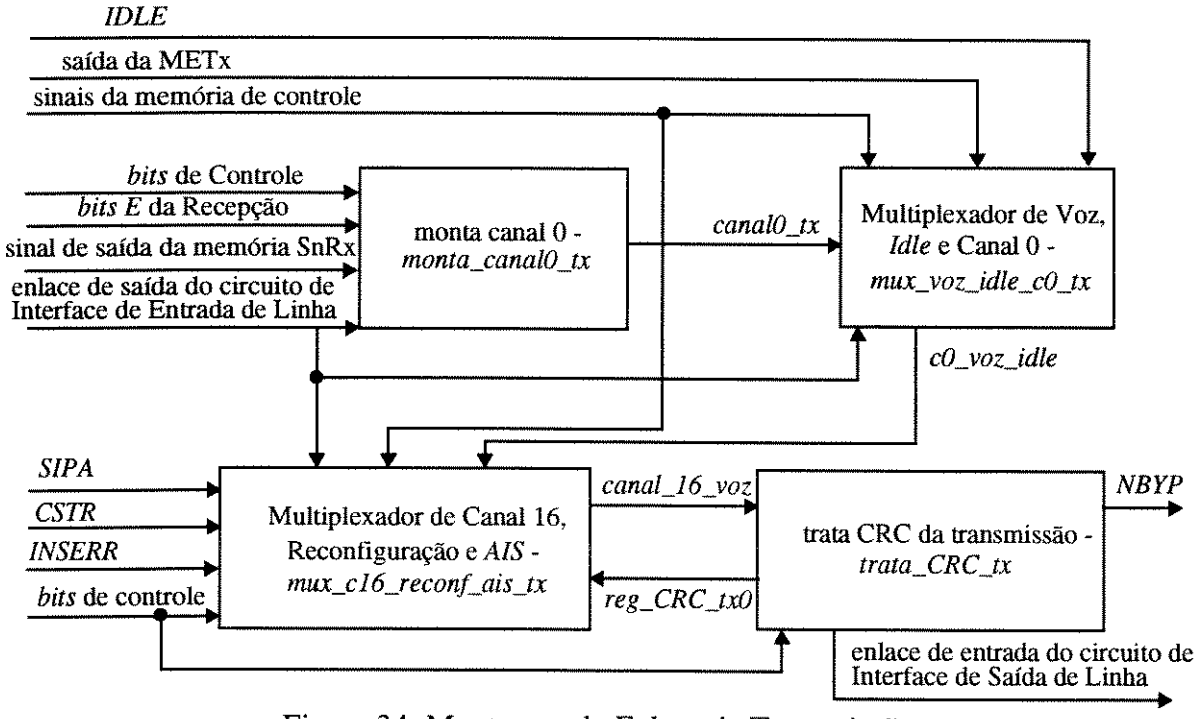


Figura 34: Montagem do Enlace de Transmissão.

- *monta_canal0_tx* (Monta Canal 0): responsável por gerar a palavra de sincronismo de quadro e multiquadro e multiplexação do canal 0.
- *mux_voz_idle_c0_tx* (Multiplexador de Voz, *Idle* e Canal 0): responsável pela multiplexação dos canais de voz alocados ou não alocados, canal 0 e conteúdo do registro *IDLE*, se habilitado pela memória de controle (seção 2.4 do capítulo IV), nos intervalos de tempo pertinentes.
- *mux_c16_reconf_ais_tx* (multiplexador de Canal 16, Reconfiguração e *AIS*): responsável por gerar o sinal indicativo de *bypass* (pino *NBYP*), inserção de erro no

enlace de transmissão, sob controle do pino de entrada *INSERR* (seção 7 do capítulo IV), controle do *bypass* do enlace de recepção, inserção de *AIS*, *loop* de reconfiguração do enlace e controle do canal 16.

- *trata_CRC_tx* (Trata CRC da Transmissão): responsável pelo tratamento do CRC do enlace a ser transmitido.

D.1. Monta Canal 0 (*monta_canal0_tx*)

Este circuito é composto de duas máquinas de estado para geração das palavras de sincronismo de quadro e multiquadro e multiplexadores para montagem do canal 0. O seu funcionamento está descrito a seguir.

- Geração da Palavra de Sincronismo de Multiquadro: máquina de estado que gera, a cada estado a sequência de *bits* definida como palavra de sincronismo de multiquadro ("001011"). Mudanças de estado são feitas no fim dos intervalos de tempo de *bit* 1, canal 0 ímpar do contador de linha. O sinal que indica final da palavra de multiquadro (*fim_sinc_mq*) é usado para liberar o envio (nos canais 13 e 15) dos *bits E*.
  - *RESET*: somente o pino de *reset* externo assíncrono (*RESQ*).
  - Estado A: faz a saída *bit_sinc_mq*='0' (primeiro *bit* da palavra de sincronismo de multiquadro). O sinal *fim_sinc_mq* só fica ativo no estado G.
  - Estado B: faz a saída *bit_sinc_mq*='0' (segundo *bit* da palavra de sincronismo de multiquadro).
  - Estado C: faz a saída *bit_sinc_mq*='1' (terceiro *bit* da palavra de sincronismo de multiquadro).
  - Estado D: faz a saída *bit_sinc_mq*='0' (quarto *bit* da palavra de sincronismo de multiquadro).
  - Estado E: faz a saída *bit_sinc_mq*='1' (quinto *bit* da palavra de sincronismo de multiquadro).
  - Estado F: faz a saída *bit_sinc_mq*='1' (sexto *bit* da palavra de sincronismo de multiquadro).
  - Estado G: gera *fim_sinc_mq*='1', que permanece ativo até um *reset* da máquina pelo sinal *bit8c31q15_lin*, vindo do circuito contador de linha.
- Gera a Palavra de Sincronismo de Quadro: máquina de estado que gera, a cada estado a sequência de *bits* definida como palavra de sincronismo de quadro ("0011011").
  - *Reset*: somente o pino de *reset* externo assíncrono (*RESQ*).
  - Estado A: aguarda fim do intervalo de tempo de *bit* 1, canal 0 par do contador de linha, que equivale ao início da palavra de sincronismo de quadro.
  - Estado B: faz a saída *bit_sinc_q*='0' (primeiro *bit* da palavra de sincronismo de quadro).
  - Estado C: faz a saída *bit_sinc_q*='0' (segundo *bit* da palavra de sincronismo de quadro).
  - Estado D: faz a saída *bit_sinc_q*='1' (terceiro *bit* da palavra de sincronismo de

quadro).

- Estado E: faz a saída *bit_sinc_q*='1' (quarto *bit* da palavra de sincronismo de quadro).
- Estado F: faz a saída *bit_sinc_q*='0' (quinto *bit* da palavra de sincronismo de quadro).
- Estado G: faz a saída *bit_sinc_q*='1' (sexto *bit* da palavra de sincronismo de quadro).
- Estado H: faz a saída *bit_sinc_q*='1' (sétimo *bit* da palavra de sincronismo de quadro).

- Multiplexador do Canal 0 (nos intervalos de tempo do contador de linha).
  - Canal 0 par: durante o *bit* 1 é transmitido '0' no enlace de transmissão. Durante os demais *bits*, é transmitida a palavra de sincronismo de quadro.
  - Canal 0 ímpar:
    - *Bit* 1: o sinal *fim_sinc_mq*='1' indica que a palavra de sincronismo de multiquadro já foi completamente gerada, ou seja, enquanto for '0' deve ser transmitida esta palavra, quando for '1', devem ser transmitidos (nos quadros 13 e 15) os *bits E*. Neste caso, é preciso verificar: se o circuito não estiver em sincronismo de quadro é inserido '0' em *E*; caso contrário, verificar o valor dos *bits E* recebidos, que se '0', deve-se transmitir '0'; se '1', deve-se transmitir o resultado dos *bits E* gerado localmente.
    - *Bit* 2: no intervalo de tempo do *bit* 2 é sempre inserido '1'.
    - *Bit* 3: no caso do intervalo de tempo do *bit* 3 do canal 0 ímpar (*bit* alarme remoto), se na recepção ele estiver ativo (igual a '1'), é transmitido '1', caso esteja inativo, deve-se verificar o modo de funcionamento (*bit ctr2.exra*). No caso de *ctr2.exra* ser igual a '0' (modo transparente), deve-se transmitir o enlace de recepção; caso contrário, se *ctr2.exra* for igual a '1', deve-se transmitir o valor de *ctr2.vxra* .
    - *Bits* 4 a 8: caso dos últimos 5 *bits* do canal 0 ímpar (*bits* S_n). Caso *ctr1.esn* seja igual a '0', trata-se do modo transparente. Caso *ctr1.esn* seja igual a '1' deve-se esperar o início do próximo multiquadro, para então enviar os *bits* S_n do registro de transmissão; e, enquanto isto deve-se enviar *dado_rec_unipolar* nos *bits* S_n.

D.2. Multiplexador de Voz, *Idle* e Canal 0 (*mux_voz_idle_c0_tx*)

Responsável pela montagem do enlace contendo o canal 0, os canais de voz alocados e não alocados e a possível inserção de um conteúdo programável (*IDLE*) em qualquer canal. O seu funcionamento está descrito a seguir.

- É verificado o estado dos *bits* de controle residentes na memória de controle (seção 2.2 do capítulo IV):
  - Se *reg_ctrl(6)*='1', enviar *IDLE*.
  - Se *reg_ctrl(6)*='0', durante o intervalo de tempo de canal 0, sai o canal 0 montado no bloco Multiplexador do Canal 0 e, nos demais canais, caso o canal esteja alocado

(*bits reg_ctrl(3:0)* ativos em '1'), sai o sinal *dados_sai_tr1*, da memória elástica de transmissão, e caso o canal não esteja alocado, sai o enlace de recepção (*dado_rec_unipolar*). Os *bits* de controle *reg_ctrl(3:0)* definem a alocação ou não de pares de *bits*, de forma a permitir o tráfego de canais a taxas de 64, 32 ou 16Kbits/s.

D.3. Multiplexador de Canal 16, Reconfiguração e *AIS* (*mux_c16_reconf_ais_tx*)

Circuito responsável pela geração final do enlace a ser transmitido, cujo funcionamento está descrito a seguir.

- Sinal indicativo de *bypass* ativo: ativa o pino de saída *NBYP* sempre que o pino *SIPA* ou o *bit* do registro de controle *ctr1.byp* estiverem ativos.
- Lógica XOR para poder inserir erros no enlace de transmissão sob controle do pino *INSERR* no enlace a ser transmitido.
- O intervalo de tempo de canal 16, pode ser constituído de somente o intervalo de tempo do canal 16, como também do intervalo de tempo dos canais 16 e 17 ou canais 16, 17 e 18, de acordo com os *bits* de controle *ctr3.c16a* e *ctr3.c16b*. Se *ctr3.c16a*='0' e *ctr3.c16b*='0', é transmitido o enlace *c0_voz_idle* (do circuito *mux_voz_idle_c0_tx*) atrasado de um *bit*, inclusive no intervalo de tempo de canal 16; qualquer outra condição, no intervalo de tempo de canal 16 atrasado de um *bit*, será transmitido o sinal do pino *CSTR* e nos demais intervalos de tempo continua sendo transmitido o enlace *c0_voz_idle* atrasado de um *bit*.
- No *loop* de reconfiguração, se o *bit ctr2.lre*='1', deve ser enviado o enlace de saída da memória elástica de transmissão (sinal *dados_sai_tr1*). Caso contrário, enviar o enlace de saída *reg_CRC_tx0* do circuito *trata_CRC_tx*.
- *AIS*: se *ctr1.xais*='1', envia "tudo um" para a entrada do circuito Interface de Saída.
- *Bypass*: este é o último estágio de multiplexação para montagem do enlace de transmissão. O sinal de saída do circuito de interface de entrada (sinal *dado_rec_unipolar*) é atrasado de 1 *bit* e conectado na entrada do circuito de interface de saída.

D.4. Trata CRC da Transmissão (*trata_CRC_tx*)

Circuito para tratamento do CRC do enlace a ser transmitido, cujo funcionamento está descrito a seguir.

- Inserção de Erro de CRC: se estiver no *bit* 2, canal 0 par de linha libera o CRC (se *ctr3.crci*='0') ou libera o CRC invertido (se *ctr3.crci*='1'). Nos demais intervalos de tempo libera o enlace para transmissão.
- Máquina de Estado para Controle do CRC de Transmissão.
  - *Reset*: somente o pino de *reset* externo (*RESQ*).
  - Estado A: aguarda início do próximo submultiquadro e passa para B.
  - Estado B: fica neste estado por todo o submultiquadro. Libera o gerador de CRC e

no fim do multiquadro armazena o resultado em *reg_CRC*.

- Estado C: chega a este estado no início do multiquadro, tendo uma amostra válida em *reg_CRC*. Assim, inicia-se o procedimento cíclico de geração e armazenagem (em *reg_CRC*) do CRC dos submultiquadros seguintes.
  - Fica no estado C até que haja um *reset* via *RESQ*.
  - A cada final de submultiquadro é gerada habilitação para carregar o CRC calculado no *reg_CRC*.
  - Nos *bits* que se seguem aos intervalos de tempo onde foi feita a carga em *reg_CRC*, deve ser zerado o gerador de CRC para poder ser reiniciado o processo de geração.
- Registro CRC de Transmissão: registro de deslocamento onde é armazenado o CRC calculado para um submultiquadro, e que vai ser enviado no submultiquadro seguinte. Se vier um pulso de *reset*, o registro de CRC é zerado. Caso contrário, sempre que for habilitado para carregar no registro, ele é carregado com o CRC calculado. Caso não esteja habilitado para ser carregado no registro, estará para deslocamento, o que é feito ao final de todo *bit* 1, canal 0 par de linha.
- Gerador de CRC de Transmissão: se vier um pulso de *reset* (*RESQ* - assíncrono ou *reset_gerador_CRC* - síncrono), o CRC calculado é zerado. Caso contrário, sempre que for habilitada a geração do CRC o mesmo é calculado, conforme o circuito definido na figura 14 do capítulo IV.

E. Registros/Memória para Comunicação com a Placa Controladora (*Status*/Controle)

A figura 35 representa todas as funções que necessitam de interface com a placa controladora (seção 3.1 do capítulo IV). Aqui, somente serão tratados os circuitos registro de controle e controle das memórias S_n de recepção. Os registros de controle representam o controle do *TB47* pela placa controladora e as memórias S_n de recepção são responsáveis pelo tratamento dos *bits* 4 a 8 dos canais ímpares da estrutura de multiquadro CRC-4, prevendo uma possível comunicação entre a central de comutação e a placa controladora, via estes *bits* (esta função não é utilizada hoje pelo sistema).

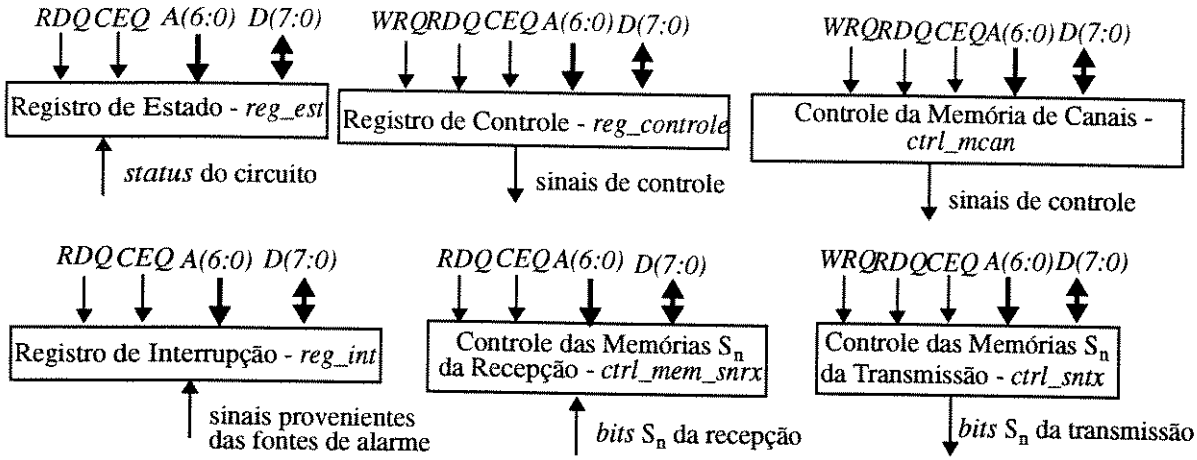


Figura 35: Registros/Memórias para Comunicação com a Placa Controladora (*Status*/ Controle).

E.1. Registros de Controle (*reg_controle*)

Este circuito permite à placa controladora enviar *bits* de controle para o *TB47*. O seu funcionamento está descrito a seguir.

- Decodificação dos endereços dos registros de controle, com posterior geração de sinais de habilitação a partir do pino de entrada *Chip Enable* (*CEQ*).
- Escrita nos Registros: se o pino de *reset RESQ* estiver ativo, todos os *bits* de controle são inicializados. Caso contrário, no começo de todo pulso de escrita, e se algum registro de controle estiver sendo endereçado, o *bit* do registro de controle recebe o *bit* correspondente do barramento de dados. Os *bits* de controle *ctr5.frs* e *ctr4.txsn* possuem sinais de *reset* diferentes de *RESQ*, que são *reset_frs* e *reset_txsn*, respectivamente, e são gerados pelo *hardware* (seção 3.1.3 do capítulo IV).
- Registro *IDLE:* registro de deslocamento circular. O registro será deslocado nos casos em que a memória de controle (seção 2.2 do capítulo IV) comandar seu envio para o enlace de transmissão (*bit* 6) ou para o *loop* interno (*bits* 4 e 5) e, caso não esteja sendo deslocado, ele é carregado com o valor do registro *IDLE* escrito pelo processador.
- Leitura dos Registros: se um registro estiver sendo endereçado (*en_reg*='1'), o barramento de dados recebe os *bits* referentes ao registro. Se o registro não estiver habilitado, o barramento de dados fica em *tri-state*.

E.2. Controle das Memórias S_n na Recepção (*ctrl_mem_snrx*)

Memória para controle dos *bits* 4 a 8 dos canais 0 ímpares da estrutura de multiquadro CRC-4 (seção 1.3 do capítulo IV). O seu funcionamento está descrito a seguir.

- Decodificação dos endereços das memórias S_n de recepção, com posterior geração de sinais de habilitação a partir do pino de entrada *Chip Enable* (*CEQ*).
- Máquina de Estado para Controle do Modo de Operação:
  - *Reset*: Máquina zerada quando do *reset* geral assíncrono (*RESQ*).

- Estado A: se estiver em sincronismo de multiquadro (*cal*='0'), verifica *ctrl.rsn*. Caso seja igual a '1' (modo congelado) vai para o estado C, caso seja igual a '0' (modo livre) vai para o estado B.
- Estado B: modo livre, prenchendo o banco de memória 1. Verifica *ctrl.rsn* para mudar de modo de operação. Caso este *bit* não esteja ativo, ao final do multiquadro passa para estado G; caso contrário passa para o estado C. Na parte combinacional, gera habilitação e *reset* para o *bit* do registro de interrupção *intl.snv* nos intervalos de tempo pertinentes.
- Estado C: modo congelado, aguardando *bit intl.snv*='0' e início de multiquadro. Verifica *ctrl.rsn* para mudar de modo de operação. Caso este *bit* não esteja ativo, aguarda que *intl.snv* seja igual a '0' e que chegue o intervalo de tempo *bit* 1, canal 0 e quadro 0 do contador de busca para mudar para o estado D; caso contrário permanece no estado C.
- Estado D: modo congelado, armazena os *bits* S_n no banco de memória 1. Verifica *ctrl.rsn* para mudar de modo de operação. Caso este *bit* esteja ativo, aguarda o intervalo de tempo *bit* 8, canal 31 e quadro 15 de linha e passa para o estado C; caso contrário passa para o estado B. Na parte combinacional, gera habilitação para o *bit intl.snv.*
- Estado G: modo livre, prenchendo o banco de memória 2. Verifica *ctrl.rsn* para mudar de modo de operação. Caso este *bit* não esteja ativo, ao final do multiquadro passa para o estado B; caso contrário passa para o estado C. Na parte combinacional, gera habilitação e *reset* para o *bit* do registro de interrupção *intl.snv.*

- Controle de Escrita e Leitura: *flip-flop* para gerar habilitação para o deslocamento dos *bits* S_n. Zerado pelo sinal de *reset* geral assíncrono (*RESQ*). Caso contrário, a cada pulso de relógio (se a habilitação estiver ativa), se estiver em sincronismo de multiquadro (*cal*='0'), se *ctrl.rsn* for igual a '0', o sinal de habilitação para escrita pelo *TB47* (sinal *en_mux_esc_hw*) é ativado nos intervalos de *bits* S_n; caso contrário, se *ctrl.rsn* for igual a '1' e a máquina anterior estiver no estado D, *en_mux_esc_hw* é ativado nos intervalos de *bits* S_n.
- *Shift* dos Registradores: se *RESQ* for igual a '0', os registradores são zerados, caso contrário, se *en_mux_esc_hw* for igual a '1', o dado unipolar da recepção é deslocado no banco 1 ou no banco 2, de acordo com o sinal *pbc1nbc2* que se igual a '0' habilita o banco 1 e se igual a '1' habilita o banco 2.
- Leitura pelo Processador das Memórias S_n de Recepção: libera os dados de saída dos bancos no barramento interno de dados.

F. Outras Funções

Os circuitos da figura 36 são responsáveis pelo controle do sinal de entrada da memória elástica de recepção e dos pinos de saída *ERUN* e *RSIGM*.

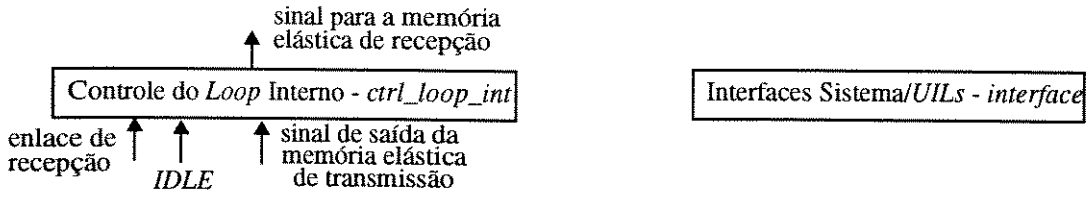


Figura 36: Outras Funções.

F.1. Controle do *Loop* Interno (*ctrl_loop_int*)

Circuito para controle da função do sinal de entrada para a memória elástica de recepção, cujo funcionamento está descrito a seguir.

- O sinal de entrada para a memória elástica de recepção é controlado pelos *bits* de saída da memória de controle *reg_ctrl(5)* e *reg_ctrl(4)*. Se *reg_ctrl(5)* for igual a '0' (funcionamento normal), passa o enlace recebido que é saída do circuito de interface de entrada (*dado_rec_unipolar*); caso contrário (*loop back* interno), se *reg_ctrl(4)* for igual a '1', passa o sinal de saída da memória elástica de transmissão (sinal *dados_sai_tr1*); e se *reg_ctrl(4)* for igual a '0', passa o conteúdo do registro *IDLE*.

F.2. Interfaces Sistema/*UILs* (*interface*)

Circuito para geração dos pinos de saída *ERUN* e *RSIGM*, cujo funcionamento está descrito a seguir.

- *ERUN*: é o pino pelo qual se transfere para o sistema o enlace de onde será retirado o canal 16. Se *ctr3.lbcs* for igual a '0', é transferido o próprio enlace vindo da recepção. Se *ctr3.lbcs* for igual a '1', deve ser feito o *loop back* do canal 16 vindo do sistema. Devido à defasagem de 1 *bit* entre este enlace e o da recepção, é preciso armazenar em um *flip-flop* o último *bit* do canal 16 de sistema para ser enviado como sendo o primeiro do próximo intervalo de tempo de canal 16.
- *RSIGM*: é o pino pelo qual o *TB47* envia para o sistema o sincronismo para identificação do canal 16.

3.2.3 Descrição *VHDL*

Esta etapa consiste em, de acordo com a partição funcional acima especificada, gerar um código *VHDL* sintetizável para cada função isolada. Esta etapa é cíclica com a partição funcional. Baseado em experiências anteriores e em [40] este circuito foi escrito diretamente em um nível sintetizável.

O *VHDL* é uma linguagem complicada que não foi originalmente projetada para suportar aplicações de síntese [24, 41]. Como resultado, existem algumas descrições que podem ser escritas em *VHDL*, mas não são sintetizáveis pelo *Autologic* ou outra ferramenta de síntese. Estas restrições estão documentadas em [8].

A técnica mais eficiente de escrever códigos *VHDL* requer conhecimento do projeto, experiência em *VHDL* e entendimento de como as ferramentas de síntese funcionam. Em geral, deve-se manter o seguinte princípio em mente: *sempre pense em termos do hardware*

esperado. Embora o *VHDL* exija habilidade de programação, ele é uma *Linguagem de Descrição de Hardware*. Você deve sempre ter em mente qual o *hardware* esperado quando estiver escrevendo as descrições *VHDL RTL*. Sempre que possível, incorpore esta informação no seu modelo. Quanto mais informação for fornecida, melhores os resultados.

Esta etapa implica na utilização de um editor de textos *ASCII (American Standard Code for Information Interchange)* para escrita dos códigos *VHDL*. Em seguida, é feita a compilação destes códigos, utilizando-se o compilador *VHDL System-1076*. Esta sequência é repetida várias vezes, enquanto for necessária a correção de quaisquer possíveis erros.

O compilador *VHDL System-1076* é único tanto para a síntese quanto para a simulação, garantindo a conformidade da linguagem, ou seja, o que você simula é o que você obtém através da síntese. Você usa este mesmo compilador *VHDL* para avaliação das descrições *VHDL* e para ver se são sintetizáveis [10].

O documento [42] apresenta a descrição *VHDL* dos blocos desenvolvidos pelo autor e descritos neste trabalho, que são:

- Máquina para Alinhamento: *prim_pal_q_mq_rx;*
- Máquina Verifica Alinhamento de Quadro: *verifica_pal_quadro_rx;*
- Registro para Geração da Palavra de Alinhamento de Multiquadro: *gera_bit_sinc_mq_rx;*
- Contador de 8ms: *cont_8ms_rx;*
- Verifica Palavra de Alinhamento de Quadro: *shift_comp_recepcao_rx;*
- Contador de Busca para Sincronismo: *cont_busca_rx;*
- Verifica CRC na Recepção: *verifica_CRC_rx;*
- Circuitos de Controle do CRC na Recepção: *ctrl_CRC_rx;*
- Contador de 1s: *cont_1s;*
- Contador de Linha: *cont_linha;*
- Monta Canal 0: *monta_canal0_tx;*
- Multiplexador de Voz, *Idle* e Canal 0: *mux_voz_idle_c0_tx;*
- Trata CRC da Transmissão: *trata_CRC_tx;*
- Multiplexador de Canal 16, Reconfiguração e *AIS*: *mux_c16_reconf_ais_tx;*
- Registro de Controle: *reg_controle;*
- Controle das Memórias S_n na Recepção: *ctrl_mem_snrx;*
- Controle do *Loop* Interno: *ctrl_loop_int;* e,
- Interfaces Sistema/*UILs*: *interface.*

3.2.4 Simulação Funcional de cada Bloco Isolado

O objetivo da simulação é validar uma descrição pela confirmação de que a mesma produz as saídas apropriadas para um dado conjunto de entradas [43, 44].

Com a partição funcional fechada é necessário a validação de cada função através das correspondentes descrições *VHDL*. Portanto, esta etapa é cíclica com as descrições *VHDL*. No final desta etapa tem-se todas as funções isoladas validadas.

Para a verificação de um projeto, deve-se montar um sistema como o mostrado na figura 37 [24].

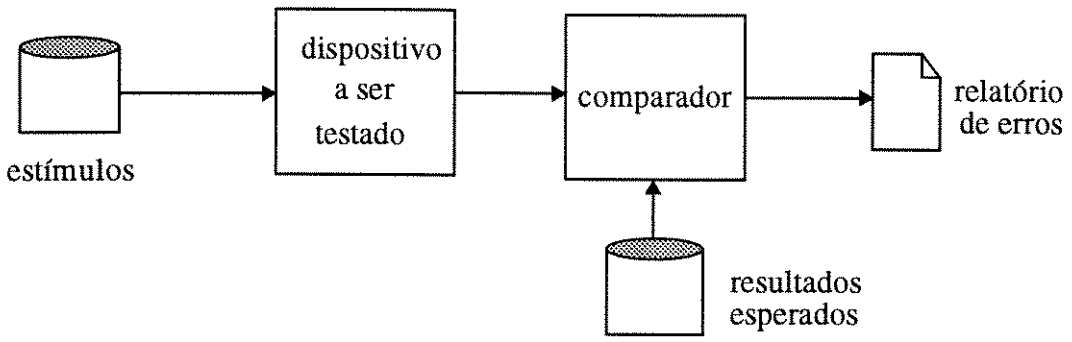


Figura 37: Sistema para Verificação do Projeto.

Os pontos essenciais na implementação deste sistema são:

1. Geração de estímulos para o circuito; e
2. Comparação dos resultados esperados com os resultados simulados.

Os estímulos foram criados utilizando-se um *Test Bench VHDL* [2], que é uma descrição VHDL do sistema onde o circuito estará inserido. O conceito de *Test Bench* implica na criação de modelos *VHDL* adicionais para fornecer os estímulos para o módulo a ser testado. O uso do *VHDL* como uma linguagem de estímulo permite a geração de programas de estímulos bem complexos pelo modelamento de dispositivos externos fornecendo entrada para o circuito. Além disto, um módulo comparador adicional pode ser criado para avaliar se os resultados estão corretos.

A função de comparação dos resultados de simulação com as saídas do circuito, também foi incluída no *VHDL Test Bench*. Em adição ao uso do *VHDL* como uma linguagem para modelar a funcionalidade do circuito e como uma linguagem de estímulo, pode ser usada para implementar a função de comparador. Um *Test Bench VHDL* completo deve incluir três componentes: o gerador de estímulos, o dispositivo a ser testado e um módulo de comparação. Todos os três componentes devem ser conectados ou por meio de um esquemático ou um modelo *VHDL* estrutural.

Com o uso destes métodos, a comparação deve ser feita em um ciclo básico usando um único ponto de *strobe*. O ponto de *strobe* deve ser selecionado para comparar o dado no mesmo tempo em que os dispositivos externos estejam amostrando os dados. A figura 38 ilustra o ponto de teste adotado para comparação entre o sinal de saída do *TB47* e o sinal esperado para o mesmo, que é igual a um tempo t_{setup} antes do final do período de relógio.

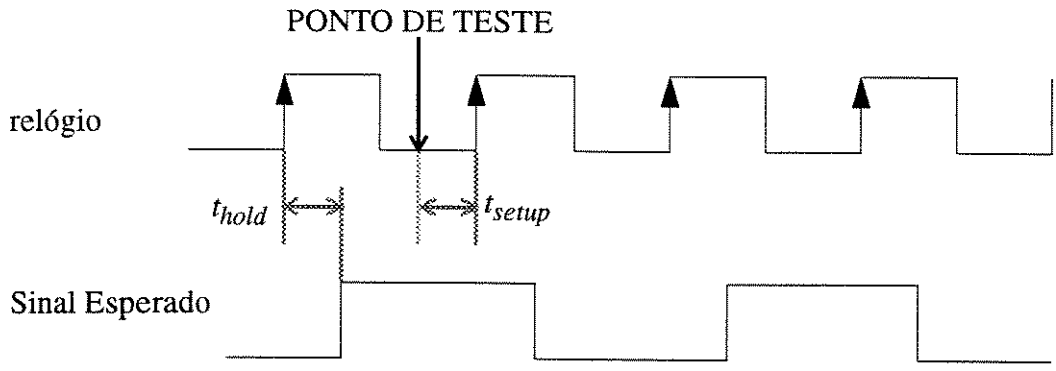


Figura 38: Ponto de Teste dos *Test Benches*.

Esta etapa implica no desenvolvimento de um *Test Bench VHDL* para cada módulo, utilizando-se um editor de textos *ASCII*, seguido de compilação *VHDL* (*System-1076*). Em seguida, é feita uma simulação de cada módulo, para verificação da sua funcionalidade. Para simulação *VHDL*, utilizou-se o simulador *Quicksim II*. Existe, portanto, uma interação não só com os códigos *VHDL* de cada módulo, como também com o código VHDL dos *Test Benches*, até estar correto.

3.2.5 União dos Blocos

Esta etapa simplesmente consiste em interligar todos os blocos funcionais já validados na etapa anterior, através de uma descrição *VHDL* estrutural, utilizando-se um editor de textos *ASCII*. Em seguida, é feita a compilação deste código, utilizando *System-1076*.

3.2.6 Simulação Funcional para Validação de Interfaces

O objetivo desta etapa é validar a interface entre todos os blocos funcionais, já validados anteriormente. Para isto foram feitas várias simulações, com o objetivo também de testar todas as funções do circuito, porém seguindo todo o fluxo do *TB47* e somente observados os resultados através dos pinos de saída. Os estímulos para estas simulações só foram aplicados aos pinos de entrada.

Para isto, utilizou-se o mesmo tipo de sistema ilustrado na figura 37, onde o dispositivo a ser testado é o próprio *TB47*. O que o componente de geração de estímulos e de comparação dos resultados deve conter em cada teste está descrito nos itens a seguir.

Portanto, esta etapa implica no desenvolvimento de *Test Benches VHDL*, utilizando-se um editor de textos *ASCII* seguido de compilação *VHDL* (*System-1076*). Em seguida, são feitas várias simulações para validação da funcionalidade do *TB47*, utilizando-se o *QuickSim II*. Existe uma interação com todos os códigos *VHDL* do *TB47* e dos *Test Benches*, até estar correto.

As simulações executadas para teste dos blocos funcionais descritos neste trabalho, estão descritas a seguir.

A. Simulação 2

Para teste da decodificação de endereços. Esta simulação foi dividida em dois itens:

A.1. Simulação 2_1:

Funções a serem testadas:

- Fazer acessos de escrita em todas as possíveis posições internas do *TB47*, com exceção das memórias S_n (que serão testadas em outra simulação pertinente), para inicialização (habilitado via pino de entrada *Chip Enable CEQ*='0').
- Fazer acessos de leitura para confirmar a escrita anterior (habilitado com *CEQ*='0').
- Fazer um acesso de escrita em determinada posição do *TB47* com o mesmo habilitado (pino de entrada *CEQ*='0').
- Fazer acessos de leitura em todas as possíveis posições internas do *TB47* e verificar que a escrita foi efetuada somente na posição endereçada (pino de entrada *CEQ*='0').
- Os dois últimos itens acima deverão ser executados várias vezes e sequencialmente, de forma a testar todos os possíveis endereços.

Como implementar este teste

- Inicializar o circuito com um pulso negativo no pino de entrada *RESQ*.
- Forçar os pinos de entrada: *POE1*='1', *POE2*='0', *POE3*='1', *TP1*='0' e *TP2*='1'.
- Forçar relógios em *RRCLK_2M*, *RRCLK_4M* e *SCLK_4M*.
- Fazer 39 acessos de escrita (*WRQ*='0') com o *TB47* habilitado (*CEQ*='0') e o barramento de dados *D(7:0)*="10101010". Os endereços devem ser os relativos a: memória de controle *MC0* a *MC31* e registros de controle *CTR1*, *CTR2*, *CTR3*, *MAS1*, *IDLE*, *LTEL* e *LTEH*.
- Fazer 64 acessos de leitura (*RDQ*='0') em todos os endereços *A(6:0)* variando de "0000000" a "0111111", com o *TB47* habilitado (*CEQ*='0') e observar o barramento de dados *D(7:0)*, conforme definido na tabela 14.

Endereço de Leitura *A(6:0)*	Mnemônico	Barramento de Dados *D(7:0)*
0000000 a 0011111	*MC0* a *MC31*	10101010
0100000 a 0100111	*SNR1* a *SNR8*	--------
0101000 a 0101111	*SNT1* a *SNT8*	--------
0110000	*CTR1*	10-01010
0110001	*CTR2*	10101-10
0110010	*CTR3*	10101010
0110011	*MAS1*	10101010
0110100	*LTEL*	10101010
0110101	*LTEH*	------10
0110110	*IDLE*	10101010
0110111	*DFI*	--------
0111000	*int1.snv*	--------
0111001	*CECL*	--------
0111010	*INT1 c/ reset*	--------

Tabela 14: Resultado Esperado para a Simulação 2_1.

Endereço de Leitura *A(6:0)*	Mnemônico	Barramento de Dados *D(7:0)*
0111011	*CECH*	--------
0111100	*INT2* c/ *reset*	--------
0111101	*INT2* s/ *reset*	--------
0111110	*INT1* s/ *reset*	--------
0111111	*PROG*	--010101

Tabela 14: Resultado Esperado para a Simulação 2_1.

- Fazer um acesso de escrita (*WRQ*='0') no endereço *A(6:0)*="0000000", com o *TB47* habilitado (*CEQ*='0') e o barramento de dados *D(7:0)*="01010101".
- Fazer 64 acessos de leitura (*RDQ*='0') em todos os endereços, *A(6:0)* variando de "0000000" a "0111111", com o *TB47* habilitado (*CEQ*='0') e observar o barramento de dados *D(7:0)*, conforme definido na tabela 15.

Endereço de Leitura *A(6:0)*	Mnem.	Barramento de Dados *D(7:0)*							
		escrita no endereço 0000000	escrita no endereço 0000001	escrita no endereço 0000010	escrita no endereço 0000011	escrita no endereço 0000100	escrita no endereço 0000101	..	escrita no endereço 0111111
0000000	*MC0*	01010101	01010101	01010101	01010101	01010101	01010101		01010101
0000001	*MC1*	tabela 14	01010101	01010101	01010101	01010101	01010101		01010101
0000010	*MC2*	tabela 14	tabela 14	01010101	01010101	01010101	01010101		01010101
0000011	*MC3*	tabela 14	tabela 14	tabela 14	01010101	01010101	01010101		01010101
0000100	*MC4*	tabela 14	tabela 14	tabela 14	tabela 14	01010101	01010101		01010101
0000101	*MC5*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	01010101		01010101
0000110	*MC6*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		01010101
0000111	*MC7*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		01010101
0001000	*MC8*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		01010101
0001001	*MC9*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		01010101
0001010	*MC10*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		01010101
0001011	*MC11*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		01010101
0001100	*MC12*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		01010101
0001101	*MC13*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		01010101
0001110	*MC14*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		01010101
0001111	*MC15*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		01010101
0010000	*MC16*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		01010101
0010001	*MC17*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		01010101
0010010	*MC18*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		01010101
0010011	*MC19*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		01010101
0010100	*MC20*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		01010101
0010101	*MC21*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		01010101
0010110	*MC22*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		01010101
0010111	*MC23*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		01010101
0011000	*MC24*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		01010101

Tabela 15: Resultado Esperado para a Simulação 2_1.

Endereço de Leitura *A(6:0)*	Mnem.	Barramento de Dados *D(7:0)*							
		escrita no endereço 0000000	escrita no endereço 0000001	escrita no endereço 0000010	escrita no endereço 0000011	escrita no endereço 0000100	escrita no endereço 0000101	...	escrita no endereço 0111111
0011001	*MC25*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		01010101
0011010	*MC26*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		01010101
0011011	*MC27*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		01010101
0011100	*MC28*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		01010101
0011101	*MC29*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		01010101
0011110	*MC30*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		01010101
0011111	*MC31*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		01010101
0100000	*SNR1*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		tabela 14
0100001	*SNR2*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		tabela 14
0100010	*SNR3*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		tabela 14
0100011	*SNR4*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		tabela 14
0100100	*SNR5*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		tabela 14
0100101	*SNR6*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		tabela 14
0100110	*SNR7*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		tabela 14
0100111	*SNR8*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		tabela 14
0101000	*SNT1*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		tabela 14
0101001	*SNT2*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		tabela 14
0101010	*SNT3*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		tabela 14
0101011	*SNT4*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		tabela 14
0101100	*SNT5*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		tabela 14
0101101	*SNT6*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		tabela 14
0101110	*SNT7*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		tabela 14
0101111	*SNT8*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		tabela 14
0110000	*CTR1*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		01-10101
0110001	*CTR2*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		01010-01
0110010	*CTR3*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		01010101
0110011	*MAS1*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		01010101
0110100	*LTEL*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		01010101
0110101	*LTEH*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		01010101
0110110	*IDLE*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		------01
0110111	*DFI*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		tabela 14
0111000	*SNV*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		tabela 14
0111001	*CECL*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		tabela 14
0111010	*INT1*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		tabela 14
0111011	*CECH*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		tabela 14
0111100	*INT2*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		tabela 14
0111101	*INT2*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		tabela 14
0111110	*INT1*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		tabela 14
0111111	*PROG*	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14	tabela 14		tabela 14

Tabela 15: Resultado Esperado para a Simulação 2_1.

- Os dois itens acima devem ser repetidos 39 vezes, de forma que seja feito um acesso de escrita em toda possível posição interna do *TB47*, seguida de leituras em todas as possíveis posições internas do *TB47*, para se ter certeza de que a escrita só foi feita na posição endereçada.

A.2. Simulação 2_2:

Funções a serem testadas:

- Fazer acessos de escrita em todas as possíveis posições internas do *TB47*, com exceção das memórias S_n, para inicialização (habilitado com *CEQ*='0').
- Fazer acessos de leitura para confirmar a escrita anterior (habilitado com *CEQ*='0').
- Fazer um acesso de escrita em determinada posição do *TB47* com o mesmo desabilitado (pino de entrada *CEQ*='1').
- Fazer acessos de leitura em todas as possíveis posições internas do *TB47* (pino de entrada *CEQ*='0') e constatar a não alteração de valores internos.
- Os dois itens acima devem ser executados várias vezes e sequencialmente, de forma a testar todos os possíveis endereços.
- Fazer acessos de leitura pelo processador com o pino de entrada *CEQ*='1' (*TB47* desabilitado) e constatar que o barramento de dados deve estar sempre em Z.

Como implementar este teste

- Inicializar o circuito com um pulso negativo no pino de entrada *RESQ*.
- Forçar os pinos de entrada: *POE1*='1', *POE2*='0', *POE3*='1', *TP1*='0' e *TP2*='1'.
- Forçar relógios em *RRCLK_2M*, *RRCLK_4M* e *SCLK_4M*.
- Fazer 39 acessos de escrita (*WRQ*='0') com o *TB47* habilitado (*CEQ*='0') e o barramento de dados *D(7:0)*="01010101". Os endereços devem ser os relativos a: memória de controle *MC0* a *MC31* e registros de controle *CTR1*, *CTR2*, *CTR3*, *MAS1*, *IDLE*, *LTEL* e *LTEH*.
- Fazer 64 acessos de leitura (*RDQ*='0') em todos os endereços *A(6:0)* variando de "0000000" a "0111111", com o *TB47* habilitado (*CEQ*='0') e observar o barramento de dados *D(7:0)*, conforme definido na última coluna da tabela 15.
- Fazer um acesso de escrita (*WRQ*='0') no endereço *A(6:0)*="0000000" com o *TB47* desabilitado (*CEQ*='1') e o barramento de dados *D(7:0)*="10101010".
- Fazer 64 acessos de leitura (*RDQ*='0') em todos os endereços, *A(6:0)* variando de "0000000" a "0111111", com o *TB47* habilitado (*CEQ*='0') e observar o barramento de dados *D(7:0)*, conforme definido na última coluna da tabela 15.
- Os dois itens acima devem ser repetidos 39 vezes, para se ter certeza de que acessos de escrita com o *TB47* desabilitado não alteram o valor de nenhuma posição interna do mesmo.
- Fazer 64 acessos de leitura com o *TB47* desabilitado e ver que o barramento de dados deve estar sempre em Z.

B. Simulação 3

Funções a serem testadas:

- Teste dos registros de controle
  - Fazer acessos de escrita pelo processador em cada registro, variando todos os *bits* do registro, seguido de acessos de leitura pelo processador para verificar se foi escrito certo.
  - Como existem dois *bits* de controle que também podem ser alterados pelo *hardware* (*ctr5.frs* e *ctr4.txsn*), esta possível alteração do *hardware* será testada nas simulações que tratarem esta função. A diferença do que foi feito aqui, é que deverão ser feitos dois acessos de leitura pelo processador, para constatar a alteração feita pelo *hardware*.

Como implementar este teste

- Inicializar o circuito com um pulso negativo no pino de entrada *RESQ*.
- Fazer acessos de escrita e leitura, em cada registro, de acordo com a sequência ("00000000" / "00000001" / "00000010" / "00000100" / "00001000" / "00010000" / "00100000" / "01000000" / "10000000" / "00000000"), de modo a exercitar todos os *bits* dos registros de controle.

C. Simulação 13

Funções a serem testadas:

Testar as funções de sincronismo de quadro, sincronismo de multiquadro, monitoração da taxa excessiva de erro, alinhamento de quadro incorreto, verificação da palavra de quadro e alarme remoto recebido, com a monitoração dos *bits* de interrupção pertinentes (*int1.ste*, *int1.rra*, *int1.los* e *int2.cal*). Testar, também, a geração dos pinos de saída *RSGIM*, *ERUN* e do sinal de *WAIT* para o processador, gerado pelo contador de erros de CRC.

Obs.:

O pedido de interrupção (bit de interrupção int1.inte e pino AINT) deve ser observado antes e depois da retirada da máscara nos bits do registro de controle MAS1.

Sempre que algum bit do registro de interrupção for observado, fazer primeiro uma leitura sem reset e constatar que os bits atuais e permanentes não são alterados pelo processador e o pedido de interrupção (se houver) não foi retirado. Em seguida, fazer uma leitura com reset e observar que os bits atuais não são alterados pelo processador, os bits permanentes são zerados no final da leitura e o pedido de interrupção é retirado.

Esta simulação foi dividida em quatro itens:

C.1. Simulação 13_1

Funções a serem testadas:

- Trabalhar com o dado de entrada unipolar não embaralhado e não invertido (*POE2*='0', *POE1*='1' e *ctr1.rdis*='0'), programar modo teste (*NTESTE1*='0' e *NTESTE2*='0') e programar *LTEL*="00000111" e *LTEH*="------00".
- Forçar a entrada em sincronismo de quadro (com todas as possíveis condições) e em seguida a perda do sincronismo de quadro (três palavras de quadro consecutivas erradas), observando

os *bits* de interrupção *int1.los* (testar tanto a borda positiva quanto a negativa, de acordo com o *bit* de controle *ctr2.cbi*) e *int1.inte* e o pino de interrupção *AINT*.

- Após a recuperação do sincronismo de quadro, testar a não entrada em sincronismo de multiquadro (2 palavras de sincronismo de multiquadro corretas em quatro - 8ms). Isto força a perda de sincronismo de quadro. Observar os *bits* de interrupção *int1.los* e *int1.inte*, o *bit* de estado *int2.cal* e o pino de interrupção *AINT*.
- Ainda neste caso, fazer acessos de leitura nos registros *CECL* e *CECH* (contador de erros de CRC) que não podem ser alterados, pois o cálculo de CRC e, consequentemente, a contagem de erros de CRC devem estar desabilitados enquanto o circuito não estiver em sincronismo de multiquadro.
- Forçar a entrada em sincronismo de quadro e, em seguida, de multiquadro. Observar os *bits* de interrupção *int1.los* e *int1.inte*, o *bit* de estado *int2.cal* e o pino de interrupção *AINT*.
- Forçar também a busca de sincronismo de quadro pelo *bits* de controle *ctr2.r915*='1' e *ctr5.frs*='1'. Observar os *bits* de interrupção *int1.los* e *int1.inte* e o pino de interrupção *AINT*.
- Fazer um acesso de leitura no *bit* de controle *ctr5.frs* e constatar que ele será feito igual a '0' pelo *hardware*, assim que o mesmo entrar em busca de sincronismo de quadro.
- Sempre observar o pino *de saída RSIGM*.

Como implementar este teste

- Inicializar o circuito com um pulso de *reset* em *RESQ*.
- Fazer *POE2*='0', *POE1*='1', *ctr1.rdis*='0' (valor inicial), *NTESTE1*='0' e *NTESTE2*='0'.
- Programar *LTEL*="00000111", com uma escrita no endereço "0110100" e *LTEH*="------00", com uma escrita no endereço "0110101".
- Tirar a máscara do *bit* de interrupção *int1.los*, escrevendo "00010000" no endereço "0110011".
- Procura de sincronismo de quadro com todas as possíveis condições: achar primeira palavra de quadro correta, *bit2*='0' na próxima palavra, achar primeira palavra de quadro correta, *bit2*='1' na próxima palavra e próxima palavra diferente da palavra de quadro.
- Fazer *ctr2.cbi*='1', para que gere interrupção na borda de descida de *int1.los*, que significa a recuperação do sincronismo de quadro.
- Recuperação do sincronismo de quadro: primeira palavra de quadro correta, *bit2*='1' na próxima palavra e segunda palavra de quadro correta.
- Fazer uma leitura sem *reset* no registro de interrupção *INT1* e constatar que *int1.los*='0' e *int1.inte*='1'. O pino *AINT*='0'.
- Fazer uma leitura com *reset* no registro de interrupção *INT1* e constatar que *int1.los*='0' e *int1.inte*='1'. Logo após esta leitura, o pino *AINT*='0'.
- Fazer uma leitura sem *reset* no registro de interrupção *INT1* e constatar que *int1.los*='0' e *int1.inte*='0'. O pino *AINT*=Z.
- Fazer *ctr2.cbi*='0', para que gere interrupção na borda de subida de *int1.los*, que signica a perda do sincronismo de quadro.

- Forçar perda de sincronismo de quadro, com três palavras de quadro consecutivas erradas.
- Fazer uma leitura sem *reset* no registro de interrupção *INT1* e constatar que *int1.los*='1' e *int1.inte*='1'. O pino *AINT*='0'.
- Fazer uma leitura com *reset* no registro de interrupção *INT1* e constatar que *int1.los*='1' e *int1.inte*='1'. Logo após esta leitura, o pino *AINT*='0'.
- Fazer uma leitura sem *reset* no registro de interrupção *INT1* e constatar que *int1.los*='1' e *int1.inte*='0'. O pino *AINT*=Z.
- Recuperar sincronismo de quadro: primeira palavra de quadro correta, *bit2*='1' na próxima palavra e segunda palavra de quadro correta.
- Fazer uma leitura sem *reset* no registro de interrupção *INT1* e constatar que *int1.los*='0' e *int1.inte*='0'. O pino *AINT*=Z.
- Fazer perder sincronismo de quadro devido a palavra de multiquadro: fazer estourar 8ms sem ter achado 2 palavras de sincronismo de multiquadro corretas, porém com as palavras de quadro corretas o que ocasiona a perda de sincronismo de quadro.
- Fazer uma leitura sem *reset* no registro de interrupção *INT1* e constatar que *int1.los*='1' e *int1.inte*='1'. O pino *AINT*='0'.
- Fazer uma leitura com *reset* no registro de interrupção *INT1* e constatar que *int1.los*='1' e *int1.inte*='1'. Logo após esta leitura, o pino *AINT*='0'.
- Fazer uma leitura sem *reset* no registro de interrupção *INT1* e constatar que *int1.los*='1' e *int1.inte*='0'. O pino *AINT*=Z.
- Fazer uma leitura sem *reset* no registro de interrupção *INT2* e constatar que *int2.cal*='1'.
- Fazer uma leitura nos registros de estado *CECL* e *CECH*, que devem estar em '0', pois o cálculo de CRC e, consequentemente a contagem de erros de CRC devem estar desabilitados enquanto o circuito nao estiver em sincronismo de multiquadro.
- Entrar em sincronismo de quadro: primeira palavra de quadro correta, *bit2*='1' na próxima palavra e segunda palavra de quadro correta.
- Entrar em sincronismo de multiquadro, com duas palavras de multiquadro consecutivas corretas.
- Fazer uma leitura sem *reset* no registro de interrupção *INT1* e constatar que *int1.los*='0' e *int1.inte*='0'. O pino *AINT*=Z.
- Fazer uma leitura sem *reset* no registro de interrupção *INT2* e constatar que *int2.cal*='0'.
- Fazer *ctr2.r915*='1' e *ctr5.frs*='1'.
- Fazer uma leitura no registro de controle *CTR2* e constatar que *ctr2.r915*='1'.
- Fazer uma leitura sem *reset* no registro de interrupção *INT1* e constatar que *int1.los*='1' e *int1.inte*='1'. O pino *AINT*='0'.
- Fazer uma leitura no registro de controle *CTR5* e constatar que *ctr5.frs*='0'.
- Fazer uma leitura com *reset* no registro de interrupção *INT1* e constatar que *int1.los*='1' e *int1.inte*='1'. Logo após esta leitura, o pino *AINT*='0'.

- Fazer uma leitura sem *reset* no registro de interrupção *INT1* e constatar que *int1.los*='1' e *int1.inte*='0'. O pino *AINT*=Z.
- Fazer *ctr2.cbi*='1', o que força interrupção por borda de descida.
- Fazer *mas1.los*='0'.
- Fazer o circuito recuperar o sincronismo de quadro: primeira palavra de quadro correta, *bit*2='1' na próxima palavra e segunda palavra de quadro correta.
- Fazer uma leitura sem *reset* no registro de interrupção *INT1* e constatar que *int1.los*='0' e *int1.inte*='0'. O pino *AINT*=Z (a interrupção não é gerada quando a máscara estiver ativa).
- Sempre observar *RSIGM*.

C.2. Simulação 13_2

Funções a serem testadas:

- Trabalhar com o dado de entrada unipolar não embaralhado e não invertido (*POE2*='0', *POE1*='1' e *ctr1.rdis*='0'), modo não teste (*NTESTE1*='1' e *NTESTE2*='1') e *ROID*='0' (fora de sincronismo de quadro).
- Fazer *CSTR*='1' no intervalo de tempo de canal 16 atrasado de 1 *bit* e igual a '0' nos demais intervalos de tempo durante dois quadros. E, sempre igual a '0' durante mais 2 quadros.
- Fazer *ctr3.lbcs*='0' durante dois quadros e igual a '1' durante mais dois quadros.
- Observar o pino de saída *ERUN*, que deve ser sempre '0' (ou seja, igual a *ROID*) nos dois primeiros quadros e, nos dois quadros seguintes igual a '1' em todo intervalo de tempo de canal 16 e igual a '0' nos demais intervalos de tempo.
- Neste caso, fazer *ctr2.fepr*='1' e *ctr2.fept*='1', para que sejam forçados erros de paridade tanto na memória elástica de recepção quanto na de transmissão e constatar que *int2.epm* será sempre igual a '0', pois enquanto o circuito não estiver em sincronismo de quadro, deve ser inibida a ativação deste *bit* de interrupção.

Como implementar este teste

- Inicializar o circuito com um pulso de *reset* em *RESQ*.
- Fazer *POE2*='0', *POE1*='1', *ctr1.rdis*='0' (valor inicial), *NTESTE1*='1', *NTESTE2*='1', *ROID*='0' (fora de sincronismo de quadro), *ctr2.fepr*='1' e *ctr2.fept*='1'.
- Fazer *CSTR*='1' no intervalo de tempo de canal 16 atrasado de 1 *bit* e igual a '0' nos demais intervalos de tempo, durante dois quadros. E, sempre igual a '0' durante os 2 quadros seguintes.
- Fazer *ctr3.lbcs*='0' durante dois quadros e igual a '1' durante os dois quadros seguintes.
- Observar o pino de saída *ERUN*, que deve ser sempre '0' nos dois primeiros quadros (*ctr3.lbcs*='0', portanto igual a *ROID*). E, nos dois quadros seguintes deve ser '1' em todo intervalo de tempo de canal 16 e '0' nos demais intervalos de tempo.
- Fazer várias leituras sem *reset* no registro de interrupção *INT2* e constatar que *int2.epm*='0'.
- Mascarar todos os *bits* do registro de interrupção, com exceção de *int2.epm* e observar o pino

de saída *AINT*, que não pode ser ativado ('0').

- A memória de controle deve ser inicializada com "0000000" em todos os canais.

C.3. Simulação 13_3

Funções a serem testadas:

- Trabalhar com o dado de entrada unipolar não embaralhado e não invertido (*POE2*='0', *POE1*='1' e *ctrl.rdis*='0'), programar modo teste (*NTESTE1*='0' e *NTESTE2*='0') e programar *LTEL*="00000111" e *LTEH*="------00".
- Forçar a entrada em sincronismo de quadro e, em seguida, de multiquadro.
- Forçar erros de CRC, de tal modo a se atingir a taxa excessiva de erro e o alinhamento de quadro incorreto (observar os registros de estado *CECL* e *CECH*). Observar os *bits* de interrupção *intl.los*, *intl.ste* e *intl.inte*, o *bit* de estado *int2.cal* e o pino de interrupção *AINT*. Neste item, fazer um acesso de leitura nos registros de estado *CECL* e *CECH*, para que seja gerado um sinal de *WAIT* para o processador (o processador não pode ler enquanto o contador estiver sendo atualizado) e observar o pino *WAIT*.

Como implementar este teste

- Inicializar o circuito com um pulso de *reset* em *RESQ*.
- Fazer *POE2*='0', *POE1*='1', *ctrl.rdis*='0' (valor inicial), *NTESTE1*='0' e *NTESTE2*='0'.
- Programar *LTEL*="00000111", com uma escrita no endereço "0110100" e *LTEH*="------00", com uma escrita no endereço "0110101".
- Tirar a máscara do *bit* de interrupção *intl.los*, escrevendo "---1----" no endereço "0110011".
- Tirar a máscara do *bit* de interrupção *intl.ste*, escrevendo "1-------" no endereço "0110011".
- Entrar em sincronismo de quadro: primeira palavra de quadro correta, *bit*2='1' na próxima palavra e segunda palavra de quadro correta.
- Entrar em sincronismo de multiquadro, com duas palavras de multiquadro consecutivas corretas.
- Montar 2 submultiquadros com CRC correto e 8 submultiquadros com CRC errado.
- Fazer acesso de leitura em *CECL* e *CECH*, enquanto o mesmo estiver sendo alterado pelo *hardware*, para que seja gerado o sinal de *WAIT* para o processador.
- Mascarar o *bit* de interrupção *intl.ste*.
- Fazer uma leitura sem *reset* no registro de interrupção *INTI* e constatar que *intl.ste*='1' e *intl.inte*='1'. O pino *AINT*='0'.
- Fazer uma leitura com *reset* no registro de interrupção *INTI* e constatar que *intl.ste*='1' e *intl.inte*='1'. Logo após esta leitura, o pino *AINT*='0'.
- Fazer uma leitura sem *reset* no registro de interrupção *INTI* e constatar que *intl.ste*='0' e *intl.inte*='0'. O pino *AINT*=Z.
- Montar 11 submultiquadros com CRC errado.

- Fazer uma leitura sem *reset* no registro de interrupção *INT1* e constatar que *int1.los*='1' e *int1.inte*='1'. O pino *AINT*='0'.
- Fazer uma leitura com *reset* no registro de interrupção *INT1* e constatar que *int1.los*='1' e *int1.inte*='1'. Logo após esta leitura, o pino *AINT*='0'.
- Fazer uma leitura sem *reset* no registro de interrupção *INT1* e constatar que *int1.los*='1' e *int1.inte*='0'. O pino *AINT*=Z.
- Fazer uma leitura sem *reset* no registro de interrupção *INT2* e constatar que *int2.cal*='1'.

C.4. Simulação 13_4

Funções a serem testadas:

- Trabalhar com o dado de entrada unipolar não embaralhado e não invertido (*POE2*='0', *POE1*='1' e *ctr1.rdis*='0') e programar modo teste (*NTESTE1*='0' e *NTESTE2*='0').
- Inserir também alarme remoto recebido e observar os *bits* de interrupção *int1.rra* (testar tanto a borda positiva quanto a negativa, de acordo com o *bit* de controle *ctr2.cbi*) e *int1.inte,* e o pino de interrupção *AINT.*
- Fazer dois acessos, seguidos de leitura pelo processador nos registros de estado *CECL* e *CECH* e constatar que, neste modo (*ctr2.r915*='1'), logo após um acesso de leitura estes registros são zerados.

Como implementar este teste

- Inicializar o circuito com um pulso de *reset* em *RESQ.*
- Fazer *POE2*='0', *POE1*='1', *ctr1.rdis*='0' (valor inicial), *NTESTE1*='0', *NTESTE2*='0', *ctr2.r915*='1' e *ctr2.cbi*='0' (valor inicial).
- Tirar a máscara do *bit* de interrupção *int1.rra*, através de escrita de "00100000" no endereço "0110011".
- Entrar em sincronismo de quadro: primeira palavra de quadro correta, *bit2*='1' na próxima palavra e segunda palavra de quadro correta.
- Entrar em sincronismo de multiquadro, com duas palavras de multiquadro consecutivas corretas.
- Montar 3 multiquadros com CRC errado, fazendo no primeiro multiquadro o *bit A (*Indicação de Alarme Remoto*)* igual a:
  - nos quadros 1 e 3, *A*='0'
  - nos quadros 5 e 7, *A*='1', com os seguintes procedimentos:
    - Fazer uma leitura sem *reset* no registro de interrupção *INT1* e constatar que *int1.rra*='1' e *int1.inte*='1'. O pino *AINT*='0'.
    - Fazer uma leitura com *reset* no registro de interrupção *INT1* e constatar que *int1.rra*='1' e *int1.inte*='1'. Logo após esta leitura, o pino *AINT*='0'.
    - Fazer uma leitura sem *reset* no registro de interrupção *INT1* e constatar que *int1.rra*='1' e *int1.inte*='0'. O pino *AINT*=Z.

- Fazer *ctr2.cbi*='1'.

- nos quadros 9 e 11, *A*='0', com os seguintes procedimentos:
  - Fazer uma leitura sem *reset* no registro de interrupção *INT1* e constatar que *int1.rra*='0' e *int1.inte*='1'. O pino *AINT*='0'.
  - Fazer uma leitura com *reset* no registro de interrupção *INT1* e constatar que *int1.rra*='0' e *int1.inte*='1'. Logo após esta leitura, o pino *AINT*='0'.
  - Fazer uma leitura sem *reset* no registro de interrupção *INT1* e constatar que *int1.rra*='0' e *int1.inte*='0'. O pino *AINT*=Z.
- nos quadros 13 e 15, *A*='1', com os seguintes procedimentos:
  - Fazer uma leitura sem *reset* no registro de interrupção *INT1* e constatar que *int1.rra*='1' e *int1.inte*='1'. O pino *AINT*='0'.
  - Fazer uma leitura com *reset* no registro de interrupção *INT1* e constatar que *int1.rra*='1' e *int1.inte*='1'. Logo após esta leitura, o pino *AINT*='0'.
  - Fazer uma leitura sem *reset* no registro de interrupção *INT1* e constatar que *int1.rra*='1' e *int1.inte*='0'. O pino *AINT*=Z.
  - Fazer *mas1.rra*='1' (mascarado).
- Fazer uma leitura no registro de estado *CECL*, que pode estar em qualquer valor.
- Fazer outra leitura no registro de estado *CECL*, que deve estar zerado, pois quando *ctr2.r915*='1', *CECL* é zerado após um pulso de leitura.
- Fazer uma leitura no registro de estado *CECH*, que pode estar em qualquer valor.
- Fazer outra leitura no registro de estado *CECH*, que deve estar zerado, pois quando *ctr2.r915*='1', *CECH* é zerado após um pulso de leitura.

D. Simulação 17

Funções a serem testadas:

Testar os circuitos multiplexadores do enlace de transmissão: montagem do canal 0; inserção do canal 16 vindo do sistema, via pino *CSTR*, no intervalo de tempo de canal 16, programado pelo *bits* de controle *ctr3.c16a* e *ctr3.c16b*; geração do CRC para transmissão; inserção de *AIS*; *bypass* e inserção de erro no enlace de transmissão.

- Montagem do canal 0:
  - Canal 0 e quadro par do contador de linha:
    - *bit* 1: '0'.
    - *bits* 2 a 8: palavra de quadro ("0011011").
  - Canal 0 e quadro ímpar do contador de linha:
    - *bit* 1 dos quadros 1, 3, 5, 7, 9 e 11: palavra de multiquadro ("001011").
    - *bit* 1 dos quadros 13 e 15: se o circuito estiver fora de sincronismo de multiquadro, será inserido '0'; senão, se os *bits E* do enlace recebido estiverem ativos, o *hardware* é

transparente a eles; caso contrário, é inserido o resultado do cálculo local.

- *bit* 2: '1'.
- *bit* 3: se o *bit A* (Indicação de Alarme Remoto) do enlace recebido estiver ativo, o *hardware* é transparente; caso contrário, fica sob controle do processador via *bits* de controle *ctr2.exra* e *ctr2.vxra*.
- *bits* 4 a 8: memórias S_n.

- Inserção do canal 16
  - inserção de *CSTR* ou *IDLE* atrasados de 1 *bit*, sob controle dos *bits ctr3.c16a* e *ctr3.c16b*.
  - definição do intervalo de tempo de canal 16 (canal 16, canais 16 e 17 ou canais 16, 17 e 18), sob controle dos *bits ctr3.c16a* e *ctr3.c16b*.
- Tratador de CRC de transmissão
  - geração de CRC para o enlace de transmissão.
  - inserção de erro de CRC sob controle do *bit ctr3.crci*.
- Inserção de *AIS* no enlace de transmissão, sob controle do *bit ctr1.xais*.
- *Bypass* do enlace de recepção, sob controle do *bit ctr1.byp* e do pino de entrada *SIPA* (esta função tem prioridade sobre qualquer outro *bit* de controle).
- Inserção do enlace de transmissão invertido sob controle do pino de entrada *INSERR*.
- Em caso de perda de alinhamento de quadro inserir os *bits E* iguais a '0' no enlace de transmissão.

Esta simulação foi dividida em quatorze itens:

D.1. Simulação 17_1

Funções a serem testadas:

- Canal 0 (com cálculo do CRC):
  - *ROID*='0' (fora de sincronismo de quadro), portanto os *bits E* devem ser '0'.
  - fazer o *bit A* (Indicação de Alarme Remoto) do enlace recebido não ativo('0') e *ctr2.exra*='0', o que faz o *hardware* transparente para o alarme remoto recebido.
  - fazer *ctr1.esn*='0' (modo transparente para os *bits* S_n), para que passe o enlace recebido nestes *bits*.
  - geração do CRC.
- Canais de Voz:
  - programar a memória de controle com operação normal e todo canal não alocado.

Como implementar este teste

- Inicializar o circuito com um pulso de *reset* em *RESQ*.

- Fazer *INSERR*='0', *SIPA*='0', *ctrl.byp*='0', *ctrl.xais*='0' (valor inicial), *ctr2.lre*='0' (valor inicial), *POE2*='0', *POE1*='0', *ctrl.rdis*='0' (valor inicial), *ctrl.esn*='0' (valor inicial - modo transparente *ROID*), *NTESTE1*='0', *NTESTE2*='0', *ROID*='0' (fora de sincronismo de quadro), *ctr2.exra*='0' (valor inicial), *ctr3.crci*='0' (valor inicial) e *bits MC6* a *MC0* da memória de controle iguais a "0000000" (operação normal com todo canal não alocado).

Com isto, o *bit A* do enlace de transmissão deve ser igual a *ROID*. Portanto, o sinal de saída *XULR* esperado, deve ser o apresentado na tabela 16.

Quadro	Canal 0	Canal 31	Quadro	Canal 0	Canal 31
0	$(C_1)0011011$	00000000	8	$(C_1)0011011$	00000000
1	01000000	00000000	9	11000000	00000000
2	$(C_2)0011011$	00000000	10	$(C_2)0011011$	00000000
3	01000000	00000000	11	11000000	00000000
4	$(C_3)0011011$	00000000	12	$(C_3)0011011$	00000000
5	11000000	00000000	13	01000000	00000000
6	$(C_4)0011011$	00000000	14	$(C_4)0011011$	00000000
7	01000000	00000000	15	01000000	00000000

Tabela 16: Resultado Esperado para a Simulação 17_1.

D.2. Simulação 17_2

Funções a serem testadas:

- Canal 0 (com cálculo do CRC):
  - *ROID*='1' (fora de sincronismo de quadro), portanto os *bits E* devem ser '0'.
  - fazer o *bit A* (Indicação de Alarme Remoto) do enlace recebido ativo ('1') e *ctr2.exra*='0', o que faz o *hardware* transparente para o alarme remoto recebido.
  - fazer *ctrl.esn*='0' (modo transparente para os *bits* S_n), para que passe o enlace recebido nestes *bits*.
  - geração do CRC.
- Canais de Voz:
  - programar a memória de controle com operação normal e todo canal não alocado.

Como implementar este teste

- Inicializar o circuito com um pulso de *reset* em *RESQ*.
- Fazer *INSERR*='0', *SIPA*='0', *ctrl.byp*='0', *ctrl.xais*='0' (valor inicial), *ctr2.lre*='0' (valor inicial), *POE2*='0', *POE1*='0', *ctrl.rdis*='0' (valor inicial), *ctrl.esn*='0' (valor inicial - modo transparente *ROID*), *NTESTE1*='0', *NTESTE2*='0', *ROID*='1' (fora de sincronismo de quadro), *ctr2.exra*='0' (valor inicial), *ctr3.crci*='0' (valor inicial) e *bits MC6* a *MC0* da memória de controle iguais a "0000000" (operação normal com todo canal não alocado).

Com isto, o *bit A* do enlace de transmissão deve ser igual a *ROID*. Portanto, o sinal de saída *XULR* esperado, deve ser o apresentado na tabela 17.

Quadro	Canal 0	Canal 31	Quadro	Canal 0	Canal 31
0	$(C_1)0011011$	11111111	8	$(C_1)0011011$	11111111
1	01111111	11111111	9	11111111	11111111
2	$(C_2)0011011$	11111111	10	$(C_2)0011011$	11111111
3	01111111	11111111	11	11111111	11111111
4	$(C_3)0011011$	11111111	12	$(C_3)0011011$	11111111
5	11111111	11111111	13	01111111	11111111
6	$(C_4)0011011$	11111111	14	$(C_4)0011011$	11111111
7	01111111	11111111	15	01111111	11111111

Tabela 17: Resultado Esperado para a Simulação 17_2.

D.3. Simulação 17_3

Funções a serem testadas:

- Canal 0 (com cálculo do CRC):
  - *ROID*='0' (fora de sincronismo de quadro), portanto os *bits E* devem ser '0'.
  - fazer o *bit A* (Indicação de Alarme Remoto) do enlace recebido não ativo ('0') e *ctr2.exra*='1' e *ctr2.vxra*='0', o que faz com que o *bit A* seja igual ao valor de *ctr2.vxra*.
  - fazer *ctr1.esn*='0' (modo transparente para os *bits* S_n), para que passe o enlace recebido nestes *bits*.
  - geração do CRC.
- Canais de Voz:
  - programar a memória de controle com operação normal e todo canal não alocado.

Como implementar este teste

- Inicializar o circuito com um pulso de *reset* em *RESQ*.
- Fazer *INSERR*='0', *SIPA*='0', *ctr1.byp*='0', *ctr1.xais*='0' (valor inicial), *ctr2.lre*='0' (valor inicial), *POE2*='0', *POE1*='1', *ctr1.rdis*='0' (valor inicial), *ctr1.esn*='0' (valor inicial - modo transparente *ROID*), *NTESTE1*='0', *NTESTE2*='0', *ROID*='0' (fora de sincronismo de quadro), *ctr2.exra*='1', *ctr2.vxra*='0' (valor inicial), *ctr3.crci*='0' (valor inicial) e *bits MC6* a *MC0* da memória de controle iguais a "0000000" (operação normal com todo canal não alocado).

Com isto, o *bit A* do enlace de transmissão deve ser igual ao valor do *bit ctr2.vxra* do registro de controle, que no caso é '0'. Portanto, o sinal de saída *XULR* esperado, deve ser o apresentado na tabela 18.

Quadro	Canal 0	Canal 31	Quadro	Canal 0	Canal 31
0	$(C_1)0011011$	00000000	8	$(C_1)0011011$	00000000
1	01000000	00000000	9	11000000	00000000
2	$(C_2)0011011$	00000000	10	$(C_2)0011011$	00000000

Tabela 18: Resultado Esperado para a Simulação 17_3.

Quadro	Canal 0	Canal 31	Quadro	Canal 0	Canal 31
3	*01000000*	*00000000*	11	*11000000*	*00000000*
4	*(C_3)0011011*	*00000000*	12	*(C_3)0011011*	*00000000*
5	*11000000*	*00000000*	13	*01000000*	*00000000*
6	*(C_4)0011011*	*00000000*	14	*(C_4)0011011*	*00000000*
7	*01000000*	*00000000*	15	*01000000*	*00000000*

Tabela 18: Resultado Esperado para a Simulação 17_3.

D.4. Simulação 17_4

Funções a serem testadas:

- Canal 0 (com cálculo do CRC):
  - *ROID*='0' (fora de sincronismo de quadro), portanto os *bits E* devem ser '0'.
  - fazer o *bit A* (Indicação de Alarme Remoto) do enlace recebido não ativo ('0') e *ctr2.exra*='1' e *ctr2.vxra*='1', o que faz com que o *bit A* seja igual ao valor de *ctr2.vxra*.
  - fazer *ctr1.esn*='0' (modo transparente para os *bits* S_n), para que passe o enlace recebido nestes *bits*.
  - *ctr3.crci*='1', o que inverte o CRC calculado.
- Canais de Voz:
  - programar a memória de controle com operação normal e todo canal não alocado.

Como implementar este teste

- Inicializar o circuito com um pulso de *reset* em *RESQ*.
- Fazer *INSERR*='0', *SIPA*='0', *ctr1.byp*='0', *ctr1.xais*='0' (valor inicial), *ctr2.lre*='0' (valor inicial), *POE2*='0', *POE1*='1', *ctr1.rdis*='0' (valor inicial), *ctr1.esn*='0' (valor inicial - modo transparente *ROID*), *NTESTE1*='0', *NTESTE2*='0', *ROID*='0' (fora de sincronismo de quadro), *ctr2.exra*='1', *ctr2.vxra*='1', *ctr3.crci*='1' e *bits MC6* a *MC0* da memória de controle iguais a "0000000" (operação normal com todo canal não alocado).

Com isto, o *bit A* do enlace de transmissão deve ser igual ao *bit ctr2.vxra* do registro de controle, que no caso é '1'. Portanto, o sinal de saída *XULR* esperado, deve ser o apresentado na tabela 19.

Quadro	Canal 0	Canal 31	Quadro	Canal 0	Canal 31
0	*($NOTC_1$)0011011*	*00000000*	8	*($NOTC_1$)0011011*	*00000000*
1	*01100000*	*00000000*	9	*11100000*	*00000000*
2	*($NOTC_2$)0011011*	*00000000*	10	*($NOTC_2$)0011011*	*00000000*
3	*01100000*	*00000000*	11	*11100000*	*00000000*
4	*($NOTC_3$)0011011*	*00000000*	12	*($NOTC_3$)0011011*	*00000000*
5	*11100000*	*00000000*	13	*01100000*	*00000000*
6	*($NOTC_4$)0011011*	*00000000*	14	*($NOTC_4$)0011011*	*00000000*
7	*01100000*	*00000000*	15	*01100000*	*00000000*

Tabela 19: Resultado Esperado para a Simulação 17_4.

D.5. Simulação 17_5

Funções a serem testadas:

- Canal 0 (com cálculo do CRC):
  - esperar o circuito entrar em sincronismo de multiquadro e os *bits E* iguais a '0' (ativos), portanto o *hardware* é transparente a eles.
  - fazer o *bit A* (Indicação de Alarme Remoto) do enlace recebido ativo ('1'), portanto o *hardware* é transparente a eles.
  - fazer *ctr1.esn*='0' (modo transparente para os *bits* S_n), para que passe o enlace recebido nestes *bits*.
  - geração do CRC.
- Canais de Voz:
  - programar a memória de controle com operação normal e todo canal alocado.

Como implementar este teste

- Inicializar o circuito com um pulso de *reset* em *RESQ*.
- Fazer *INSERR*='0', *SIPA*='0', *ctr1.byp*='0', *ctr1.xais*='0' (valor inicial), *ctr2.lre*='0' (valor inicial), *POE2*='0', *POE1*='1', *ctr1.rdis*='0' (valor inicial), *ctr1.esn*='0' (valor inicial - modo transparente *ROID*), *NTESTE1*='0', *NTESTE2*='0', *ctr2.exra*='0' (valor inicial), *ctr2.vxra*='0' (valor inicial), *ctr3.crci*='0' (valor inicial), *ctr3.lbme*='0' (valor inicial) *XDI*='0' e *bits MC6* a *MC0* da memória de controle iguais a "0001111" (operação normal com todo canal alocado).
- Fazer *ROID* receber: palavra de quadro, *bit* 2 da próxima palavra igual a '1', palavra de quadro, 2 palavras de multiquadro (entra em sincronismo de multiquadro), 2 multiquadros com estrutura de quadro correta, com o *bit A*='1', *bits* S_n iguais a '0', CRC errado e os *bits E* iguais a '0'.

Com isto os *bits E* devem ser transparentes. Portanto, o sinal de saída *XULR* esperado, deve ser o apresentado na tabela 20

Quadro	Canal 0	Canal 31	Quadro	Canal 0	Canal 31
0	$(C_1)0011011$	00000000	8	$(C_1)0011011$	00000000
1	01100000	00000000	9	11100000	00000000
2	$(C_2)0011011$	00000000	10	$(C_2)0011011$	00000000
3	01100000	00000000	11	11100000	00000000
4	$(C_3)0011011$	00000000	12	$(C_3)0011011$	00000000
5	11100000	00000000	13	01100000	00000000
6	$(C_4)0011011$	00000000	14	$(C_4)0011011$	00000000
7	01100000	00000000	15	01100000	00000000

Tabela 20: Resultado Esperado para a Simulação 17_5.

D.6. Simulação 17_6

Funções a serem testadas:

- Canal 0 (com cálculo do CRC):
  - esperar o circuito entrar em sincronismo de multiquadro e os *bits E* iguais a '1' (não ativos) e forçar CRC errado, portanto deve-se inserir o resultado do cálculo local e como CRC está errado, os *bits E* devem estar ativos ('0').
  - fazer o *bit A* (Indicação de Alarme Remoto) do enlace recebido ativo ('1'), portanto o *hardware* é transparente a eles.
  - fazer *ctr1.esn*='0' (modo transparente para os *bits* S_n), para que passe o enlace recebido nestes *bits*.
  - geração do CRC.
- Canais de Voz:
  - programar a memória de controle com operação normal e todo canal alocado.

Como implementar este teste

- Inicializar o circuito com um pulso de *reset* em *RESQ*.
- Fazer *INSERR*='0', *SIPA*='0', *ctr1.byp*='0', *ctr1.xais*='0' (valor inicial), *ctr2.lre*='0' (valor inicial), *POE2*='0', *POE1*='1', *ctr1.rdis*='0' (valor inicial), *ctr1.esn*='0' (valor inicial - modo transparente *ROID*), *NTESTE1*='0', *NTESTE2*='0', *ctr2.exra*='0' (valor inicial), *ctr2.vxra*='0' (valor inicial), *ctr3.crci*='0' (valor inicial), *ctr3.lbme*='0' (valor inicial) *XDI*='0' e *bits MC6* a *MC0* da memória de controle iguais a "0001111" (operação normal com todo canal alocado).
- Fazer *ROID* receber: palavra de quadro, *bit* 2 da próxima palavra igual a '1', palavra de quadro, 2 palavras de multiquadro (entra em sincronismo de multiquadro), 2 multiquadros com estrutura de quadro correta, com o *bit A*='1', *bits* S_n iguais a '0', CRC errado e os *bits E* iguais a '1'.

Com isto os *bits E* devem receber o resultado do cálculo local e, como CRC está errado, os *bits E* devem estar ativos ('0'). Portanto, o sinal de saída *XULR* esperado, deve ser o apresentado na tabela 21.

Quadro	Canal 0	Canal 31	Quadro	Canal 0	Canal 31
0	$(C_1)0011011$	00000000	8	$(C_1)0011011$	00000000
1	01100000	00000000	9	11100000	00000000
2	$(C_2)0011011$	00000000	10	$(C_2)0011011$	00000000
3	01100000	00000000	11	11100000	00000000
4	$(C_3)0011011$	00000000	12	$(C_3)0011011$	00000000
5	11100000	00000000	13	01100000	00000000
6	$(C_4)0011011$	00000000	14	$(C_4)0011011$	00000000
7	01100000	00000000	15	01100000	00000000

Tabela 21: Resultado Esperado para a Simulação 17_6.

D.7. Simulação 17_7

Funções a serem testadas:

- Canal 0 (com cálculo do CRC):
  - esperar o circuito entrar em sincronismo de multiquadro e os *bits E* iguais a '1' (não ativos) e forçar CRC correto, portanto deve-se inserir o resultado do cálculo local e como CRC está correto, os *bits E* devem estar não ativos ('1').
  - fazer o *bit A* (Indicação de Alarme Remoto) do enlace recebido ativo ('1'), portanto o *hardware* é transparente a eles.
  - fazer *ctr1.esn*='0' (modo transparente para os *bits* S_n), para que passe o enlace recebido nestes *bits*.
  - geração do CRC.
- Canais de Voz:
  - programar a memória de controle com operação normal e todo canal alocado.

Como implementar este teste

- Inicializar o circuito com um pulso de *reset* em *RESQ*.
- Fazer *INSERR*='0', *SIPA*='0', *ctr1.byp*='0', *ctr1.xais*='0' (valor inicial), *ctr2.lre*='0' (valor inicial), *POE2*='0', *POE1*='1', *ctr1.rdis*='0' (valor inicial), *ctr1.esn*='0' (valor inicial - modo transparente *ROID*), *NTESTE1*='0', *NTESTE2*='0', *ctr2.exra*='0' (valor inicial), *ctr2.vxra*='0' (valor inicial), *ctr3.crci*='0' (valor inicial), *ctr3.lbme*='0' (valor inicial) *XDI*='0' e *bits MC6* a *MC0* da memória de controle iguais a "0001111" (operação normal com todo canal alocado).
- Fazer *ROID* receber: palavra de quadro, *bit* 2 da próxima palavra igual a '1', palavra de quadro, 2 palavras de multiquadro (entra em sincronismo de multiquadro), 2 multiquadros com estrutura de quadro correta, com o *bit A*='1', *bits* S_n iguais a '0', CRC correto e os *bits E* iguais a '1'.

Com isto os *bits E* devem receber o resultado do cálculo local e, como CRC está correto, os *bits E* devem estar não ativos '1'. Portanto, o sinal de saída *XULR* esperado, deve ser o apresentado na tabela 22.

Quadro	Canal 0	Canal 31	Quadro	Canal 0	Canal 31
0	$(C_1)0011011$	00000000	8	$(C_1)0011011$	00000000
1	01100000	00000000	9	11100000	00000000
2	$(C_2)0011011$	00000000	10	$(C_2)0011011$	00000000
3	01100000	00000000	11	11100000	00000000
4	$(C_3)0011011$	00000000	12	$(C_3)0011011$	00000000
5	11100000	00000000	13	11100000	00000000
6	$(C_4)0011011$	00000000	14	$(C_4)0011011$	00000000
7	01100000	00000000	15	11100000	00000000

Tabela 22: Resultado Esperado para a Simulação 17_7.

D.8. Simulação 17_8

Funções a serem testadas:

- Canal 16:
  - no intervalo de tempo de canal 16, insere o enlace montado internamente.
- Canais de Voz:
  - programar a memória de controle com operação normal e todo canal alocado.

Como implementar este teste

- Inicializar o circuito com um pulso de *reset* em *RESQ*.
- Fazer *INSERR*='0', *SIPA*='0', *ctrl.byp*='0', *ctrl.xais*='0' (valor inicial), *ctr2.lre*='0' (valor inicial), *POE2*='0', *POE1*='1', *ctrl.rdis*='0' (valor inicial), *ctrl.esn*='0' (valor inicial - modo transparente *ROID*), *NTESTE1*='1', *NTESTE2*='1', *ctr3.lbme*='0' (valor inicial) *XDI*='1', *CSTR*='0', *ROID*='1', *ctr3.c16a*='0' (valor inicial), *ctr3.c16b*='0' (valor inicial) e *bits MC6* a *MC0* da memória de controle iguais a "0001111" (operação normal com todo canal alocado).

Com isto o canal 16 deve ser igual ao valor do sinal *XDI*. Portanto, o sinal de saída *XULR* esperado, deve ser o apresentado na tabela 23.

Quadro	Canal 0	Canais de Voz	Canal 16	Quadro	Canal 0	Canais de Voz	Canal 16
0	00011011	11111111	11111111	8	00011011	11111111	11111111
1	01111111	11111111	11111111	9	11111111	11111111	11111111
2	00011011	11111111	11111111	10	00011011	11111111	11111111
3	01111111	11111111	11111111	11	11111111	11111111	11111111
4	00011011	11111111	11111111	12	00011011	11111111	11111111
5	11111111	11111111	11111111	13	01111111	11111111	11111111
6	00011011	11111111	11111111	14	00011011	11111111	11111111
7	01111111	11111111	11111111	15	01111111	11111111	11111111

Tabela 23: Resultado Esperado para a Simulação 17_8.

D.9. Simulação 17_9

Funções a serem testadas:

- Canal 16:
  - no intervalo de tempo de canal 16, insere o sinal vindo do pino *CSTR*.
- Canais de Voz:
  - programar a memória de controle com operação normal e todo canal alocado.

Como implementar este teste

- Inicializar o circuito com um pulso de *reset* em *RESQ*.

- Fazer *INSERR*='0', *SIPA*='0', *ctrl.byp*='0', *ctrl.xais*='0' (valor inicial), *ctr2.lre*='0' (valor inicial), *POE2*='0', *POE1*='1', *ctrl.rdis*='0' (valor inicial), *ctrl.esn*='0' (valor inicial - modo transparente *ROID*), *NTESTE1*='1', *NTESTE2*='1', *ctr3.lbme*='0' (valor inicial) *XDI*='1', *CSTR*='0', *ROID*='1', *ctr3.c16a*='1', *ctr3.c16b*='0' (valor inicial) e *bits MC6* a *MC0* da memória de controle iguais a "0001111" (operação normal com todo canal alocado).

Com isto o canal 16 deve ser igual ao valor do sinal *CSTR*. Portanto, o sinal de saída *XULR* esperado, deve ser o apresentado na tabela 24.

Quadro	Canal 0	Canais de Voz	Canal 16	Quadro	Canal 0	Canais de Voz	Canal 16
0	*00011011*	*11111111*	*00000000*	8	*00011011*	*11111111*	*00000000*
1	*01111111*	*11111111*	*00000000*	9	*11111111*	*11111111*	*00000000*
2	*00011011*	*11111111*	*00000000*	10	*00011011*	*11111111*	*00000000*
3	*01111111*	*11111111*	*00000000*	11	*11111111*	*11111111*	*00000000*
4	*00011011*	*11111111*	*00000000*	12	*00011011*	*11111111*	*00000000*
5	*11111111*	*11111111*	*00000000*	13	*01111111*	*11111111*	*00000000*
6	*00011011*	*11111111*	*00000000*	14	*00011011*	*11111111*	*00000000*
7	*01111111*	*11111111*	*00000000*	15	*01111111*	*11111111*	*00000000*

Tabela 24: Resultado Esperado para a Simulação 17_9.

D.10. Simulação 17_10

Funções a serem testadas:

- Canal 16:
  - no intervalo de tempo dos canais 16 e 17, insere o sinal vindo do pino *CSTR*.
- Canais de Voz:
  - programar a memória de controle com operação normal e todo canal alocado.

Como implementar este teste

- Inicializar o circuito com um pulso de *reset* em *RESQ*.
- Fazer *INSERR*='0', *SIPA*='0', *ctrl.byp*='0', *ctrl.xais*='0' (valor inicial), *ctr2.lre*='0' (valor inicial), *POE2*='0', *POE1*='1', *ctrl.rdis*='0' (valor inicial), *ctrl.esn*='0' (valor inicial - modo transparente *ROID*), *NTESTE1*='1', *NTESTE2*='1', *ctr3.lbme*='0' (valor inicial) *XDI*='1', *CSTR*='0', *ROID*='1', *ctr3.c16a*='0' (valor inicial), *ctr3.c16b*='1' e *bits MC6* a *MC0* da memória de controle iguais a "0001111" (operação normal com todo canal alocado).

Com isto o canal 16 deve ser igual ao valor do sinal *CSTR*. Portanto, o sinal de saída *XULR* esperado, deve ser o apresentado na tabela 25.

Quadro	Canal 0	Canais de Voz	Canais 16/17	Quadro	Canal 0	Canais de Voz	Canais 16/17
0	*00011011*	*11111111*	*00000000*	8	*00011011*	*11111111*	*00000000*
1	*01111111*	*11111111*	*00000000*	9	*11111111*	*11111111*	*00000000*

Tabela 25: Resultado Esperado para a Simulação 17_10.

Quadro	Canal 0	Canais de Voz	Canais 16/17	Quadro	Canal 0	Canais de Voz	Canais 16/17
2	*00011011*	*11111111*	*00000000*	10	*00011011*	*11111111*	*00000000*
3	*01111111*	*11111111*	*00000000*	11	*11111111*	*11111111*	*00000000*
4	*00011011*	*11111111*	*00000000*	12	*00011011*	*11111111*	*00000000*
5	*11111111*	*11111111*	*00000000*	13	*01111111*	*11111111*	*00000000*
6	*00011011*	*11111111*	*00000000*	14	*00011011*	*11111111*	*00000000*
7	*01111111*	*11111111*	*00000000*	15	*01111111*	*11111111*	*00000000*

Tabela 25: Resultado Esperado para a Simulação 17_10.

D.11. Simulação 17_11

Funções a serem testadas:

- Todo o enlace de transmissão é invertido (*INSERR*='1').
- Canal 16:
  - no intervalo de tempo dos canais 16, 17 e 18, insere o sinal vindo do pino *CSTR*.
- Canais de Voz:
  - programar a memória de controle com operação normal e todo canal alocado.

Como implementar este teste

- Inicializar o circuito com um pulso de *reset* em *RESQ*.
- Fazer *INSERR*='1', *SIPA*='0', *ctrl.byp*='0', *ctrl.xais*='0' (valor inicial), *ctr2.lre*='0' (valor inicial), *POE2*='0', *POE1*='1', *ctrl.rdis*='0' (valor inicial), *ctrl.esn*='0' (valor inicial - modo transparente *ROID*), *NTESTE1*='1', *NTESTE2*='1', *ctr3.lbme*='0' (valor inicial) *XDI*='1', *CSTR*='0', *ROID*='1', *ctr3.c16a*='1', *ctr3.c16b*='1' e *bits MC6* a *MC0* da memória de controle iguais a "0001111" (operação normal com todo canal alocado).

Com isto o canal 16 deve ser igual ao sinal em *CSTR*. Portanto, o sinal de saída *XULR* esperado, deve ser o apresentado na tabela 26.

Q	Canal 0	Canais Voz	Canais 16/17/18
0	*NOT(00011011)*	*NOT(11111111)*	*NOT(00000000)*
1	*NOT(01111111)*	*NOT(11111111)*	*NOT(00000000)*
2	*NOT(00011011)*	*NOT(11111111)*	*NOT(00000000)*
3	*NOT(01111111)*	*NOT(11111111)*	*NOT(00000000)*
4	*NOT(00011011)*	*NOT(11111111)*	*NOT(00000000)*
5	*NOT(11111111)*	*NOT(11111111)*	*NOT(00000000)*
6	*NOT(00011011)*	*NOT(11111111)*	*NOT(00000000)*
7	*NOT(0111111)*	*NOT(11111111)*	*NOT(00000000)*
8	*NOT(00011011)*	*NOT(11111111)*	*NOT(00000000)*
9	*NOT(11111111)*	*NOT(11111111)*	*NOT(00000000)*
10	*NOT(00011011)*	*NOT(11111111)*	*NOT(00000000)*

Tabela 26: Resultado Esperado para a Simulação 17_11.

Q	Canal 0	Canais Voz	Canais 16/17/18
11	*NOT(11111111)*	*NOT(11111111)*	*NOT(00000000)*
12	*NOT(00011011)*	*NOT(11111111)*	*NOT(00000000)*
13	*NOT(01111111)*	*NOT(11111111)*	*NOT(00000000)*
14	*NOT(00011011)*	*NOT(11111111)*	*NOT(00000000)*
15	*NOT(01111111)*	*NOT(11111111)*	*NOT(00000000)*

Tabela 26: Resultado Esperado para a Simulação 17_11.

D.12. Simulação 17_12

Funções a serem testadas:

- Teste das funções de inserção de *AIS* e *IDLE* no enlace de transmissão.

Como implementar este teste

- Inicializar o circuito com um pulso de *reset* em *RESQ*.
- Fazer *INSERR*='0', *SIPA*='0', *ctr1.byp*='0', *ctr2.lre*='0' (valor inicial), *NTESTE1*='1', *NTESTE2*='1', *XDI*='1'.
- Fazer os *bits MC6* a *MC0* da memória de controle iguais a "1001111" nos canais pares (inserção de *IDLE* no enlace de transmissão) e iguais a "0001111" nos canais ímpares (operação normal com todo canal alocado).
- Fazer *ctr1.xais*='1' durante dois multiquadros e observar *XULR* que deve ser sempre igual a '1' e *ctr1.xais*='0' durante os dois multiquadros seguintes.

Com isto o sinal de saída *XULR* esperado, deve ser igual a "01010100" nos canais pares e igual a "11111111" nos canais ímpares.

D.13. Simulação 17_13

Funções a serem testadas:

- Teste da função de *bypass* sob controle do *bit ctr1.byp*.

Como implementar este teste

- Inicializar o circuito com um pulso de *reset* em *RESQ*.
- Fazer *INSERR*='0', *SIPA*='0', *ctr2.lre*='0' (valor inicial), *NTESTE1*='0', *NTESTE2*='0', *XDI*='1', *ctr3.lbme*='0' (valor inicial), *ctr1.xais*='0' (valor inicial), *ctr1.esn*='0' (valor inicial), *ctr2.exra*='1', *ctr2.vxra*='1', *POE2*='0', *POE1*='1', *ctr1.rdis*='0' (valor inicial) e fazer os *bits MC6* a *MC0* da memória de controle iguais a "0001111" (operação normal com todo canal alocado).
- Fazer o sinal do pino de entrada *ROID* igual ao apresentado na tabela 27.

Quadro	Canal 0	Canal 31	Quadro	Canal 0	Canal 31
0	*00011011*	*00000000*	8	*00011011*	*00000000*
1	*01000000*	*00000000*	9	*11000000*	*00000000*
2	*00011011*	*00000000*	10	*00011011*	*00000000*
3	*01000000*	*00000000*	11	*11000000*	*00000000*
4	*00011011*	*00000000*	12	*00011011*	*00000000*
5	*11000000*	*00000000*	13	*01000000*	*00000000*
6	*00011011*	*00000000*	14	*00011011*	*00000000*
7	*01000000*	*00000000*	15	*01000000*	*00000000*

Tabela 27: Sinal de Entrada no Pino *ROID* para a Simulação 17_13.

- Fazer *ctrl.byp*='1' (valor inicial) durante dois multiquadros e observar *XULR* que deve ser igual a *ROID* atrasado de 1 *bit*.
- Fazer *ctrl.byp*='0' durante os dois multiquadros seguintes e observar *XULR* que deve ser igual ao apresentado na tabela 28.

Quadro	Canal 0	Canal 31	Quadro	Canal 0	Canal 31
0	*00011011*	*11111111*	8	*00011011*	*11111111*
1	*01100000*	*11111111*	9	*11100000*	*11111111*
2	*00011011*	*11111111*	10	*00011011*	*11111111*
3	*01100000*	*11111111*	11	*11100000*	*11111111*
4	*00011011*	*11111111*	12	*00011011*	*11111111*
5	*11100000*	*11111111*	13	*01100000*	*11111111*
6	*00011011*	*11111111*	14	*00011011*	*11111111*
7	*01100000*	*11111111*	15	*01100000*	*11111111*

Tabela 28: Resultado Esperado para a Simulação 17_13.

Nota: O importante a observar aqui, é que na mudança de funcionamento normal para bypass, a fase do sinal de saída XULR tem que ser a mesma.

D.14. Simulação 17_14

Funções a serem testadas:

- Teste da função de *bypass* sob controle do pino de entrada *SIPA*.

Como implementar este teste

- Repetir a simulação anterior, porém fazer o controle do *bypass* pelo pino *SIPA*, deixando *ctrl.byp*='0'.

3.2.7 Síntese/Otimização Lógica

Nesta etapa, a partir de uma descrição do projeto em *VHDL*, o objetivo é produzir inicialmente um circuito genérico independente de tecnologia e, então, uma implementação a nível de portas [45], como pode ser visto na figura 39 [46].

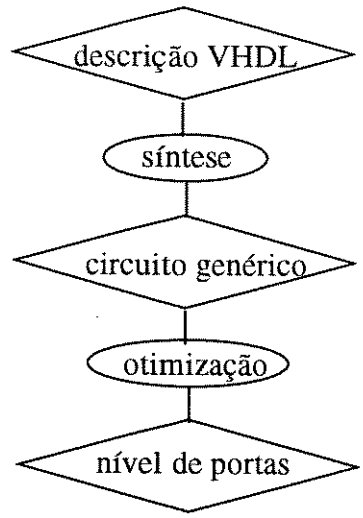


Figura 39: Detalhamento de Parte do Fluxo de Projeto.

Aqui, utilizou-se como ferramenta a família *Autologic* da *Mentor Graphics Corporation*, que fornece síntese alto nível, otimização lógica e mapeamento tecnológico, automatizando a implementação a nível de portas de dispositivos *ASICs* e *FPGAs*. O *Autologic* melhora a produtividade do projeto, pois diminui dramaticamente o tempo necessário para a geração de esquemáticos e elimina os erros de projeto inerentes a metodologia de geração não automática de esquemático. Pode, também, facilmente redirecionar circuitos já mapeados em uma tecnologia para outra, permitindo o re-uso de projetos existentes.

A família *Autologic* inclui o *Autologic VHDL* que gera projetos a nível de portas diretamente do *VHDL*; e o *Autologic*, que após a síntese de alto nível pode direcionar seu projeto para uma tecnologia apropriada, utilizando otimização lógica e algorítmos de mapeamento tecnológico [10].

Esta etapa consiste em, inicialmente, através do *Autologic VHDL*, fazer uma síntese do código *VHDL* de todo o *TB47* para portas genéricas, independente de tecnologia. Em seguida, utilizando-se o *Autologic* é feita uma otimização lógica e mapeamento para a tecnologia destino. Para isto é necessário definir alguns parâmetros que estão explicados abaixo.

Quando existem restrições muito apertadas, a otimização pode ser um problema complexo. Porém, a maioria dos circuitos pode ser otimizada usando um simples procedimento com bons resultados dentro de um tempo razoável [24], o que é o caso do *TB47*.

Os passos básicos nesta etapa estão descritos a seguir:

1. Definição de Parâmetros de Ambiente

Os parâmetros de ambiente no *Autologic* tipicamente controlam a temperatura, tensão e *corner* de processo usados para calcular os valores de *timing* no circuito. Normalmente, os valores *default* são suficientes para a maioria dos usuários. Na maioria dos *Design Kits* os valores *default* modelam as piores condições de *timing* possíveis, o que foi utilizado no caso do *TB47*.

2. Definição das restrições do circuito

Para cada pino I/O do circuito o usuário deve fazer as seguintes perguntas e usar a informação para definir as restrições apropriadas.

a. Qual é a relação de *timing* entre o pino e o(s) principal(ais) relógio(s) do circuito? (quando se tem um único relógio, normalmente esta pergunta se torna fácil).

b. A restrição de carga *default* (específica da tecnologia) é adequada para este sinal? Isto é especialmente importante para sinais de relógio, linhas de *reset* e qualquer outro sinal que tenha um grande *fan in* ou *fan out*.

É importante que o projetista entenda bem as relações de *timing* do circuito. Definir as restrições impropriamente ou fora da realidade pode conduzir a tempos maiores de execução e uma grande frustração. Estas restrições, podem ser definidas como um conjunto de relações no domínio do tempo entre os sinais do circuito, que são [47]:

- *setup*: os sinais de dados devem estar estáveis por um determinado tempo antes do sinal de relógio transicionar;
- *hold*: os sinais de dados devem manter seus valores por algum tempo depois do relógio transicionar (ver figura 40);

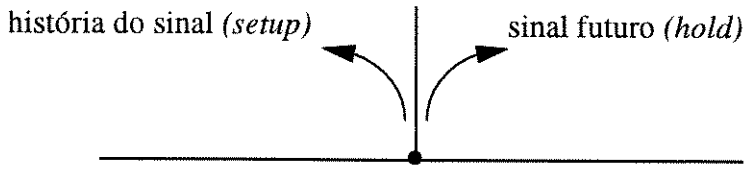


Figura 40: Algumas Restrições de *Timing*.

- *waveform*: inclui largura de pulso alta e baixa e transições de subida e descida;
- *spikes*.

No caso do *TB47*, as relações de *timing* possuem uma boa folga. Portanto, não foi necessário restringir estes itens.

3. Definição dos *Recipes* de Otimização

Um conjunto de algorítmos de otimização referem-se a um *recipe* de otimização. Com o *Autologic* são fornecidos vários *recipes* básicos para desempenhar otimização de área e *timing*. É necessário adquirir uma certa experiência para saber como otimizar o uso dos *recipes*. Uma boa regra é executar um mínimo de dois *recipes* em uma determinada otimização, para se conseguir resultados bons e consistentes [24].

a. Para Otimizar os Resultados de Área

Para se conseguir os melhores resultados de área do *Autologic*, deve-se executar múltiplas otimizações. Para a maioria dos circuitos, o conjunto básico de algorítmos que você deve aplicar para conseguir bons resultados de área (área mínima), são (caso do *TB47*):

"Opt_Area low Factor"

"Opt_Area high Factor"

"Opt_Timing low"

O passo de *timing* geralmente irá limpar o circuito, fazendo com que *nets* de alto *fan out* tenham carga suficiente para eliminar grandes tempos de subida. Se você ainda necessitar de melhorias, execute passos adicionais *"Opt_Area high"* até não precisar mais de melhorias.

b. Para Otimizar os Resultados de *Timing*

Para circuitos simples, o melhor método para conseguir bons resultados de *timing* (menor atraso) é executar os *recipes* de área e *timing* mencionados acima e então executar um para *"Opt_Timing high"*.

3.2.8 Alocação/Roteamento (*Layout*)

Com o esquemático resultante da etapa de síntese e otimização lógica, a próxima etapa consiste em convertê-lo para um netlist no formato *Xilinx* para, então executar a alocação e roteamento em uma *FPGA*, utilizando o *software Partition, Place and Route* (*PPR*) da *Xilinx*.

Para isto, serão executados os passos descritos a seguir [33, 48], que no caso do *TB47* serão executados duas vezes, uma para cada componente *Xilinx* que será utilizado.

1. Conversão esquemático *Autologic* para *XNF (Xilinx Netlist Format)*:

Conversão do esquemático gerado pelo *AutoLogic* para um netlist no formato *Xilinx*, utilizando-se primeiro o programa *enwrite* da *Mentor* que faz a conversão do esquemático para um formato *EDIF*. Em seguida, utilizou-se o programa *edif2xnf* da *Xilinx* que converte o formato *EDIF* para o formato *XNF* (ver figura 41).

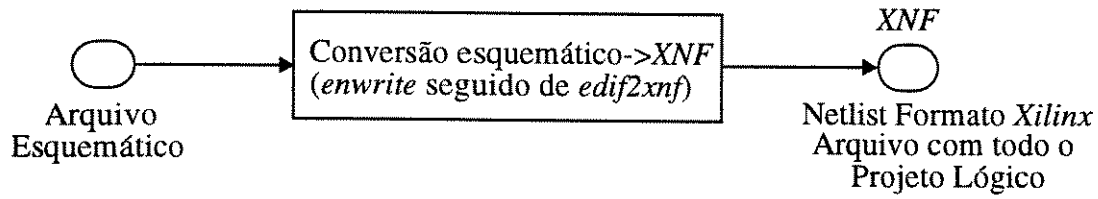


Figura 41: Conversão *XNF*.

Comandos utilizados:

a. enwrite mg_design -rcf config_file -wef tb47_X.edif

onde:

mg_design: nome do projeto *Mentor Graphics* a ser lido.

-rcf: especificação de um arquivo de configuração (config_file).

-wef: nome do arquivo *EDIF* a ser gerado (tb47_1.edif/tb47_2.edif).

b. edif2xnf tb47_X.edif -p 4008pq208

onde:

tb47_X.edif: arquivo de entrada no formato *EDIF*

-p: define qual componente *FPGA Xilinx* será usado (*4008pq208*).

O arquivo de saída será tb47_X.xnf.

2. Partição, Alocação e Roteamento:

O programa *Partition, Place and Route (PPR)*, da *Xilinx*, usa algorítmos sofisticados para determinar a partiçao, alocação e roteamento ótimos para o projeto. Lógicas não usadas (se houver) serão deletadas e a lógica remanescente será agrupada (mapeada) em *Configurable Logic Blocks* e *I/O Blocks*. Portanto, utilizando um arquivo *XNF* como entrada, o programa particiona a lógica em *CLBs* e *IOBs*, posiciona a lógica particionada dentro de posições da *LCA* (*Logic Cell Array*) e, então roteia o projeto. O resultado desta etapa é um arquivo *LCA* alocado e roteado (ver figura 42).

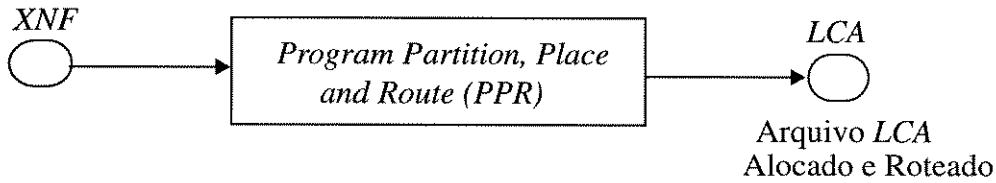


Figura 42: Partição, Alocação e Roteamento.

O comando utilizado é:

a. PPR tb47_X.xnf

onde:

tb47_X.xnf: arquivo de entrada no formato XNF.

O arquivo de saída será tb47_X.lca.

3.2.9 Simulação a Nível de Portas com Informações de *Timing*

Através do processo de implementação do projeto, cada bloco da lógica deve ser simulado contra o modelo *VHDL* original para assegurar que sua funcionalidade esteja correta. Ou seja, a verificação formal de um circuito consiste em constatar, que para todos os valores possíveis de entrada, a implementação do projeto (como ele é construído) realiza a sua especificação (seu comportamento esperado) [49]. Pois é um requisito das ferramentas de síntese que a saída de uma simulação VHDL seja igual a saída de uma simulação do hardware sintetizado, com os mesmos vetores aplicados as suas entradas [45]. Portanto, com o mesmo sistema ilustrado na figura 37, e utilizando-se os mesmos estímulos e resultados esperados gerados na simulação funcional para validação das interfaces, é feita a avaliação do *TB47* mapeado para a tecnologia destino e com informações de *timing*. As informações de *timing*, que são resultantes da etapa de alocação/roteamento (*layout*), ficam armazenadas em um arquivo que precisa ser incorporado ao projeto [50].

Aqui foi utilizada a ferramenta *QuickSim II*. Problemas aqui encontrados, podem implicar na repetição das seguintes etapas:

- Alocação/Roteamento: como o resultado desta etapa é aleatório, ou seja, pode apresentar resultados bem diferentes cada vez que for executada, os atrasos resultantes podem ser muito grandes, tornando-se necessário repetir esta etapa até se conseguir resultados (atrasos) satisfatórios.
- Síntese/Otimização Lógica: alteração das restrições do circuito ou dos *recipes* de otimização, de acordo com o necessário.

- Alteração dos Códigos *VHDL*: apesar da funcionalidade estar correta, a estrutura a ser sintetizada é muito dependente de como foi descrito o código *VHDL*.

3.2.10 Implementação Física (*Bitstream Generation*)

A implementação física deste projeto consiste na conversão de um arquivo *LCA* alocado e roteado em um arquivo para programação de uma *EPROM*. Para isto, foram executados os seguintes passos [33, 48]:

1. Geração do Arquivo *Bitstream*:

Esta etapa consiste na execução do programa *Makebits* da *Xilinx*, para compilar o projeto em *bitstreams* de configuração para projetos *LCA* (*Logic Cell Array*). O arquivo *BIT* contém os dados de configuração binária para programação de um dispositivo *LCA*, para desempenhar a função do projeto, descrevendo suas funções lógicas internas e interconexões (ver figura 43).

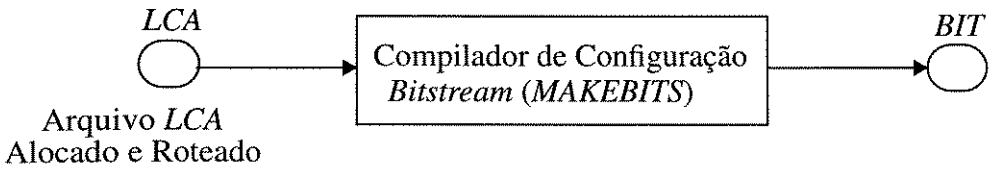


Figura 43: Geração do *Bitstream*.

No caso do *TB47*, está etapa será executada duas vezes, uma para cada arquivo *LCA*. O comando utilizado é:

a. makebits tb47_X.lca

onde:

tb47_X.lca: arquivo *LCA* de entrada.

O arquivo de saída será tb47_X.bit.

2. Verificação do Projeto *In-circuit*, tempo real:

A verificação *in-circuit* permite que você veja imediatamente como seu projeto *LCA* funciona. O programa *MAKEPROM*, da *Xilinx*, formata o arquivo *bitstream* de configuração (*BIT*) para conter informações de configuração *PROM* para os dispositivos *LCA*. O arquivo de saída está no formato *PROM MCS-86* (*Intel*) para ser utilizado em um programador de *EPROM*, para configuração da mesma (ver figura 44).

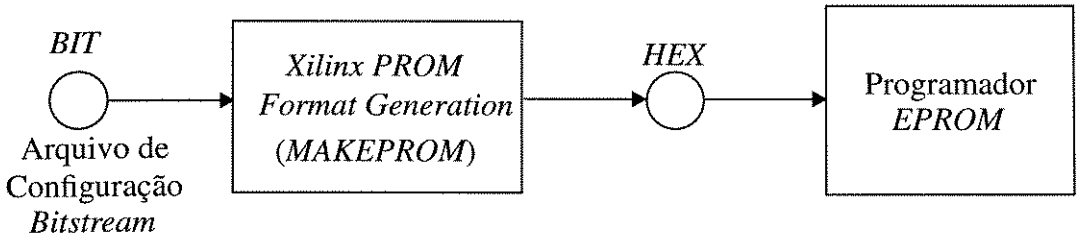


Figura 44: Verificação do Projeto *In-circuit*, tempo real.

No caso do *TB47*, tem-se uma única *EPROM* com informações para as duas *FPGAs* a serem utilizadas. Portanto, deve-se gerar um único arquivo para programação da *EPROM*, através de dois arquivos no formato *BIT*. O comando utilizado é:

a. makeprom -s 512 -f MCS -u 00 tb47_2.bit tb47_1.bit -o tb47.mcs

onde:

-s: define o tamanho da *PROM* (512 Kbytes)

-f: Define o formato da *PROM* (*MCS-86 Intel*)

-u: carrega o arquivo *BIT* a partir do endereço especificado (00Hex)

-o: define o nome do arquivo de saída (tb47.mcs)

Os arquivos de entrada são tb47_2.bit e tb47_1.bit e, serão armazenados na *EPROM* nesta sequência.

3. Programação da *EPROM*:

Com o arquivo tb47.mcs gerado na etapa anterior, e utilizando-se um programador de *EPROM* paralela (caso do *TB47*) ou serial, a *EPROM* é programada com os dados de configuração da *LCA*. Na placa, os componentes *LCA* e a *EPROM* são conectados juntos e de uma forma conveniente, de tal forma que os dados de configuração para definir a função e interconexão dentro de um *Logic Cell Array* são carregados automaticamente no *power-up*, quando *Vcc* atingir a tensão na qual o dispositivo *LCA* começa a operar (nominalmente 2.5 a 3 V) - ver figura 45.

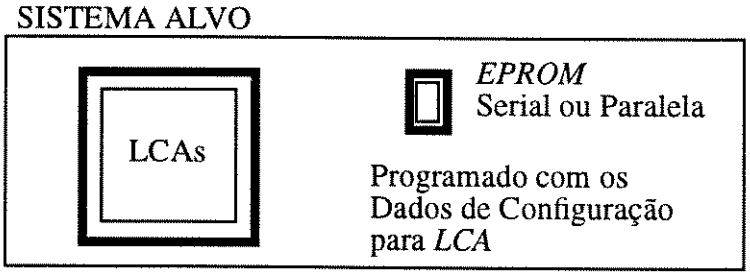


Figura 45: Programação da *EPROM*.

CAPÍTULO VI

Conclusão

Este circuito foi testado a nível de sistema em bancada de laboratório, com uma simulação da interface do processador. A próxima fase, em curso, consiste em um teste de bancada com a interface real do processador, ou seja, com a implementação do *software* para este sistema.

Este circuito foi testado isolado do sistema no esquipamento de teste *TERADYNE*, onde foram exercitadas todas as condições simuladas no *Quicksim II* do *software Mentor.*

Como esta primeira implementação é em *FPGA*, esta avaliação funcional feita, foi suficiente para validação do circuito, pois estruturalmente estes componentes são testados pelo fabricante.

Porém, é importante salientar que esta implementação inicial em *FPGA* é somente como protótipo para validação do sistema, por se tratar de uma idéia nova.

Para a fase de produção, depois do sistema já estar validado, a implementação prevista é em *Standard Cells,* onde será necessária a implementação de testabilidade. Como o circuito foi descrito em linguagem de alto nível *VHDL*, esta migração de uma tecnologia para outra é relativamente simples.

Atualmente, já existe um novo protótipo em *Standard Cells* para teste de campo, que já foi testado funcionalmente no equipamento *TERADYNE* e está funcionando.

A etapa final, que está em fase de acabamento, consiste em implementar testabilidade neste circuito (incluindo auto teste do circuito, teste das memórias e *boundary scan*) e, dependendo das realimentações do teste de campo, alterar ou não parte da funcionalidade do mesmo.

Este circuito embora tenha sido concebido para o sistema *CLAD*, pode ser empregado em qualquer outro sistema que necessite de interface com equipamentos digitais (família de multiplexadores) e centrais telefônicas digitais, que empreguem o *PCM-30*/CRC-4.

CAPÍTULO VII

Bibliografia

1. *Getting Started*, version 3.0, November, 1992, Xilinx.

2. D. R. Coelho, *Check your Designs with VHDL Test Benches*, Electronic Design, December 19, 1991, page 73.

3. Perfil Tecnológico TRÓPICO RA, Realização DA/DDRH/CNTr/Tecnologia em Capacitação e DPD/CPqD/DDS/Seção de Especificação e Validação de Sistemas, Produzido e impresso no CNTr/TELEBRÁS.

4. Estudo de Viabilidade do Circuito Integrado *TB47*, novembro de 1993, Divisão de Microeletrônica, Departamento de Tecnologias Básicas, CPqD Telebrás.

5. Especificação de Características - Placa Controladora da UDL do CLAD (CUC) - Projeto CLAD, 11 de novembro de 1993, Divisão de Arquiteturas Básica, Departamento de Desenvolvimento de Sistemas, CPqD Telebrás.

6. CCITT Blue Book.

7. V. A. Valenzuela, Código Detector de Erros para um Sistema de Transmissão Digital Baseado numa Configuração *Descrambler-Scrambler* - Características e Implementação, Relatório Técnico CPqD Telebrás, 1981.

8. *Autologic VHDL Synthesis Guide*, April 1992, Mentor Graphics Corporation.

9. H. Konuk and F. E. Marschner, *A VHDL-Driven Synthesis Environment*, J. Mermet (ed.), *VHDL for Simulation, Synthesis and Formal Proofs of Hardware*, 1992 Kluwer Academic Publishers.

10. *Autologic Family for Top Down Design*, Product Description, 1992, Mentor Graphics Corporation.

11. D. R. Coelho, *Follow Simple Rules to Create VHDL Model*, Electronic Design, June 14, 1990, page 65.

12. *IEEE Standard VHDL Language Reference Manual*, 1988, IEEE Std 1076-1987.

13. J. R. Armstrong, *Chip-Level Modeling with VHDL*, Prentice Hall, 1989.

14. R. Lipsett, E. Marschner and M. Shahdad, *VHDL - the language*, IEEE Design & Test, April 1986, 28-41.

15. R. Lipsett, C. Schaefer, and C. Ussery, *VHDL: Hardware Description and Design*, Kluwer Academic Publishers, Boston, MA, 1989.

16. *An Introduction Modeling VHDL*, 1992, Mentor Graphics Corporation.

17. D. R. Coelho, *The VHDL Handbook*, Boston Kluwer, Norwell, Mass., 1989.

18. J. Lis, and D. Gajski, *Synthesis from VHDL*, Proc. Int'l Conf. Computer Design, IEEE

Computer Soc. Press, Los Alamitos, Calif., 1988.

19. M. Crastes and A. Fonkoua, *HDLs and Logic Synthesis*, Esprit Project 2072 Report.
20. R. Camposano, L. F. Saunders and R. M. Tabet, *VHDL as Input for High-Level Synthesis*, IEEE Design & Test of Computers, March 1991.
21. J. R. Fox, *A Higher Level of Synthesis*, IEEE Spectrum, March, 1993, page 43.
22. K. Vahtra, *HDL Design Methods Simplify Specs and Boost Productivity*, EDN, June 21, 1990, page 247.
23. *Inside ASIC/FPGA, ASIC/FPGA Products, Top Down Design Solver*, 1995, Mentor Graphics Corporation.
24. *Application Note - Methods for Using Autologic in Top Down Design*, April 1994, Mentor Graphics Corporation.
25. J. Bergeron, *Guidelines for Writing VHDL Models in a Team Environment*, Bell-Northern Research Ltd.
26. *Mentor Graphics Glossary*, 1991-1995, Mentor Graphics Corporation.
27. *Design Architect Reference Manual*, 1992-1994, Mentor Graphics Corporation.
28. *System-1076 Design and Model Development Manual*, 1991-1994, Mentor Graphics Corporation.
29. *Quicksim II User's Manual*, 1990-1995, Mentor Graphics Corporation.
30. *Autologic VHDL Reference Manual*, 1990-1994, Mentor Graphics Corporation.
31. *Autologic II Reference Manual*, 1995, Mentor Graphics Corporation.
32. *Design Creation Glossary*, 1992-1994, Mentor Graphics Corporation.
33. *The Programmable Gate Array Data Book*, 1991, Xilinx.
34. W. A. M. V. Noije, *Design and Characterization of Uncommitted Logic Arrays in Single and Double Level Metal CMOS*, Ph.D Thesis, Maio 1995.
35. *Product Description and Selection Guide, FPGAs, EPLDs and Development System Software*, January 1994, Xilinx.
36. A. Hohl, *Incremental Design - Application of a Software-based Method for High-level Hardware Design with VHDL*, J. Mermet (ed.), *VHDL for Simulation, Synthesis and Formal Proofs of Hardware*, 1992 Kluwer Academic Publishers.
37. K. Khordoc, M. Biotteau, and E. Cerny, *Switch-Level Models in Multi-level VHDL Simulations*, J. Mermet (ed.), *VHDL for Simulation, Synthesis and Formal Proofs of Hardware*, 1992 Kluwer Academic Publishers.
38. F. Lémery, J. P. Morin, E. Nercessian, J. Benkoski, and D. Samani, *Behavioral Models for Complex Top Down Analog/Digital System Simulation*, 1994 European Simulation Conference, Modelling & Simulation, June 1994, Barcelona.
39. S. Carlson, and E. Girczyc, *Understanding Synthesis Begins with Knowing the Terminology*, EDN, September 3, 1992, page 125.
40. B. Tuck - Senior Editor, *The VHDL/Verilog debate continues. How will they share the*

coveted crown?, Computer Design, June, 1991.

41. V. Nagasamy, N. Berry, and C. Dangelo, *Specification, Planning and Synthesis in a VHDL Design Environment,* IEEE Design & Test of Computers, vol.9, no.2, June 1992.
42. J. Mouallem, J. M. Furtado Neto e R. C. Di Giorgio, Descrição VHDL das funções do TB47, Documento Interno do Centro de Pesquisa e Desenvovimento da Telebrás.
43. F. E. Marschner, *Evolutionary Processes in Language, Software, and System Design*, J. Mermet (ed.), *VHDL for Simulation, Synthesis and Formal Proofs of Hardware*, 1992 Kluwer Academic Publishers.
44. *An Introduction to Digital Simulation*, 1989, Mentor Graphics Corporation.
45. J. Elliott, and P. Harper, *Aspects of Optimization and Accuracy for VHDL Synthesis*, J. Mermet (ed.), *VHDL for Simulation, Synthesis and Formal Proofs of Hardware*, 1992 Kluwer Academic Publishers.
46. L. Lundberg, *Generating VHDL for Simulation and Synthesis from a High-Level DSP Design Tool*, J. Mermet (ed.), *VHDL for Simulation, Synthesis and Formal Proofs of Hardware*, 1992 Kluwer Academic Publishers.
47. F. Liu, and A. Pawlak, *Timing Constraint Checks in VHDL - a comparative study*, J. Mermet (ed.), *VHDL for Simulation, Synthesis and Formal Proofs of Hardware*, 1992 Kluwer Academic Publishers.
48. *XACT Reference Guide*, Volume I, October 1992, Xilinx.
49. D. Borrione, L. Pierre and A. Salem, *Formal Verification of VHDL Descriptions in Boyer-Moore: first results*, J. Mermet (ed.), *VHDL for Simulation, Synthesis and Formal Proofs of Hardware*, 1992 Kluwer Academic Publishers.
50. P. Connor, S. Nayak, J. Kraley, and V. Berman, *Delay Calculation and Back Annotation in VHDL Addressing the Requirements of ASIC Design*, J. Mermet (ed.), *VHDL for Simulation, Synthesis and Formal Proofs of Hardware*, 1992 Kluwer Academic Publishers.