UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA
DEPARTAMENTO DE COMUNICAÇÕES

MODELOS EM LINGUAGEM VHDL PARA EQUIPAMENTOS DA HIERARQUIA DIGITAL SÍNCRONA

Aleandro Soares Macedo
Orientador: Prof. Dr. Rege Romeu Scarabucci

Tese de Mestrado apresentada à Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas, como parte dos requisitos exigidos para a obtenção do título de MESTRE EM ENGENHARIA ELÉTRICA.

Agosto/1993

AGRADECIMENTOS

Ao Prof. Dr. Rege R. Scarabucci, meu orientador, pelo apoio e crédito que me tem dado desde meu ingresso na UNICAMP;

Aos meus amigos e colegas da FEE, principalmente, Marcio Dutra, José Carmelo e Bartolomeu Uchôa;

Ao CNPq, pelo apoio financeiro;

Aos meus familiares, especialmente meus pais e meus irmãos.

Dedico esta tese a meus pais,
Severiano e Maria Lucia,
e à minha namorada,
Lis.

Resumo

Em 1988, o CCITT padronizou um novo método para multiplexação digital. O novo padrão chamado Hierarquia Digital Síncrona (HDS), possibilita maior eficiência no transporte dos sinais nas futuras redes de telecomunicações.

O CCITT estabeleceu que os equipamentos HDS são compostos por blocos funcionais bem caracterizados, de tal modo que, pelo agrupamento desses vários blocos funcionais, obtém-se a funcionalidade completa.

As funções dos equipamentos HDS são de crosconexão transversal, "add-drop" e de terminação de linha. O que define a função do equipamento é o arranjo e os tipos dos blocos funcionais que o compõem.

A proposta do trabalho de tese é o desenvolvimento de modelos de circuitos lógicos para os blocos funcionais HDS.

A ferramenta computacional utilizada é a linguagem VHDL que permite projetar circuitos lógicos através de sua descrição comportamental. Utiliza-se esta característica da linguagem para superar a complexidade dos modelos. Uma outra característica da linguagem, o projeto estrutural, permite fazer conexões entre os modelos desenvolvidos para os blocos funcionais e assim constituir modelos para equipamentos HDS. Da mesma forma, conexões entre modelos de equipamentos permite constituir modelos para redes de equipamentos HDS.

Através de um simulador VHDL, os modelos são validados ao nível de rede, ou seja, simula-se uma rede de equipamentos HDS constituidos pelos modelos desenvolvidos para os blocos funcionais HDS.

GLOSSÁRIO

AIS - Alarm Indication Signal (Sinal de Indicação de Alarme)
APS - Automatic Protection Switching (Chaveamento Automático de Proteção)
AU - Administrative Unit (Unidade Administrativa)
AUG - Administrative Unit Group (Grupamento de Unidades Administrativas)
BER - Bit Error Ratio (Taxa de Erros de Bit)
BIP - Bit Interleaved Parity
CM - Connection Matrix (Matriz de Conexão)
DCC - Data Communications Channel (Canal de Comunicação de Dados)
EOW - Engineering Order - Wire (Canal de Serviço)
FAL - Frame Alignment Loss (Perda de Alinhamento de Quadro)
FEBE - Far End Block Error
FERF - Far End Receive Failure
HPA - Higher order Path Adaptation (Adaptação de Rotas de Alta Ordem)
HPC - Higher order Path Connection (Conexão de Rotas de Alta Ordem)
HPT - Higher order Path Termination (Terminação de Rotas de Alta Ordem)
LOP - Loss Of Pointer (Perda de Ponteiro)
LOS - Loss Of Signal (Perda de Sinal)
LPA - Lower order Path Adaptation (Adaptação de Rotas de Baixa Ordem)
LPC - Lower order Path Connection (Conexão de Rotas de Baixa Ordem)
LPT - Lower order Path Termination (Terminação de Rotas de Baixa Ordem)
MCF - Message Communications Function (Sistema de Comunicação de Mensagem)
MRTIE - Maximum Relative Time Interval Error
MS - Multiplex Section (Seção Multiplex)
MSOH - Multiplex Section Overhead (Supervisão de Seção Multiplex)
MSP - Multiplex Section Protection (Proteção de Seção Multiplex)
MST - Multiplex Section Termination (Terminação de Seção Multiplex)
MTG - Multiplex Timing Generator (Gerador de Tempo do Multiplex)
MTIE - Maximum Time Interval Error
MTPI - Multiples Timing Phisical Interface (Interface Física do Temporizador)
MTS - Multiplexer Timing Source (Temporizador do Multiplex)
NDF - New Data Flag (Sinalização de Novos Dados)
NU - National Use (Uso Nacional)
OHA - Overhead Access (Função de Acesso de Supervisão)
OOF - Out Of Frame (Fora de Alinhamento)
PDH - Plesiochronous Digital Hierarchy (Hierarquia Digital Plesiócrona)
PI - Phisical Interface (Interface Física)
POH - Path Overhead (Supervisão de Rota)
RS - Regenerator Section (Seção Regeneradora)
RSOH - Regenerator Section Overhead (Supervisão de Seção Regeneradora)
RST - Regenerator Section Termination (Terminação de Seção Regeneradora)
SA - Section Adaptation (Adaptação de Seção)
SDH - Synchronous Digital Hierarchy (Hierarquia Digital Síncrona)
SEMF - Synchronous Equipment Management Function (Gerenciamento de Equipamento Síncrono)
SPI - SDH Phisical Interface
STM - Synchronous Transport Module (Módulo Síncrono de Transporte)
TU - Tributary Unit (Unidade Afluente)
TUG - Tributary Unit Group (Grupamento de Unidades Afluentes)
VC - Virtual Container (Container Virtual)
VHDL - VHSIC Hardware Description Language
VHSIC - Very High Speed Integrated Circuit

Conteúdo

Capítulo 1

Introdução

Há poucas décadas atrás, foi inicada nos Estados Unidos a transição da transmissão analógica para a transmissão digital nas redes de telefonia. A transmissão digital permite transportar sinais por maiores distâncias e com maior qualidade. Dependendo da quantidade de dados a serem transmitidos entre localidades diferentes na rede, várias taxas de transmissão são utilizadas. Na Europa, no Brasil e na maioria dos países, estas taxas são de 2048 kb/s, 8448 kb/s, 34368 kb/s e 139264 kb/s.

Na menor taxa, 2048 kb/s, cada sinal contém 30 canais de telefone, incluindo dados de sinalização em posições bem definidas no sinal. A frequência de 2048 kb/s pode ter uma variação máxima de 50 ppm. Com o objetivo de transmitir mais informação em uma única linha, quatro sinais de 2048 kb/s são multiplexados para formar um sinal de 8448 kb/s. Devido ao fato de que os quatro sinais de 2048 kb/s têm frequências variáveis, com precisão de 50 ppm, eles devem ser mutuamente sincronizados antes de serem multiplexados.

Se uma taxa ainda maior é requerida, então quatro sinais de 8448 kb/s são multiplexados para formar um sinal de 34368 kb/s. Para se chegar finalmente à taxa de 139264 kb/s, quatro sinais de 34368 kb/s são multiplexados. Formou-se portanto, uma hierarquia de multiplexações que se convencionou chamar de Hierarquia Digital Plesiócrona.

No primeiro nível de multiplexação desta hierarquia, perde-se a possibilidade de retirar facilmente do sinal de 8448 kb/s um canal de telefone. Isto ocorre devido à necessidade de sincronização mutua dos sinais de 2048 kb/s. Isto também significa que deve-se passar por cada nível intermediário quando se vai demultiplexando do nível maior para o nível menor, por exemplo, não se pode obter um sinal de 2048 kb/s diretamente de um sinal de 34368 kb/s.

Uma outra característica problemática da Hierarquia Digital Plesiócrona é que não há padronização para a codificação do sinal óptico e também para os sinais de supervisão nos equipamentos de linha, que são específicos para cada fabricante.

Os requerimentos para o que deve ser transportado e a sua destinação na rede variam nos diferentes níveis na hierarquia. Isto significa que as companhias administradoras de telecomunicações frequentemente devem rearranjar as conexões entre os equipamentos de transmissão. Também, pode ser difícil utilizar a capacidade total da linha. Como exemplo, um sinal de 140 Mb/s com capacidade para 2000 canais de telefone, frequentemente pode transportar apenas 76% disso.

Para superar as deficiências apresentadas pela Hierarquia Digital Plesiócrona, discutidas acima, buscou-se o desenvolvimento de uma nova hierarquia de transmissão digital. Este trabalho começou nos Estados Unidos em 1985 e lá se chamou SONET, que significa "Synchronous Optical Network". O padrão SONET foi a base para o CCITT desenvolver o padrão SDH ("Synchronous Digital Hierarchy") ou HDS (Hierarquia Digital Síncrona). O nome indica que os sinais são construidos usando multiplexação síncrona e todos os sinais devem ser mutuamente sincronizados. O meio de transmissão deve ser preferencialmente a fibra óptica. Além disso, a nova hierarquia estabeleceu que as interfaces dos equipamentos fossem padronizadas para tornar possível a interconexão entre equipamentos de diferentes fabricantes. Isto significa que não só os sinais de linha devam ser padronizados, mas também as funções relativas à manutenção, supervisão, chaveamento de proteção e gerenciamento da rede.

O CCITT considerou também o mapeamento na Hierarquia Digital Síncrona, de células ATM (Asynchronous Transfer Mode) que tende a se tornar a forma dominante de transporte de sinais na Rede Digital de Serviços Integrados de Faixa Larga (B-ISDN).

Com as recomendações G.707, G.708 e G.709, o CCITT formou um conjunto coerente de especificações para a interface dos equipamentos HDS.

Na recomendação G.783, o CCITT define vários blocos funcionais para a composição de um equipamento HDS. A descrição é genérica e não sugere nenhuma partição física obrigatória de funções. Isto permite que os fabricantes tenham liberdade para desenvolver seus equipamentos da forma que melhor lhes convém.

A proposta deste trabalho é desenvolver modelos de circuitos lógicos para os blocos funcionais definidos pelo CCITT em sua recomendação G.783. Desenvolvem-se modelos comportamentais para os circuitos lógicos destes blocos. Um modelo comportamental é a etapa inicial na técnica de projeto "top-down". A partir de um modelo comportamental já testado e validado, pode se partir com segurança para as etapas seguintes de detalhamento do projeto.

O modelo comportamental permite obter um entendimento detalhado do circuito e permite verificar erros de lógica que fatalmente se comete em projetos de circuitos complexos. O trabalho de correção destes erros em um modelo comportamental significa um trabalho de depuração em "software". É portanto bem mais barato e rápido do que um trabalho de depuração em "hardware".

A ferramenta computacional utilizada é a linguagem VHDL que permite projetar circuitos lógicos através de sua descrição comportamental. Utiliza-se esta característica da linguagem para superar a complexidade dos modelos. Uma outra característica da linguagem, o projeto estrutural, permite fazer conexões entre os modelos desenvolvidos para os blocos funcionais e assim constituir modelos para equipamentos HDS. Da mesma forma, conexões entre modelos de equipamentos permite constituir modelos para redes de equipamentos HDS.

Através de um simulador VHDL, os modelos são validados ao nível de rede, ou seja, simula-se uma rede de equipamentos HDS constituidos pelos modelos desenvolvidos para os blocos funcionais HDS.

Além desta introdução, o trabalho está organizado em mais 5 capítulos, assim descritos:

- **Capítulo 2** - Apresentam-se os pontos de maior interesse da recomendação CCITT G.709, que trata das regras de multiplexação, alinhamento (processamento de ponteiro) e mapeamento das estruturas de multiplexagem síncrona.

- **Capítulo 3** - Apresentam-se os blocos funcionais definidos pelo CCITT na recomendação G.783 e analisa-se o processamento dos sinais por eles gerados.

- **Capítulo 4** - Faz-se uma breve apresentação da linguagem VHDL.

- **Capítulo 5** - Desenvolvem-se modelos em linguagem VHDL para blocos funcionais dos equipamentos HDS. Estes modelos são interfaceados para se obter modelos de equipamentos HDS. Estes por sua vez, são interfaceados para se obter um modelo de rede de equipamentos HDS. O modelo de rede é então simulado e os resultados são analisados.

- **Capítulo 6** - São apresentados os comentários finais.

- **Apêndice** - São mostradas as listagens de todas as entidades de projeto desenvolvidas.

Obs: os capítulos 2 e 3 apresentam os conhecimentos sobre a Hierarquia Digital Síncrona que são básicos para o entendimento dos modelos desenvolvidos no capítulo 5. Eles deram ao trabalho um volume exagerado. Porém, por se tratar de um tema ainda pouco divulgado, houve por bem mantê-los.

Capítulo 2

Estudo da Recomendação CCITT G-709

2.1 Introdução

A Recomendação G-709 trata da Estrutura de Multiplexação Síncrona a ser seguida pelos equipamentos da Hierarquia Digital Síncrona. Estes equipamentos têm a capacidade de mapear sinais com diversas finalidades. Especificamente, mostra-se o mapeamento dos sinais afluentes definidos pela Recomendação G-703:

- 1.544 kbits/s
- 2.048 kbits/s
- 6.312 kbits/s
- 34.368 kbits/s
- 44.736 kbits/s
- 13.9264 kbits/s

Para isto foram definidas três operações:

- Mapeamento
- Alinhamento
- Multiplexagem

Estas operações definem a estrutura geral de multiplexação da Hierarquia Digital Síncrona ilustrada na figura 2.1.

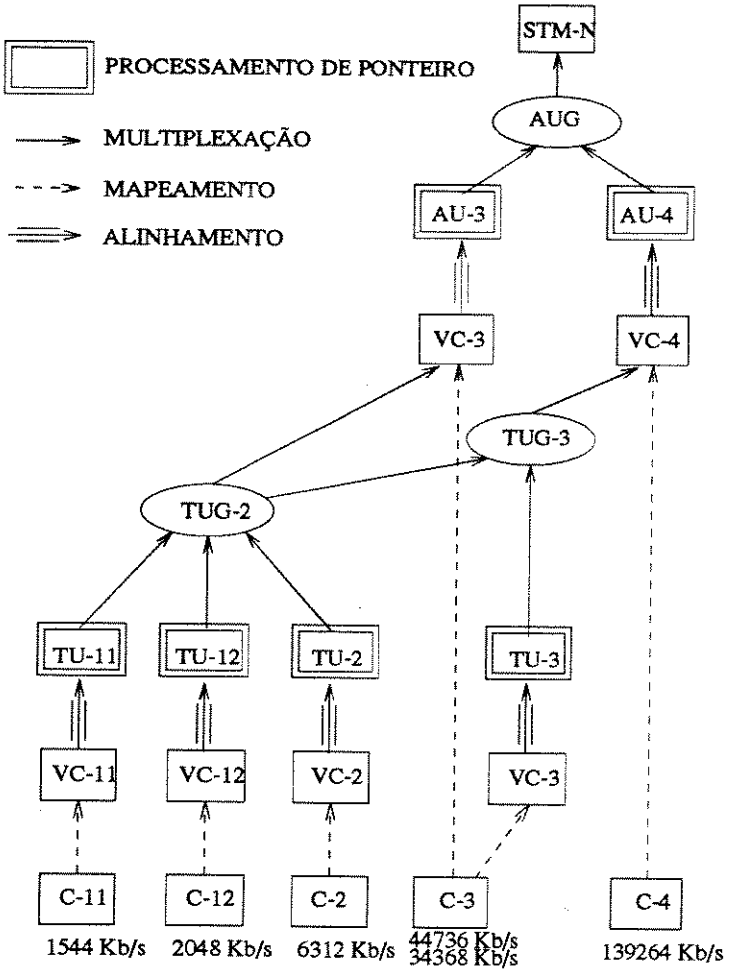


Figura 2.1: Estrutura de Multiplexação HDS.

Cada quadro na figura 2.1 simboliza uma sequência de bytes que é transmitida/processada uma vez a cada 125 μs [1]. O quadro que é realmente transmitido pela rede digital síncrona é o STM-N(Synchronous Transport Module), os demais quadros são resultado de processamentos realizados nos equipamentos HDS. Estes processamentos se referem as três operações listadas acima.

Legenda para a figura 2.1:

- STM - Módulo de Transporte Síncrono ("Synchronous Transport Module")
- VC - Container Virtual ("Virtual Container")
- AU - Unidade Administrativa ("Administrative Unit")
- TU - Unidade Afluente ("Tributary Unit")
- C - Container

[1] VC-1 e VC-2 são processados uma vez a cada 500μs

2.2 A Operação de Multiplexação

Multiplexação na Hierarquia Digital Síncrona é o processo pelo qual múltiplos quadros de menor ordem são adaptados para formar um quadro de maior ordem.

2.2.1 Multiplexação de AUs dentro do STM-N

Multiplexação de AUGs dentro do STM-N

A multiplexação de N AUGs dentro de um STM-N é ilustrada na figura 2.2. O AUG é uma estrutura de 9 linhas por 261 colunas de bytes, mais 9 bytes colocados na quarta linha(ponteiro(s) do(s) AU(s)). O STM-N é composto por SOH (Supervisão de Seção) mais uma estrutura de 9 linhas por N×261 colunas de bytes com N×9 bytes na quarta linha(ponteiros do AU). As N AUGs são entrelaçadas byte a byte e têm fase fixa em relação ao STM-N.

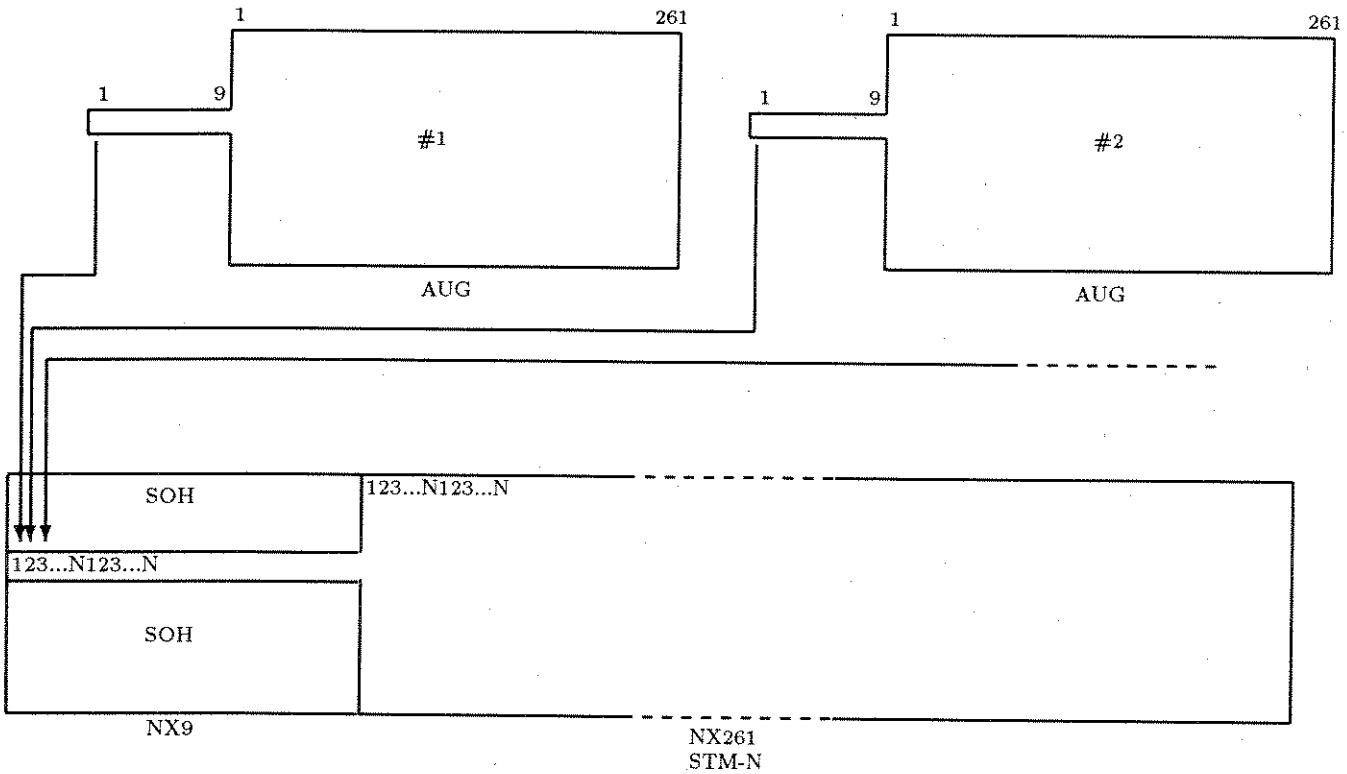


Figura 2.2: Multiplexação de N AUGs dentro de um STM-N.

Multiplexação de AU-4s via AUG

A multiplexação de uma única AU-4 via AUG é ilustrada na figura 2.3. Os 9 bytes no começo da quarta linha são alocados para o ponteiro do AU-4. Os bytes restantes (9 linhas por 261 colunas)

são alocados para o VC-4. A fase do VC-4 não é fixa em relação ao AU-4. A localização do primeiro byte do VC-4 em relação ao ponteiro do AU-4 é dada pelo valor do ponteiro. O AU-4 é colocado diretamente no AUG.

Multiplexação de AU-3s via AUG

A multiplexação de três AU-3 via AUG é ilustrada na figura 2.4. Os 3 bytes no começo da quarta linha são alocados para o ponteiro do AU-3. Os bytes restantes (9 linhas por 87 colunas) são alocados para o VC-3 mais 2 colunas de enchimento ("stuffing"). A fase do VC-3 mais as 2 colunas de "stuffing" não é fixa em relação ao AU-3. A localização do primeiro byte do VC-3 em relação ao ponteiro do AU-3 é dada pelo valor do ponteiro. As 3 AU-3 são entrelaçadas byte a byte para formar o AUG.

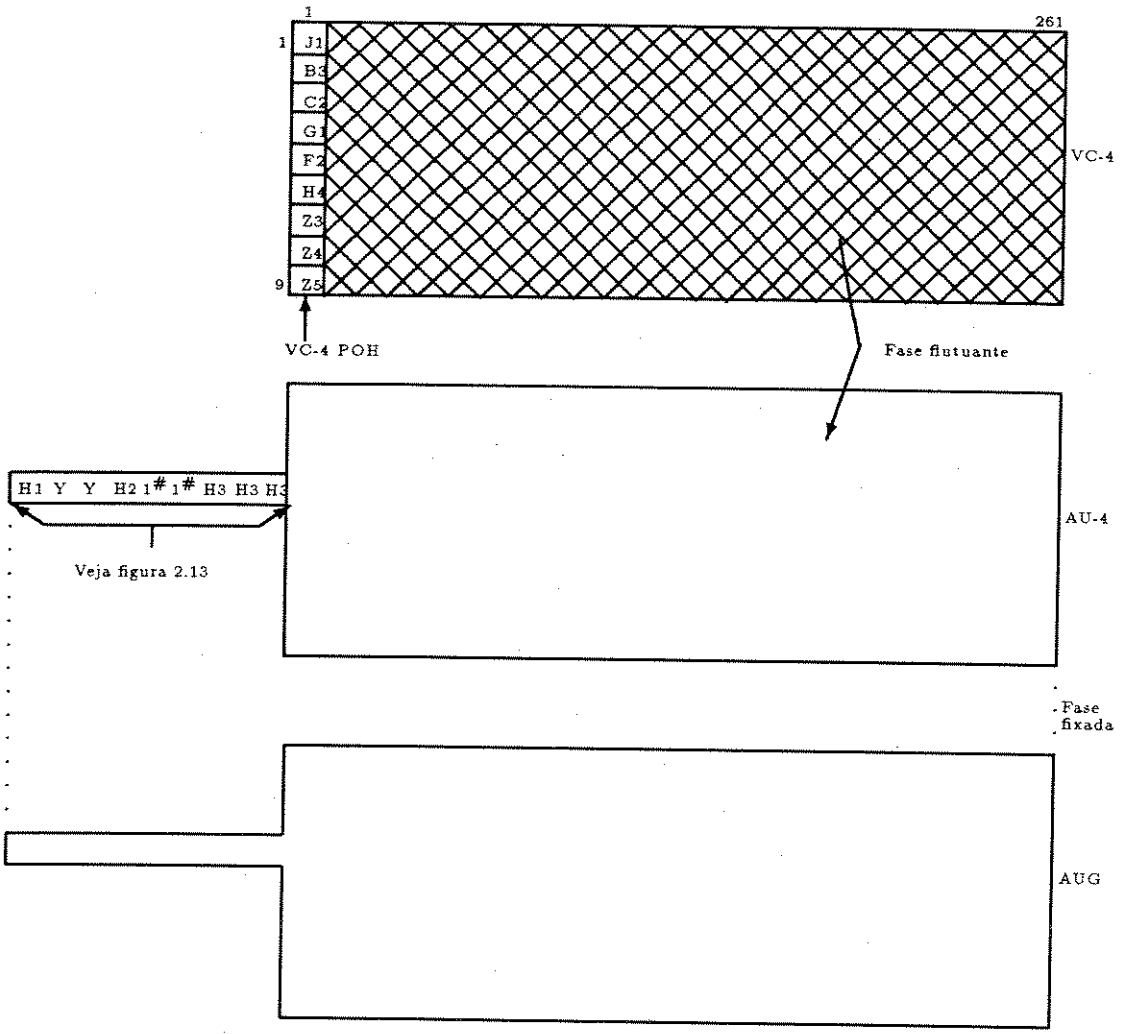


Figura 2.3: Multiplexação de AU-4s via AUG.

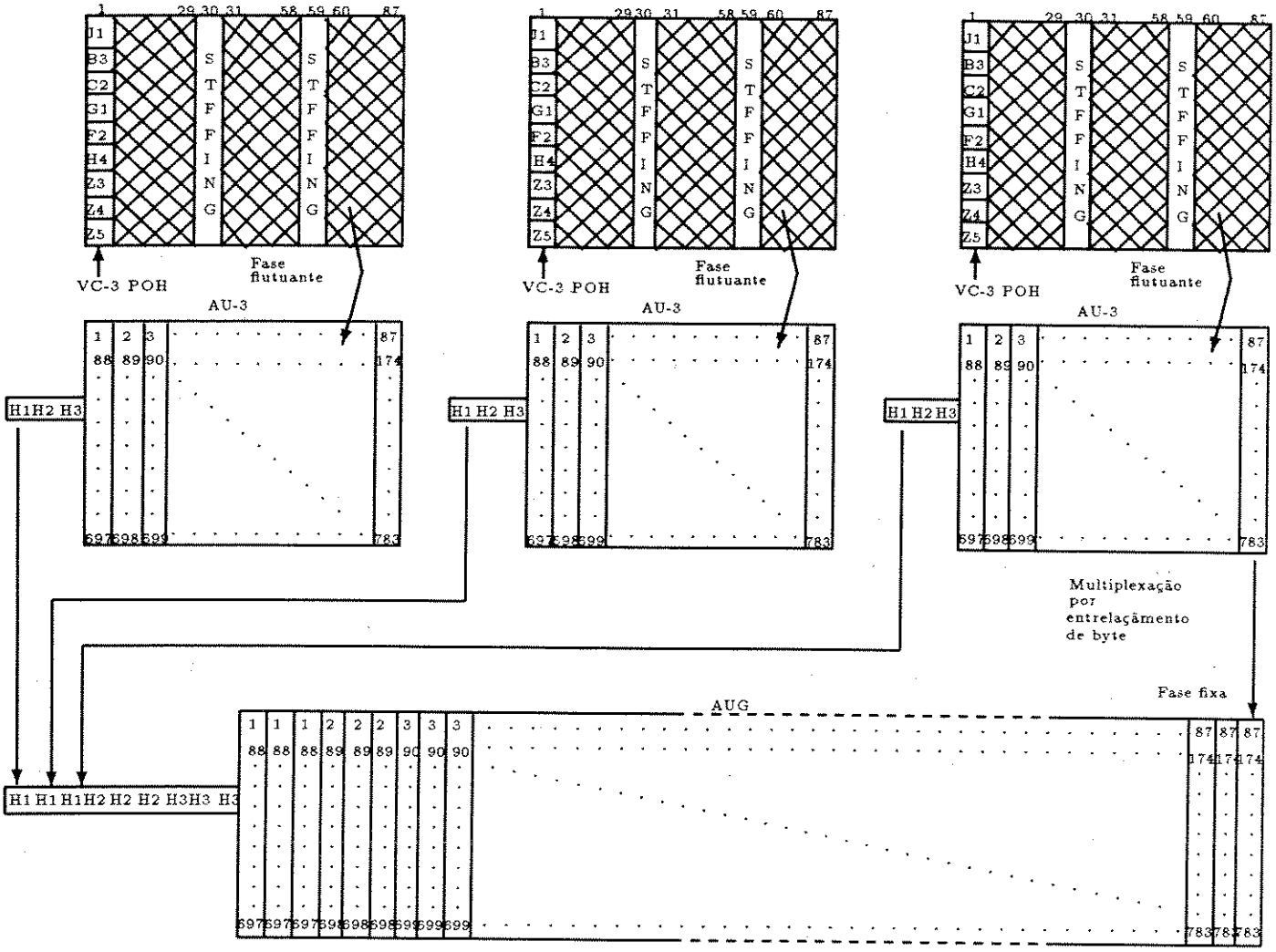


Figura 2.4: Multiplexação de AU-3s via AUG.

2.2.2 Multiplexação de TUs dentro do VC-3 e VC-4

Multiplexação de TUG-3s dentro do VC-4

A multiplexação de três TUG-3s dentro de um VC-4 é ilustrada na figura 2.5. O TUG-3 é uma estrutura de 9 linhas por 86 colunas de bytes. O VC-4 é composto por uma coluna de POH (Supervisão de Rota), duas colunas de "stuffing" mais 258 colunas de carga útil. Os 3 TUG-3s são entrelaçados byte a byte para formar a carga útil do VC-4 e têm fase fixa em relação ao VC-4. Como descrito anteriormente, a fase do VC-4 em relação ao AU-4 é dada pelo ponteiro do AU-4.

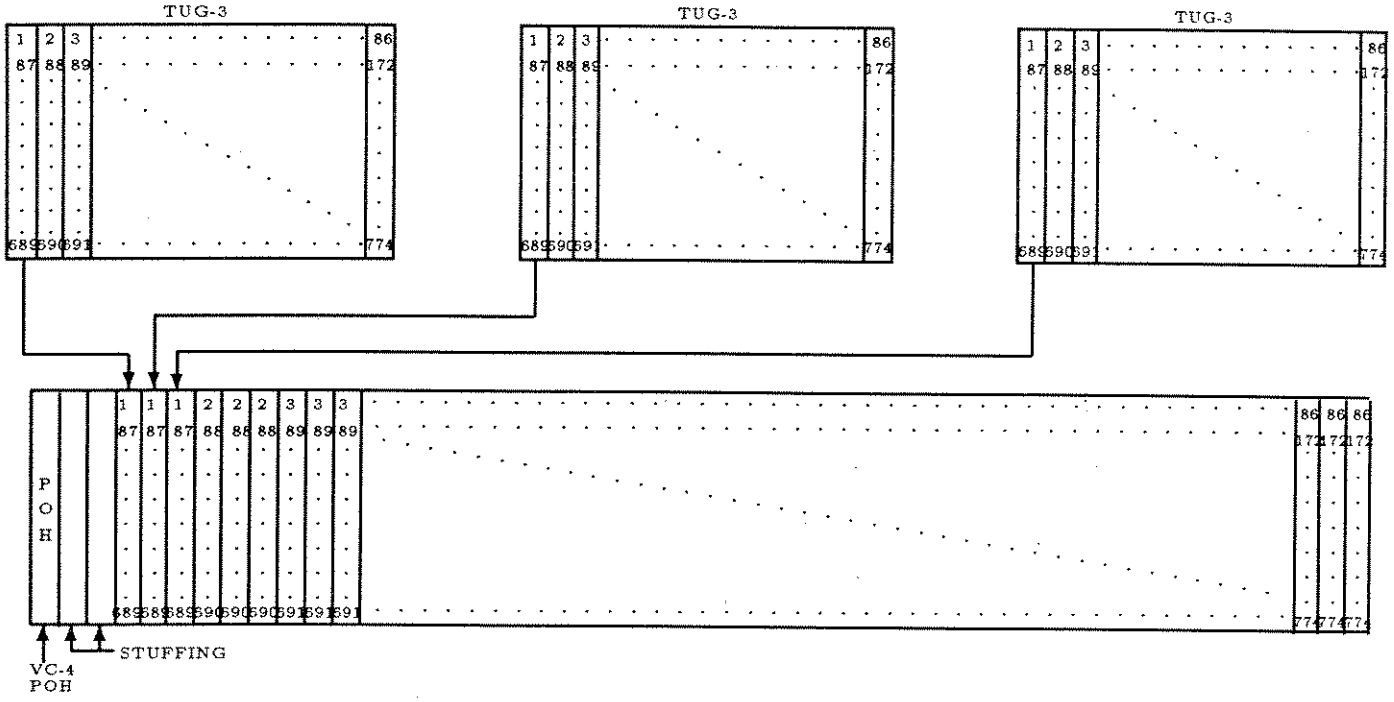


Figura 2.5: Multiplexação de 3 TUG-3s dentro do VC-4.

Multiplexação de TU-3s via TUG-3

A multiplexação de um único TU-3 via TUG-3 é ilustrada na figura 2.6. O TU-3 é composto pelo VC-3 com seus 9 bytes de POH mais o ponteiro do TU-3. A primeira coluna do quadro de 9 linhas por 86 colunas é alocada para o ponteiro do TU-3(bytes H1, H2 e H3) mais 6 bytes de "stuffing". A fase do VC-3 em relação ao TUG-3 é indicada pelo ponteiro do TU-3.

Multiplexação de TUG-2s via TUG-3

A multiplexação de 7 TUG-2s via TUG-3 é ilustrada na figura 2.7. Os 7 TUG-2s são entrelaçados byte a byte para formar as ultimas 84 colunas de um TUG-3.

O TUG-3 é uma estrutura de 9 linhas por 86 colunas sendo que as duas primeiras colunas acomodam o seguinte:

- Um indicador de ponteiro nulo(NPI) contido nos dois primeiros bytes da primeira coluna. Este NPI pode ser usado para distinguir TUG-3s contendo TU-3s de TUG-3s contendo TUG-2s.
- A segunda coluna tem seus bytes preenchidos com "stuffing".

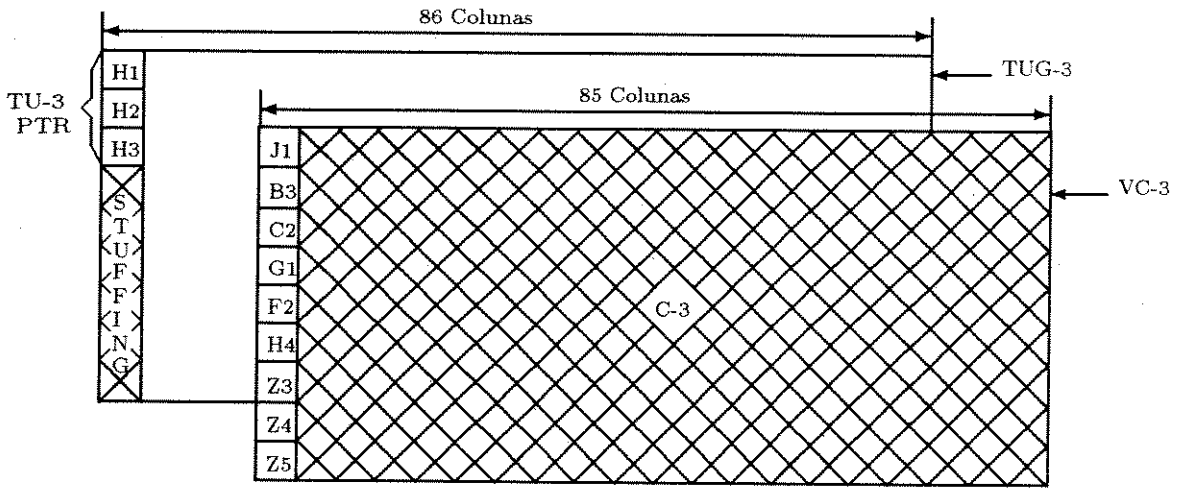


Figura 2.6: Multiplexação de um TU-3 via um TUG-3.

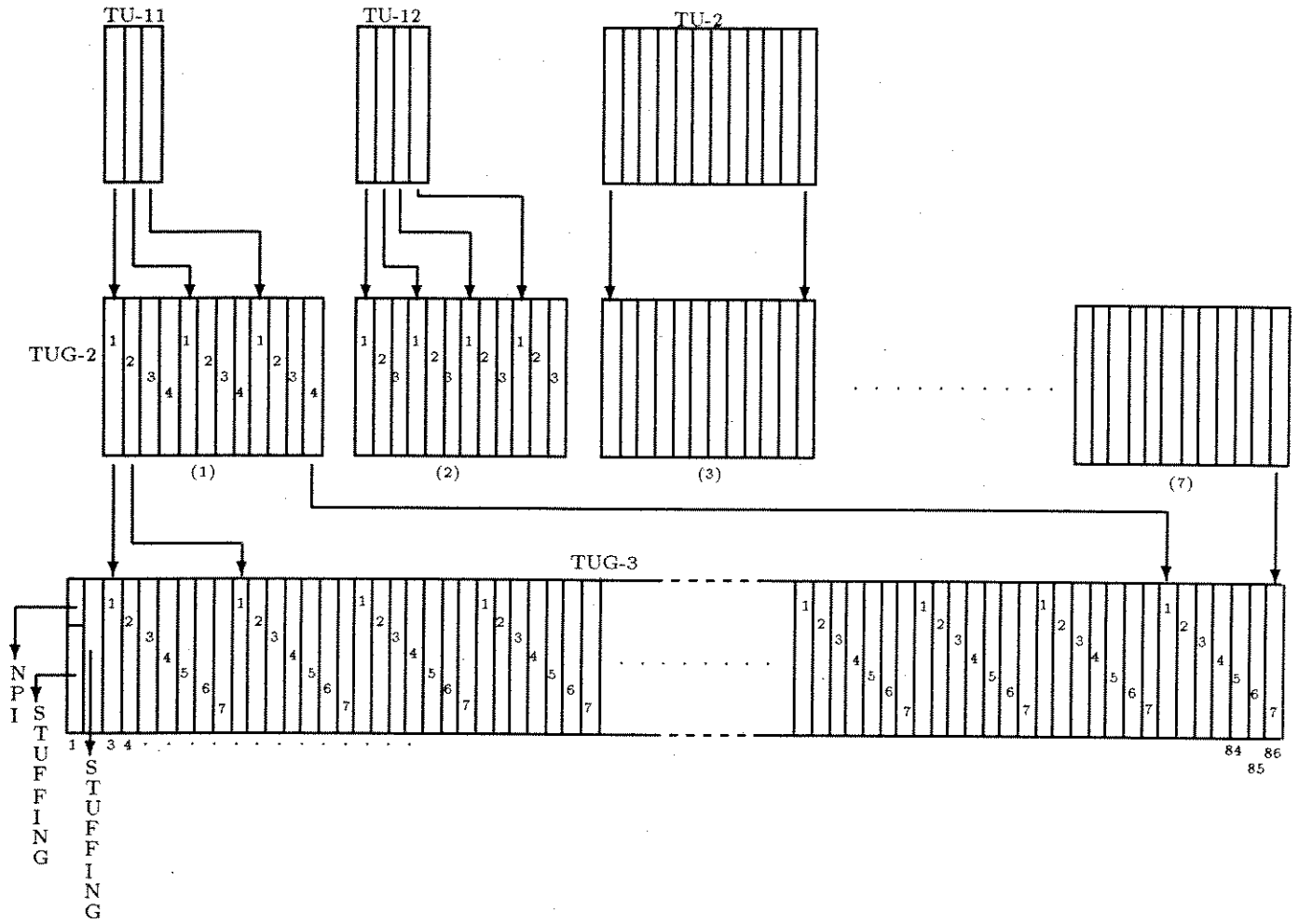


Figura 2.7: Multiplexação de sete TUG-2s via um TUG-3.

Multiplexação de TUG-2s via VC-3

A multiplexação de 7 TUG-2s via TUG-3 é ilustrada na figura 2.8. Os 7 TUG-2s são entrelaçados byte a byte para formar as ultimas 84 colunas de um VC-3. Um TUG-2 individual tem fase fixa em relação ao VC-3.

O VC-3 é composto por uma coluna de POH mais 84 colunas de carga útil.

Multiplexação de TU-2s via TUG-2

A multiplexação de um único TU-2 via TUG-2 é ilustrada na figura 2.8. O TU-2 é colocado diretamente no TUG-2.

Multiplexação de TU-1s via TUG-2

A multiplexação de quatro TU-11 ou três TU-12 é ilustrada na figura 2.8. Os TU-1s são entrelaçados byte a byte para formar o TUG-2.

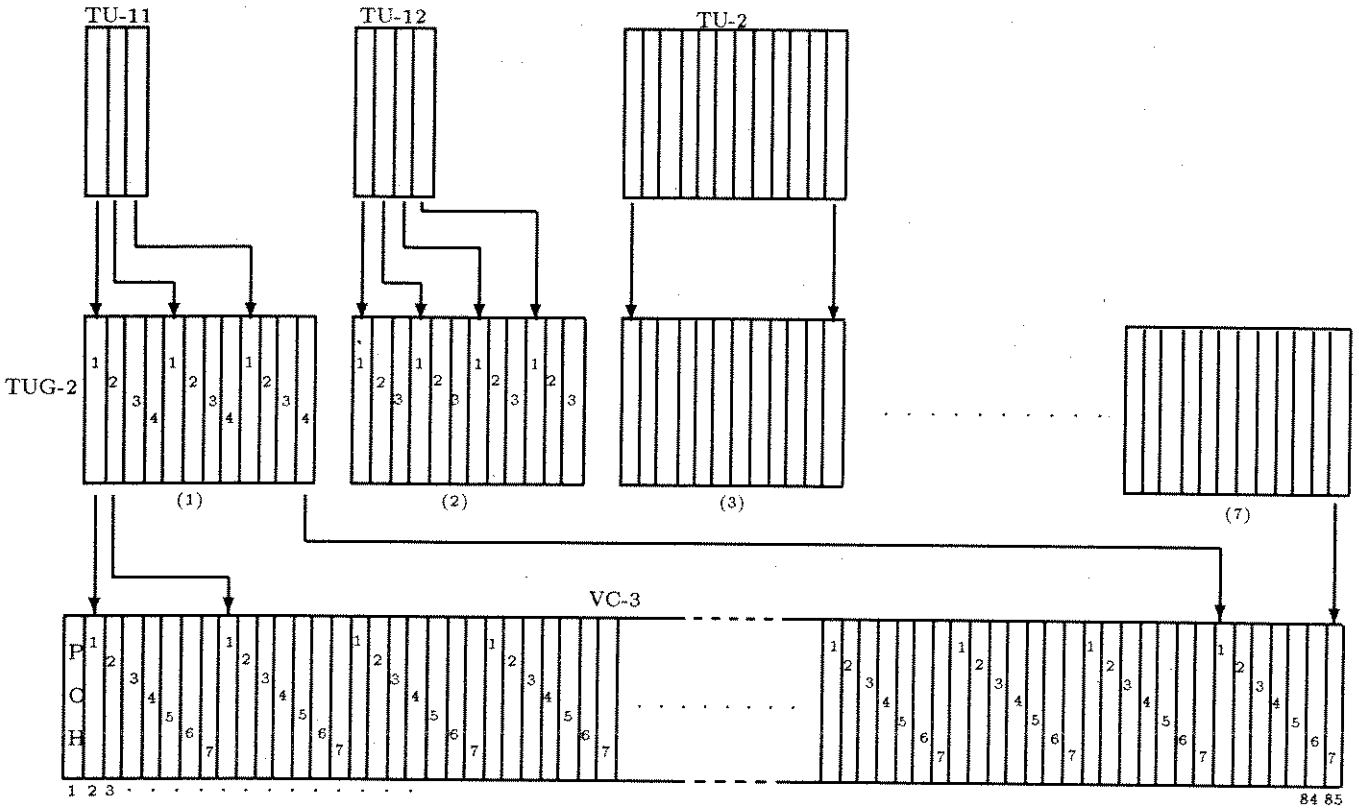


Figura 2.8: Multiplexação de sete TUG-2s dentro um VC-3.

2.3 A Operação de Alinhamento

Na Hierarquia Digital Síncrona, alinhamento é o processo pelo qual a informação relativa a diferença de fase entre um quadro de suporte de rota (VC-nm) e outro quadro de adaptação (TU-n ou AU-m), é processada e colocada em bytes especificos deste ultimo quadro. Estes bytes compoem o *Ponteiro* do quadro de adaptação que provê alinhamento flexivel e dinâmico de um VC em relação a um AU/TU. Flexivel no sentido de o VC poder "flutuar" acomodando diferenças de fase e dinâmico por permitir diferemças de frequência entre os quadros.

2.3.1 Os Ponteiros dos Quadros AU-3/4 e TU-3

Alocação dos Ponteiros

Os quadros AU-4, AU-3 e TU-3 reservam três de seus bytes para alocação de seus ponteiros. Estes bytes são H1,H2 e H3 como mostrado nas figuras 2.9, 2.10e 2.11, para AU-4, AU-3 e TU-3 respectivamente.

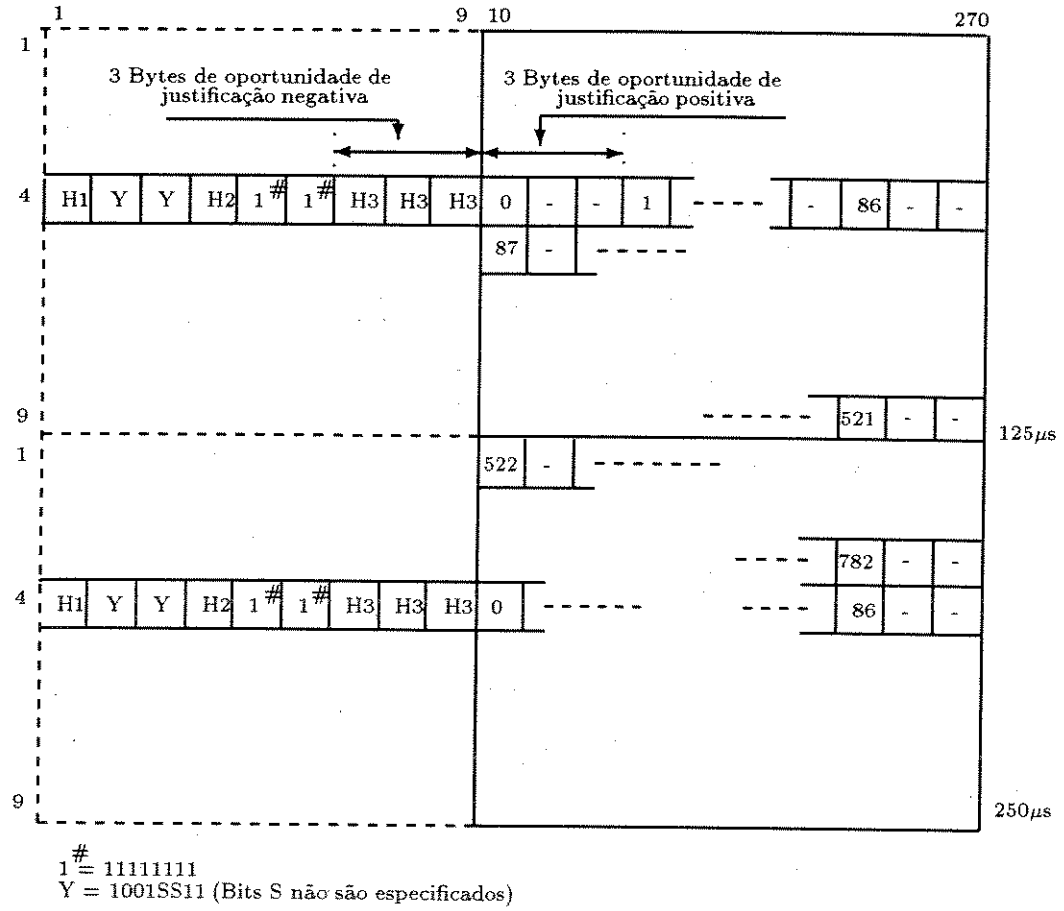


Figura 2.9: Alocação do Ponteiro no Quadro AU-4

Os Conteúdos dos Ponteiros

Os bytes H1 e H2 definem a posição do primeiro byte do VC contido no quadro AU/TU-3. Estes dois bytes formam a palavra mostrada na figura 2.12.

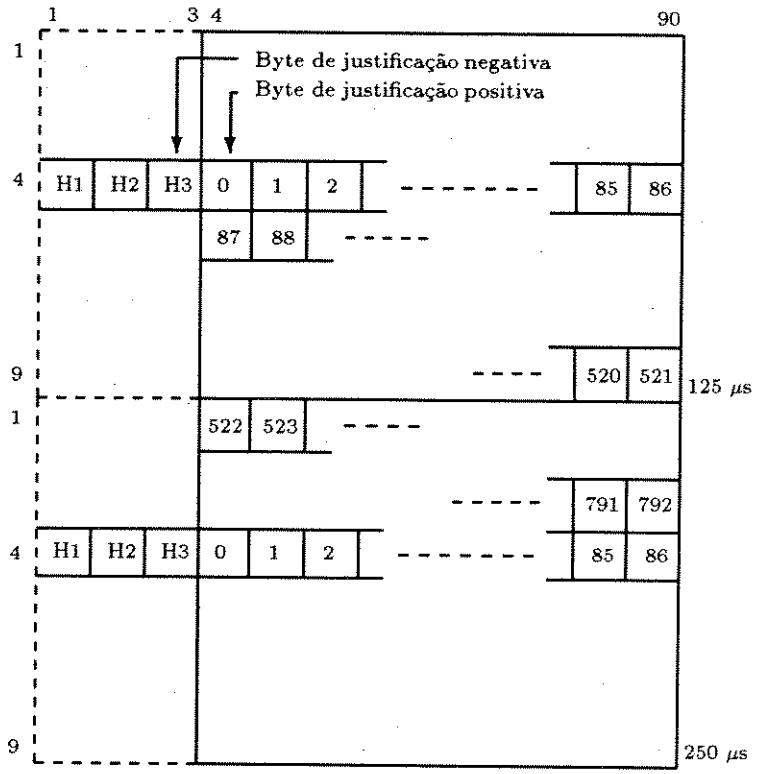


Figura 2.10: Alocação do Ponteiro no Quadro AU-3

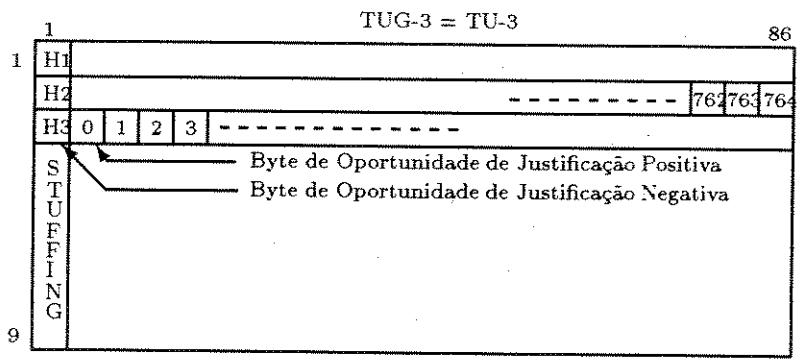
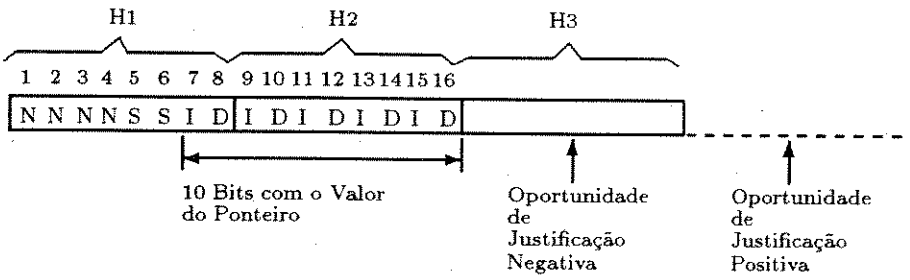


Figura 2.11: Alocação do Ponteiro no Quadro TU-3



I - Bit de Incremento
D - Bit de Decremento
N - Sinalização de Novos Dados(NDF)

Novo Valor de Ponteiro(NDF)

- Abilitado: 1001
- Desabilitado: 0110

Justificação Negativa

- Inverte os 5 bits D
- Aceita votação majoritária

Justificação Positiva

- Inverte os 5 bits I
- Aceita votação majoritária

Valor do Ponteiro(bits 7-16):

AU-4 e AU-3: 0 - 782 decimal
TU-3: 0 - 764 decimal

Indicador de Concatenação(CI):

H1H2 = 1001SS1111111111
(bits S não são especificados)

Indicador de Ponteiro Nulo(NPI):

H1H2 = 1001SS1111100000
(bits S não são especificados)

Valor de SS	Tipo de AU/TU
10	AU-4, AU-3, TU-3

Nota 1 - NPI se aplica somente a ponteiros de TU-3.

Nota 2 - Todos os bits do ponteiro são "1" quando ocorre AIS.

Figura 2.12: Conteúdos dos Ponteiros dos Quadros AU/TU-3

Os ultimos 10 bits (7 - 16) da palavra H1 + H2 carregam o valor do ponteiro. Como mostrado na figura 2.12, o valor do ponteiro é um número binário contido nas seguintes faixas:

- 0 a 792 para AU-4 e AU-3
- 0 a 764 para TU-3

Este número indica o deslocamento entre o ponteiro e o primeiro byte do VC. Para AU-3 e TU-3 este deslocamento é contado byte a byte, ou seja, se o ponteiro do quadro AU-3 contém o número 10 por exemplo, então entre o byte H3 e o primeiro byte do VC-3 relacionado a este AU-3 existem 10 bytes. O mesmo se passa com o TU-3.

Para o AU-4 o deslocamento é contado de três em três bytes, ou seja, se o seu ponteiro contém por exemplo o número 10, então existem 30 bytes entre o terceiro byte H3 e o primeiro byte do VC-4 relacionado a este AU-4.

A figura 2.12 também indica um valor de ponteiro adicional, o indicador de concatenação(CI). Quadros AU-4s podem ser concatenados para juntos formarem um quadro AU-4-Xc capaz de transportar cargas úteis que requerem capacidades maiores que a de um C-4. Os quadros VC-4s contidos em um AU-4-Xc formam uma estrutura chamada VC-4-Xc na qual a primeira coluna é usada para POH e todas as demais colunas estão disponiveis para a carga útil. O indicador de concatenação(CI), usado para indicar que a integridade desta carga útil multipla de C-4 carregada em um unico VC-4-Xc deve ser mantida, esta contido no ponteiro de cada AU-4 usado para formar o AU-4-Xc, com exceção do primeiro AU-4. O primeiro AU-4 do AU-4-Xc deve ter operação normal de ponteiro. Todos os demais AU-4s contidos no AU-4-Xc devem ter seus ponteiros assim:

- Bits de 1 a 4 : CI = 1001.
- Bits 5 e 6 : não especificados.
- Bits 7 a 16 : dez bits "1"s.

O CI determina que o processador de ponteiro deve realizar as mesmas operaçoês realizadas sobre o primeiro AU-4 do AU-4-Xc.

Quando quadros TUG-2s são multiplexados dentro de um quadro VC-3, o ponteiro do quadro TU-3 é preenchido com indicador de ponteiro nulo(NPI). Isto se dá porque os quadros TUG-2s já estão previamente sincronizados com o quadro TU-3. NPI é indicado por:

- Bits 1-4 : 1001.
- Bits 5-6 : não especificados.
- Bits 7-11 : 11111.
- Bits 12-16 : 00000.

A figura 2.12 também indica o valor de ponteiro para NPI.

Justificação de Frequência

Se existe uma diferença de frequência entre o quadro AU/TU-3 e o quadro VC correspondente, então o valor do ponteiro será incrementado ou decrementado conforme a necessidade, seguido de uma justificação(positiva ou negativa) de byte ou bytes(no caso do AU-4 a justificação se dá de 3 em 3 bytes). Operaçoẽs consecutivas de ponteiros devem ser separadas de no mínimo 3 quadros nos quais o valor do ponteiro permanece inalterado.

Se a taxa do VC é menor que a do AU/TU-3, então periodicamente o alinhamento do VC deve ser atrasado no tempo e o valor do ponteiro deve ser incrementado de um. Esta operação é indicada invertendo-se os bits 7, 9, 11, 13 e 15 (bits Is) do ponteiro. Para superar erros de transmissão, é tomada decisão majoritária nos cinco bits Is. No caso do AU-4, três bytes de justificação positiva aparecem imediatamente após o ultimo byte H3 do ponteiro contendo bits Is invertidos. No caso do AU-3/TU-3, um byte de justificação positiva aparece imediatamente após o byte H3 do ponteiro contendo bits Is invertidos. Ponteiros subsequentes irão conter o novo deslocamento. Este processo é ilustrado na figura 2.13 para AU-4 e na figura 2.14 para AU-3.

Se a taxa do VC é maior que a do AU/TU-3, então periodicamente o alinhamento do VC deve ser adiantado no tempo e o valor do ponteiro deve ser decrementado de um. Esta operação é indicada invertendo-se os bits 8, 10, 12, 14 e 16 (bits Ds) do ponteiro. Para superar erros de transmissão, é tomada decisão majoritária nos cinco bits Ds. No caso do AU-4, três bytes de justificação negativa aparecem no bytes H3s do ponteiro contendo bits Ds invertidos. No caso do AU-3/TU-3, um byte de justificação positiva aparece no byte H3 do ponteiro contendo bits Ds invertidos. Ponteiros subsequentes irão conter o novo deslocamento. Este processo é ilustrado na figura 2.15 para AU-4 e na figura 2.16 para AU-3.

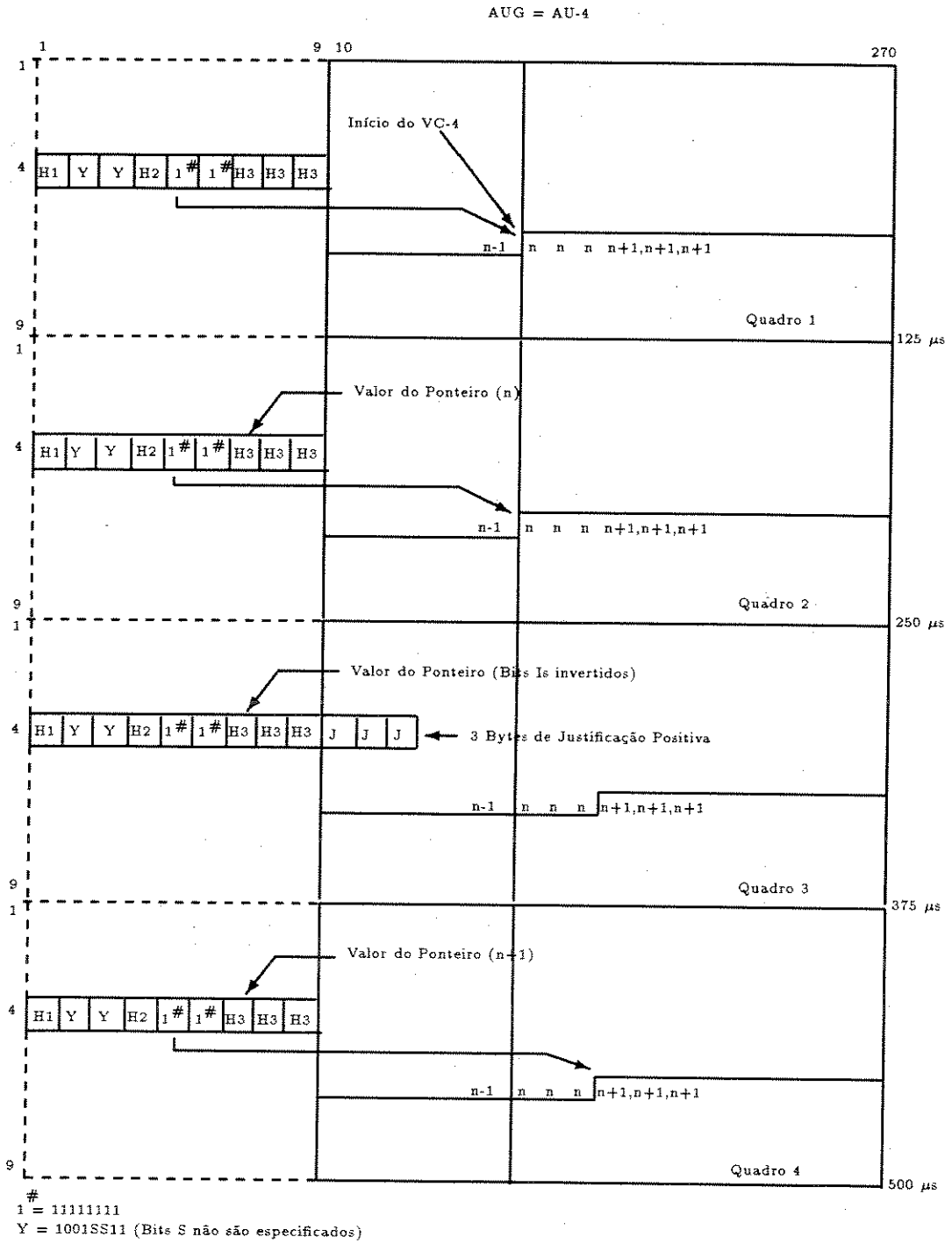


Figura 2.13: Operação de ajustamento do ponteiro do AU-4 — Justificação Positiva

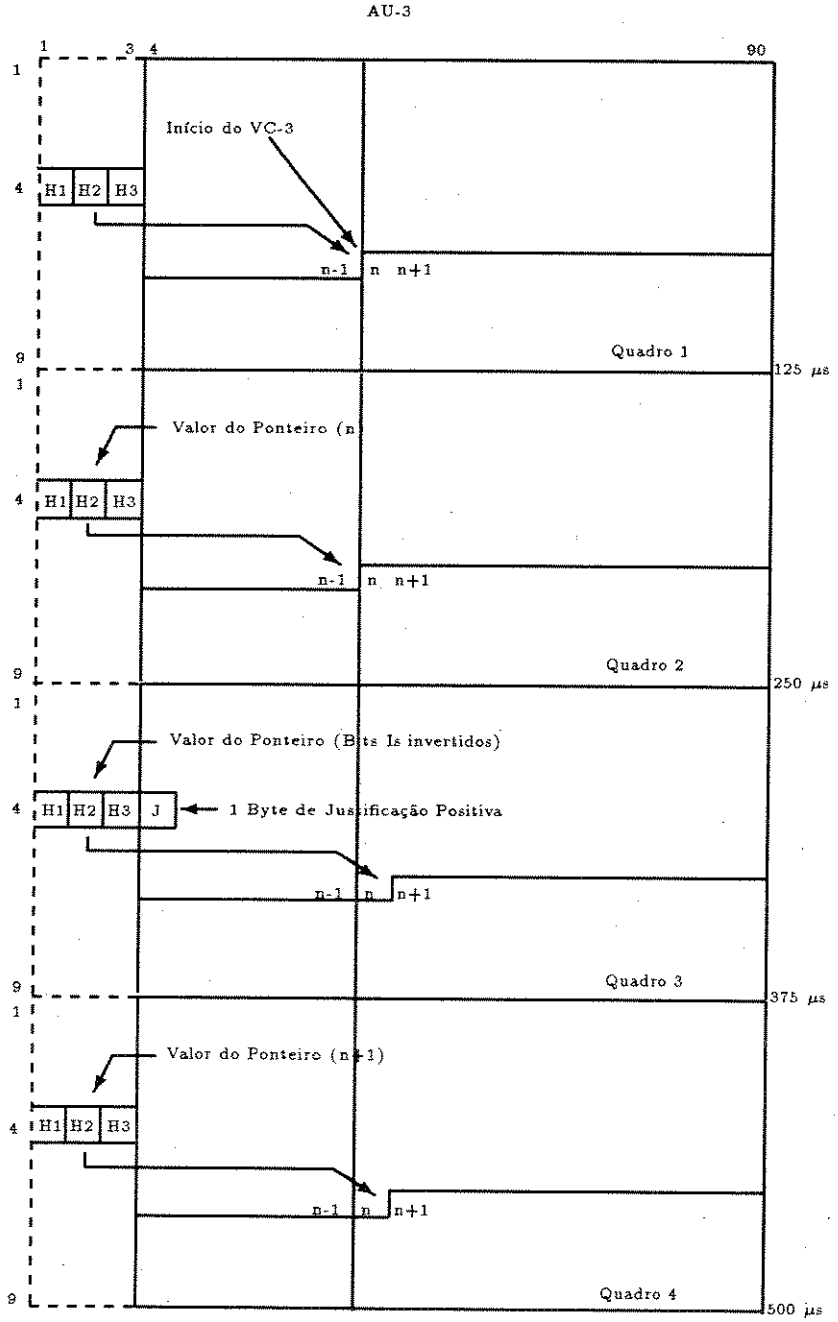


Figura 2.14: Operação de ajustamento do ponteiro do AU-3 — Justificação Positiva

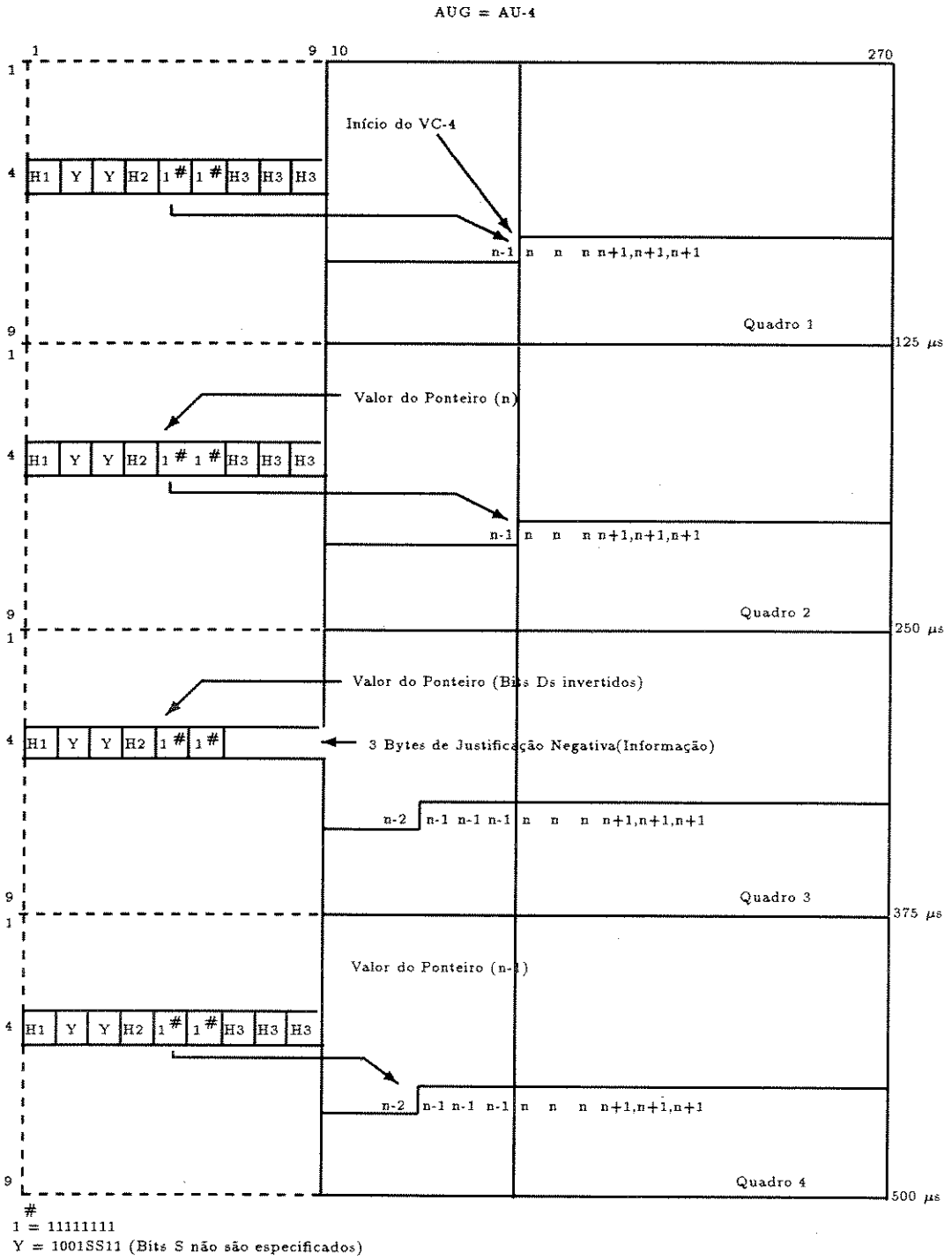


Figura 2.15: Operação de ajustamento do ponteiro do AU-4 — Justificação Negativa

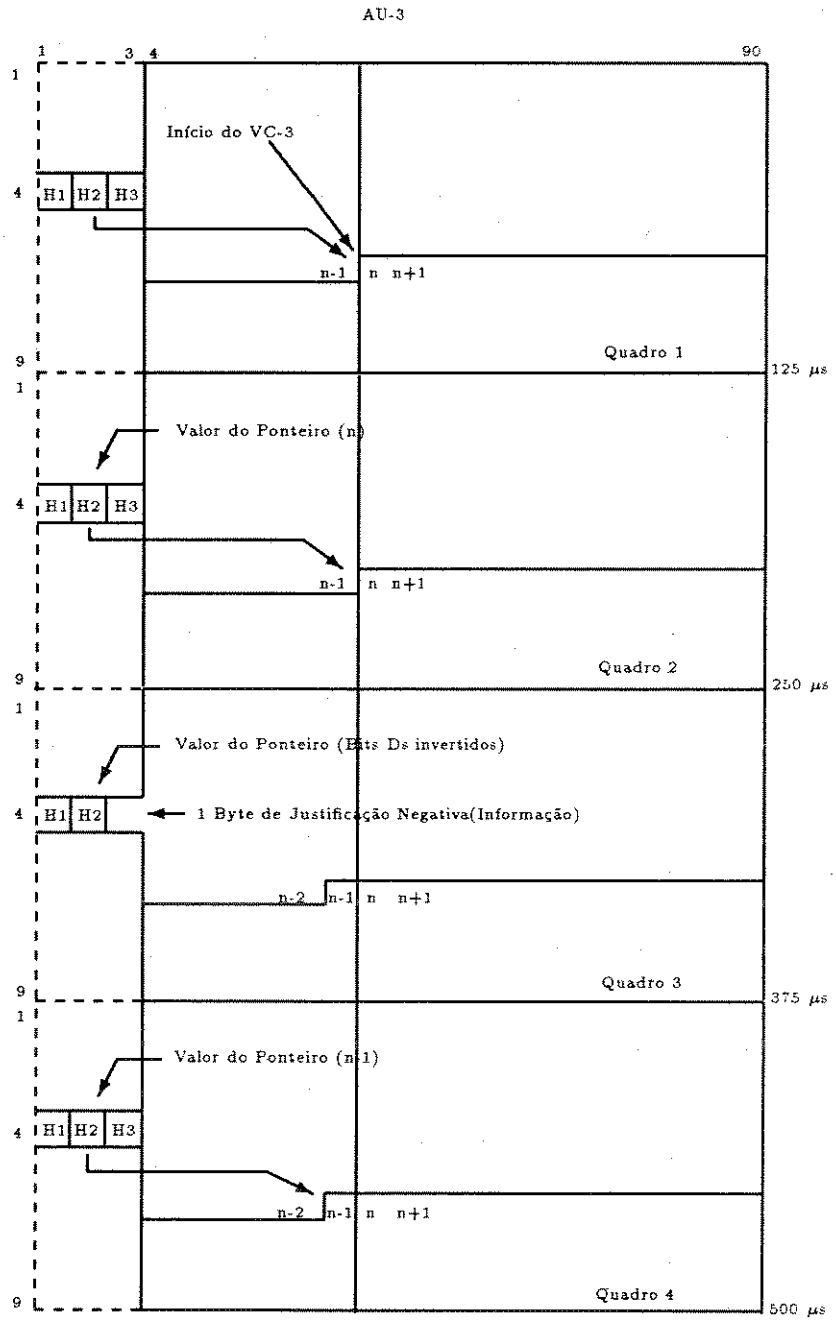


Figura 2.16: Operação de ajustamento do ponteiro do AU-3 — Justificação Negativa

2.3.2 Os Ponteiros dos Quadros TU-1 e TU-2

Diferentemente dos ponteiros dos quadros AU-4, AU-3 e TU-3 onde o processamento é feito uma vez a cada 125 μs, nos ponteiros dos quadros TU-1 e TU-2 o processamento é feito uma vez a cada 500 μs.

Alocação dos Ponteiros

Os quadros TU-1 e TU-2 reservam três de seus bytes para alocação de seus ponteiros. Estes bytes são V1, V2 e V3 como mostrado na figura 2.17.

Os Conteúdos dos Ponteiros

Os bytes V1 e V2 designam a localização do primeiro byte do VC contido no quadro AU/TU-3. Estes dois bytes formam a palavra mostrada na figura 2.18.

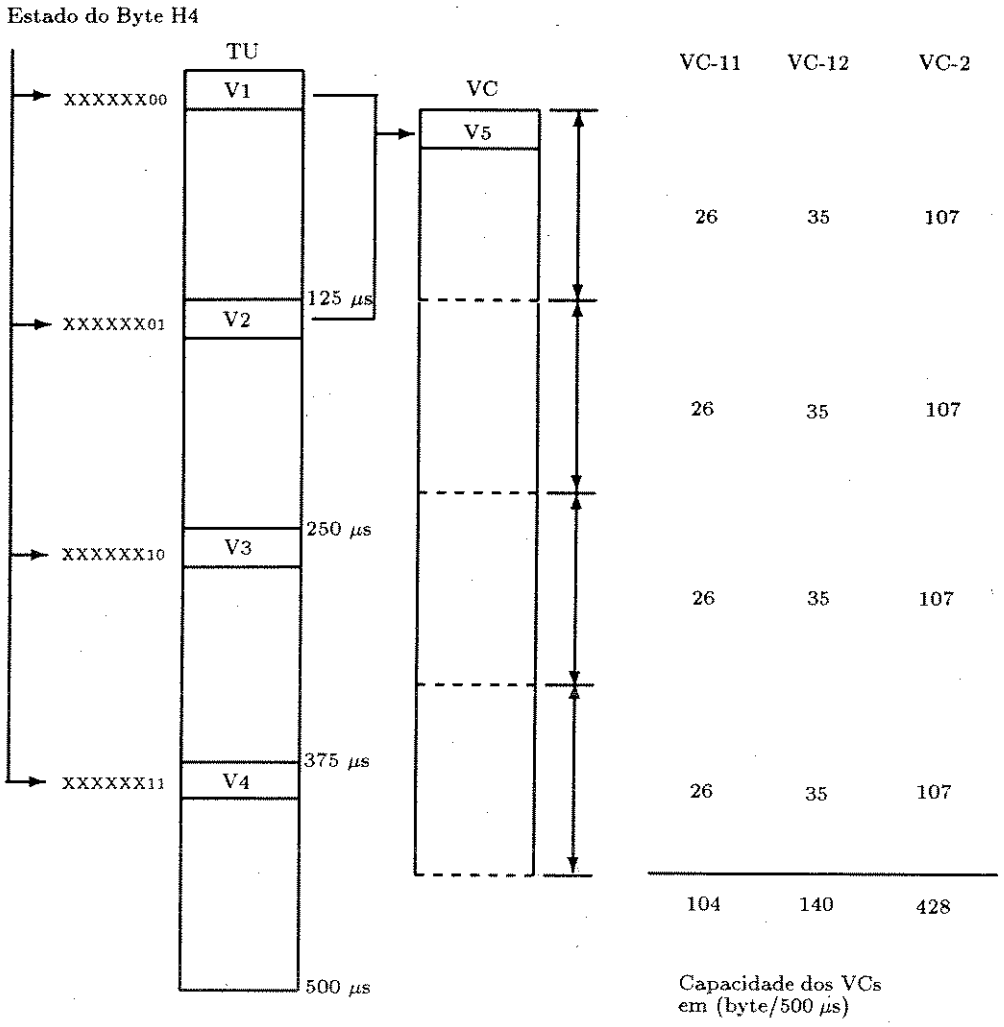


Figura 2.17: Mapeamento de VCs em Estruturas TUs de Super Quadro.

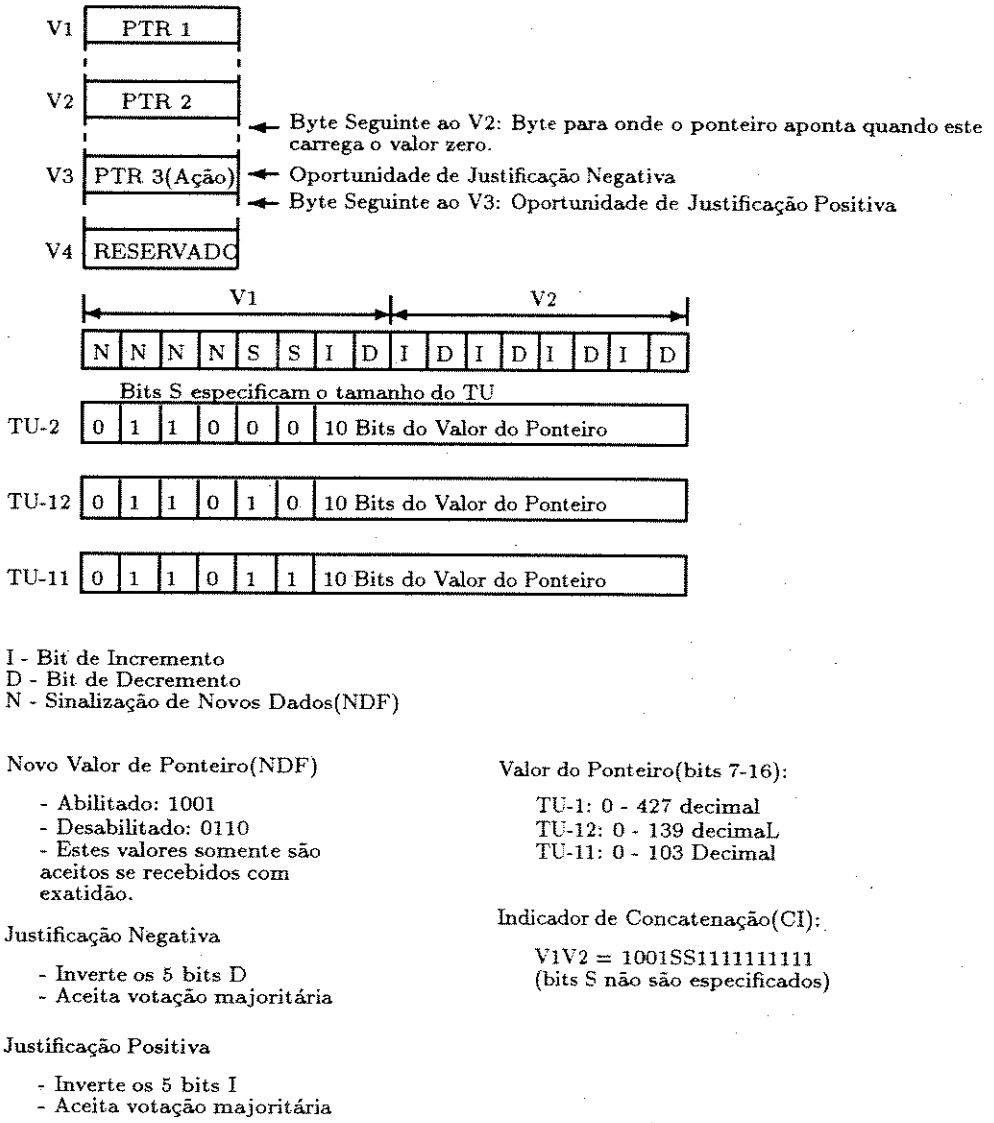


Figura 2.18: Conteúdos dos Ponteiros dos Quadros TU-1 e TU-2.

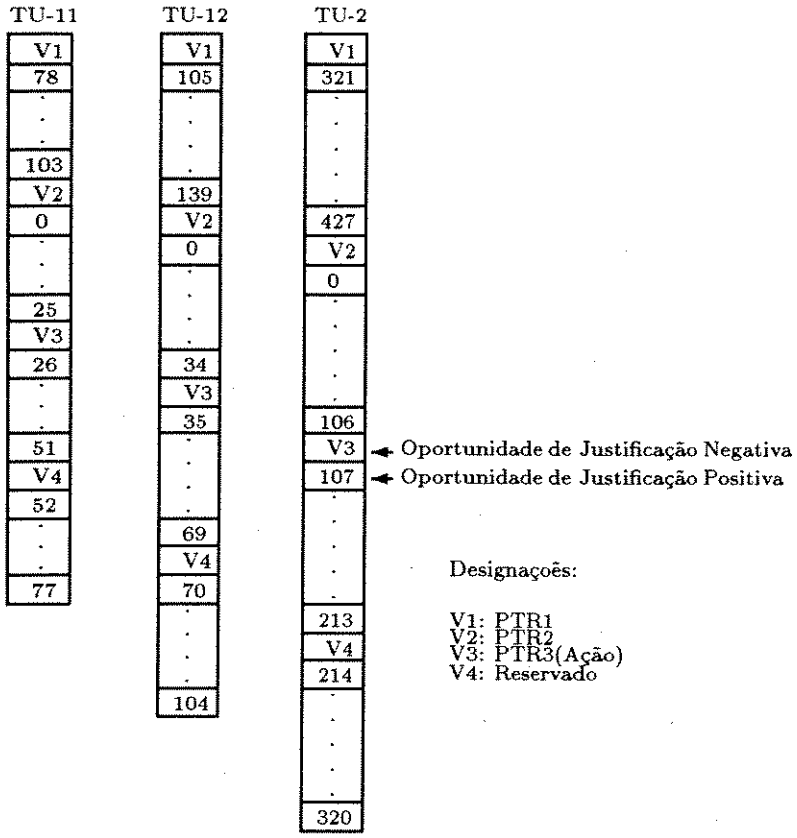


Figura 2.19: Alocação dos Bytes V1, V2, V3 e V4 do Quadro TU-1/TU-2

Os últimos 10 bits (7 - 16) da palavra V1 + V2 carregam o valor do ponteiro. Como mostrado nas figuras 2.18 e 2.19 , o valor do ponteiro é um número binário contido nas seguintes faixas:

- 0 a 427 para TU-2
- 0 a 139 para TU-12
- 0 a 103 para TU-11

Este número indica o deslocamento entre o ponteiro e o primeiro byte do VC-1/VC-2. Este deslocamento é contado byte a byte, ou seja, se o ponteiro do quadro TU-1/TU-2 contém o número 10 por exemplo, então entre o byte V2 e o primeiro byte do VC relacionado a este TU existem 10 bytes. Os bytes V1, V2, V3 e V4 não devem ser contados para a composição do valor do ponteiro. Os dois bits S(Bits 5 e 6 da palavra V1 + V2) indicam o tipo de TU, assumindo os valores:

- 00 para TU-2
- 10 para TU-12
- 11 para TU-11

O Byte Indicador de Super Quadro

Multiplos quadros TU-1/TU-2 podem ser processados conjuntamente para formar uma de três estruturas de superquadro existentes. Estas estruturas são multiplexadas dentro de um VC-3 ou VC-4 e devem ser identificadas por um byte indicador de superquadro(H4) que faz parte do POH do VC-3/VC-4. A figura 2.20 ilustra o caso em que o byte H4 é usado para indicar a estrutura de superquadro composta por 4 quadros.

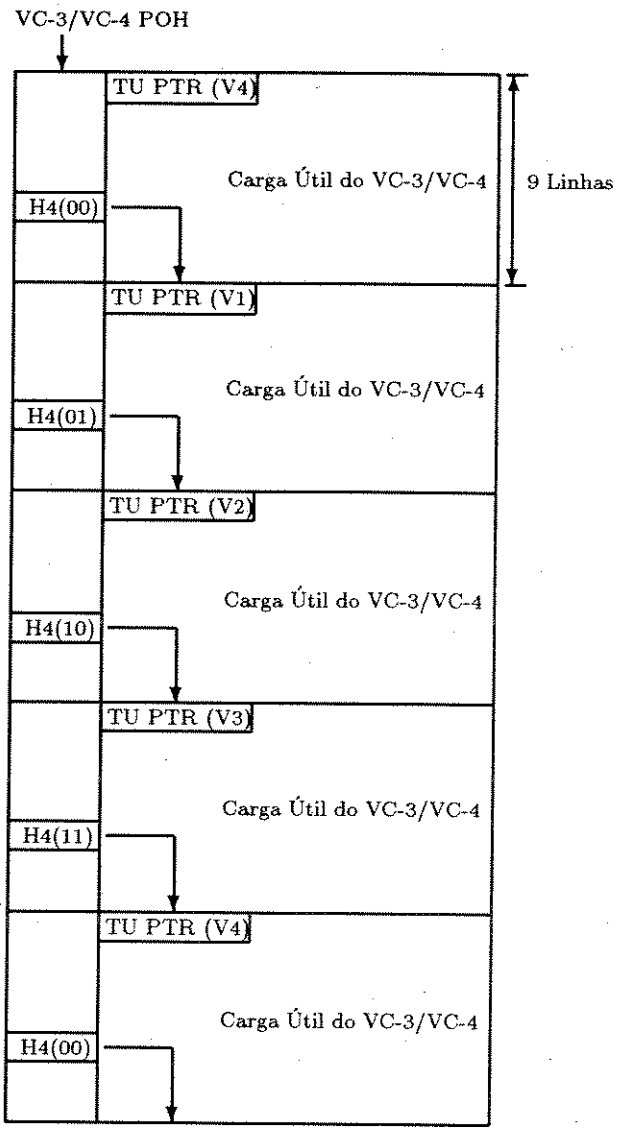


Figura 2.20: Um exemplo de um indicador de superquadro para TU-1/TU-2

As estruturas de superquadro podem ser formadas por 4, 16 ou 24 quadros de 125 μs, formando estruturas de superquadro de 500 μs, 2 ms e 3 ms respectivamente. A figura 2.21 mostra os bits de H4 que são utilizados para indicar cada um dos três modos de operação.

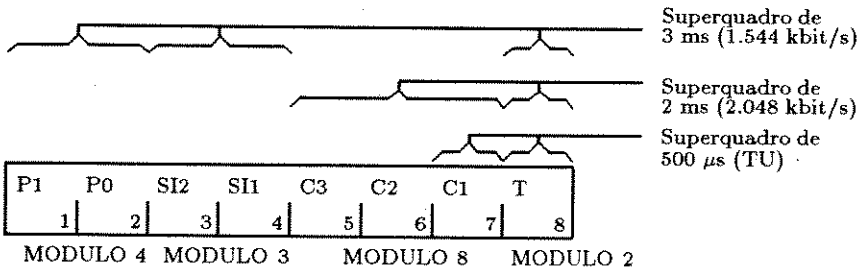


Figura 2.21: O Byte Indicador de Superquadro (H4)

A figura 2.22 ilustra uma sequência completa para o byte H4. A figura 2.20 ilustra o modo de operação para a estrutura de 4 quadros(500 μs).

Sequência completa do código do byte H4:
Obrigatório no modo TU amarrado
Opcional no modo TU flutuante

BIT: 1 2	3 4	5 6 7	8	QUADROS	TEMPO
0 0	0 0	0 0 0	0	0	0
0 0	0 0	0 0 0	1	1	
0 0	0 1	0 0 1	0	2	
0 0	0 1	0 0 1	1	3	500 μs TU Superquadro
0 0	1 0	0 1 0	0	4	
0 0	1 0	0 1 0	1	5	
0 1	0 0	0 1 1	0	6	
0 1	0 0	0 1 1	1	7	
0 1	0 1	1 0 0	0	8	
0 1	0 1	1 0 0	1	9	
0 1	1 0	1 0 1	0	10	
0 1	1 0	1 0 1	1	11	
1 0	0 0	1 1 0	0	12	
1 0	0 0	1 1 0	1	13	
1 0	0 1	1 1 1	0	14	
1 0	0 1	1 1 1	1	15	2 ms 2.048 kbit/s(ciclo de sinalização)
1 0	1 0	0 0 0	0	16	
1 0	1 0	0 0 0	1	17	
1 1	0 0	0 0 1	0	18	
1 1	0 0	0 0 1	1	19	
1 1	0 1	0 1 0	0	20	
1 1	0 1	0 1 0	1	21	
1 1	1 0	0 1 1	0	22	
1 1	1 0	0 1 1	1	23	3 ms 1.544 kbit/s(ciclo de sinalização)
0 0	0 0	1 0 0	0	24	
0 0	0 0	1 0 0	1	25	
0 0	0 1	1 0 1	0	26	
0 0	0 1	1 0 1	1	27	
0 0	1 0	1 1 0	0	28	
0 0	1 0	1 1 0	1	29	
0 1	0 0	1 1 1	0	30	
0 1	0 0	1 1 1	1	31	
0 1	0 1	0 0 0	0	32	
0 1	0 1	0 0 0	1	33	
0 1	1 0	0 0 1	0	34	
0 1	1 0	0 0 1	1	35	
1 0	0 0	0 1 0	0	36	
1 0	0 0	0 1 0	1	37	
1 0	0 1	0 1 1	0	38	
1 0	0 1	0 1 1	1	39	
1 0	1 0	1 0 0	0	40	
1 0	1 0	1 0 0	1	41	
1 1	0 0	1 0 1	0	42	
1 1	0 0	1 0 1	1	43	
1 1	0 1	1 1 0	0	44	6 ms = tempo de repetição do ciclo
1 1	0 1	1 1 0	1	45	
1 1	1 0	1 1 1	0	46	
1 1	1 0	1 1 1	1	47	

Figura 2.22: Sequência completa do Indicador de Superquadro H4.

Justificação de Frequência

Os ponteiros dos quadros TU-1/TU-2 são usados para justificação de frequência do quadro VC-1/VC-2 seguindo as mesmas regras estabelecidas para o ponteiro do quadro TU-3 usado para justificação de frequência do quadro VC-3. Nesta analogia, os bytes V1, V2 e V3 do quadro TU-1/TU-2 se associam respectivamente aos bytes H1, H2 e H3 do quadro TU-3, exercendo funções análogas no processo de justificação de frequência. Todo o processo de justificação esta bem esclarecido nas figuras 2.18 e 2.19.

2.3.3 Sinalização de Novos Dados (NDF)

Os primeiros 4 bits(N bits) da palavra formada pelos bytes H1 e H2 (AU-4/AU-3/TU-3) ou V1 e V2 (TU-1/TU-2) carregam sinalização de novos dados(NDF) que permite uma mudança arbitraria no valor do ponteiro para comportar uma mudança de carga útil. No caso do AU-3/AU-4/TU-3 a decodificação de NDF pode ser aceita se no mínimo três dos quatro bits coincidirem com o código de NDF. Já no caso do TU-1/TU-2, os quatro bits devem coincidir exatamente com o código de NDF. Este código é o seguinte:

- 0110 para operação normal.
- 1001 para indicação de novo alinhamento.

O ponteiro seguinte à operação de NDF carrega o novo alinhamento que o VC deve tomar. Para o TU-1/TU-2 este ponteiro pode trazer também uma mudança no tamanho do VC. Quando isto ocorre, todos os ponteiros de TUs contidos em um TUG-2 devem indicar simultaneamente NDF com o mesmo tamanho de VC.

2.3.4 Geração e Interpretação do Ponteiro

Geração

1. Durante operação normal o ponteiro localiza o início do quadro VC dentro do quadro AU/TU. Os quatro bits de NDF são preenchidos com "0110".
2. O valor do ponteiro pode ser mudado somente pelas operaçoẽs 3, 4 ou 5.
3. Se justificação positiva é requerida, os bits Is do ponteiro corrente são invertidos e o byte ou bytes relativos a oportunidade de justificação positiva é preenchido ou são preenchidos com informação inválida. Os ponteiros subsequentes contêm o valor do ponteiro anterior à justificação, incrementado de 1. Nenhuma operação subsequente de incremento ou decremento no valor do ponteiro é permitida para no mínimo os próximos três quadros seguintes a esta operação.

4. Se justificação negativa é requerida, os bits Ds do ponteiro corrente são invertidos e o byte ou bytes relativos a oportuninade de justificação negativa é usado ou são usados para transportar informação válida. Os ponteiros subsequentes contêm o valor do ponteiro anterior à justificação, decrementado de 1. Nenhuma operação subsequente de incremento ou decremento no valor do ponteiro é permitida para no mínimo os próximos três quadros seguintes a esta operação.

5. Se o alinhamento do quadro VC muda por qualquer outra razão que não sejam as regras 3 ou 4, o novo valor de ponteiro deve vir acompanhado pelos bits de NDF preenchidos com "1001". Os bits de NDF são preenchidos com "1001" somente no primeiro ponteiro que contém o novo valor, os ponteiros seguintes trazem bits de NDF com "0110". A nova localização do VC é indicada pelo novo valor de ponteiro e já é válida no primeiro ponteiro com o novo valor. Nenhuma operação subsequente de incremento ou decremento no valor do ponteiro é permitida para no mínimo os próximos três quadros seguintes a esta operação.

6. Regra adicional válida para o ponteiro do AU-4: Se um sinal de AU-4-Xc esta sendo transmitido, o ponteiro é gerado somente para o primeiro AU-4. O indicador de concatenação(CI) é gerado no lugar dos ponteiros dos outros AU-4s do AU-4-Xc. Todas as operaçoẽs realizadas pelo ponteiro do primeiro AU-4, se aplicam também aos ponteiros dos demais AU-4s do AU-4-Xc.

7. Regra adicional válida para o ponteiro TU-1/TU-2: Se o tamanho do TU dentro de um TUG-2 muda, então uma operação de NDF como descrito na regra 5 deve ser realizada simultaneamente em todos os TUs de tamanhos novos contidos neste TUG-2.

Interpretação

1. Durante operação normal o ponteiro localiza o início do quadro VC dentro do quadro AU/TU.

2. Qualquer mudança no valor do ponteiro é ignorada a não ser que um novo valor consistente seja recebido 3 vezes consecutivas ou ele é precedido por uma das regras 3, 4 ou 5. Qualquer valor consistente recebido 3 vezes consecutivas tem prioridade sobre as regras 3 ou 4.

3. Se a maioria dos bits Is do ponteiro são invertidos, uma operação de justificação positiva é indicada. Valores de ponteiros subsequentes devem ser incrementados de 1.

4. Se a maioria dos bits Ds do ponteiro são invertidos, uma operação de justificação negativa é indicada. Valores de ponteiros subsequentes devem ser decrementados de 1.

5. Se os bits de NDF são recebidos como sendo"1001", então o valor do ponteiro que acompanha esta operação de NDF deve substituir o valor do ponteiro corrente independente do estado do receptor.

6. Regra adicional válida para o ponteiro TU-3: Se o ponteiro do TU-3 traz indicação de ponteiro nulo(NPI), então qualquer modificação no seu valor é ignorada a não ser que um valor seja recebido de forma consistente por três vezes consecutivas.

7. Regra adicional válida para o ponteiro do AU-4: Se o ponteiro contém o indicador de concatenação(CI), então todas as operaçoẽs realizadas sobre o ponteiro do primeiro AU-4, se aplicam também aos ponteiros dos demais AU-4s do AU-4-Xc. Qualquer variação do CI é ignorada a não ser que um novo valor consistente de ponteiro seja recebido três vezes consecutivas.

8. Regra adicional válida para o ponteiro TU-1/TU-2: Se são recebidos simultaneamente os bits de NDF como "1001" e um novo tamanho arbitrário de TU em todos os TUs contidos em um TUG-2, então valores coincidentes de ponteiros e tamanhos devem substituir os correntes imediatamente.

2.4 A Operação de Mapeamento

Mapeamento é o procedimento pelo qual sinais afluentes são adaptados dentro dos VCs. Aqui se mostra o mapeamento dos sinais afluentes definidos na Recomendação G-703.

2.4.1 Mapeamento de Sinais Afluentes no VC-4

Mapeamento Assíncrono de 139.264 kbit/s

Um sinal de 139.264 kbit/s pode ser mapeado em um VC-4 de um quadro STM como mostram as figuras 2.23 e 2.24.

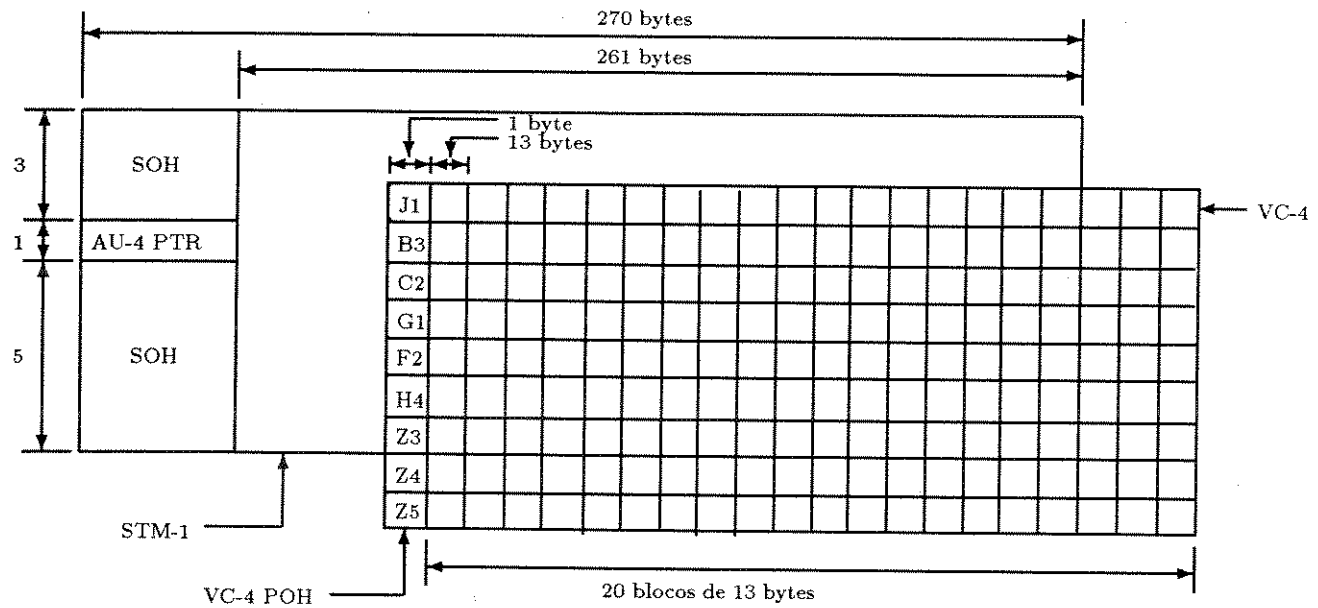


Figura 2.23: Mapeamento do VC-4 no STM-1 e Estrutura de Blocos do VC-4 para Mapeamento Assíncrono de 139.264 kbit/s

O VC-4 consiste de 9 bytes de POH (uma coluna) mais um quadro de 9 linhas por 260 colunas de bytes relativos a carga útil como mostrado na figura 2.23. Este quadro pode ser usado para carregar um sinal de 139.264 kbit/s da seguinte maneira:

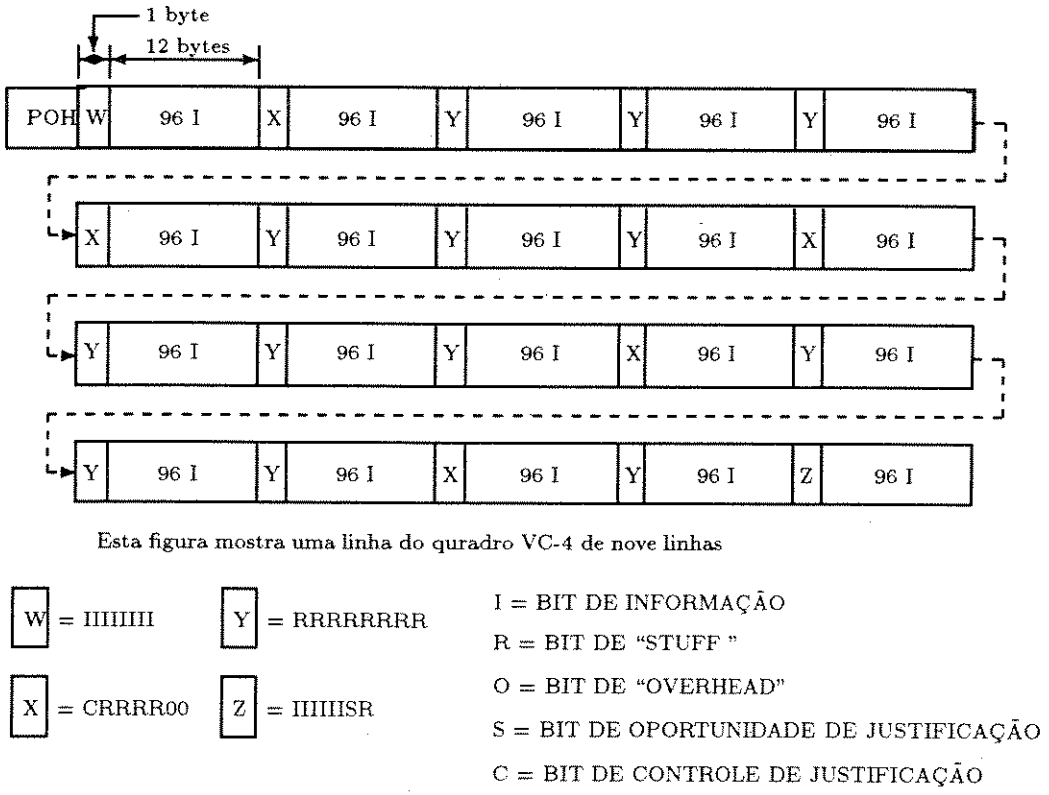


Figura 2.24: Mapeamento Assíncrono de 139.264 kbit/s no VC-4

- Cada uma das 9 linhas é particionada em 20 blocos, cada um contendo 13 bytes. Veja figura 2.23.
- Cada linha traz um bit de oportunidade de justificação(S) e cinco bits de controle de justificação. Veja figura 2.24.
- O primeiro byte de cada bloco pode ser um byte W, X, Y ou Z.

  São as seguintes as composições dos bytes de cada bloco:

  - Byte W: Oito bits de informação.
  - Byte X: Um bit de controle de justificação(C), cinco bits de enchimento ("stuff") mais dois bits de supervisão ("overhead").
  - Byte Y: Oito bits de "stuff".
  - Byte Z: Seis bits de informação, um bit de oportunidade de justificação mais um bit de "stuff".
  - Os doze ultimos bytes de cada bloco consistem de informação.

  A figura 2.24 mostra com detalhes a sequência destes bytes. Os cinco bits de controle de justificação(C) controlam o bit de oportunidade de justificação(S) em cada linha. Quando CCCCC=00000 então S é um bit de informação. Do contrário se CCCCC=11111 então S é um bit de justificação. Para superar erros de transmissão, é tomada decisão majoritária nos cinco bits C.

- Existem 104 bits de informação no primeiro grupo de 13 bytes, 102 no último e 96 nos demais. Têm-se 20 grupos de 13 bytes por linha e 9 linhas por quadro de 125 μs. Portanto a taxa de transmissão deveria ser de:

$$\frac{(104 + 102 + 18 \times 96) \times 9\ bits}{125\mu s} = 139.248\ kbit/s \tag{2.1}$$

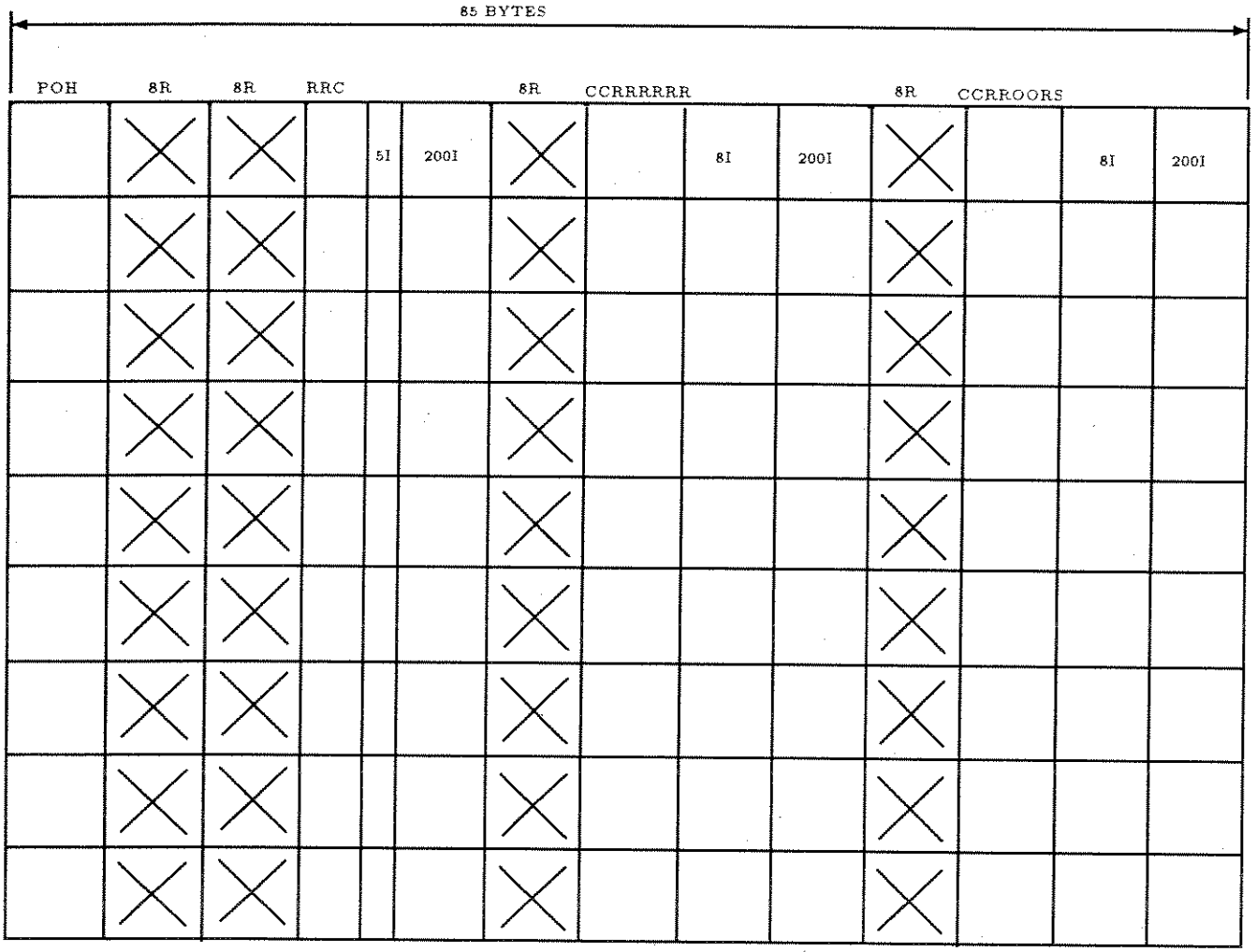
  Para completar a taxa nominal de 139.264 kbit/s faltam 16 kbit/s; como 1 bit no quadro de 125 μs equivale a 8 kbit/s, significa que se tem em média 2 justificações com sinal válido por quadro, completando o valor nominal de 139.264 kbit/s.

2.4.2 Mapeamento de Sinais Afluentes no VC-3

Mapeamento Assíncrono de 44.736 kbit/s

Um sinal de 44.736 kbit/s pode ser mapeado em um VC-3 como mostrado na figura 2.25. O VC-3 consiste de 9 subquadros a cada 125 μs. Cada subquadro tem a seguinte composição:

- 1 byte de POH.
- 621 bits de dados (I).
- 5 bits de controle de justificação (C).
- 1 bit de oportunidade de justificação (S).
- 2 bits de "overhead" (O).
- Os bits restantes são "stuff" (R).



R = "STUFFING"
C = BIT PARA CONTROLE DE JUSTIFICAÇÃO
S = BIT DE OPORTUNIDADE DE JUSTIFICAÇÃO
I = BIT DE INFORMAÇÃO
O = BIT DE "OVERHEAD"

Figura 2.25: Mapeamento Assíncrono de 44.736 kbit/s no VC-3

Os cinco bits de controle de justificação(C) controlam o bit de oportunidade de justificação(S) em cada linha. Quando CCCCC=00000 então S é um bit de informação. Do contrário se CCCCC=11111 então S é um bit de justificação. Para superar erros de transmissão, é tomada decisão majoritária nos cinco bits C.

Existem 9 subquadros em 125 μs, cada um carregando 621 bits de dados. Portanto a taxa de transmissão deveria ser de:

$$\frac{621 \times 9\ bits}{125\mu s} = 44.712\ kbit/s \tag{2.2}$$

Faltam 24 kbit/s para completar a taxa nominal de 44.736 kbit/s. Portanto na média, 3 dos bits "S" são usados para transportar informação a cada quadro de 125 μs.

Mapeamento Assíncrono de 34.368 kbit/s

Um sinal de 34.368 kbit/s pode ser mapeado em um VC-3 como mostrado na figura 2.26.

Além de 1 coluna de 9 bytes de POH, o VC-3 contém um quadro de carga útil de 9x84 bytes a cada período de 125 μs. Este quadro é dividido em 3 subquadros, cada um consistindo de:

- 1.431 bits de informação (I).
- 2 conjuntos de 5 bits de controle de justificação (C1 e C2).
- 2 bits de oportunidade de justificação (S1 e S2).
- 573 bits de "stuff" (R).

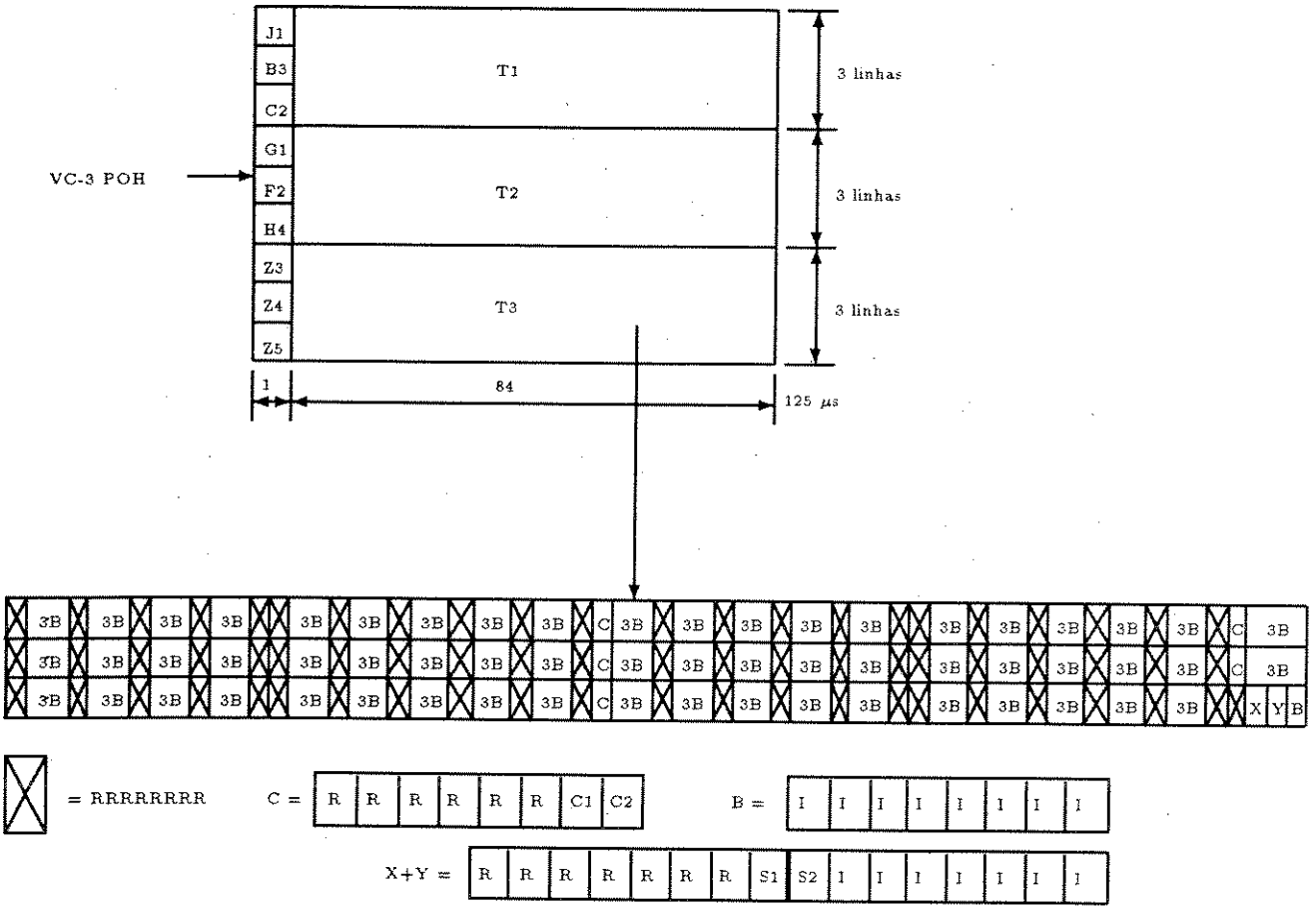
Os cinco bits de controle de justificação(C1) controlam o bit de oportunidade de justificação(S1) . Quando C1C1C1C1C1=00000 então S1 é um bit de informação. Do contrário se C1C1C1C1C1=11111 então S1 é um bit de justificação. Para superar erros de transmissão, é tomada decisão majoritária nos cinco bits C1.

Os cinco bits de controle de justificação(C2) controlam o bit de oportunidade de justificação(S2) . Quando C2C2C2C2C2=00000 então S2 é um bit de informação. Do contrário se C2C2C2C2C2=11111 então S2 é um bit de justificação. Para superar erros de transmissão, é tomada decisão majoritária nos cinco bits C2.

Como existem 3 subquadros no periodo de 125 μs, cada um transportando 1431 bits de informação, a taxa de transmissão deveria ser de:

$$\frac{3 \times 1.431\ bits}{125\mu s} = 34.344\ kbit/s \tag{2.3}$$

Faltam 24 kbit/s para completar a taxa nominal de 34.368 kbit/s. Portanto, na média, 3 das 6 posições de S1 e S2 são usadas para transportar informação.



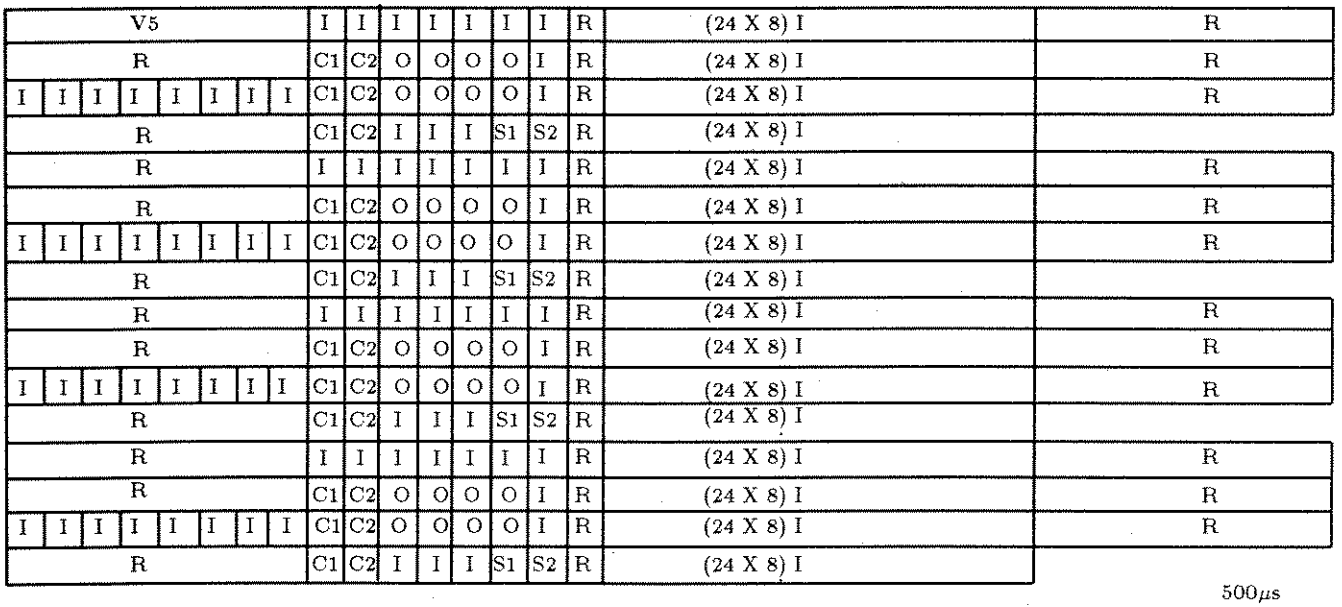
R = "STUFFING"
C1, C2 = CONTROLE DE JUSTIFICAÇÃO
S1, S2 = OPORTUNIDADE DE JUSTIFICAÇÃO
I = BIT DE INFORMAÇÃO

Figura 2.26: Mapeamento Assíncrono de 34.368 kbit/s no VC-3

2.4.3 Mapeamento de Sinais Afluentes no VC-2

Mapeamento Assíncrono de 6.312 kbit/s

Um sinal de 6.312 kbit/s pode ser mapeado em um VC-2. A figura 2.27 mostra isto em um período de 500 μs. Além de POH (V5), o VC-2 leva:



R = "STUFFING"
C = CONTROLE DE JUSTIFICAÇÃO
S = OPORTUNIDADE DE JUSTIFICAÇÃO
I = BIT DE INFORMAÇÃO
O = "OVERHEAD"

Figura 2.27: Mapeamento Assíncrono de 6.312 kbit/s no VC-2

- 3.152 bits de dados (I).
- 24 bits de controle de justificação (C1 e C2).
- 8 bits de oportunidade de justificação (S1 e S2).
- 32 bits de "overhead" (O) reservados para o futuro.
- O restante dos bits é "stuffing".

Em cada 125 μs, 3 bits de controle de justificação(C1) controlam 1 bit de oportunidade de justificação(S1) . Quando C1C1C1=000 então S1 é um bit de informação. Do contrário se C1C1C1=111 então S1 é um bit de justificação. Para superar erros de transmissão, é tomada decisão majoritária nos 3 bits C1.

Em cada 125 μs, 3 bits de controle de justificação(C2) controlam 1 bit de oportunidade de justificação(S2) . Quando C2C2C2=000 então S2 é um bit de informação. Do contrário se C2C2C2=111 então S2 é um bit de justificação. Para superar erros de transmissão, é tomada decisão majoritária nos 3 bits C2.

No período de 500 μs existem 3.152 bits de informação. Portanto a taxa de transmissão deveria ser de:

$$\frac{3.152\ bits}{500\mu s} = 6.304\ kbit/s \tag{2.4}$$

Faltam 8 kbit/s para completar a taxa nominal de 6.312 kbit/s. Portanto na média, dos 8 bits de oportunidade de justificação (S1 e S2), 4 são usados para transportar informação.

Mapeamento de 6.312 kbit/s por sincronismo de bit

A figura 2.28 ilustra este tipo de mapeamento.

V5	I	I	I	I	I	I	I	R	(24 X 8) I	R
R	1	0	O	O	O	O	I	R	(24 X 8) I	R
I I I I I I I I	1	0	O	O	O	O	I	R	(24 X 8) I	R
R	1	0	I	I	I	R	I	R	(24 X 8) I	
R	I	I	I	I	I	I	I	R	(24 X 8) I	R
R	1	0	O	O	O	O	I	R	(24 X 8) I	R
I I I I I I I I	1	0	O	O	O	O	I	R	(24 X 8) I	R
R	1	0	I	I	I	R	I	R	(24 X 8) I	
R	I	I	I	I	I	I	I	R	(24 X 8) I	R
R	1	0	O	O	O	O	I	R	(24 X 8) I	R
I I I I I I I I	1	0	O	O	O	O	I	R	(24 X 8) I	R
R	1	0	I	I	I	R	I	R	(24 X 8) I	
R	I	I	I	I	I	I	I	R	(24 X 8) I	R
R	1	0	O	O	O	O	I	R	(24 X 8) I	R
I I I I I I I I	1	0	O	O	O	O	I	R	(24 X 8) I	R
R	1	0	I	I	I	R	I	R	(24 X 8) I	

500μs

R = "STUFFING"
I = BIT DE INFORMAÇÃO
O = "OVERHEAD"

Figura 2.28: Mapeamento de 6.312 kbit/s no VC-2 por Sincronismo de Bit

A única diferença para o mapeamento assíncrono é que os bits de controle de justificação assumem sempre os seguintes valores:

- C1 é sempre 1.

- C2 é sempre 0.

Com isto, o mesmo desincronizador pode ser usado tanto para mapeamento síncrono como para mapeamento assíncrono.

2.4.4 Mapeamento de Sinais Afluentes no VC-12

Mapeamento Assíncrono de 2.048 kbit/s

Um sinal de 2.048 kbit/s pode ser mapeado em um VC-12. A figura 2.29 mostra isto no modo assíncrono em um período de 500 μs.

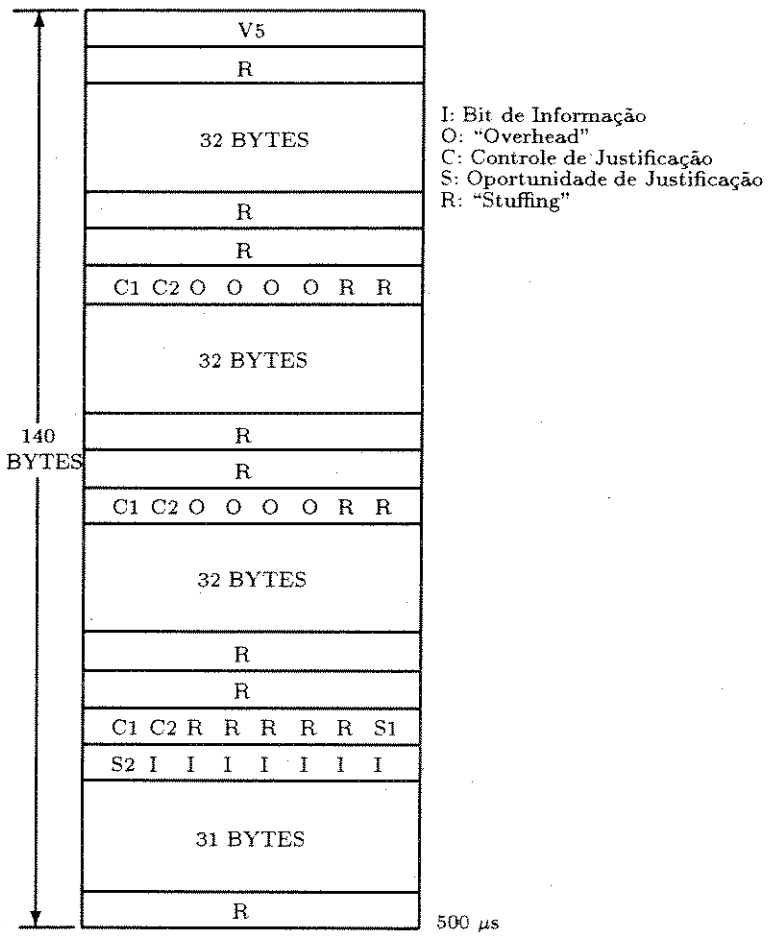


Figura 2.29: Mapeamento Assíncrono de 2.048 kbit/s no VC-12

Além de POH (V5), o VC-12 leva:

- 1.023 bits de dados (I).

- 6 bits de controle de justificação (C1 e C2).
- 2 bits de oportunidade de justificação (S1 e S2).
- 8 bits de "overhead" (O) reservados para o futuro.
- O restante dos bits é "stuffing".

Em cada 500 μs, os 3 bits de controle de justificação(C1) controlam o bit de oportunidade de justificação(S1) . Quando C1C1C1=000 então S1 é um bit de informação. Do contrário se C1C1C1=111 então S1 é um bit de justificação. Para superar erros de transmissão, é tomada decisão majoritária nos 3 bits C1.

Em cada 500 μs, os 3 bits de controle de justificação(C2) controlam o bit de oportunidade de justificação(S2) . Quando C2C2C2=000 então S2 é um bit de informação. Do contrário se C2C2C2=111 então S2 é um bit de justificação. Para superar erros de transmissão, é tomada decisão majoritária nos 3 bits C2.

No período de 500 μs existem 1.023 bits de informação. Portanto a taxa de transmissão deveria ser de:

$$\frac{1.023\ bits}{500\mu s} = 2.046\ kbit/s \tag{2.5}$$

Faltam 2 kbit/s para completar a taxa nominal de 2.048 kbit/s. Portanto na média, dos 2 bits de oportunidade de justificação (S1 e S2), 1 é usado para transportar informação.

Mapeamento de 2.048 kbit/s por sincronismo de bit

O mapeamento do sinal de 2.048 kbit/s por sincronismo de bit é mostrado na figura 2.30.

Note que o mesmo desincronizador pode ser usado tanto para mapeamento síncrono como para mapeamento assíncrono.

Mapeamento de 2.048 kbit/s por sincronismo de byte

A figura 2.31 mostra o mapeamento por sincronismo de byte para 30 canais(2.048 kbit/s), empregando *sinalização associada a canal* (CAS).

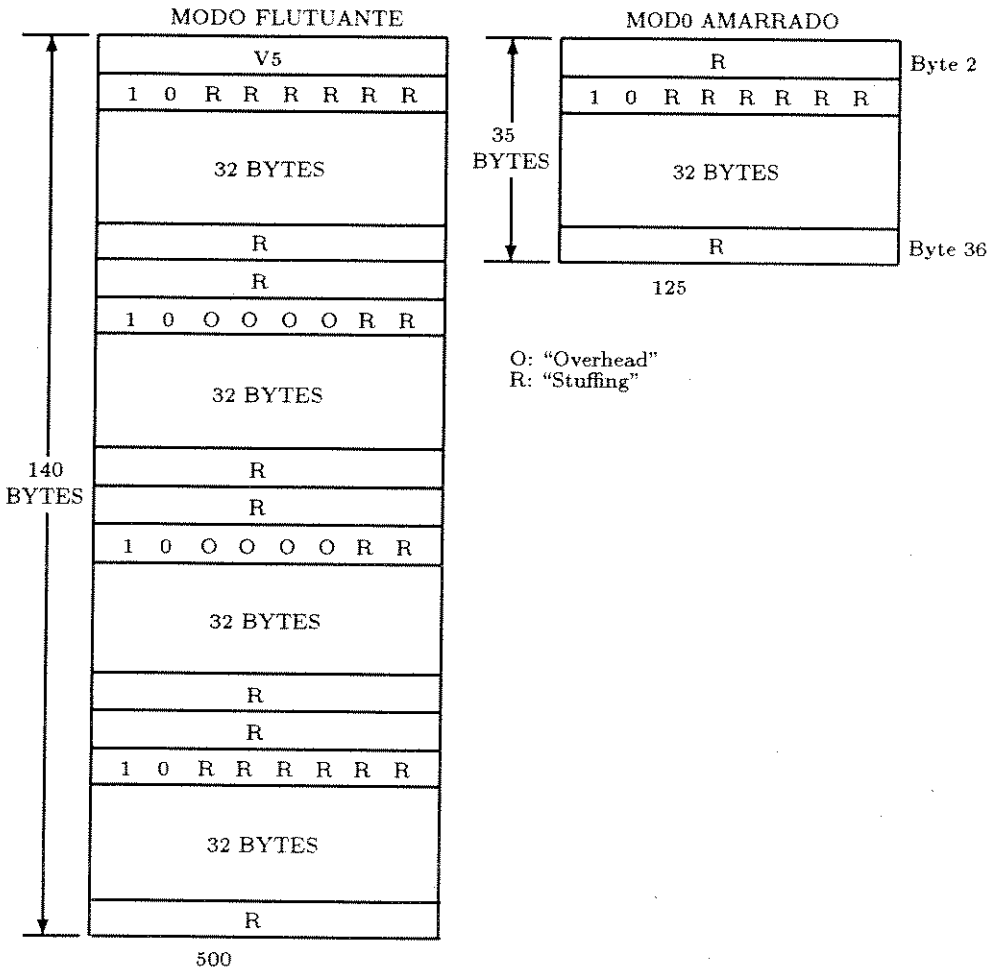


Figura 2.30: Mapeamento de 2.048 kbit/s no VC-12 por Sincronismo de Bit

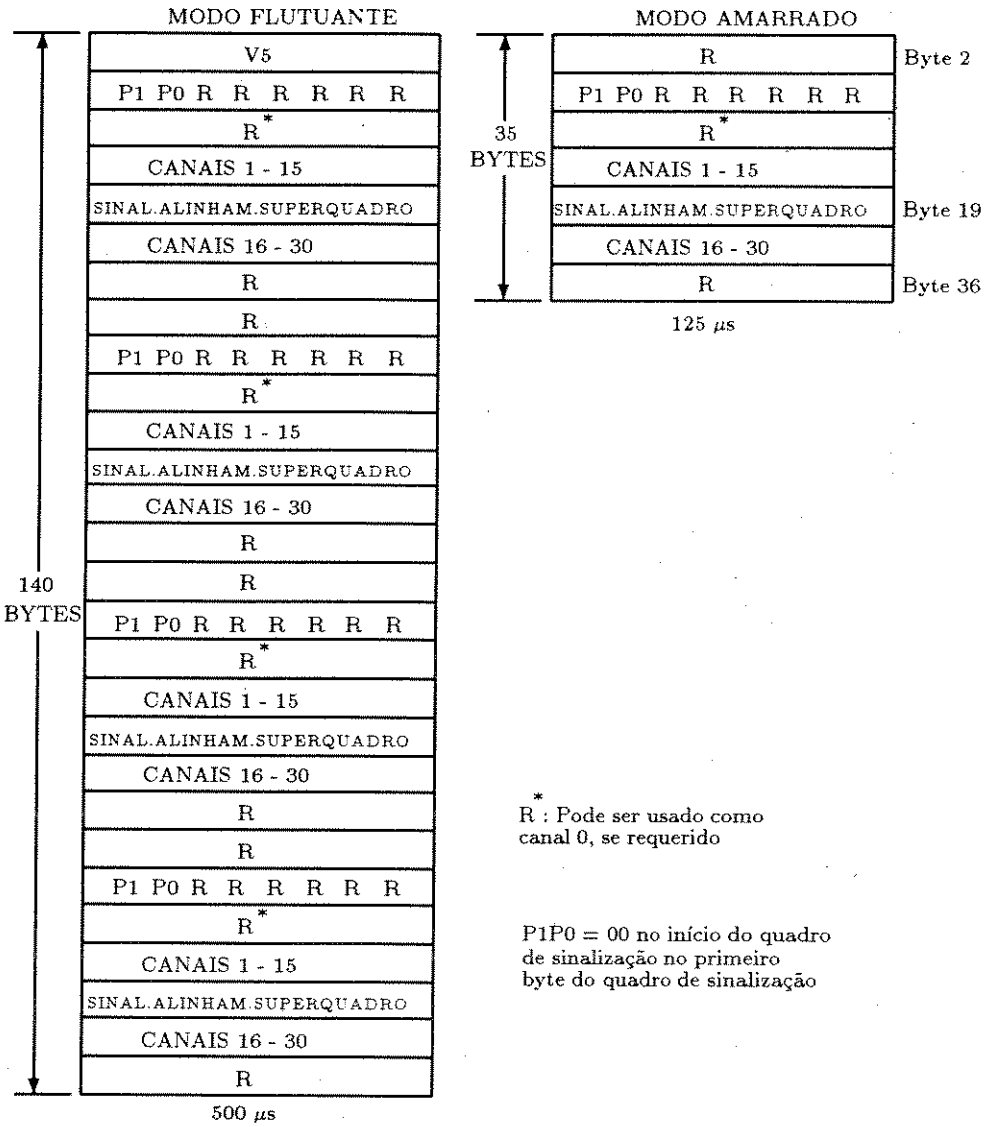


Figura 2.31: Mapeamento de 2.048 kbit/s no VC-12 por Sincronismo de Byte (30 Canais com Sinalização Associada a Canal)

2.4.5 Mapeamento de Sinais Aflentes no VC-11

Mapeamento Assíncrono de 1.544 kbit/s

Um sinal de 1.544 kbit/s pode ser mapeado em um VC-11. A figura 2.32 mostra isto no modo assíncrono em um período de 500 μs.

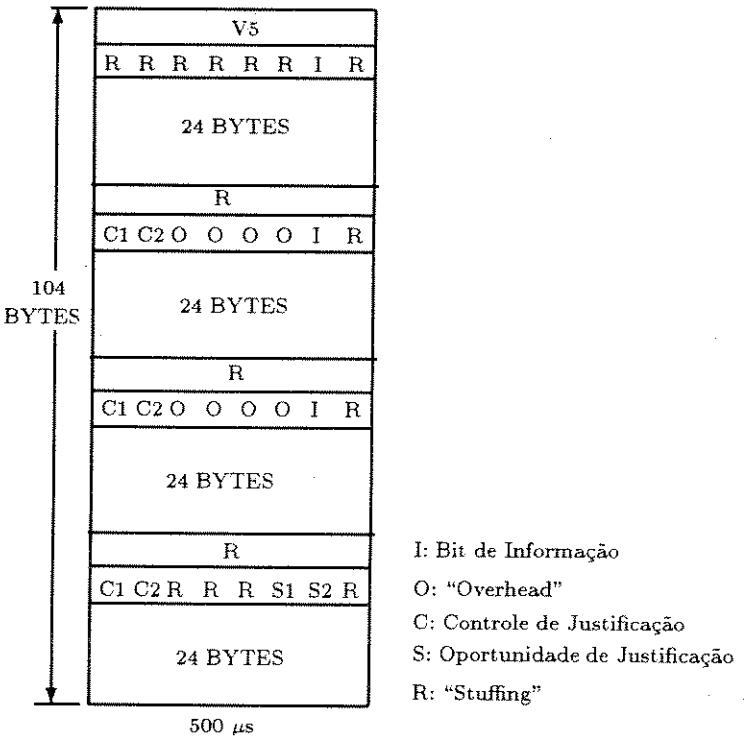


Figura 2.32: Mapeamento Assíncrono de 1.544 kbit/s no VC-11

Além de POH (V5), o VC-11 leva:

- 771 bits de dados (I).
- 6 bits de controle de justificação (C1 e C2).
- 2 bits de oportunidade de justificação (S1 e S2).
- 8 bits de "overhead" (O) reservados para o futuro.
- O restante dos bits é "stuffing".

Em cada 500 μs, os 3 bits de controle de justificação(C1) controlam o bit de oportunidade de justificação(S1) . Quando C1C1C1=000 então S1 é um bit de informação. Do contrário se C1C1C1=111 então S1 é um bit de justificação. Para superar erros de transmissão, é tomada decisão majoritária nos 3 bits C1.

Em cada 500 μs, os 3 bits de controle de justificação(C2) controlam o bit de oportunidade de justificação(S2) . Quando C2C2C2=000 então S2 é um bit de informação. Do contrário se C2C2C2=111 então S2 é um bit de justificação. Para superar erros de transmissão, é tomada decisão majoritária nos 3 bits C2.

No período de 500 μs existem 771 bits de informação. Portanto a taxa de transmissão deveria ser de:

$$\frac{771\ bits}{500\mu s} = 1.542\ kbit/s \tag{2.6}$$

Faltam 2 kbit/s para completar a taxa nominal de 1.544 kbit/s. Portanto na média, dos 2 bits de oportunidade de justificação (S1 e S2), 1 é usado para transportar informação.

Mapeamento de 1.544 kbit/s por sincronismo de bit

O mapeamento do sinal de 1.544 kbit/s por sincronismo de bit é mostrado na figura 2.33.

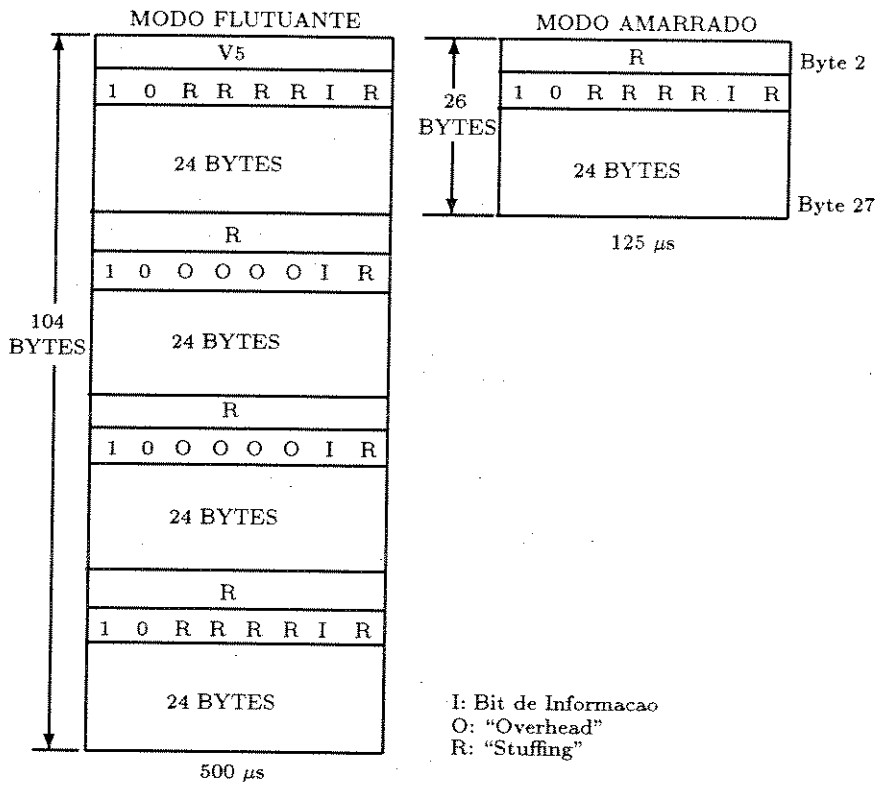


Figura 2.33: Mapeamento de 1.544 kbit/s no VC-11 por Sincronismo de Bit

Note que o mesmo desincronizador pode ser usado tanto para mapeamento síncrono como para mapeamento assíncrono.

Mapeamento de 1.544 kbit/s por sincronismo de byte

A figura 2.34 mostra o mapeamento por sincronismo de byte para 24 canais(1.544 kbit/s).

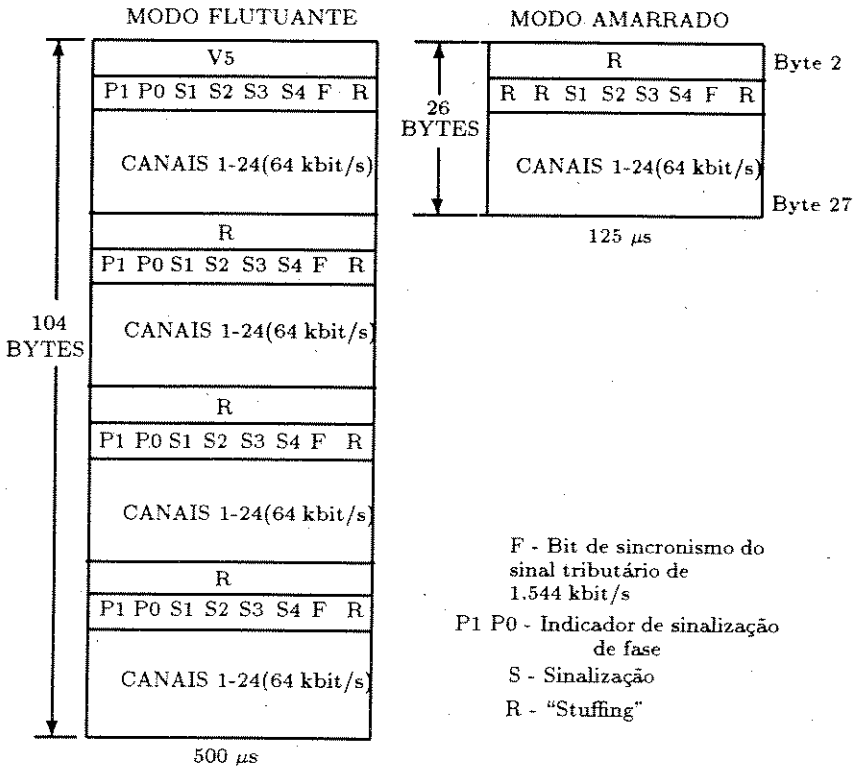


Figura 2.34: Mapeamento de 1.544 kbit/s no VC-11 por Sincronismo de Byte

Capítulo 3

Blocos Funcionais de Equipamentos da Hierarquia Digital Síncrona

Obs: Este capítulo corresponde a uma parte do curso "Introdução à Rede Síncrona", ministrado pelo Prof. Rege Romeu Scarabucci no Curso de Pós-Graduação da Faculdade de Engenharia Elétrica da UNICAMP em 1992, cujas notações são aqui reproduzidas.

3.1 Introdução

Neste capítulo analisa-se o processamento dos sinais que se interagem dentro de um equipamento multiplex. O CCITT estabeleceu blocos funcionais bem caracterizados, de tal modo que, pelo agrupamento desses vários blocos funcionais, obtém-se a funcionalidade completa de qualquer equipamento SDH. Na primeira parte deste capítulo, os blocos funcionais que compõem um multiplex generalizado são estudados.

Na segunda parte, os vários equipamentos de aplicações específicas são apresentados, a partir dos blocos funcionais já estudados.

3.2 Diagrama de Blocos do Equipamento Geral

O diagrama de blocos contendo todas as funções básicas de um equipamento geral está mostrado a seguir na figura 3.1.

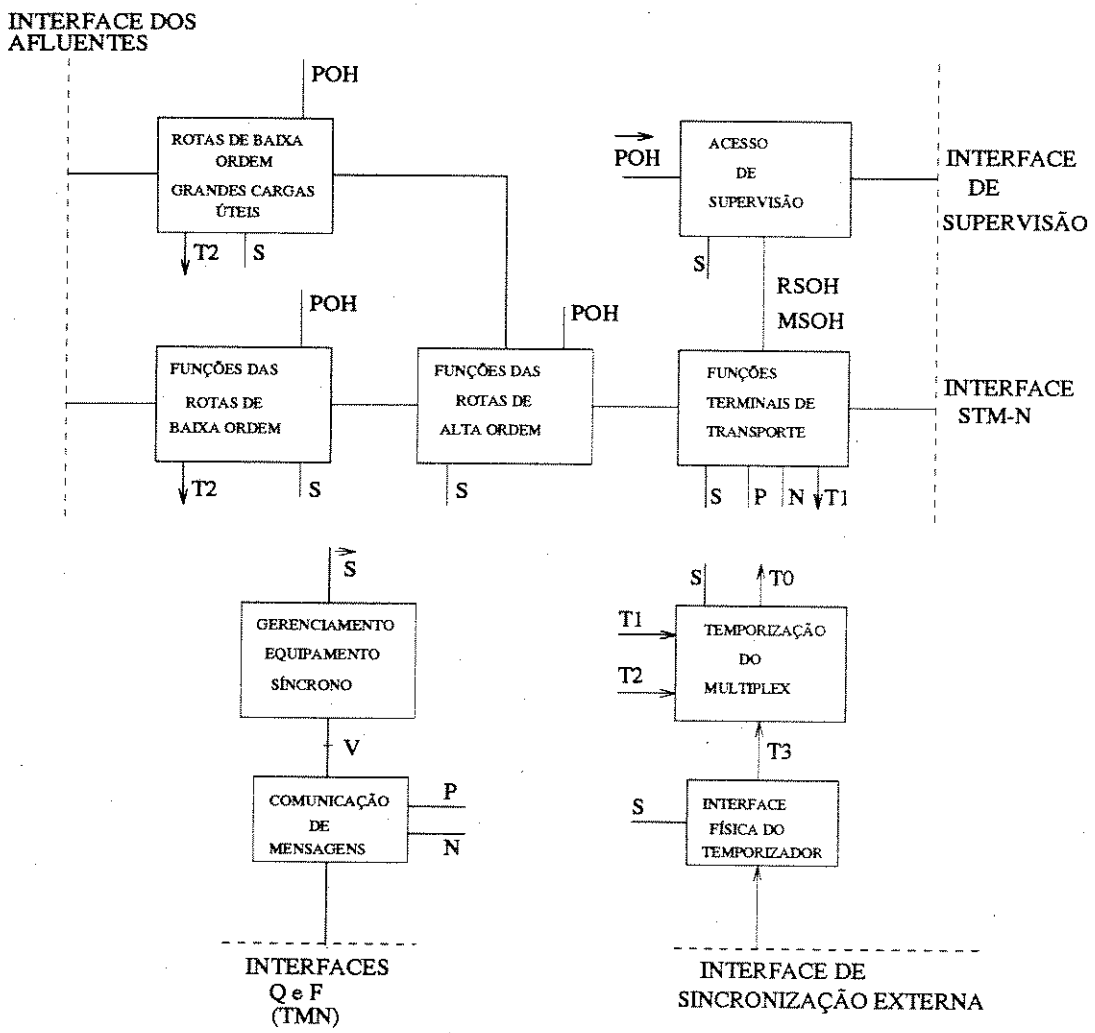


Figura 3.1: Diagrama de blocos com funções básicas do equipamento geral

As Funções Terminais de Transporte executam as seguintes tarefas:

- Interface Física SDH
- Terminação de Seção Regeneradora
- Terminação de Seção Multiplex
- Proteção da Seção Multiplex
- Adaptação (Montagem/Desmontagem de AUG's)

Como os nomes indicam, estas funções interfaceiam o equipamento com a rede SDH física; na direção de transmissão ocorre a montagem dos Grupamentos de Unidades Administrativas (AUG's), a montagem dos octetos de supervisão da Seção Multiplex e da Seção Regeneradora e a interface física, seja ela elétrica ou óptica; na direção de recepção ocorre a interface de sinais (e/e ou o/e), a deteção do sincronismo, a recepção dos bytes de supervisão da Seção Regeneradora e da Seção Multiplex, atuação ou não do chaveamento de proteção e, finalmente, a desmontagem geral da carga em suas Unidades Administrativas.

As Funções das Rotas de Alta Ordem processam os mapeamentos VC-3/VC-4 em AU's ou os mapeamentos dos Containers de Baixa Ordem em TU's e daí em AU's. Neste sentido estas funções são responsáveis por

- conexão in/out dos containers de alta ordem
- função de terminação para rotas de alta ordem

Funções das Rotas de Baixa Ordem

Existem 5 capacidades básicas para as rotas de entrada, para as quais são usados índices 11, 12, 2, 3 e 4. Em adição, com as concatenações definidas para o nível 2, é possível a criação de até 21 novas capacidades de rotas. Os sinais afluentes são adaptados para formarem containers que são então alocados em rotas de alta ordem.

As seguintes funções são realizadas:

- Interface Física de Afluentes
- Adaptação (mapeamento/desmapeamento)
- Terminação de Rota (in/out dos octetos de supervisão - POH)
- Conexão das Rotas de Baixa Ordem (LVC $\leftrightarrow$ HVC)

Para sinais de alta velocidade, que são mapeados diretamente nos containers de alta ordem, economiza-se processamentos intermediários, interfaceando estes sinais com as funções das Rotas de Alta Ordem.

Acesso de Supervisão provê a interface dos sinais de supervisão (overhead) do equipamento com o mundo exterior.

A Função de Gerenciamento do Equipamento Síncrono é a de converter dados de desempenho e alarmes específicos do hardware em mensagens "object-oriented" para transmissão através dos octetos DCC (Data Communication Channel) e/ou através da interface $\vec{Q}$. Todos os blocos do equipamento são supervisionados (saída/entrada S).

A Fonte de Temporização do Multiplex provê referência T0 de temporização (relógio do equipamento) para todos os blocos do equipamento e representa o relógio SDH deste elemento da rede.

3.3 Funções Terminais de Transporte

A descrição dos bytes de supervisão de seção é um pré-requisito para o entendimento das Funções Terminais de Transporte. Portanto, eles são descritos a seguir.

3.3.1 CARGA DE SUPERVISÃO DE SEÇÃO

Para supervisão de seção alocam-se bytes para supervisão da seção regeneradora e outros para a seção multiplex. São denominados RSOH (Regenerator Section Overhead) e MSOH (Multiplex Section Overhead) respectivamente.

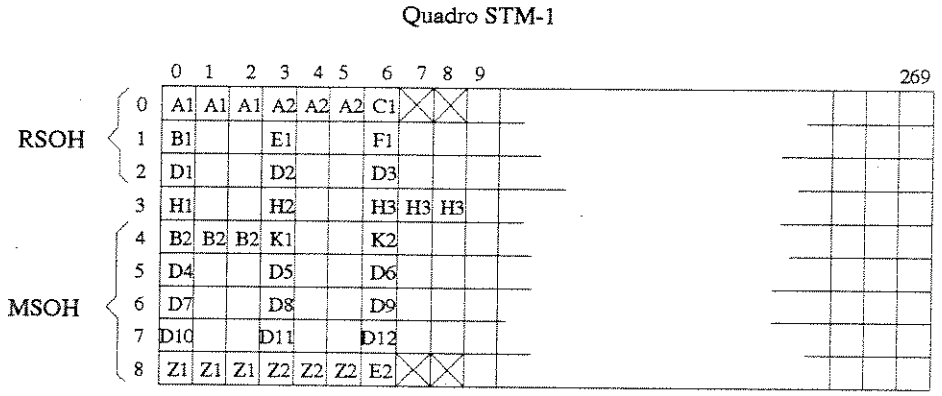


Figura 3.2: Quadro STM-1 com Bytes de Supervisão de Seção (SOH).

ALINHAMENTO DE QUADRO (Bytes A_1 e A_2)

Os bytes A_1 e A_2 são usados para Alinhamento de Quadro.

$A_1 \equiv 11110110 \qquad A_2 \equiv 00101000$

A_1 e A_2 são replicados N vezes em um STM-N.

$(A_1 + A_2) \equiv$ FAW (Frame Alignment Word) é uma PAQ de 16 bits. Aqui não se especifica um algorítmo para recuperação do sincronismo. Ao invés disto especifica-se (Recomendação G.783):

- Tempo para detectar OOF (Condição Out of Frame)

- Tempo médio entre deteções falsas de OOF para BER fixada
- Tempo máximo de recuperação de sincronismo a partir de condição boa de sinal
- Máxima probabilidade de recuperação equivocada de sincronismo de quadro com sinal bom.

Diferentemente do caso PDH, aqui na SDH, dado um intervalo de tempo entre a perda do relógio de sincronização e a sua volta, há uma grande chance do oscilador local ainda estar em fase com o sincronismo.

BYTE DE IDENTIFICAÇÃO DO STM - Byte C_1

C_1 identifica o STM#i em um sinal STM-N. Ele tem o valor binário equivalente à posição da intercalação. Assim, em um STM-16, o C_1 correspondente ao STM#1 tem valor 00000001, enquanto aquele correspondente ao STM#16 tem o valor 00010000. Este byte pode ser auxiliar na deteção de sincronismo.

Os bytes marcados X à direita de C_1 são reservados para usos nacionais.

ADAPTAÇÃO DA SEÇÃO REGENERADORA PARA O MEIO FÍSICO

As funções de

- gerar e recuperar FAW ($A_1 + A_2$)
- recuperação de PAQ

são consideradas partes da função de adaptação na interface RS↔OS (Seção de Regeneração↔Seção Óptica).

Além disso, faz parte da função de adaptação RS↔OS:

- embaralhamento do sinal antes de transmitir
- formatação do pulso de excitação da cabeça óptica: RZ, duty cycle de 50%.

Existe uma função equivalente de adaptação para interface RS↔ES (sinal elétrico para cabo coaxial) que é feita através de código CMI (Coded Mark Inversion). Embaralhamento é aplicado a todo o sinal STM-N, excetuando-se a primeira linha do RSOH, a fim de se manter uma boa densidade de transições e balanceamento dc. A informação PAQ não deve ser embaralhada pois o embaralhamento retira a sua sincronização do quadro STM e que precisa ser recuperada antes que o resto do sinal seja desembaralhado.

Embaralhador: *síncrono*, com polinômio gerador $1 + x^6 + x^7$. A figura 3.3 mostra o circuito do embaralhador.

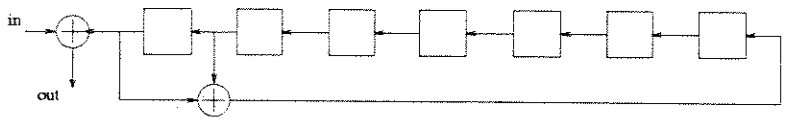


Figura 3.3: Embaralhador.

O embaralhador é colocado no estado "tudo um" no 1º bit do 1º byte da carga útil em cada quadro. Este bit (1º da carga útil) e todos os outros a serem embaralhados são somados módulo 2 à saída do embaralhador, que deve rodar até o final do quadro. Este tipo de embaralhador mantém boa densidade de marcas e valor adequado de dc no sinal transmitido.

Os bytes que seguem C_1 (X X) e que são de uso nacional, não são também embaralhados - recomenda-se que sejam usados mantendo-se o balanceamento dc (mesmo n^o de zeros e uns).

MONITORAÇÃO DE ERRO EM SEÇÃO REGENERADORA (Byte B_1)

Somente um byte é alocado para monitorar a taxa de erro na seção regeneradora. Em sistemas STM-4 e STM-16 somente o primeiro STM transporta um byte B_1 válido. O mecanismo de monitoração é denominado "bit interleaved parity" (BIP): o número de " UNS " existentes na posição i de cada byte é somado módulo 2 ao longo de todo o quadro e o resultado é colocado no bit i de B_1 do próximo quadro. Portanto, oito paridades independentes são geradas em B_1 e o processo denomina-se BIP-8.

As paridades são recalculadas no receptor e as discrepâncias detectadas são evidências de erros ocorridos no bloco de um quadro.

CANAIS DE SERVIÇO E DE USUÁRIO NA SEÇÃO REGENERADORA (Bytes E_1 e F_1)

O canal de serviço (Engineering Order Wire - EOW) provê um canal de voz para o pessoal de manutenção em locais de regeneração intermediária. O Canal de Serviço é um procedimento padronizado na PDH. Aqui até que não seria tão necessário, pois em princípio as regenerações devem ocorrer em locais controlados e facilmente acessíveis.

O canal de usuário F_1 é principalmente para ser usado transportando alarmes físicos de locais remotos ou outro uso qualquer. Só um byte na SDH.

CANAIS DE COMUNICAÇÕES DE DADOS NA RS (Bytes D_1, D_2 e D_3)

Capacidade de 192 kb/s para mensagens entre os regeneradores ou seções regeneradoras. Deverá ser usado pela empresa operadora em gerenciamento e supervisão de sistemas, sejam eles da SDH ou não.

Os bytes D_1, D_2 e D_3 podem ser tornados inoperantes.

MONITORAÇÃO DE ERRO NA SEÇÃO MULTIPLEX (Byte B_2)

3 bytes B_2 são alocados *em cada* STM do sinal agregado. Os processos de monitoração de erro são

$$\begin{cases} BIP\text{–}24 \quad para \quad STM\text{–}1 \quad (3\,bytes) \\ BIP\text{–}96 \quad para \quad STM\text{–}4 \quad (12bytes) \\ BIP\text{–}384 \quad para \quad STM\text{–}16 \quad (48bytes) \end{cases}$$

As 3 primeiras linhas do SOH são omitidas no cálculo da paridade mas todo o restante é usado, inclusive K_1 e K_2. Processo idêntico ao do BIP-8.

CHAVEAMENTO AUTOMÁTICO DE PROTEÇÃO

Os bytes (K_1 e K_2) são alocados no STM inicial com a função de coordenar o chaveamento de proteção de um grupo de seções multiplex. APS (Automatic Protection Switching) é um processo complexo de coordenar em uma interface aberta. Deu muita discussão durante o processo de padronização. Bits 6, 7 e 8 de K_2 são usados para sinalizar MS-FERF (110) - Far End Receiver Failure. Utiliza-se quando é detetada uma falha no receptor local (envia-se FERF).

Recepção de MS-AIS nestes bits é entendida como uma indicação de falha de um regenerador na direção "up-stream" e é usado para suprimir os alarmes locais.

REDES AUXILIARES DA SEÇÃO MULTIPLEX

A seção multiplex MS suporta duas redes auxiliares:

- um canal de comunicação (D_4-D_{12}) de 576 kb/s para mensagens entre terminações de seção multiplex em nós adjacentes da rede. Pode ser usado para comunicações entre entidades de gerenciamento da TMN (Telecommunications Managment Network).
- O byte E_2 é um simples canal de 64 kb/s para comunicação de voz entre terminações de seção multiplex em nós da rede. O CCITT não especificou a maneira de se usar este canal.

O diagrama de blocos que executa as funções terminais de transporte está mostrado na figura 3.4.

O relógio T1, retirado do sinal que entra em A, é fornecido ao sistema de Temporização do equipamento. Todos os blocos são supervisionados através do sistema de gerenciamento do equipamento (interfaces Si bidirecionais). T0 é o relógio SDH local fornecido pelo sistema de temporização. As interfaces com o Acesso de Supervisão também estão mostradas na figura 3.4.

3.3.2 Interface Física HDS (SPI)

A função SPI é a intermediação entre o meio físico de transmissão A e a Terminação de Seção Regeneradora (RST) no ponto B.

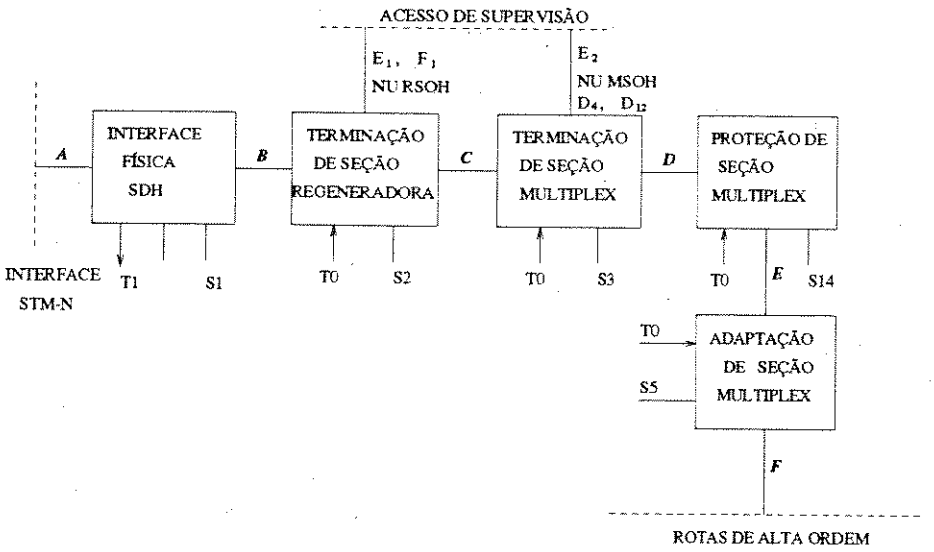


Figura 3.4: Funções Terminais de Transporte

A entrada do lado *A* pode ser elétrica ou óptica. Quando a interface é elétrica e o STM é o STM-1, então está padronizado o código de linha CMI (Coded Mark Inversion) para 155 Mb/s. (ver G.703). Para meio físico óptico, o sinal de interface está padronizado na recomendação G.957.

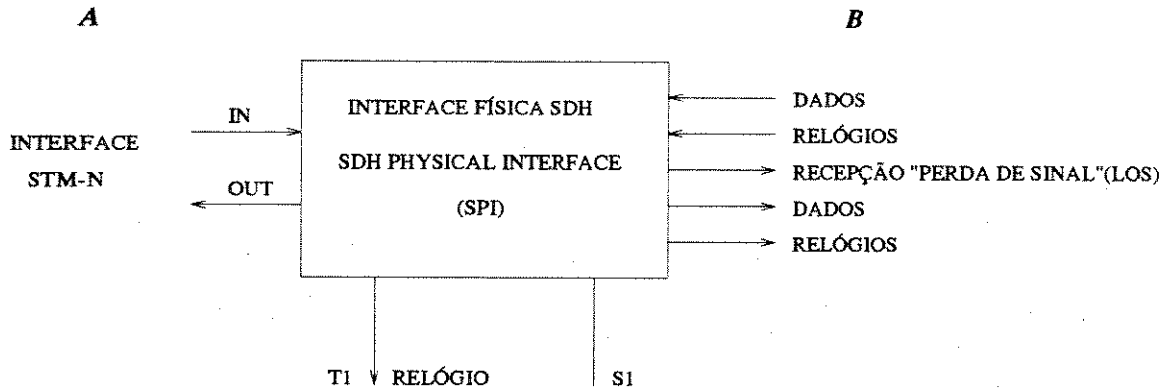


Figura 3.5: Interface Física SDH

Fluxo $A \rightarrow B$

- sinal degenerado dentro de limites
- regeneração do sinal
- saída Dados + Relógio
- relógio recuperado T1 enviado ao MTS
- Se houver perda do sinal em $A \rightarrow$ gera LOS para RST em B e para SEMF em S1. (critérios para LOS na G.958)
- Exemplo interface elétrica na figura 3.6

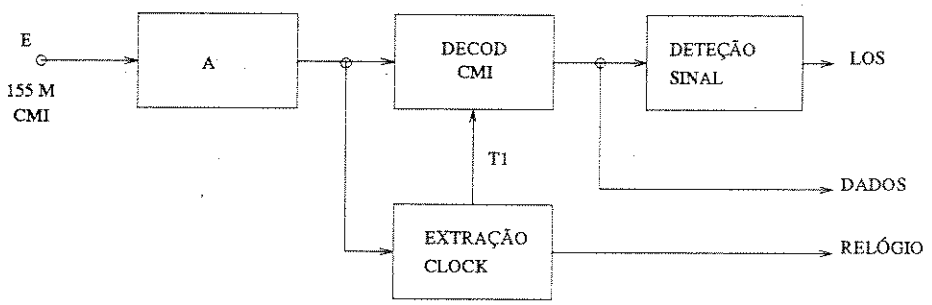


Figura 3.6: Interface Elétrica SDH

Fluxo B → A

- Dados e Relógios Transmitidos pelo RST completamente formatados de acordo com G.707/708 e 709.
- SPI condiciona os dados para transmissão no meio ligado em B.
- Parâmetros de desempenho do transmissor reportados em S1 (para meio óptico, parâmetros na G.958)
  - TX fail
  - TX degraded (nível óptico na saída, corrente de polarização do LASER, etc)

3.3.3 Terminação de Seção Regeneradora (RST)

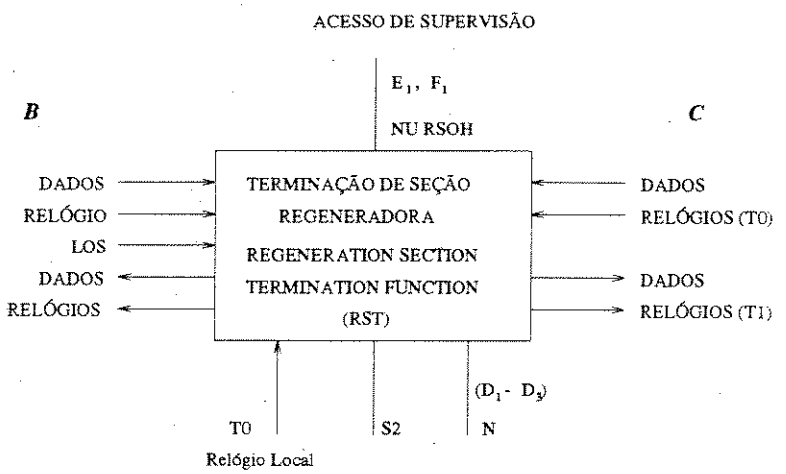


Figura 3.7: Terminação de Seção Regeneradora

A Terminação de Seção Regeneradora é fonte e destinatária dos octetos de supervisão de seção regeneradora (RSOH). Uma seção regeneradora é uma entidade de manutenção que inclui dois RST. Para Regenerador *Intermediário* os octetos A_1, A_2 e C_1 podem passar de forma transparente, ao invés de serem terminados e gerados, como indicado abaixo.

Fluxo B → C

Os dados formatados e regenerados STM-N com seus relógios associados são recebidos em B. oriundos de SPI. A função do RST é recuperar o alinhamento do quadro e identificar a posição de início do quadro nos dados a serem entregues em C. Para isto, o sinal STM-N é primeiramente desembaralhado (excetuando-se a primeira linha do RSOH) e os bytes de RSOH são recuperados antes de apresentar os DADOS STM-N e os relógios em C.

- PAQ é determinada procurando-se os bytes A_1 e A_2 nos dados de chegada.
- Algumas condições estabelecidas pelo CCITT:

- Estado inicial: com sincronismo de quadro → máximo tempo sem detectar quadro (Out of Frame - OOF) é de 625 μs (5 quadros) para sinal aleatório sem formatação de quadro.
- Para operação normal, uma taxa de erro de 10^{-3} não deve causar OOF falso mais do que 1 para cada 6 minutos.
- Estado Inicial: fora de sincronismo: máximo tempo para entrar em estado de sincronismo deve ser de 250 μs para um sinal sem erro e sem formatação de quadro.
- Algorítmo para recuperar o sincronismo deve ser tal que a probabilidade de falsa recuperação de sincronismo com sinal aleatório sem formatação de quadro não seja maior que 10^{-5} para um intervalo de 250 μs.
- Para estado OOF persistindo por (x) segundos → Perda de quadro deve ser declarada.
- Os eventos OOF devem ser reportados em S2 para monitoração de desempenho pelo SEMF. Perda de quadro LOF idem.

- O byte Identificador de STM, C_1, está presente no RSOH mas não há necessidade de processamento no RST.

- Depois do desembaralhamento, o byte de monitoração de erro B_1 é recuperado e comparado com o valor BIP-8 computado sobre todos os bits do quadro STM-N anterior em B antes do desembaralhamento. Erros são reportados em S2 como número de erros dentro do byte B_1 por quadro.

- O byte E_1 de serviço é recuperado e passado para o bloco de Acesso de Supervisão. Idem para o octeto do canal do usuário , F_1.

- O Canal de Comunicação de Dados (DCC_R), formado pelos bytes $D_1D_2D_3$ (192 kb/s), é recuperado e passado ao Sistema de Comunicação de Mensagens no ponto de referência N.

- Um ou mais dos octetos reservados para uso nacional ou futura padronização internacional podem ser recuperados e passados para o Acesso de Supervisão (OHA). O RST deve ignorar estes octetos.

- LOS ou LOF detectado ⇒ colocar tudo "1" nos DADOS em C para que isto seja recebido pela Terminação de Seção Multiplex. Tempos envolvidos ainda não determinados.

Fluxo $C \rightarrow B$

Os DADOS em C são um sinal STM-N como especificado nas recomendações G.707, G.708 e G.709, temporizado com relógio T0 e tendo uma Carga de Supervisão do Multiplex (MSOH) perfeitamente válida. Porém os octetos de Supervisão de Seção Regeneradora (RSOH) ainda não estão incluidos nos dados em C.

Depois de determinados todos os bytes RSOH, o sinal STM-N será embaralhado e apresentado no ponto B. O embaralhamento exclue os bytes RSOH da primeira linha (9×N bytes, incluindo A_1, A_2, C_1 e alguns bytes reservados para usos nacionais e internacionais).

- Os bytes de alinhamento de quadro A_1 e A_2 (3N de cada um) são gerados e inseridos na linha 1 do RSOH.

- Os bytes C_1 identificadores de STM-N são inseridos na primeira linha do RSOH. C_1 é um número binário correspondendo à ordem do STM que aparece no quadro.
  - O 1º STM a aparecer no quadro é identificado por $C_1 = (00000001) \rightarrow 1$
  - O 2º STM a aparecer no quadro é identificado por $C_1 = (00000010) \rightarrow 2$, etc.

  Se o sinal a sair em B for um STM-1, então o uso de C_1 é opcional.

- O octeto B_1 de monitoração de erro é colocado no STM-N para monitorar o erro na Seção Regeneradora. Deverá ser um código do tipo "bit interleaved parity 8" (BIP-8), usando paridade par como definido na recomendação G.708. O BIP-8 é computado sobre todos os bits do quadro STM-N anterior depois do seu embaralhamento. O resultado é colocado no byte B_1 do RSOH antes do seu embaralhamento.

- O octeto de serviço E_1 (Order Wire) é retirado do Acesso de Supervisão (OHA) e colocado na posição E_1 do RSOH. Este byte deve terminar em cada RST. Opcionalmente E_1 pode ser usado como canal de 64 kb/s, reservado como comunicação de voz entre *elementos da rede* → não está claro como isto se dá.

- O byte F_1 de Canal do Usuário é retirado do Acesso de Supervisão (OHA) e colocado na posição F_1 do RSOH. Ele é reservado para ser usado pela empresa operadora da rede. Em regeneradores intermediários, o acesso a F_1 é opcional. O modo de se usar F_1 ainda está em estudos.

- Os 3 bytes $D_1D_2D_3$ (192 kb/s) são retirados do Sistema de Comunicação de Mensagens no ponto N e colocados nas posições $D_1D_2D_3$ do RSOH. É um canal com mensagens para alarmes, manutenção controle, monitoração, administração e outras necessidades entre Funções RST. Canal disponível para mensagens geradas internamente, externamente ou específicas do fabricante do equipamento. Protocolo a ser usado → G.784.

- Bytes reservados para uso nacional/internacional → mesmos comentários já feitos anteriormente.

- Caso seja recebido fluxo de DADOS "TUDO UM" do MST (ou de outro RST, no caso de um regenerador intermediário) no ponto C, então um sinal AIS deve ser aplicado nos dados no ponto B. Este AIS se chama "MULTIPLEX SECTION ALARM INDICATION SIGNAL" (MS-AIS).

3.3.4 Terminação de Seção Multiplex (MST)

Fluxo $C \rightarrow D$

Os dados do STM-N, com quadro baseado no relógio T1, de onde os bytes RSOH já foram recuperados em RST, chegam ao ponto C com relógios $\vec{T1}$. A função MST recupera os bytes de supervisão MSOH e entrega (Dados, Relógios) no ponto D.

Os 3N bytes B_2 de monitoração de erros são recuperados do MSOH. Calcula-se um código BIP-24N para todo o quadro STM-N. O valor calculado BIP-24N para o quadro (k+1) e os erros detectados

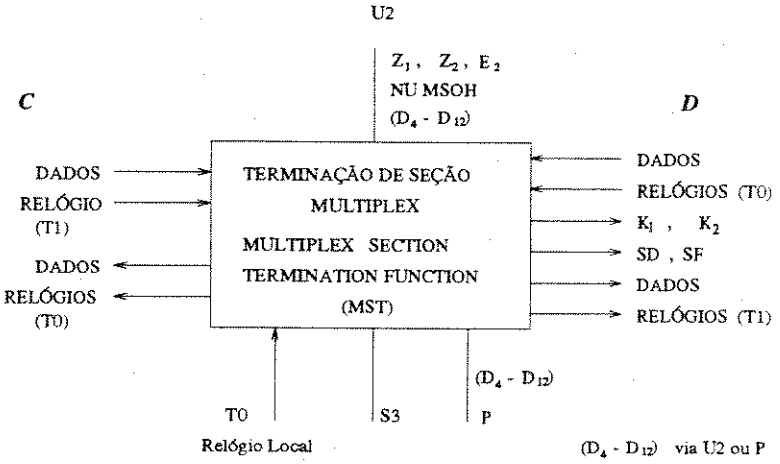


Figura 3.8: Terminanação de Seção Multiplex

são relatados no ponto S3 para filtragens providenciais do SEMF (Gerenciamento do Equipamento). Os erros BIP-24N detectados são também processados no próprio MST, a fim de detectar BER excessivo e degradação do sinal (Signal Degrade SD)

BER	Tempo p/ deteção
10^{-3}	10 ms
10^{-4}	100 ms
10^{-5}	1 s
10^{-6}	10 s
10^{-7}	100 s
10^{-8}	1000 s
10^{-9}	10^4 s

Figura 3.9: Tabela de tempos para deteção de erros

$$\begin{cases} BER\ Excessivo \rightarrow BER > 10^{-3} \Rightarrow signal\ failure\ SF \\ SD \rightarrow limiar\ dentro\ de\ 10^{-9} < BER < 10^{-5} \end{cases}$$

Os requisitos de tempo de detecção de erros estão na tabela da figura 3.9.

- SF e SD devem ser passados para o próximo bloco em D. Além disso, SF e SD devem ser passados para SEMF em S3.
- Os bytes K_1 e K_2 são retirados do MSOH e são passados para o Sistema de Proteção da Seção Multiplex (MSP) em D.
- Os bytes D_4 a D_{12} do DCC da Seção Multiplex (DCC_M) são recuperados do MSOH e passados ao Sistema de Comunicação de Mensagens (MCF) no Ponto P. Estes bytes também podem ser

repassados ao bloco de Acesso de Supervisão OHA no ponto U2. Não existe ainda especificação para estes bytes.

- Outros bytes do MSOH que são passados para OHA: Z_1, Z_2 e E_2

  $E_2 \rightarrow$ order wire Z_i: uso ainda não definido (NZ_1 + NZ_2)

- Um defeito grave na seção multiplex deve ser detectado quando o padrão AIS (111) aparecer nos bits 6, 7 e 8 do byte K_2 em *3 quadros consecutivos.* Remoção da condição AIS na MS $\rightarrow$ quando o padrão recebido nos bits 6, 7 e 8 de K_2 for diferente de (111) em 3 quadros sucessivos.

- Um defeito grave que tenha ocorrido no MS do outro lado é reportado por "110" no bits 6, 7 e 8 do byte K_2 por 3 quadros consecutivos. Para retirar este estado: 3 vezes sem "110"(FERF).

- Além disso, as condições MS-AIS e MS-FERF devem ser reportadas ao SEMF.

- Na verdade: (MS-AIS) ou (BER excessivo) $\Rightarrow \left\{ \begin{array}{l} Dados : Tudo\,UM \\ SD \end{array} \right\}$ em D.

<u>*Fluxo D $\rightarrow$ C*</u>

Os dados no ponto D correspondem a um sinal STM-N temporizado por T0, com carga útil como recomendado em G.709, mas com bytes MSOH e RSOH ainda indeterminados. Os bytes MSOH serão aqui determinados e inseridos nos dados.

- Bytes B_2 calculados e inseridos (BIP-24N)

- Os bytes para Proteção via Chaveamento Automático (APS) que são trazidos do bloco Proteção de Seção Multiplex (MSP) e chegam em D são colocados nas posições K_1 e K_2

  - *Byte* $K_2 \rightarrow$ bits 6, 7 e 8 reservados para proteção do tipo ("nested") nos mux ADM. bits "111" ou "110" não podem aparecer nos bits 6, 7 e 8 de K_2 para função de proteção. pois são reservadas estas palavras para AIS e FERF.

- Os nove bytes D_4/D_{12} enviados pelo MCF são colocados em suas respectivas posições. Estes bytes são considerados um canal de mensagens para alarmes, manutenção, controle, monitoração, administração e outras necessidades de comunicação. Protocolo⇒G.784. Este mesmo canal pode ser usado para mensagens geradas internamente, geradas externamente e outras específicas do fabricante. Estes 9 bytes podem também ser enviadas pelo Acesso de Supervisão OHA via ponto U2 para constituir um canal transparente de dados usando interfaçe externa OHA.

- Outros bytes que vêm do OHA são $\left\{ \begin{array}{l} 3N.Z_1 \\ 3N.Z_2 \\ E_2 \end{array} \right.$

  E_2 é um canal opcional de 64 kb/s não restrito e reservado para comunicação por voz entre terminais.

- Caso-se receba um padrão "TUDO UM" nos dados do ponto D, então um sinal indicativo de alarme de Rota AU (AU PATH AIS) deve ser aplicado nos dados de saída no ponto C.

- Detectando-se Signal Failure (SF) no ponto D, então deve-se aplicar FERF dentro de 250 μs no sinal de saída (DADOS) no ponto C. Um sinal de dados STM-N é um sinal MS-FERF quando tem o código "110" nas posições 6, 7 e 8 do byte K_2.

3.3.5 Proteção de Seção Multiplex (MSP)

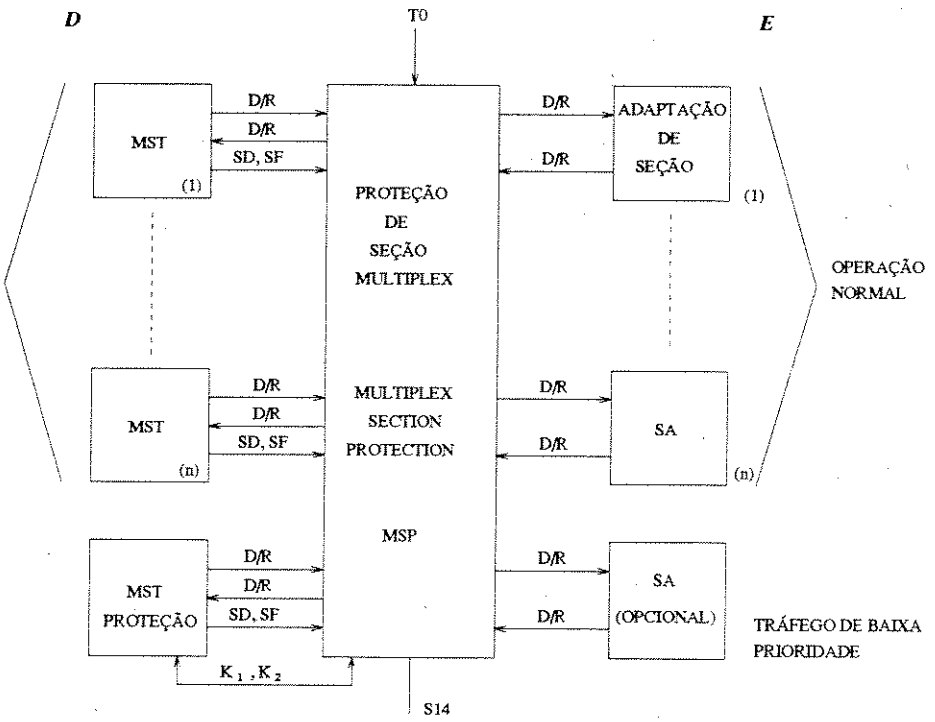


Figura 3.10: Proteção de Seção Multiplex

O Sistema de Proteção da Seção Multiplex (MSP) atua na proteção do sinal STM-N contra falhas associadas ao canal e dentro de Seção Multiplex, ou seja, engloba proteção MST, RST, SPI e ao meio físico, partindo de um MST onde os bytes de supervisão de seção (SOH) são inseridos até o outro MST onde existe a terminação de SOH.

Na figura 3.11 mostra-se o caminho seguido pelo sinal STM-N quando existe falha em uma seção multiplex.

A proteção funciona nas duas pontas da mesma maneira, monitorando sinais STM-N com relação a falhas, avaliando o status do sistema e levando em consideração prioridades estabelecidas e chaveamentos solicitados remotamente. O chaveamento de uma seção de (MST/RST/SPI/REG + Fibra Óptica) para o equipamento semelhante de proteção ocorre dos dois lados.

Os dois sistemas MSP se comunicam através de um protocolo definido para os bytes MSP: são os bytes K_1 e K_2 do MSOH. O protocolo está no Anexo A da Recomendação G.783; não será tratado aqui.

O MSP recebe parâmetros de controle e solicitações externas de chaveamento no ponto S14 através do Equipamento de Gerenciamento (SEMF) e também envia para ele endicadores de status em S14 quando recebe comandos de chaveamento (seção 2 do Anexo A da G.783).

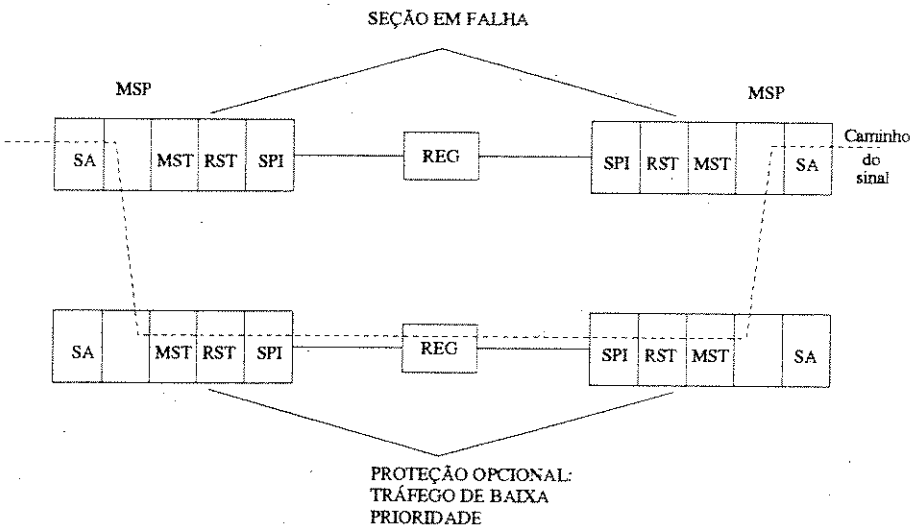


Figura 3.11: Caminho do Sinal STM-N em Caso de Falha na Seção Multiplex

Fluxo $D \rightarrow E$

Dados sincronizados em quadros STM-N e relógios chegam na entrada D. Os bytes de supervisao do RSOH e do MSOH já foram recuperados antes. As condições de falha SF e SD são recebidas em D provenientes de todos os MST dos equipamentos que estão sendo protegidos, juntamente com bytes K_1 e K_2.

Os bytes K_1 e K_2 do MST de Proteção também são recebidos em D (ver figura 3.10).

Em condições normais o MSP passa (Dados + Relógios) de cada MST para o bloco seguinte (que é o de Adaptação - SA). Os (dados + relógios) do MST de Proteção também são passados para um SA de tráfego de baixa prioridade (arranjo opcional) em um arranjo de proteção (1:n).

Caso seja necessário se fazer um chaveamento, então (Dados + Relógios) recebidos do MST de proteção em D são chaveados para o SA em E cuja seção está em falha (ou solicitado remotamente - exercício/manutenção, etc). Os (Dados + Relógios) do MST que estava operando são terminados no MSP.

- Chaveamento de Proteção precisa ser completado em 50 ms após recebimento da condição SD, SF que solicita chaveamento.
- *Restauração do Serviço* - No modo reversível de operação, o canal de trabalho deve ter sua operação restaurada, isto é, o chaveamento deve promover a volta do canal original, quando a seção se recupera da falha. A restauração permite que um outro equipamento operante que entre em falha ou um canal de baixa prioridade usem a seção de proteção.

  Para evitar operação frequente de chaveamento de proteção devido a falha intermitente (BER próximo do limiar de restauração), exige-se que a restauração só ocorra quando a seção em falha ficar isenta de falhas durante um período a ser determinado entre 5 e 12 minutos - este tempo é chamado "Wait to Restore - WTR". Existe porém condição SD e SF que tornam WTR sem efeito.

Fluxo E → D

Dados recebidos em E correpondem a sinal STM-N temporizado em T0 sem os bytes MSOH e RSOH determinados.

- Arquitetura (1+1):
  - Sinais em E provenientes de SA são passados em D permanentemente para ambos MST's do outro lado, um operante e outro de proteção.
- Arquitetura (1:n):
  - Sinal recebido em E de cada SA operante é passado em D ao seu MST correspondente. O sinal de um SA de tráfego de baixa prioridade (opcional) também é passado em D ao seu MST. Quando é necessário fazer uma ponte para proteger um canal de tráfego normal, então o sinal em E deste SA que carrega tráfego normal é conectado em D ao MST de proteção e o tráfego do canal de baixa prioridade é terminado.

Os bytes K_1 e K_2, gerados de acordo com as regras mostradas no Anexo A da G.783 são apresentadas em D ao MST de Proteção e aos outros MST também.

3.3.6 Adaptação de Seção Multiplex (MSA)

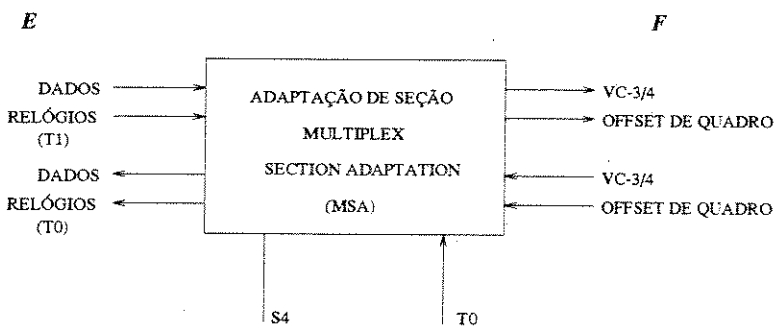


Figura 3.12: Adaptação de Seção Multiplex

Este bloco marca a interface da Terminação de Transporte com as funções internas do multiplex. Adaptação corresponde sempre a montagem e desmontagem de estruturas.

No caso da Adaptação de *Seção* temos neste bloco uma adaptação das Rotas de Alta Ordem em Unidades Administrativas, montagem e desmontagem de Grupamentos de Unidades Administrativas, multiplexagem e demultiplexagem, geração/interpretação/processamento de ponteiro.

Fluxo E → F

Na direção "para dentro do equipamento", as cargas úteis STM-N recebidas em E são desintercaladas e os containers VC-3/4 são recuperados, usando os ponteiros em AU. Em particular a informação contida nos ponteiros é fundamental, principalmente quando se tem offset de quadro

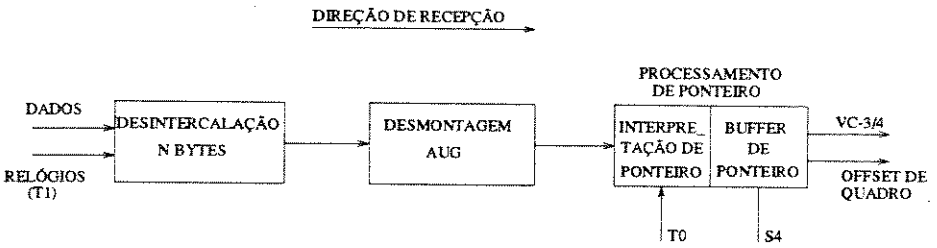


Figura 3.13: Fluxo EF na Adaptação de Seção Multiplex

continuamente variando, devido a recepção de STM-N derivado de fonte plesiócrona em relação ao relógio local de referência.

O processamento de ponteiro provê acomodação do offset do sinal recebido em relação à referência T0, seja o offset devido a *wander* ou a sinal plesiócrono. Obviamente não haverá processamento para T0 = T1, isto é, T0 retirado do sinal que chega.

O processamento de ponteiro pode ser modelado como uma memória elástica onde os dados são escritos na velocidade dada por T1* e são lidos na velocidade dada por T0. T1* significa relógio T1 com buracos sistemáticos e aqueles associados por ajuste de ponteiro → portanto, traduzindo o relógio original do VC, seja ele oriundo de perto ou dos mais variados pontos da rede.

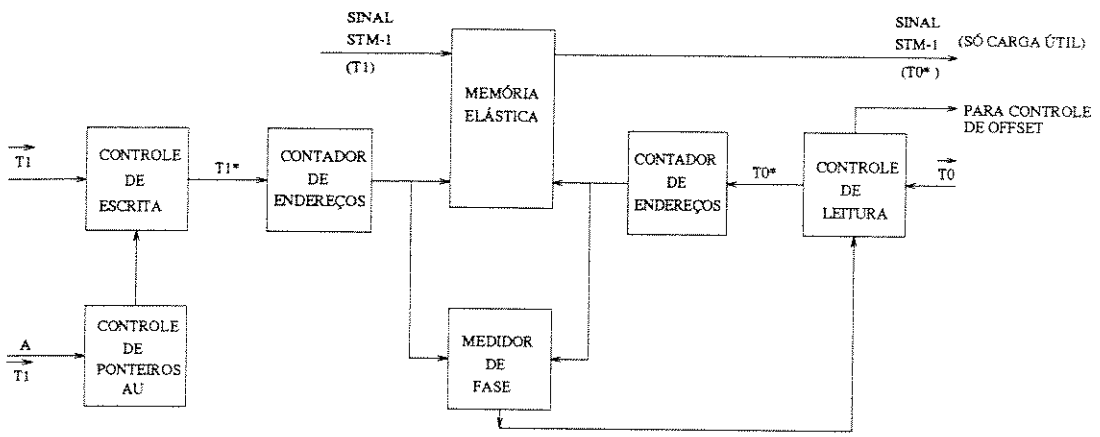


Figura 3.14: Processamento de Ponteiro AU

Quando o relógio T1* de escrita é mais rápido que o relógio T0* de leitura, a memória elástica vai se enchendo. Acontece o contrário, ou seja, a memória vai se esvaziando para T1* mais lento que T0* . Dois limiares de ocupação da memória, um superior S e outro inferior I, determinam quando o ponteiro deve ser ajustado. Uma das funções da memória elástica é reduzir a taxa de ajustes do ponteiro na rede, permitindo pequenas defasagens de relógios sem ocasionar mudanças de offset de quadro.

Quando os dados em um buffer sobem acima de S para um determinado VC, o valor associado do offset de quadro é diminuido em um byte (VC-3) ou 3 bytes (VC-4) e há então uma leitura momentaneamente mais rápida de um byte ou 3 bytes (VC-3 e VC-4 respectivamente). Ao contrário, para os dados esvaziando o buffer e caindo abaixo de I em um VC particular, então o valor do offset de quadro será incrementado de um byte ou 3 bytes (VC-3 ou VC-4) e haverá então uma leitura

momentaneamente mais lenta → um byte deixará de ser lido (VC-3); no caso do VC-4, três bytes não serão lidos.

O CCITT especifica que a memória elástica deve ter um espaço entre os limiares (histerese) de no mínimo de 12 bytes para o AU-4 e 4 bytes para o AU-3 (corresponde a um MRTIE - maximum relative time interval error - de 640 ns entre T0 e T1 do sinal STM-N que chega). Observe que 6,43 ns é o intervalo de 1 bit, de modo que 640 ns corresponde a $\sim$ 100 intervalos $\simeq$ 12 bytes.

O mecanismo de processamento de ponteiro está mostrado na figura 3.15, onde se usou a seguinte notação:

- PE(k) $\equiv$ valor do Ponteiro de AU dentro do equipamento
- PR(k) $\equiv$ valor do Ponteiro de AU recebido
- PME(k) $\equiv$ valor de saída da Memória Elástica do Ponteiro AU
- PT(k) $\equiv$ valor do Ponteiro de AU Transmitido
- D(k) $\equiv$ Dados
- NDF(k) $\equiv$ bits New Data Flag transmitidos no Ponteiro AU
- SSME(k) $\equiv$ bits SS de concatenação na saída da Memória Elástica
- SS(k) $\equiv$ bits SS transmitidos no Ponteiro de AU
- I/D → inverter bits I ou D do Ponteiro AU

Observação - A indicação de concatenação deve ser interpretada aqui. Por exemplo, o primeiro AU-4 de uma carga útil concatenada AU-4-Xc deve ser interpretado de acordo com o fluxograma da figura 3.15; para os outros AU-4, os ponteiros contém indicativos de concatenação e o processador de ponteiro deve realizar as mesmas operações aplicadas ao AU-4 lider.

Lembrar ainda que o fluxograma é aplicável para cada AU que compõe o AUG; além disso, ao analizar o fluxograma, verificar que o processamento relativo a Geração de ponteiro se dá *na direção de transmissão* em outra Adaptação de Seção, agora com temporização T0.

O algoritimo para deteção de ponteiro, mostrado na fluxograma da figura 3.15, está definido no Anexo B da recomendação G.783. Duas condições de falhas podem ser detectadas pelo interpretador de ponteiro:

- Perda de Ponteiro (Loss of Pointer - LOP)
- Rota AU com AIS

Se uma ou ambas condições são detectadas, então "TUDO UM" deve ser aplicado em F. Estes defeitos devem ser reportados em S4 para interpretação/providências pelo SEMF. Os eventos de justificação são também monitorados em S4

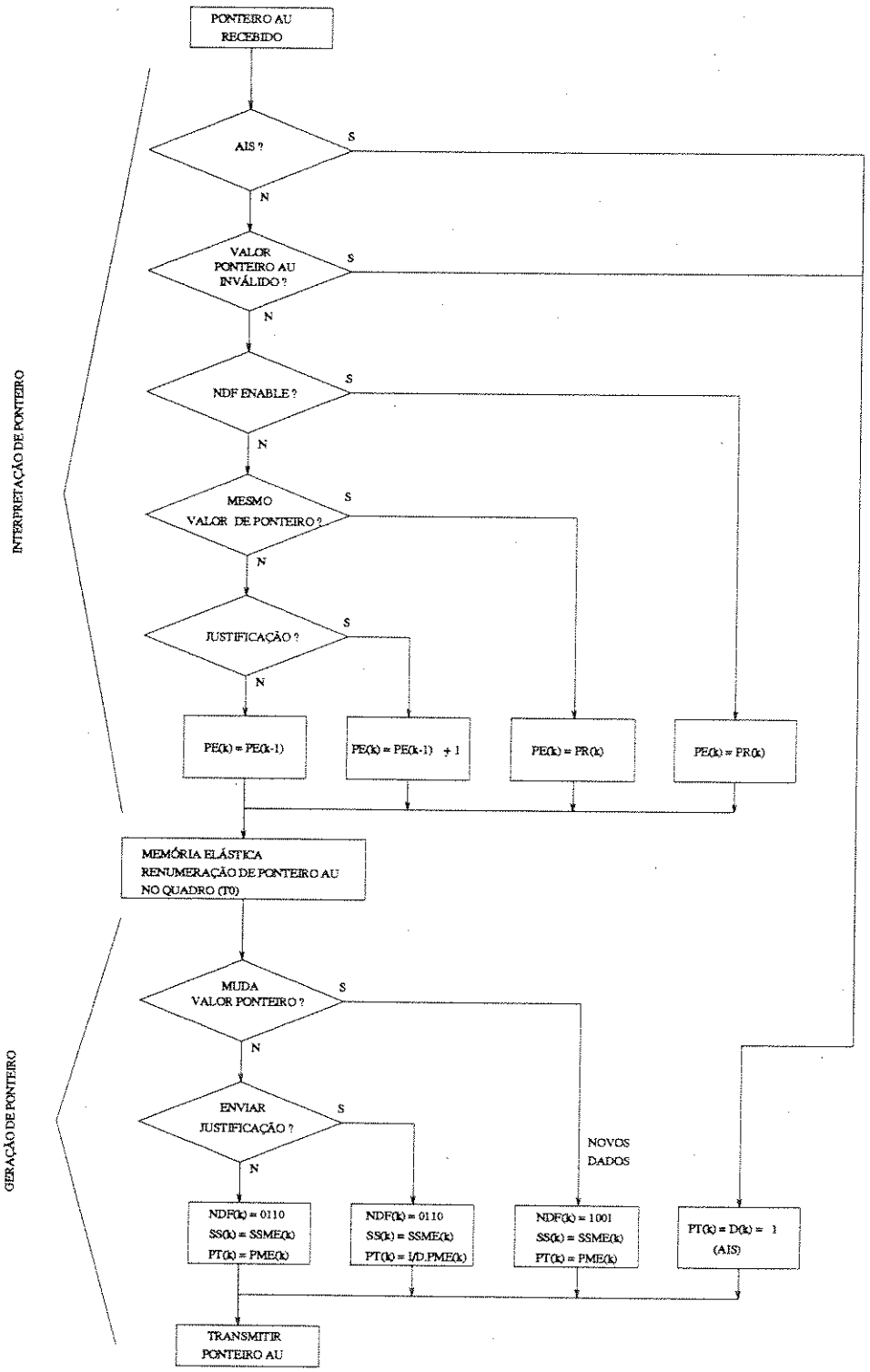


Figura 3.15: Fluxograma de Processamento de Ponteiro

Fluxo $F \rightarrow E$

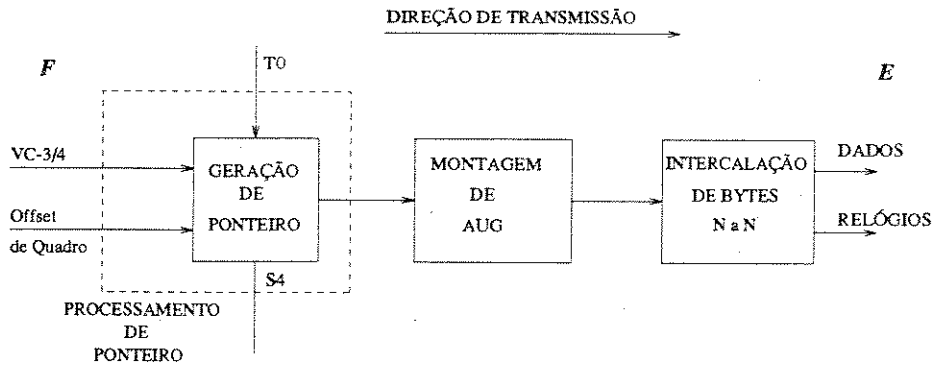


Figura 3.16: Fluxo FE na Adaptação de Seção

Os container de Alta Ordem que chegam em F são mapeados em AU's. Estes AUG's são intercalados N a N (intercalação de bytes) e entregues no ponto E. Por exemplo, para transmissão através de um Módulo Síncrono de Transporte de Nível 4, STM-4, teríamos à esquerda 4 entradas independentes de (VC-4 + offset), montaríamos 4 AUG independentes e faríamos em seguida a intercalação dos 4 AUG's byte a byte.

A informação de Offset de Quadro é usada pelo Gerador de Ponteiro para gerar ponteiro de acordo com regras da recomendação CCITT G.709.

Dados STM-N em E estão temporizados por T0, o relógio local. Para D(k) = $\vec{1}$ em F, isto irá provocar valor inválido de offset de quadro (valor 1024) $\rightarrow$ perda de Ponteiro AU, então AIS deverá aparecer também no valor do ponteiro em E.

3.4 Funções das Rotas de Alta Ordem

A descrição dos bytes de supervisão de rota é um pré-requisito para o entendimento das Funções da Rotas de Alta Ordem. Portanto, eles são descritos a seguir.

CARGA DE SUPERVISÃO DE ROTA (POH) PARA VC-3 E VC-4

A carga de supervisão de rota (POH) tem a mesma estrutura de uma coluna de 9 bytes tanto para o VC-3 como para o VC-4. A figura 3.17 mostra a localização dos bytes de POH nos containers VC-3 e VC-4.

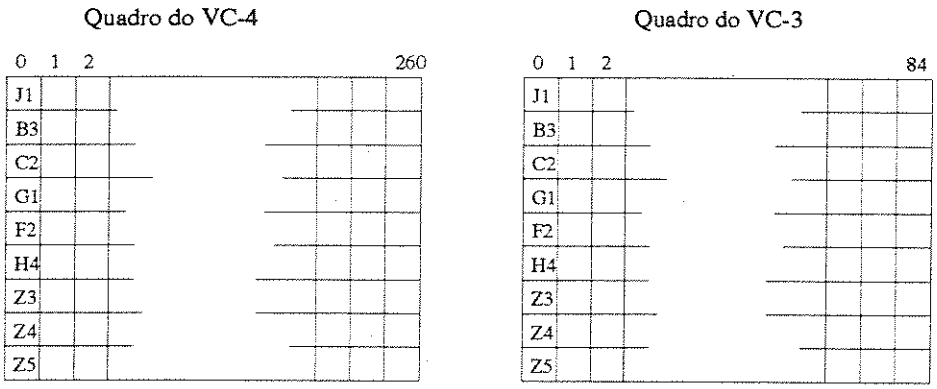


Figura 3.17: Localização dos Bytes de Supervisão de Rota (POH).

J1 → Rastreamento (Path Trace)

O byte J1 proporciona validação de rota e função de rastreamento. Um identificador único é inserido no byte J1 pela função de terminação de rota na origem. Este byte é recuperado na outra ponta da rota e comparado com o valor esperado, que pode estar armazenado localmente, ou então enviado ao sistema de gerenciamento para análise.

A especificação de mensagem em J1 é uma sequência de 64 bytes, mas o formato preferencial é a opção ASCII de 15 bytes especificada em E.164, precedida por um "flag" de início de quadro de um byte, formando portanto um padrão de 16 bytes.

B3 → Monitoração da Taxa de Erro

B3 propicia uma função de monitoração de erro, usando o CRC especializado chamado BIP-8 (Bit Interleaved Parity - depth 8). O BIP-8 é calculado ao longo do quadro do VC anterior e inserido no byte B3 na função de terminação de Rota na fonte. O receptor remoto faz uma computação idêntica àquela feita na fonte, de forma independente, e compara o B3 calculado localmente com o byte recebido. As violações detectadas podem ser reportadas separadamente ou haver simplesmente indicação de erro no bloco recebido com uma ou mais violações de paridade.

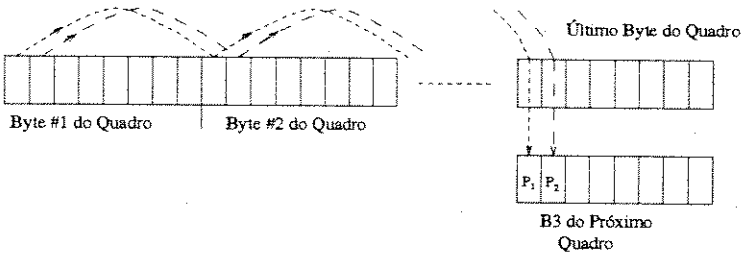


Figura 3.18: Soma-se módulo 2 todos os bits #i de um byte e coloca-se no bit #i de B3 o resto que completa zero (paridade par)

C2 → Nota de Despacho (Path Label)

O byte C2 carrega informação a respeito da composição da carga útil. Dois valores para C2 já foram especificados dos 256 possíveis: 00000000 é enviado por uma função de terminação de rota se a função de adaptação estiver não equipada ou a partir de uma função de conexão transversal (crosconnecting function) e não houver conexão transvesal; 00000001 é enviado para indicar que a função de adaptação está equipada. De qualquer modo C2 provê a possibilidade de acertar opções de adaptações remotamente.

G1 → Status da Rota

O byte G1 provê informação de status do destino (Terminação Remota da Rota de Destino). O formato é mostrado na figura 3.19.

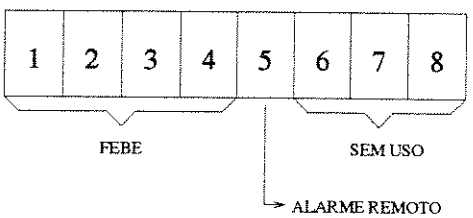


Figura 3.19: Byte G1.

FEBE→ Retorno da contagem de violações do BIP-8 remoto como um número binário. Existem 9 valores válidos, ou seja: 0 a 8 violações.

Bit#5→ Far End Receiver Fail (FERF): toma o valor 1 quando falha é detetada, 0 sem falha. O critério para enviar FERF é a) deteção de AIS, b)descasamento na validação da rota (J1) - ou seja, falha na rota.

Byte H4

Específico da carga útil → será descrito no capítulo 5.

As Rotas de Alta Ordem foram definidas de acordo com os containers virtuais VC-3 e VC-4. Estes VC's podem ser criados de duas maneiras:

1. por mapeamento direto em AU's (o que pode ocorrer para 34 Mb/s no VC-3, 140 Mb/s no VC-4 e outras grandes cargas especiais);
2. por mapeamento indireto de sinais de baixa ordem em TU's, que são então mapeados em AU's.

Estas duas possibilidades estão mostradas no diagrama geral da figura 3.1. As Funções das Rotas de Alta Ordem envolvem:

- CONEXÃO DE ROTAS DE ALTA ORDEM
- TERMINAÇÃO DAS ROTAS DE ALTA ORDEM
- ADAPTAÇÃO DAS ROTAS DE ALTA ORDEM

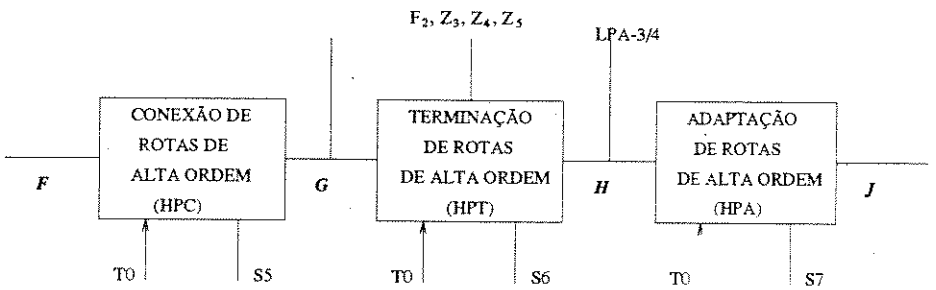


Figura 3.20: Funções Das Rotas de Alta Ordem

Na figura 3.20, mapeamento direto ocorre em H, através de Adaptação de Rotas de Baixa Ordem com n = 3 e 4. Para o mapeamento dos containers de Baixa Ordem que são menores, a interface é pelo ponto J.

- Conexão envolve comutação semi-permanente
- Terminação envolve FVI, com monitoração de parâmetros
- Adaptação envolve montagem e desmontagem de estruturas

3.4.1 Conexão das Rotas de Alta Ordem (HPC)

A função HPC-n consigna containers virtuais de alta ordem montados a espaços VC-n disponíves em uma seção MUX. A inclusão ou não de HPC em um multiplex possibilita ou não comutação semi-permanente de cargas entre rotas de mesma terminação. Existem alguns tipos de MUX SDH que não possuem a função HPC, como será visto futuramente.

Deste modo, containers VC-n que estão chegando em F são consignados a espaços VC-n disponíveis em G; e, reciprocamente, VC-n's que chegam em G são consignados a ocuparem espaços

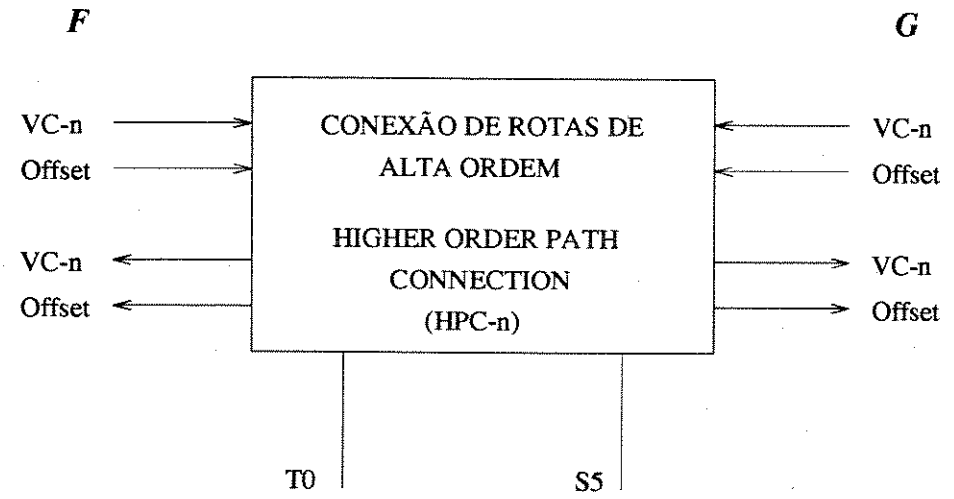


Figura 3.21: Conexão de Rotas de Alta Ordem

VC-n's disponíveis em F. Portanto os sinais em F e em G são semelhantes, diferindo entre si somente na sequência lógica de VC-n's.

A consignação biunívoca que ocorre de um lado para o outro $G \leftrightarrow F$, define um padrão de conexão (connection pattern) que pode ser descrito por uma matriz de conexão de 2 colunas CM(V_i,V_j), onde V_i identifica o $i^{ésimo}$ canal de container virtual no ponto F e V_j identifica o canal VC $j^{ésimo}$ no ponto G. Para alguns padrões de conexão, V_j é identificado por parâmetros adicionais k e l que indicam a $k^{ésima}$ porta em l portas tributárias (esclarecimento posterior).

Através do equipamento de gerenciamento SEMF é possível articular as seguintes opções com a matriz de conexão:

- consignar uma determinada saída (SEMF→HPC-n)
- requerer relatório da matriz (SEMF→HPC-n)
- reportar CM (HPC-n→SEMF)

A temporização de HPC-n é através de T0.

Dependendo do tipo de multiplex, pode-se ter uma flexibilidade maior ou menor no padrão de conexão. Na verdade existem alguns tipos de multiplex que não possuem a função de conexão através de matriz; as conexões são fixas.

A figura 3.22 mostra um exemplo de Matriz de Conexão.

Fluxo $G \rightarrow F$ (Semelhante $F \rightarrow G$)

HPC-n consigna HVC-n montados que chegam de um lado a espaços disponíveis HVC-n do outro lado. Esta correspondência é baseada no padrão de conexão estabelecido, que pode ser fixo ou configurável.

HPC–n para ADM

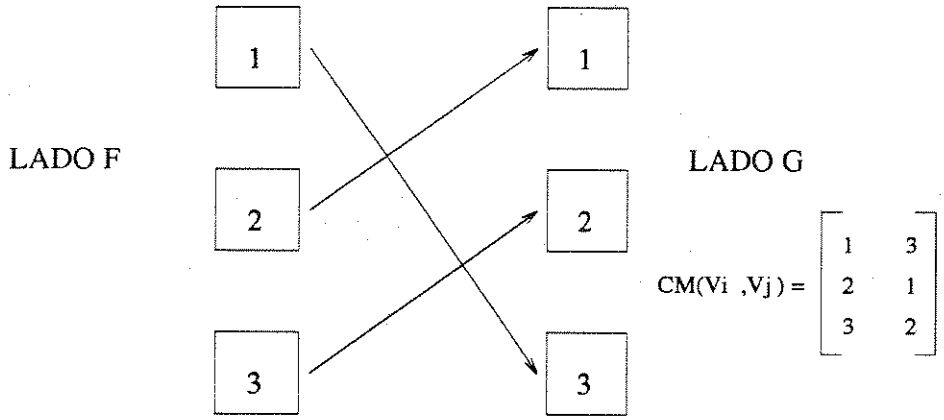


Figura 3.22: Exemplo de Matriz de Conexão

Para multiplex que desempenha as funções de inserir (Add) ou retirar (Drop) canais [Add-Drop Mux], HPC-n é uma matriz de conexão com as caracteristicas mostradas na figura 3.23.

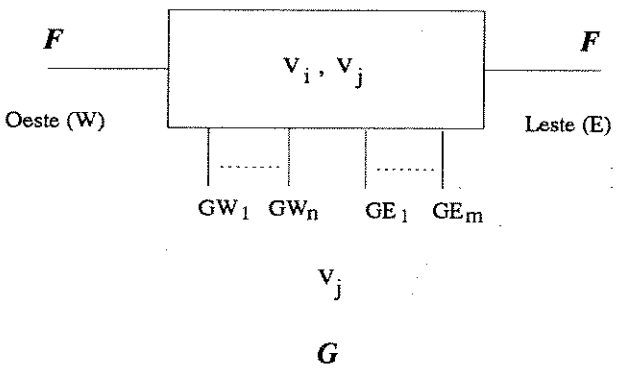


Figura 3.23: HPC-n para ADM

Nos lados Oeste e Leste estão as cabeças ópticas (ponto F, terminação de transporte $\rightarrow$ saídas STM-N para ambos os lados). As terminações de entrada/saidas (Add-Drop points) $GW_1...GW_n$ e $GE_1...GE_m$ em geral suportam VC-n's de baixa capacidade.

No caso mais geral de ADM uma função de conexão transversal (cross-connect) será desempenhada por HPC, quando então qualquer Vi em F.Oeste e F.Leste pode ser retirado por qualquer V_j em G nas terminações $GW_1...GW_n$ e $GE_1...GE_m$. A figura 3.24 mostra um exemplo de Padrão de Conexão.

HPC–n para MUX diferente do tipo ADM

Neste caso o lado F tem capacidade de VC-n dada pelo módulo STM-M. As terminações G_1, G_2, ..., G_l, cada uma suporta uma capacidade em VC-n equivalente a um STM-N, onde M>N. A capacidade total G_1 a G_l não pode exceder a capacidade em F.

Na matriz de conexão $CM(V_i, V_{jk})$ destes multiplexadores, V_i identifica um canal VC-n em F e V_{jk} identifica o $j^{ésimo}$ canal VC-n em G_k com k = 1, 2 ... l. Isto requer que um determinado canal VC-n V_{jk} em G seja conectado a um determinado canal V_i em F.

Além dos tipos mostrados de matrizes, existem alguns tipos de MUX que não necessitam de

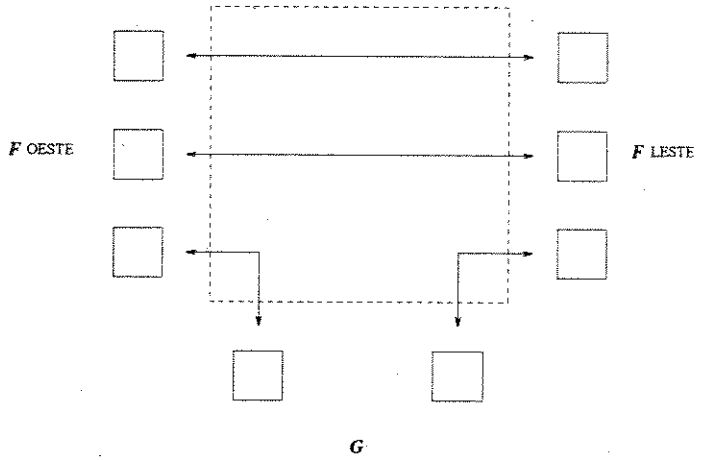


Figura 3.24: Exemplo de Padrão de Conexão

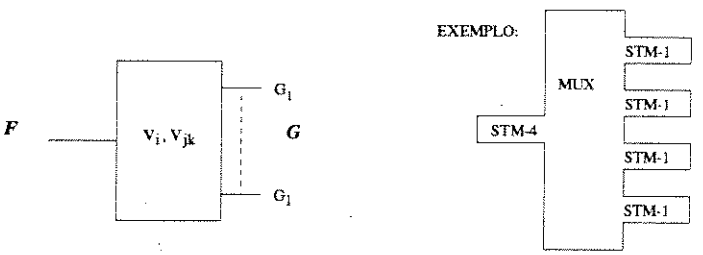


Figura 3.25: HPC-n para MUX diferente do Tipo ADM

HPC-n como veremos posteriormente.

3.4.2 Terminação da Rotas de Alta Ordem (HPT)

Como no caso MST, a Terminação de Rotas de Alta Ordem é Fonte e Destino para a carga de supervisão de rotas de Alta Ordem (VC-n POH, n = 3 e 4).

Uma rota de Alta Ordem é uma entidade de manutenção definida entre duas terminações de rotas de alta ordem.

A temporização da HPT é T0.

Fluxo $G \rightarrow H$

Dados em G é um VC-n (n = 3, 4) tendo sua carga útil e sua carga de supervisão POH com bytes J_1, B_3, C_2, G_1, F_2, H_4, Z_3, Z_4, Z_5. Observe que estamos descrevendo a função para um container; este fluxo se repete para todos os HVC. Os bytes de POH são recuperados como parte da função HPT e o VC-n é enviado para a saída H.

- Bytes J_1, C_2, G_1 recuperados:

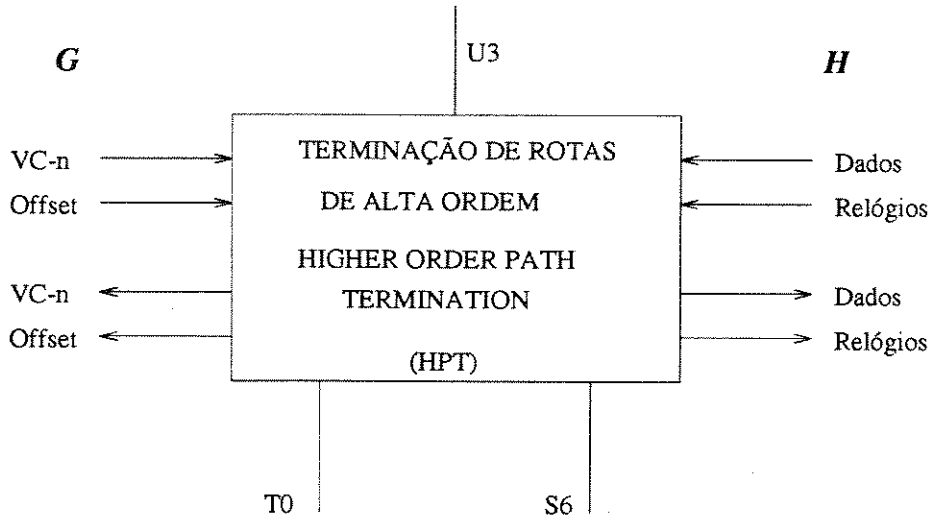


Figura 3.26: Terminação de Rotas de Alta Ordem

$$\begin{pmatrix} J_1 \\ G_1 \\ C_2 \end{pmatrix} \equiv \begin{pmatrix} Rastreamento \\ Status\ de\ Rota \\ Nota\ de\ Despacho \end{pmatrix} \rightarrow passado\ via\ S6\ para\ SEMF$$

- *Do byte G_1*

  FEBE decodificado dos bits (1 a 4)→ Path Termination Error Report em S6.

  FERF decodificado no bit 5 (valor 1)→ relatado como Remote Alarm Indication em S6.

- *H_4*

  No caso de cargas úteis que requerem alinhamento de multi-quadro, um indicador de superquadro é retirado do byte H4. Existe um processo de verificação da fase de H_4 ao longo dos quadros recebidos. Caso vários valores de H_4 sejam recebidos não corretamente ao longo do tempo em forma consecutiva então deve ser reportado em S6 Perda de Superquadro (Loss of Multiframe - LOM). Para cessar LOM é necessário receber muitos H_4 na forma correta (tempo pensado na faixa de 2 a 10 ms).

- *Byte B_3*

  O byte B_3 de monitoração de erro é recuperado do quadro VC-n e é computado BIP-8 para o quadro recebido. BIP-8 computado no quadro (k-1) é comparado com o byte B_3 recuperado no quadro (k) e os erros são reportados no ponto S6 como quantidade de erros dentro de B_3 por quadro. Isto permite monitoração de desempenho através do SEMF (Supervisão).

- *F_2, Z_3, Z_4, Z_5*

  F_2 é colocado para comunicação do usuário (operadora). Ele é retirado do POH e passado para o Acesso de Supervisão (OHA) em U3. Os 3 bytes Z_3, Z_4 e Z_5 também são passados para OHA via U3. São reservados para uso futuro.

Fluxo H → G

Dados em H é um VC-n (n = 3 ou 4) tendo uma carga útil de acordo com a recomenda,ão G.709, mas com carga de supervisão de Rota (POH) ainda indeterminada. O processamento se dá em todos os VC-n's presentes em H (aqui descrevemos o que ocorre com um). O processamento dos bytes de POH e a sua colocação junto a carga útil para formar um HVC são partes da função HPT que envia o HVC completo para G.

- Rastreamento (Path Trace), Status da Rota e Nota de Despacho (signal label) são recebidos em S6 e colocados nas posições dos bytes J_1, G_1 e C_2 respectivamente.

  Se um relatório de "erro de terminação de rota" (path termination error) indicar um bloco com erro, então os bits 1 a 4 de G_1 são codificados de acordo com as especificações da G.709 para enviar FEBE. Além disso se receber em G um "AU Path AIS", ou seja Rota de Unidade Administrativa com sinal indicativo de alarme, então uma indicação FERF deve ser enviada pelo bit 5 do byte G_1.

- Paridade Intercalada de Bit (BIP-8) é computada sobre todos os bits do quadro k e colocado na posição do byte B_3.

- Indicador de Super Quadro é gerado como recomendado na rec. G.709 e colocado na posição do byte H_4.

- $F_2 \Rightarrow$ Retira-se a informação de U3 e coloca-se a mesma na posição do byte F_2. Estes 64 kb/s são reservados para comunicação dos usuários (operadora). O mesmo ocorre para Z_3, Z_4 e Z_5 que são bytes reservados para usos futuros.

3.4.3 Adaptação das Rotas de Alta Ordem (HPA)

A Função de Adapatação de Rotas de Alta Ordem define o processamento de ponteiro das unidades afluentes (TU). Corresponde à montagem e desmontagem dos containers Alta Ordem↔Baixa Ordem.

A Adaptação se dá através de 3 funções:

- Geração de Ponteiro (direção Baixa Ordem→Alta Ordem)
- Interpretação de Ponteiro (na desmontagem Alta Ordem→Baixa Ordem)
- Justificação de Frequência

1. Observar que o esquema da figura 3.27 está invertido: Entramos do lado STM-N e, para dentro do equipamento, estaríamos chegando agora em H, depois de HPT.
2. Funções semelhantes às mostradas acima já foram analisadas na Adaptação de Seção (AUG↔HVC). Aqui teremos (LVC↔HVC).

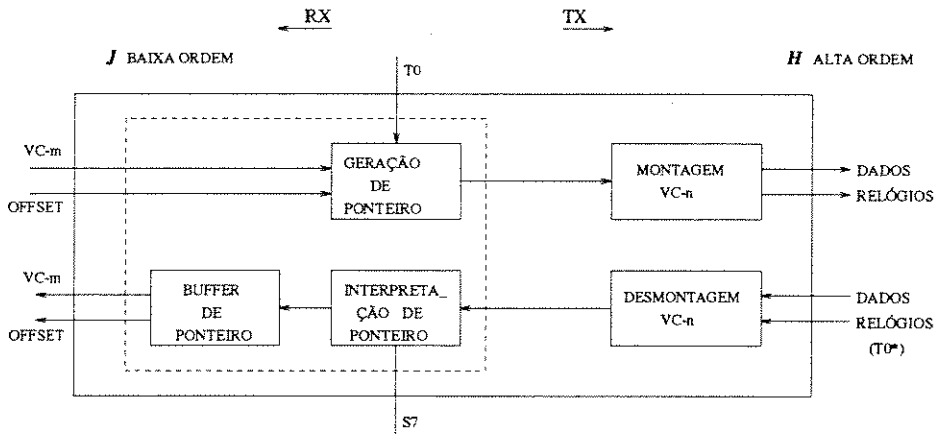


Figura 3.27: Adaptação de Rotas de Alta Ordem

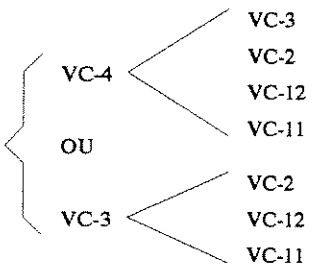


Figura 3.28: Desmontagem de HVC em LVC

Fluxo H → J (DESMONTAGEM)

Nesta direção o Adaptador HPA desmonta os HVC's nos LVC's, como mostrado na figura 3.28.

Então o ponteiro TU de cada container de baixa ordem LVC é decodificado para se obter informação sobre o offset de quadros (em bytes) entre o HVC e o específico LVC. Este processo deve permitir ajustes frequentes de ponteiro, pois a frequência do relógio do nó de origem onde o LVC foi gerado pode ser bem diferente do relógio T0 local. A máxima diferença de frequência esperada na rede afeta o tamanho da memória elástica que faz parte do desincronizador (que é semelhante àquele da figura 3.14). O CCITT especificou que a máxima diferença relativa de frequência na rede deve ser de $(\Delta f/f) = 4,6 \times 10^{-6}$ (4,6 ppm), para relógios operando no modo "free-running". Abordaremos o problema de sincronização futuramente.

Portanto, em J teremos containers de baixa ordem, cada um com sua taxa original e algum possível jitter embutido (ainda é tudo digital). Observar que aqui, o relógio de entrada T0* e seus ajustes de Ponteiro traduzem o relógio original do container e T0 local é o relógio portador do LVC para o terminal local do container.

O algoritimo para deteção de ponteiro é o mesmo já visto anteriormente. Medidas de LOP e TU Path AIS. Gerenciamento através de S7.

Fluxo J → H (MONTAGEM)

A função de Adaptação HPA monta os VC's de baixa ordem como Unidades Afluentes (TU's) dentro do VC de alta ordem de saída.

O offset de quadro (em bytes) entre o LVC e o HVC que o suporta é indicado por Ponteiro TU: o método de geração de ponteiro já foi discutido. Observe que se o LVC que está sendo assemblado no HVC tiver sido gerado localmente, a posição do ponteiro vai ficar fixa durante todo o tempo.

3.5 Funções das Rotas de Baixa Ordem

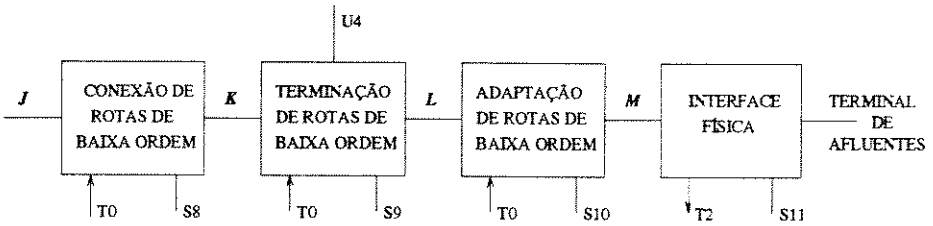


Figura 3.29: Funções Das Rotas de Baixa Ordem

As recomendações G.708 e G.709 definem 5 capacidades básicas de containers C-11, C-12, C-2, C-3 e C-4. Por outro lado, concatenações de C-2 permitem a criação de capacidades VC-2-Xc, com X variando de 1 a 21 (Pois 21 VC-2's completam um VC-4).

Os sinais dos usuários são adaptados em containers, que são então alocados em rotas de alta ordem.

Para grandes cargas economiza-se uma adaptação intermediária, razão pela qual o diagrama de blocos da figura 3.1 possui um caminho separado para grandes cargas (a ser visto a frente).

3.5.1 Conexão de Rotas de Baixa Ordem (LPC)

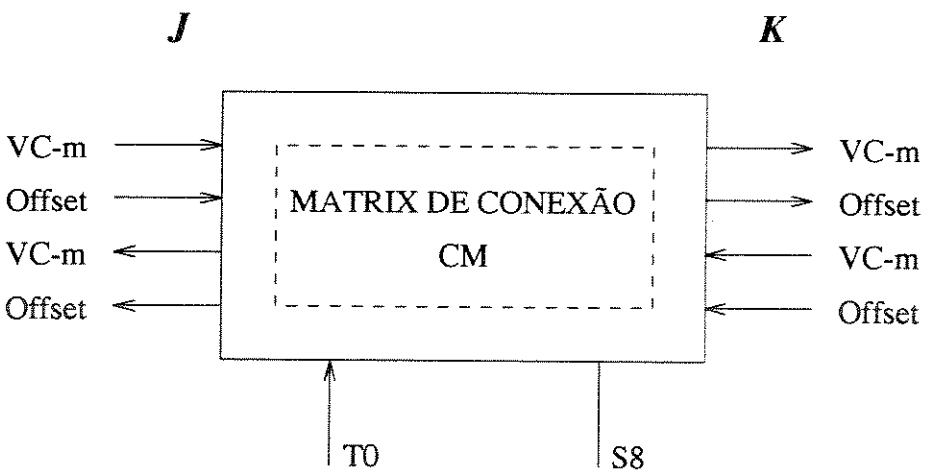


Figura 3.30: Conexão de Rotas de Baixa Ordem

A Função de Conexão de Rotas de Baixa Ordem corresponde a designação de containers VC de nível m (m = 1, 2, 3) a espaços disponíveis VC-m em rotas de alta ordem (ver HPA onde esta função interfaceia). Alguns tipos de multiplex não possuem esta função de comutação. No MUX tipo ADM é necessário o uso de LPC para permitir topologias de rede tipo BUS ou ANEL.

Os VC's que chegam em J são designados para espaços disponíveis VC do lado K e vice-versa. Os formatos dos sinais em J e K são portanto semelhantes, mudando-se somente a sequência lógica dos VC's dos dois lados. Aqui também define-se um padrão de conexão, caracterizado por uma matriz

de conexão CM(V_i, V_j), onde V_i identifica o $i^{ésimo}$ canal VC em J e V_j identifica o $j^{ésimo}$ canal VC em K. Os multiplex podem ser caracterizados pela matriz de conexão.

No ponto S8 pode-se comunicar com a matriz de conexão, dentro dos seguintes comandos:

- SET MATRIZ → designa determinado terminal de acordo com a matriz de conexão (SEMF→LPC)
- Request CM Report (SEMF→LPC)
- Report CM (LPC→SEMF)

Fluxo $J \rightarrow K$ (semelhante $K \rightarrow J$)

LPC designa VC-m's montados chegando em K e espaços disponíveis VC-m's do lado J para entrarem nas cargas dos containers de alta ordem. A designação é baseada na matriz de conexão que pode ser fixa ou configurável.

Matriz para ADM

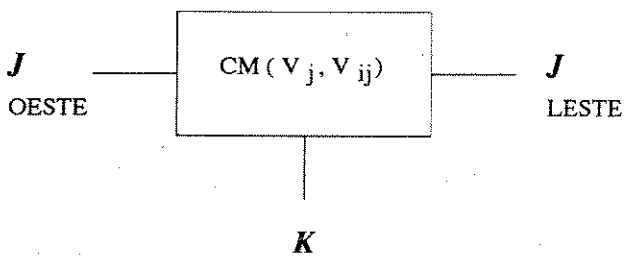


Figura 3.31: Matriz de Conexão para ADM

Os sinais nos terminais J.Oeste e J.Leste suportam (cada um) uma capacidade VC-m equivalente às rotas de alta ordem que devem ser acessadas. Do lado K deve-se ter capacidade menor ou igual. A função de conexão permite a retirada de VC-n's de J.Oeste e J.Leste para K sem modificar o tráfego global. Ou ainda a inserção de VC-n's em J.Oeste e J.Leste provenientes de K.

Descrição da matriz de conexão:

$$\begin{cases} V_j & j^{ésimo}\ canal\ VC{-}n\ do\ ponto\ K\ (baixa\ ordem) \\ V_{ij} & j^{ésimo}\ canal\ VC{-}n\ em\ J.Oeste\ para\ i = 1 \\ ou & j^{ésimo}\ canal\ VC{-}n\ em\ J.Leste\ para\ i = 2 \\ ou & j^{ésimo}\ canal\ VC{-}n\ em\ J.Oeste\ e/ou\ J.Leste\ para\ i = 3 \end{cases}$$

Ou seja na direção $K \rightarrow J$(Oeste, Leste) a transmissão ocorre em ambos os canais, enquanto que na direção J.Oeste/J.Leste→ K seleciona-se o canal J.Oeste ou J.Leste

Para i = 3 pode-se operar o multiplex em topologia de anel, com proteção de rota garantida por roteamento alternativo e sem intervenção das funções de camadas mais altas da rede. Veja figura 3.33.

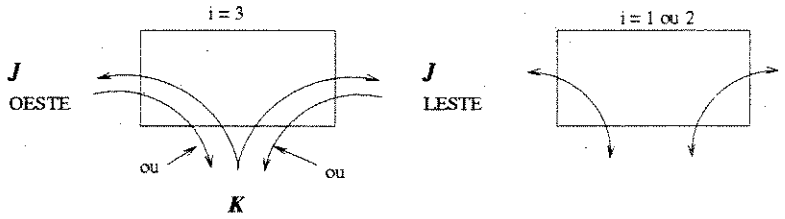


Figura 3.32: Matriz de Conexão com i=3 e i= (1 ou 2)

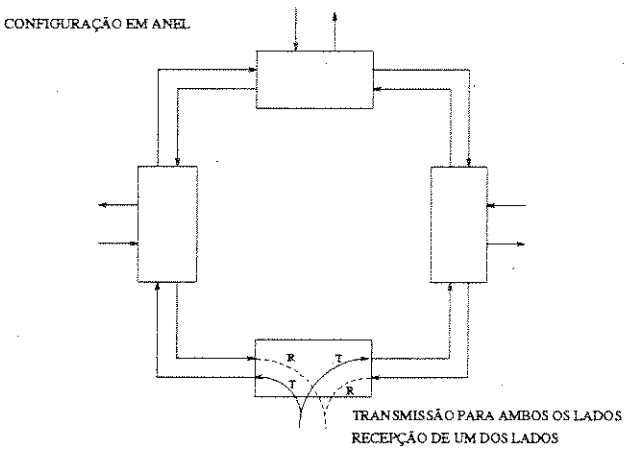


Figura 3.33: Configuração em Anel

3.5.2 Terminação de Rotas de Baixa Ordem (LPT)

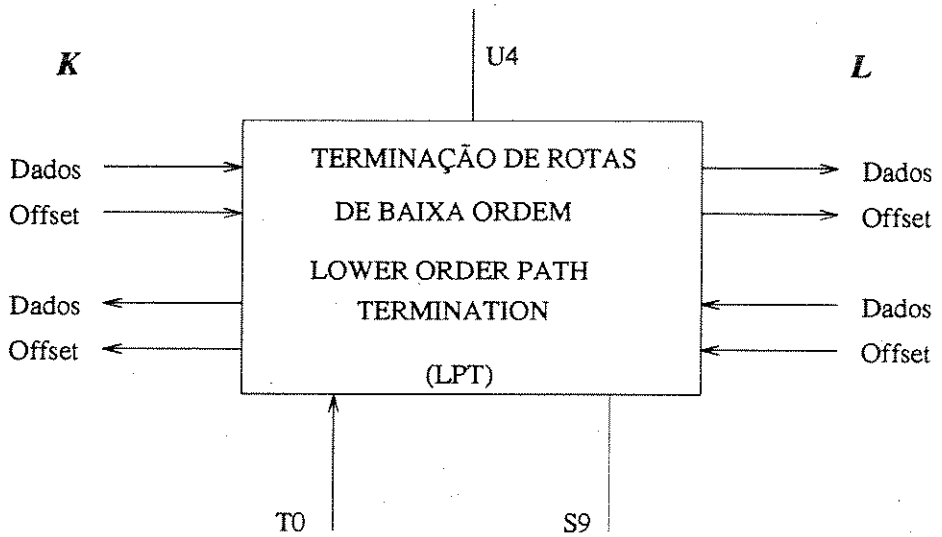


Figura 3.34: Terminação de Rotas de Baixa Ordem

A função da Terminação de Rotas de Baixa Ordem (LPT) cria, em uma direção, Container Virtual VC-m (m = 1, 2 ou 3) através da geração e adição de Supervisão de Rota (POH) a um Container C-m. Na outra direção de transmissão o LPT é uma terminação de POH que aí é processada para monitorar certos atributos da Rota.

No ponto L os Dados são alocados a um container C-m (m = 1, 2 ou 3) que é sincronizado com T0. o relógio local. O sinal de informação que foi mapeado sincronamente (adaptação com justificação de bit) em um container e o respectivo offset de quadro do container são recebidos em L. Adiciona-se POH aos Dados e estes novos Dados são passados ao ponto K, juntamente com o Offset de quadro.

Carga de Supervisão-Para containers VC-m com m = 1 ou 2, a carga de supervisão é transportada no byte V5; geração e terminação de POH para estes casos é visto a seguir.

Para container VC-3, as funções de terminação já foram analizadas em HPT - Terminação de Rotas de Alta Ordem.

Fluxo $K \rightarrow L$

- TU Path AIS recebido em $K \rightarrow$ colocar condição TUDO UM nos Dados em L e reportar Path AIS em S9 (Sinal Indicativo de Alarme na Rota). Alem disso, uma indicação de FERF (Far End Receiver Failure) deve ser enviada no bit 8 nos dados na direção contrária.
- A deteção dos bits 5, 6 e 7 (Nota de despacho - Signal Label) deve ser feita e reportada em S9.
- Deve-se recuperar os bits de monitoração de erro 1 e 2 (BIP-2). Alem disso deve-se computar BIP-2 para o quadro do VC e compará-lo com os bits 1 e 2 do próximo quadro; o número de erros detectados (0, 1 ou 2) no bloco deve ser relatado em S9 como Relatório de Erro de Terminação de Rota (Path Termination Error Report).
- FEBE (Far End Block Error) deve ser recuperado no bit 3 e reportado em S9.

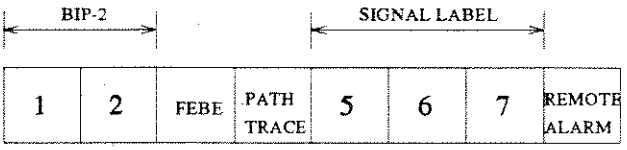


Figura 3.35: Byte de Supervisão de Rotas de Baixa Ordem - V5

- FERF de Rota deve ser detectado no bit 8 e relatado como Indicação de Alarme Remoto (Remote Alarm Indication) em S9.
- O bit 4 (Rastreamento - Path Trace) deve ser detectado e passado para S9.

Fluxo $L \rightarrow K$

Esta é a direção da montagem:

- Signal Label em S9 $\rightarrow$ bits 5, 6 e 7 de V5.
- BIP-2 computado no quadro (k-1) e emviado pelos bits 1 e 2 de V5 no quadro k.
- Se houver deteção na outra direção de erro no BIP-2 e, como consequência, um Path Termination Error Report, então o bit 3 de FEBE deve ser igual a 1 no próximo quadro.

3.5.3 Adaptação de Rotas de Baixa Ordem (LPA)

A função de Adaptação de Rota de Baixa Ordem opera no terminal de acesso da rede síncrona e faz a adaptação dos dados do usuário para transporte em forma síncrona. Para usuário assíncrono esta adaptação envolve justificação de bit.

Mapeamentos de containers grandes (VC-3 e VC-4) são feitos diretamente por grandes cargas úteis de informação (isto pode ser visto na figura 3.1).

A função de Adaptação mapeia sinais da recomendação G.703 em containers de baixa ordem que são em seguida mapeados em containers de alta ordem.

Observação: sinais DS1 (1,5 e 2 Mb/s) podem ser mapeados diretamente em HVC's usando mapeamento amarrado (locked mode).

As funções de Adaptação de Rotas de Baixa Ordem (LPA) são definidas para todos os níveis existentes na hieraquia plesiócrona (PDH). Cada LPA define de que modo cada sinal do usuário pode ser mapeado dentro de um container síncrono C de tamanho apropriado.

Deste modo temos:

- LPA-11 $\rightarrow$ mapeamento $\left\{ \begin{array}{l} \textit{assíncrono bit a bit} \\ \textit{síncrono bit a bit} \\ \textit{síncrono byte a byte} \\ \textit{modo amarrado} \end{array} \right\} \rightarrow$ container C-11 (1,544 Mb/s)

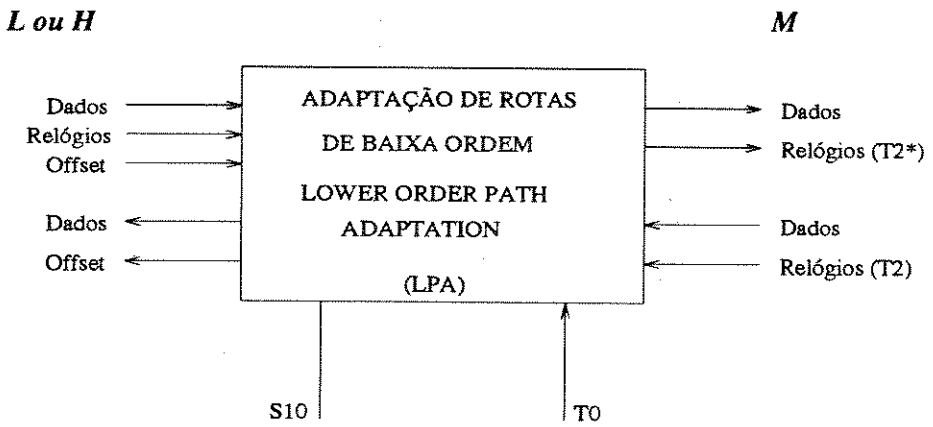


Figura 3.36: Adaptação de Rotas de Baixa Ordem

- LPA-12 $\rightarrow$ mapeamento $\left\{ \begin{array}{l} \textit{assíncrono bit a bit} \\ \textit{síncrono bit a bit} \\ \textit{síncrono byte a byte} \\ \textit{modo amarrado} \end{array} \right\} \rightarrow$ container C-12 (2,048 Mb/s)
- LPA-2 $\rightarrow$ mapeamento $\left\{ \begin{array}{l} \textit{assíncrono} \\ \textit{síncrono} \end{array} \right\} \rightarrow$ container C-2 (6,312 Mb/s)
- LPA-3 $\rightarrow$ mapeamento assíncrono $\rightarrow$ container C-3 (44 e 34 Mb/s)
- Mapeamentos Diretos : $\left\{ \begin{array}{l} LPA\text{–}3,\ \textit{assíncrono},\ C\text{–}3\ (44\ e\ 34\ Mb/s) \\ LPA\text{–}4,\ \textit{assíncrono},\ C\text{–}4\ (139\ Mb/s) \end{array} \right\}$

Fluxo (L ou H) $\rightarrow$ M

Os dados no ponto L (ou H no caso de mapeamento direto) são apresentados em um container juntamente com seu offset de quadro. O fluxo de carga útil que deve ser entregue ao usuário é recuperado do container, junto com seu relógio (T2* gerado no transmissor distante), que deve ser passado para o lado M. Isto envolve desmapeamento e dessincronização; o relógio gerado no transmissor remoto é recuperado com algum jitter.

Quando é reportado AIS de Rota em S10, o LPA gera AIS em concordância com os procedimentos recomendados pela série G.700 do CCITT (procedimento normal na PDH).

Fluxo M $\rightarrow$ (L ou H)

Os dados em M estão na forma de um fluxo de bits entregues pela Interface Física (Dados + Relógios T2). Os dados são adaptados de acordo com um LPA-ij apropriado para a sua taxa. Isto envolve sincronização e mapeamento do fluxo de bits de entrada em um container. O container é passado para o lado L (ou H no caso de mapeamento direto) na forma de dados (Dados + offset de Quadro), onde aqui o offset é a distância (em bytes) que separa determinado byte do quadro criado com temporização T0 e o início da carga útil. Em mapeamento síncrono byte a byte, isto requer o conhecimento do sincronismo de canais do sinal afluente.

Perda de sincronismo de quadro (Frame Alignment Loss) é enviada ao SEMF (Gerenciamento) em S10. O processo de deteção/indicação de FAL é descrito na rec. G.706.

3.5.4 Interface Física (PI)

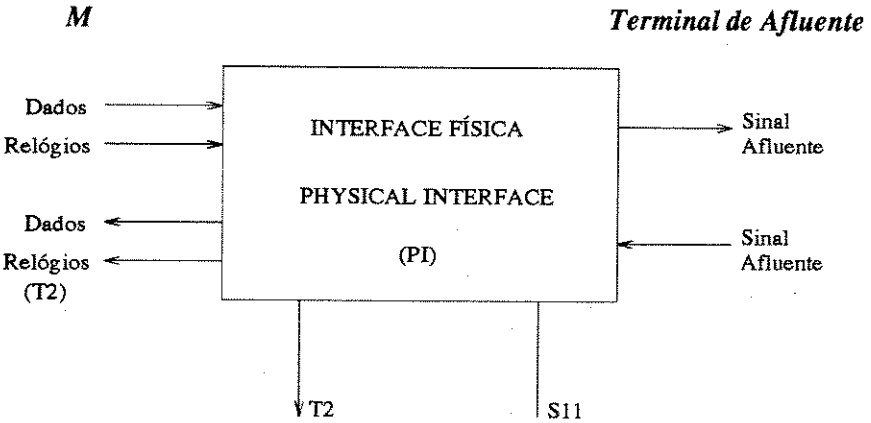


Figura 3.37: Interface Física

A Função da Interface Física é prover uma interface entre o multiplexador e o meio físico que transporta um sinal afluente, cujas características são aquelas previstas na recomendação CCITT G.703 e, em alguns casos, na G.704.

Fluxo $M \rightarrow$ Terminal

Nesta direção deve ser feita a codificação de linha e a adaptação ao meio físico de transmissão. Portanto, PI toma (dados + relógios T2* em M e transmite pelo terminal o sinal afluente na forma prevista pela G.703/704. Códigos usuais encontrados neste lado (terminal) são o código HDB-3 e o CMI.

Fluxo (Terminal Afluente) $\rightarrow M$

A Interface Física atua no sinal de entrada, extraindo o seu relógio T2 e decodificando o código de linha. Dados detectados e relógios são passados ao ponto M. O relógio T2 extraido é trasferido para o módulo de Temporização do Multiplex (MTS) para possível uso.

Na eventualidade de haver perda de sinal (Loss of Signal - LOS) na entrada do afluente, gera-se AIS na forma de "TUDO UM" que é transmitido por M (Dados), acompanhado de relógio apropriado tomado localmente. LOS é reportado em S11.

3.6 Gerenciamento de Equipamento Síncrono

A função de Gerenciamento de Equipamento Síncrono (Synchronous Equipment Management Function - SEMF) provê os meios através dos quais um elemento da rede síncrona é gerenciado por um Gerenciador , seja ele interno ou externo. Caso o elemento da rede contenha um Gerenciador, ele será parte da SEMF.

Através da SEMF há a interação com blocos funcionais do equipamento, trocando informação por meio dos pontos Si de referência. A SEMF possui vários "filtros" que possibilitam interpretação mais abrangente e redução dos dados de saída sobre a informação recebida nos vários pontos Si.

A recomendação G.784 do CCITT trata do problema de gerenciamento da SDH (Synchronous Digital Hierarchy Management), levando em consideração a rede especializada em gerenciamento de telecomunicações (Telecomunications Management Network - TMN) e os protocolos definidos para as interfaces Q e F. Nesta seção resume-se somente a funcionalidade interna do equipamento SDH.

O gerenciamento do equipamento SDH é visto como um subconjunto da rede de gerenciamento de telecomunicações (TMN), descrita na recomendação M.30.

Nos sinais STM-N existem 2 canais de Dados, ou seja, canal ($D_1 - D_3$) de 192 kb/s e canal ($D_4 - D_{12}$) de 576 kb/s. O canal DCC_R de 192 kb/s é acessível a todos elementos da rede SDH, enquanto que o canal DCC_M de 576 kb/s, não sendo parte da carga de supervisão de seção regeneradora, não é acessível aos regeneradores. O canal DCC_M é usado como um canal de comunicação de uso geral para grande área geográfica de suporte à TMN, incluindo aplicações fora da SDH.

No contexto da TMN temos as seguintes definições:

- Função Aplicação de Gerenciamento (Management Application Function - MAF)
  - um processo de aplicação para gerenciamento de sistema. A MAF inclue um Agente (sendo gerenciado) e um *Gerenciador*. A MAF é a origem e a terminação para todas mensagens TMN.
- Objeto Gerenciável (Managed Object - MO)
  - um recurso dentro do ambiente de telecomunicações que pode ser gerenciado via um Agente. Exemplos de objetos gerenciáveis são: equipamento, terminal de recepção, terminal de transmissão, fonte de alimentação, cartão de circuito impresso, container virtual, seção multiplex, seção regeneradora.
- Agente (Agent)
  - parte da MAF capaz de responder a operações da rede de gerenciamento ordenadas por um Gerenciador e que pode executar operações em Objetos Gerenciáveis, produzindo eventos.

Dadas as definições acima, pode-se então analisar o gerenciamento de equipamento síncrono (SEMF) através do diagrama de blocos mostrado na figura 3.38

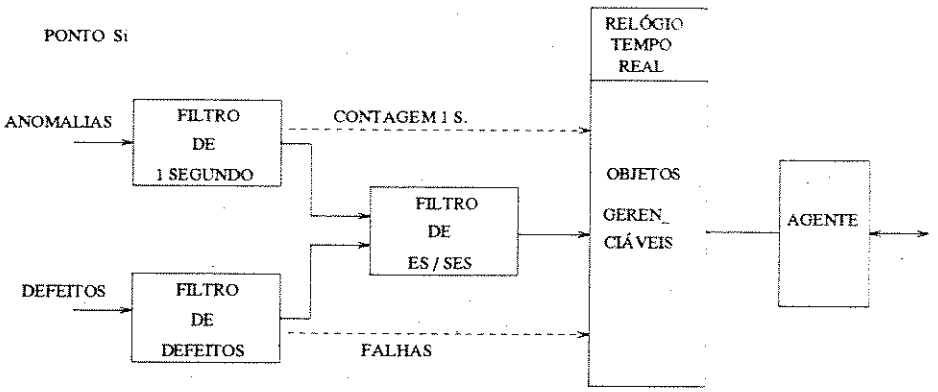


Figura 3.38: Gerenciamento de Equipamento Síncrono (SEMF)

As saídas dos filtros são colocadas à disposição do "Agente" através do bloco "Objetos Gerenciáveis".

O bloco "Objetos Gerenciáveis" processa eventos e armazena informações sobre o funcionamento do equipamento, além de uniformizar os formatos das informações. O 'Agente" converte as informações recebidas em mensagens para a TMN (Gerenciador) e representa a interface no ponto V de referência com o Sistema de Comunicação de Mensagens (ver figura 3.1.

As funções de processamento de eventos e armazenagem do bloco "Objetos Gerenciáveis" são descritas na recomendação G.784, incluindo limiares de desempenho. Esta análise não é tratada aqui.

Fluxo de Informações nos Pontos Si de Referência

As informações que aparecem nos pontos Si devidas a anomalias ou defeitos verificados no equipamento são sumarizadas em tabelas da recomendação G.783. Abaixo mostra-se uma tabela exemplo, no caso para a Terminação da Seção Multiplex. A compilação de todas essas tabelas fornece as ações de gerenciamento esperadas da SEMF.

Filtragens

As filtragens são um mecanismo de redução da taxa de dados apresentados nos pontos de referência Si devidos a defeitos e anomalias. Três tipos de Filtragem são usados:

- filtragem de um segundo

- filtragem de defeito

- filtragem "duração de estado em erro"

- Filtragem de 1 Segundo

  A filtragem de um segundo executa uma simples integração de anomalias relatadas, contando um intervalo de 1 segundo (considerado satisfatório para propósitos de supervisão da rede de conexão, identificação de falha e de seção em falha). Ao fim do intervalo de 1 segundo, os contadores fornecem os dados para o bloco "Objetos Gerenciáveis". As seguintes saídas de

TABELA PARA TERMINAÇÃO DE SEÇÃO MULTIPLEX (EXEMPLO)

FLUXO DE SINAL	ANOMALIAS E DEFEITOS	RELATO EM	FILTRAGEM SEMF		AÇÕES CONSEQUENTES	
	AIS NA SEÇÃO MULTIPLEX	S3	ALARME	DESEM_ PENHO	FERF INSERIDO	AIS INSERIDO
		SIM	SIM		SIM	SIM (1)
	TAXA EXCESSIVA DE ERRO (B_2)	SIM	SIM		SIM (2)	SIM (1, 2)
C→ D	SINAL DEGRADADO B_2	SIM	SIM			
	NÚMERO DE ERROS EM B_2	SIM		SIM		
	FERF FALHA NO RECEPTOR REMOTO	SIM	SIM			
D→ C						

Observações: (1) - Também aplicável para D_4 - D_{12}
Idem Z_1 , Z_2 e bytes não usados na MSOH

(2) - Deve ser possível desarmar a inserção de FERF e AIS para deteção de defeito "Taxa Excessiva de Erro (B_2) " através de ação da SEMF.

contadores devem estar disponíveis:

- Erros em Seção Regeneradora (B_1)
- Eventos de "Out of Frame" (OOF) em Seção Regeneradora
- Erros na Seção Multiplex (B_2)
- Erros em Rota de Alta Ordem (B_3)
- Erros em Rota de Baixa Ordem (B_3/V5)
- Erros remotos (Far End Block Error - G_1) em Rota de Alta Ordem
- Idem G_1/V5 para Rotas de Baixa Ordem
- Eventos de justificação em AU (em estudos)
- Eventos de justificação em TU (em estudos)

- Filtragem de Defeitos

  Esta filtragem provê uma verificação persistente em defeitos relatados através dos pontos Si. Como todos os defeitos irão aparecer na entrada deste filtro, pode-se correlacioná-los a fim de se reduzir a quantidade de informação a ser oferecida (como falha) para o Agente. As seguintes indicações de falha são fornecidas:

  - Perda do Sinal (LOS)
  - Perda de Alinhamento de Quadro
  - Perda de Ponteiro AU

- AIS em Seção Multiplex
- AIS em Rota de Alta Ordem
- AIS em Rota de Baixa Ordem
- Falha no Receptor Remoto
- FERF em Rota de Alta Ordem
- FERF em Rota de Baixa Ordem
- etc, etc (tabelas da rec. G.783)

- Filtragem "Duração de Estado em Erro"

  Nessa filtragem a informação disponível nos filtros de 1 segundo e de defeitos é processada para se obter a duração do estado em erro, ou seja:

  - Segundos em Erro (Errored Seconds - ES)
  - Segundos em Erro Severo (Severely Errored Seconds - SES)

  Valores ES e SES são passados ao Agente.

3.6.1 Sistema de Comunicação de Mensagem (MCF)

O Sistema de Comunicação de Mensagens (Message Communications Function - MCF) provê facilidades para o transporte de mensagens da TMN para e da MAF, além do trânsito das mesmas. O MCF não origina e nem é terminação de mensagens.

O MCF interfaceia o Sistema de Gerenciamento do Equipamento (SEMF) no ponto V de referência. É trânsito para os canais DCC_R e DCC_M através dos pontos de referência N e P respectivamente. Para o meio exterior, conecta-se através de interfaces padronizadas Q e F com a TMN - veja figura 3.1.

3.7 Funções de Temporização do Multiplex

As Funções de Temporização do Multiplex são realizadas por dois blocos funcionais:

-Temporização do Multiplex

-Interface Física do Temporizador

- Temporizador do Multiplex (MTS)

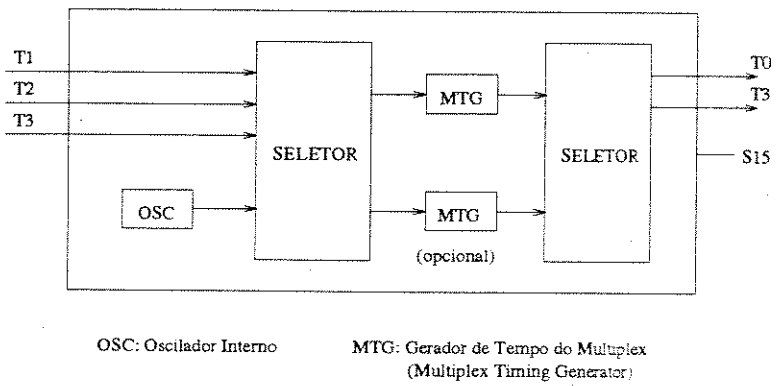


Figura 3.39: Temporizador do Multiplex - Multiplex Timing Source (MTS)

O temporizador do Multiplex tem a função de fornecer a referência de Tempo (T0) para os seguintes blocos funcionais: LPA, LPT, LPC, HPA, HPT. HPC, SA, MSP, MST e RST. O MST é o relógio do equipamento SDH. Ele inclui:

- um oscilador interno (OSC)
- um (ou dois) Gerador de Tempo (MTG)

A fonte de sincronização pode ser selecionada de qualquer uma das entradas:

- T1→ proveniente de outro STM-N (ver observação no fim)
- T2→ proveniente de uma interface de sinal afluente
- T3→ sinal externo especial para sincronização (sincronismo de rede)

ou ainda utilizar a temporização dada pelo oscilador interno.

O relógio T3 (sincronização externa) deve satisfazer as especificações do CCITT em relação a precisão e estabilidade de curto prazo.

Internamente, o gerador de tempo do multiplex (MTG) filtra a referência de tempo selecionada, a fim de garantir que as especificações de relógio sejam satisfeitas. Além disso, o MTG deve filtrar o salto de frequência, de tal modo que a variação de frequência no ponto T0 não ultrapasse a Hz/s, onde o valor a está para ser especificado. Isto é aplicável nos 3 seguintes casos:

- mudança de uma fonte de referência para outra;
- mudança de uma fonte de referência para o oscilador interno;

– mudança do oscilador interno para uma fonte de referência.

- Interface Física do Temporizador (MTPI)

  A função da Interface Física do Temporizador (Multiplexer Timing Physical Interface Function - MTPI) é realizar uma interface entre o sinal externo de sincronização e o MTS. Deve ter neste terminal as características físicas das interfaces de sincronização de uma daquelas da G.703.

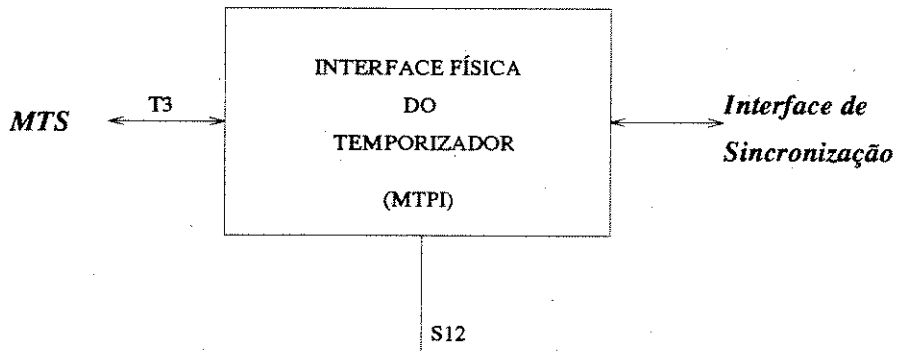


Figura 3.40: Interface Física do Temporizador

  – fluxo do MTS para a Interface de sincronização

    Este fluxo só existe se o MTS do equipamento puder fornecer sincronização externa. Neste caso as funções realizadas pelo MTPI são de codificação e adaptação ao meio físico. O MTPI recebe "timimg" do MTS para formar o sinal de sincronismo de transmissão; a informação de "timing" é passada transparentemente para a interface de sincronização.

  – fluxo da Interface de sincronização para o MTS

    O MTPI extrai "timing" do sinal externo de sincronismo recebido, depois de decodificá-lo. "Timing" é passado para MTS.

Observação - O sinal STM-N pode ser usado para sincronizar o MTS (relógio T1). O MTS deve ser capaz de acomodar os valores máximos de jitter e wander presentes no sinal STM-N.

3.8 Função de Acesso de Supervisão (OHA)

No equipamento multiplex SDH pode ser especificado um acesso integrado para as funções de supervisão (Overhead Access Function -OHA). Esta especificação está sendo estudada ainda no CCITT. Estão definidos os pontos de referência U, através dos quais informações podem ser trocadas com os vários blocos funcionais do equipamento.

Um acesso externo obviamente necessário é o correspondente à função EOW (Engineering Order Wire) de serviço, que é usado pelo pessoal de manutenção para contatos por voz entre o local de um regenerador e locais de terminais de linha.

3.9 Tipos de Equipamentos da SDH

Nesta seção vamos descrever alguns tipos de equipamentos SDH que deverão aparecer no mercado. A descrição é sucinta e baseada nos blocos funcionais mostrados nas seções anteriores (2 a 8). Para isso definimos os seguintes blocos funcionais compostos:

- Função Terminal de Transporte - TTF (Transport Terminal Function)
  Composição (Ver figura 3.41):
  - Interface Física SDH (SPI)
  - Terminação de Seção Regeneradora (RST)
  - Terminação de Seção Multiplex (MST)
  - Proteção de Seção Multiplex (MSP)
  - Adaptação de Seção Multiplex (MSA)

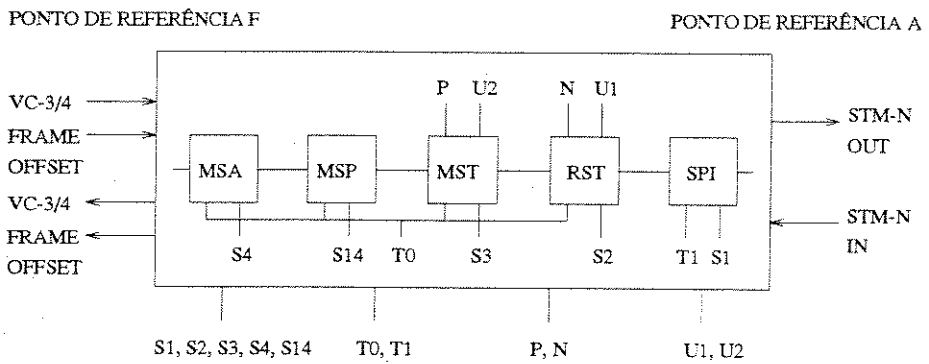


Figura 3.41: Função Terminal de transporte (TTF)

- Interface de Alta Ordem - HOI (Higher Order Interface)
  Composição (Ver figura 3.42):
  - Interface Física SDH (PI)
  - Adaptação de Rota de Baixa Ordem (LPA)
  - Terminação de Rota de Alta Ordem (HPT)

- Interface de Baixa Ordem - LOI (Lower Order Interface)
  Composição (Ver figura 3.43):

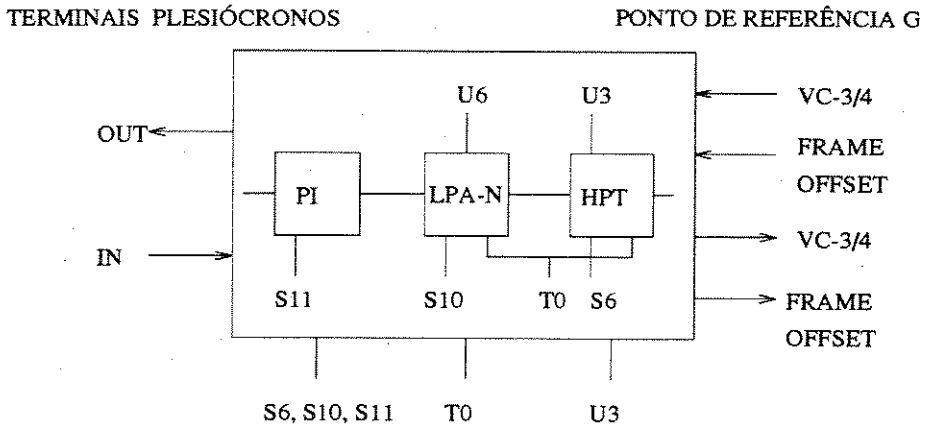


Figura 3.42: Interface de Alta Ordem (HOI)

- Interface Física SDH (PI)
- Adaptação de Rota de Baixa Ordem (LPA)
- Terminação de Rota de Baixa Ordem (LPT)

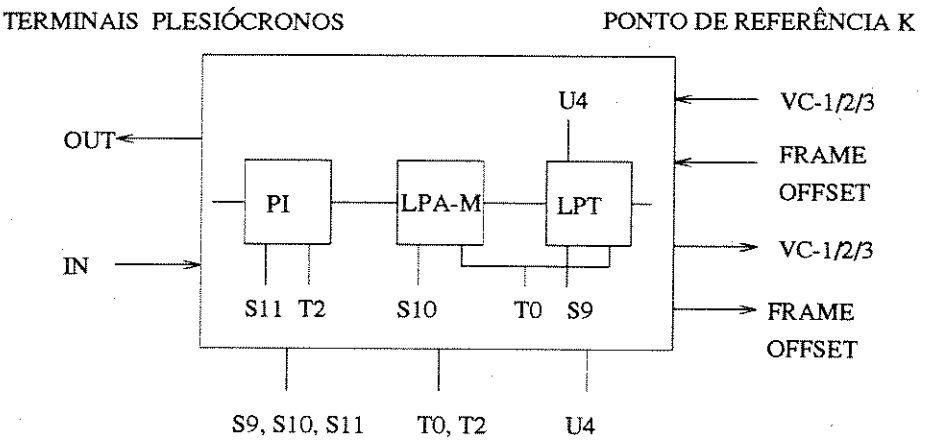


Figura 3.43: Interface de Baixa Ordem (LOI)

- Montador de Alta Ordem - HOA (Higher Order Assembler)
  Composição (Ver figura 3.44):
  - Adaptação de Rota de Alta Ordem (HPA)
  - Terminação de Rota de Alta Ordem (HPT)

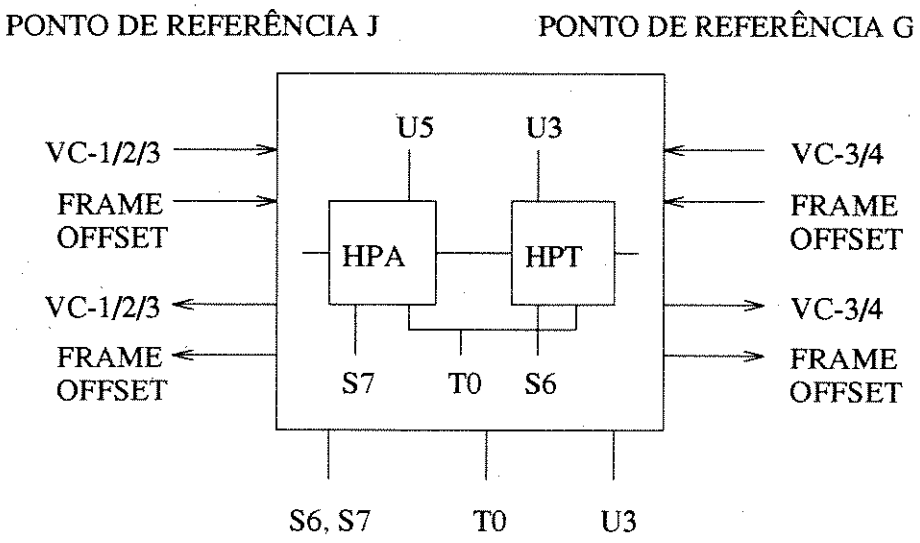


Figura 3.44: Montador de Alta Ordem (HOA)

3.9.1 Diagrama de Blocos do Multiplex Geral

O multiplex mais geral pode ser construido com os blocos funcionais definidos nas seções (2 a 8), ou pode ainda ser representado mais simplesmente com os blocos funcionais compostos que acabamos de definir acima. Os respectivos diagramas de blocos estão mostrados nas figuras 3.45 e 3.46.

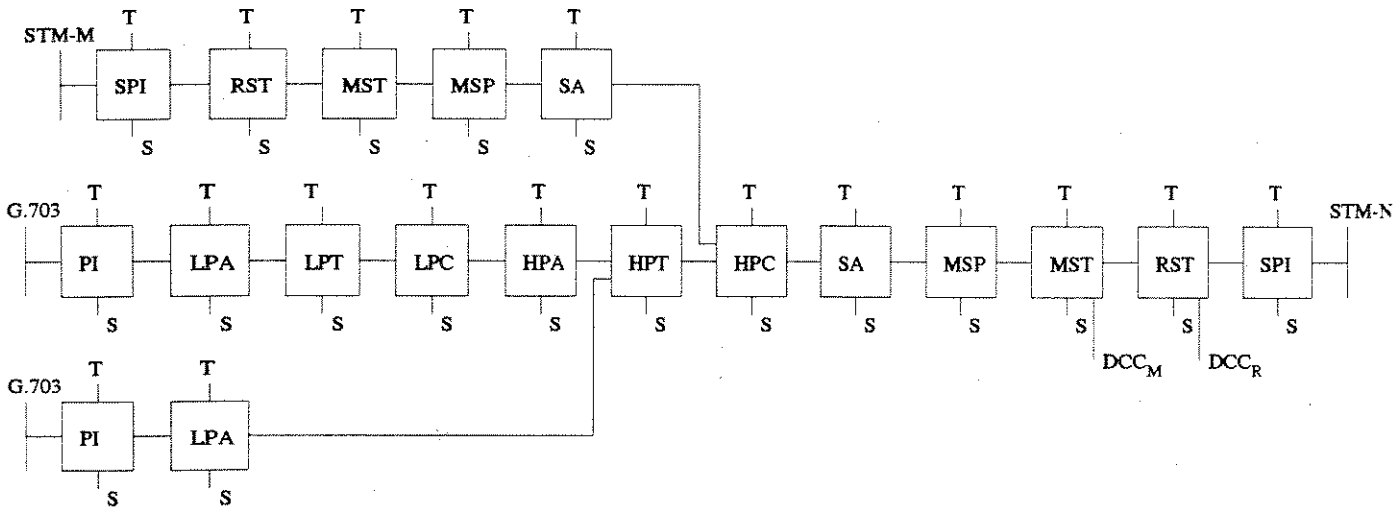


Figura 3.45: Diagrama de Blocos do Multiplex Geral (Blocos Elementares)

- DCC_R - DCC bytes $D_1 - D_3$
- DCC_M - DCC bytes $D_4 - D_{12}$
- HPA - Higher Order Path Adaptation
- HPC - Higher Order Path Connection
- HPT - Higher Order Path Termination
- LPA - Lower Order Path Adaptation
- LPC - Lower Order Path Connection
- LPT - Lower Order Path Termination
- MSP - Multiplex Section Protection
- MST - Multiplex Section Termination
- PI - Physical Interface
- RST - Regenerator Section Termination
- S - monitor points, alarms, controls
- SA - Section Adaptation
- SPI - SDH Physical Interface
- T - Timing
- Nota 1 → Opções SPI
  - interface elétrica interna de estação

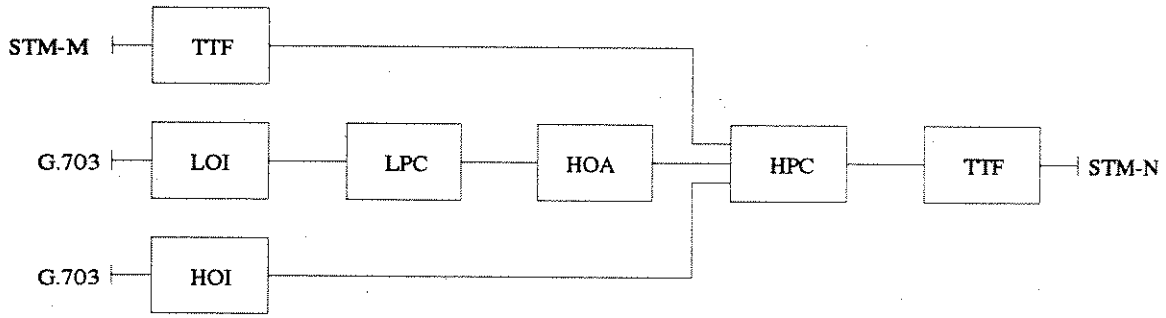


Figura 3.46: Diagrama de Blocos do Multiplex Geral (Blocos Compostos)

- interface óptica interna de estação
- interface óptica entre centrais

- TTF - Transport Termination Function
- LOI - Lower Order Interface
- HOI - Higher Order Interface
- HOA - Higher Order Assembler
- HPC - Higher Order Path Connection

O multiplex geral da figura 3.45 (ou 3.46) é capaz de comutar containers de alta ordem nas direções STM-M ou STM-N, além de admitir cargas úteis de grande e pequeno porte.

3.9.2 Funções Auxiliares em Equipamentos SDH

Nos vários tipos de equipamentos que discutiremos a seguir, as funções auxiliares de

- Temporização (MTS + MTPI)
- Gerenciamento (SEMF + MCF)
- Acesso de Supervião (OHA)

estarão obrigatóriamente presentes mas não serão mostradas. Deste modo, o conjunto mostrado na figura 3.47 fará parte de todos os tipos de equipamentos SDH, podendo ser iguais ou semelhantes.

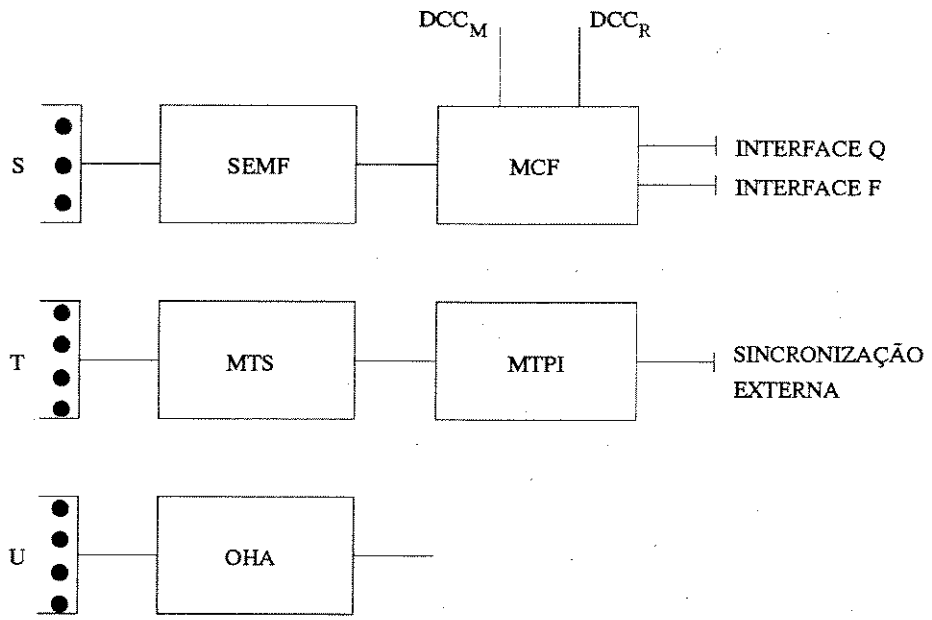


Figura 3.47: Funções Auxiliares

Multiplex Tipo I.1

É o multiplex mais simples, agrupando afluentes da recomendação G.703 e agregando-os no STM-N. Por exemplo, 63 canais de 2048 kb/s podem ser multiplexados em um STM-1. Neste MUX as posições dos afluentes são fixas no sinal agregado; dado um terminal afluente de entrada, o seu mapeamento está determinado no STM-N (Não existe LPC). Ver figura 3.48.

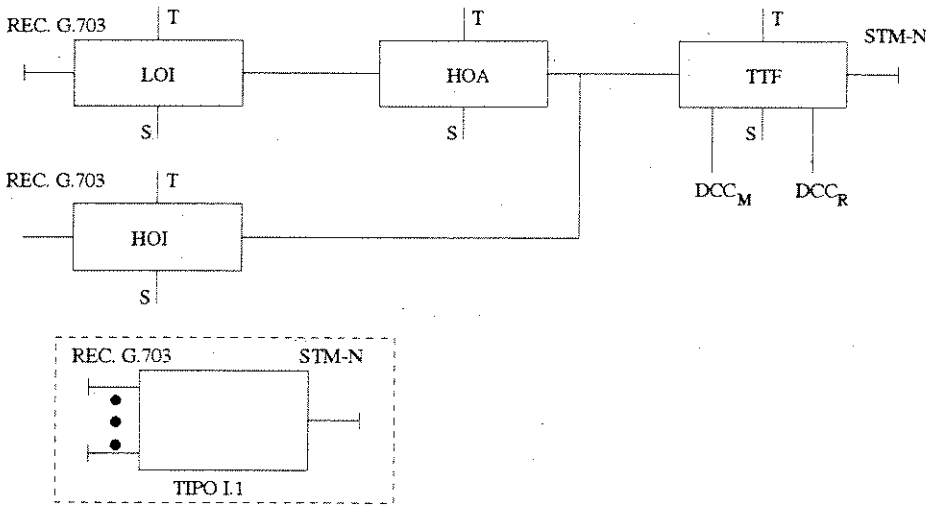


Figura 3.48: MUX tipo I.1

Multiplex Tipo I.2

Neste Caso, pela inclusão da função de conexão de rota (LPC para VC-1/2, HPC para VC-3/4 e/ou), tem-se então a possibilidade de alocar de modo flexivel uma entrada a qualquer posição do quadro STM. Ver figura 3.49.

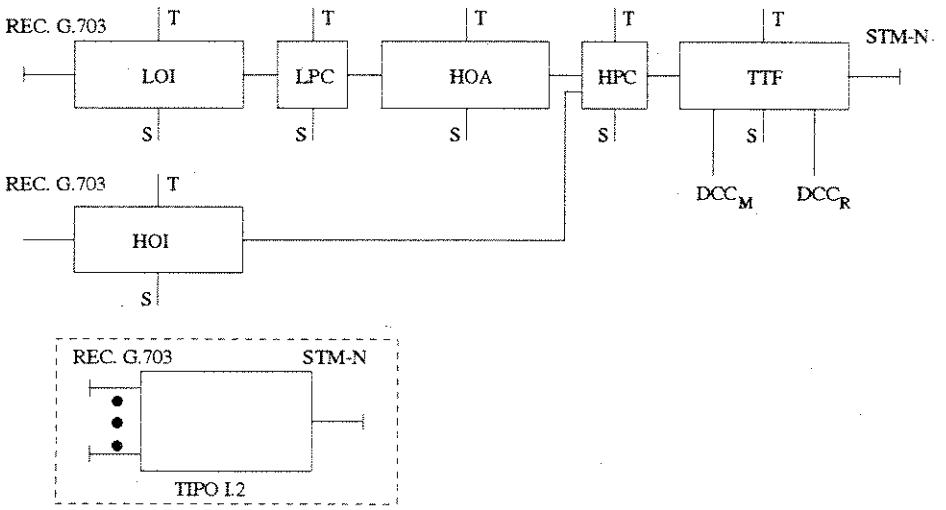


Figura 3.49: MUX tipo I.2

Multiplex Tipo II.1

Este tipo de MUX permite combinar um determinado número de sinais STM-N em um agregado STM-M, com M>N. Por exemplo, multiplexar 4 sinais em um STM-4. As terminações STM-1 podem ser elétricas ou ópticas.

A posição de cada VC-3/4 dos sinais STM-N afluentes é fixa no sinal agregado STM-M. Ver figura 3.50.

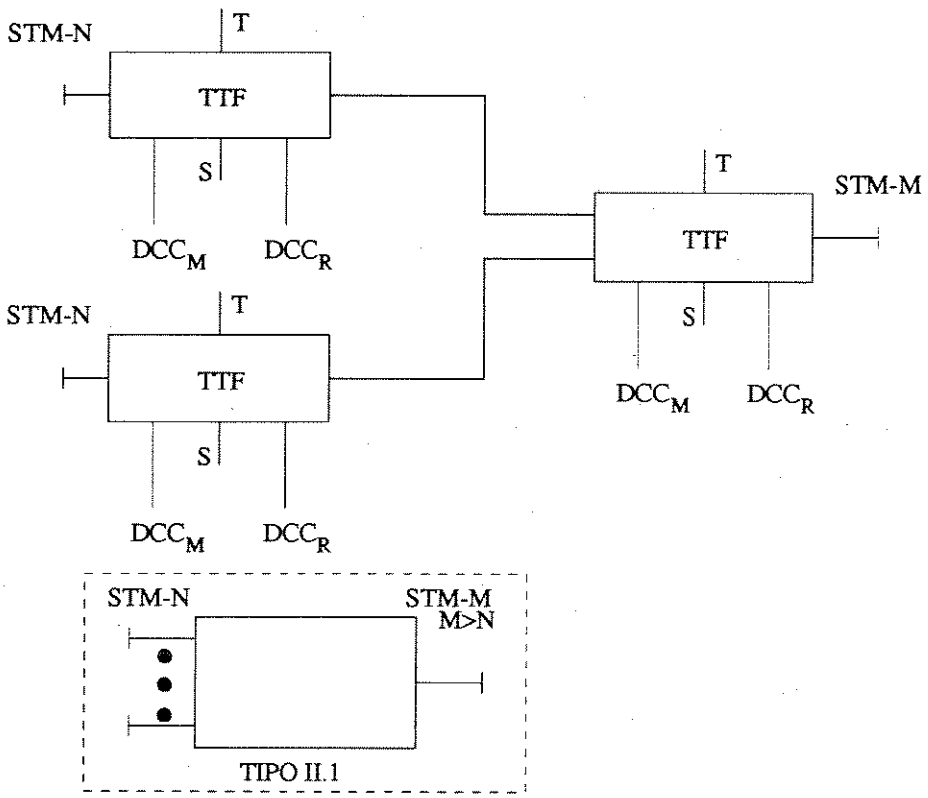


Figura 3.50: MUX tipo II.1

Multiplex Tipo II.2

Introduz-se neste tipo de MUX a facilidade de alocação flexível de containers VC-3/4 em qualquer posição do quadro do sinal agregado STM-M. Para isso inclui-se a função de conexão de rota. Ver figura 3.51.

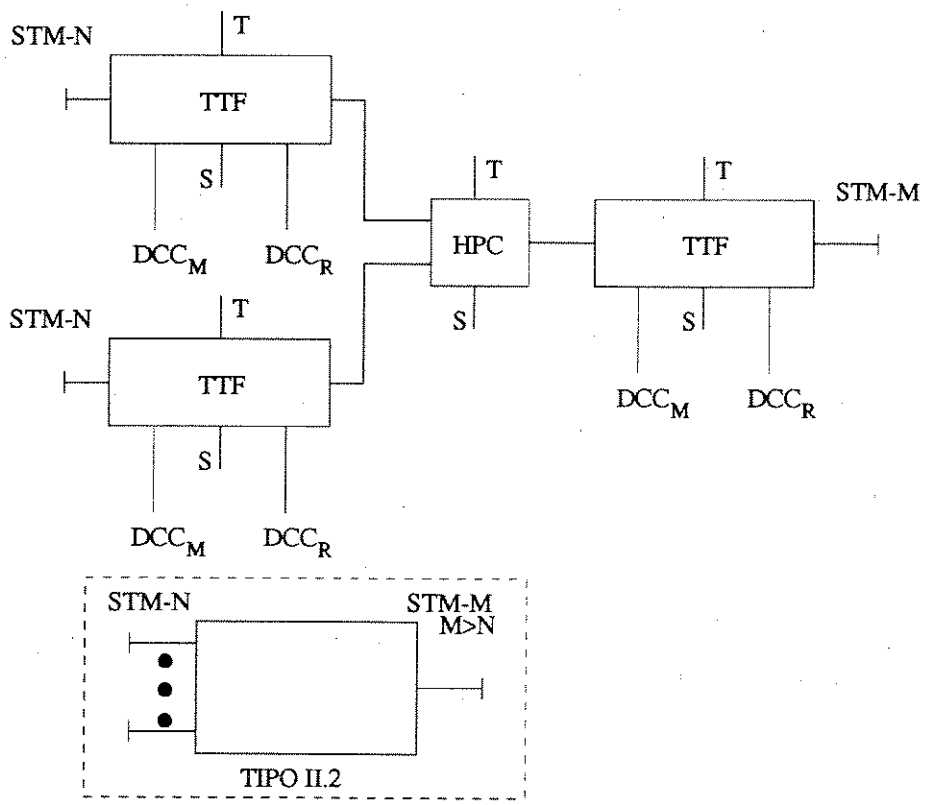


Figura 3.51: MUX tipo II.2 (com comutação)

Multiplex Tipo III

Este tipo de MUX tem a propriedade de poder acessar qualquer sinal da composição de um STM-N, sem necessidade de demultiplexá-lo e retirá-lo através de um terminal. A interface que se coloca para acessar os sinais pode ser da rec. G.703 ou então da própria SDH.

Este tipo de MUX é também conhecido como ADM (Add-Drop Multiplex). Mais detalhes a seguir.

Multiplex Tipo III.1

Na figura 3.52 mostra-se um MUX tipo III, onde o acesso ao sinal STM é através de uma interface G.703.

A função de conexão de rota de alta ordem (HPC) permite que os sinais VC-3/4 em um STM-N sejam terminados localmente ou então multiplexados novamente para transmissão. Também permite que sinais VC-3/4 gerados localmente sejam designados a ocupar qualquer posição no STM-N de saída. Por outro lado, a função de conexão de rota de baixa ordem (LPC) permite que sinais VC-1/2 aqui terminados pelos containers C-3/4 sejam enviados aos terminais de saída, ou então sejam novamente multiplexados de volta em um VC-3/4 na direção STM. O LPC propicia que os sinais de containers VC-1/2 gerados localmente sejam roteados para qualquer posição vaga em um container C-3/4 de saída.

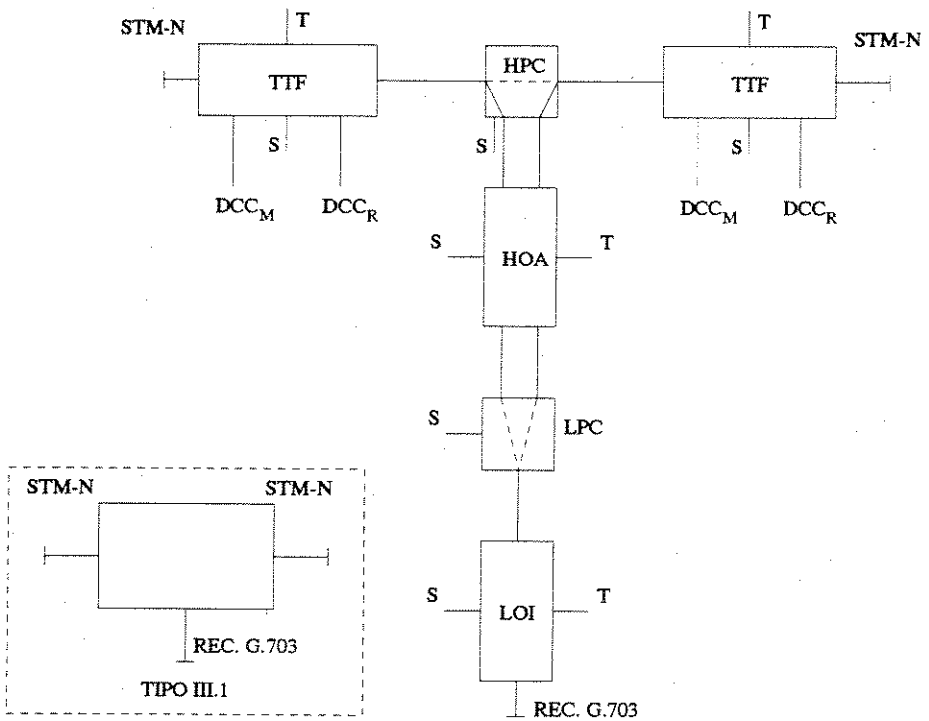


Figura 3.52: MUX tipo III.1

Multiplex Tipo III.2

Na figura 3.53 mostra-se o MUX tipo III, no qual o acesso aos sinais constituintes do STM-N é através de uma interface STM-M. Ele possui algumas funções adicionais em relação àquelas do tipo III.1, para demultiplexagem do sinal STM-N em sinais VC-1/2. É um MUX de 3 cabeças - as 3 podem ser interfaces ópticas.

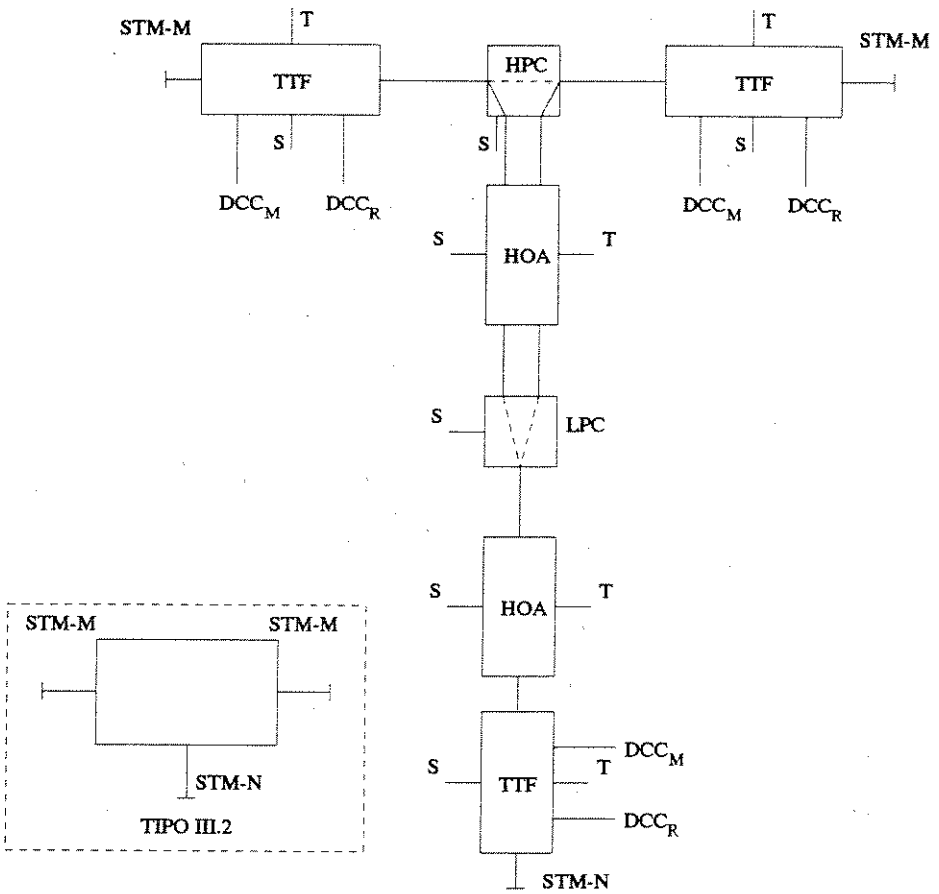


Figura 3.53: MUX tipo III.2

Multiplex Tipo IV

Este tipo de MUX possue a função de translação que viabiliza que cargas úteis C-3 transportadas em VC-3 trafeguem em uma rede que use equipamento SDH que não suporta AU-3. Ver figura 3.54

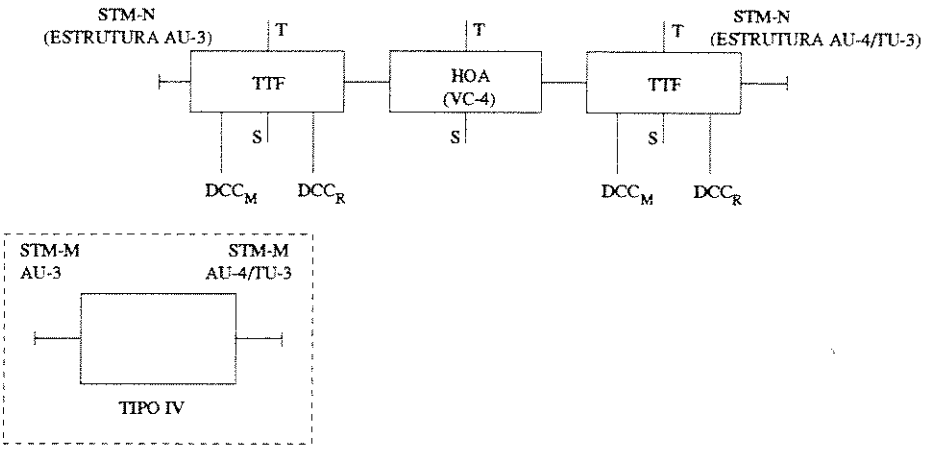


Figura 3.54: MUX tipo IV

Capítulo 4

VHDL (uma breve apresentação)

VHDL quer dizer: VHSIC Hardware Description Language. VHSIC (Very High Speed Integrated Circuit) é um programa fundado pelo Departamento de Defesa Americano com o objetivo de produzir a próxima geração de circuitos integrados a partir do início da década de 80.

Os objetivos do programa VHSIC foram atingidos admiravelmente, mas, no processo de desenvolvimento de circuitos integrados extremamente complexos, os projetistas descobriram que as ferramentas de que dispunham eram inadequadas. As ferramentas então disponíveis eram na maioria baseadas no nível de portas lógicas. Criar projetos de centenas de milhares de portas usando ferramentas que trabalham ao nível de portas lógicas era uma tarefa extremamente difícil.

Em função dessas dificuldades, novos métodos e ferramentas de projeto de hardware foram desenvolvidos; entre eles a linguagem VHDL se popularizou tanto que em Dezembro de 1987 ela se tornou um padrão IEEE.

VHDL é uma linguagem extensa e aprende-la por completo é uma tarefa difícil. Entretanto não é necessário aprende-la por completo para-se escrever modelos úteis. Não será necessário portanto, extender muito sobre o assunto.

Em VHDL uma lógica de circuito é representada por uma Entidade de Projeto. A lógica de circuito representada pode ser tão complexa como um microprocessador ou tão simples quanto uma porta AND.

4.1 Entidades de Projeto

Em VHDL um circuito lógico é representado como sendo uma *Entidade de Projeto.* A *Entidade de Projeto* consiste de dois tipos diferentes de descrições: A descrição de interface e a descrição de arquitetura. Ilustra-se este conceito com um exemplo. Considere a descrição de interface para o circuito que conta o número de 1's em um vetor entrada de comprimento 3 mostrado na figura 4.1

Pode-se notar que a descrição de interface dá nome à entidade e descreve as suas entradas e saídas. A descrição dos sinais de interface incluem o modo do sinal (IN ou OUT) e o tipo do sinal,

```
ENTITY contador_de_uns IS
  PORT(A: IN BIT_VECTOR(0 to 2); C: OUT BIT_VECTOR(0 to 1));
END contador_de_uns;
```

Figura 4.1: Descrição de interface para o contador de uns.

neste caso BIT_VECTOR(0 to 2) e BIT_VECTOR(0 to 1) (isto é, vetores de comprimentos 3 e 2 bits).

4.2 Arquitetura

A descrição de interface basicamente define somente as entradas e saídas da entidade de projeto. Além disto, o que se requer é um meio de se especificar o comportamento da entidade. Em VHDL isto é feito desenvolvendo-se uma *Arquitetura.* Como será demonstrado, esta arquitetura pode determinar o comportamento da entidade diretamente (isto é, ser uma arquitetura primitiva) ou pode ser uma decomposição estrutural em termos de componentes mais simples.

No início do processo de projeto, os projetistas usualmente têm um algoritmo em mente que eles gostariam de implementar. Inicialmente, entretanto, eles gostariam de checar a exatidão do algoritmo sem especificar uma implementação detalhada. Portanto a primeira arquitetura a ser implementada é uma puramente comportamental. A figura 4.2 mostra tal arquitetura para o CONTADOR_DE_UNS.

```
ARCHITECTURE puramente_comportamental OF contador_de_uns IS
BEGIN
 PROCESS(A)
  VARIABLE NUM: INTEGER RANGE 0 to 3;
 BEGIN
  NUM := 0;
  FOR i IN 0 TO 2 LOOP
    IF A(i) = '1' THEN
      NUM := NUM + 1;
    END IF;
  END LOOP;
  CASE NUM IS
    WHEN 0 => C <= "00";
    WHEN 1 => C <= "01";
    WHEN 2 => C <= "10";
    WHEN 3 => C <= "11";
  END CASE;
END PROCESS;
END puramente_comportamental;
```

Figura 4.2: Descrição puramente comportamental do contador de uns.

Note na arquitetura da figura 4.2 que o "LOOP" varre as entradas de A(0) até A(2) e incrementa a variável "NUM" quando um bit particular for "1". Baseado no valor final de "NUM", o comando "CASE" seleciona o valor binário para ser transferido para a saída.

Este corpo comportamental descreve a operação do algoritmo perfeitamente, mas a sua correspondência com o hardware real é fraca.

Continuando o exemplo, suponha que o projeto do contador de uns se desenvolva para o estágio de projeto lógico. Seja C1 e C0 as saídas do circuito contador de uns e A2, A1 e A0 as suas entradas. então a partir de mapas de Karnaugh o projetista pode determinar as equações booleanas para C1 e C0, que são:

$$C1 = A1.A0 + A2.A0 + A2.A1$$

$$C0 = A2.\overline{A1}.\overline{A0} + \overline{A2}.\overline{A1}.A0 + A2.A1.A0 + \overline{A2}.A1.\overline{A0}$$

Ele deve também notar, para uso posterior, que C1 é a função majoritária das três entradas A2, A1 e A0 [MAJ3(A2,A1,A0)] e que C0 é a função de paridade par das três entradas [PARP(A2,A1,A0)].

Neste ponto o projetista pode substituir a arquitetura puramente comportamental pela arquitetura mostrada na figura 4.3.

```
ARCHITECTURES equacoes_booleanas OF contador_de_uns IS
BEGIN
  C(1) <= (A(1) AND A(0)) OR (A(2) AND A(0)) OR (A(2) AND A(1));

  C(0) <= (A(2) AND NOT(A(1)) AND NOT(A(0)))
          OR (NOT(A(2)) AND NOT(A(1)) AND  A(0))
          OR (A(2) AND A(1) AND A(0))
          OR (NOT(A(2)) AND A(1) AND NOT(A(0)));
END equacoes_booleanas;
```

Figura 4.3: Arquitetura de segundo nível para o contador de uns.

A lógica mostrada na figura 4.3 implica em uma estrutura de portas lógicas padronizadas AND e OR e inversores. Entretanto como C1 e C0 podem ser computados pelas funções a serem definidas MAJ3 e PARP3 respectivamente, uma arquitetura ainda mais simples de nível macro pode ser a da figura 4.4.

```
ARCHITECTURE macro OF contador_de_uns IS
BEGIN
  C(1) <= MAJ3(A);
  C(0) <= PARP3(A);
END macro;
```

Figura 4.4: Arquitetura de nível macro para o contador de uns.

Esta arquitetura implica na existência das funções MAJ e PARP a nível de hardware. Em termos de descrição em VHDL, ela requer que as funções MAJ e PARP tenham sido declaradas e definidas previamente.

As três arquiteturas vistas para o contador de uns são comportamentais no sentido que elas especificam a resposta entrada/saída da entidade de projeto sem especificar exatamente a estrutura interna. Agora cria-se uma arquitetura estrutural para o contador de uns. No tratamento feito aqui, usa-se a hierarquia de projeto mostrada na figura 4.5; isto é, o contador de uns é primeiro decomposto nas portas MAJ3 e PARP3, que são por sua vez decompostas em portas AND e OR. As figuras 4.6, 4.7 e 4.8 mostram as descrições de interface e as arquiteturas para as portas MAJ3, AND2 e OR3 respectivamente.

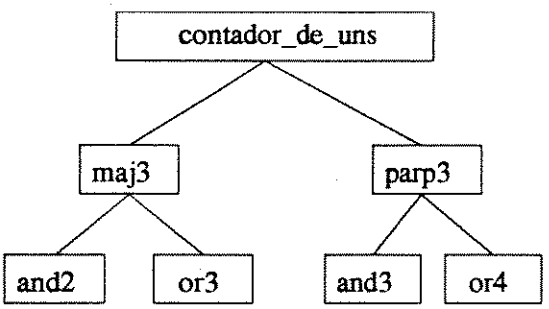


Figura 4.5: Hierarquia de projeto estrutural para o contador de uns.

```
ENTITY maj3 IS
 PORT (X : IN BIT_VECTOR(0 to 2); Z : OUT BIT);
END maj3;

ARCHITECTURE and_or OF maj3 IS
 COMPONENT and2
  PORT (I1, I2 : IN BIT; O : OUT BIT);
 END COMPONENT;

 COMPONENT or3
  PORT (I1, I2, I3 : IN BIT; O : OUT BIT);
 END COMPONENT;

 SIGNAL A1, A2, A3 : BIT;

BEGIN
 G1: and2
     PORT MAP (X(0) , X(1) , A1);
 G2 : and2
     PORT MAP (X(0) , X(2) , A2);
 G3 : and2
     PORT MAP (X(1) , X(2) , A3);
 G4 : or3
     PORT MAP (A1, A2 , A3 , Z);
END and_or;
```

Figura 4.6: Descrição da entidade MAJ3.

```
ENTITY and2 IS
 PORT (I1, I2 : IN BIT; O : OUT BIT);
END and2;

ARCHITECTURE comportamento OF and2 IS
BEGIN
 O <= I1 AND I2;
END comportamento;
```

Figura 4.7: Descrição da entidade AND2.

A seguir explica-se a arquitetura AND_OR da entidade de projeto MAJ3 em detalhes. Um diagrama lógico desta estrutura é mostrado na figura 4.9. Referindo-se novamente à figura 4.6, nota-se que na seção de declaração da arquitetura AND_OR, dois componentes (AND2 e OR3) são declarados. Os comandos PORT especificados são idênticos aos comandos PORT usados nas descrições de interfaces destes componentes (veja figuras 4.7 e 4.8). Depois, três sinais (A1, A2 e A3) são declarados. As saídas das três portas AND devem ser conectadas às entradas da porta OR. A declaração dos sinais A1, A2 e A3 permite representar estas conexões.

Depois da palavra chave "BEGIN", quatro componentes são *instanciados*, isto é, para cada

```
ENTITY or3 IS
 PORT (I1, I2, I3 : IN BIT; O : OUT BIT);
END or3;

ARCHITECTURE comportamento OF or3 IS
BEGIN
 O <= I1 OR I2 OR I3;
END comportamento;
```

Figura 4.8: Descrição da entidade OR3.

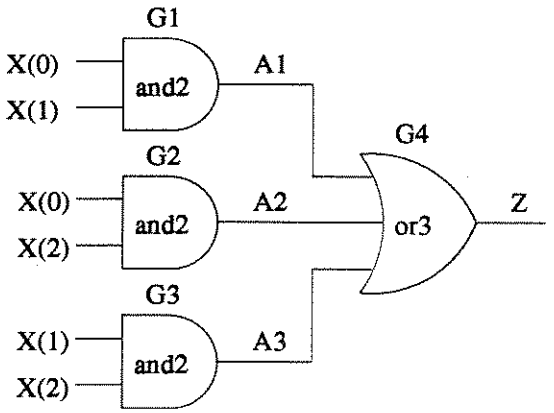


Figura 4.9: Estrutura da porta de função majoritária.

componente é criada uma instância específica de uma entidade de componente geral. Note que cada instanciamento tem um único rótulo associado a ele (isto é, G1, G2, G3 e G4), bem como um comando PORT MAP. O comando PORT MAP cria uma associação entre as entradas e saídas do componente declarado e do componente instanciado.

Para definir a operação da entidade MAJ3 completamente, devem existir descrições de interfaces e arquiteturas para todos os componentes instanciados dentro da arquitetura da entidade MAJ3. As figuras 4.7 e 4.8 mostram estas descrições de interface e arquiteturas.

A estrutura PARP3 é definida de maneira similar. Depois dos componentes MAJ3 e PARP3 terem sido definidos, a arquitetura estrutural para o contador de uns pode ser dada como mostrado na figura 4.10.

```
ARCHITECTURE estrutural OF contador_de_uns IS
 COMPONENT maj3
  PORT (X : IN BIT_VECTOR(0 TO 2); Z: OUT BIT);
 END COMPONENT;
 COMPONENT parp3
  PORT (X : IN BIT_VECTOR(0 TO 2); Z : OUT BIT);
 END COMPONENT;
BEGIN
 component_1 : maj3
  PORT MAP (A, C(1));
 component_2 : parp3
  PORT MAP (A, C(0));
END estrutural;
```

Figura 4.10: Arquitetura estrutural para o contador de uns.

4.3 Processos

Um importante elemento de modelamento em VHDL é o PROCESSO. A figura 4.2 mostra um exemplo de um processo que representa a descrição comportamental do circuito contador de uns. Note que o processo neste caso começa com o comando "PROCESS(A)". O vetor A é o único sinal da lista sensitiva do processo. Sempre que um ou mais dos sinais da lista sensitiva mudar de valor, o processo é ativado e as declarações dentro do bloco do processo são executadas.

4.4 Tipos de Dados

A maioria das linguagens de programação suportam uma variedade de tipos de dados, e VHDL não é exceção. Entretanto, como esta linguagem é usada para representar projetos de hardware em uma variedade de maneiras, a sua característica quanto aos tipos de dados é particularmente importante. Por exemplo, ela permite que se represente um barramento como: (1) um conjunto de bits, (2) um inteiro, ou (3) um código mneumônico.

A tabela 1 mostra os tipos pré-definidos encontrados em VHDL.

Tabela 1 Tipos de Dados Pré-definidos em VHDL

BOOLEAN	INTEGER	CHARACTER
BIT	POSITIVE	STRING
BIT VECTOR	NATURAL	
	REAL	

A coluna da esquerda traz os tipos lógicos. O tipo BOOLEAN consiste dos valores TRUE e FALSE. Todas as declarações IF da linguagem devem testar objetos e expressões deste tipo. O tipo BIT consiste dos valores '0' e '1'. O tipo BIT_VECTOR define um conjunto de bits, por exemplo, BIT_VECTOR(0 to 7).

A coluna do meio da tabela 1 traz os tipos e subtipos aritméticos. Pode ser surpriendente ver o tipo REAL em uma linguagem de descrição de hardware. Entretanto, este tipo é muito útil para descrições comportamentais de alto nível.

A coluna da direita da tabela 1 traz os tipos caracteres. O tipo CHARACTER essencialmente consiste dos caracteres ASCII. O tipo STRING é um conjunto de caracteres.

4.5 Operadores

VHDL tem um conjunto completo de operadores correspondendo aos tipos de dados disponíveis. A tabela 2 mostra um sumário destes operadores.

Tabela 2 Operadores VHDL

Classe	Membros da Classe
Logical	not | and | or | nand | nor | xor
Relational	= | /= | < | <= | > | >=
Adding	+ | - | &
Signing	+ | -
Multiplying	* | / | mod | rem
Miscellaneous	** | abs

4.6 Classes de Objetos

Em VHDL existem três classes de objetos: constantes, variáveis e sinais. Em VHDL, as constantes e variáveis têm as mesmas caracteristicas destes objetos em linguagens convensionais, os objetos sinais é que tornam VHDL uma linguagem própria para pojeto de hardware. Todos os objetos são criados quando eles são declarados.

4.6.1 Constantes e Variáveis

Uma constante é um objeto cujo valor não pode ser mudado. Uma variável é um objeto cujo valor pode ser mudado mas não tem dimensão no tempo.

Exemplos de declarações de constantes:

```
CONSTANT int_vector : BIT_VECTOR(1 to 7) := "00001000";

CONSTANT pi: REAL := 3.14;
```

Exemplos de declaração de variáveis:

```
VARIABLE cont: INTEGER := 0;

VARIABLE addr: BIT_VECTOR(0 to 11);
```

4.6.2 Sinais

Sinais são objetos cujos valores podem ser mudados e têm dimensão no tempo. Os valores dos sinais são mudados por declarações de transferência, por exemplo:

```
A <= B + C AFTER 50 ns;

D <= E NOR C;
```

Note que a declaração de transferência do valor do sinal usa o simbolo <= para se diferenciar do simbolo := ulilizado para a declaração de transferência do valor para as variáveis. No primeiro

exemplo, "after 50 ns" significa que o sinal A potencialmente irá mudar de valor 50 ns após o presente ciclo de simulação. Se diz "potencialmente" porque dentro do modelo pode haver mais de uma declaração de transferência do valor afetando o sinal A.

4.7 Atributos

Atributos de sinais são importantes para o modelamento. Alguns exemplos:

1. S'EVENT é do tipo BOOLEAN e retorna valor TRUE se um evento ocorreu no sinal S durante o ciclo de simulação corrente. É útil para checar se ocorrem mudanças nos valores dos sinais.
2. S'STABLE(T) é to tipo BOOLEAN e retorna valor TRUE se o sinal S vem se mantendo estável nas últimas T unidades de tempo. É escrito como S'STABLE para T igual a zero.
3. S'DELAYED(T) é o valor do sinal S, T unidades de tempo mais cedo. Tem o mesmo tipo de S.

4.8 Funções e Procedimentos

Funções podem ser declaradas em VHDL especificando-se:

1. O nome da função
2. Os parâmetros de entrada, se existir algum
3. O tipo do valor de retorno
4. Qualquer declaração requerida pela própria função
5. Um algoritmo para o cálculo do valor de retorno

Na arquitetura MACRO para a entidade CONTADOR_DE_UNS, as funções MAJ3(X,Y,Z) e PARP(X,Y,Z) foram empregadas. Para se usar estas funções, elas deveriam ter sido declaradas como a seguir:

```
FUNCTION MAJ3(X: BIT_VECTOR(0 to 2)) RETURN BIT is
BEGIN
   RETURN (X(0) AND X(1)) OR (x(0) AND X(2)) OR (X(1) AND X(2));
END MAJ3;
```

A função PARP3 poderia ser declarada de forma similar.

Procedimentos também podem ser escritos em VHDL. Um procedimento é declarado especificando-se:

1. O nome do procedimento
2. Os parâmetros de entrada e de saída, se existir algum
3. Qualquer declaração requerida pelo próprio procedimento
4. Um algoritmo

A figura 4.11 mostra a declaração de um procedimento que conta os zeros e uns de um vetor de 3 bits. Note que a lista de parâmetros especifica as entradas e saídas. O algoritmo implementado é a sequência que vem depois da palavra chave *BEGIN*.

```
PROCEDURE contador_de_uns_e_zeros
  VARIABLE  X : IN BIT_VECTOR(0 TO 2);
  VARIABLE  N_UNS, N_ZEROS : OUT BIT_VECTOR(0 TO 1) is

  VARIABLE NUM1 : INTEGER RANGE 0 TO 3 := 0;
  VARIABLE NUM0 : INTEGER RANGE 0 TO 3 := 0;

BEGIN
  FOR i IN 0 TO 2 LOOP
   IF X(i) = '1' THEN
    NUM1 :=NUM1 + 1;
   ELSE
    NUM0 := NUM0 + 1;
   END IF;
  END LOOP;
  CASE NUM1 IS
   WHEN 0 => N_UNS := "00";
   WHEN 1 => N_UNS := "01";
   WHEN 2 => N_UNS := "10";
   WHEN 3 => N_UNS := "11";
  END CASE;
  CASE NUM0 is
   WHEN 0 => N_ZEROS := "00";
   WHEN 1 => N_ZEROS := "01";
   WHEN 2 => N_ZEROS := "10";
   WHEN 3 => N_ZEROS := "11";
  END CASE;
END contador_de_uns_e_zeros;
```

Figura 4.11: Exemplo de procedimento.

4.9 Declarações de Controle

As declarações de controle que são utilizadas neste trabalho são:

- IF
- CASE
- LOOP

A forma completa da declaração IF é como a seguir:

IF CONDIÇÃO_1 then

sequência de declarações

ELSIF CONDIÇÃO_2 then

sequência de declarações

ELSE

sequência de declarações

END IF;

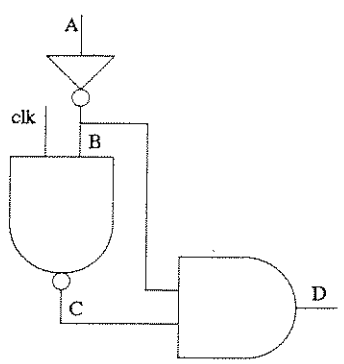
Ambas, CONDIÇÃO_1 e CONDIÇÃO_2 devem ser do tipo BOOLEANO. Qualquer número de cláusulas *ELSIF* pode ser incluido. As cláusulas *ELSIF* e *ELSE* são opcionais.

A figura 4.11 mostra exemplos de declarações *CASE* e *LOOP*. Não se comenta as demais declarações de controle por serem de pouco interesse aqui.

4.10 Execução Concorrente dos Comandos VHDL

Em linguagens típicas de programação tais como C ou Pascal, os comandos de um programa são executados um depois do outro em uma ordem especificada. A ordem de execução é determinada pela ordem de declaração dos comandos no arquivo fonte do programa. Dentro de uma arquitetura VHDL não há nenhuma ordem especificada para a execução dos comandos. A ordem de execussão é determinada unicamente pela ocorrência de eventos nos sinais aos quais os comandos são sensíveis.

Ilustra-se isto com o exemplo de circuito simples e seu modelo VHDL mostrado na figura 4.12.




```
1  ENTITY circuito_simples;
2    PORT(clk, A : IN BIT;  D: OUT BIT);
3  END circuito_simples;

4  ARCHITECTURE comportamento OF circuito_simples IS
5    SIGNAL B, C : BIT;
6  BEGIN
7    B <= NOT(A);
8    C <= B NAND clk;
9    D <= B AND C;
10 END comportamento;
```

Figura 4.12: Circuito simples e seu modelo em VHDL.

Examina-se o comando da linha 9 da figura 4.12, como mostrado abaixo.

D <= B AND C;

O sinal D é associado a lógica AND dos sinais B e C pelo simbolo "<=". Este comando será executado sempre que um dos sinais B, C ou ambos sofrerem uma ocorrência de evento. Um evento

em um sinal é uma mudança no seu valor. Portanto o comando acima é sensível aos sinais B e C, assim como qualquer comando é sensível aos sinais existentes no lado direito do simbolo "<=".

Continua-se a análise da execução do modelo da figura 4.12, supondo que as suas portas lógicas apresentam tempos de atraso nulos.

Em um circuito real, os sinais se propagam pelo circuito de forma concorrente. Neste circuito por exemplo, se o sinal A sofre uma transição, o sinal B também irá sofrer uma transição. Como o sinal B é entrada para a porta AND e para a porta NAND, então ao simular esse modelo, o simulador deve avaliar essas duas portas ao mesmo tempo, já que é assim que ocorre em um circuito real. Porém, como o simulador dispõe de apenas uma CPU, na prática essas portas são avaliadas uma de cada vez após uma transição no sinal B. Sendo assim, o simulador deve escolher uma delas para avaliar primeiro, mas será visto que escolhas diferentes podem levar a resultados diferentes na simulação e em seguida será apresentada a solução existente em VHDL para superar este problema.

Suponha que inicialmente os sinais estejam estáveis e com os seguintes valores iniciais:

A = '1', clk = '1', B = '0', C = '1' e D = '0'

Em um instante de tempo qualquer o sinal A sofre uma transição (A→'0'). Se a porta AND for avaliada primeiro tem-se o seguinte resultado:

AND primeiro	
avalia	resultado
B <= NOT(A) B<=NOT('0')	B = '1'
D <= B AND C D <= '1' AND '1'	D = '1'
C <= clk NAND B C <= '1' NAND '1'	C = '0'
D <= B AND C D <= '1' AND '0'	D = '0'

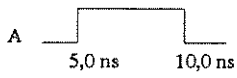
Agora, avaliando a porta NAND primeiro tem-se:

NAND primeiro	
avalia	resultado
B <= NOT(A) B <= NOT('0')	B = '1'
C <= clk NAND B C <= '1' NAND '1'	C = '0'
D <= B AND C D <= '1' AND '0'	D = '0'

Para essas duas sequências, tem-se dois resultados diferentes. Avaliando-se a porta AND primeiro, o sinal D sofre uma transição de '0' para '1' e depois retorna ao valor '0'. Avaliando-se a porta NAND primeiro o sinal D não sofre nenhuma transição.

A solução existente na linguagem VHDL para esse problema é o *Mecanismo de Atraso Delta*. A

figura 4.13 mostra como ele funciona supondo que o circuito simples tem como entrada o sinal A mostrado na mesma figura através de um diagrama temporal.



Mecanismo de Atraso Delta

tempo 5,0 ns	delta(1)	A <= '1' avalia B <= NOT(A);

	delta(2)	B <= '0'
		avalia D <= B AND C;
		avalia C <= clk NAND B;

	delta(3)	D <= '0' C <= '1' avalia D <= B AND C;
	delta(4)	D <= '0'
tempo 5,1 ns	nada	
tempo 5,2 ns	nada	
tempo 10,0 ns	delta(1)	A <= '0' avalia B <= NOT(A);

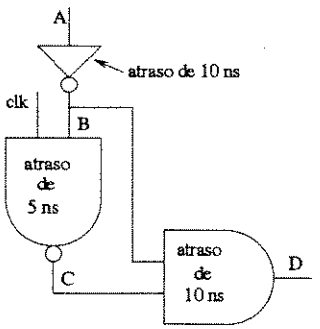
	delta(2)	B <= '1'
		avalia D <= B AND C;
		avalia C <= clk NAND B;

	delta(3)	D <= '1' C <= '0' avalia D <= B AND C;
	delta(4)	D <= '0'
tempo 10,1 ns	nada	
tempo 10,2 ns	nada	

Figura 4.13: Mecanismo de Atraso Delta.

Pode-se observar que o tempo é incrementado de 0,1 em 0,1 ns (esse valor pode ser alterado no simulador, mas se mantém fixo durante a simulação). Durante um período de 0,1 ns de *tempo não real*, o simulador avalia todo o circuito e só incrementa o tempo depois que o circuito estiver estável. Isto é chamado de ciclo de simulação e pode ser visto entre duas linhas horizontais da figura 4.13. Essas linhas horizontais representam os instantes de tempo em que os resultados obtidos após 0,1 ns são anunciados pelo simulador. Dessa forma os resultados de tornam independentes da ordem de avaliação das portas lógicas do circuito. Por exemplo, entre os tempos 10 e 10,1 ns o sinal D sofre uma transição para '1' e retorna para '0', mas como os resultados só são anunciados no tempo 10,1 ns, então esta transição não é mostrada nos resultados, o que equivaleria ter avaliado a porta NAND primeiro.

Até aqui, considerou-se que as portas lógicas do circuito exemplo apresentam tempo de atraso nulo. Para simular esse circuito de maneira mais precisa, levando em consideração os tempos de atraso provocados por capacitâncias nas portas lógicas, deve-se usar o comando AFTER. A figura 4.14 mostra um modelo para o circuito no qual foram levados em consideração tempos de atraso. Considera-se tempos de atraso de 10, 5 e 10 ns para as portas INVERSOR, NAND e AND respectivamente.




```
ENTITY circuito_simples_com_atraso;
 PORT(clk, A : IN BIT; D: OUT BIT);
END circuito_simples_com_atraso;

ARCHITECTURE comportamento OF circuito_simples_com_atraso IS
  SIGNAL B, C : BIT;
BEGIN
  B <= NOT(A) AFTER 10 ns;
  C <= B NAND clk AFTER 5 ns;
  D <= B AND C AFTER 10 ns;
END comportamento;
```

Figura 4.14: Circuito simples com atrasos e seu modelo em VHDL.

Nesse caso, o *Mecanismo de Atraso Delta* funciona como mostrado na figura 4.15.

Havia-se falado que o grande problema dos métodos convencionais de projeto de hardware, era o fato do projeto ter que ser encarado gate por gate, o que é um absurdo para projetos com centenas de milhares de gates. Analisou-se até aqui um exemplo com apenas 3 gates , e por isso não foi possível entender a vantagem do VHDL.

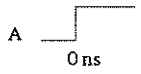
Para entender melhor o que significa projetar em alto nível, acrescenta-se ao circuito um latch, como mostrado na figura 4.16.

A figura 4.17 mostra um modelo escrito em VHDL para esse circuito, no qual o comportamento do latch é modelado por um "PROCESS".

Pode-se entender como o comando "PROCESS" é avaliado comparando-o com as portas INVERSOR, NAND E AND do circuito. Na verdade, é como se essas portas também fossem PROCESSOS. A porta AND por exemplo é avaliada toda vez que o sinal B ou o sinal C ou ambos sofrerem uma transição, já que o comando que modela a porta AND é sensível aos sinais B e C. No modelo da figura 4.17, logo depois da palavra PROCESS, aparecem entre parenteses os sinais E e D que compõem a lista sensitiva do PROCESSO. Portanto, toda vez que o sinal E ou o sinal D ou ambos sofrerem uma transição, o processo será avaliado.

Avaliar um processo significa que todas as linhas de comandos que ficam entre os comandos "BEGIN" e "END PROCESS" são executadas em série. Essas linhas de comandos ditam o comportamento do processo. Nesse caso o PROCESSO dita o comportamento de um simples latch, mas pode-se criar processos para modelar circuitos (ou partes de circuitos) bem mais complexos.

Seria possível representar a função do latch usando-se apenas duas portas NAND ao invés de se escrever um PROCESSO. De fato, nesse caso específico isso seria mais simples, mas da mesma forma que se usa um PROCESSO para representar duas portas lógicas, pode-se usar um outro para representar, diga-se, 200 gates. Aí a vantagem de escrever PROCESSOS para representar funções de circuitos fica clara.



Mecanismo de Atraso Delta

tempo 0,0 ns	delta(1)	A <= '0' avalia B <= NOT(A); - B é programado para asumir o valor '1' no tempo 10,0 ns.
tempo 0,1 ns	nada	
tempo 0,2 ns	nada	
⋮		

tempo 10,0 ns	delta(1)	B <= '1'
		avalia D <= B AND C;
		avalia C <= clk NAND B;
		- D é programado para assumir
		o valor '1' no tempo 20,0 ns.
		- C é programado para assumir
		o valor '0' no tempo 15,0 ns.

tempo 10,1 ns	nada	
tempo 10,2 ns	nada	
⋮		
tempo 15,0 ns	delta(1)	C <= '0' avalia D <= B AND C; - A programação para D assumir o valor '1' no tempo 20,0 ns é cancelada.
tempo 15,1 ns	nada	
⋮		

Figura 4.15: Mecanismo de Atraso Delta para circuito com atrasos de tempo.

4.11 Sumário

Fez-se uma introdução à linguagem VHDL. O propósito aqui foi o de dar um entendimento básico da linguagem, restringiu-se particularmente às caractiristicas que são utilizadas nos modelos desenvolvidos no capítulo seguinte. A apresentação aqui é obviamente incompleta mas será suficiente como suporte para o ententimento dos modelos a serem analisados.

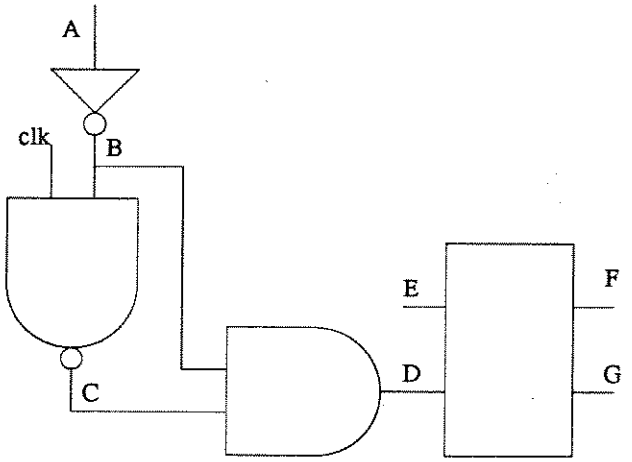


Figura 4.16: Circuito simples com latch

```
ENTITY simples_com_latch;
 PORT(clk, A, E : IN BIT; F, G: OUT BIT);
END simples_com_latch;

ARCHITECTURE comportamento OF simples_com_latch IS
  SIGNAL B, C, D: BIT;
BEGIN
  B <= NOT(A);
  C <= B NAND clk;
  D <= B AND C;
  latch: PROCESS(E, D)
   BEGIN
     IF (D = '1') THEN
      IF (E = '1') THEN
        F <= '0';
        G <= '1';
      ELSE
        F <= '1';
        G <= '0';
      END IF;
     END IF;
   END PROCESS;
END comportamento;
```

Figura 4.17: Modelo VHDL para circuito simples com latch

Capítulo 5

Projetos em Alto Nível para Blocos Funcionais da Hierarquia Digital Síncrona

5.1 Introdução

No capítulo 3 estudou-se os blocos lógicos do multiplexador generalizado recomendado pelo CCITT. Eles representam os passos necessários para se montar várias cargas úteis e multiplexa-las para se formar um sinal STM-N. Neste capítulo criam-se modelos de alto nível em linguagem VHDL para alguns destes blocos. Neste contexto, alto nível significa que os modelos não serão apresentados na forma de estruturas de portas lógicas ou de circuitos integrados, os modelos serão apresentados na forma de algoritmos que determinam a evolução dos sinais de saída em relação aos sinais de entrada de cada bloco.

5.1.1 Especificação

São as seguintes as funções dos blocos funcionais a serem projetados:

- No sentido da transmissão (multiplexagem):
  - Adaptar, em modo flutuante, containers VC-12 em unidades afluentes TU-12
  - Multiplexar conjuntos de 63 unidades afluentes TU-12 para formar a carga útil de um container VC-4.
  - Adaptar o container VC-4 em uma unidade administrativa AU-4.
  - Multiplexar a unidade administrativa AU-4 em um sinal STM-1.
- No sentido da recepção (demultiplexagem):
  - Todos os itens anteriores no sentido inverso.

A figura 5.1 mostra os blocos funcionais arranjados de forma a compor um equipamento terminal de linha. Esta figura serve apenas para ilustrar um dos possíveis arranjos dos blocos funcionais. À frente mostram-se outros arranjos que os blocos funcionais podem assumir.

Existem algumas diferenças entre o diagrama de blocos da figura 5.1 e o diagrama funcional do CCITT visto no capítulo 3. São elas:

- Na figura 5.1 os blocos funcionais foram divididos em blocos de recepção e blocos de transmissão.
- O bloco DEMUX que não existe no diagrama funcional do CCITT, aqui foi criado com as funções de fazer a conversão do sinal STM-1 recebido de serial para paralelo (bit para byte).
- O bloco MUX que não existe no diagrama funcional do CCITT, aqui foi criado com as funções de fazer a conversão do sinal STM-1 a ser transmitido de paralelo para serial (byte para bit).
- Alguns blocos geradores de sinais de controle que eram implícitos no diagrama funcional do CCITT, foram definidos na figura 5.1.
- Os blocos de conexão, proteção e interfaces físicas não constam na figura 5.1 por necessidade de simplificação do modelo.

5.1.2 Metodologia

Os blocos MUX e DEMUX da figura 5.1 foram inspirados em circuitos integrados apresentados em [1]. Por isso não serão criados modelos VHDL para estes blocos mas serão feitas análises dos seus funcionamentos.

Os blocos de terminação (RST, MST, HPT e LPT) não são analisados. Da forma como foram colocados na figura 5.1, eles não geram nenhum sinal de controle e se aproveitam dos sinais gerados pelos demais blocos para executarem suas tarefas de processar, ler e inserir bytes de supervisão dos quadros de suas respectivas ordens.

Os blocos LPA_R e LPA_T que fazem a adaptação dos sinais afluentes nos containers VC-12 também não serão projetados. A razão para isto é que, como foi visto no capítulo 2, existem várias formas de se mapear sinais afluentes, por isso prefere-se aqui, deixar indefinida a forma de mapeamento.

Serão criados modelos em linguagem VHDL para os demais blocos. Cada um deles será detalhado em um diagrama de blocos onde cada bloco será representado por processos ou linhas de comandos em linguagem VHDL. No final, os modelos serão interfaceados para a simulação e validação.

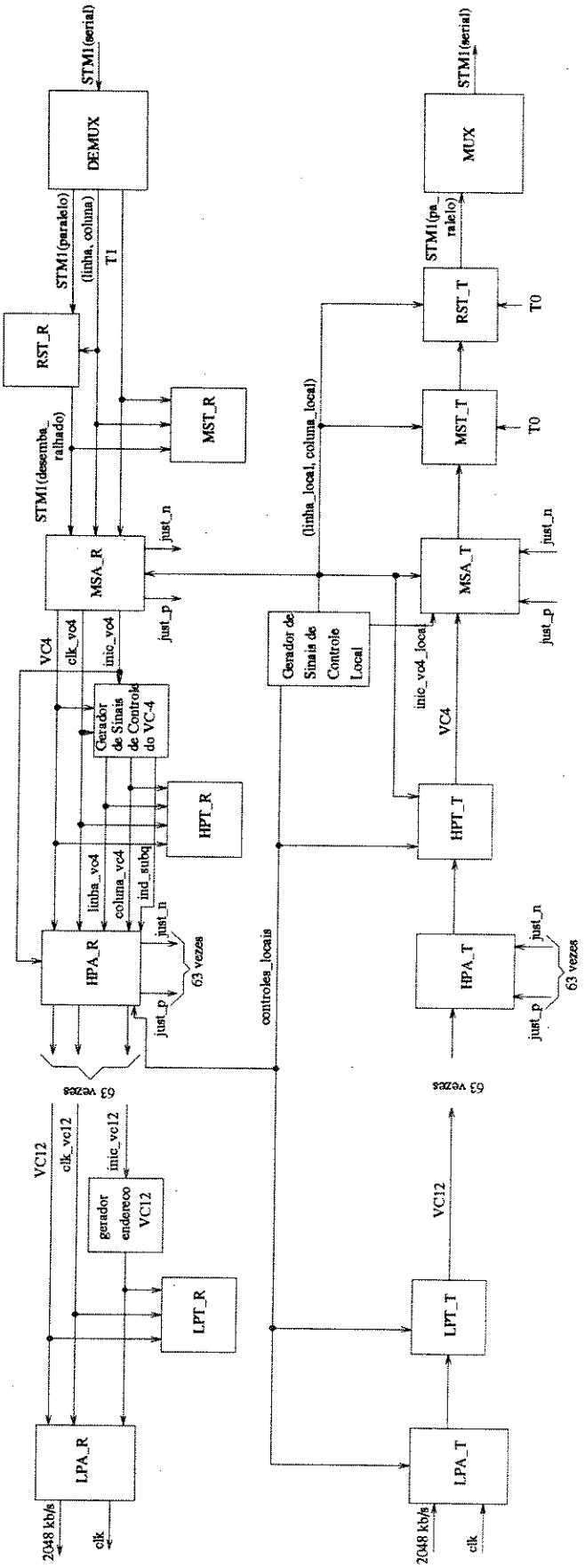


Figura 5.1: Arranjo dos blocos funcionais HDS para terminal de linha.

5.2 Bloco DEMUX

A figura 5.2 mostra o detalhamento para o bloco DEMUX. A seguir define-se a função de cada um de seus blocos.

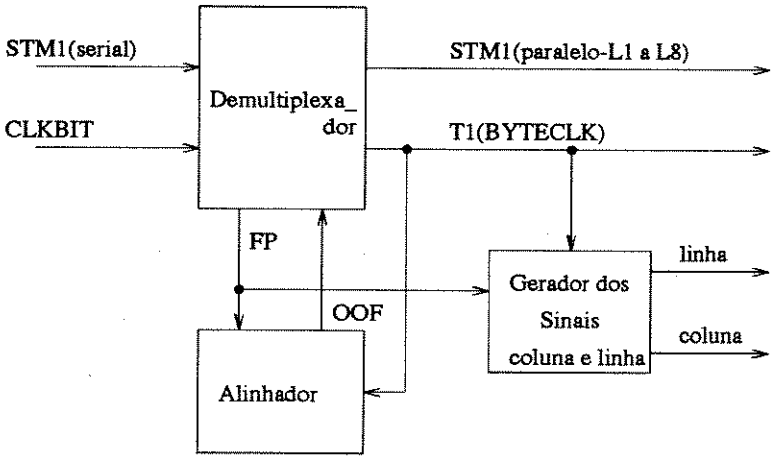


Figura 5.2: Detalhamento do Bloco DEMUX

5.2.1 Demultiplexador

Em [1] mostra-se a estrutura de um circuito integrado que recebe o sinal STM-N na forma serial e o demultiplexa para deixa-lo na forma paralela. A figura 5.3 ilustra esta estrutura que tem também a função de detecção do alinhamento do quadro STM-N através do detector de padrão A1A2A2. Aqui o interesse é processar sinais STM-1, por isso de agora em diante o texto se refere ao sinal STM-1 embora o circuito integrado que se analisa seja capaz de processar sinais STM-N genéricos.

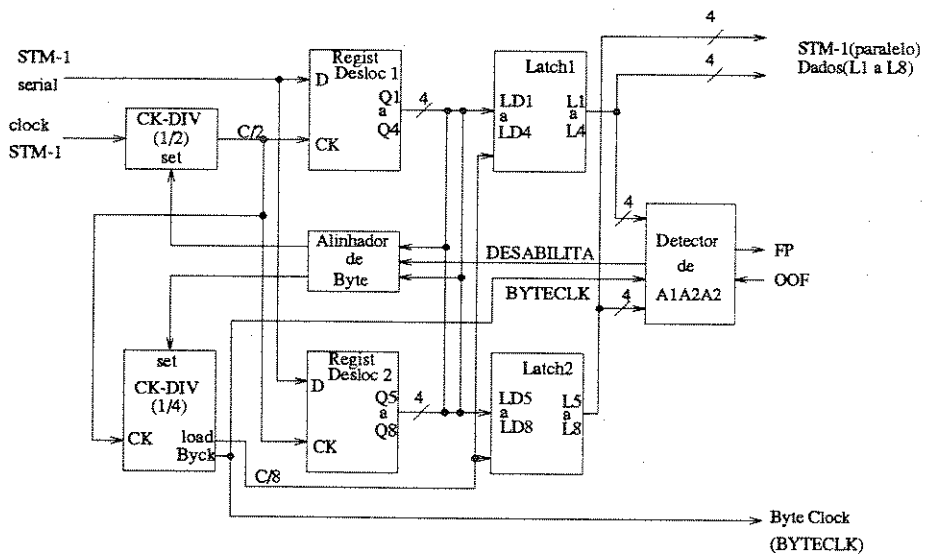


Figura 5.3: Demultiplexador SDH com Alinhador de Byte e Detector de Padrão A1A2A2.

Em um quadro STM-1 os 6 primeiros bytes (3 A1 + 3 A2) são bytes de alinhamento de quadro. Isto foi mostrado no capítulo 3 onde A1 = 11110110 e A2 = 00101000. No esquema apresentado em

[1], o alinhamento é obtido em 2 etapas. A primeira etapa é o alinhamento de byte e a segunda é a detecção de alinhamento de quadro.

O alinhamento de byte é um processo que alinha as saídas do demultiplexador com os bytes do quadro STM-1. Isto pode ser obtido identificando-se os bytes A1 do quadro STM-1. A detecção de alinhamento é um processo que detecta o contorno do quadro STM-1 recebido. Isto é conseguido identificando-se um padrão válido de alinhamento, A1A2A2, no começo de cada quadro STM-1. O padrão A1A2A2 de alinhamento é selecionado porque ele marca a transição dos bytes A1's para os bytes A2's na sequência de bytes do quadro STM-1, e ele pode ser facilmente detectado usando se um circuito simples de detecção de quadro. Além disso a probabilidade de uma imitação aleatória destes 24 bits é de 2^{-24}. Isto garante um pequeno tempo (médio) de realinhamento e infrequentes falsos alinhamentos.

Na entrada de dados em alta velocidade(sinal serial), os bits do sinal STM-1 são armazenados em dois registradores de deslocamento, utilizando o clock dividido por 2, C/2. Os bits entram no registrador de deslocamento 1 durante a borda de subida de C/2 e entram no registrador de deslocamento 2 durante a borda de descida de C/2. Os dois registradores são construidos de tal forma que os períodos de extração dos seus bits são cuidadosamente alinhados. As saídas dos registradores de deslocamento são então carregadas em paralelo para dentro de 2 latches de 4 bits utilizando-se o relógio C/8. Ambos, C/2 e C/8 são gerados dentro do CI utilizando divisores de freqência para dividir por 2 e por 4 respectivamente. Estes relógios são a chave para o controle de temporização na operação do demultiplexador. Finalmente a saída de 8 bits dos latches são enviadas para fora do CI para o processamento paralelo do sinal STM-1.

Durante o processo de recuperação do alinhamento, o alinhador de byte procura um byte A1 nas saídas dos registradores de deslocamento (1 e 2), e seta os circuitos divisores de frequência (1/2 e 1/4) para o estado apropriado de tal forma que as saídas dos latches (1 e 2) estejam apresentando bytes do quadro STM-1 a cada pulso de C/8. O alinhador de byte consiste simplesmente de comparadores e de flip-flops. Estes comparadores são usados para se detectar bytes A1, e gerar o sinal set para os divisores de frequência.

Uma vez que o alinhamento de bytes foi conseguido, o alinhamento de quadro pode ser atingido detectando-se o padrão A1A2A2, na saída do demultiplexador. O circuito requerido para desempenhar esta função inclui um comparador de byte A1, um comparador de byte A2 e um gerador de pulso de alinhamento, como mostrado na figura 5.4. Os comparadores de A1 e A2, construidos utilizando-se portas lógicas simples, comparam as saídas dos latches com A1 (11110110) e A2 (00101000). Se for detectado um A1 seguido por 2 A2 consecutivos, o gerador de pulso se alinhamento irá desabilitar o alinhador de byte, de tal forma que nenhum outro reajustamento de tempo irá ocorrer no demultiplexador. Entretando o gerador de pulso de alinhamento irá também enviar um pulso de alinhamento de quadro FP, para fora do CI afim de sincronizar o restante do receptor SDH. A figura 5.5 mostra o diagrama de tempo para o detector de alinhamento durante o processo de alinhamento de quadro, onde BYTECLK é o clock de byte gerado pelo divisor de frequência (1/4) (este clock é chamado de T1 nos modelos VHDL que se vai apresentar), L1 a L8 (sinal STM-1 paralelo) são as saídas dos latches, A1M e A2M são saídas dos comparadores de A1 e A2 respectivamente, FP é o pulso de alinhamento, DESABILITA (ativo quando alto) é o sinal temporal utilizado para desabilitar o alinhador de bytes quando um padrão A1A2A2 é detectado. Note que nesta figura, L1 carrega o bit mais significativo e L8 o menos significativo do byte STM-1. O indicador de perda de alinhamento OOF (analisado a seguir) é alto durante o tempo de detecção.

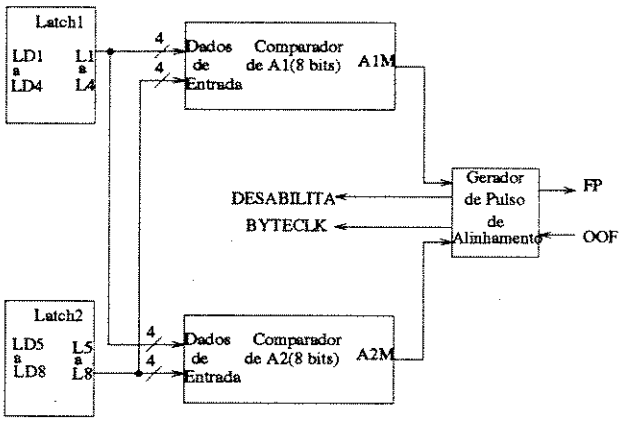


Figura 5.4: Detector de Padrão A1A2A2.

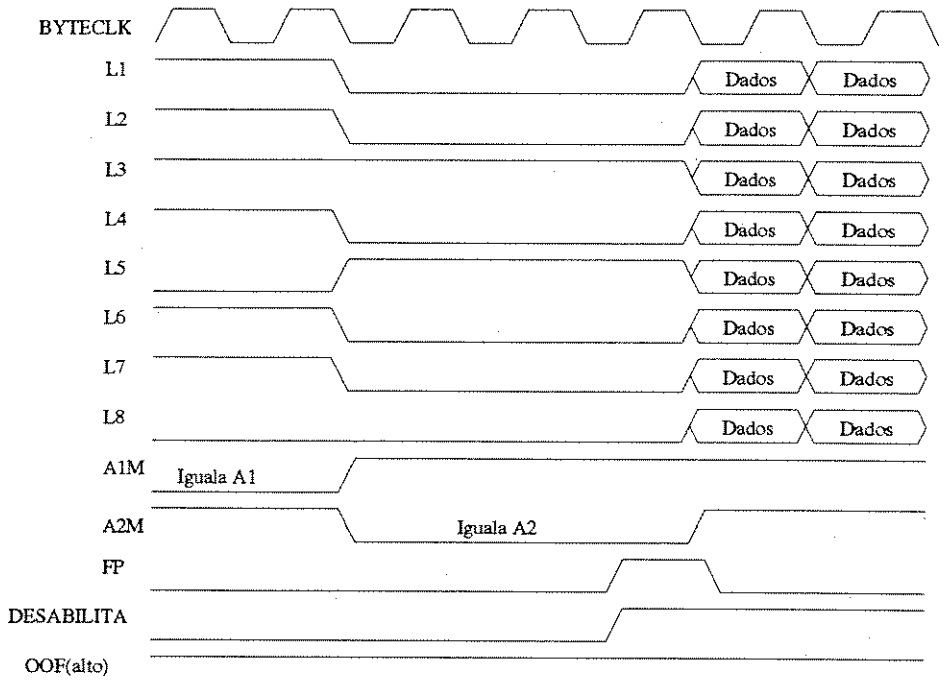


Figura 5.5: Diagrama Temporal para o Detector de Padrão A1A2A2.

5.2.2 Alinhador

O sinal FP gerado internamente no demultiplexador pode ser monitorado externamente por um processador paralelo de sinal. Se padrões válidos de alinhamento A1A2A2 são detectados em 2 quadros STM-1 consecutivos, o Alinhador se declara "alinhado", e seta o sinal OOF, que é entrada do demultiplexador, para nível lógico zero. Isto indica que o Alinhador está em alinhamento de quadro. Uma vez que o Alinhador esteja no estado alinhado normal, ele terá que detectar pelo menos 4 padrões de alinhamento inválidos antes dele poder se declarar "fora de alinhamento" (OOF). Isto se faz para garantir que o Alinhador não irá considerar frequentes falsas perdas de alinhamento devido a presença de erros de transmissão. A figura 5.6 mostra um diagrama de estados para a alinhamento de um sistema de recepção SDH. Uma vez no estado "fora de alinhamento" o Alinhador irá fazer o sinal OOF ir para nível lógico zero na entrada do demultiplexador. Isto irá abilitar o circuito do alinhador de byte do demultiplexador, e começar a procura de um novo quadro STM-1.

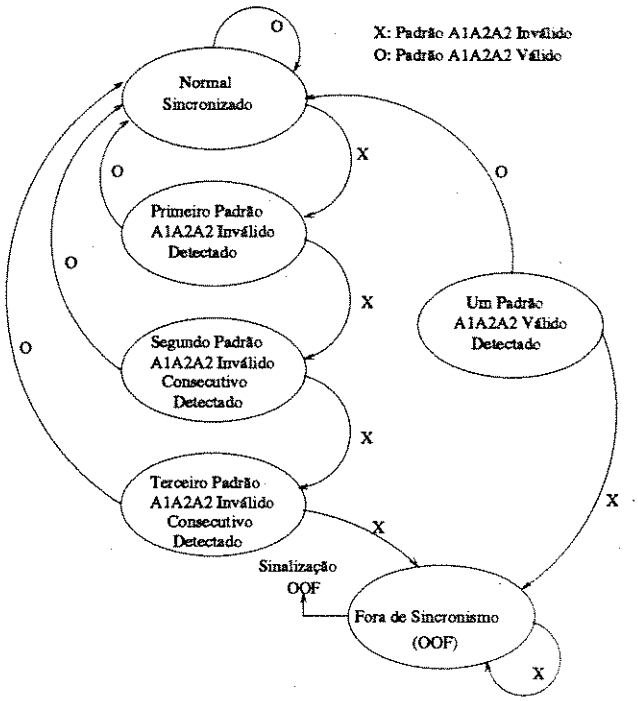


Figura 5.6: Diagrama de Estados para Alinhamento do Sinal STM-1

5.2.3 Gerador dos sinais *coluna* e *linha*

A figura 5.7 mostra o diagrama temporal dos sinais *coluna* e *linha* que são gerados por este bloco em relação ao relógio *T1* (BYTECLK) e o sinal *STM1*. Observa-se na figura que estes sinais foram arranjados de forma semelhante a um quadro de 9 linhas por 270 colunas afim de destacar a relação com o quadro STM-1.

O sinal *linha* é do tipo inteiro[1] e evolui de 0 a 8, sendo capaz de identificar as 9 linhas do quadro STM-1. O sinal *coluna* também é do tipo inteiro e evolui de 0 a 269, sendo capaz de identificar as 270 colunas do quadro STM-1. O relógio *T1* apresenta 2430 (9×270) pulsos em 125μs, portanto um pulso para cada byte do quadro STM-1. Finalmente, o sinal *STM1* provém de um barramento de 8 bits (L1 a L8 do demultiplexador). Ele traz um novo byte do quadro STM-1 a cada pulso de *T1*. Observa-se que durante as bordas de subida do relógio *T1* os sinais *coluna, linha* e *STM1* estão estáveis, portanto estes são os instantes nos quais os bytes do sinal *STM-1* podem ser lidos.

[1] Deve-se lembrar que em linguagem VHDL pode-se definir sinais do tipo inteiro. Isto torna o modelo mais didático, porém na prática estes sinais devem ser imaginados como sendo barramentos binários.

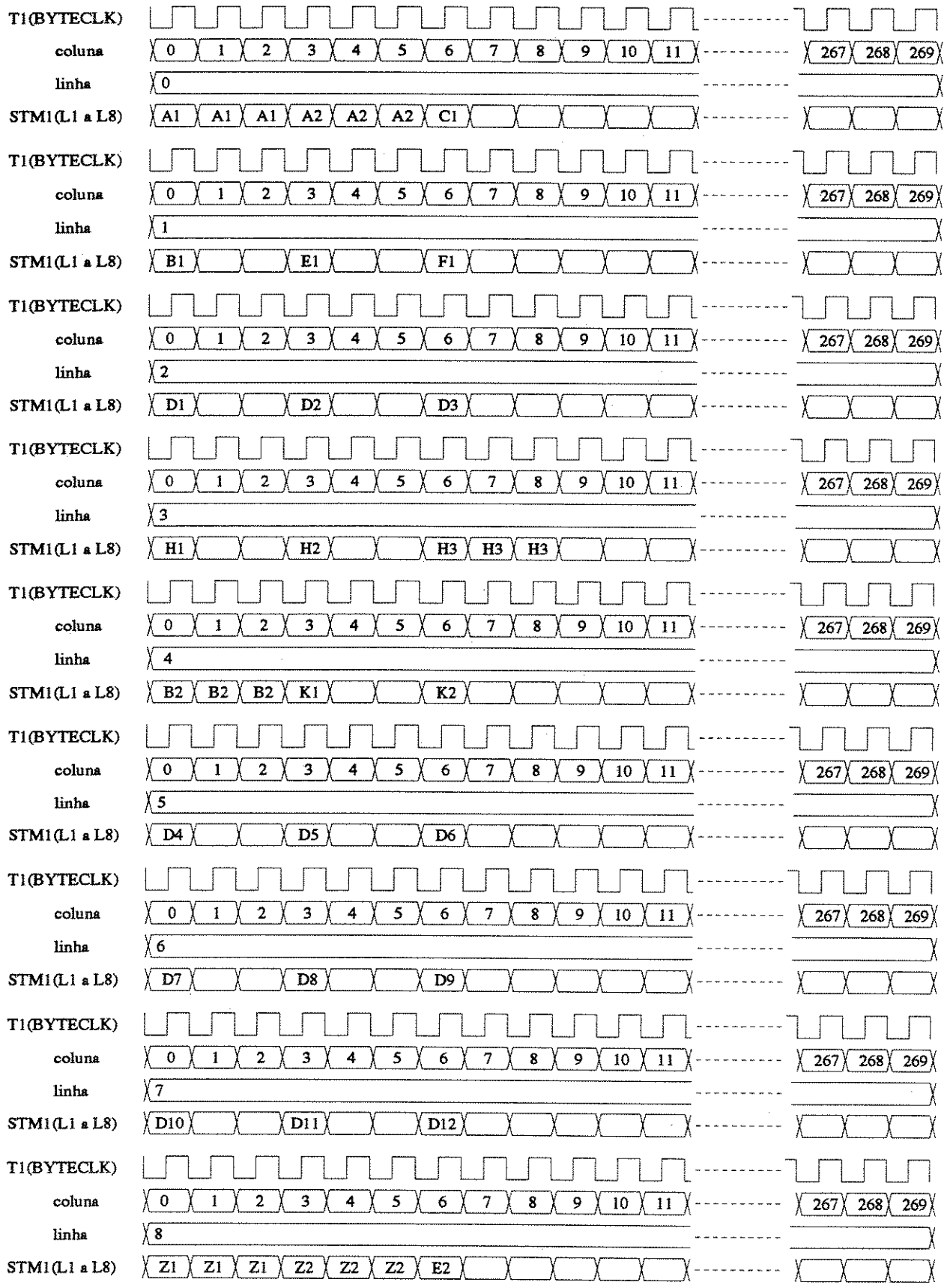


Figura 5.7: Diagrama temporal dos sinais *coluna* e *linha* em relação ao relógio *T1* (BYTECLK) e o sinal *STM1*.

5.3 Adaptação de Seção Multiplex na Recepção (SA_R)

A função de Adaptação de Seção Multiplex na Recepção é detalhada através do diagrama de blocos da figura 5.8. A seguir analisa-se cada um dos blocos da figura 5.8 através de diagramas temporais de seus sinais de entrada e de saída e dos comandos VHDL que os modelam.

Nota: Além dos sinais *coluna*, *linha* e do relógio *T1*, este bloco tem como sinais de entrada os sinais *coluna_local*, *linha_local* e o relógio *T0* que estão ilustrados na figura 5.9.

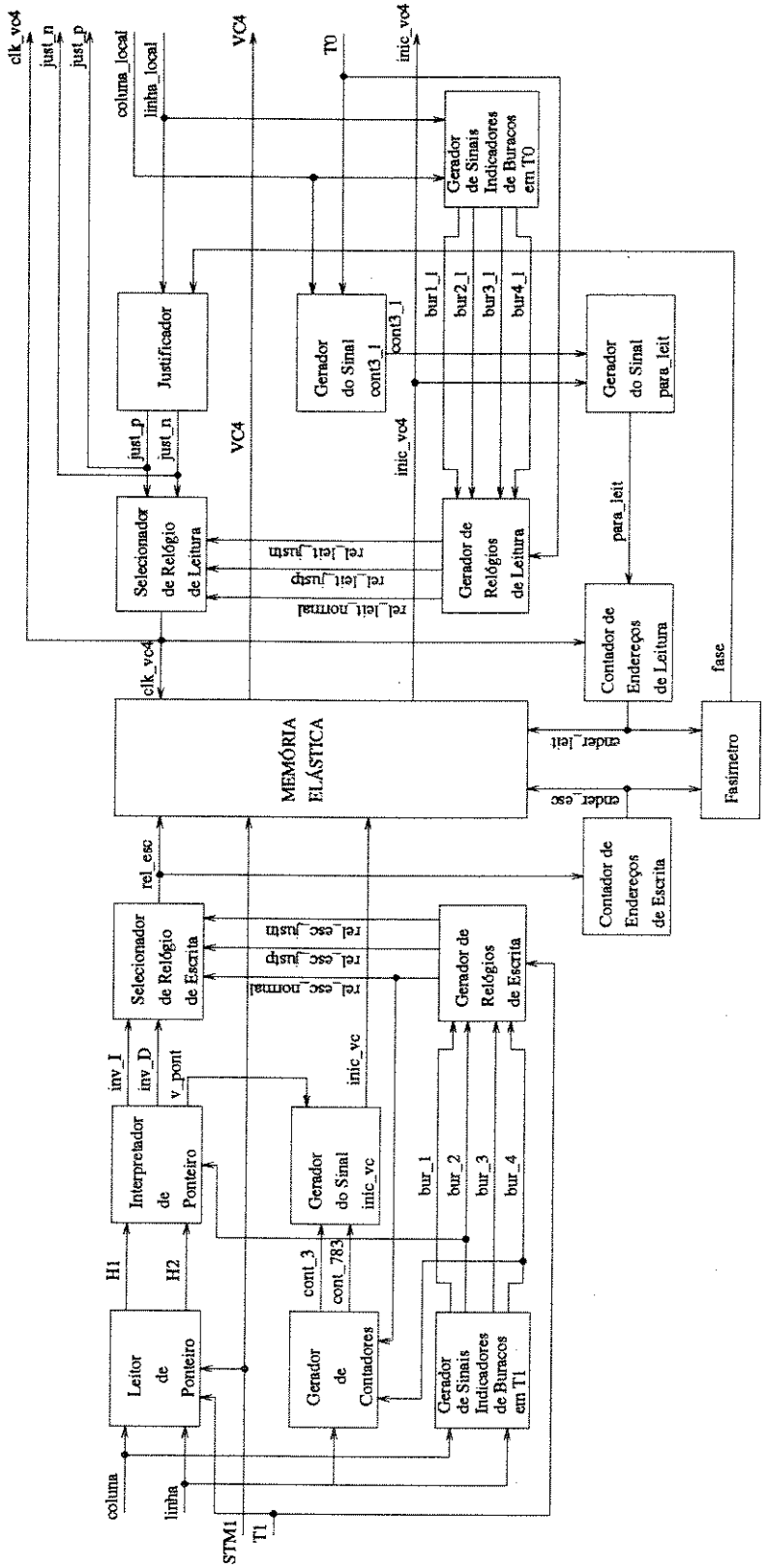


Figura 5.8: Detalhamento do bloco SA_R.

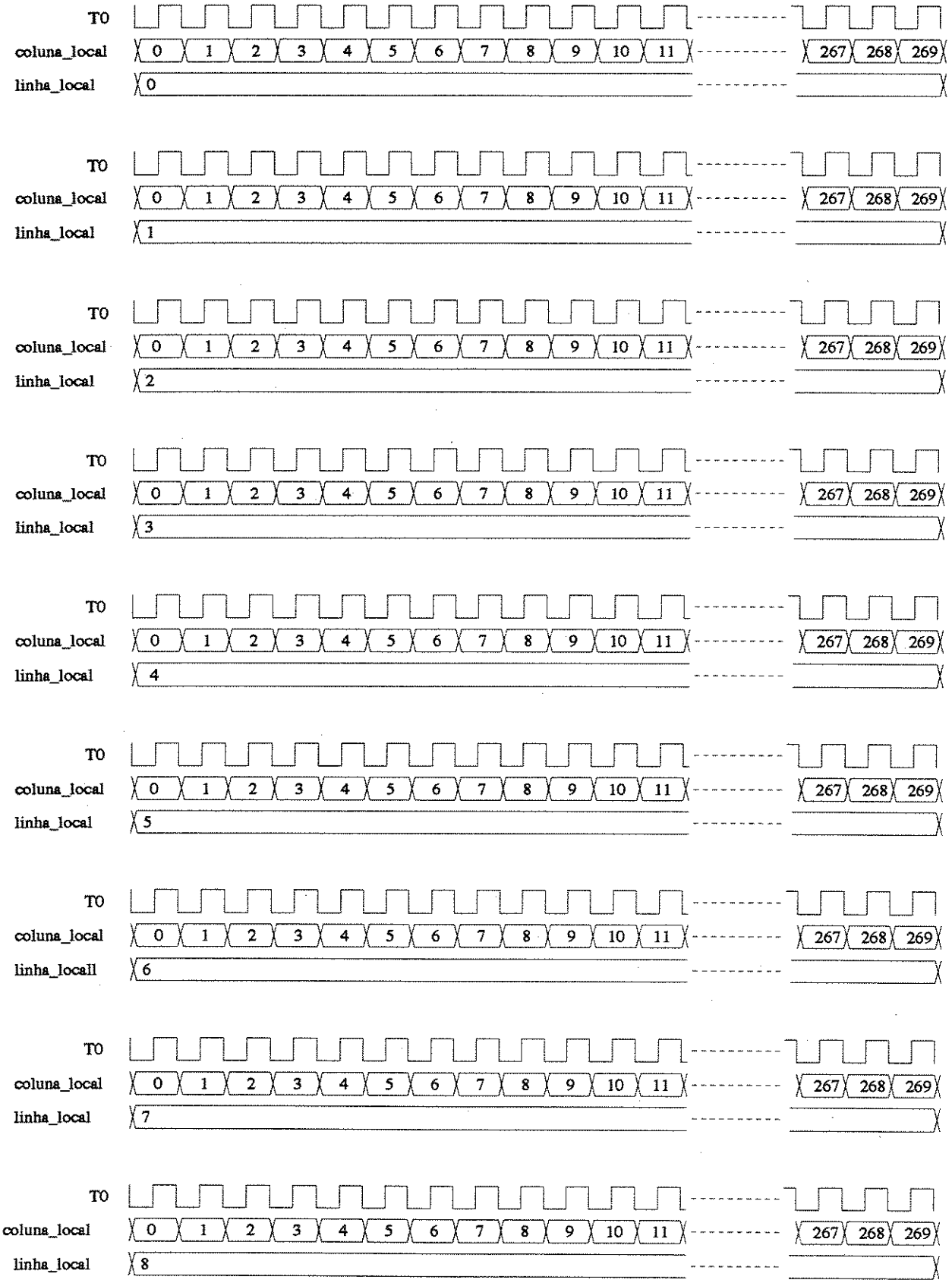


Figura 5.9: Diagrama temporal dos sinais *coluna_local* e *linha_local* em relação ao relógio *T0*.

5.3.1 Leitor de Ponteiro

O processo abaixo modela este bloco.

```
leitura_ponteiro : PROCESS(T1)
BEGIN
    IF (T1 = '1') AND (T1'EVENT) THEN
        IF (linha = 3) THEN
            CASE coluna IS
                WHEN 0 => H1 <= STM1;
                WHEN 3 => H2 <= STM1;
                WHEN OTHERS => NULL;
            END CASE;
        END IF;
    END IF;
END PROCESS;
```

A função deste bloco é ler do sinal *STM1* os bytes do ponteiro (H1 e H2). Isto está ilustrado no diagrama temporal da figura 5.10.

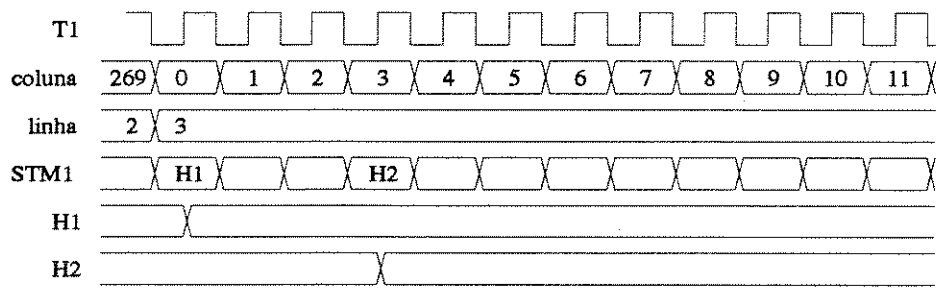


Figura 5.10: Diagrama temporal ilustrando a leitura do ponteiro.

Durante a borda de subida de *T1* o sinal *H1* é o resultado da leitura do sinal *STM1* quando (*linha = 3, coluna = 0*). Durante a borda de subida de *T1*, o sinal *H2* é o resultado da leitura do sinal *STM1* quando (*linha = 3, coluna = 3*).

5.3.2 Gerador de Sinais Indicadores de Buracos em *T1*

As linhas de comandos abaixo modelam este bloco.

```
bur_1 <= '1' WHEN coluna >= 0 AND coluna <= 8 AND linha /= 3 ELSE
         '0';
bur_2 <= '1' WHEN coluna >= 0 AND coluna <= 5 AND linha = 3 ELSE
         '0';
bur_3 <= '1' WHEN coluna >= 6 AND coluna <= 8 AND linha = 3 ELSE
         '0';
bur_4 <= '1' WHEN coluna >= 9 AND coluna <= 11 AND linha = 3 ELSE
         '0';
```

Os diagramas temporais das figuras 5.11 e 5.12 ilustram os sinais gerados por este bloco. Como

se vê a seguir, estes sinais são necessários na geração do relógio de escrita em memória elástica. Este relógio escreve em uma memória elástica somente os bytes do quadro VC-4 transportado pelo sinal *STM1*.

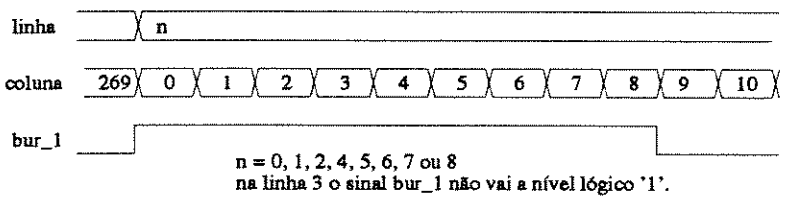


Figura 5.11: Diagrama temporal do sinal *bur_1*

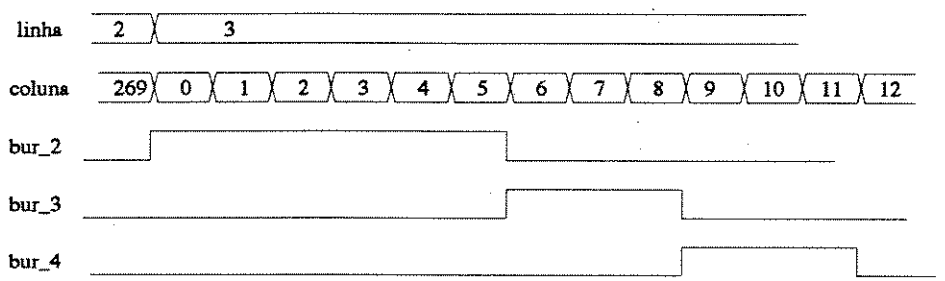


Figura 5.12: Diagrama temporal dos sinais *bur_2*, *bur_3* e *bur_4*

5.3.3 Gerador de Relógios de Escrita

As linhas de comandos abaixo modelam este bloco.

```
rel_esc_normal <= T1 AND not(bur_1) AND not(bur_2) AND not(bur_3);
rel_esc_justp <= T1 AND not(bur_1) AND not(bur_2) AND not(bur_3) AND not(bur_4);
rel_esc_justn <= T1 AND not(bur_1) AND not(bur_2);
```

Sua função é a de gerar relógios de escrita em memória elástica para serem utilizados em três situações distintas: quando não há justificação utiliza-se o relógio *rel_esc_normal*, quando há justificação negativa utiliza-se o relógio *rel_esc_justn* e quando há justificação positiva utiliza-se o relógio *rel_esc_justp*. Os diagramas temporais das figuras 5.13 e 5.14 ilustram estes sinais.

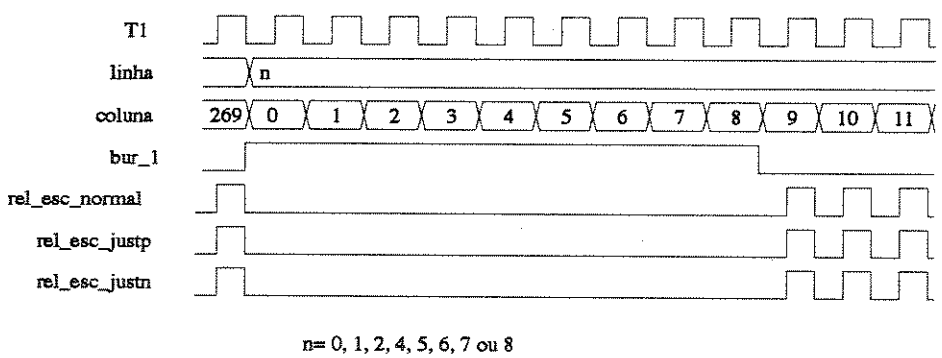


Figura 5.13: Relógios de escrita nas linhas 0, 1, 2, 4, 5, 6, 7 e 8.

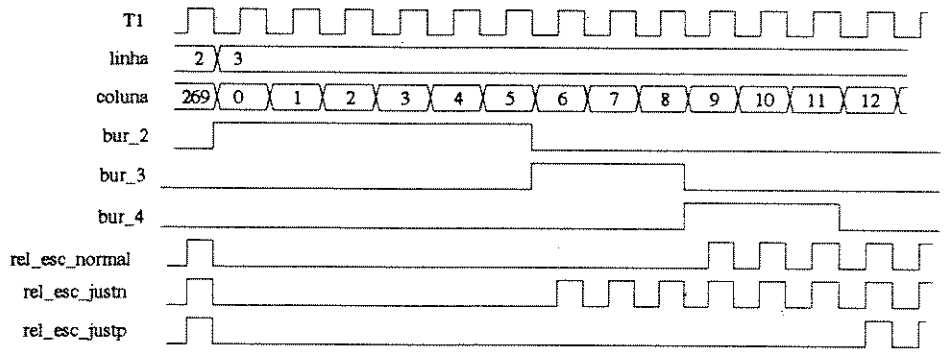


Figura 5.14: Relógios de escrita na linha 3

Vê-se pelas figuras que o relógio *rel_esc_normal* tem buracos nas nove primeiras colunas do quadro STM-1 em todas as suas linhas. Os relógios *rel_esc_justn* e *rel_esc_justp* diferem do *rel_esc_normal* somente na quarta linha (*linha = 3*) do quadro STM-1. Nesta linha o relógio *rel_esc_justn* tem buracos nas seis primeiras colunas e o relógio *rel_esc_justp* tem buracos nas doze primeiras colunas do quadro STM-1.

5.3.4 Gerador de Contadores

Este bloco é modelado pelo processo abaixo. O diagrama temporal da figura 5.15 ilustra o seu funcionamento.

```
contadores : PROCESS(rel_esc_normal, bur_4)
BEGIN
    IF (bur_4 = '1') AND (bur_4'EVENT) THEN
       cont_3 <= 0;
       cont_783 <= 0;
    END IF;
    IF (rel_esc_normal = '0') AND (rel_esc_normal'EVENT) THEN
       IF (cont_3 = 2) THEN
          cont_3 <= 0;
          cont_783 <= cont_783 + 1;
       ELSE
          cont_3 <= cont_3 + 1;
    END IF;
END PROCESS;
```

Vê-se que durante a borda de subida do sinal *bur_4*, os sinais *cont_3* e *cont_783* são zerados. A partir daí toda vez que ocorre uma borda de descida do relógio *rel_esc_normal* o sinal *cont_3* é incrementado ou zerado se o seu valor anterior for 2. O sinal *cont_783* é incrementado toda vez que o sinal *cont_3* é zerado.

A motivação para a criação dos sinais *cont_3* e *cont_783* é a necessidade de se saber onde se encontra o primeiro byte do quadro VC-4 transportado pelo sinal *STM1*. Será visto à frente que através da interpretação do ponteiro (H1H2) obtem-se o sinal *v_pont* (valor do ponteiro). Como foi visto no capítulo 2, o valor do ponteiro é um decimal na faixa de 0 a 782. Porém existem 2349 (3×783) bytes do quadro STM-1 para o transporte do quadro VC-4. Por isso o valor do ponteiro

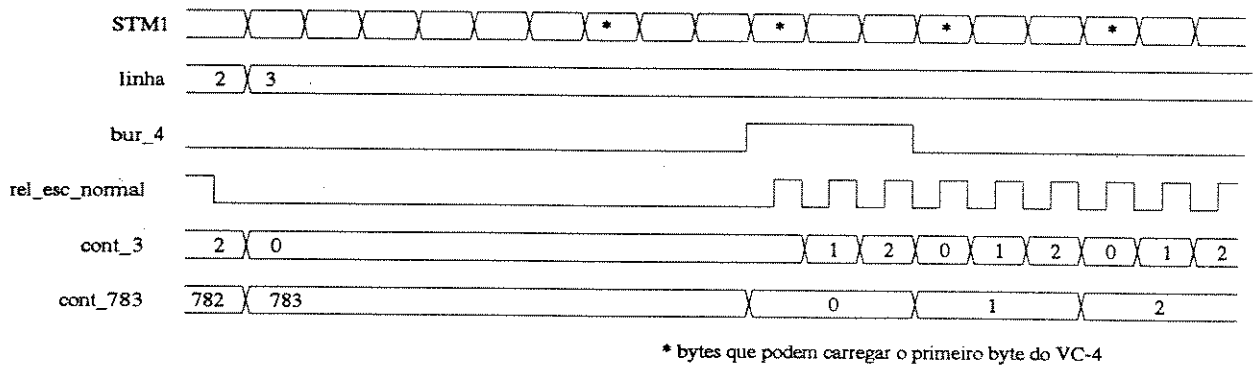


Figura 5.15: Diagrama temporal dos sinais *cont_3* e *cont_783*.

aponta para um grupo de 3 bytes do sinal STM-1 e implicitamente fica definido que o primeiro byte deste grupo transporta o primeiro byte do quadro VC-4. Assim o sinal *cont_783* é utilizado para identificar cada um dos 783 grupos de 3 bytes do quadro STM-1 que transportam o quadro VC-4 e o sinal *cont_3* é utilizado para identificar separadamente os 3 bytes de cada grupo.

5.3.5 Interpretador de Ponteiro

Este bloco é modelado pelo processo abaixo. O diagrama temporal da figura 5.16 ilustra o seu funcionamento.

```
int_ponteiro : PROCESS(bur_2)
variable palavra, palavra_ant, palavra_ant_ant : bit_vector(1 to 16);
variable cont_I, cont_D, cont_NDF : integer;
BEGIN
   IF (bur_2 = '0') AND (bur_2'EVENT) THEN
      palavra_ant_ant := palavra_ant;
      palavra_ant := palavra;
      palavra := byte_pra_palavra(H1, H2);
      cont_I := 0;
      cont_D := 0;
      for i in 7 to 16 loop
         IF (i mod 2 /= 0) THEN
            IF (palavra(i) /= palavra_ant(i) THEN
               cont_I := cont_I + 1;
            END IF;
            IF (palavra(i+1) /= palavra_ant(i+1) THEN
               cont_D := cont_D + 1;
            END IF;
         END IF;
      END loop;
      IF (inv_I = '0' AND inv_D = '0') THEN
         IF (cont_I >= 3) THEN
            inv_I <= '1';
            IF (v_pont = 782) THEN
               v_pont <= 0;
            ELSE
               v_pont <= v_pont + 1;
            END IF;
```

```
            END IF;
            IF (cont_D >= 3) THEN
               inv_D <= '1';
               IF (v_pont = 0) THEN
                  v_pont <= 782;
               ELSE
                  v_pont <= v_pont - 1;
               END IF;
            END IF;
         ELSE
            inv_I <= '0';
            inv_D <= '0';
         END IF;
         IF (palavra = palavra_ant) AND
               (palavra = palavra_ant_ant) THEN
            v_pont <= palavra_pra_inteiro(palavra);
         END IF;
         cont_NDF := 0;
         for i in 1 to 4 loop
            IF (palavra(i) = v_NDF(i)) THEN
               cont_NDF := cont_NDF + 1;
            END IF;
         END loop;
         IF (cont_NDF >= 3) THEN
            NDF <= '1';
            inv_I <= '0';
            inv_D <= '0';
            v_pont <= palavra_pra_inteiro(palavra);
         ELSE
            NDF <= '0';
         END IF;
      END IF;
   END PROCESS;
```

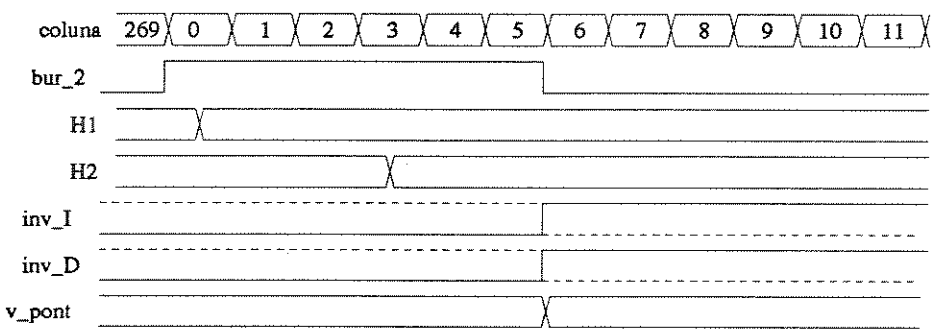


Figura 5.16: Diagrama temporal dos sinais *inv_I*, *inv_D* e *v_pont*.

Observe no processo acima o uso das funções *byte_pra_palavra* e *palavra_pra_inteiro*. Estas duas funções têm seus desenvolvimentos mostrados na "package" *pac* do apêndice A. A primeira é usada para transformar dois vetores de 8 bits (H1H2) em um vetor de 16 bits. A segunda é usada para transformar os 10 bits menos significativos de um vetor de 16 bits em um número inteiro (o valor do ponteiro).

Este bloco gera os sinais *inv_I*, *inv_D* e *v_pont* a partir dos sinais *H1* e *H2* respeitando as regras

de interpretação do ponteiro vistas no capítulo 2, verificando inclusive a indicação de novos dados (NDF).

O sinal *inv_I* quando em nível lógico 1 indica que o ponteiro traz bits I's invertidos e o sinal *inv_D* quando em nível lógico 1 indica que o ponteiro traz bits D's invertidos.

O sinal *v_pont* também é resultado da interpretação do ponteiro e indica em qual dos 783 grupos de 3 bytes identificados pelo sinal *cont_783* se encontra o primeiro byte do quadro VC-4. Será visto a seguir que este sinal é necessário na geração do sinal *inic_vc* que marca o início do quadro VC-4.

5.3.6 Gerador do Sinal *inic_vc*

Este bloco é modelado pelas linhas abaixo.

```
inic_vc <= '1' WHEN cont_3 = 0 AND cont_783 = v_pont ELSE
           '0';
```

Este bloco gera o sinal *inic_vc* que vai a nível lógico 1 durante o tempo em que o sinal *STM1* apresenta o byte J1 (primeiro byte do quadro VC-4). Isto é ilustrado no diagrama temporal da figura 5.17, o sinal *inic_vc* é 1 quando *cont_3 = 0* e *cont_783 = v_pont*.

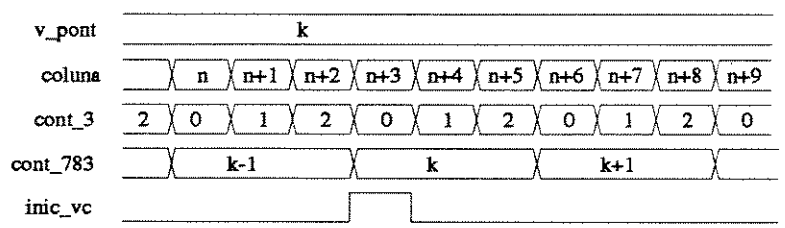


Figura 5.17: Diagrama temporal do sinal *inic_vc* (situação 1).

5.3.7 Selecionador de Relógios de Escrita

Este bloco é modelado pelas linhas abaixo.

```
rel_esc <= rel_esc_justn WHEN inv_D = '1' ELSE
           rel_esc_justp WHEN inv_I = '1' ELSE
           rel_esc_normal;
```

Este bloco simplesmente seleciona um dos 3 relógios de escrita (*re_esc_normal, rel_esc_justn* e *rel_esc_justp*) para ser o relógio *rel_esc* que é utilizado para se escrever os bytes do quadro VC-4 na memória elástica. Se o sinal *inv_I* estiver em nível lógico 1 indicando que o quadro STM-1 traz uma justificação positiva, então o relógio *rel_esc_justp* é o selecionado. Se o sinal *inv_D* estiver em nível lógico 1 indicando que o quadro STM-1 traz uma justificação negativa, então o relógio *rel_esc_justn* é o selecionado. Do contrário o relógio *rel_esc_normal* é o selecionado para ser o relógio de escrita *rel_esc*.

5.3.8 Contador de Endereço de Escrita

Este bloco é modelado pelo processo abaixo. O diagrama temporal na figura 5.18 ilustra o seu funcionamento.

```
cont_end_escrita : PROCESS(rel_esc)
BEGIN
    IF (rel_esc = '0') THEN
        IF (ender_esc = 47) THEN
            ender_esc <= 0;
        ELSE
            ender_esc <= ender_esc + 1;
        END IF;
    END IF;
END PROCESS;
```

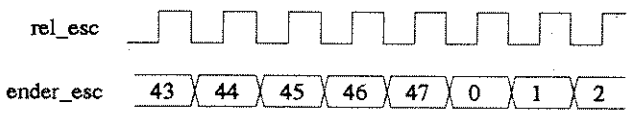


Figura 5.18: Diagram temporal do sinal *ender_esc*.

O sinal *ender_esc* fornece os enderços da memória elástica onde devem ser escritos, durante as bordas de subida do relógio *rel_esc*, os bytes do quadro VC-4.

5.3.9 Memória Elástica

Este bloco é modelado pelos 2 processos abaixo.

```
esc_mem_elastica : PROCESS(rel_esc)
BEGIN
    IF (rel_esc = '1') THEN
        mem_elast(ender_esc) <= STM1;
        nono_bit(ender_esc) <= inic_vc;
    END IF;
END PROCESS;
```

```
leit_mem_elastica : PROCESS(clk_vc4)
BEGIN
    IF (clk_vc4 = '1') THEN
        VC4 <= mem_elast(ender_leit);
        inic_vc4 <= nono_bit(ender_leit);
    END IF;
END PROCESS;
```

A base do funcionamento do bloco SA_R é a existência de uma memória elástica onde se escrevem os bytes do quadro VC-4 com o relógio esburacado de escrita *rel_esc* derivado do relógio *T1*. Estes bytes são lidos pelo relógio esburacado *clk_vc4* que é derivado do relógio *T0* (clock local do equipamento). Se a taxa de *T1* excede a taxa de *T0* a memória gradualmente se enche ou, se a taxa

de *T1* for menor que a taxa de *T0* então a memória gradualmente se esvazia. Existem um limiar superior e um limiar inferior na ocupação da memória que quando atingidos causam a ocorrência de justificações negativa e positiva respectivamente, no relógio *clk_vc4* (menos buracos e mais buracos em *clk_vc4* respectivamente). A figura 5.19 ilustra uma memória elástica.

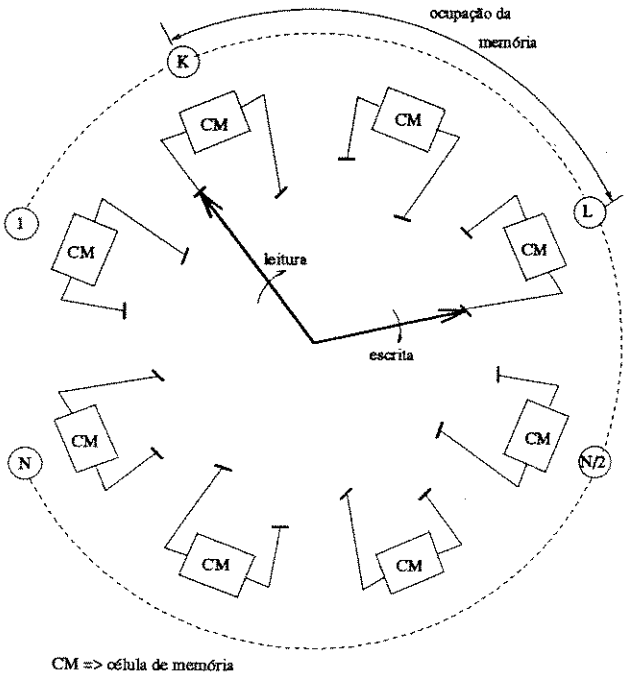


Figura 5.19: Representação de memória elástica.

Cada célula de armazenameto deve ser capaz de armazenar 1 byte do quadro VC-4. A cada borda de subida do relógio de leitura *clk_vc4* os bytes do VC-4 armazenados na memória são lidos para ocuparem posições adequadas no quadro STM-1 local. Afim de proceder o desmonte do quadro STM-1 local em estruturas menores (quadros TU-12's) ou gerar e inserir o ponteiro (bytes H1 e H2) para a sua retransmissão, é necessário saber em qual de seus bytes está sendo inserido o primeiro byte do VC-4. Isto é resolvido através do seguinte mecanismo. Cada célula de armazenamento da memória elástica é composta de 9 bits, sendo 8 para o armazenamento de um dos bytes do VC-4. O nono bit é preenchido com o valor lógico 1 quando a célula armazenar o primeiro byte do VC-4 e preenchido com o valor lógico 0 quando armazenar qualquer um dos demais 2348 bytes do VC-4. Assim, quando a célula de memória é lida, pode-se saber se ela armazena o primeiro byte do VC-4 analisando-se o seu nono bit.

A figura 5.20 simboliza uma memória elástica, onde:

- Rl: relógio de leitura.
- Re: relógio de escrita.
- Rm - Rl: limiar mínimo de ocupação.
- RM - Rl: limiar máximo de ocupação.

Podem ocorrer paradas nos relógios Rl e Re de até 12 bytes. A estes 12 bytes acrescenta-se 3 bytes de margem para jitter e se obtem:

- Rm - Rl = 12 + 3 = 15 bytes.

- Rl - RM = 12 + 3 = 15 bytes.

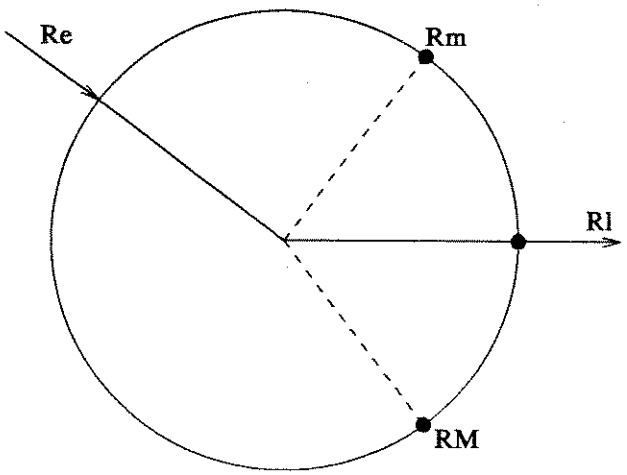


Figura 5.20: Simbolização de memória elástica com limiares de ocupação.

O CCITT pede uma margem mínima para variação da fase sem necessidade de ajuste de ponteiro (ΔR = RM - Rm) de 12 bytes. Coloca-se o valor ΔR em 18 bytes e assim obtém-se uma memória elástica de 48 bytes com limiar mínimo de ocupação de 15 bytes e limiar máximo de 33 bytes.

5.3.10 Gerador de Sinais Indicadores de Buracos em *T0*

As linhas de comandos abaixo modelam este bloco.

```
bur1_l <= '1' WHEN coluna_local >= 0 AND coluna_local <= 8 AND linha_local /= 3 ELSE
          '0';
bur2_l <= '1' WHEN coluna_local >= 0 AND coluna_local <= 5 AND linha_local = 3 ELSE
          '0';
bur3_l <= '1' WHEN coluna_local >= 6 AND coluna_local <= 8 AND linha_local = 3 ELSE
          '0';
bur4_l <= '1' WHEN coluna_local >= 9 AND coluna_local <= 11 AND linha_local= 3 ELSE
          '0';
```

A estrutura deste bloco é idêntica à estrutura do bloco Gerador de Sinais Indicadores de Buracos em *T1*. Os diagramas temporais das figuras 5.21 e 5.22 ilustram o seu funcionamento.

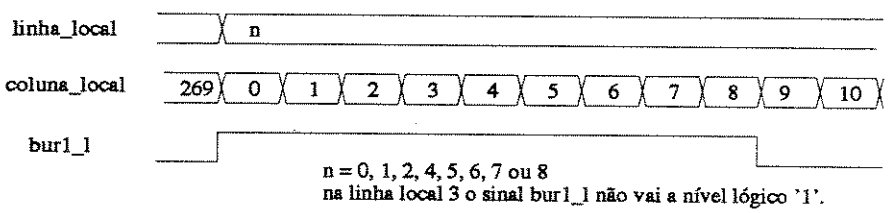


Figura 5.21: Diagrama temporal do sinal *bur1_l*.

5.3.11 Gerador de Relógios de Leitura

As linhas abaixo modelam este bloco.

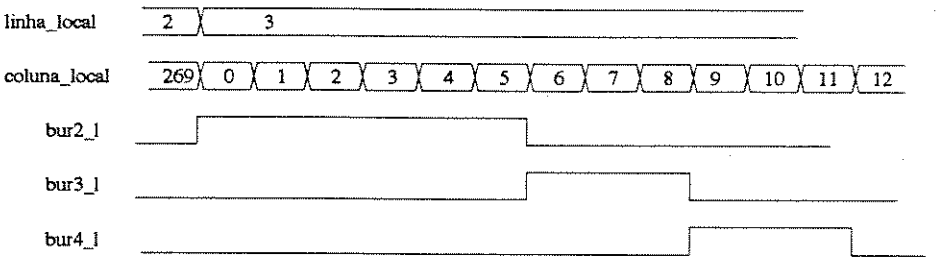


Figura 5.22: Diagram temporal dos sinais *bur2_l, bur3_l* e *bur4_l*.

---

rel_leit_normal <= T0 AND not(bur1_l) AND not(bur2_l) AND not(bur3_l);
rel_leit_justp <= T0 AND not(bur1_l) AND not(bur2_l) AND not(bur3_l) AND not(bur4_l);
rel_leit_justn <= T0 AND not(bur1_l) AND not(bur2_l);

A estrutura deste bloco é idêntica a estrutura do bloco Gerador de Relógios de Escrita. Sua função é a de gerar relógios de leitura de memória elástica para serem utilizados em três situações distintas: quando não há justificação utiliza-se o relógio *rel_leit_normal*, quando há justificação negativa utiliza-se o relógio *rel_leit_justn* e quando há justificação positiva utiliza-se o relógio *rel_leit_justp*. Os diagramas temporais das figuras 5.23 e 5.24 ilustram estes sinais.

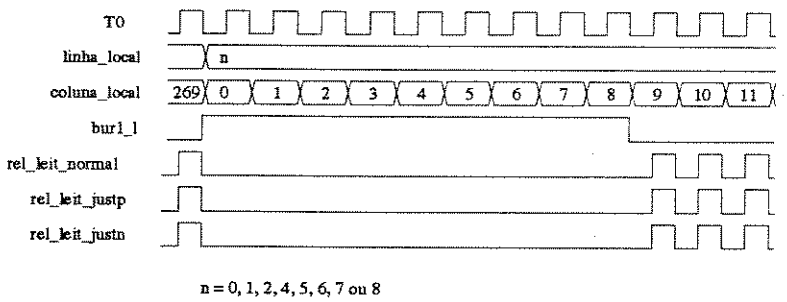


Figura 5.23: Relógios de leitura nas linhas 0, 1, 2, 4, 5, 6, 7 e 8.

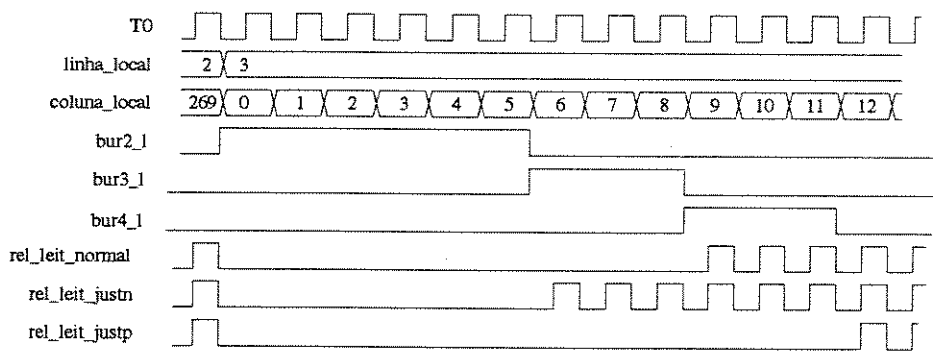


Figura 5.24: Relógios de leitura na linha 3.

Vê-se pelas figuras que o relógio *rel_leit_normal* tem buracos nas nove primeiras colunas do quadro STM-1 local em todas as suas linhas. Os relógios *rel_leit_justn* e *rel_leit_justp* diferem do *rel_leit_normal* somente na quarta linha (*linha_local = 3*) do quadro STM-1 local. Nesta linha o relógio *rel_leit_justn* tem buracos nas seis primeiras colunas e o relógio *rel_leit_justp* tem buracos nas doze primeiras colunas do quadro STM-1.

5.3.12 Selecionador de Relógios de Leitura

Este bloco é modelado pelas linhas abaixo.

```
clk_vc4 <= rel_leit_justn WHEN just_n = '1' ELSE
           rel_leit_justp WHEN just_p = '1' ELSE
           rel_leit_normal;
```

A estrutura deste bloco é idêntica à estrutura do bloco Selecionador de Relógios de Escrita. Ele simplesmente seleciona um dos 3 relógios de leitura (*re_leit_normal, rel_leit_justn* e *rel_leit_justp*) para ser o relógio *clk_vc4* que é utilizado para se ler os bytes do quadro VC-4 da memória elástica. Se o sinal *just_p* estiver em nível lógico 1 indicando que a memória elástica atingiu seu limiar inferior de ocupação, então o relógio *rel_leit_justp* é o selecionado. Se o sinal *just_n* estiver em nível lógico 1 indicando que a memória elástica atingiu seu limiar superior de ocupação, então o relógio *rel_leit_justn* é o selecionado. Do contrário o relógio *rel_leit_normal* é o selecionado para ser o relógio de leitura *clk_vc4*.

5.3.13 Justificador

Este bloco é modelado pelo processo abaixo. Ele gera os sinais *just_p* e *just_n* que indicam quando o quadro STM-1 local apresenta justificações positivas e negativas respectivamente.

```
justificador : PROCESS(linha_local)
BEGIN
    IF (linha_local = 2) AND (linha_local'EVENT) THEN
       IF (just_dep_3 = 2) THEN
          IF (fase <= 15) THEN
             just_p <= '1';
             just_dep_3 <= 0;
          END IF;
          IF (fase >= 33) THEN
             just_n <= '1';
             just_dep_3 <= 0;
          END IF;
       ELSE
          just_dep_3 <= just_dep_3 + 1;
          just_p <= '0';
          just_n <= '0';
       END IF;
    END IF;
END PROCESS;
```

No instante em que o sinal *linha_local* atinge o valor 2 a decisão de se justificar é tomada.

O sinal *just_dep_3* é interno a este bloco. Ele existe para permitir que as justificações só ocorram de 3 em 3 quadros de 125 μs, de acordo com a regra de geração de ponteiro vista no capítulo 2.

Se já tiverem decorridos 3 quadros desde a última justificação então a fase entre o contador de

endereço de leitura *end_leit* e contador de endereço de escrita *end_esc* é verificada. Se a fase for menor ou igual a 15 bytes então o sinal de controle *just_p* toma o estado lógico '1', sinalizando para a necessidade de uma justificação positiva. Se a fase for maior ou iqual a 33 bytes então o sinal de controle *just_n* toma o valor '1', sinalizando para a necessidade de uma justificação negativa.

5.3.14 Gerador do Sinal *cont3_l*

Este bloco é modelado pelo processo abaixo. O diagrama temporal da figura 5.25 ilustra o seu funcionamento.

```
cont3_local : PROCESS(T0)
BEGIN
   IF (T0 = '1') AND (T0'EVENT) THEN
      IF (coluna_local = 0) THEN
         cont3_l <= 0;
      ELSE
         IF (cont3_l = 2) THEN
            cont3_l <= 0;
         ELSE
            cont3_l <= cont3_l + 1;
         END IF;
      END IF;
   END IF;
END PROCESS;
```

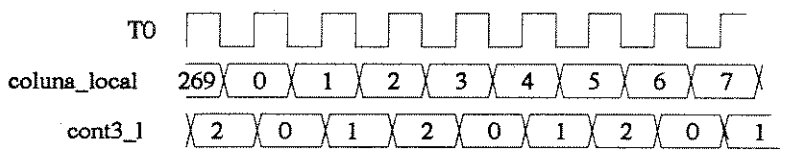


Figura 5.25: Diagrama temporal do sinal *cont3_l*.

Será visto a seguir que o sinal *cont3_l* é necessário na geração do sinal *para_leit*.

5.3.15 Gerador do Sinal *para_leit*

Este bloco é modelado pelas linhas abaixo. Na figura 5.26 tem-se diagramas temporais apresentando 3 situações diferentes.

```
para_leit <= '1' WHEN inic_vc4 = '1' AND cont3_l = 1 ELSE
             '1' WHEN inic_vc4 = '1' AND cont3_l = 2 ELSE
             '0';
```

Na situação 1 o sinal *inic_vc4* vai a nível lógico 1 quando o sinal *cont3_l = 1*. Isto significa que o primeiro byte do VC-4 estaria sendo inserido no segundo byte de um dos 783 grupos de 3 bytes do quadro STM-1 local. Esta é uma situação inadequada e por isso o sinal *para_leit* vai a nível lógico 1 e permanece neste valor até atingir a situação 3. Observe que a situação 1 leva à situação 2 que

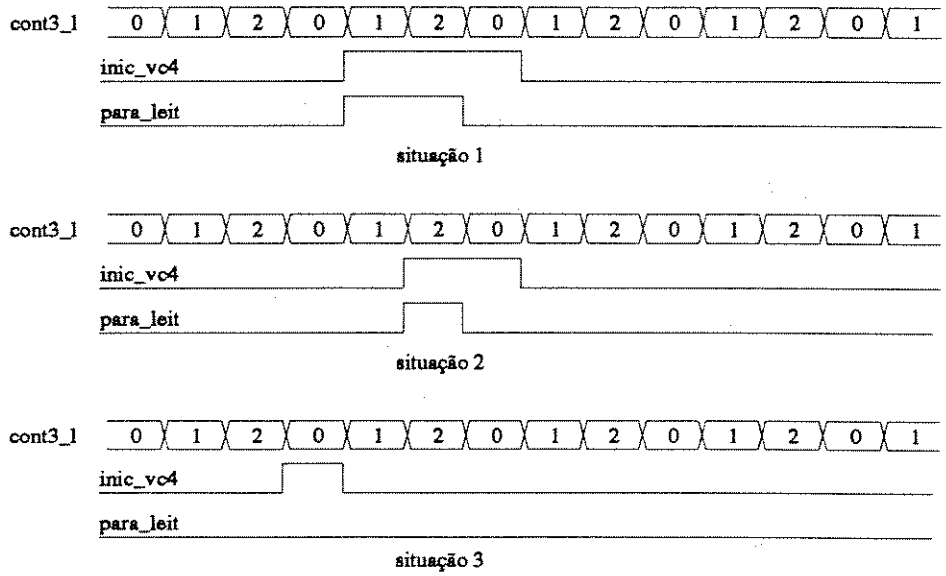


Figura 5.26: Diagramas temporais do sinal *para_leit* para 3 situações relacionado os sinais *cont3_l* e *inic_vc_l*.

por sua vez leva à situação 3. Somente a situação 3 pode ser aceita, isto é, o primeiro byte do VC-4 deve ser inserido no primeiro byte de um dos 783 grupos de 3 bytes que carregam o quadro VC-4 no quadro STM-1 local.

5.3.16 Contador de Endereços de Leitura

Este bloco é modelado pelo processo abaixo.

```
cont_end_leitura : PROCESS(rel_leit)
BEGIN
    IF (clk_vc4 = '0') AND (clk_vc4'EVENT) THEN
       IF (para_leit = '0') THEN
          IF (ender_leit = 47) THEN
             ender_leit <= 0;
          ELSE
             ender_leit <= ender_leit + 1;
          END IF;
       END IF;
    END IF;
END PROCESS;
```

Este bloco é semelhante ao bloco Contador de Endereços de Escrita. A diferença entre estes dois blocos é que o contador de endereços de leitura tem seu funcionamento paralisado caso o sinal *para_leit* esteja em nível lógico 1. Isto está ilustrado na figura 5.27.

Verifica-se na figura 5.27 que quando o sinal *para_leit* vai a nível lógico 1 o endereço de leitura na memória elástica não é incrementado. Como consequência a leitura da memória elástica é paralisada. Isto acontece para que o primeiro byte do quadro VC-4 passe a ser inserido em uma posição adequada do quadro STM-1 local.

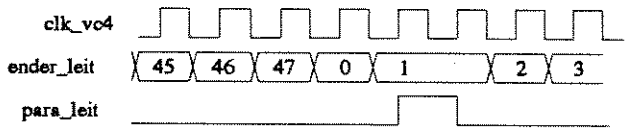


Figura 5.27: Diagrama temporal do sinal *ender_leit*.

5.3.17 Fasimetro

Este bloco é modelado pelo processo abaixo. Sua função é simplesmente a de medir a distância entre o endereço de leitura do endereço de escrita em memória elástica e reportar o valor através do sinal *fase*.

```
fasimetro : PROCESS(ender_esc, ender_leit)
variable v_fase : integer := 0;
BEGIN
    v_fase := ender_esc - ender_leit;
    IF (v_fase < 0) THEN
       fase <= 48 + v_fase;
    ELSE
       fase <= v_fase;
    END IF;
END PROCESS;
```

5.4 Gerador de Sinais de Controle do Quadro VC-4

O bloco Gerador de Sinais de Controle do Quadro VC-4 está detalhado na figura 5.28.

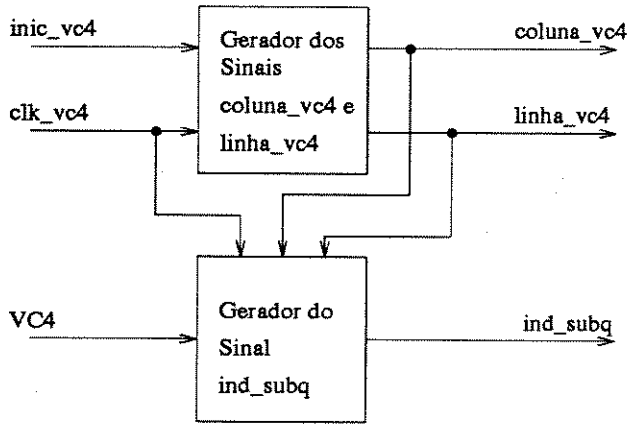


Figura 5.28: Gerador de Sinais de Controle do Quadro VC-4.

5.4.1 Gerador dos Sinais *coluna_vc4* e *linha_vc4*

O processo abaixo modela este bloco.

```
linha_coluna_vc4: PROCESS(clk_vc4)
BEGIN
    IF (clk_vc4 = '1') AND (clk_vc4'EVENT) THEN
       IF (coluna_vc4 = 260) THEN
          IF (linha_vc4 = 8) THEN
             linha_vc4 <= 0;
             coluna_vc4 <= 0;
          ELSE
             linha_vc4 <= linha_vc4 + 1;
             coluna_vc4 <= 0;
          END IF;
       ELSE
          coluna_vc4 <= coluna_vc4 + 1;
       END IF;
    END IF;
    IF (clk_vc4 = '0') AND (clk_vc4'EVENT) THEN
      IF (inic_vc4 = '1') THEN
          coluna_vc4 <= 0;
          linha_vc4 <= 0;
       END IF;
    END IF;
END PROCESS;
```

O diagrama temporal da figura 5.29 ilustra o funcionamento deste bloco. Observe que os sinais *linha_vc4* e *coluna_vc4* são zerados quando o sinal *inic_vc4* é igual a '1' e ocorre a borda de descida do relógio *clk_vc4*. O sinal *coluna_vc4* é incrementado a cada borda de subida do sinal *clk_vc4* e zerado quando atinge o valor 260. O sinal *linha_vc4* é incrementado cada vez que o sinal *coluna_vc4* é zerado.

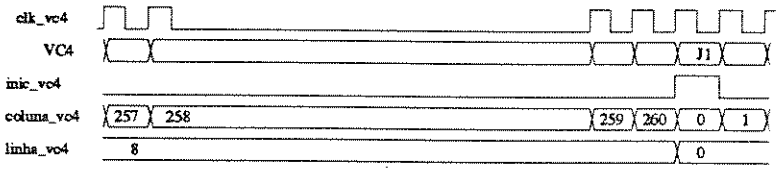


Figura 5.29: Diagrama temporal dos sinais *linha_vc4* e *conuna_vc4*.

A figura 5.30 mostra o diagrama temporal dos sinais *linha_vc4* e *coluna_vc4*. Apesar da figura 5.30 apresentar o relógio *clk_vc4* com clocks iguais, existem locais onde este relógio fica regularmente parado por 9 pulsos e ocasionalmente ocorrem paradas de 6 e 12 pulsos quando há justificação negativa e positiva respectivamente.

Se o relógio *clk_vc4* não fosse desdentado então poderia-se ilustrar os sinais *coluna_vc4* e *linha_vc4* de uma forma mais didática como na figura 5.30.

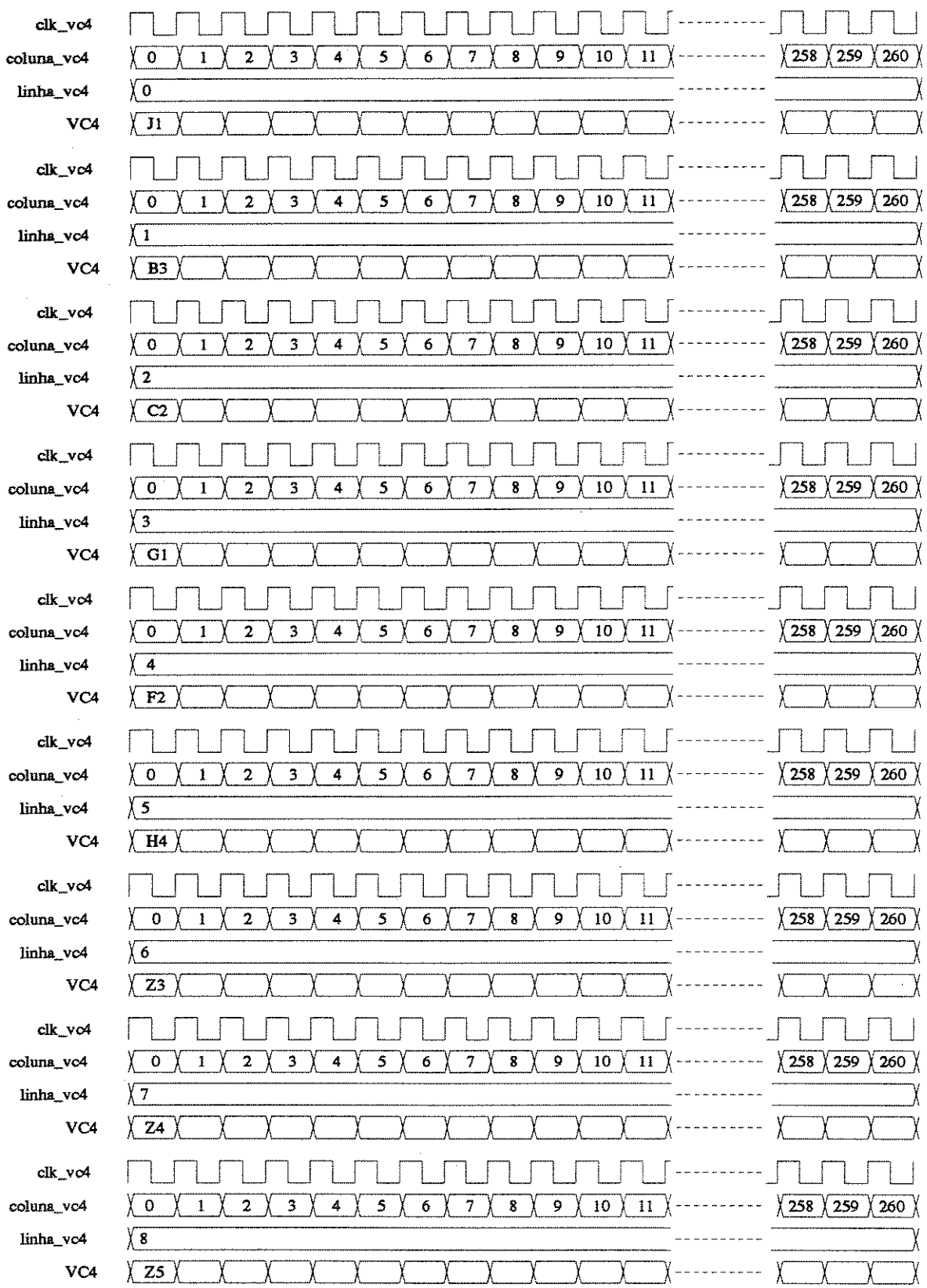


Figura 5.30: Diagrama temporal dos sinais *linha_vc4* e *coluna_vc4* durante um quadro VC-4 completo.

5.4.2 Gerador do Sinal Indicador de Subquadro (*ind_subq*)

Este bloco é modelado pelo processo abaixo.

```
indicador_de_subquadro: PROCESS(clk_vc4)
VARIABLE H4 : BIT_VECTOR(1 TO 8);
BEGIN
    IF (clk_vc4 = '0') THEN
       IF (coluna_vc4 = 0) AND (linha_vc4 = 5) THEN
          H4 := VC4;
         IF H4(7) = '0' AND H4(8) = '0' THEN
            ind_subq <= 0;
         ELSIF H4(7) = '0' AND H4(8) = '1' THEN
            ind_subq <= 1;
         ELSIF H4(7) = '1' AND H4(8) = '0' THEN
            ind_subq <= 2;
         ELSE
            ind_subq <= 3;
         END IF;
       END IF;
   END IF;
END PROCESS;
```

A figura 5.31 mostra o momento que o byte H4 é lido do sinal *VC4* para a geração do sinal *ind_subq* que indica o subquadro de 125 μs do quadro de 500 μs presente no VC-4. Este sinal é gerado a partir dos 2 bits menos significativos do byte H4.

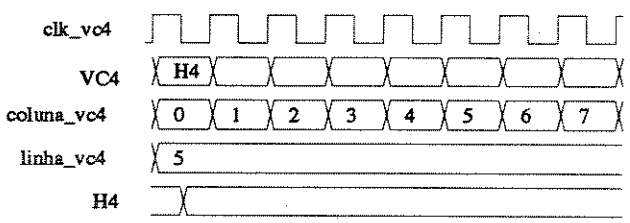


Figura 5.31: Geração do sinal *ind_subq*.

5.5 Adaptação de Rotas de Alta Ordem na Recepção (HPA_R

O bloco HPA_R está detalhado na figura 5.32.

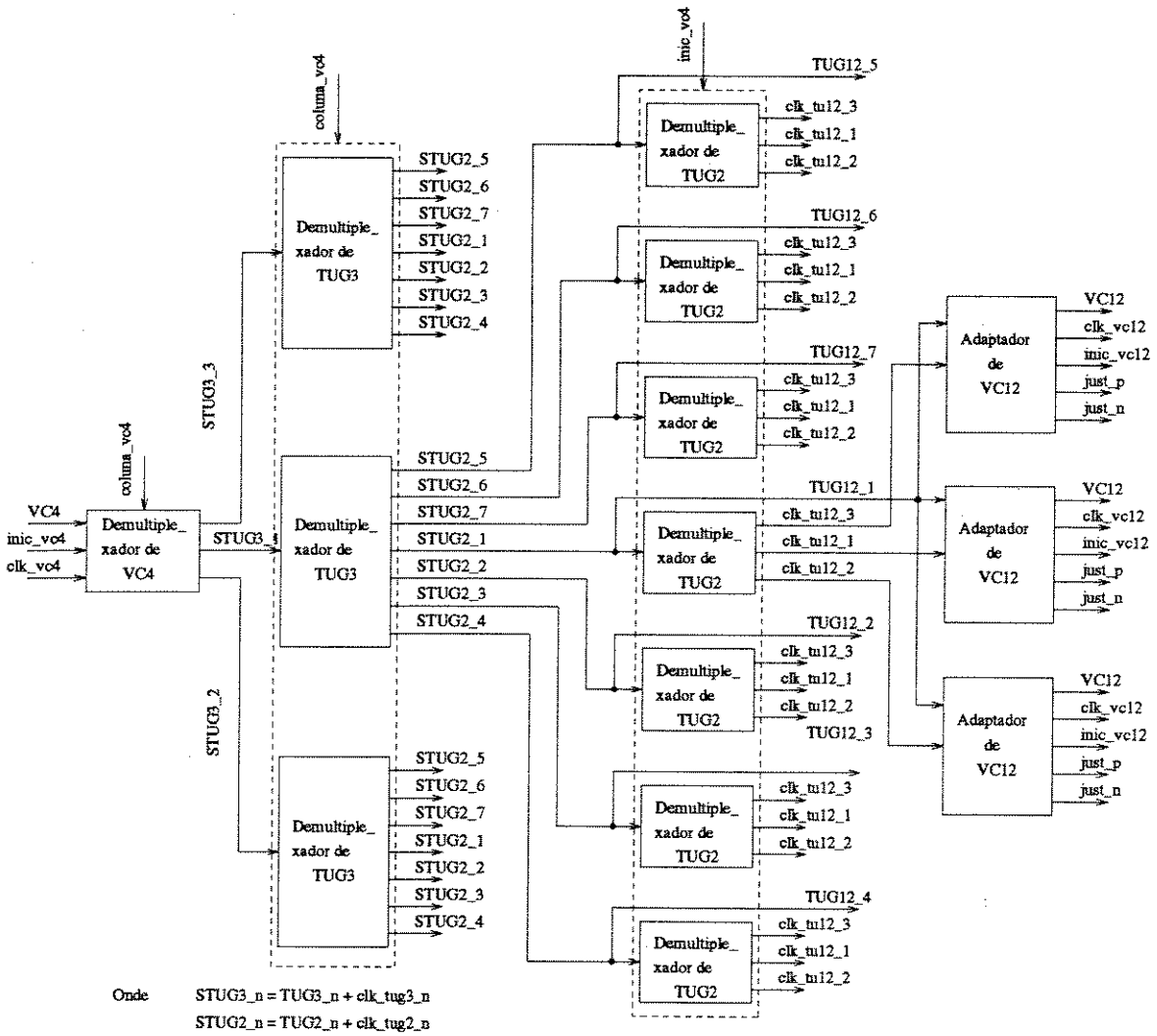


Figura 5.32: Adaptação de Rotas de Alta Ordem na Recepção (HPA_R).

A figura 5.32 representa um bloco Demultiplexador de VC-4, 3 blocos Demultiplexadores de TUG-3, 21 blocos Demultiplexadores de TUG-2 e 63 blocos Adaptadores de VC-12.

A seguir detalha-se os quatro tipos de blocos da figura 5.32.

5.5.1 Demultiplexador de VC-4

As figuras 5.33 e 5.34 ilustram como 3 TUG-3 são multiplexados em um quadro VC-4.

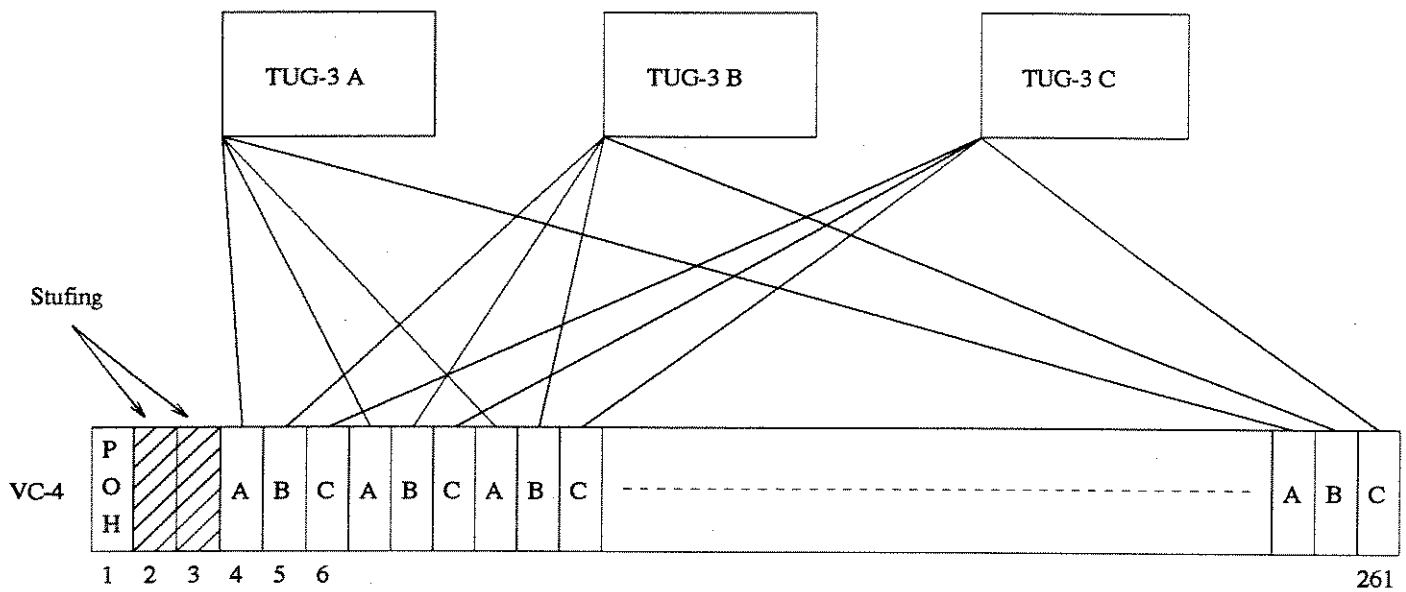


Figura 5.33: Multiplexagem de 3 TUG-3 no quadro VC-4.

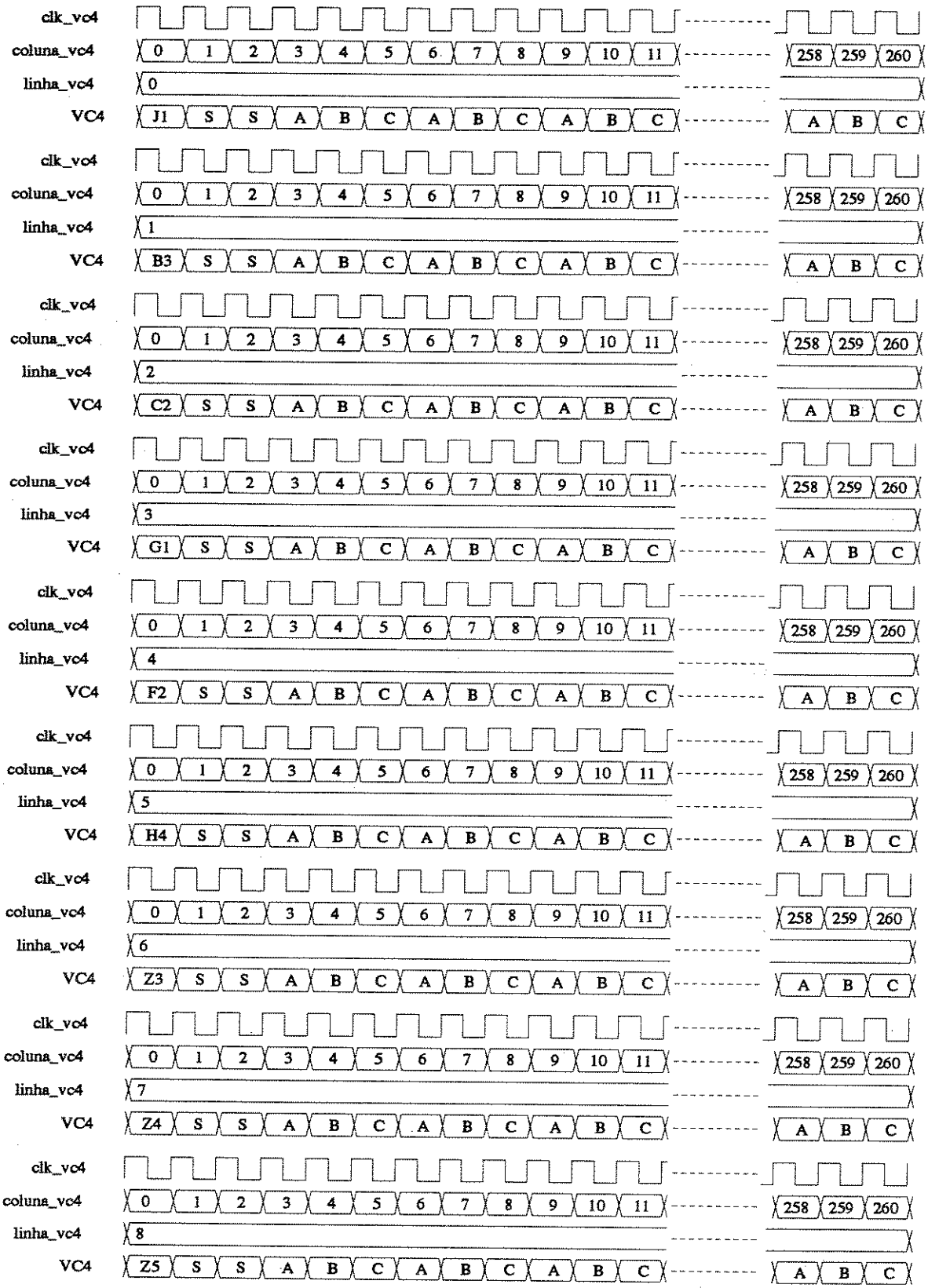


Figura 5.34: Diagrama temporal mostrando a multiplexagem de 3 TUG-3 no quadro VC-4.

Evidentemente a figura 5.34 não mostra os buracos existentes no relógio *clk_vc4* mas é uma forma mais ilustrativa de se representar a relação entre os seus sinais.

O Demultiplexador de VC-4 é detalhado através da figura 5.35.

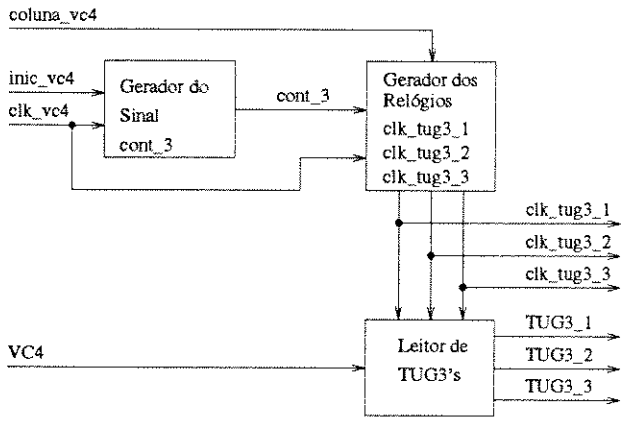


Figura 5.35: Demultiplexador de VC-4.

Gerador do Sinal *cont_3*

O processo abaixo modela este bloco. Observe que na borda de subida do sinal *inic_vc4* o sinal *cont_3* assume o valor zero. Observe também que a cada borda de subida do relógio *clk_vc4* o sinal *cont_3* é incrementado ou zerado se o seu valor for 2.

```
ger_cont_3: PROCESS(clk_vc4)
BEGIN
    IF (clk_vc4 = '1') AND (clk_vc4'EVENT) THEN
       IF (cont_3 = 2) THEN
          cont_3 <= 0;
       ELSE
          cont_3 <= cont_3 + 1;
       END IF;
    END IF;
    IF (clk_vc4 = '0') AND (clk_vc4'EVENT) THEN
      IF (inic_vc4 = '1') THEN
          cont_3 <= 0;
       END IF;
    END IF;
END PROCESS;
```

O diagrama temporal da figura 5.36 ilustra o funcionamento deste bloco.

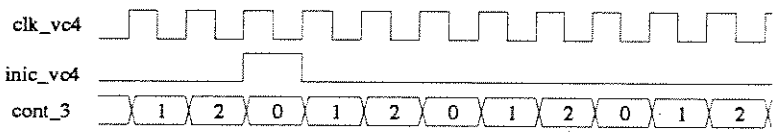


Figura 5.36: Diagrama temporal do sinal *cont_3*.

Gerador dos Relógios *clk_tug3_1, clk_tug3_2* e *clk_tug3_3*

As linhas abaixo modelam este bloco.

```
clk_tug3_1 <= '1' WHEN (coluna_vc4 > 2) AND (cont_3 = 0) AND (clk_vc4 = '0') ELSE
    '0';
clk_tug3_2 <= '1' WHEN (coluna_vc4 > 2) AND (cont_3 = 1) AND (clk_vc4 = '0') ELSE
    '0';
clk_tug3_3 <= '1' WHEN (coluna_vc4 > 2) AND (cont_3 = 2) AND (clk_vc4 = '0') ELSE
    '0';
```

O diagrama temporal da figura 5.37 ilustra o funcionamento deste bloco.

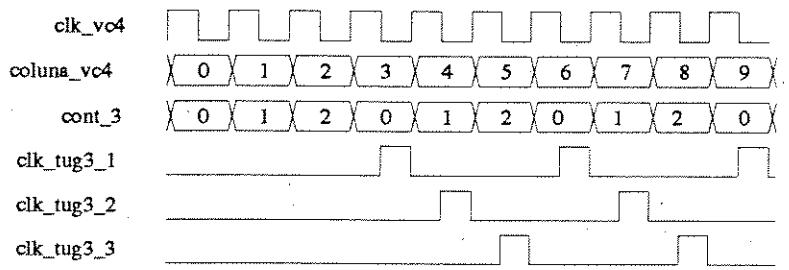


Figura 5.37: Diagrama temporal dos relógios *clk_tug3_1, clk_tug3_2* e *clk_tug3_3* .

Observe que não há pulsos nos relógios *clk_tug3_1, clk_tug3_2* e *clk_tug3_3* nas 3 primeiras colunas do quadro VC-4 pois a primeira coluna transporta POH e as duas seguintes são enchimento (stufing).

Leitor de TUG3's

O processo abaixo modela este bloco.

```
leitor_de_tug3: PROCESS(clk_tug3_1, clk_tug3_2, clk_tug3_3)
BEGIN
    IF (clk_tug3_1 = '1') AND (clk_tug3_1'EVENT) THEN
        TUG3_1 <= VC4;
    END IF;
    IF (clk_tug3_2 = '1') AND (clk_tug3_2'EVENT) THEN
        TUG3_2 <= VC4;
    END IF;
    IF (clk_tug3_3 = '1') AND (clk_tug3_3'EVENT) THEN
        TUG3_3 <= VC4;
    END IF;
END PROCESS;
```

O diagrama temporal da figura 5.38 ilustra o funcionamento deste bloco.

Observe que os bytes do primeiro TUG-3 são lidos do sinal *VC4* durante a borda de subida do relógio *clk_tug3_1*. O mesmo acontece com os outros 2 TUG3's em relação aos seus respectivos relógios.

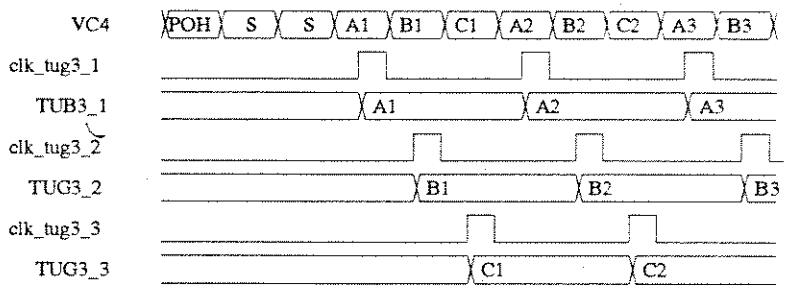


Figura 5.38: Diagrama temporal dos sinais *TUG3_1*, *TUG3_2* e *TUG3_3*.

5.5.2 Demultiplexador de TUG-3

A figura 5.39 ilustra como 7 TUG-2 são multiplexados em um quadro TUG3.

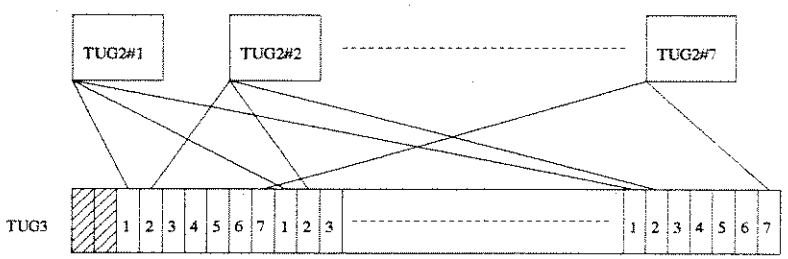


Figura 5.39: Multiplexagem de 7 TUG-2 no quadro TUG-3.

O Demultiplexador de TUG-3 é detalhado através da figura 5.40. Aqui *TUG3* e *clk_tug3* podem ser qualquer um dos 3 *TUG3s* que saem do Demultiplexador de VC-4 acompanhado de seu relógio.

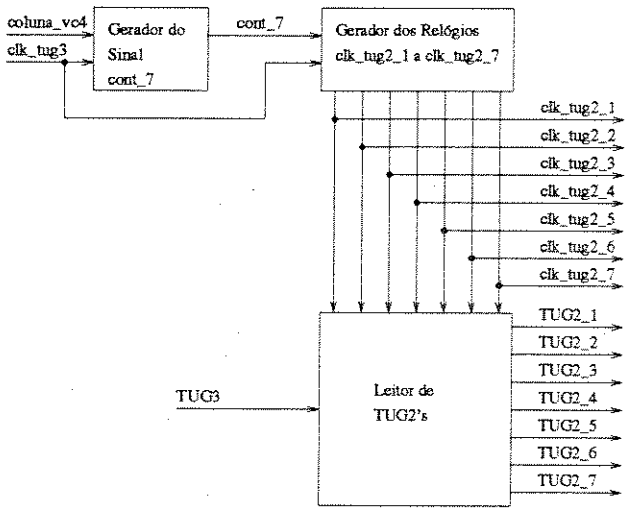


Figura 5.40: Demultiplexador de TUG-3.

Gerador do Sinal *cont_7*

O processo abaixo modela este bloco.

```
ger_cont_7: PROCESS(clk_tug3)
BEGIN
    IF (clk_tug3 = '1') AND (clk_tug3'EVENT) THEN
       IF (coluna_vc4 > 2) AND (coluna_vc4 < 9) THEN
          cont_7 <= 7;
       ELSE
          IF (cont_7 = 6) OR (cont_7 = 7) THEN
             cont_7 <= 0;
          ELSE
             cont_7 <= cont_7 + 1;
          END IF;
       END IF;
    END IF;
END PROCESS;
```

O diagrama temporal da figura 5.41 mostra os pulsos dos relógios dos 3 TUG-3's que acompanham os 2 bytes de enchimento (stufing) existentes em cada linha de cada quadro TUG-3. Observe que isto acontece quando o sinal *coluna_vc4* é menor que 9. Esta observação é importante para a construção do bloco Gerador do Sinal *cont_7*.

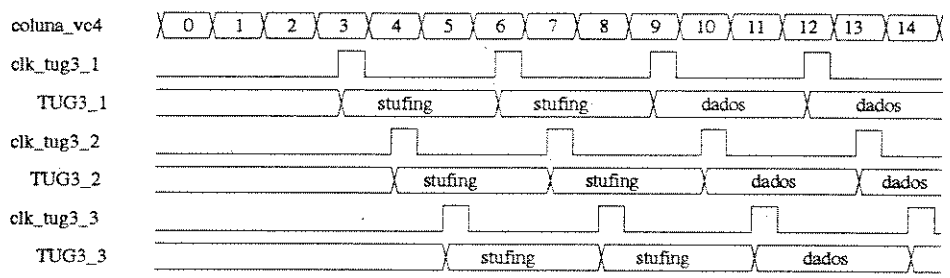


Figura 5.41: Pulsos dos relógios dos 3 TUG-3's que acompanham bytes de enchimento (stufing).

A figura 5.42 mostra o diagrama temporal do sinal *cont_7* gerado no Demultiplexador de TUG-3 que processa o TUG3_1. Observe que o sinal *cont_7* assume o valor 7 nos bytes da primeira e segunda colunas do quadro TUG3_1. O mesmo acontece para os outros 2 TUG-3's. Isto é uma forma de sinalizar a existência de bytes de "stufing" no barramento *TUG3*.

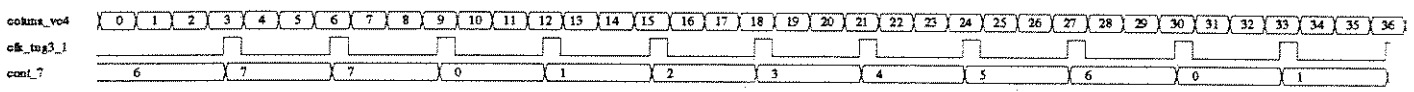


Figura 5.42: Diagrama temporal do sinal *cont_7*.

Gerador dos Relógios *clk_tug2_1* **a** *clk_tug2_7*

As linhas abaixo modelam este bloco.

```
clk_tug2_1 <= '1' WHEN (cont_7 = 0) AND (clk_tug3 = '0') ELSE
    '0';
clk_tug2_2 <= '1' WHEN (cont_7 = 1) AND (clk_tug3 = '0') ELSE
    '0';
clk_tug2_3 <= '1' WHEN (cont_7 = 2) AND (clk_tug3 = '0') ELSE
    '0';
clk_tug2_4 <= '1' WHEN (cont_7 = 3) AND (clk_tug3 = '0') ELSE
    '0';
clk_tug2_5 <= '1' WHEN (cont_7 = 4) AND (clk_tug3 = '0') ELSE
    '0';
clk_tug2_6 <= '1' WHEN (cont_7 = 5) AND (clk_tug3 = '0') ELSE
    '0';
clk_tug2_7 <= '1' WHEN (cont_7 = 6) AND (clk_tug3 = '0') ELSE
    '0';
```

O diagrama temporal da figura 5.43 ilustra o funcionamento deste bloco. Observe que quando o sinal *cont_7* assume o valor 7 os relógios *clk_tug2's* não apresentam pulsos. Isto se faz para que os bytes de "stufing" não sejam lidos.

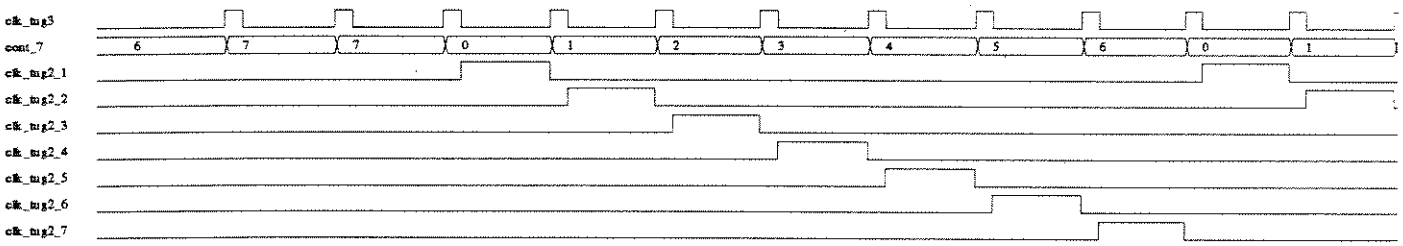


Figura 5.43: Diagrama temporal dos relógios *clk_tug2_1* a *clk_tug2_7*.

Leitor de TUG2's

O processo abaixo modela este bloco.

```
leitor_de_tug2: PROCESS(clk_tug2_1, clk_tug2_2, clk_tug2_3, clk_tug2_4, clk_tug2_5, clk_tug2_6, clk_tug2_7)
BEGIN
    IF (clk_tug2_1 = '1') AND (clk_tug2_1'EVENT) THEN
        TUG2_1 <= TUG3;
    END IF;
    IF (clk_tug2_2 = '1') AND (clk_tug2_2'EVENT) THEN
        TUG2_2 <= TUG3;
    END IF;
    IF (clk_tug2_3 = '1') AND (clk_tug2_3'EVENT) THEN
        TUG2_3 <= TUG3;
    END IF;
    IF (clk_tug2_4 = '1') AND (clk_tug2_4'EVENT) THEN
        TUG2_4 <= TUG3;
    END IF;
    IF (clk_tug2_5 = '1') AND (clk_tug2_5'EVENT) THEN
        TUG2_5 <= TUG3;
    END IF;
    IF (clk_tug2_6 = '1') AND (clk_tug2_6'EVENT) THEN
        TUG2_6 <= TUG3;
    END IF;
    IF (clk_tug2_7 = '1') AND (clk_tug2_7'EVENT) THEN
        TUG2_7 <= TUG3;
    END IF;
END PROCESS;
```

O diagrama temporal da figura 5.44 ilustra o funcionamento deste bloco.

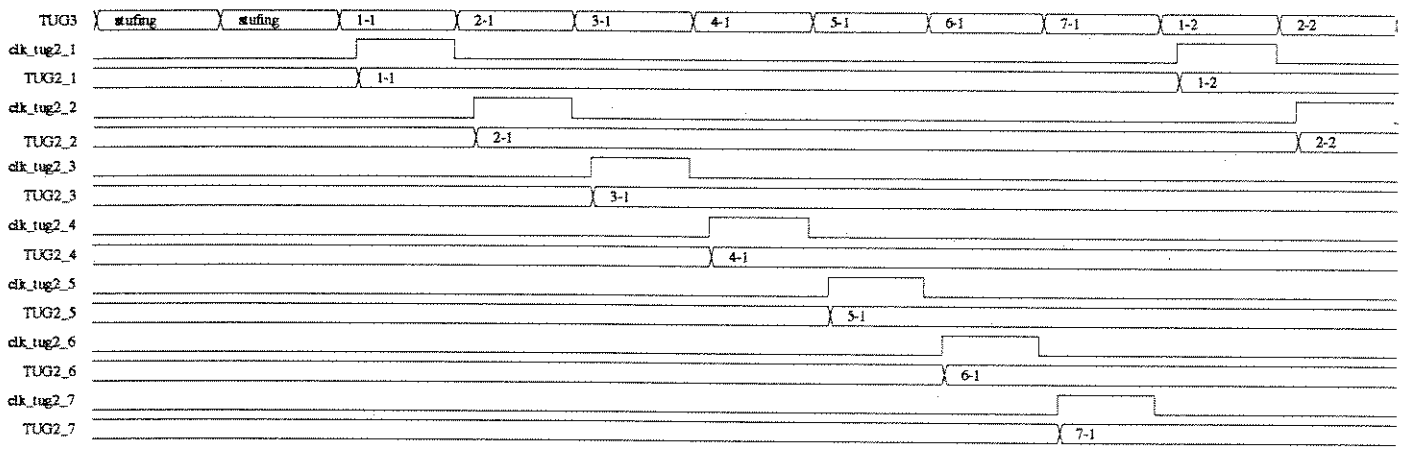


Figura 5.44: Diagrama temporal dosdos sinais *TUG2_1* a *TUG2_7*.

Observe que os bytes do primeiro TUG-2 são lidos do sinal *TUG3* durante as bordas de subida do relógio *clk_tug2_1*. O mesmo acontece com os outros 6 TUG2's em relação aos seus respectivos relógios.

5.5.3 Demultiplexador de TUG-2

A figura 5.45 ilustra como 3 TU-12 são multiplexados em um quadro TUG-2.

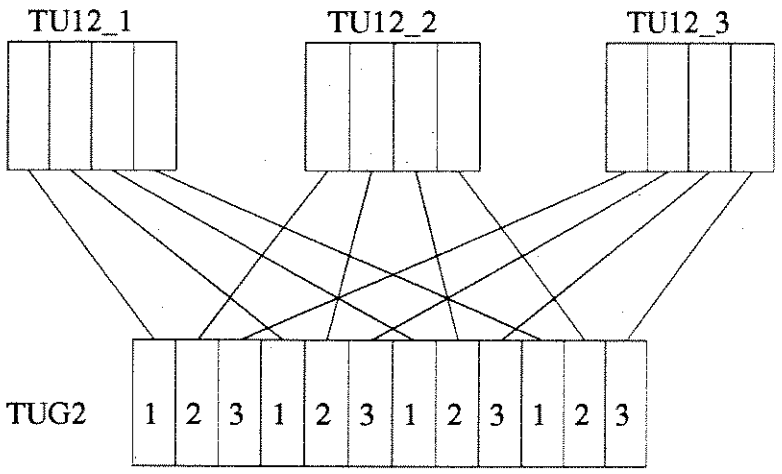


Figura 5.45: Multiplexagem de 3 TU-12 no quadro TUG-2.

O Demultiplexador de TUG-12 é detalhado através da figura 5.46.

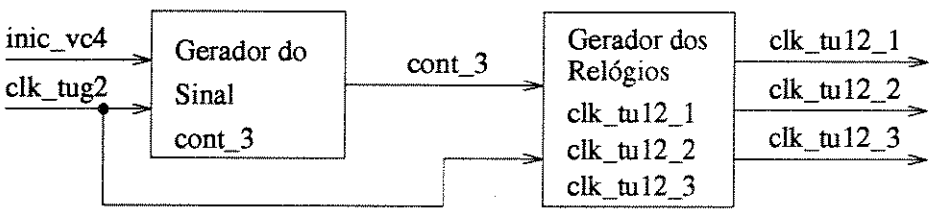


Figura 5.46: Demultiplexador de TU-12.

Comparando a figura 5.46 com as figuras 5.40 e 5.35 verifica-se que este bloco não é propriamente um demultiplexador pois ele não tem um bloco leitor de TU-12's. A tarefa de ler os TU-12's dos sinais *TUG2* é deixada para os blocos adaptadores de VC-12.

Este esquema é válido para qualquer um dos 7 TUG2.

Gerador do Sinal *cont_3*

O processo abaixo modela este bloco.

```
ger_cont_3: PROCESS(clk_tug2, inic_vc4)
BEGIN
    IF (inic_vc4 = '0') AND (inic_vc4'EVENT) THEN
       cont_3 <= 2;
    END IF;
    IF (clk_tug2 = '1') AND (clk_tug2'EVENT) THEN
       IF (cont_3 = 2) THEN
          cont_3 <= 0;
       ELSE
          cont_3 <= cont_3 + 1;
       END IF;
    END IF;
END PROCESS;
```

A figura 5.47 mostra o diagrama temporal do sinal *cont_3*.

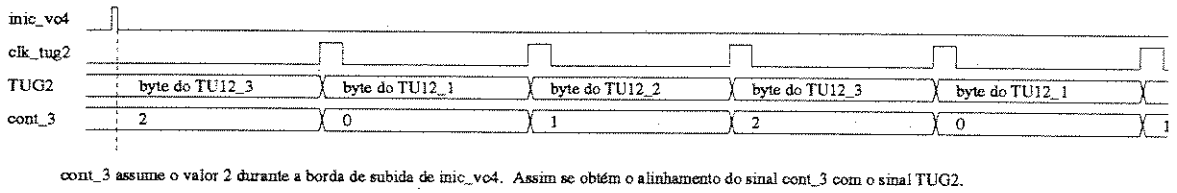


Figura 5.47: Diagrama temporal do sinal *cont_3*.

Observe que o sinal *cont_3*, durante as bordas de descida do sinal *inic_vc4*, assume o valor 2. Isto se faz para alinhar o sinal *cont_3* de tal forma que ele assuma o valor zero durante os tempos em que o sinal *TUG2* apresenta bytes do primeiro TU-12.

Gerador dos Relógios *clk_tu12_1*, *clk_tu12_2* e *clk_tu12_3*

As linhas abaixo modelam este bloco.

```
clk_tu12_1 <= '1' WHEN (cont_3 = 0) AND (clk_tug2 = '0') ELSE
    '0';
clk_tu12_2 <= '1' WHEN (cont_3 = 1) AND (clk_tug2 = '0') ELSE
    '0';
clk_tu12_3 <= '1' WHEN (cont_3 = 2) AND (clk_tug2 = '0') ELSE
    '0';
```

O diagrama temporal da figura 5.48 ilustra o funcionamento deste bloco.

O diagrama temporal da figura 5.49 mostra como os relógios *clk_tu12_1*, *clk_tu12_2* e *clk_tu12_3* podem ser usados para a leitura dos bytes dos 3 TU-12's presentes no sinal *TUG2*. O mesmo sinal *TUG2* é enviado para 3 blocos adaptadores de VC-12 e cada um dos 3 recebe um dos relógios *clk_tu12*. Desta forma cada adaptador de VC-12 pode retirar do sinal *TUG2* os bytes de um dos 3 TU-12's multiplexados no sinal *TUG2*.

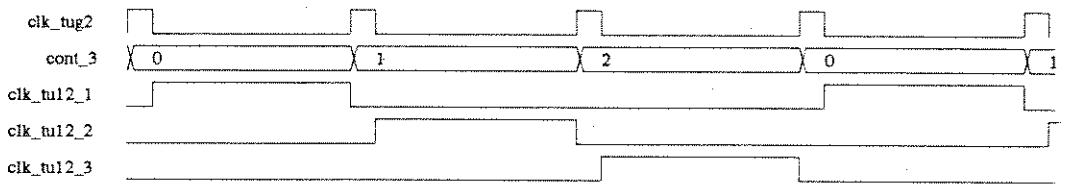


Figura 5.48: Diagrama temporal dos relógios *clk_tu12_1*, *clk_tu12_2* e *clk_tu12_3*.

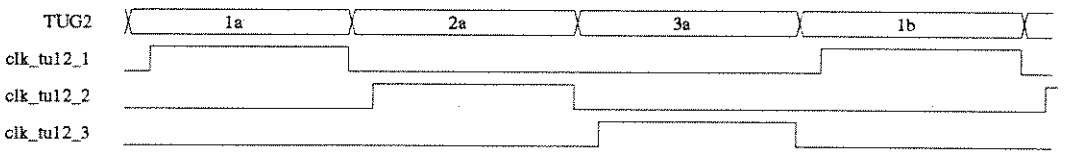


Figura 5.49: Diagrama temporal dos relógios *clk_tu12's* em relação ao sinal *TUG2*.

5.5.4 Adaptador de VC-12 na Recepção

Este bloco esta detalhado na figura 5.50. A seguir analisa-se cada um dos blocos da figura 5.50 através de diagramas temporais de seus sinais de entrada e de saída e dos comandos VHDL que os modelam.

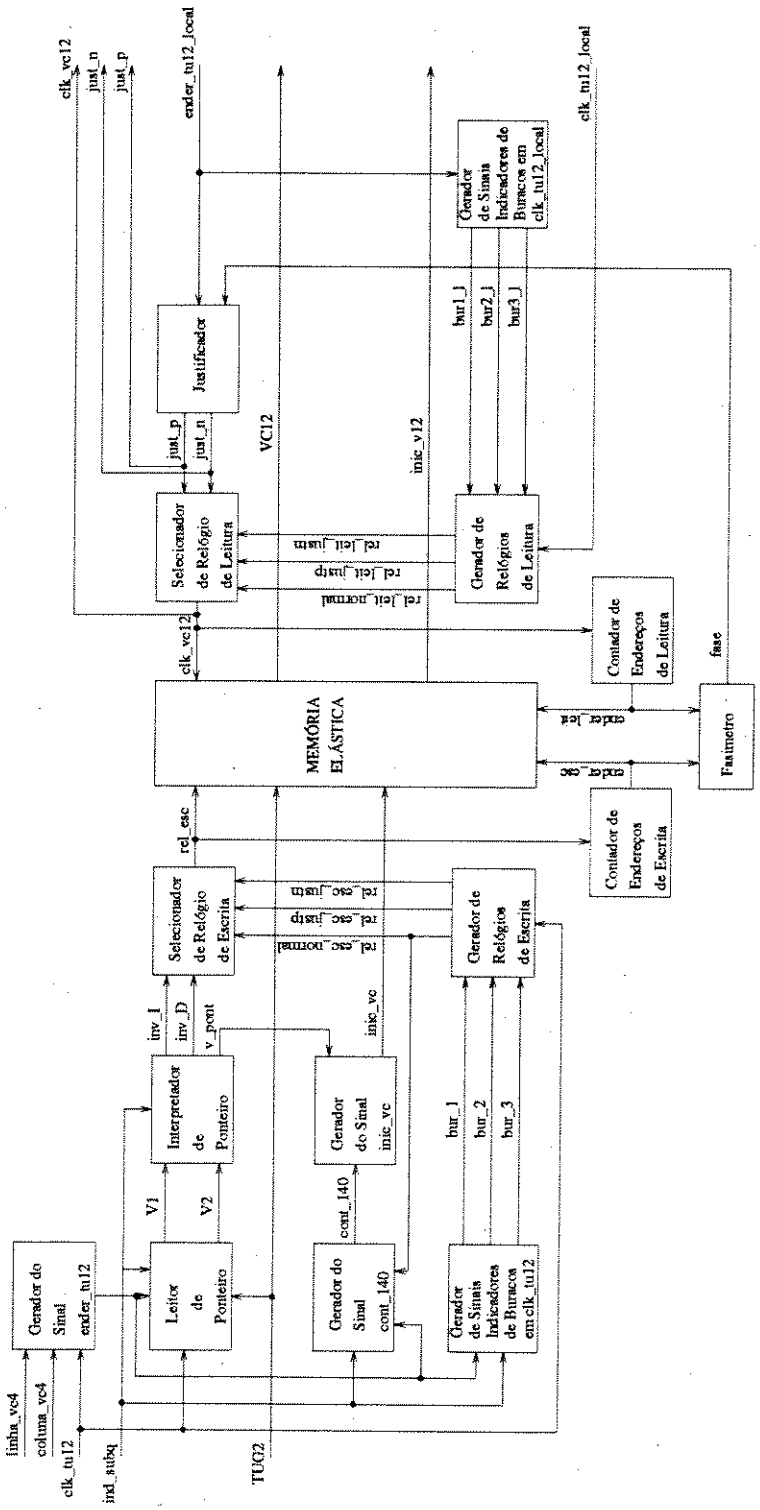


Figura 5.50: Detalhamento do bloco adaptador de VC-12.

Nota: Além dos sinais *TUG2*, *inic_vc4*, *ind_subq* e do relógio *clk_tu12*, este bloco tem como sinais de entrada os sinais *ender_tu12_local* e o relógio *clk_tu12_local* que estão ilustrados na figura 5.51.

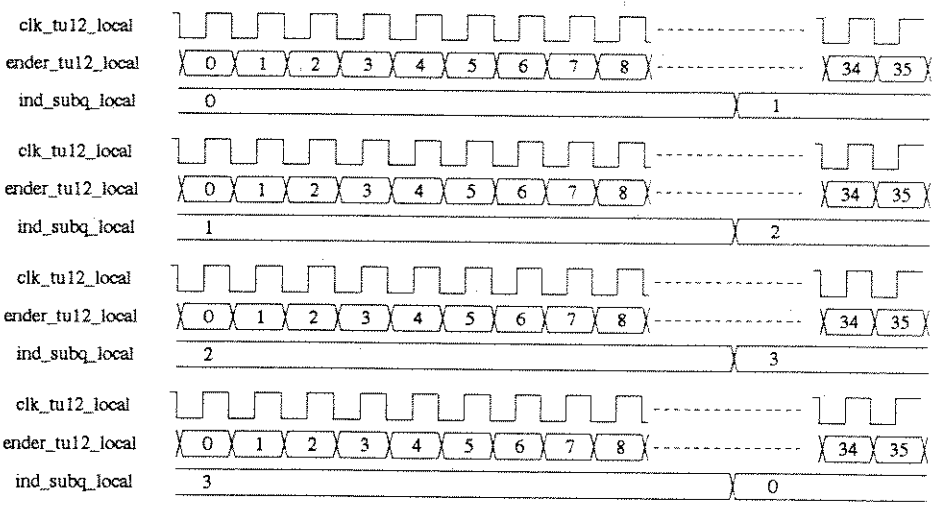


Figura 5.51: Diagrama temporal dos sinais *ender_tu12_local* e *ind_subq_local* em relação ao relógio *clk_tu12_local*.

Gerador do Sinal *ender_tu12*

Este bloco é modelado pelo processo abaixo.

```
ger_ender_tu12: PROCESS(coluna_vc4, clk_tu12)
    BEGIN
    IF (coluna_vc4 = 9) AND (coluna_vc4'EVENT) AND (linha_vc4 = 0) THEN
      IF (clk_tu12 = '1') THEN
         ender_tu12 <= 35;
      ELSE
         ender_tu12 <= 0;
      END IF;
   END IF;
   IF (clk_tu12 = '0') AND (clk_tu12'EVENT) THEN
      IF (ender_tu12 = 35) THEN
         ender_tu12 <= 0;
      ELSE
         ender_tu12 <= ender_tu12 + 1;
      END IF;
   END IF;
END PROCESS;
```

Os diagramas temporais das figuras 5.52 e 5.53 ilustram o funcionamento deste bloco.

A cada borda de descida do relógio *clk_tu12* o sinal *ender_tu12* é incrementado ou zerado se o seu valor for 35. Mas é preciso que ele seja alinhado afim de indicar qual dos 36 bytes do subquadro TU12 está presente no barramento *TUG2* durante a borda de subida do relógio *clk_tu12*.

Um ponto adequado para se alinhar o sinal *ender_tu12* é o ponto em que o sinal *coluna_vc4* atinge o valor 9 e o sinal *linha_vc4* se encontra com o valor 0. Neste instante, alguns dos 63 relógios *clk_tu12*

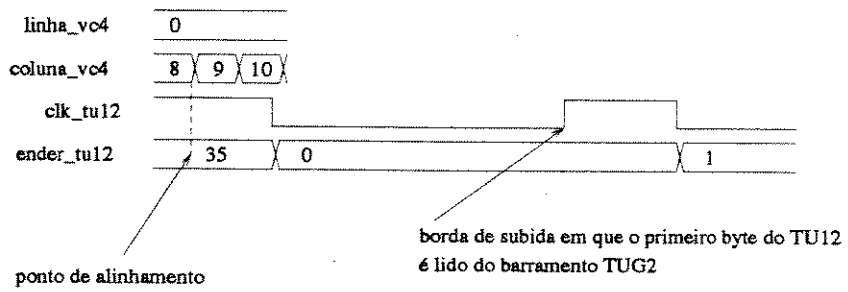


Figura 5.52: Diagrama temporal do sinal *ender_tu12* alinhado com valor 35.

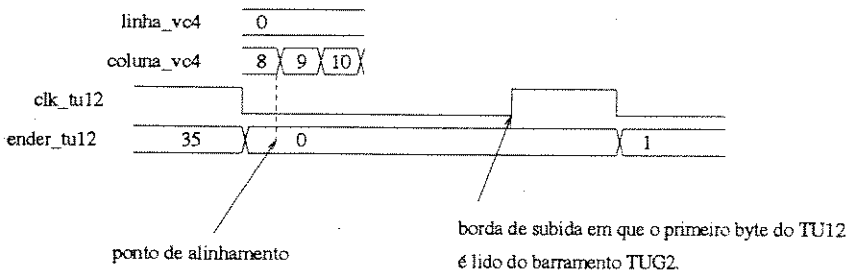


Figura 5.53: Diagrama temporal do sinal *ender_tu12* alinhado com valor 0.

têm nível lógico '1' e por isso os sinais *ender_tu12* devem ser alinhados com o valor 35. Desta forma, a primeira borda de descida leva corretamente o sinal *ender_tu12* para o valor zero, como mostrado na figura 5.52. Outros relógios *clk_tu12* já se encontram no nível lógico zero no instante do alinhamento, por isso os seus correspondentes sinais *ender_tu12* são alinhados com o valor zero, como mostrado na figura 5.53.

O sinal *ender_tu12* e o sinal *ind_subq* identificam separadamente cada um dos bytes do quadro TU12 presentes no barramento *TUG2* durante as bordas de subida do relógio *clk_tu12*.

Leitor de Ponteiro

O processo abaixo modela este bloco.

```
leitura_ponteiro: PROCESS(clk_tu12)
BEGIN
    IF (clk_tu12 = '1') THEN
       IF (ender_tu12 = 0) THEN
          CASE ind_subq IS
             WHEN 0 => V1 <= TUG2;
             WHEN 1 => V2 <= TUG2;
             WHEN OTHERS => NULL;
          END CASE;
       END IF;
    END IF;
END PROCESS;
```

A função deste bloco é ler do sinal *TUG2* os bytes do ponteiro (V1 e V2) do TU-12. Isto está ilustrado nos diagramas temporais das figura 5.54 e 5.55.

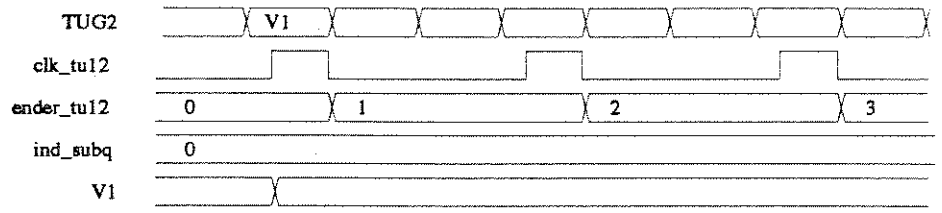


Figura 5.54: Diagrama temporal ilustrando a leitura do byte V1 do ponteiro do TU-12.

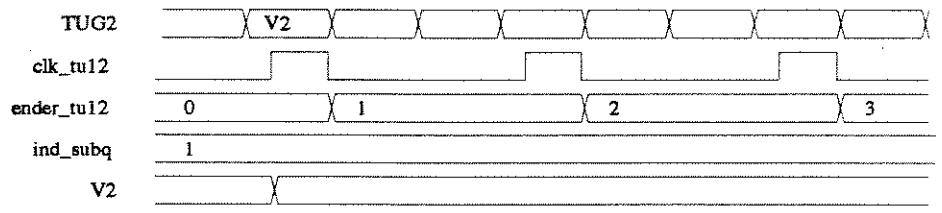


Figura 5.55: Diagrama temporal ilustrando a leitura do byte V2 do ponteiro do TU-12.

Durante a borda de subida de *clk_tu12* o sinal *V1* é o resultado da leitura do sinal *TUG2* quando (*ender_tu12 = 0, ind_subq = 0*). Durante a borda de subida de *clk_tu12* o sinal *V2* é o resultado da leitura do sinal *TUG2* quando (*ender_tu12 = 0, ind_subq = 1*).

Observe que o sinal *TUG2* carrega 3 quadros TU-12 mas o relógio *clk_tu12* só apresenta pulsos durante os bytes de um único TU-12.

Gerador de Sinais Indicadores de Buracos em *clk_tu12*

As linhas de comandos abaixo modelam este bloco.

```
bur_1 <= '1' WHEN ender_tu12 = 0 AND ind_subq /= 2 ELSE
    '0';
bur_2 <= '1' WHEN ender_tu12 = 0 AND ind_subq = 2 ELSE
    '0';
bur_3 <= '1' WHEN ender_tu12 = 1 AND ind_subq = 2 ELSE
    '0';
```

Os diagramas temporais das figuras 5.56 e 5.57 ilustram os sinais gerados por este bloco. O sinal *bur_1* vai a nível lógico '1' quando o sinal *ender_tu12* for igual a 0 e o sinal *ind_subq* for igual a 0, 1 ou 3. O sinal *bur_2* vai a nível lógico '1' quando o sinal *ender_tu12* for igual a 0 e o sinal *ind_subq* for igual a 2. O sinal *bur_3* vai a nível lógico '1' quando o sinal *ender_tu12* for igual a 1 e o sinal *ind_subq* for igual a 2.

Como se vê a seguir, estes sinais são necessários na geração do relógio de escrita em memória elástica. Este relógio escreve em uma memória elástica somente os bytes do quadro VC-12 transportado pelo TU-12.

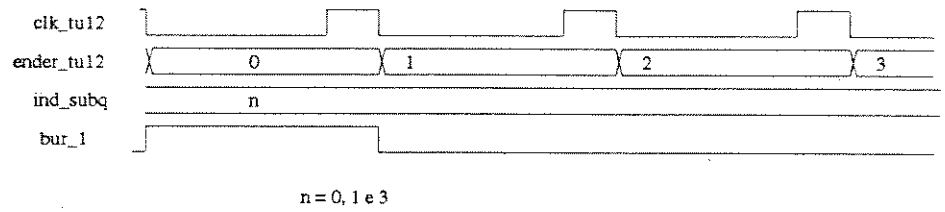


Figura 5.56: Diagrama temporal do sinal *bur_1*

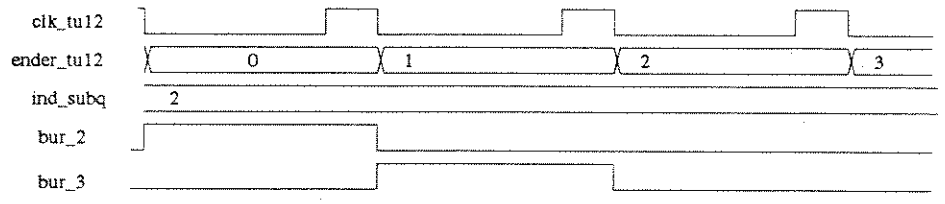


Figura 5.57: Diagrama temporal dos sinais *bur_2* e *bur_3*

Gerador de Relógios de Escrita

As linhas de comandos abaixo modelam este bloco.

```
rel_esc_justn <= clk_tu12 AND NOT(bur_1);
rel_esc_justp <= clk_tu12 AND NOT(bur_1) AND NOT(bur_2) AND NOT(bur_3);
rel_esc_normal <= clk_tu12 AND NOT(bur_1) AND NOT(bur_2);
```

Sua função é a de gerar relógios de escrita em memória elástica para serem utilizados em três situações distintas: quando não há justificação utiliza-se o relógio *rel_esc_normal*, quando há justificação negativa utiliza-se o relógio *rel_esc_justn* e quando há justificação positiva utiliza-se o relógio *rel_esc_justp*. Os diagramas temporais das figuras 5.58 e 5.59 ilustram estes sinais.

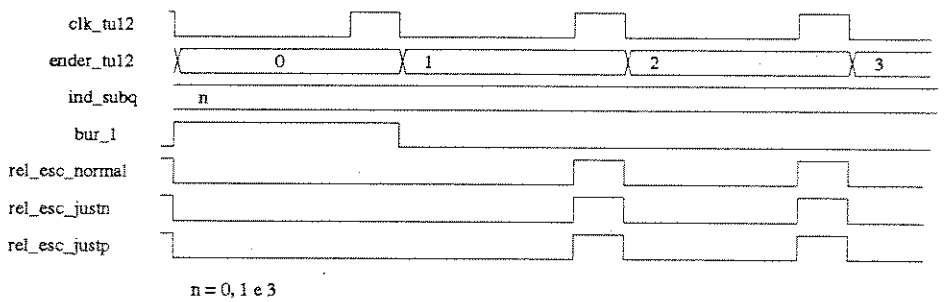


Figura 5.58: Relógios de escrita nos subquadros 1, 2, e 4.

Vê-se pelas figuras que o relógio *rel_esc_normal* tem um buraco durante o primeiro byte do quadro TU-12 de qualquer subquadro. Os relógios *rel_esc_justn* e *rel_esc_justp* diferem do *rel_esc_normal* somente no terceiro subquadro (*ind_subq* = *2*). Neste subquadro o relógio *rel_esc_justn* não tem nenhum buraco e o relógio *rel_esc_justp* tem buracos nos dois primeiros bytes do quadro TU-12.

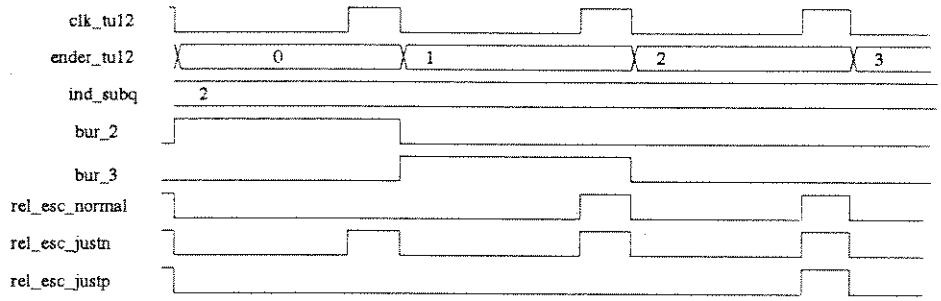


Figura 5.59: Relógios de escrita no subquadro 3

Gerador do Sinal *cont_140*

Este bloco é modelado pelo processo abaixo. O diagrama temporal da figura 5.60 ilustra o seu funcionamento.

```
contador: PROCESS(rel_esc_normal, ender_tu12)
BEGIN
    IF (ender_tu12 = 1) AND (ender_tu12'EVENT) THEN
        IF (ind_subq = 1) THEN
            cont_140 <= 0;
        END IF;
    END IF;
    IF (rel_esc_normal = '0') AND (rel_esc_normal'EVENT) THEN
            cont_140 <= cont_140 + 1;
    END IF;
END PROCESS;
```

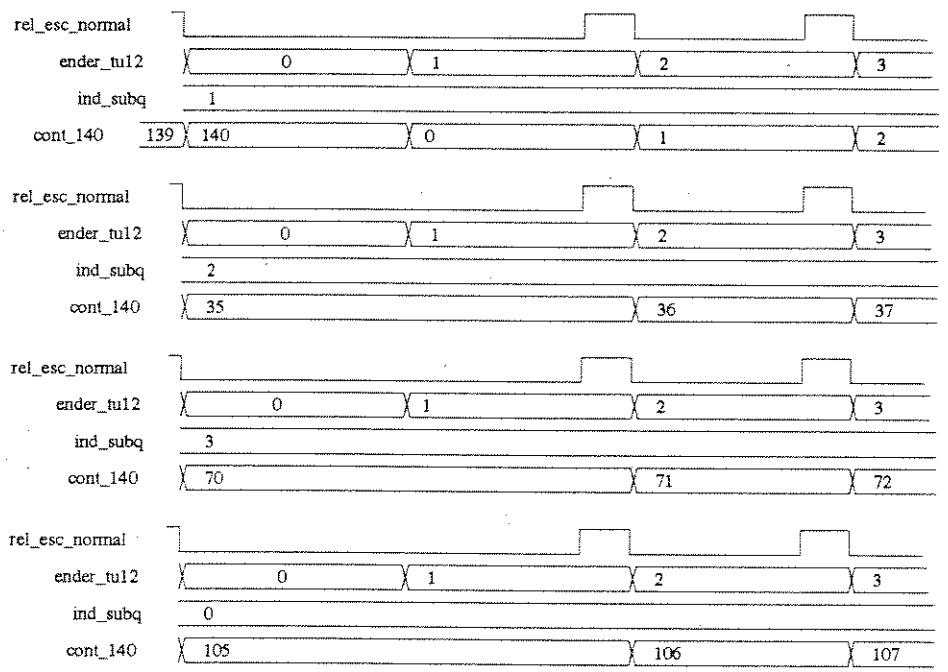


Figura 5.60: Diagrama temporal do sinal *cont_140*.

Vê-se que quando o sinal *ind_subq* é igual a 1 e o sinal *ender_tu12* assume o valor um, o sinal

cont_140 é zerado. A partir daí toda vez que ocorre uma borda de descida do relógio *rel_esc_normal* o sinal *cont_140* é incrementado.

A motivação para a criação do sinal *cont_140* é a necessidade de se saber onde se encontra o primeiro byte do quadro VC-12 transportado pelo sinal *TUG2*. Será visto à frente que através da interpretação do ponteiro (V1V2) obtém-se o sinal *v_pont* (valor do ponteiro). Como foi visto no capítulo 2, o valor do ponteiro é um decimal na faixa de 0 a 139. Assim o sinal *cont_140* é utilizado para identificar cada um dos 140 bytes do quadro TU-12 utilizados para transportar o quadro VC-12.

Interpretação do Ponteiro

Este bloco é modelado pelo processo abaixo. O diagrama temporal da figura 5.61 ilustra o seu funcionamento. Observe que este bloco atua quando o sinal *ender_tu12* muda do valor 35 para o valor 0 e o sinal *ind_subq* tem valor 2. Este ponto do quadro TU-12 foi escolhido porque ele ocorre logo antes do byte de oportunidade de justificação negativa.

```
int_ponteiro : PROCESS(ender_tu12)
variable palavra, palavra_ant, palavra_ant_ant : bit_vector(1 to 16);
variable cont_I, cont_D, cont_NDF : integer;
BEGIN
  IF (ender_tu12 = 0) THEN
   IF (ind_subq = 2) THEN
      palavra_ant_ant := palavra_ant;
      palavra_ant := palavra;
      palavra := byte_pra_palavra(V1, V2);
      cont_I := 0;
      cont_D := 0;
      for i in 7 to 16 loop
         IF (i mod 2 /= 0) THEN
            IF (palavra(i) /= palavra_ant(i) THEN
               cont_I = cont_I + 1;
            END IF;
            IF (palavra(i+1) /= palavra_ant(i+1) THEN
               cont_D = cont_D + 1;
            END IF;
         END IF;
      END loop;
      IF (inv_I = '0' AND inv_D = '0') THEN
         IF (cont_I >= 3) THEN
            inv_I <= '1';
            IF (v_pont = 139) THEN
               v_pont <= 0;
            ELSE
               v_pont <= v_pont + 1;
            END IF;
         END IF;
         IF (cont_D >= 3) THEN
            inv_D <= '1';
            IF (v_pont = 0) THEN
               v_pont <= 139;
            ELSE
               v_pont <= v_pont - 1;
```

```
            END IF;
        END IF;
    ELSE
        inv_I <= '0';
        inv_D <= '0';
    END IF;
    IF (palavra = palavra_ant) AND
          (palavra = palavra_ant_ant) THEN
        v_pont <= palavra_pra_inteiro(palavra);
    END IF;
    cont_NDF := 0;
    for i in 1 to 4 loop
        IF (palavra(i) = v_NDF(i)) THEN
            cont_NDF := cont_NDF + 1;
        END IF;
    END loop;
    IF (cont_NDF >= 3) THEN
        NDF <= '1';
        inv_I <= '0';
        inv_D <= '0';
        v_pont <= palavra_pra_inteiro(palavra);
    ELSE
        NDF <= '0';
    END IF;
  END IF;
 END IF;
END PROCESS;
```

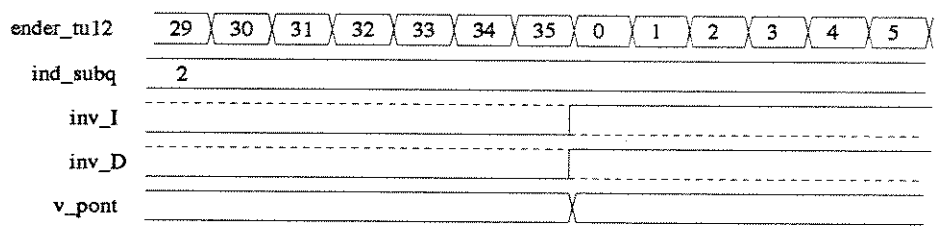


Figura 5.61: Diagrama temporal dos sinais *inv_I*, *inv_D* e *v_pont*.

Este bloco gera os sinais *inv_I*, *inv_D* e *v_pont* a partir dos sinais *V1* e *V2* respeitando as regras de interpretação do ponteiro vistas no capítulo 2, verificando inclusive a indicação de novos dados (NDF).

O sinal *inv_I* quando em nível lógico 1 indica que o ponteiro traz bits I's invertidos e o sinal *inv_D* quando em nível lógico 1 indica que o ponteiro traz bits D's invertidos.

O sinal *v_pont* também é resultado da interpretação do ponteiro e indica em qual dos 140 bytes identificados pelo sinal *cont_140* se encontra o primeiro byte do quadro VC-12. Será visto a seguir que este sinal é necessário na geração do sinal *inic_vc* que marca o início do quadro VC-12.

Gerador do Sinal *inic_vc*

Este bloco é modelado pelas linhas abaixo.

```
inic_vc12 <= '1' WHEN (cont_140 = v_pont) ELSE
    '0';
```

Este bloco gera o sinal *inic_vc* que vai a nível lógico 1 durante o tempo em que o sinal *TUG2* apresenta o byte V5 (primeiro byte do quadro VC-12). O funcionamento deste bloco está ilustrado no diagrama temporal da figura 5.62, o sinal *inic_vc* é 1 quando *cont_140 = v_pont*.

Observe que quando ocorre uma justificação negativa e o valor do ponteiro passa de 0 para 139 o bloco fica um quadro TU-12 sem gerar o pulso indicador de inicio do VC-12. Isto não causa nenhum problema porque o sinal *inic_vc* tem a função de alinhamento, e este não se perde em apenas um quadro. Além disso esta é uma situação muito pouco provável de ocorrer.

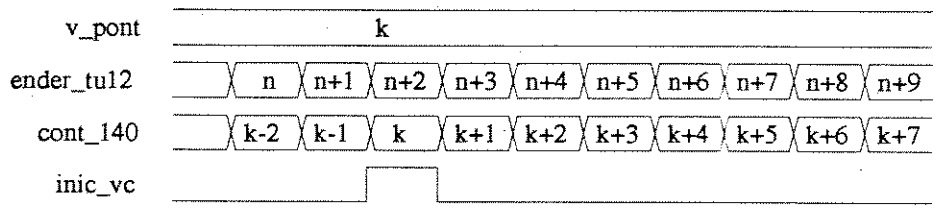


Figura 5.62: Diagrama temporal do sinal *inic_vc*.

Selecionador de Relógios de Escrita

Este bloco é modelado pelas linhas abaixo.

```
rel_esc <= rel_esc_justn WHEN inv_D = '1' ELSE
           rel_esc_justp WHEN inv_I = '1' ELSE
           rel_esc_normal;
```

Este bloco simplesmente seleciona um dos 3 relógios de escrita (*re_esc_normal, rel_esc_justn* e *rel_esc_justp*) para ser o relógio *rel_esc* que é utilizado para se escrever os bytes do quadro VC-12 na memória elástica. Se o sinal *inv_I* estiver em nível lógico 1 indicando que o quadro TU-12 traz uma justificação positiva, então o relógio *rel_esc_justp* é o selecionado. Se o sinal *inv_D* estiver em nível lógico 1 indicando que o quadro TU-12 traz uma justificação negativa, então o relógio *rel_esc_justn* é o selecionado. Do contrário o relógio *rel_esc_normal* é o selecionado para ser o relógio de escrita *rel_esc*.

Contador de Endereço de Escrita

Este bloco é modelado pelo processo abaixo. O diagrama temporal na figura 5.63 ilustra o seu funcionamento.

```
cont_end_esc : PROCESS(rel_esc)
BEGIN
    IF (rel_esc = '0') THEN
        IF (ender_esc = 7) THEN
            ender_esc <= 0;
        ELSE
            ender_esc <= ender_esc + 1;
        END IF;
    END IF;
end PROCESS;
```

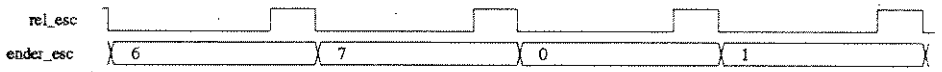


Figura 5.63: Diagram temporal do sinal *ender_esc*.

O sinal *ender_esc* fornece os enderços da memória elástica onde devem ser escritos, durante as bordas de subida do relógio *rel_esc*, os bytes do quadro VC-12.

Memória Elástica

Este bloco é modelado pelos 2 processos abaixo.

```
esc_mem_elast : PROCESS(rel_esc)
BEGIN
    IF (rel_esc = '1') AND (rel_esc'EVENT) THEN
        mem_elast(ender_esc) <= TUG2;
        nono_bit(ender_esc) <= inic_vc;
    END IF;
END PROCESS;
```

```
leit_mem_elast : PROCESS(clk_vc12)
BEGIN
    IF (clk_vc12 = '1') AND (clk_vc12'EVENT) THEN
        VC12 <= mem_elast(ender_leit);
        inic_vc12 <= nono_bit(ender_leit);
    END IF;
END PROCESS;
```

Esta memória elástica é igual à memória elástica do bloco de adaptação de seção diferindo apenas no tamanho.

Aqui define-se uma memória elástica de tamanho oito, com limiar mínimo de ocupação de 2 bytes e limiar máximo de 6 bytes.

Gerador de Sinais Indicadores de Buracos em *clk_tu12_local*

As linhas de comandos abaixo modelam este bloco.

```
bur1_l <= '1' WHEN ender_tu12_local = 0 AND ind_subq_local /= 2 ELSE
```

```
'0';
bur2_l <= '1' WHEN ender_tu12_local = 0 AND ind_subq_local = 2 ELSE
'0';
bur3_l <= '1' WHEN ender_tu12_local = 1 AND ind_subq_local = 2 ELSE
'0';
```

A estrutura deste bloco é idêntica a estrutura do bloco Gerador de Sinais Indicadores de Buracos em *clk_tu12*. Os diagramas temporais das figuras 5.64 e 5.65 ilustram o seu funcionamento.

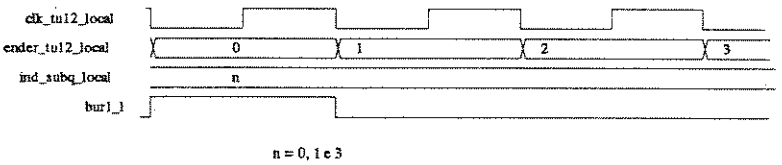


Figura 5.64: Diagrama temporal do sinal *bur1_l*.

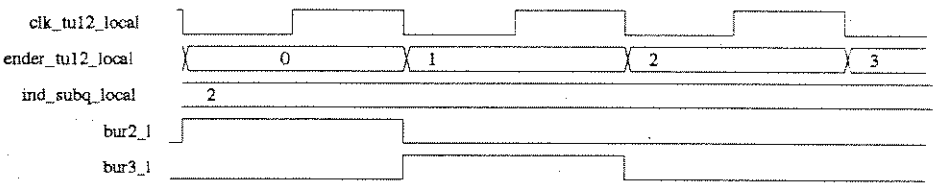


Figura 5.65: Diagram temporal dos sinais *bur2_l* e *bur3_l*.

Gerador de Relógios de Leitura

As linhas abaixo modelam este bloco.

```
rel_leit_normal <= clk_tu12_local AND not(bur1_l) AND not(bur2_l);
rel_leit_justp <= clk_tu12_local AND not(bur1_l) AND not(bur2_l) AND not(bur3_l);
rel_leit_justn <= clk_tu12_local AND not(bur1_l);
```

A estrutura deste bloco é idêntica a estrutura do bloco Gerador de Relógios de Escrita. Sua função é a de gerar relógios de leitura de memória elástica para serem utilizados em três situações distintas: quando não há justificação utiliza-se o relógio *rel_leit_normal*, quando há justificação negativa utiliza-se o relógio *rel_leit_justn* e quando há justificação positiva utiliza-se o relógio *rel_leit_justp*. Os diagramas temporais das figuras 5.66 e 5.67 ilustram estes sinais.

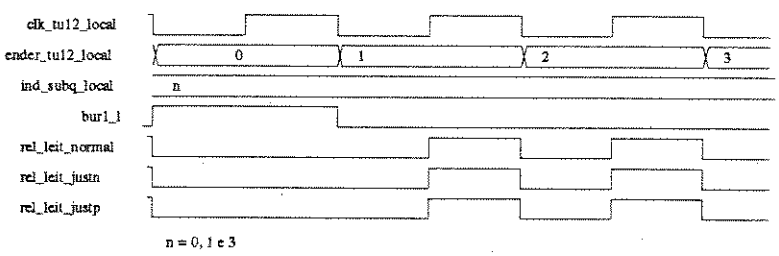


Figura 5.66: Relógios de leitura nos subquadros 0, 1, e 3.

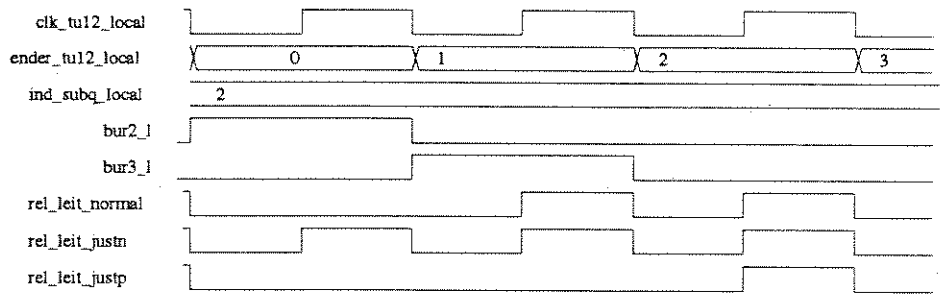


Figura 5.67: Relógios de leitura no subquadro 2.

Vê-se pelas figuras que o relógio *rel_leit_normal* tem um buraco durante o primeiro byte do quadro TU-12 de qualquer subquadro. Os relógios *rel_leit_justn* e *rel_leit_justp* diferem do *rel_leit_normal* somente no terceiro subquadro (*ind_subq_local = 2*). Neste subquadro o relógio *rel_leit_justn* não tem nenhum buraco e o relógio *rel_leit_justp* tem buracos nos dois primeiros bytes do quadro TU-12 local.

Selecionador de Relógios de Leitura

Este bloco é modelado pelas linhas abaixo.

```
clk_vc12 <= rel_leit_justn WHEN just_n = '1' ELSE
            rel_leit_justp WHEN just_p = '1' ELSE
            rel_leit_normal;
```

A estrutura deste bloco é idêntica à estrutura do bloco Selecionador de Relógios de Escrita. Ele simplesmente seleciona um dos 3 relógios de leitura (*re_leit_normal, rel_leit_justn* e *rel_leit_justp*) para ser o relógio *clk_vc12* que é utilizado para se ler os bytes do quadro VC-12 da memória elástica. Se o sinal *just_p* estiver em nível lógico 1 indicando que a memória elástica atingiu seu limiar inferior de ocupação, então o relógio *rel_leit_justp* é o selecionado. Se o sinal *just_n* estiver em nível lógico 1 indicando que a memória elástica atingiu seu limiar superior de ocupação, então o relógio *rel_leit_justn* é o selecionado. Do contrário o relógio *rel_leit_normal* é o selecionado para ser o relógio de leitura *clk_vc12*.

Justificador

Este bloco é modelado pelo processo abaixo. Ele gera os sinais *just_p* e *just_n* que indicam quando o quadro STM-1 local apresenta justificações positivas e negativas respectivamente.

```
justificador : PROCESS(ind_subq_local)
BEGIN
    IF (ind_subq_local = 3) AND (ind_subq_local'EVENT) THEN
       IF (just_dep_3 = 2) THEN
          IF (fase <= 2) THEN
             just_p <= '1';
             just_dep_3 <= 0;
          END IF;
          IF (fase >= 6) THEN
```

```
            just_n <= '1';
            just_dep_3 <= 0;
        END IF;
    ELSE
        just_dep_3 <= just_dep_3 + 1;
        just_p <= '0';
        just_n <= '0';
    END IF;
  END IF;
END PROCESS;
```

No instante em que o sinal *ind_subq_local* atinge o valor 3 a decisão de se justificar é tomada. Se já tiverem decorridos 3 quadros desde a última justificação então a fase entre o contador de endereço de leitura *end_leit* e contador de endereço de escrita *end_esc* é verificada. Se a fase for menor ou igual a 2 bytes então o sinal de controle *just_p* toma o estado lógico '1', sinalizando para a necessidade de uma justificação positiva. Se a fase for maior ou igual a 6 bytes então o sinal de controle *just_n* toma o valor '1', sinalizando para a necessidade de uma justificação negativa.

Contador de Endereços de Leitura

Este bloco é modelado pelo processo abaixo. O diagrama temporal da figura 5.68 ilustra o seu funcionamento.

```
cont_end_leit : PROCESS(clk_vc12)
BEGIN
  IF (clk_vc12 = '0') AND (clk_vc12'EVENT) THEN
      IF (ender_leit = 7) THEN
          ender_leit <= 0;
      ELSE
          ender_leit <= ender_leit + 1;
      END IF;
  END IF;
END PROCESS;
```

Este bloco é igual ao bloco Contador de Endereços de Escrita.

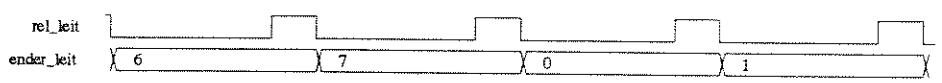


Figura 5.68: Diagrama temporal do sinal *ender_leit*.

Fasimetro

Este bloco é modelado pelo processo abaixo. Sua função é simplesmente medir a distância entre o endereço de leitura do endereço de escrita em memória elástica e reportar o valor através do sinal *fase*.

```
fasimetro : PROCESS(ender_esc, ender_leit)
variable v_fase : integer := 0;
BEGIN
    v_fase := ender_esc - ender_leit;
    IF (v_fase < 0) THEN
       fase <= 8 + v_fase;
    ELSE
       fase <= v_fase;
    END IF;
END PROCESS;
```

5.6 Gerador dos Sinais de Controle Locais

A figura 5.69 apresenta o detalhamento deste bloco. Sua função é gerar sinais de controle e temporização para a montagem das estruturas a serem transmitidas. A seguir analisa-se cada um dos blocos da figura 5.69 através de diagramas temporais de seus sinais de entrada e de saída e dos comandos VHDL que os modelam.

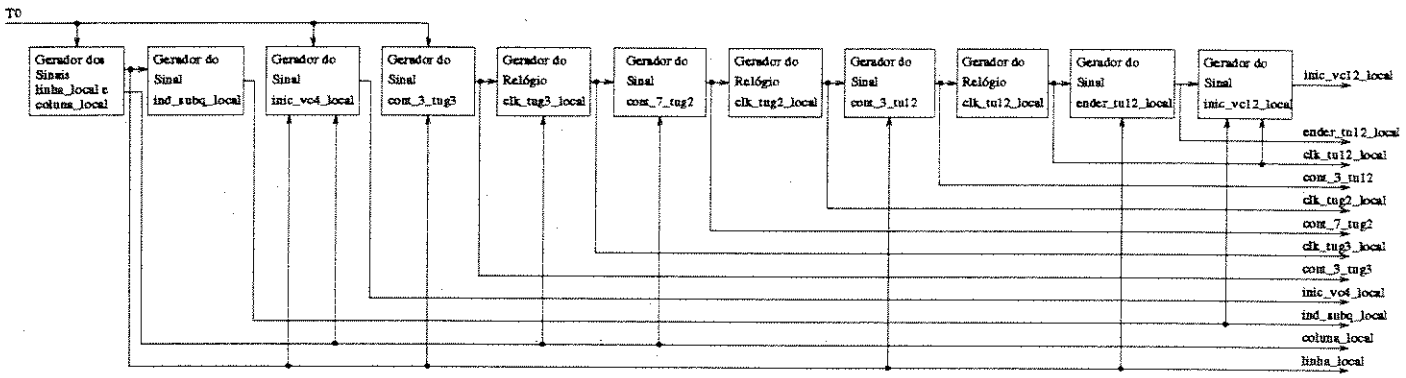


Figura 5.69: Detalhamento do Bloco Gerador dos Sinais de Controle Locais.

5.6.1 Gerador dos Sinais *linha_local* e *coluna_local*

Modelo em VHDL:

```
Gerador_dos_Sinais_linha_local_e_coluna_local :
PROCESS(T0)
BEGIN
    IF (T0 = '0') AND (T0'EVENT) THEN
       IF (coluna_local = 269) THEN
          IF (linha_local = 8) THEN
             linha_local <= 0;
             coluna_local <= 0;
          ELSE
             linha_local <= linha_local + 1;
             coluna_local <= 0;
          END IF;
       ELSE
          coluna_local <= coluna_local + 1;
       END IF;
    END IF;
END PROCESS;
```

O diagrama temporal dos sinais *coluna_local* e *linha_local* já foi apresentado anteriormente na figura 5.9. Observe naquela figura que as bordas de subida do relógio *T0* e os valores dos sinais *linha_local* e *coluna_local* marcam os instantes de inserção dos 2430 bytes do quadro STM-1 local a ser transmitido.

5.6.2 Gerador do Sinal *ind_subq_local*

Modelo em VHDL:

```
Gerador_do_Sinal_ind_subq_local :
PROCESS(linha_local)
BEGIN
    IF (linha_local = 8) AND (linha_local'EVENT) THEN
        IF (ind_subq_local = 3) THEN
            ind_subq_local <= 0;
        ELSE
            ind_subq_local <= ind_subq_local + 1;
        END IF;
    END IF;
END PROCESS;
```

O diagrama temporal do sinal *ind_subq_local* está mostrado na figura 5.70. Observe que quando o sinal *linha_local* atinge o valor 8 o sinal *ind_subq_local* é incrementado ou zerado se o seu valor for 3. Portanto o ponto do quadro STM-1 escolhido para se modificar o valor do sinal *ind_subq_local* foi o ponto no qual o sinal *linha_local* atinge o valor 8. A razão para a escolha deste ponto pode ser entendida analisando-se a figura 5.71. Esta figura mostra a posição fixa ocupada pelo byte H4 do VC-4 montado localmente a ser transmitido pelo STM-1 local. O sinal *ind_subq_local* é ultilizado para se gerar o byte H4 e por isso o ponto no qual o sinal *linha_local* atinge o valor 8 é adequado para a atualização do sinal *ind_subq_local*, uma vez que este ponto acontece pouco antes da inserção do byte H4 no quadro STM-1 local.

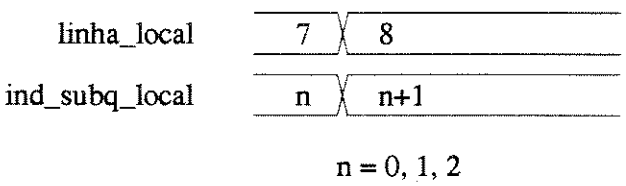


Figura 5.70: Diagrama temporal do sinal *ind_subq_local*.

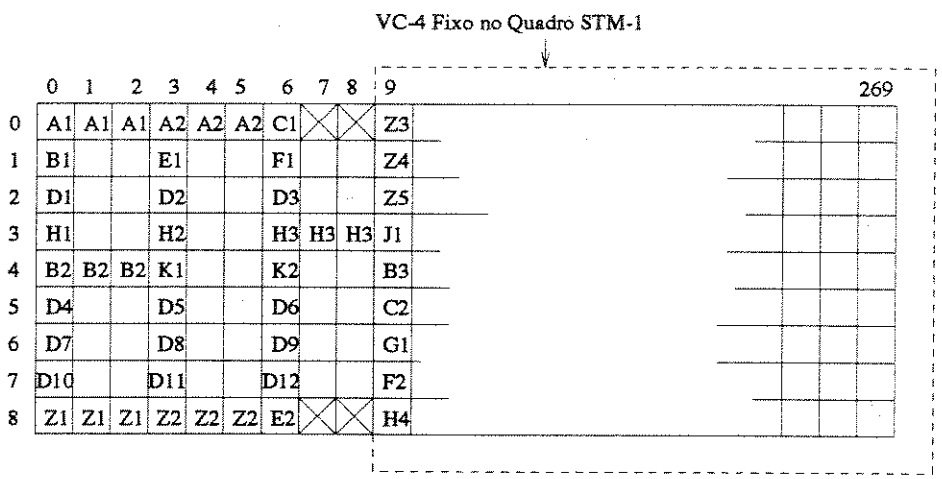


Figura 5.71: Quadro STM-1 montado localmente contendo quadro VC-4 em posição fixa.

5.6.3 Gerador do Sinal *inic_vc4_local*

Modelo em VHDL:

```
Gerador_do_Sinal_inc_vc4_local :
PROCESS(T0)
BEGIN
    IF (T0 = '0') AND (T0'EVENT) THEN
        IF (coluna_local = 9) AND (linha_local = 3) THEN
            inic_vc4_local <= '1';
        ELSE
            inic_vc4_local <= '0';
        END IF;
    END IF;
END PROCESS;
```

A figura 5.72 mostra o diagrama temporal do sinal *inic_vc4_local* em relação aos sinais *coluna_local*, *linha_local* e *T0*. Observe que o sinal *inic_vc4_local* vai a nível lógico 1 quando ocorre a borda de subida do relógio *T0* na qual os sinais *linha_local* e coluna_local estão com valores 3 e 9 respectivamente. A figura 5.71 mostra a posição fixa do byte J1(primeiro byte do VC-4 montado localmente). Observe que ele se encontra na linha 3 e coluna 9 do quadro STM-1 local. Portanto o sinal *inic_vc4_local* vai a nível lógico 1 quando o byte J1 do VC-4 local está sendo inserido no STM-1 local.

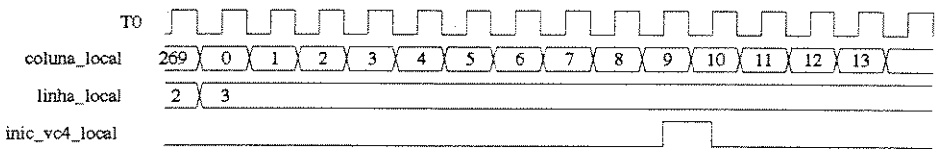


Figura 5.72: Diagrama temporal do sinal *inic_vc4_local*.

5.6.4 Gerador do Sinal *cont_3_tug3*

Modelo em VHDL:

```
Gerador_do_Sinal_cont_3_tug3 :
PROCESS(T0, linha_local)
BEGIN
    IF (T0 = '0') AND (T0'EVENT) THEN
        IF (cont_3_tug3 = 2) THEN
            cont_3_tug3 <= 0;
        ELSE
            cont_3_tug3 <= cont_3_tug3 + 1;
        END IF;
    END IF;
    IF (linha_local = 0) AND (linha_local'EVENT) THEN
        cont_3_tug3 <= 0;
    END IF;
END PROCESS;
```

A figura 5.73 mostra o diagrama temporal do sinal *cont_3_tug3* em relação ao sinais *linha_local* e *T0*. Observe que o sinal *cont_3_tug3* rotula os pulsos de *T0* com os valores 0, 1 e 2. No ponto no qual o sinal *linha_local* atinge o valor 8 se faz o alinhamento do sinal *cont_3_tug3*, isto é, seu valor deve ser zero durante o primeiro byte do quadro STM-1 local. Será visto que este sinal é necessário na multiplexação de sinais *TUG3's*.

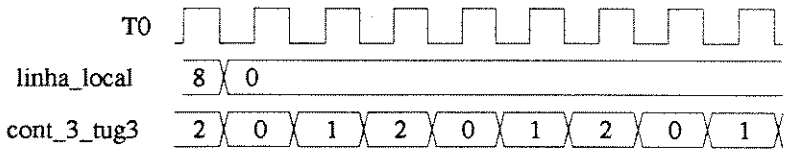


Figura 5.73: Diagrama temporal do sinal *cont_3_tug3*.

5.6.5 Gerador do Relógio *clk_tug3_local*

Modelo em VHDL:

Gerador_do_Relógio_clk_tug3_local :
clk_tug3_local <= '1' WHEN (coluna_local > 11) AND (cont_3_tug3 = 0) ELSE
'0';

A figura 5.74 mostra o diagram temporal do relógio *clk_tug3_local* em relação aos sinais *coluna_local*, *cont_3_tug3* e *T0*. Será visto que as bordas de subida do relógio *clk_tug3_local* são usadas para a montagem dos quadros TUG-3 locais.

Observe que o relógio *clk_tug3_local* vai a nível lógico 1 quando o sinal *coluna_local* é maior que 11 e o sinal *cont_3_tug3* é zero. Observe também que entre 2 bordas de subida consecutivas do relógio *clk_tug3_local* ocorrem 3 bordas de subida do relógio *T0* que são utilizadas para a multiplexagem de 3 quadros TUG-3's locais.

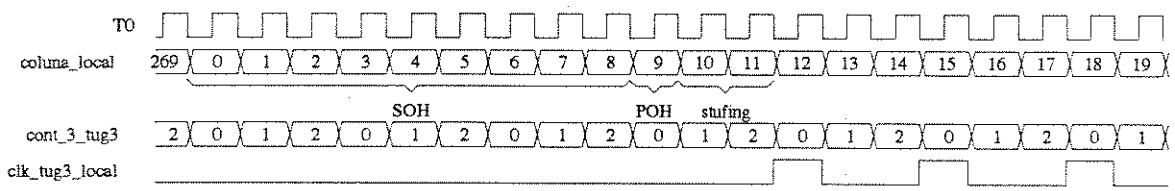


Figura 5.74: Diagram temporal do relógio *clk_tug3_local*.

5.6.6 Gerador do Sinal *cont_7_tug2*

Modelo em VHDL:

```
Gerador_do_Sinal_cont_7_tug2 :
PROCESS(clk_tug3_local, coluna_local)
BEGIN
    IF (clk_tug3_local = '0') AND (clk_tug3_local'EVENT) AND
        (coluna_local > 14) AND (coluna_local < 267) THEN
       IF (cont_7_tug2 = 6) OR (cont_7_tug2 = 7)THEN
          cont_7_tug2 <= 0;
       ELSE
          cont_7_tug2 <= cont_7_tug2 + 1;
       END IF;
    END IF;
    IF (coluna_local <= 15) OR (coluna_local >= 268) AND (coluna_local'EVENT) THEN
       cont_7_tug2 <= 7;
    END IF;
END PROCESS;
```

A figura 5.75 mostra o diagrama temporal do sinal *cont_7_tug2* em relação aos sinais *coluna_local* e *clk_tug3_local*. Será visto que este sinal é necessário na multiplexação de quadros TUG-2's.

Pela figura 5.39 vê-se que as 2 primeiras colunas do quadro TUG3 são de "stufing". Por isso o sinal *cont_7_tug2* assume o valor 7 durante os 2 primeiros pulsos do relógio *clk_tug3_local* para informar que se trata de "stufing".

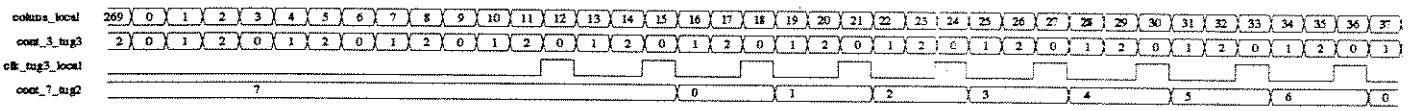


Figura 5.75: Diagrama temporal do sinal *cont_7_tug2*.

5.6.7 Gerador do Relógio *clk_tug2_local*

Modelo em VHDL:

```
Gerador_do_Relógio_clk_tug2_local :
clk_tug2_local <= '1' WHEN (cont_7_tug2 = 0) ELSE
    '0';
```

A figura 5.76 mostra o diagrama temporal do relógio *clk_tug2_local* em relação ao sinal *cont_7_tug2* e ao relógio *clk_tug3_local*.

Observe que o relógio *clk_tug2_local* vai a nível lógico 1 quando o sinal *cont_7_tug2* assume o valor zero. Observe também que entre 2 bordas de subida consecutivas do relógio *clk_tug2_local* ocorrem 7 bordas de subida do relógio *clk_tug3_local*. Esta observação é importante para a multiplexagem de quadros TUG-2's em quadros TUG-3's.

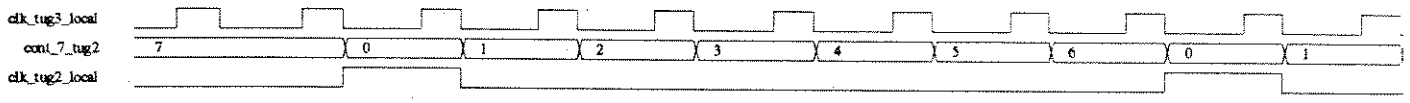


Figura 5.76: Diagrama temporal do relógio *clk_tug2_local*.

5.6.8 Gerador do Sinal *cont_3_tu12*

Modelo em VHDL:

```
Gerador_do_Sinal_cont_3_tu12 :
PROCESS(clk_tug2_local, linha_local)
BEGIN
    IF (linha_local = 0) AND (linha_local'EVENT) THEN
       cont_3_tu12 <= 0;
    END IF;
    IF (clk_tug2_local = '0') AND (clk_tug2_local'EVENT) THEN
       IF (cont_3_tu12 = 2) THEN
          cont_3_tu12 <= 0;
       ELSE
          cont_3_tu12 <= cont_3_tu12 + 1;
       END IF;
    END IF;
END PROCESS;
```

A figura 5.77 mostra o diagrama temporal do sinal *cont_3_tu12* em relação aos sinais *linha_local* e *clk_tug2_local*. O ponto no qual o sinal *linha_local* assume o valor zero é usado para alinhar o sinal *cont_3_tu12*. Neste ponto o sinal *cont_3_tu12* deve ter valor zero. A cada borda de descida do relógio *clk_tug2_local* o sinal *cont_3_tu12* é incrementado ou assume o valor zero se o seu valor for 2.

Será visto que este sinal é necessário na multiplexação de quadros TU-12's locais.

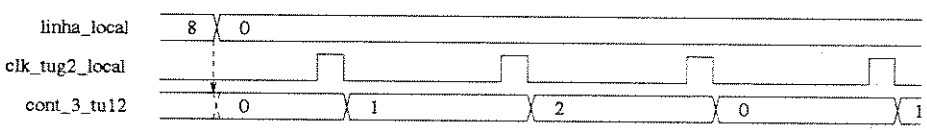


Figura 5.77: Diagrama temporal do sinal *cont_3_tu12*.

5.6.9 Gerador do Relógio *clk_tu12_local*

Modelo em VHDL:

```
Gerador_do_Relógio_clk_tu12_local :
clk_tu12_local <= '1' WHEN (cont_3_tu12 = 0) ELSE
    '0';
```

A figura 5.78 mostra o diagrama temporal do relógio *clk_tu2_local* em relação aos sinais *cont_3_tu12* e *clk_tug2_local*. Observe que o relógio *clk_tu12_local* vai a nível lógico 1 quando o sinal *cont_3_tu12* assume o valor zero.

Observe que entre 2 bordas de subida consecutivas do relógio *clk_tu12_local* ocorrem 3 bordas de subida do relógio *clk_tug2_local*. Esta observação é importante para a multiplexação de quadros TU-12's em quadros TUG-2's.

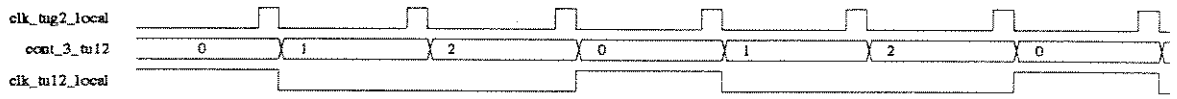


Figura 5.78: Diagrama temporal do relógio *clk_tu2_local*.

5.6.10 Gerador do Sinal *ender_tu12_local*

Modelo em VHDL:

```
Gerador_do_Sinal_ender_tu12_local :
PROCESS(clk_tu12_local, linha_local)
BEGIN
    IF (linha_local = 3) AND (linha_local'EVENT) THEN
       ender_tu12_local <= 0;
    END IF;
    IF (clk_tu12_local = '0') AND (clk_tu12_local'EVENT) THEN
       IF (ender_tu12_local = 35) THEN
          ender_tu12_local <= 0;
       ELSE
          ender_tu12_local <= ender_tu12_local + 1;
       END IF;
    END IF;
END PROCESS;
```

A figura 5.79 mostra o diagrama temporal do sinal *ender_tu12_local* em relação aos sinais *linha_local* e *coluna_local*. Observe que o ponto no qual o sinal linha_local muda seu valor de 2 para 3 serve para alinhar o sinal *ender_tu12_local*(assume o valor zero neste ponto). A cada borda de descida do relógio *clk_tu12_local* o sinal *ender_tu12_local* é incrementado ou zerado se o seu valor for 35.

O sinal *ender_tu12_local* serve para identificar os 36 bytes do quadro TU-12 local.

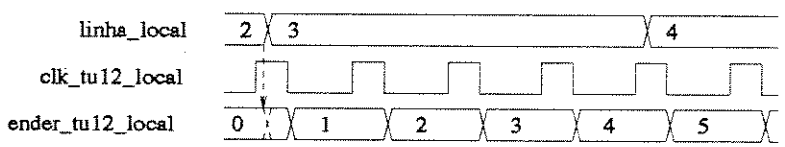


Figura 5.79: Diagrama temporal do sinal *ender_tu12_local*.

5.6.11 Gerador do Sinal *inic_vc12_local*

Modelo em VHDL:

```
Gerador_do_Sinal_inic_vc12_local :
PROCESS(clk_tu12_local)
BEGIN
    IF (clk_tu12_local = '1') AND (clk_tu12_local'EVENT) THEN
        IF (ind_subq_local = 1) AND (ender_tu12_local = 1) THEN
            inic_vc12_local <= '1';
        ELSE
            inic_vc12_local <= '0';
        END IF;
    END IF;
END PROCESS;
```

A figura 5.80 mostra o diagrama temporal do sinal *inic_vc12_local* em relação aos sinais *ender_tu12_local*, *ind_subq_local* e *clk_tu12_local*. Observe que o sinal *inic_vc12_local* vai a nível lógico 1 quando ocorre a borda de subida do relógio *clk_tu12_local* na qual os sinais *ender_tu12_local* e *ind_subq_local* estão com valores 1 e 2 respectivamente. Portanto o sinal *inic_vc12_local* vai a nível lógico 1 quando o byte V5 do VC-12 local está sendo inserido no TU-12 local.

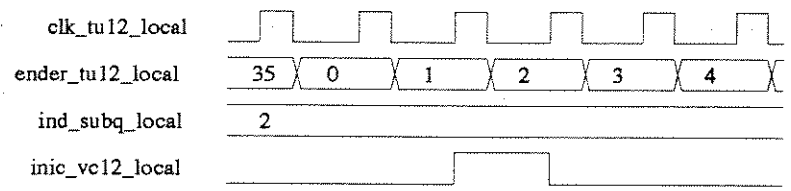


Figura 5.80: Diagrama temporal do sinal *inic_vc12_local*.

5.7 Adaptação de Rotas de Alta Ordem na Transmissão (HPA_T)

O bloco HPA_T está detalhado na figura 5.81.

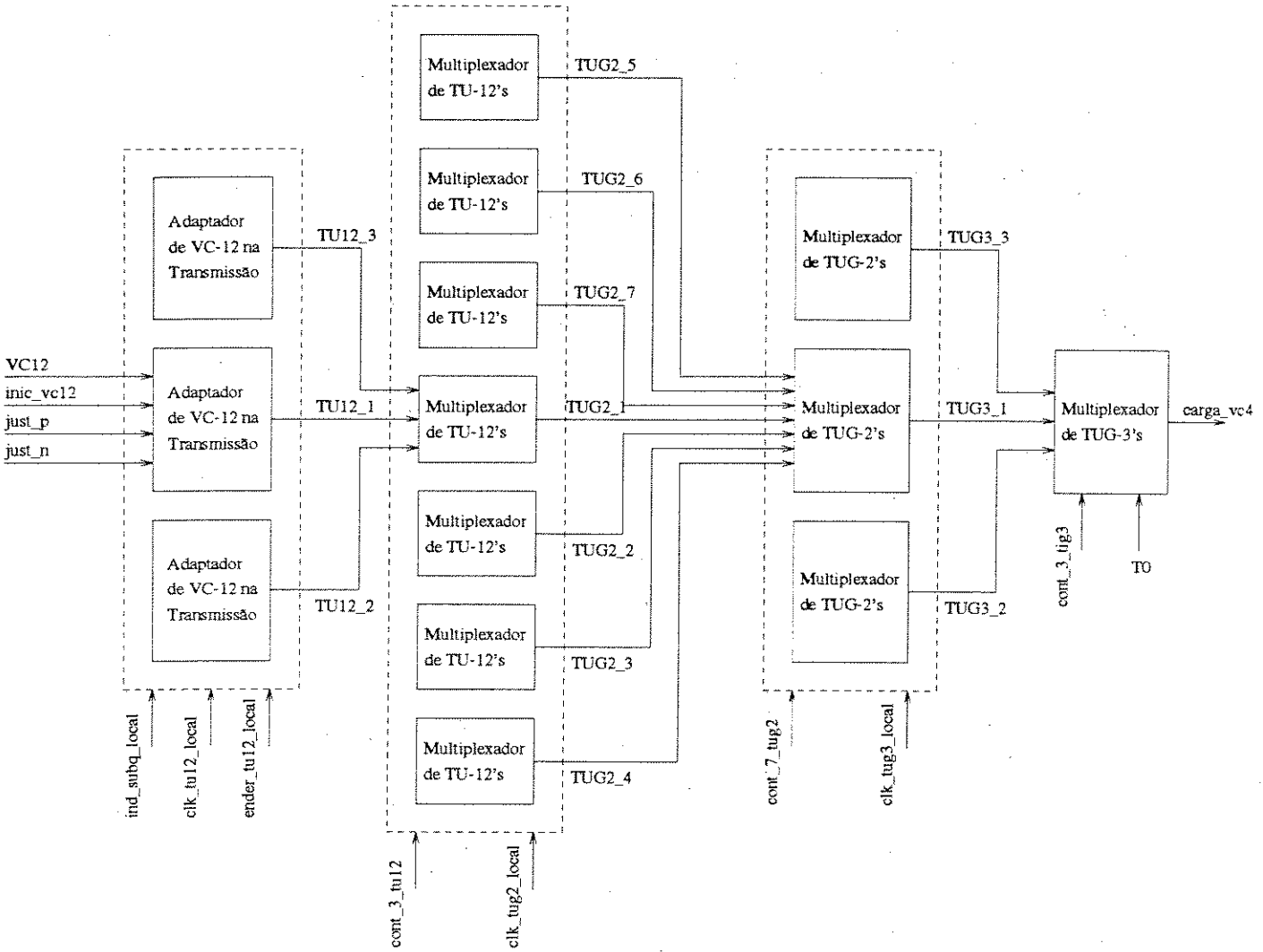


Figura 5.81: Adaptação de Rotas de Alta Ordem na Transmissão (HPA_T).

A figura 5.81 representa um bloco Multiplexador de TUG-3's, 3 blocos Multiplexadores de TUG-2's, 21 blocos Multiplexadores de TU-12's e 63 blocos Adaptadores de VC-12 na Transmissão.

A seguir detalha-se os quatro tipos de blocos da figura 5.81.

5.7.1 Adaptador de VC-12 na Transmissão

O Adaptador de VC-12 na Transmissão está detalhado através do diagrama de blocos na figura 5.82. A seguir analisa-se cada um dos blocos da figura 5.82 através de diagramas temporais de seus sinais de entrada e de saída e dos comandos VHDL que os modelam.

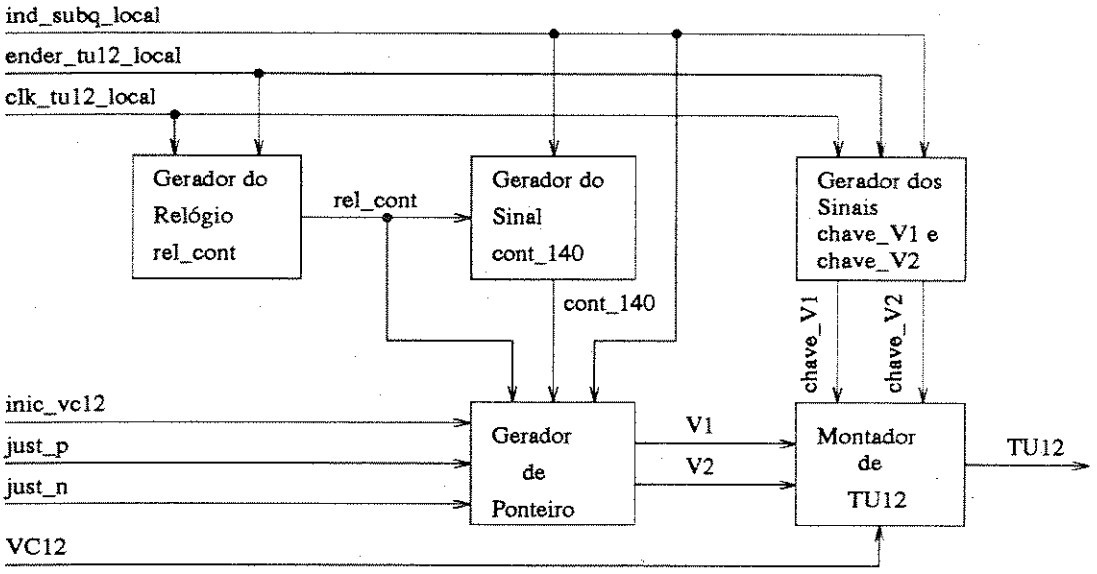


Figura 5.82: Detalhamento do bloco Adaptador de VC-12 na Transmissão.

Gerador do Relógio *rel_cont*

O processo abaixo modela este bloco.

```
rel_cont <= clk_tu12_local WHEN ender_tu12_local /= 0 ELSE
            '0';
```

A figura 5.83 ilustra o funcionamento deste bloco. Sua função é gerar o relógio *rel_cont* que é o relógio *clk_tu12_local* com buracos quando o sinal *ender_tu12_local* assume valor zero.

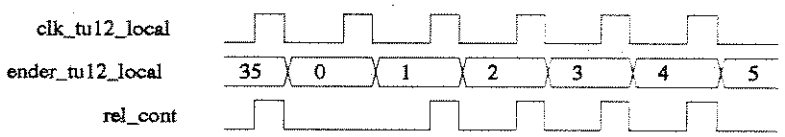


Figura 5.83: Diagrama temporal do sinal *rel_cont*.

Gerador do Sinal *cont_140*

Este bloco é modelado pelo processo abaixo. Sua função é criar um sinal que conta de 0 até 139 para identificar cada um dos 140 bytes da carga útil do container VC-12.

```
contador : PROCESS(rel_cont, ender_tu12_local)
BEGIN
   IF (ender_tu12_local = 1) AND (ender_tu12_local'EVENT) AND (ind_subq_local = 1) THEN
      cont_140 <= 0;
    END IF;
   IF (rel_cont = '0') AND (rel_cont'EVENT) THEN
      cont_140 <= cont_140 + 1;
   END IF;
END PROCESS;
```

Gerador de Ponteiro

Este bloco é modelado pelo processo abaixo.

```
gerador_ponteiro: PROCESS(inic_vc12, ind_subq_local)
BEGIN
   IF (ender_tu12_local = 0) AND (ender_tu12_local'EVENT) THEN
      IF (ind_subq_local = 3) THEN
         IF (NDF = '1') THEN
            monta_ponteiro_com_NDF(V1, V2, v_pont);
         ELSE
            IF (just_p = '1') THEN
               inverte_bits_I(V1, V2);
            ELSIF (just_n = '1') THEN
               inverte_bits_D(V1, V2);
            ELSE
               monta_ponteiro(V1, V2, v_pont);
            END IF;
         END IF;
      ELSIF (ind_subq_local = 2) THEN
         IF (just_p = '1') THEN
            IF (v_pont = 139) THEN
               v_pont <= 0;
            ELSE
               v_pont <= v_pont + 1;
            END IF;
         ELSIF (just_n = '1') THEN
            IF (v_pont = 0) THEN
               v_pont <= 139;
            ELSE
               v_pont <= v_pont - 1;
            END IF;
         END IF;
      END IF;
   END IF;
    IF (inic_vc12 = '1') AND (inic_vc12'EVENT) THEN
      IF (cont_140 = v_pont) THEN
         NDF <= '0';
      ELSE
         NDF <= '1';
         v_pont <= cont_140;
      END IF;
```

END IF;
END PROCESS;

Este bloco atua em dois instantes do quadro STM-1 local. O primeiro está ilustrado nas figuras 5.84 e 5.85.

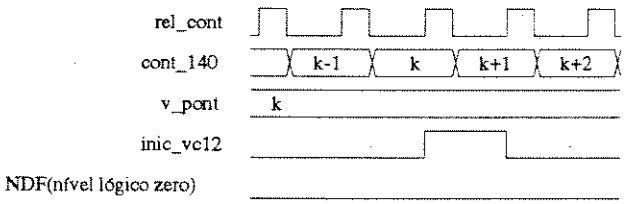


Figura 5.84: Gerador de Ponteiro detectando posição correta do início do VC-12.

A figura 5.84 pode ser assim interpretada: a borda de subida do sinal *inic_vc12* indica que o byte presente no sinal *VC12* é o primeiro do quadro VC-12, então é verificado se o sinal *v_pont* se iguala ao sinal *cont_140*. Caso eles sejam iguais, significa que o sinal *v_pont* está indicando a posição correta de início do quadro VC-12 dentro do quadro TU-12 local. Se os sinais *v_pont* e *cont_140* não forem iguais, significa que o sinal *v_pont* não está indicando a posição correta do primeiro byte do VC-12. Neste caso o sinal *v_pont* assume o valor do sinal *cont_140* e o sinal *NDF* (interno ao bloco Gerador de Ponteiro) assume o valor lógico 1, como mostrado na figura 5.85.

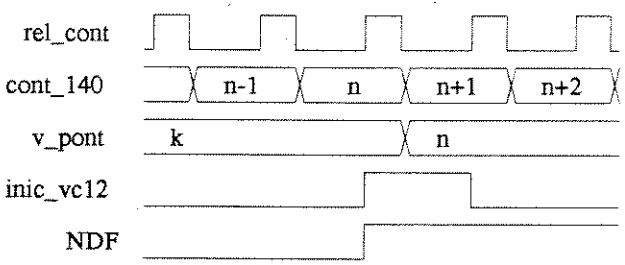


Figura 5.85: Gerador de Ponteiro detectando NDF (novos dados).

O segundo instante de atuação do bloco Gerador de Ponteiro ocorre quando o sinal *ind_subq_local* muda seu valor de 3 para 0. Isto está mostrado nas figuras 5.86, 5.87 e 5.88.

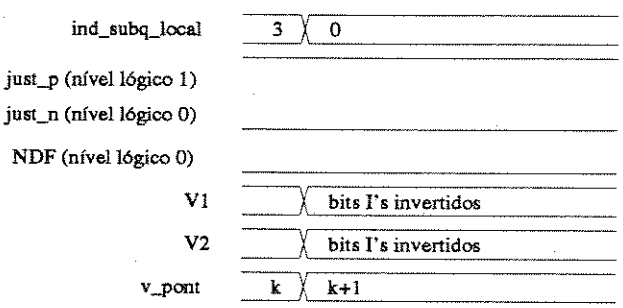


Figura 5.86: Gerador de Ponteiro operando durante justificação positiva.

A figura 5.86 ilustra o funcionamento do bloco Gerador de Ponteiro quando ele recebe uma indicação de justificação positiva (sinal *just_p* = '1'). Observe que nesta ocasião os sinais *V1* e *V2* têm os seus bits I's invertidos.

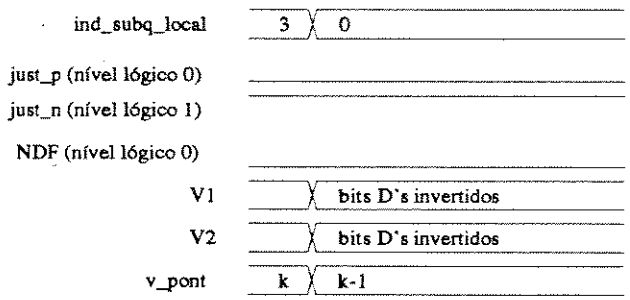


Figura 5.87: Gerador de Ponteiro operando durante justificação negativa.

A figura 5.87 ilustra o funcionamento do bloco Gerador de Ponteiro quando ele recebe uma indicação de justificação negativa (sinal *just_n* = '1'). Observe que que nesta ocasião os sinais *V1* e *V2* têm os seus bits D's invertidos.

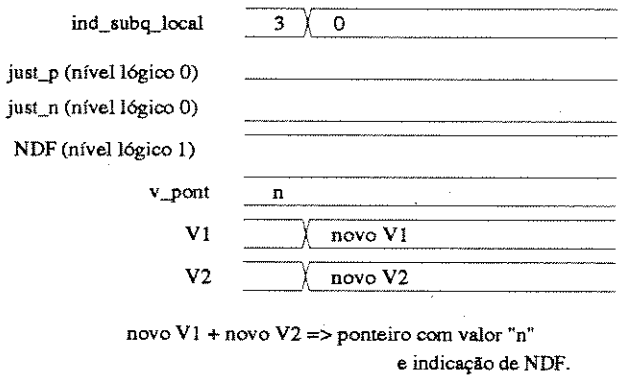


Figura 5.88: Gerador de Ponteiro operando depois da detecção de NDF.

A figura 5.88 iustra a funcionamento do Gerador de Ponteiro depois da detecção de uma indicação de novos dados (sinal *NDF* = '1'). Observe que que nesta ocasião os sinais *V1* e *V2* sinalizam a existência de novos dados e indicam um novo valor de ponteiro.

5.7.2 Gerador dos Sinais *chave_V1* e *chave_V2*

Modelo em VHDL:

```
chaves_para_V1_e_V2 : PROCESS(clk_tu12_local)
BEGIN
    IF (clk_tu12_local = '1') AND (clk_tu12_local'EVENT) THEN
        IF (ind_subq_local = 0) AND (ender_tu12_local = 0) THEN
            chave_V1 <= '1';
        ELSIF (ind_subq_local = 1) AND (ender_tu12_local = 0) THEN
            chave_H2 <= '1';
        ELSE
            chave_H1 <= '0';
            chave_H2 <= '0';
        END IF;
    END IF;
END PROCESS;
```

A figura 5.89 ilustra o funcionamento deste bloco.

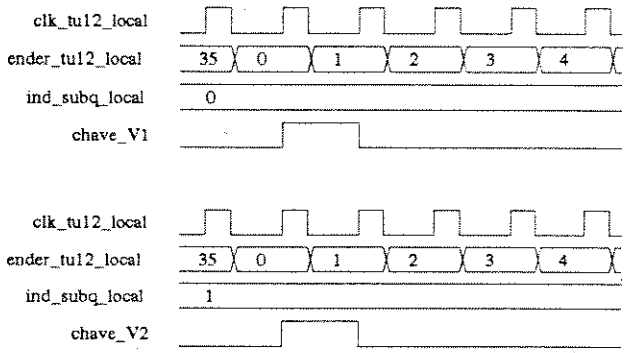


Figura 5.89: Diagrama temporal dos sinais *chave_V1* e *chave_V2*.

Estes sinais servem para inserir os bytes V1 e V2 de ponteiro no quadro TU-12 local.

5.7.3 Montador de TU-12

Modelo em VHDL:

```
TU12 <= V1 WHEN chave_V1 = '1' ELSE
    V2 WHEN chave_V2 = '1' ELSE
    VC12;
```

A figura 5.90 ilustra o funcionamento deste bloco. Ele simplesmente insere o byte V1 na posição *ind_subq_local = 0, ender_tu12_local = 0* e o byte V2 na posição *ind_subq_local = 1, ender_tu12_local = 0*.

5.7.4 Multiplexador de TU-12's

Modelo em VHDL:

```
Multiplexador_de_TU-12 : PROCESS(clk_tug2_local)
BEGIN
   IF (clk_tug2_local = '1') AND (clk_tug2_local'EVENT) THEN
      CASE cont_3_tu12 IS
         WHEN 0 => TUG2 <= TU12_1;
         WHEN 1 => TUG2 <= TU12_2;
         WHEN 2 => TUG2 <= TU12_3;
         WHEN OTHERS => NULL;
      END CASE;
   END IF;
END PROCESS;
```

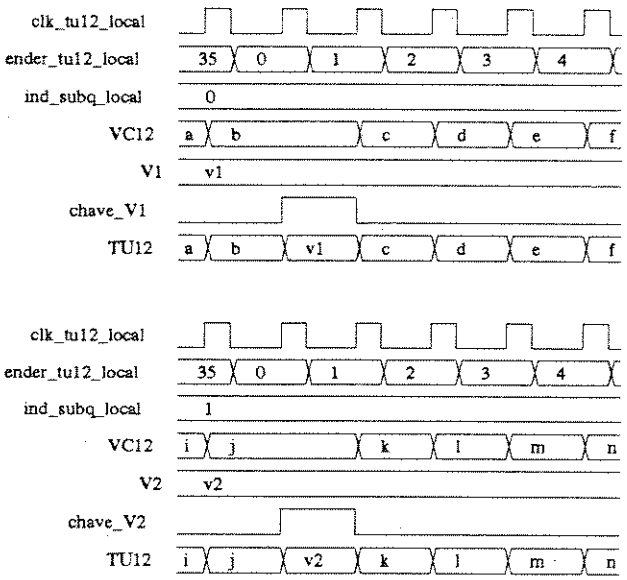


Figura 5.90: Montagem do sinal *TU12*.

Este bloco é modelado apenas pelo processo acima. Sua função é multiplexar os sinais *TU12_1*, *TU12_2* e *TU12_3* para formar o sinal *TUG2*. A figura 5.91 mostra o diagrama temporal destes sinais.

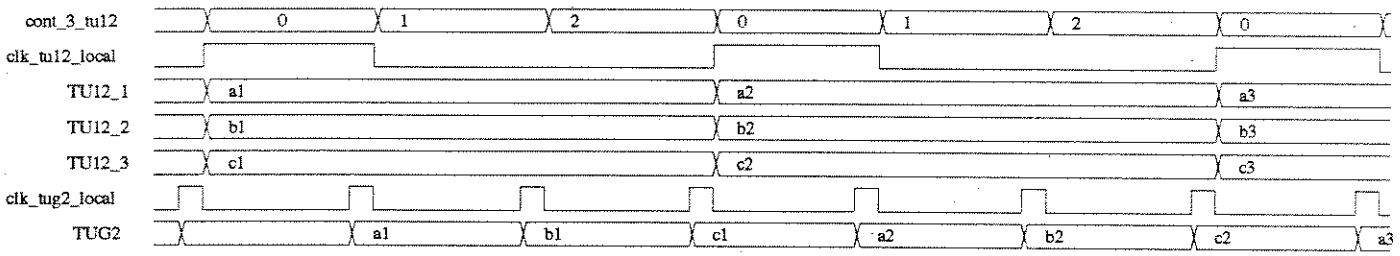


Figura 5.91: Diagrama temporal dos sinais *TU12_1*, *TU12_2* e *TU12_3*.

5.7.5 Multiplexador de TUG-2's

Modelo em VHDL:

```
Multiplexador_de_TUG-2 : PROCESS(clk_tug3_local)
BEGIN
    IF (clk_tug3_local = '1') AND (clk_tug3_local'EVENT) THEN
        CASE cont_7_tug2 IS
            WHEN 0 => TUG3 <= TUG2_1;
            WHEN 1 => TUG3 <= TUG2_2;
            WHEN 2 => TUG3 <= TUG2_3;
            WHEN 3 => TUG3 <= TUG2_4;
            WHEN 4 => TUG3 <= TUG2_5;
            WHEN 5 => TUG3 <= TUG2_6;
            WHEN 6 => TUG3 <= TUG2_7;
            WHEN OTHERS => NULL;
        END CASE;
```

```
    END IF;
END PROCESS;
```

Este bloco é modelado apenas pelo processo acima. Sua função é multiplexar os sinais de *TUG2_1* a *TUG2_7* para formar o sinal *TUG3*. A figura 5.92 mostra o diagrama temporal destes sinais.

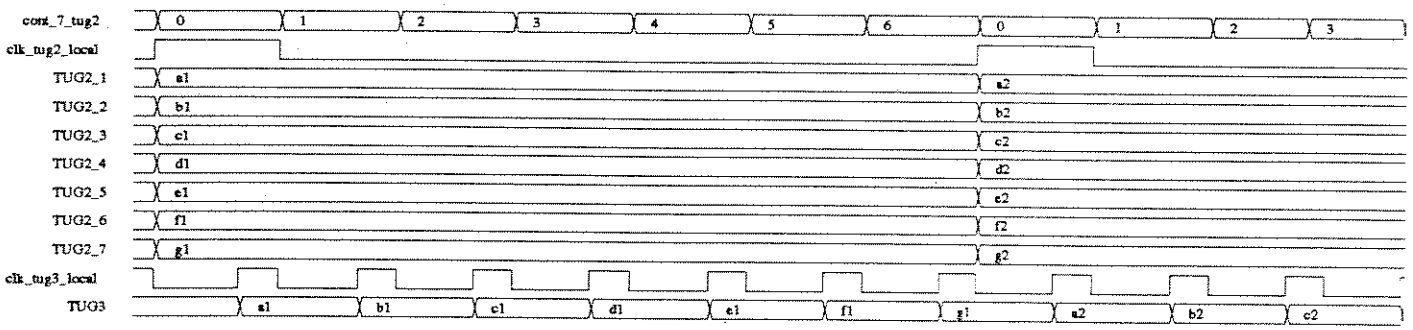


Figura 5.92: Diagrama temporal dos sinais *TUG2_1* a *TUG2_7*.

5.7.6 Multiplexador de TUG-3's

Modelo em VHDL:

```
Multiplexador_de_TUG-3 : PROCESS(T0)
BEGIN
    IF (T0 = '1') AND (T0'EVENT) THEN
        CASE cont_3_tug3 IS
            WHEN 0 => carga_vc4 <= TUG3_1;
            WHEN 1 => carga_vc4 <= TUG3_2;
            WHEN 2 => carga_vc4 <= TUG3_3;
            WHEN OTHERS => NULL;
        END CASE;
    END IF;
END PROCESS;
```

Este bloco é modelado apenas pelo processo acima. Sua função é multiplexar os sinais de *TUG3_1*, *TUG3_2* e *TUG3_3* para formar o sinal *carga_vc4* que representa a carga útil do quadro VC-4. A figura 5.93 mostra o diagrama temporal destes sinais.

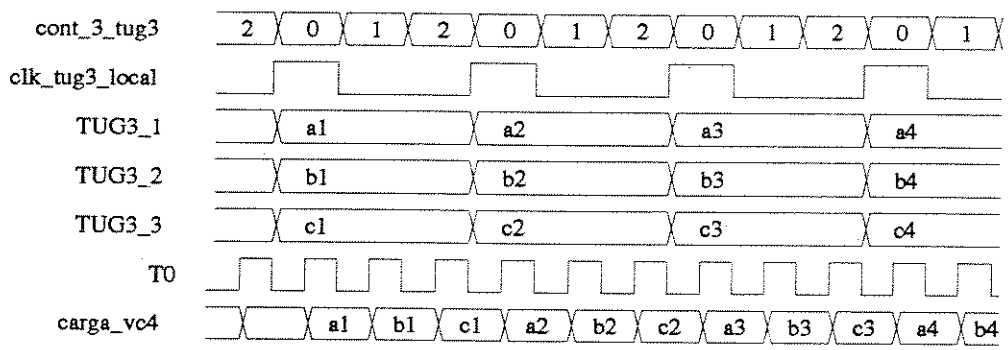


Figura 5.93: Diagrama temporal dos sinais *TUG3_1*, *TUG3_2* e *TUG3_3*.

5.8 Inseridor de Byte H4

O sinal *carga_vc4* obtido do Multiplexador de TUG-3's não possui ainda os bytes de POH do quadro VC-4. A inserção destes bytes fica a cargo do bloco HPT_T (Terminação de Rotas de Alta Ordem na Transmissão), mas como foi dito, este bloco não é modelado. Porém como foi visto no bloco Gerador de Sinais de Controle do VC-4, o byte H4 que faz parte do POH do VC-4 é necessário para a geração do sinal *ind_subq*. Por isso modela-se aqui o bloco Iseridor de Byte H4 com a função de gerar e inserir o byte H4 no sinal *carga_vc4*.

Modelo em VHDL:

```
ger_chave_H4 : PROCESS(T0)
BEGIN
   IF (T0 = '1') AND (T0'EVENT) THEN
      IF (linha_local = 8) AND (coluna_local = 9) THEN
         chave_H4 <= '1';
      ELSE
         chave_H4 <= '0';
      END IF;
   END IF;
END PROCESS;


ger_H4: PROCESS(ind_subq_local)
BEGIN
   CASE ind_subq_local IS
      WHEN 0 => H4 <= "00000000";
      WHEN 1 => H4 <= "00000001";
      WHEN 2 => H4 <= "00000010";
      WHEN 3 => H4 <= "00000011";
      WHEN OTHERS => NULL;
   END CASE;
END PROCESS;


VC4 <= H4 WHEN chave_H4 = '1' ELSE
   carga_vc4;
```

O processo *ger_chave_H4* gera o sinal *chave_H4* que vai a nível lógico 1 quando a posição de inserção do byte H4 ocorrer nos sinais *linha_local* e *coluna_local*.

O processo *ger_H4* gera o byte H4 a partir do sinal *ind_subq_local*.

Finalmente, o último comando deste modelo, insere o byte H4 no sinal *carga_vc4* para gerar o sinal *AU-4*.

5.9 Adaptação de Seção Multiplex na Transmissão (MSA_T)

A função de Adaptação de Seção Multiplex na Transmissão está detalhada através do diagrama de blocos na figura 5.94. A seguir analisa-se cada um dos blocos da figura 5.94 através de diagramas temporais de seus sinais de entrada e de saída e dos comandos VHDL que os modelam.

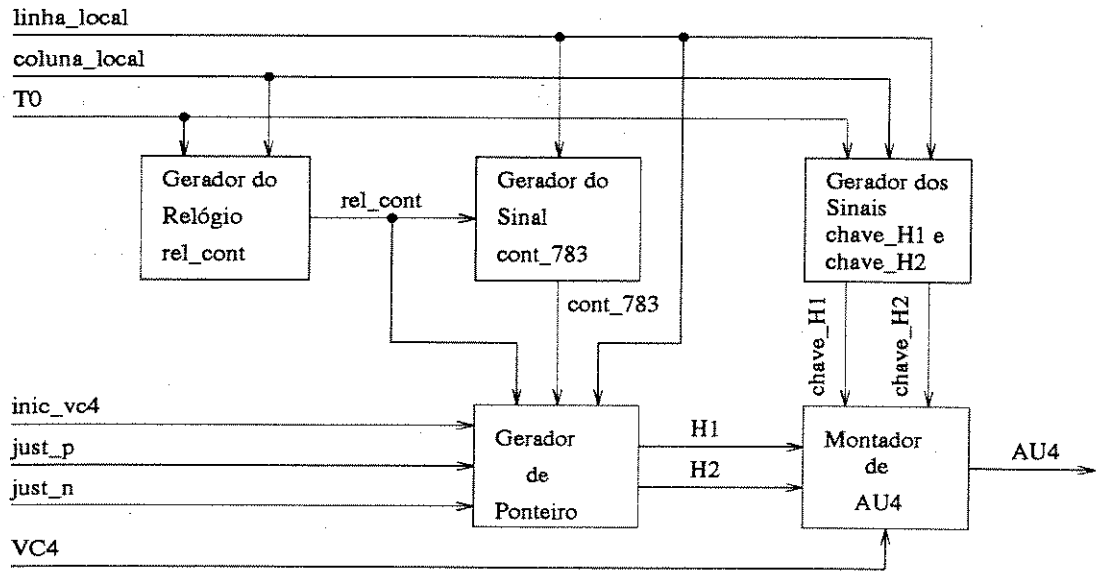


Figura 5.94: Detalhamento do bloco SA_T.

5.9.1 Gerador do Relógio *rel_cont*

O processo abaixo modela este bloco.

```
rel_cont <= T0 WHEN coluna_local <= 8 ELSE
            '0';
```

A figura 5.95 ilustra o funcionamento deste bloco. Sua função é gerar o relógio *rel_cont* que é o relógio *T0* com buracos nas 9 primeiras colunas de todas as 9 linhas do quadro STM-1 local.

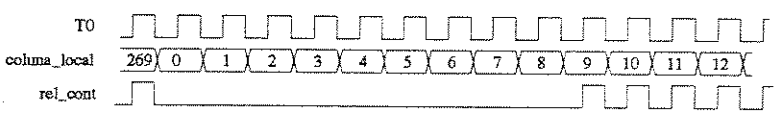


Figura 5.95: Diagrama temporal do sinal *rel_cont*.

5.9.2 Gerador do Sinal *cont_783*

Este bloco é modelado pelo processo abaixo. O diagrama temporal da figura 5.96 ilustra o seu funcionamento.

```
contador : PROCESS(rel_cont, linha_local)
BEGIN
    IF (linha_local = 3) AND (linha_local'EVENT) THEN
       cont_3 <= 0;
       cont_783 <= 0;
    END IF;
    IF (rel_cont = '0') AND (rel_cont'EVENT) THEN
       IF (cont_3 = 2) THEN
          cont_3 <= 0;
          cont_783 <= cont_783 + 1;
       ELSE
          cont_3 <= cont_3 + 1;
    END IF;
END PROCESS;
```

Observe que o sinal *cont_783* identifica cada um dos 783 grupos de 3 bytes que comportam os bytes do quadro VC-4.

O sinal *cont_3* é interno ao bloco gerador do sinal *cont_783*.

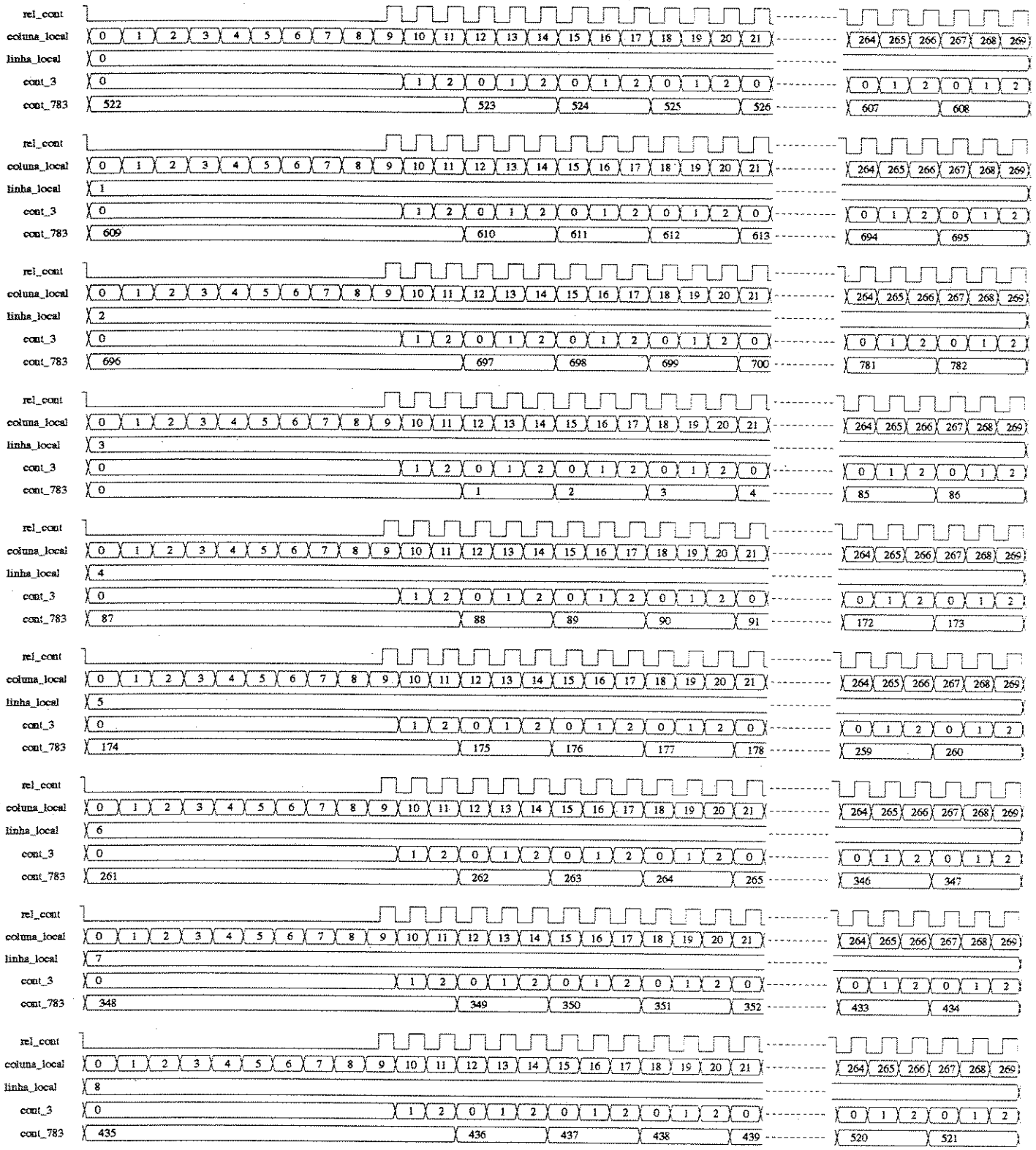


Figura 5.96: Diagrama temporal dos sinais *cont_3* e *cont_783*.

5.9.3 Gerador de Ponteiro

Este bloco é modelado pelo processo abaixo.

```
gerador_ponteiro: PROCESS(rel_esc_normal, linha_local)
BEGIN
   IF (linha_local = 3) AND (linha_local'EVENT) THEN
      IF (NDF = '1') THEN
         indica_NDF(H1, H2, v_pont);
      ELSE
         IF (just_p = '1') THEN
            inverte_bits_I(H1, H2);
            IF (v_pont = 782) THEN
               v_pont <= 0;
            ELSE
               v_pont <= v_pont + 1;
            END IF;
         ELSIF (just_n = '1') THEN
            inverte_bits_D(H1, H2);
            IF (v_pont = 0) THEN
               v_pont <= 782;
            ELSE
               v_pont <= v_pont - 1;
            END IF;
         ELSE
            monta_ponteiro(H1, H2, v_pont);
         END IF;
      END IF;
   END IF;
   IF (rel_esc_normal = '0') AND (rel_esc_normal'EVENT) THEN
      IF (inic_vc4 = '1') THEN
         IF (cont_783 = v_pont) THEN
            NDF <= '0';
         ELSE
            NDF <= '1';
            v_pont <= cont_783;
         END IF;
      END IF;
   END IF;
END PROCESS;
```

Este bloco atua em dois instantes do quadro STM-1 local. O primeiro está ilustrado nas figuras 5.97 e 5.98.

A figura 5.97 pode ser assim interpretada: nas bordas de descida do relógio *rel_cont* o sinal *inic_vc4* é verificado. Caso o sinal *inic_vc4* esteja em nível lógico 1 indicando que o byte presente no sinal *VC4* é o primeiro do quadro VC-4, então é verificado se o sinal *v_pont* se iguala ao sinal *cont_783*. Caso eles sejam iguais, significa que o sinal *v_pont* está indicando a posição correta de início do quadro VC-4 dentro do quadro STM-1 local. Se os sinais *v_pont* e *cont_783* não forem iguais, significa que o sinal *v_pont* não está indicando a posição correta do primeiro byte do VC-4. Neste caso o sinal *v_pont* assume o valor do sinal *cont_783* e o sinal *NDF* (interno ao bloco Gerador de Ponteiro) assume o valor lógico 1, como mostrado na figura 5.98.

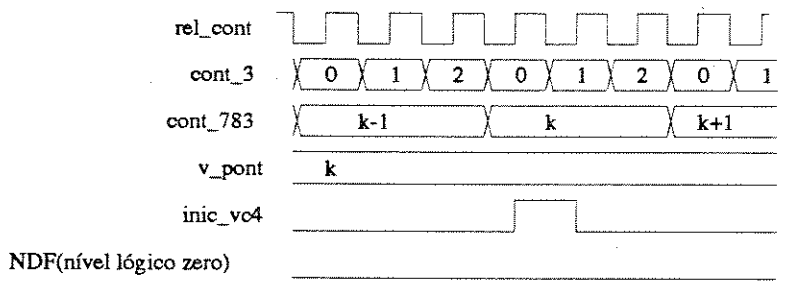


Figura 5.97: Gerador de Ponteiro detectando posição correta do início do VC-4.

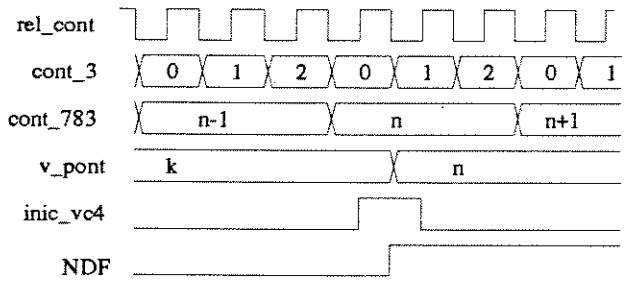


Figura 5.98: Gerador de Ponteiro detectando NDF (novos dados).

O segundo instante de atuação do bloco Gerador de Ponteiro ocorre quando o sinal *linha_local* muda seu valor de 2 para 3. Isto está mostrado nas figuras 5.99, 5.100 e 5.101.

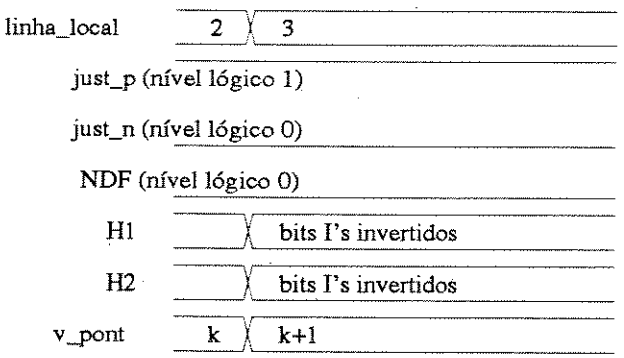


Figura 5.99: Gerador de Ponteiro operando durante justificação positiva.

A figura 5.99 ilustra o funcionamento do bloco Gerador de Ponteiro quando ele recebe uma indicação de justificação positiva (sinal *just_p* = '1'). Observe que nesta ocasião os sinais *H1* e *H2* têm os seus bits I's invertidos.

A figura 5.100 ilustra o funcionamento do bloco Gerador de Ponteiro quando ele recebe uma indicação de justificação negativa (sinal *just_n* = '1'). Observe que nesta ocasião os sinais *H1* e *H2* têm os seus bits D's invertidos.

A figura 5.101 ilustra a funcionamento do Gerador de Ponteiro depois da detecção de uma indicação de novos dados (sinal *NDF* = '1'). Observe que nesta ocasião os sinais *H1* e *H2* sinalizam a existência de novos dados e indicam um novo valor de ponteiro.

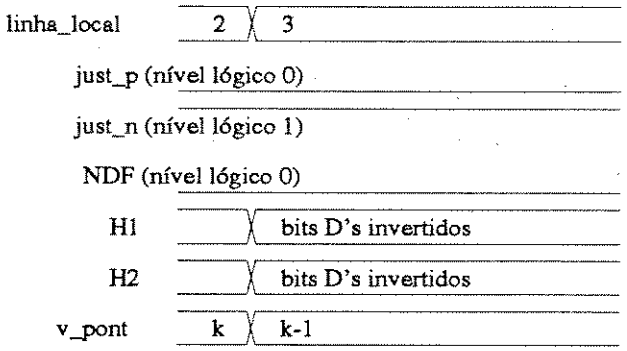


Figura 5.100: Gerador de Ponteiro operando durante justificação negativa.

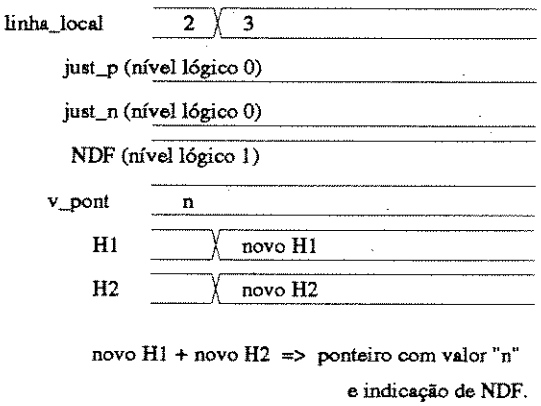


Figura 5.101: Gerador de Ponteiro operando depois da detecção de NDF.

5.9.4 Gerador dos Sinais *chave_H1* e *chave_H2*

Modelo em VHDL:

```
chaves_para_H1_e_H2 : PROCESS(T0)
BEGIN
   IF (T0 = '1') AND (T0'EVENT) THEN
      IF (linha_local = 3) THEN
         CASE coluna_local IS
            WHEN 0 => chave_H1 <= '1';
            WHEN 3 => chave_H2 <= '1';
            WHEN OTHERS => chave_H1 <= '0'; chave_H2 <= '0';
         END CASE;
      END IF;
   END IF;
END PROCESS;
```

A figura 5.102 ilustra o funcionamento deste bloco.

Estes sinais servem para inserir os bytes H1 e H2 do ponteiro no quadro STM-1 local.

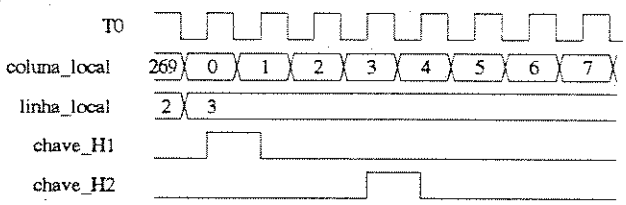


Figura 5.102: Diagrama temporal dos sinais *chave_H1* e *chave_H2*.

5.9.5 Montador de AU-4

Modelo em VHDL:

```
AU4 <= H1 WHEN chave_H1 = '1' ELSE
    H2 WHEN chave_H2 = '1' ELSE
    VC4;
```

A figura 5.103 ilustra o funcionamento deste bloco. Ele simplesmente insere o byte H1 na posição *linha_local = 3, coluna_local = 0* e o byte H2 na posição *linha_local = 3, coluna_local = 3*.

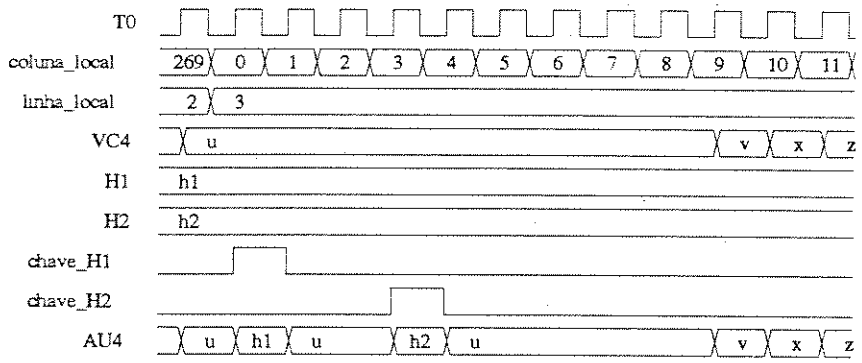


Figura 5.103: Montagem do sinal *AU4*.

5.10 Bloco MUX

Não se desenvolve um modelo para o bloco MUX. Em [1] mostra-se a estrutura de um circuito integrado que recebe um sinal STM-N na forma paralela (8 linhas) e o coloca na forma serial (1 linha).

5.11 Simulação e Validação dos Modelos

Modelos em linguagem VHDL devem ser associados a declaraçôes de entidades para que possam ser simulados. Apresenta-se a seguir as declarações de entidades para todos os blocos que foram modelados.

5.11.1 Entidade *sar* (Adaptação de Seção Multiplex na Recepção)

Observe que a figura 5.104 ilustra exatamente a declaração da entidade abaixo, ou seja, mostra os sinais de entrada e de saída para o componente *sar*.

Veja que o sinal *fase* é um dos sinais de saída desta entidade. Embora este sinal seja somente de uso interno, aqui ele está sendo colocado para fora afim de ser monitorado, como será visto à frente.

Note que alguns dos sinais têm o modo *BUFFER*. No capítulo 4 foram estudados os modos *IN* e *OUT*, que são usados para declarar sinais de entrada e de saída da entidade respectivamente. O modo *BUFFER* é usado para se exteriorizar sinais com funções internas à entidade. Por exemplo, o relógio *clk_vc4* que é utilizado internamente como relógio de leitura da memória elástica, é também colocado pra fora da entidade *sar* por ter sido declarado com modo *BUFFER*.

Declaração de Entidade em VHDL:

```
ENTITY sar IS
    PORT(STM1: IN BIT_VECTOR(1 TO 8);
       linha, coluna, linha_local, coluna_local: IN INTEGER:=0;
       T0, T1: IN BIT;
       just_p, just_n, inic_vc4, clk_vc4: BUFFER BIT;
       fase: BUFFER INTEGER :=0;
       VC4: OUT BIT_VECTOR(1 TO 8));
END sar;
```

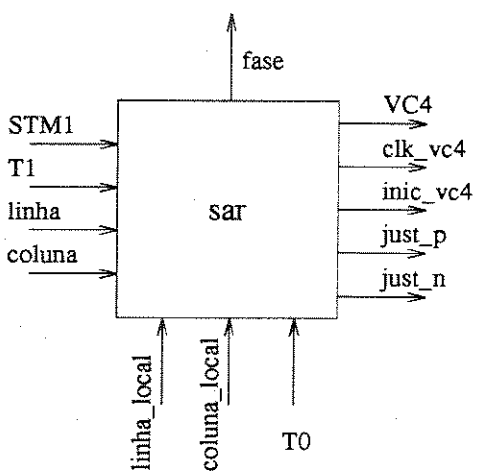


Figura 5.104: Ilustração do componente *sar*.

5.11.2 Entidade *control_vc4* (Gerador dos Sinais de Controle do Quadro VC4)

Declaração de Entidade em VHDL:

```
ENTITY control_vc4 IS
    PORT(VC4: IN BIT_VECTOR(1 TO 8);
        inic_vc4, clk_vc4: IN BIT;
        linha_vc4, coluna_vc4: BUFFER INTEGER;
        ind_subq: OUT INTEGER);
END control_vc4;
```

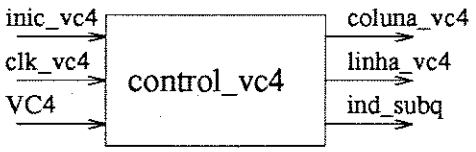


Figura 5.105: Ilustração do componente *control_vc4*.

5.11.3 Entidade *dmult_vc4* (Demultiplexador de VC4)

Declaração de Entidade em VHDL:

```
ENTITY dmult_vc4 IS
    PORT(VC4: IN BIT_VECTOR(1 TO 8);
        clk_vc4, inic_vc4: IN BIT;
        coluna_vc4: IN INTEGER;
        TUG3_1, TUG3_2, TUG3_3: OUT BIT_VECTOR(1 TO 8);
clk_tug3_1, clk_tug3_2, clk_tug3_3: BUFFER BIT);
END dmult_vc4;
```

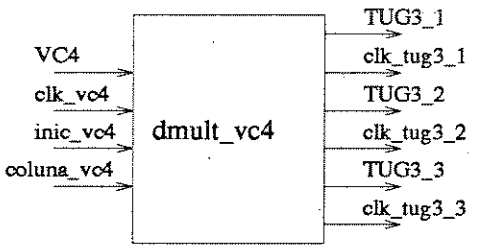


Figura 5.106: Ilustração do componente *dmult_vc4*.

5.11.4 Entidade *dmult_tug3* (Demultiplexador de TUG-3)

Declaração de Entidade em VHDL:

```
ENTITY dmult_tug3 IS
    PORT(TUG3: IN BIT_VECTOR(1 TO 8);
      coluna_vc4: IN INTEGER;
      clk_tug3: IN BIT;
      TUG2_1, TUG2_2, TUG2_3, TUG2_4, TUG2_5, TUG2_6,
      TUG2_7: OUT BIT_VECTOR(1 TO 8);
      clk_tug2_1, clk_tug2_2, clk_tug2_3, clk_tug2_4,
      clk_tug2_5, clk_tug2_6, clk_tug2_7: BUFFER BIT);
END dmult_tug3;
```

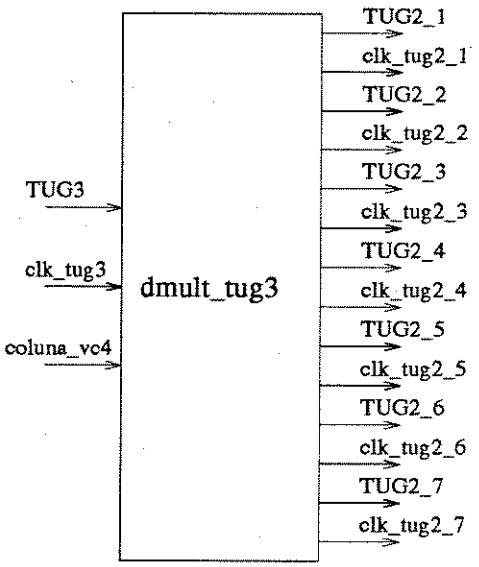


Figura 5.107: Ilustração do componente *dmult_tug3*.

5.11.5 Entidade *dmult_tug2* (Demultiplexador de TUG-2)

Declaração de Entidade em VHDL:

```
ENTITY dmult_tug2 IS
    PORT(clk_tug2, inic_vc4: IN BIT;
      clk_tu12_1, clk_tu12_2, clk_tu12_3: OUT BIT);
END dmult_tug2;
```

5.11.6 Entidade *adapt_vc12_r* (Adaptador de VC12 na Recepção.)

Veja que o sinal *fase* é um dos sinais de saída desta entidade. Embora este sinal seja somente de uso interno, aqui ele está sendo colocado para fora afim de ser monitorado, como será visto à frente.

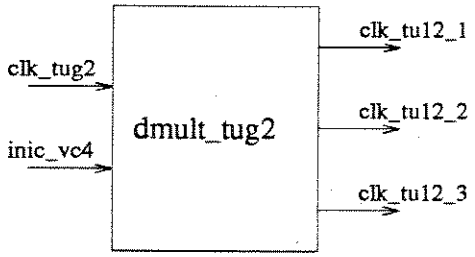


Figura 5.108: Ilustração do componente *dmult_tug2*.

Declaração de Entidade em VHDL:

```
ENTITY adapt_vc12_r IS
   PORT(TUG2: IN BIT_VECTOR(1 TO 8);
      ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4: IN INTEGER:=0;
      clk_tu12, clk_tu12_local: IN BIT;
      just_p, just_n, inic_vc12, clk_vc12: BUFFER BIT;
      fase: BUFFER INTEGER:=0;
      VC12: OUT BIT_VECTOR(1 TO 8));
END adapt_vc12_r;
```

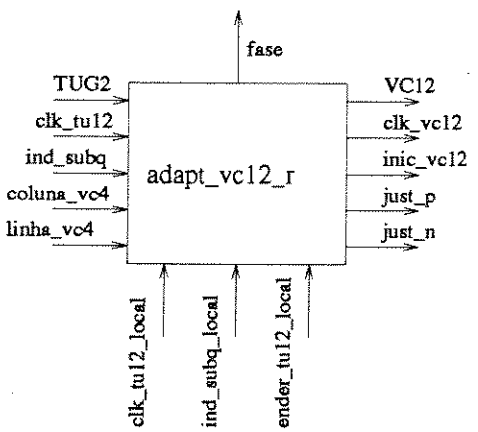


Figura 5.109: Ilustração do componente *adapt_vc12_r*.

5.11.7 Entidade *adapt_vc12_t* (Adaptador de VC12 na Transmissão.)

Declaração de Entidade em VHDL:

```
ENTITY adapt_vc12_t IS
   PORT(VC12: IN BIT_VECTOR(1 TO 8);
      ind_subq_local, ender_tu12_local: IN INTEGER:=0;
      clk_tu12_local, inic_vc12, just_p, just_n: IN BIT;
      TU12: OUT BIT_VECTOR(1 TO 8));
END adapt_vc12_t;
```

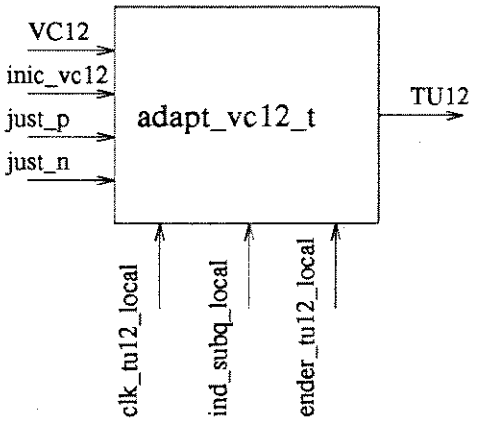


Figura 5.110: Ilustração do componente *adapt_vc12_t.*

5.11.8 Entidade *mult_tu12* (Multiplexador de TU-12.)

Declaração de Entidade em VHDL:

```
ENTITY mult_tu12 IS
    PORT(TU12_1, TU12_2, TU12_3: IN BIT_VECTOR(1 TO 8);
      cont_3_tu12: IN INTEGER;
      clk_tug2_local: IN BIT;
      TUG2: OUT BIT_VECTOR(1 TO 8));
END mult_tu12;
```

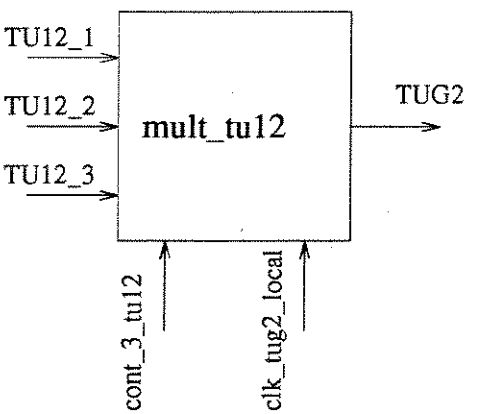


Figura 5.111: Ilustração do componente *mult_tu12.*

5.11.9 Entidade *mult_tug2* (Multiplexador de TUG-2.)

Declaração de Entidade em VHDL:

```
ENTITY mult_tug2 IS
   PORT(TUG2_1, TUG2_2, TUG2_3, TUG2_4, TUG2_5, TUG2_6, TUG2_7:
            IN BIT_VECTOR(1 TO 8);
      cont_7_tug2: IN INTEGER;
      clk_tug3_local: IN BIT;
      TUG3: OUT BIT_VECTOR(1 TO 8));
END mult_tug2;
```

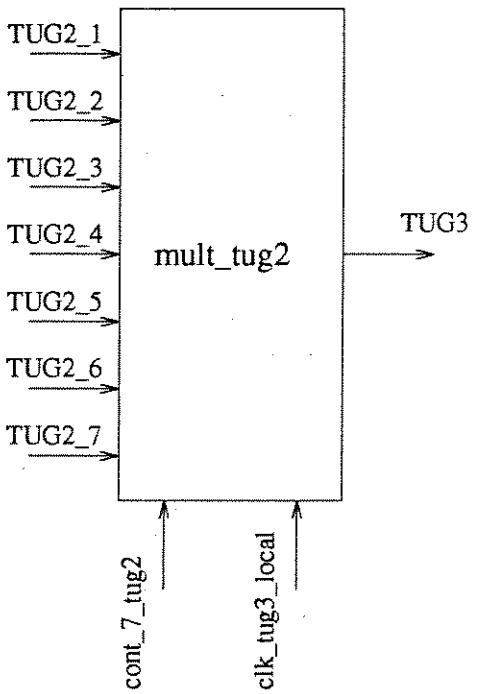


Figura 5.112: Ilustração do componente *mult_tug2*.

5.11.10 Entidade *mult_tug3* (Multiplexador de TUG-3.)

Declaração de Entidade em VHDL:

```
ENTITY mult_tug3 IS
   PORT (TUG3_1, TUG3_2, TUG3_3: IN BIT_VECTOR(1 TO 8);
      cont_3_tug3: IN INTEGER;
      T0: IN BIT;
      carga_vc4: OUT BIT_VECTOR(1 TO 8));
END mult_tug3;
```

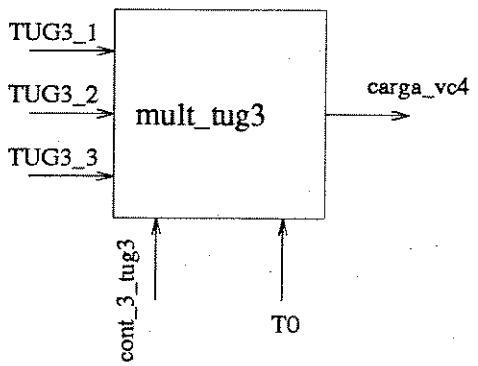


Figura 5.113: Ilustração do componente *mult_tug3*.

5.11.11 Entidade *insere_H4* (Gerador e Inseridor de Byte H4.)

Declaração de Entidade em VHDL:

```
ENTITY insere_h4 IS
   PORT(carga_vc4: IN BIT_VECTOR(1 TO 8);
      linha_local, coluna_local, ind_subq_local: IN INTEGER:=0;
      T0: IN BIT;
      VC4: OUT BIT_VECTOR(1 TO 8));
END insere_h4;
```

5.11.12 Entidade *sat* (Adaptação de Seção Multiplex na Transmissão.)

Declaração de Entidade em VHDL:

```
ENTITY sat IS
   PORT(VC4: IN BIT_VECTOR(1 TO 8);
      linha_local, coluna_local: IN INTEGER:=0;
      T0, inic_vc4, just_p, just_n: IN BIT;
      AU4: OUT BIT_VECTOR(1 TO 8));
END sat;
```

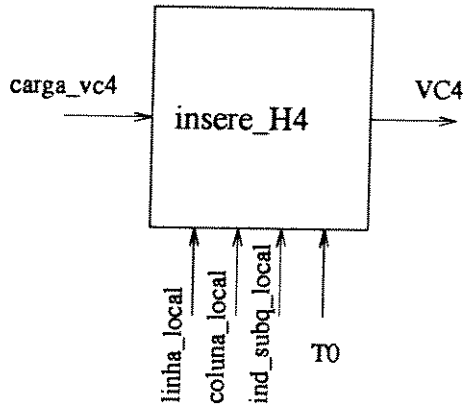


Figura 5.114: Ilustração do componente *insere_h4*.

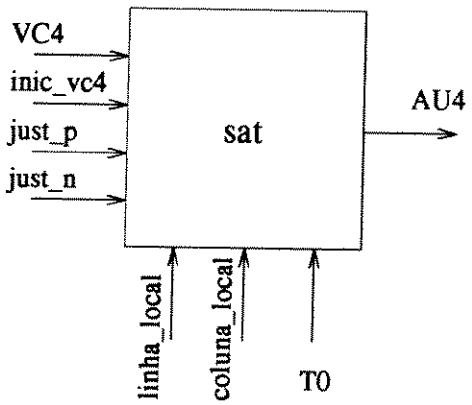


Figura 5.115: Ilustração do componente *sat*.

5.11.13 Entidade *control_local* (Gerador de Sinais de Controle Local.)

Declaração de Entidade em VHDL:

```
ENTITY control_local IS
   PORT(T0: IN BIT;
      linha_local, coluna_local, ind_subq_local, cont_3_tug3,
      cont_7_tug2, cont_3_tu12, ender_tu12_local: BUFFER INTEGER:=0;
      inic_vc4_local, clk_tug3_local, clk_tug2_local, clk_tu12_local, inic_vc12_local: BUFFER BIT);
END control_local;
```

5.12 Arranjos de Blocos SDH

Para simular os modelos desenvolvidos utiliza-se software Mentor Graphics e o equipamento estação de trabalho SUN Sparc Staion 2, disponíveis no Laboratório de Computação Aplicada a Engenharia Elétrica (LCAEe - UNICAMP). Este software permite visualizar a evolução dos sinais a medida que a simulação avança.

Na figura 5.117 tem-se uma janela de tempo de uma simulação mostrando o resultado de um teste

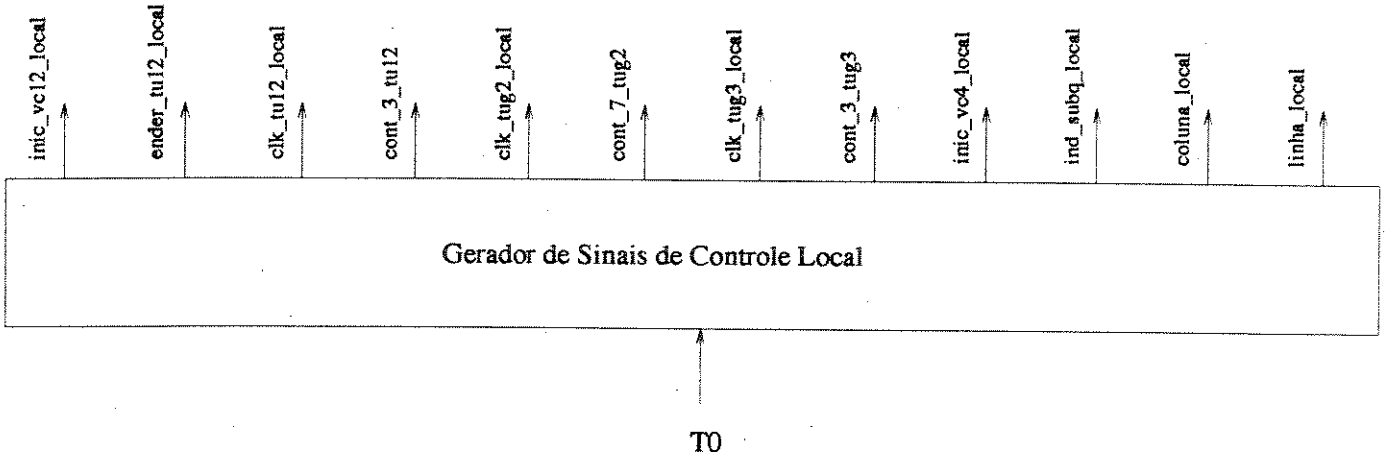


Figura 5.116: Ilustração do componente *control_local.*

para se verificar o comportamento do modelo na ocasião em que um quadro STM-1 é recebido com justificação negativa (bits D's invertidos). Pode-se notar que o modelo processa convenientemente esta informação através da sinalização do sinal *inv_D.* Vê-se também que o relógio de escrita em memória elástica *rel_esc* apresenta pulsos durante os bytes de oportunidade de justificação negativa.

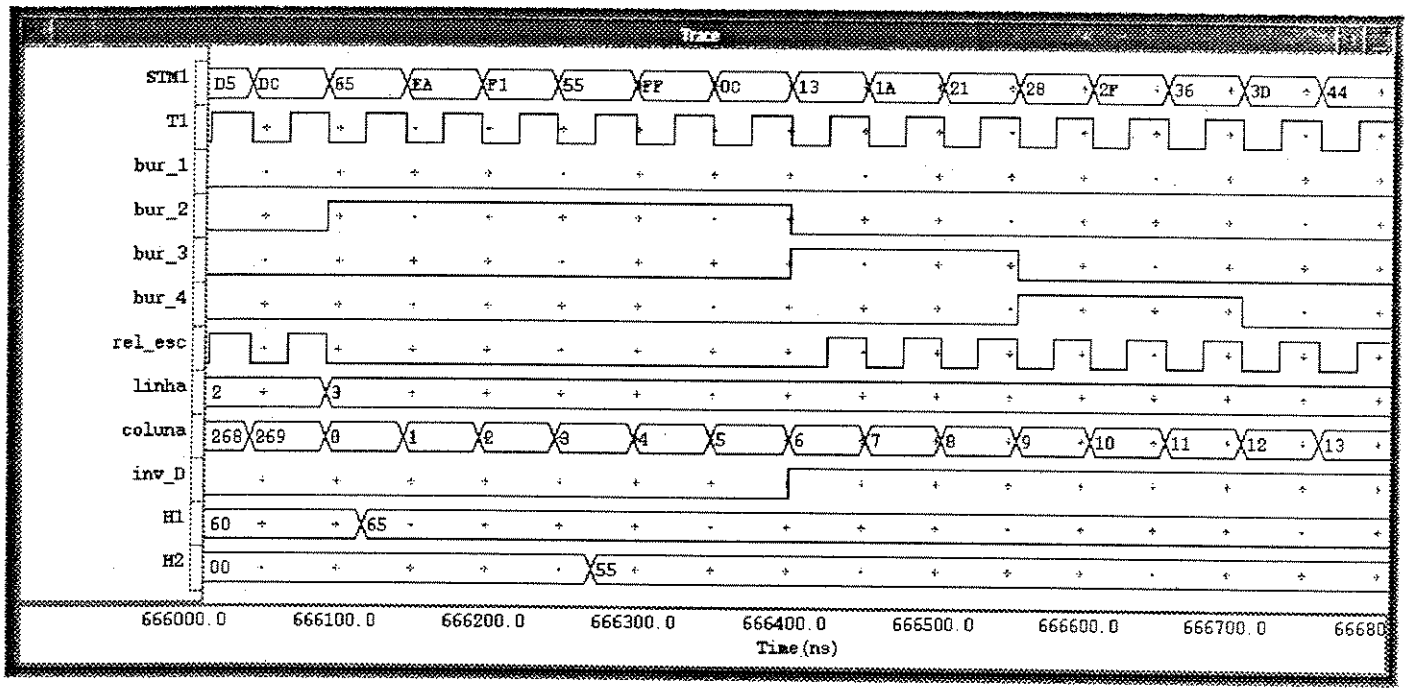


Figura 5.117: Exemplo de simulação.

Este tipo de visualização dos resultados de simulação é importante para se verificar detalhes nas formas de onda dos sinais gerados, mas não é adequado para se verificar o funcionamento dos modelos ao longo de um período de tempo maior. A seguir apresenta-se arranjos desenvolvidos com a finalidade de se testar o funcionamento conjunto dos modelos. Apresenta-se também alguns blocos desenvolvidos para se registrar alguns eventos mais significativos durante a simulação.

As entidades que acabaram de ser apresentadas são interfaceáveis. Por exemplo, as saídas do bloco *demult_vc4* são as entradas para três blocos *dmult_tug3.* Portanto, interfaceando-se as várias entidades pode-se montar modelos com funções de equipamentos da Hierarquia Digital Síncrona. Por isso o conjunto de blocos deve trabalhar de forma harmoniosa, já que o mal funcionamento de um

ou mais blocos acarreta o mal funcionamento de toda a estrutura.

Pode-se testar conjuntamente todos os blocos (entidades) a partir dos 4 arranjos discutidos a seguir.

Arranjo A

A figura 5.118 ilustra o arranjo A.

O objetivo deste arranjo é testar a funcionalidade dos blocos SDH quando eles são arranjados para compor a parte de transmissão de um equipamento terminal de linha SDH. Neste arranjo 63 containers VC-12 são multiplexados para compor um quadro AU-4.

Arranjo B

A figura 5.119 ilustra o arranjo B.

O objetivo deste arranjo é testar a funcionalidade dos blocos SDH quando eles são arranjados para compor um equipamento "Add-Drop" SDH de baixa ordem. Neste arranjo o sinal STM-1 é demultiplexado para se obter 63 containers VC-12, qualquer um deles poderia ser retirado e substituido por outro montado localmente ou ser novamente multiplexado na mesma posição ou em uma posição diferente daquela que ocupava no quadro VC-4 recebido. Os 63 containers VC-12 são então multiplexados para compor um quadro AU-4.

Arranjo C

A figura 5.120 ilustra o arranjo C.

O objetivo deste arranjo é testar a funcionalidade dos blocos SDH quando eles são arranjados para compor um equipamento "Cross-Connect" de alta ordem. Neste arranjo o sinal STM-1 é demultiplexado para se obter 1 quadro VC-4. Juntamente com este quadro VC-4, outros quadros VC-4 poderiam ser obtidos a partir de outros sinais STM-1 com origens em nós diferentes da rede. O equipamento deixaria todos os quadros VC-4 em condições de serem retirados e substituidos por outros montados localmente ou serem novamente multiplexados em um sinal AU-4 para serem retransmitidos.

Arranjo D

A figura 5.121 ilustra o arranjo D.

O objetivo deste arranjo é testar a funcionalidade dos blocos SDH quando eles são arranjados para compor a parte de recepção de um equipamento terminal de linha SDH. Neste arranjo um sinal STM-1 é demultiplexado para se obter 63 containers VC-12.

5.13 Declarações de Entidades para os Arranjos de Blocos SDH

Tem-se interesse de testar os modelos de blocos SDH não só ao nível de equipamento mas também ao nível de rede. Para isto declara-se como entidades os arranjos A, B, C e D. Pode-se então considerá-los como nós de uma rede.

5.13.1 Entidade *equip_a* (para o arranjo A)

Declaração de Entidade em VHDL:

```
ENTITY equip_a IS
   PORT(VC12_01, VC12_02, VC12_03, VC12_04, VC12_05, VC12_06, VC12_07,
      VC12_08, VC12_09, VC12_10, VC12_11, VC12_12, VC12_13, VC12_14,
      VC12_15, VC12_16, VC12_17, VC12_18, VC12_19, VC12_20, VC12_21,
      VC12_22, VC12_23, VC12_24, VC12_25, VC12_26, VC12_27, VC12_28,
      VC12_29, VC12_30, VC12_31, VC12_32, VC12_33, VC12_34, VC12_35,
      VC12_36, VC12_37, VC12_38, VC12_39, VC12_40, VC12_41, VC12_42,
      VC12_43, VC12_44, VC12_45, VC12_46, VC12_47, VC12_48, VC12_49,
      VC12_50, VC12_51, VC12_52, VC12_53, VC12_54, VC12_55, VC12_56,
      VC12_57, VC12_58, VC12_59, VC12_60, VC12_61, VC12_62, VC12_63
                  : IN BIT_VECTOR(1 TO 8);
      Ta : IN BIT;
      inic_vc12_local_a, clk_tu12_local_a: BUFFER BIT;
      coluna_local_a, linha_local_a, ender_tu12_local_a: BUFFER INTEGER:=0;
      AU4_a: OUT BIT_VECTOR(1 TO 8));
END equip_a;
```

5.13.2 Entidade *equip_b* (para o arranjo B)

Declaração de Entidade em VHDL:

```
ENTITY equip_b IS
   PORT(STM1_b: IN BIT_VECTOR(1 TO 8);
      Tb, T1_b : IN BIT;
      coluna_b, linha_b :IN INTEGER:= 0;
      coluna_local_b, linha_local_b, fase_b: BUFFER INTEGER:=0;
      au4_just_p_b, au4_just_n_b: BUFFER BIT;
      AU4_b: OUT BIT_VECTOR(1 TO 8));
END equip_b;
```

5.13.3 Entidade *equip_c* (para o arranjo C)

Declaração de Entidade em VHDL:

```
ENTITY equip_c IS
   PORT(STM1_c: IN BIT_VECTOR(1 TO 8);
      Tc, T1_c : IN BIT;
      coluna_c, linha_c :IN INTEGER:= 0;
      coluna_local_c, linha_local_c, fase_c: BUFFER INTEGER:=0;
      au4_just_p_c, au4_just_n_c: BUFFER BIT;
      AU4_c: OUT BIT_VECTOR(1 TO 8));
END equip_c;
```

5.13.4 Entidade *equip_d* (para o arranjo D)

Declaração de Entidade em VHDL:

```
ENTITY equip_d IS
   PORT(STM1_d: IN BIT_VECTOR(1 TO 8);
      Td, T1_d : IN BIT;
      coluna_d, linha_d :IN INTEGER:= 0;
      inic_vc12_01, inic_vc12_02, inic_vc12_03, inic_vc12_04, inic_vc12_05, inic_vc12_06, inic_vc12_07,
      inic_vc12_08, inic_vc12_09, inic_vc12_10, inic_vc12_11, inic_vc12_12, inic_vc12_13, inic_vc12_14,
      inic_vc12_15, inic_vc12_16, inic_vc12_17, inic_vc12_18, inic_vc12_19, inic_vc12_20, inic_vc12_21,
      inic_vc12_22, inic_vc12_23, inic_vc12_24, inic_vc12_25, inic_vc12_26, inic_vc12_27, inic_vc12_28,
      inic_vc12_29, inic_vc12_30, inic_vc12_31, inic_vc12_32, inic_vc12_33, inic_vc12_34, inic_vc12_35,
      inic_vc12_36, inic_vc12_37, inic_vc12_38, inic_vc12_39, inic_vc12_40, inic_vc12_41, inic_vc12_42,
      inic_vc12_43, inic_vc12_44, inic_vc12_45, inic_vc12_46, inic_vc12_47, inic_vc12_48, inic_vc12_49,
      inic_vc12_50, inic_vc12_51, inic_vc12_52, inic_vc12_53, inic_vc12_54, inic_vc12_55, inic_vc12_56,
      inic_vc12_57, inic_vc12_58, inic_vc12_59, inic_vc12_60, inic_vc12_61, inic_vc12_62, inic_vc12_63,
      clk_vc12_01, clk_vc12_02, clk_vc12_03, clk_vc12_04, clk_vc12_05, clk_vc12_06, clk_vc12_07,
      clk_vc12_08, clk_vc12_09, clk_vc12_10, clk_vc12_11, clk_vc12_12, clk_vc12_13, clk_vc12_14,
      clk_vc12_15, clk_vc12_16, clk_vc12_17, clk_vc12_18, clk_vc12_19, clk_vc12_20, clk_vc12_21,
      clk_vc12_22, clk_vc12_23, clk_vc12_24, clk_vc12_25, clk_vc12_26, clk_vc12_27, clk_vc12_28,
      clk_vc12_29, clk_vc12_30, clk_vc12_31, clk_vc12_32, clk_vc12_33, clk_vc12_34, clk_vc12_35,
      clk_vc12_36, clk_vc12_37, clk_vc12_38, clk_vc12_39, clk_vc12_40, clk_vc12_41, clk_vc12_42,
      clk_vc12_43, clk_vc12_44, clk_vc12_45, clk_vc12_46, clk_vc12_47, clk_vc12_48, clk_vc12_49,
      clk_vc12_50, clk_vc12_51, clk_vc12_52, clk_vc12_53, clk_vc12_54, clk_vc12_55, clk_vc12_56,
      clk_vc12_57, clk_vc12_58, clk_vc12_59, clk_vc12_60, clk_vc12_61, clk_vc12_62, clk_vc12_63,
      au4_just_p_d, au4_just_n_d : BUFFER BIT;
      fase_01, fase_02, fase_03, fase_04, fase_05, fase_06, fase_07,
      fase_08, fase_09, fase_10, fase_11, fase_12, fase_13, fase_14,
      fase_15, fase_16, fase_17, fase_18, fase_19, fase_20, fase_21,
      fase_22, fase_23, fase_24, fase_25, fase_26, fase_27, fase_28,
      fase_29, fase_30, fase_31, fase_32, fase_33, fase_34, fase_35,
      fase_36, fase_37, fase_38, fase_39, fase_40, fase_41, fase_42,
      fase_43, fase_44, fase_45, fase_46, fase_47, fase_48, fase_49,
      fase_50, fase_51, fase_52, fase_53, fase_54, fase_55, fase_56,
      fase_57, fase_58, fase_59, fase_60, fase_61, fase_62, fase_63, fase_d: BUFFER INTEGER:=0;
      VC12_01, VC12_02, VC12_03, VC12_04, VC12_05, VC12_06, VC12_07,
      VC12_08, VC12_09, VC12_10, VC12_11, VC12_12, VC12_13, VC12_14,
      VC12_15, VC12_16, VC12_17, VC12_18, VC12_19, VC12_20, VC12_21,
      VC12_22, VC12_23, VC12_24, VC12_25, VC12_26, VC12_27, VC12_28,
      VC12_29, VC12_30, VC12_31, VC12_32, VC12_33, VC12_34, VC12_35,
      VC12_36, VC12_37, VC12_38, VC12_39, VC12_40, VC12_41, VC12_42,
      VC12_43, VC12_44, VC12_45, VC12_46, VC12_47, VC12_48, VC12_49,
      VC12_50, VC12_51, VC12_52, VC12_53, VC12_54, VC12_55, VC12_56,
      VC12_57, VC12_58, VC12_59, VC12_60, VC12_61, VC12_62, VC12_63
            : OUT BIT_VECTOR(1 TO 8));
END equip_d;
```

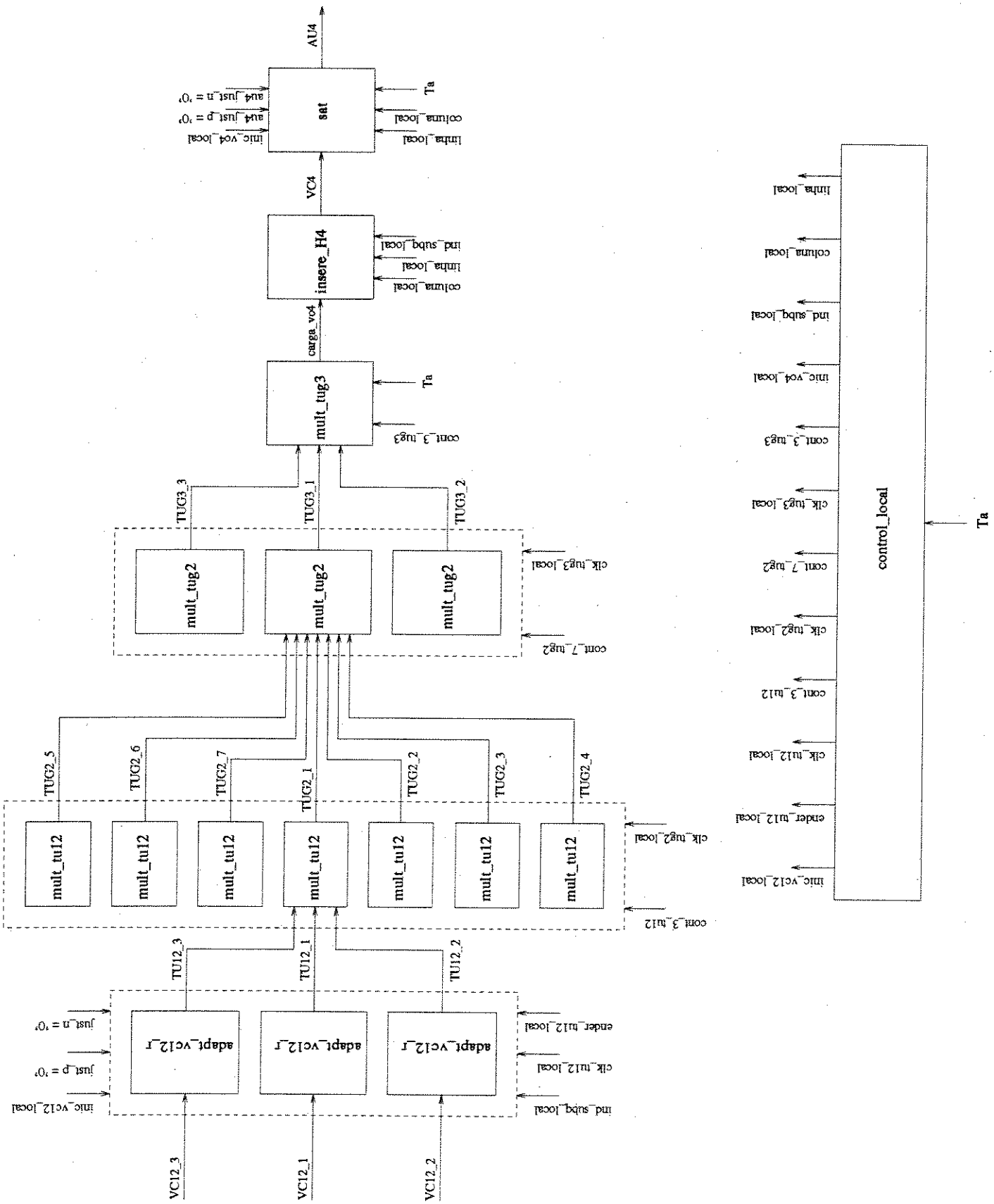


Figura 5.118: Arranjo A.

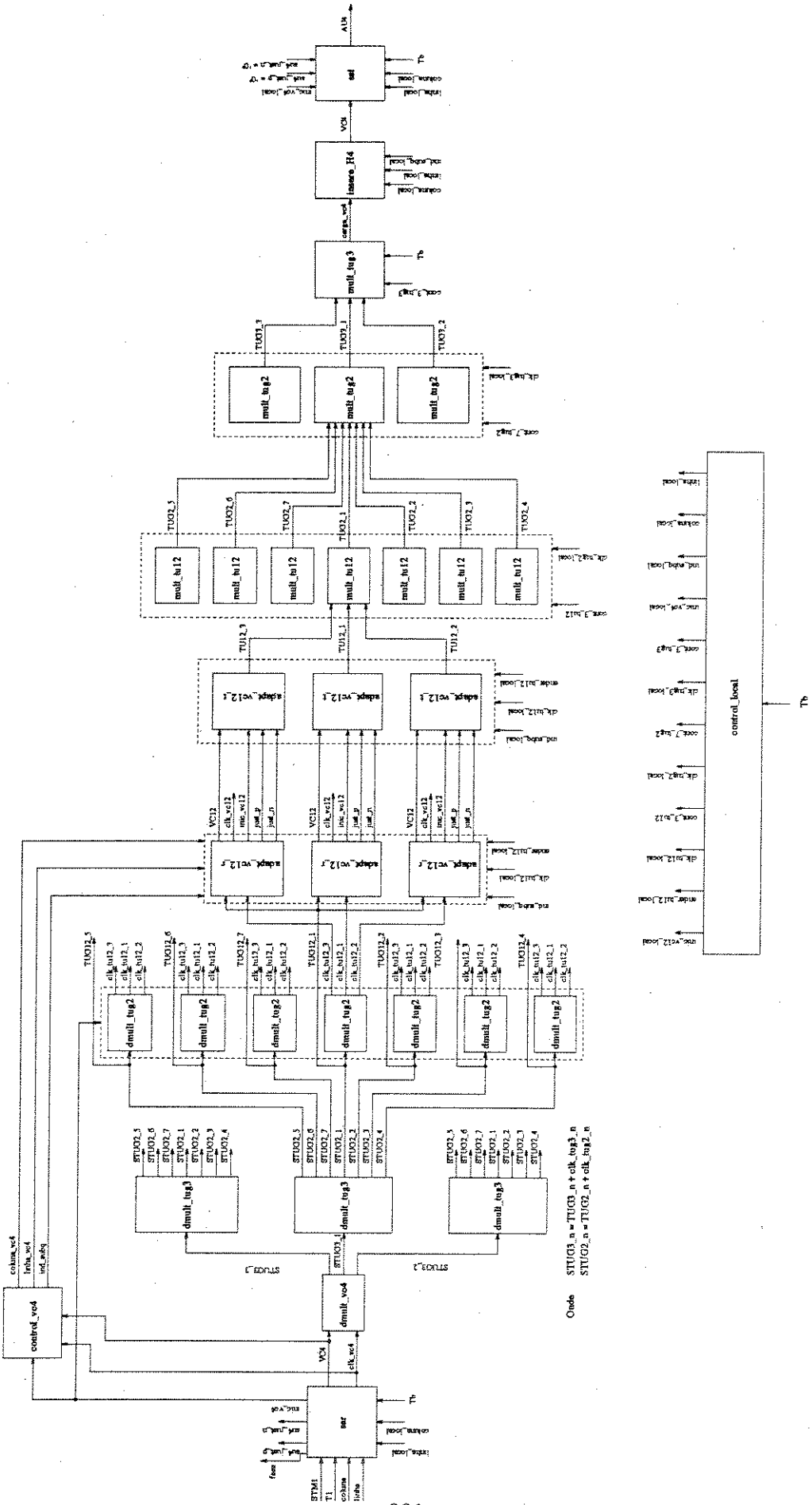


Figura 5.119: Arranjo B.

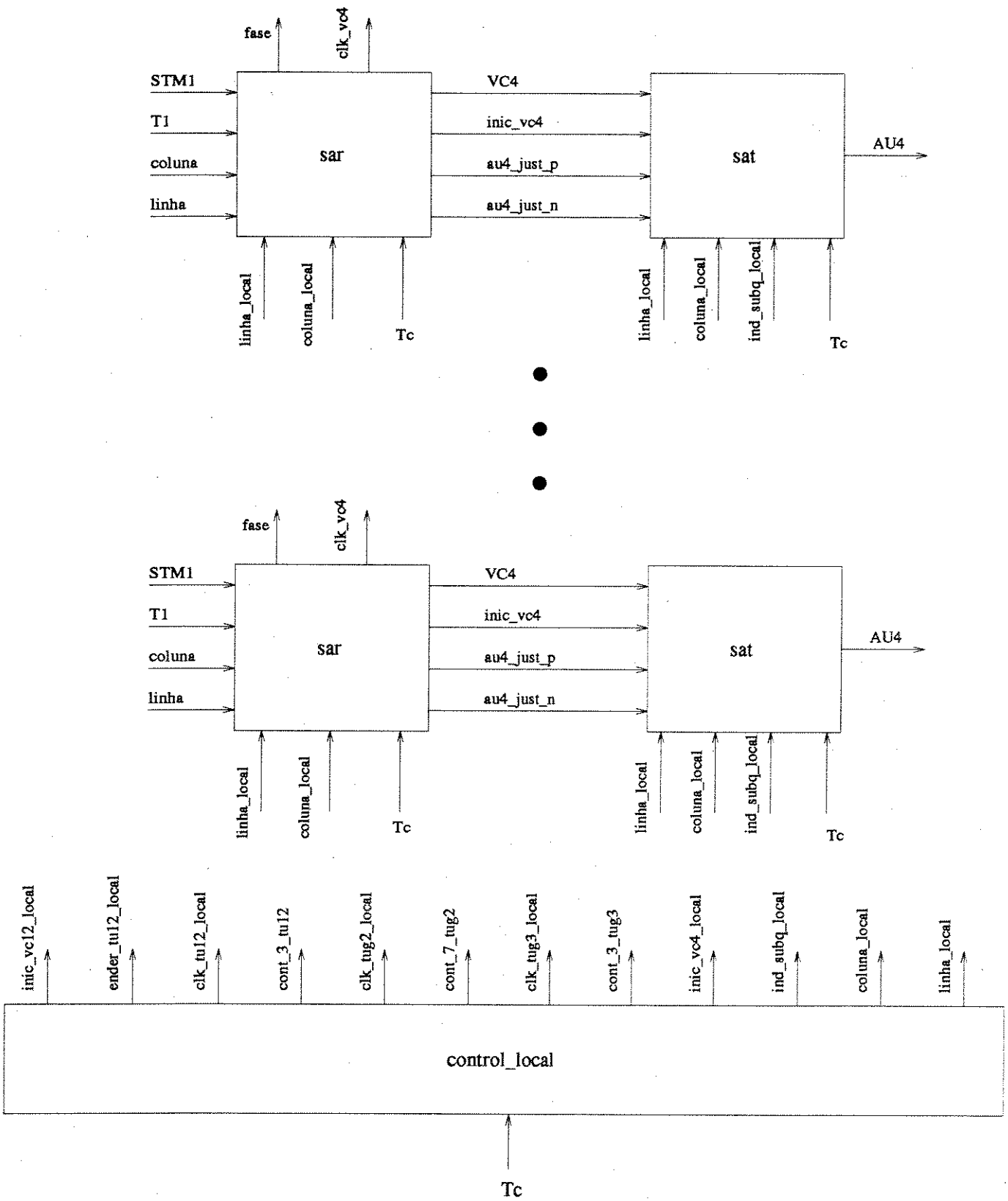


Figura 5.120: Arranjo C.

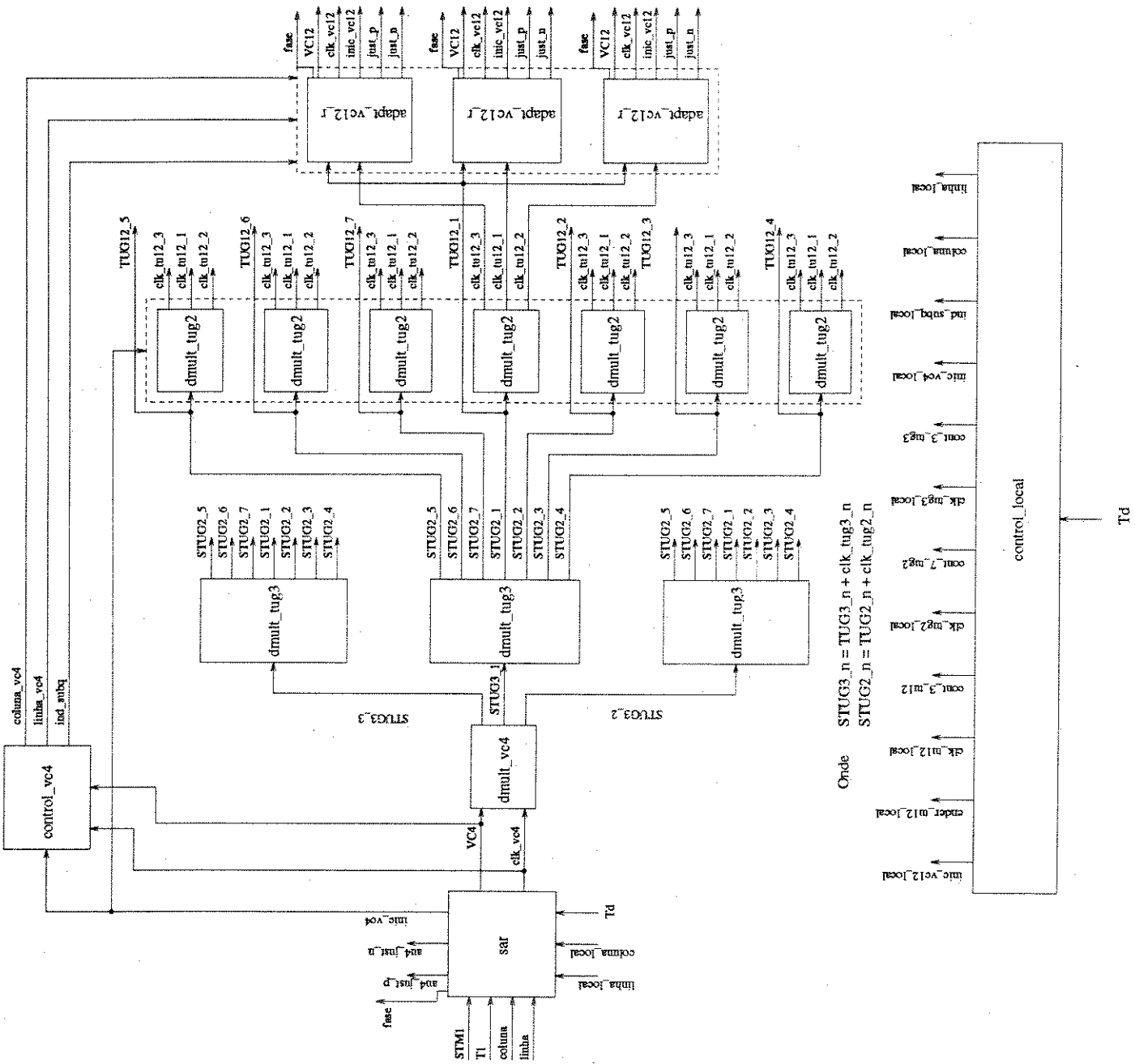


Figura 5.121: Arranjo D.

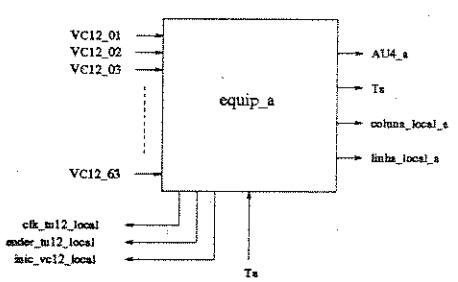


Figura 5.122: Ilustração do Equipamento *equip_a*.

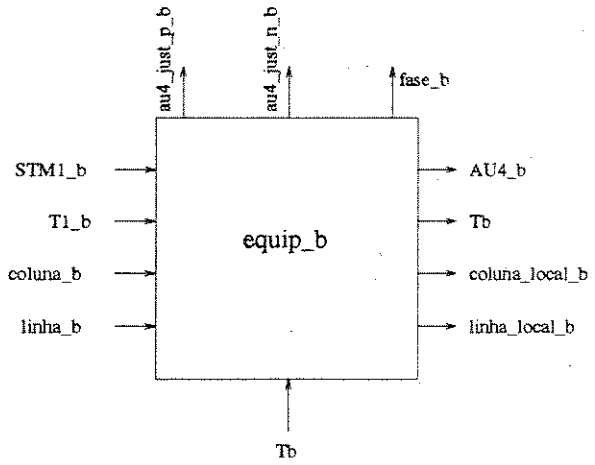


Figura 5.123: Ilustração do Equipamento *equip_b*.

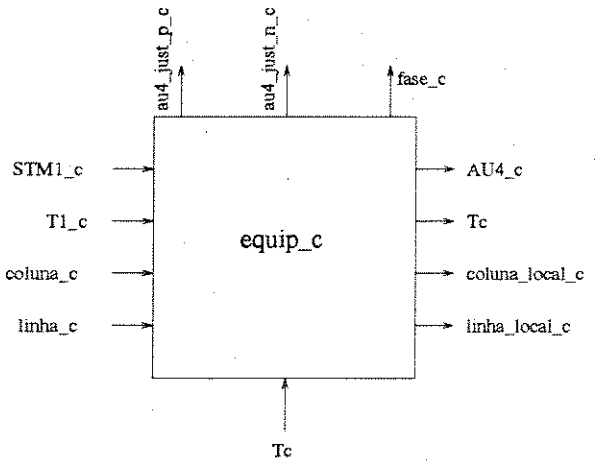


Figura 5.124: Ilustração do Equipamento *equip_c*.

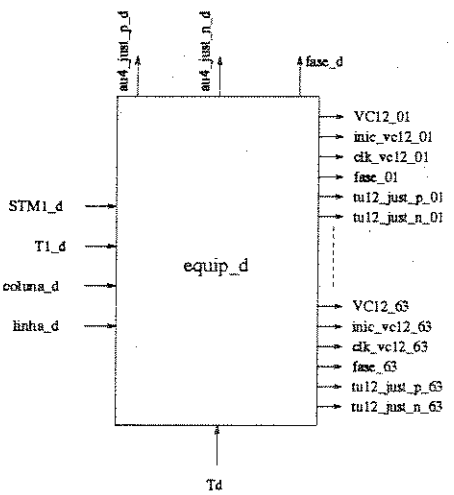


Figura 5.125: Ilustração do Equipamento *equip_d*.

5.14 Simulação dos Modelos ao Nível de Rede

A figura 5.126 mostra uma montagem para simulação dos modelos ao nível de rede.

A seguir descreve-se modelos em linguagem VHDL para os blocos *Gerador dos Sinais VC12*, *Simulador de Meio Físico*, *Registrador de Justificações no Quadro AU4*, *Registrador de Justificações no Quadro TU12* e *Registrador de Dados não Esperados em VC12*.

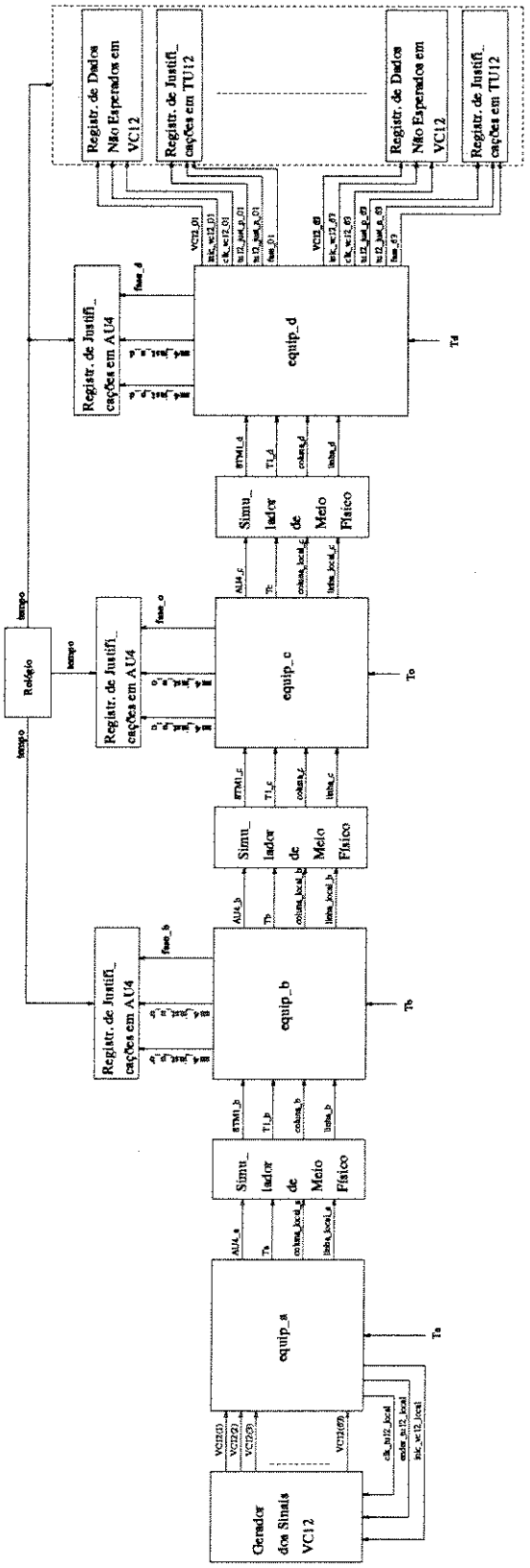


Figura 5.126: Montagem para Simulação dos Modelos ao Nível de Rede.

5.14.1 Gerador dos Sinais VC12

Este bloco gera sequências de bytes para os sinais VC12. Estas mesmas sequências devem ser recebidas nos blocos Registradores de Dados não Esperados que considera como sendo um dado não esperado qualquer diferença entre as sequências recebidas e as geradas.

Modelo em VHDL:

```
ger_sinais_vc12:PROCESS(clk_tu12_local)
BEGIN
   IF (clk_tu12_local = '1') AND (clk_tu12_local'EVENT) AND (ender_tu12_local /= 0) THEN
      IF (inic_vc12_local = '1') THEN
         FOR i IN 1 TO 63 LOOP
            VC12(i) <= inteiro_pra_byte(i);
         END LOOP;
      ELSE
         FOR i IN 62 DOWNTO 1 LOOP
            VC12(i) <= VC12(i+1) AFTER 2 ns;
         END LOOP;
         VC12(63) <= VC12(1) AFTER 2 ns;
      END IF;
   END IF;
END PROCESS;
```

O diagrama temporal da figura 5.127 ilustra o funcionamento deste bloco.

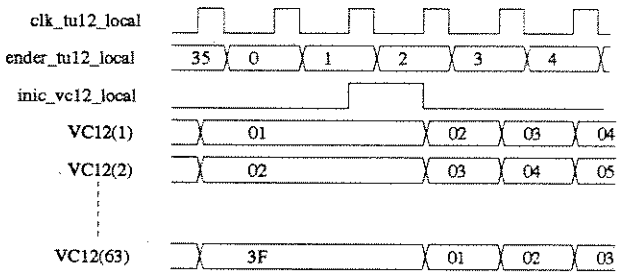


Figura 5.127: Diagrama temporal dos sinais *VC12*.

Observe que o sinal *VC12(1)* recebe o valor "00000001" (01 em hexadecimal), bem como o sinal *VC12(i)* (i = 2, 3, ..., 63) recebe o valor i, quando o sinal *inic_vc12_local* (recebido do bloco *equip_a*) vai a nível lógico 1.

Durante as bordas de subida do relógio *clk_tu12_local* (recebido do bloco *equip_a*) para as quais o sinal *ender_tu12_local* (recebido do bloco *equip_a*) é diferente de zero, o sinal *VC12(63)* recebe o valor *VC12(1)*, bem como o sinal *VC12(i)* (i = 1, 2, 3, ..., 62) recebe o valor *VC12(i+1)*.

5.14.2 Simulador de Meio Físico

Vê-se pela figura 5.126 que entre os blocos *equip_a* e *equip_b* existe o bloco *Simulador de Meio Físico* que recebe os sinais *AU4_a*, *Ta*, *linha_local_a* e *coluna_local_a* e libera os sinais *STM1_b*, *T1_b*, *linha_b* e *coluna_b*.

Modelo em VHDL:

```
ger_coluna_linha_b: PROCESS(Ta)
BEGIN
   IF (Ta = '1') THEN
      coluna_b <= coluna_local_a;
      linha_b <= linha_local_a;
   END IF;
END PROCESS;
T1_b <= NOT(Ta);
STM1_b <= AU4_a;
```

A figura 5.128 ilustra o funcionamento deste bloco.

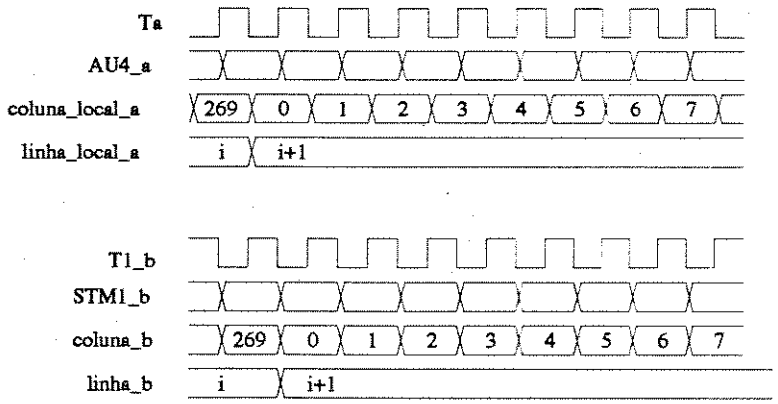


Figura 5.128: Diagrama temporal dos sinais de entrada e de saída do bloco *Simulador de Meio Físico.*

Veja que este simulador de meio físico considera a situação ideal na qual o relógio *Ta* é recuperado como sendo *T1_b* sem nenhum wander ou jitter. Também o sinal *AU4_a* é recebido como sendo *STM1_b* sem nenhum erro de trasmissão. Os sinais *linha_b* e *coluna_b* deveriam ser gerados no bloco *equip_b* a partir da palavra de alinhamento de quadro do sinal *STM1_b*. Isto seria função do bloco *DEMUX* para o qual não se desenvolveu um modelo em VHDL. Por isso aproveita se os sinais *linha_local_a* e *coluna_local_a* para a geração dos sinais *linha_b* e *coluna_b*.

Os demais blocos simuladores de meio físico da figura 5.126 são idênticos a este estudado.

5.14.3 Relógio

Este bloco simplesmente gera o sinal *tempo* a partir do processo abaixo.

```
rel: PROCESS
BEGIN
WAIT FOR 1 ns;
   tempo <= tempo + 1.0e-9;
END PROCESS;
```

O sinal *tempo* é a medida do tempo a partir do início da simulação. Ele é útil para se registrar, com tempo determinado, alguns eventos no simulador.

5.14.4 Registrador de Justificações no Quadro AU4

Vê-se pela figura 5.126 que atrelado ao bloco *equip_b* existe o bloco *Registr. de Justificações em AU4* que recebe os sinais *au4_just_p_b*, *au4_just_n_b*, *fase_b* e *tempo* para registrar todas as justificações ocorridas na Adaptação de Seção Multiplex do equipamento *equip_b*. Ele também registra o tempo em que ocorre a justificação e a fase entre o relógio de escrita e o relógio de leitura na memória elástica na hora da justificação.

Modelo em VHDL:

```
registr_justif: PROCESS(au4_just_p_b, au4_just_n_b)
FILE data_out : text IS OUT "../regist_just_b";
VARIABLE out_line: LINE;
VARIABLE cont_p, cont_n: INTEGER:=0;
CONSTANT jp : string:="justificacao positiva ";
CONSTANT jn : string:="justificacao negativa ";
CONSTANT fs : string:=" fase = ";
CONSTANT tp : string:=" tempo = "
BEGIN
   IF (au4_just_p_b = '1') AND (au4_just_p_b'EVENT) THEN
      cont_p := cont_p + 1;
      WRITE(out_line, cont_p);
      WRITE(out_line, jp);
      WRITE(out_line, tp);
      WRITE(out_line, tempo);
      WRITE(out_line, fs);
      WRITE(out_line, fase_b);
      WRITELINE(data_out, out_line);
   END IF;
   IF (au4_just_n_b = '1') AND (au4_just_n_b'EVENT) THEN
      cont_n := cont_n + 1;
      WRITE(out_line, cont_n);
      WRITE(out_line, jn);
      WRITE(out_line, tp);
      WRITE(out_line, tempo);
      WRITE(out_line, fs);
      WRITE(out_line, fase_b);
      WRITELINE(data_out, out_line);
   END IF;
END PROCESS;
```

Durante as bordas de subida do sinal *au4_just_p_b* este processo escreve em uma linha do arquivo *regist_just_b*:

- o número da justificação positiva
- o string " justificacao positiva ".
- o string " tempo = "
- o tempo em que a justificação ocorre.
- o string " fase = ".

- a fase da memória elástica na hora da justificação.

Durante as bordas de subida do sinal *au4_just_n_b* este processo escreve em uma linha do arquivo *regist_just_b*:

- o número da justificação negativa
- o string " justificacao negativa ".
- o string " tempo = "
- o tempo em que a justificação ocorre.
- o string " fase = ".
- a fase da memória elástica na hora da justificação.

Assim, tem-se um relatório de todas as justificações do sinal *AU4_b* acompanhadas dos tempos em que elas ocorrem a partir do início da simulação e da fase da memória elástica na hora da justificação. Como exemplo, mostra-se abaixo as primeiras linhas do arquivo *regist_just_b* gerado.

1 justificacao positiva tempo = 2.778840e-04 fase = 1
2 justificacao positiva tempo = 6.530270e-04 fase = 4
3 justificacao positiva tempo = 1.028170e-03 fase = 7
4 justificacao positiva tempo = 1.403314e-03 fase = 11
1 justificacao negativa tempo = 4.029318e-03 fase = 33
2 justificacao negativa tempo = 4.404461e-03 fase = 33
3 justificacao negativa tempo = 4.779604e-03 fase = 33
4 justificacao negativa tempo = 5.154748e-03 fase = 33
5 justificacao negativa tempo = 5.529891e-03 fase = 33
6 justificacao negativa tempo = 5.905035e-03 fase = 33

Os demais blocos registradores de justificações em AU4 da figura 5.126 são idênticos a este estudado.

5.14.5 Registrador de Justificações no Quadro TU12

Vê-se pela figura 5.126 que para cada sinal VC12 que sai do bloco *equip_d* existe um bloco *Registr. de Justificações em TU12*. Para o sinal VC12_01 em específico, este bloco recebe os sinais *tu12_just_p_01* e *tu12_just_n_01* e registra todas as justificações sinalizadas por estes sinais. O modelo para este bloco é idêntico ao mostrado para o registrador de justificações em AU4.

Os demais blocos registradores de justificações em tu12 da figura 5.126 são idênticos a este estudado.

5.14.6 Registrador de Dados não Esperados em VC12

Vê-se pela figura 5.126 que para cada sinal VC12 que sai do bloco *equip_d* existe um bloco *Registrador de Dados não Esperados em VC12*. Para o sinal VC12_01 em específico, este bloco recebe os sinais

VC12_01, *inic_vc12_01* e *clk_vc12_01* e registra todos os dados recebidos em VC12_01 que não estão de acordo com a sequência de bytes injetada no equipamento *equip_a* para o sinal VC12(1).

Modelo em VHDL:

```
reg_dados_nao_esp_01: PROCESS(clk_vc12_01)
FILE data_out : text IS OUT "../regist_01";
VARIABLE out_line: LINE;
VARIABLE cont: INTEGER:=0;
VARIABLE byte: BIT_VECTOR(1 TO 8);
CONSTANT be :string:=" byte esperado = ";
CONSTANT br :string:=" byte recebido = ";
BEGIN
   IF (clk_vc12_01 = '0') THEN
      IF (inic_vc12_01 = '1') THEN
         cont := 1;
      ELSE
         IF (cont = 63) THEN
            cont := 1;
         ELSE
            cont := cont+1;
         END IF;
      END IF;
      byte := inteiro_pra_byte(cont)
      IF (VC12_01_d /= byte) THEN
        IF (tempo > 4.0e-3) THEN
         WRITE(out_line, tp);
         WRITE(out_line, tempo);
         WRITE(out_line, be);
         WRITE(out_line, byte);
         WRITE(out_line, br);
         WRITE(out_line, VC12_01_d);
         WRITELINE(data_out, out_line);
      END IF;
      END IF;
   END IF;
END PROCESS;
```

Durante as bordas de descida do relógio *clk_vc12_01* é verificado se elas acompanham o primeiro byte do quadro VC12_01 através do sinal *inic_vc12_01*. Então é verificado se o primeiro byte do VC12_01 é igual a 01 (hexadecimal), se o segundo byte do VC12_01 é igual a 02...

Observe que este bloco só inicia seus registros depois de decorridos 4 ms de simulação. Isto é feito para se evitar que os dados recebidos no início da simulação que não são significativos sejam registrados. Estes dados iniciais não têm significado porque os equipamentos levam algus milisegundos para entrarem em funcionamento correto.

Depois dos 4 ms iniciais, caso algum byte recebido não seja igual ao byte esperado então este processo escreve no arquivo *regist_01* o tempo corrido de simulação, o byte esperado e o byte recebido. Assim, tem-se um relatório de todos os bytes recebidos irregularmente no sinal *VC12_01*

Os demais blocos registradores de justificações em tu12 da figura 5.126 são similares. Observe que

para o sinal VC12_02 o primeiro byte esperado é 02 (hexadecimal), para o sinal VC12_03 o primeiro byte esperado é 03...

5.14.7 Colocando os Relógios *Ta, Tb, Tc* e *Td* em Oscilação

Estes relógios são colocados oscilação através das seguintes linhas de comandos:

```
Ta <= NOT(Ta) AFTER 25.72 ns;
Tb <= NOT(Tb) AFTER 25.73 ns;
Tc <= NOT(Tc) AFTER 25.72 ns;
Td <= NOT(Td) AFTER 25.73 ns;
```

Assim, os relógios *Ta* e *Tc* oscilam com período de 51,44 ns e os relógios *Tb* e *Td* oscilam com período de 51,46 ns.

O CCITT recomenda que equipamentos HDS trabalhando no modo "free running" tenham relógios com precisão de $4{,}6\times10^{-6}$. Obviamente os relógios acima estão com precisão muito abaixo disso. Isto foi feito de propósito para permitir que haja justificações mais frequêntes durante a simulação e assim os resultados possam ser analisados mais facilmente.

5.15 Apresentação e Validação dos Resultados de Simulação

. Simulou-se o modelo de rede durante um tempo para se obter um volume de dados suficiente para validar os modelos. A validação é obtida através da comparação entre dados obtidos na simulação e dados calculados ou esperados. Estes dados se referem à quantidade de justificações registradas nos blocos registradores de justificações e aos dados obtidos nos registradores de dados não esperados.

O resultado mais importante a ser observado, é a obtenção de relatórios de dados não esperados vazios, ou seja, as sequências de bytes (sinais VC12) injetadas no equipamento A, após terem sofrido processamentos nos quatro equipamentos, devem ser recebidas idênticas nos registradores de dados não esperados.

Apresenta se a seguir os relatórios obtidos nos registradores de justificações e de dados não esperados.

5.15.1 Relatório do Registrador de Justificações em AU4 do equipamento *equip_b*

Abaixo mostra-se as primeiras e as últimas linhas do relatório de justificações ocorridas no bloco *sar* do equipamento *equip_b*.

1 justificacao positiva tempo = 2.778840e-04 fase = 1

2 justificacao positiva tempo = 6.530270e-04 fase = 4
3 justificacao positiva tempo = 1.028170e-03 fase = 7
4 justificacao positiva tempo = 1.403314e-03 fase = 11
1 justificacao negativa tempo = 4.029318e-03 fase = 33
2 justificacao negativa tempo = 4.404461e-03 fase = 33
3 justificacao negativa tempo = 4.779604e-03 fase = 33
4 justificacao negativa tempo = 5.154748e-03 fase = 33
5 justificacao negativa tempo = 5.529891e-03 fase = 33
6 justificacao negativa tempo = 5.905035e-03 fase = 33
7 justificacao negativa tempo = 6.405226e-03 fase = 33
8 justificacao negativa tempo = 6.780369e-03 fase = 33
9 justificacao negativa tempo = 7.155513e-03 fase = 33
10 justificacao negativa tempo = 7.530656e-03 fase = 33
11 justificacao negativa tempo = 7.905799e-03 fase = 33
12 justificacao negativa tempo = 8.280943e-03 fase = 33

.
.
.

397 justificacao negativa tempo = 1.659662e-01 fase = 33
398 justificacao negativa tempo = 1.663414e-01 fase = 33
399 justificacao negativa tempo = 1.667165e-01 fase = 33
400 justificacao negativa tempo = 1.672167e-01 fase = 33

A figura 5.129 ilustra a evolução do relógio de leitura em relação ao relógio de escrita na memória elástica do bloco *sar* do equipamento *equip_b*. Observe que as primeiras justificações (positivas) ocorrem para forçar a fase a ir rapidamente para a região entre 15 e 33 bytes. Depois de entrar nesta região , como o relógio *Tb* é mais lento que o relógio *Ta*, a fase vai se aproximando aos poucos do valor 33. Quando o valor 33 é atingido, ocorre a primeira justificação negativa, que leva a fase para o valor 30. Novamente a fase se aproxima do valor 33 pois o relógio *Tb* é sempre mais lento que *Ta*. Uma nova justificação negativa ocorre assim que o valor 33 é novamente atingido. Este processo fica se repetindo pelo resto da simulação.

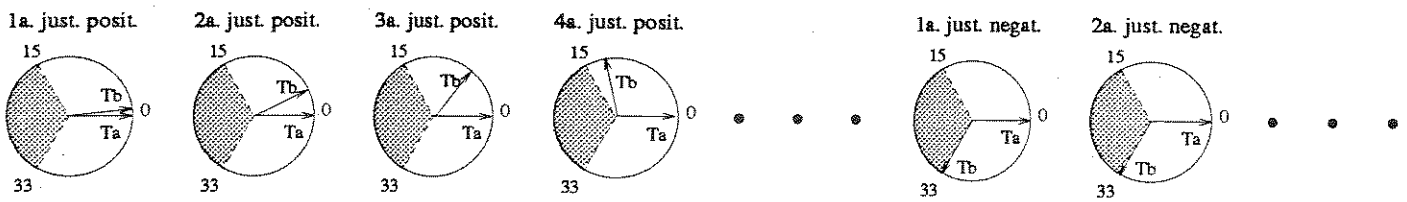


Figura 5.129: Relógio de leitura em memória elástica em relaçao ao relógio de escrita nos intantes das justificações.

O tempo entre a primeira justificação negativa e a última é de:

$1,672167 \times 10^{-1} - 4,029318 \times 10^{-3} = 1,6318738 \times 10^{-1}$

Para um tempo de 51,44 ns (1 período de *Ta*), o relógio *Tb* se atrasa de 0,02 ns. Assim pode-se calcular o atraso de *Tb* em relação a *Ta* para o tempo de $1,6318738 \times 10^{-1}$, entre a primeira justificação negativa e a última.

$51,44ns \longrightarrow 0,02ns$

$1,6318738 \times 10^{-1} \longrightarrow x \Longrightarrow x = 6,344766 \times 10^{-5}$

O tempo x equivale à $\frac{6,344766 \times 10^{-5}}{51,44 \times 10^{-9}} \simeq 1.233$ períodos de relógio, ou à $\frac{1.233}{3} \simeq 411$ justificações.

Na verdade, verificou-se a ocorrência de 399 justificações entre a primeira e a última justificação negativa. Este fato pode ser explicado com a ajuda da figura 5.130. Esta figura mostra o relógio de leitura da memória elástica em relação ao relógio de escrita em instantes distantes no tempo de 125 μs. É feita a consideração de que o relógio de leitura se atrasa em relação ao relógio de escrita de 1 período a cada 125 μs.

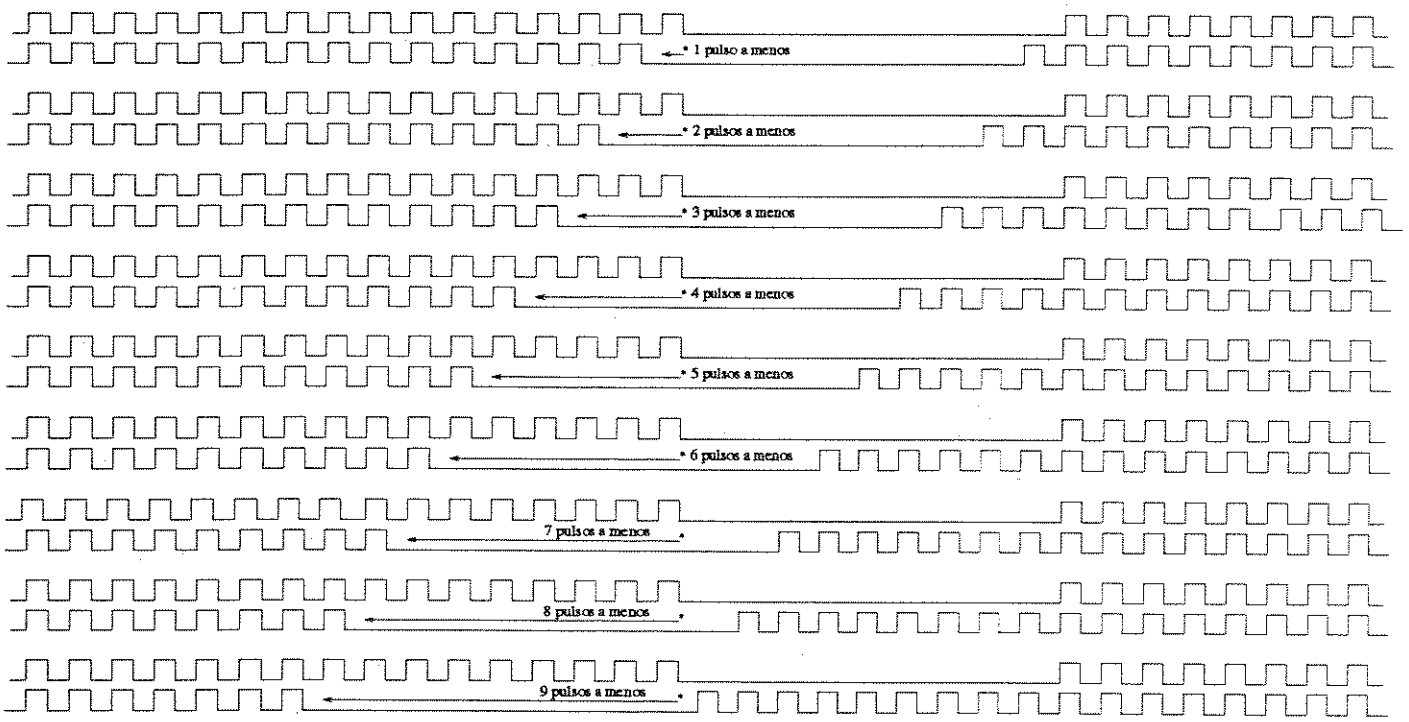


Figura 5.130: Relógio de leitura em memória elástica em relaçao ao relógio de escrita.

O ponto marcado com um "*" é o ponto em que a decisão de se justificar é tomada. Observe que no instante 1 ocorre 1 pulso a menos no relógio de escrita em relaçao ao relógio de leitura até o momento da tomada de decisão. No instante 2 ocorrem 2 pulsos a menos, no instante 3 ocorrem 3 pulsos a menos ...

Portanto, cada vez que a região esburacada do relógio de escrita passa pelo ponto de tomada de decisão, ocorrem 9 pulsos a menos no relógio de escrita em relação ao relógio de leitura, e por isso, deixam de ocorrer 3 justificações negativas.

O tempo entre duas passagens da região esburacada sobre o ponto de tomada de decisão é calculado assim:

$0,02ns \longrightarrow 51,44ns$

$51,44ns \times 270 \longrightarrow x \Longrightarrow x = 3,5721994 \times 10^{-2}$

Dividindo o tempo de simulação por este valor:

$\frac{1.672167 \times 10^{-1}}{3,5721994 \times 10^{-2}} = 4,68$

Portanto, ocorreram 4 passagens da região esburacada sobre o ponto de tomada de decisão e consequentemente deixaram de existir 12 justificações. Exatamente as 12 que estavam faltando.

5.15.2 Relatório do Registrador de Justificações em AU4 do equipamento *equip_c*

Abaixo mostra-se o relatório de justificações ocorridas no bloco *sar* do equipamento *equip_c*.

1 justificacao negativa tempo = 2.777760e-04 fase = 46
2 justificacao negativa tempo = 6.527730e-04 fase = 40
3 justificacao negativa tempo = 1.027771e-03 fase = 34
1 justificacao positiva tempo = 3.152757e-03 fase = 15
2 justificacao positiva tempo = 3.527755e-03 fase = 15
3 justificacao positiva tempo = 4.027752e-03 fase = 15
4 justificacao positiva tempo = 4.402749e-03 fase = 15
5 justificacao positiva tempo = 4.777747e-03 fase = 15
6 justificacao positiva tempo = 5.152744e-03 fase = 15
7 justificacao positiva tempo = 5.527742e-03 fase = 15
8 justificacao positiva tempo = 5.902740e-03 fase = 15
9 justificacao positiva tempo = 6.402736e-03 fase = 15
10 justificacao positiva tempo = 6.777734e-03 fase = 15
11 justificacao positiva tempo = 7.152732e-03 fase = 15
12 justificacao positiva tempo = 7.527729e-03 fase = 15

.
.
.

399 justificacao positiva tempo = 1.660267e-01 fase = 15
400 justificacao positiva tempo = 1.664017e-01 fase = 15
401 justificacao positiva tempo = 1.667767e-01 fase = 15
402 justificacao positiva tempo = 1.671517e-01 fase = 15

A figura 5.131 ilustra a evolução do relógio de leitura em relação ao relógio de escrita na memória elástica do bloco *sar* do equipamento *equip_c*. Observe que as primeiras justificações (negativas) ocorrem para forçar a fase a ir rapidamente para a região entre 15 e 33 bytes. Depois de entrar nesta região, como o relógio *Tc* é mais rápido que o relógio *Tb*, a fase vai se aproximando aos poucos do valor 15. Quando o valor 15 é atingido, ocorre a primeira justificação positiva, que leva a fase para o valor 18. Novamente a fase se aproxima do valor 15 pois o relógio *Tc* é sempre mais rápido que *Tb*. Uma nova justificação positiva ocorre assim que o valor 15 é novamente atingido. Este processo fica se repetindo pelo resto da simulação.

O tempo entre a primeira justificação positiva e a última é de:

$1,671517 \times 10^{-1} - 3,152757 \times 10^{-3} = 1,64 \times 10^{-1}$

Para um tempo de 51,44 ns (1 período de *Tc*), o relógio *Tc* se adianta de 0,02 ns. Assim pode-se calcular o adiantamento de *Tc* em relação a *Tb* para o tempo de $1,64 \times 10^{-1}$.

$51,44ns \longrightarrow 0,02ns$

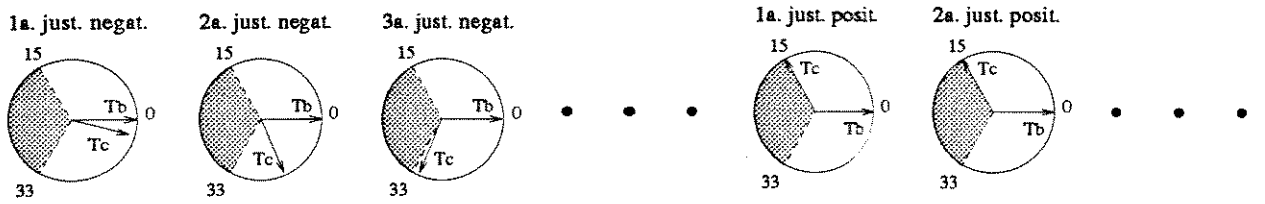


Figura 5.131: Relógio de leitura em memória elástica em relaçao ao relógio de escrita nos intantes das justificações.

$1,64 \times 10^{-1} \longrightarrow x \Longrightarrow x = 6,3763197 \times 10^{-5}$

O tempo x equivale à $\frac{6,3763197 \times 10^{-5}}{51,44 \times 10^{-9}} \simeq 1.239$ períodos de relógio, ou à $\frac{1.239}{3} \simeq 413$ justificações.

Na verdade, verificou-se a ocorrência de 401 justificações entre a primeira e a última justificação positiva. A explicação para este fato é similar à explicação dada para a ocorrência de um menor número de justificações negativas no equipamento *equip_b*.

5.15.3 Relatório do Registrador de Justificações em AU4 do equipamento *equip_d*

Abaixo mostra-se o relatório de justificações ocorridas no bloco *sar* do equipamento *equip_d*.

1 justificacao positiva tempo = 2.778840e-04 fase = 1
2 justificacao positiva tempo = 6.530270e-04 fase = 7
3 justificacao positiva tempo = 1.028170e-03 fase = 13
1 justificacao negativa tempo = 3.153983e-03 fase = 33
2 justificacao negativa tempo = 3.491612e-02 fase = 33
3 justificacao negativa tempo = 3.529127e-02 fase = 33
4 justificacao negativa tempo = 3.579146e-02 fase = 33

Observou-se a ocorrência de apenas as justificações mostradas acima, apesar do relógio *Td* ser mais lento do que o relógio *Tc*. Isto pode se justificado da seguinte maneira. O equipamento *equip_d* recebe do equipamento *equip_c* o sinal *STM1_d* que traz justificações positivas periódicas. Com isso, a taxa de dados recebida pelo equipamento *equip_d* é menor, na verdade esta taxa é igual à taxa de dados enviada pelo equipamento *equip_b* ao equipamento *equip_c*. Uma vez que o relógio *Tc* tem a mesma taxa de *Td*, explica-se portanto a não ocorrência de justificações no bloco *sar* de *equip_d*.

As únicas justificações que foram registradas têm sua causa atribuida ao tempo inicial de ajuste.

5.15.4 Relatórios dos Registradores de Justificações em TU12 do equipamento *equip_d*

Abaixo mostra-se o relatório de justificações ocorridas no bloco *adapt_vc12_r* do equipamento *equip_d*. Especificamente está se mostrando o relatório referente ao primeiro VC12. Os demais 62 relatórios são semelhantes.

1 justificacao negativa tempo = 3.612490e-04 fase = 7

2 justificacao negativa tempo = 1.036507e-02 fase = 6
3 justificacao negativa tempo = 1.886832e-02 fase = 6
4 justificacao negativa tempo = 2.687138e-02 fase = 6
5 justificacao negativa tempo = 3.537463e-02 fase = 6
6 justificacao negativa tempo = 4.587865e-02 fase = 6
7 justificacao negativa tempo = 5.438190e-02 fase = 6
8 justificacao negativa tempo = 6.288515e-02 fase = 6
9 justificacao negativa tempo = 7.088821e-02 fase = 6
10 justificacao negativa tempo = 8.189241e-02 fase = 6
11 justificacao negativa tempo = 8.989547e-02 fase = 6
12 justificacao negativa tempo = 9.839872e-02 fase = 6
13 justificacao negativa tempo = 1.069020e-01 fase = 6
14 justificacao negativa tempo = 1.174060e-01 fase = 6
15 justificacao negativa tempo = 1.259092e-01 fase = 6
16 justificacao negativa tempo = 1.344125e-01 fase = 6
17 justificacao negativa tempo = 1.424155e-01 fase = 6

Estas justificações ocorrem porque o sinal VC12 é montado no equipamento *equip_a* com um relógio derivado do relógio *Ta* e é retirado do equipamento *equip_d* com um relógio derivado do relógio *Td*.

O tempo entre a primeira justificação e a última é de:

$1.424155 \times 10^{-1} - 3.612490 \times 10^{-4} = 1,4205425 \times 10^{-1}$

Para um tempo de 51,44 ns (1 período de *Tc*), o relógio *Tc* se adianta de 0,02 ns. Assim pode-se calcular o adiantamento de *Tb* em relação a *Tb* para o tempo de $1,4205425 \times 10^{-1}$.

$51,44ns \longrightarrow 0,02ns$

$1,4205425 \times 10^{-1} \longrightarrow x \Longrightarrow x = 5,5231 \times 10^{-5}$

O período do relógio *TU12*é de $3,4722 \times 10^{-6}s$. Dividindo o tempo x pelo tempo de 1 período de *TU12*, tem-se $\frac{5,5231 \times 10^{-5}}{3,4722 \times 10^{-6}} \simeq 16$ períodos de relógio do quadro *TU12*, ou $\simeq 16$ justificações.

Realmente verificou-se a ocorrência de 16 justificações neste período.

5.15.5 Relatórios dos Registradores de Dados Não Esperados em VC12

Todos os 63 registradores de dados não esperados em VC12 foram encontrados vazios após a simulação. Portanto as sequências injetadas no equipamento *equip_a* foram retiradas exatas no equipamento *equip_d*.

Observe a importância deste resultado para a validação dos modelos. Os sinais VC12 foram multiplexados no equipamento *equip_a*, sofreram processamentos no equipamento *equip_b*, atravessaram o equipamento *equip_c*, montados em um container VC4 e finalmente foram retirados exatos no equipamento *equip_d*. Porém, este resultado só foi alcançado depois de um extenso trabalho de depuração de erros de lógica nos modelos.

Capítulo 6

Comentários Finais

O uso da linguagem VHDL posibilitou o desenvolvimento e teste de modelos comportamentais de circuitos lógicos para blocos funcionais de equipamentos de Hierarquia Digital Síncrona. Porém, antes de se chegar aos modelos finais, vários erros de lógica tiveram que ser apurados. Isto representou um trabalho e um custo bem menor do que representaria se os modelos não fossem comportamentais, mas estivessem em uma forma mais detalhada, tipo "gate array". O trabalho de depuração dos modelos representou modificações em linhas de comando VHDL. Portanto, foi essencialmente um trabalho de depuração em "software".

Através de simulação, observou-se o funcionamento conjunto dos modelos. Com quatro modelos de equipamentos montados a partir de arranjos dos modelos dos blocos funcionais, verificou-se o comportamento de uma rede de multiplexadores de Hierarquia Digital Síncrona.

Agora que os modelos estão validados mesmo ao nível de uma rede de multiplexadores, eles podem dar origem a três linhas para futuros desenvolvimentos.

- **Primeira linha para futuros desenvolvimentos**

  A primeira linha de desenvolvimento seria o detalhamento dos modelos ao nível de portas lógicas. Uma vez que o funcionamento conjunto dos modelos foi aprovado, pode-se então tratar cada bloco funcional isoladamente. Para este trabalho, o uso da linguagem VHDL continuaria sendo importante, pois, como foi visto no capítulo 4, a linguagem VHDL permite o desenvolvimento de um modelo comportamental e posterior detalhamento ao nível de portas lógicas. Neste nível de detalhamento, o modelo fica bem próximo do "hardware" real, inclusive com atrasos de propabação dos sinais considerados.

  Com esta técnica, o trabalho de desenvolvimento do "hardware" propriamente dito se torna muito seguro, pois ele conta com o suporte de um modelo detalhado. Este trabalho pode ainda ser facilitado com o uso de ferramentas de síntese existentes que aceitam um modelo em linguagem VHDL como entrada para gerar o projeto do "hardware".

- **Segunda linha para futuros desenvolvimentos**

  A segunda linha de desenvolvimento seria a obtenção de modelos comportamentais para os demais blocos funcionais que não foram considerados neste trabalho.

Das 3 operações apresentadas no capítulo 2; multiplexagem, alinhamento e mapeamento, somente as duas primeiras foram contempladas neste trabalho. Portanto, o modelo comportamental de multiplexador HDS ficaria completo com o desenvolvimento de modelos para os blocos funcionais encarregados do mapeamento.

Faltaria ainda desenvolver modelos para os blocos de proteção, para os blocos de conexão e para os blocos que processam os bytes de supervisão, ou blocos de terminação.

- **Terceira linha para futuros desenvolvimentos**

  A terceira linha seria a posibilidade de se utilizar os modelos como suporte para o desenvolvimento de inovações que possibilitem maior eficiência no "hardware" dos equipamentos HDS. Nesta mesma linha, com o desenvolvimento de modelos para todos os blocos funcionais, seria possível simular uma rede detalhada de multiplexadores HDS. Esta rede seria um laboratório para o desenvolvimento de "softwares" de supervisão e controle de rede síncrona.

A seguir, no apêndice, são colocadas todas as listagens de programas. Se forem simuladas exatamente como estão, elas irão gerar os mesmos resultados aqui alcançados.

Bibliografia

[1] Kong Dennis T., "2.488 Gb/s SONET Multiplexer/Demultiplexer with Frame Detection Capacity", IEEE Journal of Selected Areas in Communications, Vol. 9, no. 5, pp. 726-731, June 1991.

[2] CCITT Rec. G.709 - Synchronous multiplexing structure.

[3] CCITT Rec. G.783 - Characteristics of synchronous digital hierarchy (SDH) multiplexing equipment functional blocks.

[4] Perry D. L., "VHDL", McGraw-Hill, 1991.

[5] Armstrong J. R., "Chip-Level Modeling with VHDL", Prentice Hall, 1989.

Apêndice A

Listagens dos programas

```
ENTITY control_local IS
 PORT(T0: IN BIT;
      linha_local, coluna_local, ind_subq_local, cont_3_tug3,
      cont_7_tug2, cont_3_tu12, ender_tu12_local: BUFFER INTEGER:=0;
      inic_vc4_local, clk_tug3_local, clk_tug2_local, clk_tu12_local,
      inic_vc12_local: BUFFER BIT);
END control_local;

ARCHITECTURE comportamental OF control_local IS

BEGIN

 Gerador_dos_Sinais_linha_local_e_coluna_local :
PROCESS(T0)
BEGIN
 IF (T0 = '0') AND (T0'EVENT) THEN
  IF (coluna_local = 269) THEN
   IF (linha_local = 8) THEN
    linha_local <= 0;
    coluna_local <= 0;
   ELSE
    linha_local <= linha_local + 1;
    coluna_local <= 0;
   END IF;
  ELSE
   coluna_local <= coluna_local + 1;
  END IF;
 END IF;
END PROCESS;

 Gerador_do_Sinal_ind_subq_local :
PROCESS(linha_local)
BEGIN
 IF (linha_local = 8) AND (linha_local'EVENT) THEN
  IF (ind_subq_local = 3) THEN
   ind_subq_local <= 0;
  ELSE
   ind_subq_local <= ind_subq_local + 1;
  END IF;
 END IF;
END PROCESS;

 Gerador_do_Sinal_inic_vc4_local :
PROCESS(T0)
BEGIN
 IF (T0 = '1') AND (T0'EVENT) THEN
  IF (coluna_local = 9) AND (linha_local = 3) THEN
   inic_vc4_local <= '1';
  ELSE
   inic_vc4_local <= '0';
  END IF;
 END IF;
END PROCESS;

 Gerador_do_Sinal_cont_3_tug3 :
PROCESS(T0, linha_local)

BEGIN
 IF (T0 = '0') AND (T0'EVENT) THEN
  IF (cont_3_tug3 = 2) THEN
   cont_3_tug3 <= 0;
  ELSE
   cont_3_tug3 <= cont_3_tug3 + 1;
  END IF;
 END IF;
 IF (linha_local = 0) AND (linha_local'EVENT) THEN
  cont_3_tug3 <= 0;
 END IF;

END PROCESS;

clk_tug3_local <= '1' WHEN (coluna_local > 11) AND (cont_3_tug3 =0) ELSE
                  '0';

 Gerador_do_Sinal_cont_7_tug2 :
PROCESS(clk_tug3_local, coluna_local)
BEGIN
 IF ((coluna_local <= 15) OR (coluna_local >= 268)) AND
                                        (coluna_local'EVENT) THEN
  cont_7_tug2 <= 7;
 END IF;
 IF (clk_tug3_local = '0') AND (clk_tug3_local'EVENT) AND
                (coluna_local > 14) AND (coluna_local < 267) THEN
  IF (cont_7_tug2 = 6) OR (cont_7_tug2 = 7)  THEN
   cont_7_tug2 <= 0;
  ELSE
   cont_7_tug2 <= cont_7_tug2 + 1;
  END IF;
 END IF;

END PROCESS;

clk_tug2_local <= '1' WHEN (cont_7_tug2 = 0) ELSE
                  '0';

 Gerador_do_Sinal_cont_3_tu12 :
PROCESS(clk_tug2_local, linha_local)
BEGIN
 IF (linha_local = 0) AND (linha_local'EVENT) THEN
  cont_3_tu12 <= 0;
 END IF;
 IF (clk_tug2_local = '0') AND (clk_tug2_local'EVENT) THEN
  IF (cont_3_tu12 = 2) THEN
   cont_3_tu12 <= 0;
  ELSE
   cont_3_tu12 <= cont_3_tu12 + 1;
  END IF;
 END IF;
END PROCESS;

clk_tu12_local <= '1' WHEN (cont_3_tu12 = 0) ELSE
                  '0';

 Gerador_do_Sinal_ender_tu12_local :
PROCESS(clk_tu12_local, linha_local)
BEGIN
 IF (linha_local = 3) AND (linha_local'EVENT) THEN
  ender_tu12_local <= 0;
 END IF;
 IF (clk_tu12_local = '0') AND (clk_tu12_local'EVENT) THEN
  IF (ender_tu12_local = 35) THEN
   ender_tu12_local <= 0;
  ELSE
   ender_tu12_local <= ender_tu12_local + 1;
  END IF;
 END IF;

END PROCESS;

 Gerador_do_Sinal_inic_vc12_local :
PROCESS(clk_tu12_local)
BEGIN
 IF (clk_tu12_local = '1') AND (clk_tu12_local'EVENT) THEN
  IF (ind_subq_local = 1) AND (ender_tu12_local = 1) THEN
   inic_vc12_local <= '1';
  ELSE
   inic_vc12_local <= '0';
  END IF;
 END IF;
END PROCESS;

END comportamental;

USE work.pac.ALL;
ENTITY sar IS
 PORT(STM1: IN BIT_VECTOR(1 TO 8);
      linha, coluna, linha_local, coluna_local: IN INTEGER:=0;
      T0, T1: IN BIT;
      just_p, just_n, inic_vc4, clk_vc4: BUFFER BIT;
      fase: BUFFER INTEGER:=0;
      VC4: OUT BIT_VECTOR(1 TO 8));
END sar;

ARCHITECTURE comportamental OF sar IS

SIGNAL H1, H2: BIT_VECTOR(1 TO 8);

SIGNAL bur_1, bur_2, bur_3, bur_4, bur1_1, bur2_1, bur3_1, bur4_1,
       rel_esc_normal, rel_esc_justp, rel_esc_justn, rel_esc, rel_leit_normal,
       rel_leit_justp, rel_leit_justn, rel_leit, inv_I, inv_D, NDF, inic_vc,
       para_leit: BIT;

SIGNAL cont_3, cont_783, v_pont, ender_esc, ender_leit, just_dep_3,
       cont3_1: INTEGER := 0;

SIGNAL mem_elast: ender_mem;
SIGNAL nono_bit: ender_mem_inic_vc;

BEGIN

 LEITOR_PONTEIRO : PROCESS(T1)

BEGIN

 IF (T1 = '1') AND (T1'EVENT) THEN
  IF (linha = 3) THEN
   CASE coluna IS
    WHEN 0 => H1 <= STM1;
    WHEN 3 => H2 <= STM1;
    WHEN others => null;
   END CASE;
  END IF;
 END IF;
END PROCESS;

bur_1 <= '1' WHEN coluna >= 0 AND coluna <= 8 AND  linha /= 3 ELSE
         '0';

bur_2 <= '1' WHEN coluna >= 0 AND coluna <= 5 AND  linha = 3 ELSE
         '0';

bur_3 <= '1' WHEN coluna >= 6 AND coluna <= 8 AND  linha = 3 ELSE
         '0';

bur_4 <= '1' WHEN coluna >= 9 AND coluna <= 11 AND linha = 3 ELSE
         '0';

rel_esc_normal <= T1 AND NOT(bur_1) AND NOT(bur_2) AND NOT(bur_3);

rel_esc_justp <= T1 AND NOT(bur_1) AND NOT(bur_2) AND NOT(bur_3) AND NOT(bur_4);

rel_esc_justn <= T1 AND NOT(bur_1) AND NOT(bur_2);

CONTADORES : PROCESS(rel_esc_normal, bur_4)
BEGIN
 IF (bur_4 = '1') AND (bur_4'EVENT) THEN
  cont_3 <= 0;
  cont_783 <= 0;
 END IF;
 IF (rel_esc_normal = '0') AND (rel_esc_normal'EVENT) THEN
  IF (cont_3 = 2) THEN
   cont_3 <= 0;
   cont_783 <= cont_783 + 1;
  ELSE
   cont_3 <= cont_3 + 1;
  END IF;
 END IF;
END PROCESS;

 INT_PONTEIRO : PROCESS(bur_2)
variable palavra, palavra_ant, palavra_ant_ant : bit_vector(1 to 16);
variable  cont_I, cont_D, cont_NDF : integer;

BEGIN
 IF (bur_2 = '0') THEN
  palavra_ant_ant := palavra_ant;
  palavra_ant := palavra;
  palavra := byte_pra_palavra(H1, H2);
  cont_I := 0;
  cont_D := 0;
  for i in 7 to 16 loop
   IF (i mod 2 /= 0) THEN
    IF (palavra(i) /= palavra_ant(i)) THEN
     cont_I := cont_I + 1;
    END IF;
    IF (palavra(i+1) /= palavra_ant(i+1)) THEN
     cont_D := cont_D + 1;
    END IF;
   END IF;
  END loop;
  IF (inv_I = '0' AND inv_D = '0') THEN
   IF (cont_I >= 3) THEN
    inv_I <= '1';
    IF (v_pont = 782) THEN
     v_pont <= 0;
    ELSE
```

```
     v_pont <= v_pont + 1;
    END IF;
   END IF;
   IF (cont_D >= 3) THEN
    inv_D <= '1';
    IF (v_pont = 0) THEN
     v_pont <= 782;
    ELSE
     v_pont <= v_pont - 1;
    END IF;
   END IF;
  ELSE
   inv_I <= '0';
   inv_D <= '0';
  END IF;
  IF (palavra = palavra_ant) AND (palavra = palavra_ant_ant) THEN
   v_pont <= palavra_pra_inteiro(palavra);
  END IF;
  cont_NDF := 0;
  for i in 1 to 4 loop
   IF (palavra(i) = v_NDF(i)) THEN
    cont_NDF := cont_NDF + 1;
   END IF;
  END loop;
  IF (cont_NDF >= 3) THEN
   NDF <= '1';
   inv_I <= '0';
   inv_D <= '0';
   v_pont <= palavra_pra_inteiro(palavra);
  ELSE
   NDF <= '0';
  END IF;
 END IF;
END PROCESS;

inic_vc <= '1' WHEN cont_3 = 0 AND cont_783 = v_pont ELSE
           '0';

rel_esc <= rel_esc_justn WHEN inv_D = '1' ELSE
           rel_esc_justp WHEN inv_I = '1' ELSE
           rel_esc_normal;

 CONT_END_ESCRITA : PROCESS(rel_esc)
BEGIN
 IF (rel_esc = '0') THEN
  IF (ender_esc = 47) THEN
   ender_esc <= 0;
  ELSE
   ender_esc <= ender_esc + 1;
  END IF;
 END IF;
END PROCESS;

 ESC_MEM_ELASTICA : PROCESS(rel_esc)
BEGIN
 IF (rel_esc = '1') THEN
  mem_elast(ender_esc) <= STM1;
  nono_bit(ender_esc) <= inic_vc;
 END IF;
END PROCESS;

 LEIT_MEM_ELASTICA : PROCESS(clk_vc4)
BEGIN
 IF (clk_vc4 = '1') THEN
  VC4 <= mem_elast(ender_leit);
  inic_vc4 <= nono_bit(ender_leit);
 END IF;
END PROCESS;

para_leit <= '1' WHEN inic_vc4 = '1' AND cont3_1 = 1 ELSE
             '1' WHEN inic_vc4 = '1' AND cont3_1 = 2 ELSE
             '0';

 CONT_END_LEITURA : PROCESS(clk_vc4)
BEGIN
 IF (clk_vc4 = '0') THEN
  IF (para_leit = '0') THEN
   IF (ender_leit = 47) THEN
    ender_leit <= 0;
   ELSE
    ender_leit <= ender_leit + 1;
   END IF;
  END IF;
 END IF;
END PROCESS;

 FASIMETRO : PROCESS(ender_esc, ender_leit)
variable v_fase : integer := 0;
BEGIN
 v_fase := ender_esc - ender_leit;
 IF (v_fase < 0) THEN
  fase <= 48 + v_fase;
 ELSE
  fase <= v_fase;
 END IF;
END PROCESS;

END comportamental;


ENTITY control_vc4 IS
 PORT(VC4: IN BIT_VECTOR(1 TO 8);
      inic_vc4, clk_vc4: IN BIT;
      linha_vc4, coluna_vc4: BUFFER INTEGER;
      ind_subq: OUT INTEGER);
END control_vc4;


ARCHITECTURE comportamental OF control_vc4 IS
BEGIN

 linha_coluna_vc4: PROCESS(clk_vc4)
BEGIN
 IF (clk_vc4 = '1') AND (clk_vc4'EVENT) THEN
  IF (coluna_vc4 = 260) THEN
   IF (linha_vc4 = 8) THEN
    linha_vc4 <= 0;
    coluna_vc4 <= 0;
   ELSE
    linha_vc4 <= linha_vc4 + 1;
    coluna_vc4 <= 0;
   END IF;
  ELSE
   coluna_vc4 <= coluna_vc4 + 1;
  END IF;
 END IF;
 IF (clk_vc4 = '0') AND (clk_vc4'EVENT) THEN
  IF (inic_vc4 = '1') THEN
   coluna_vc4 <= 0;
   linha_vc4 <= 0;
  END IF;
 END IF;
END PROCESS;


 indicador_de_subquadro: PROCESS(clk_vc4)
 VARIABLE H4 : BIT_VECTOR(1 TO 8);
BEGIN
 IF (clk_vc4 = '0') THEN
  IF (coluna_vc4 = 0) AND (linha_vc4 = 5) THEN
   H4 := VC4;
   IF H4(7) = '0' AND H4(8) = '0' THEN
    ind_subq <= 0;
   ELSIF H4(7) = '0' AND H4(8) = '1' THEN
    ind_subq <= 1;
   ELSIF H4(7) = '1' AND H4(8) = '0' THEN
    ind_subq <= 2;
   ELSE
    ind_subq <= 3;
   END IF;
  END IF;
 END IF;
END PROCESS;


END comportamental;


ENTITY dmult_vc4 IS
 PORT(VC4: IN BIT_VECTOR(1 TO 8);
      clk_vc4, inic_vc4: IN BIT;
      coluna_vc4: IN INTEGER;
      TUG3_1, TUG3_2, TUG3_3: OUT BIT_VECTOR(1 TO 8);
      clk_tug3_1, clk_tug3_2, clk_tug3_3: BUFFER BIT);
END dmult_vc4;

ARCHITECTURE comportamental OF dmult_vc4 IS

SIGNAL cont_3: INTEGER;

BEGIN


GER_CONT_3: PROCESS(clk_vc4)
BEGIN
 IF (clk_vc4 = '1') AND (clk_vc4'EVENT) THEN
  IF (cont_3 = 2) THEN
   cont_3 <= 0;
  ELSE
   cont_3 <= cont_3 + 1;
  END IF;
 END IF;
 IF (clk_vc4 = '0') AND (clk_vc4'EVENT) THEN
  IF (inic_vc4 = '1') THEN
   cont_3 <= 0;
  END IF;
 END IF;
END PROCESS;


clk_tug3_1 <= '1' WHEN (coluna_vc4 > 2) AND (cont_3 = 0) AND
                                                (clk_vc4 = '0') ELSE
              '0';

clk_tug3_2 <= '1' WHEN (coluna_vc4 > 2) AND (cont_3 = 1) AND
                                                (clk_vc4 = '0') ELSE
              '0';
```

```
 END IF;
END PROCESS;

bur1_1 <= '1' WHEN coluna_local >= 0 AND coluna_local <= 8 AND
                                          linha_local /= 3 ELSE
          '0';

bur2_1 <= '1' WHEN coluna_local >= 0 AND coluna_local <= 5 AND
                                          linha_local = 3 ELSE
          '0';


bur3_1 <= '1' WHEN coluna_local >= 6 AND coluna_local <= 8 AND
                                          linha_local = 3 ELSE
          '0';

bur4_1 <= '1' WHEN coluna_local >= 9 AND coluna_local <= 11  AND
                                          linha_local= 3 ELSE
          '0';


rel_leit_normal <= T0 AND NOT(bur1_1) AND NOT(bur2_1) AND NOT(bur3_1);
rel_leit_justp <= T0 AND NOT(bur1_1) AND NOT(bur2_1) AND NOT(bur3_1) AND
                                                        NOT(bur4_1);

rel_leit_justn <= T0 AND NOT(bur1_1) AND NOT(bur2_1);


clk_vc4 <= rel_leit_justn WHEN just_n = '1' ELSE
           rel_leit_justp WHEN just_p = '1' ELSE
           rel_leit_normal;

 JUSTIFICADOR : PROCESS(linha_local)
BEGIN
 IF (linha_local = 2) AND (linha_local'EVENT) THEN
  IF (just_dep_3 = 2) THEN
   IF (fase <= 15) THEN
    just_p <= '1';
    just_dep_3 <= 0;
  END IF;
   IF (fase >= 33) THEN
    just_n <= '1';
    just_dep_3 <= 0;
   END IF;
  ELSE
   just_dep_3 <= just_dep_3 + 1;
   just_p <= '0';
   just_n <= '0';
  END IF;
 END IF;
END PROCESS;


 CONT3_LOCAL : PROCESS(T0)
BEGIN
 IF (T0 = '1') THEN
  IF (coluna_local = 0) THEN
   cont3_1 <= 0;
  ELSE
   IF (cont3_1 = 2) THEN
    cont3_1 <= 0;
   ELSE
    cont3_1 <= cont3_1 + 1;
   END IF;
  END IF;
 END IF;
```

```
clk_tug3_3 <= '1' WHEN (coluna_vc4 > 2) AND (cont_3 = 2) AND
                                                    (clk_vc4 = '0') ELSE
              '0';



leitor_de_tug3: PROCESS(clk_tug3_1, clk_tug3_2, clk_tug3_3)
BEGIN
 IF  (clk_tug3_1 = '1') AND (clk_tug3_1'EVENT) THEN
  TUG3_1 <= VC4;
 END IF;
 IF  (clk_tug3_2 = '1') AND (clk_tug3_2'EVENT) THEN
  TUG3_2 <= VC4;
 END IF;
 IF  (clk_tug3_3 = '1') AND (clk_tug3_3'EVENT) THEN
  TUG3_3 <= VC4;
 END IF;
END PROCESS;

END comportamental;

ENTITY dmult_tug3 IS
 PORT(TUG3: IN BIT_VECTOR(1 TO 8);
      coluna_vc4: IN INTEGER;
      clk_tug3: IN BIT;
      TUG2_1, TUG2_2, TUG2_3, TUG2_4, TUG2_5, TUG2_6,
      TUG2_7: OUT BIT_VECTOR(1 TO 8);
      clk_tug2_1, clk_tug2_2, clk_tug2_3, clk_tug2_4,
      clk_tug2_5, clk_tug2_6, clk_tug2_7: BUFFER BIT);
END dmult_tug3;

ARCHITECTURE comportamental OF dmult_tug3 IS

SIGNAL cont_7: INTEGER:=0;

BEGIN


 ger_cont_7: PROCESS(clk_tug3)
BEGIN
 IF (clk_tug3 = '1') AND (clk_tug3'EVENT) THEN
  IF (coluna_vc4 > 2) AND (coluna_vc4 < 9) THEN
   cont_7 <= 7;
  ELSE
   IF (cont_7 = 6) OR (cont_7 = 7) THEN
    cont_7 <= 0;
   ELSE
    cont_7 <= cont_7 + 1;
   END IF;
  END IF;
 END IF;
END PROCESS;


 clk_tug2_1 <= '1' WHEN (cont_7 = 0) AND (clk_tug3 = '0') ELSE
               '0';

clk_tug2_2 <= '1' WHEN (cont_7 = 1) AND (clk_tug3 = '0') ELSE
              '0';

clk_tug2_3 <= '1' WHEN (cont_7 = 2) AND (clk_tug3 = '0') ELSE
              '0';

clk_tug2_4 <= '1' WHEN (cont_7 = 3) AND (clk_tug3 = '0') ELSE
              '0';
```

```
clk_tug2_5 <= '1' WHEN (cont_7 = 4) AND (clk_tug3 = '0') ELSE
              '0';

clk_tug2_6 <= '1' WHEN (cont_7 = 5) AND (clk_tug3 = '0') ELSE
              '0';

clk_tug2_7 <= '1' WHEN (cont_7 = 6) AND (clk_tug3 = '0') ELSE
              '0';



leitor_de_tug2: PROCESS(clk_tug2_1, clk_tug2_2, clk_tug2_3, clk_tug2_4,
                        clk_tug2_5, clk_tug2_6, clk_tug2_7)
BEGIN
 IF  (clk_tug2_1 = '1') AND (clk_tug2_1'EVENT) THEN
  TUG2_1 <= TUG3;
 END IF;
 IF  (clk_tug2_2 = '1') AND (clk_tug2_2'EVENT) THEN
  TUG2_2 <= TUG3;
 END IF;
 IF  (clk_tug2_3 = '1') AND (clk_tug2_3'EVENT) THEN
  TUG2_3 <= TUG3;
 END IF;
 IF  (clk_tug2_4 = '1') AND (clk_tug2_4'EVENT) THEN
  TUG2_4 <= TUG3;
 END IF;
 IF  (clk_tug2_5 = '1') AND (clk_tug2_5'EVENT) THEN
  TUG2_5 <= TUG3;
 END IF;
 IF  (clk_tug2_6 = '1') AND (clk_tug2_6'EVENT) THEN
  TUG2_6 <= TUG3;
 END IF;
 IF  (clk_tug2_7 = '1') AND (clk_tug2_7'EVENT) THEN
  TUG2_7 <= TUG3;
 END IF;
END PROCESS;

END comportamental;


ENTITY dmult_tug2 IS
 PORT(clk_tug2, inic_vc4: IN BIT;
      clk_tu12_1, clk_tu12_2, clk_tu12_3: OUT BIT);
END dmult_tug2;

ARCHITECTURE comportamental OF dmult_tug2 IS

SIGNAL cont_3: INTEGER:=0;

BEGIN

 ger_cont_3: PROCESS(clk_tug2, inic_vc4)
BEGIN
 IF (inic_vc4 = '0') AND (inic_vc4'EVENT) THEN
  cont_3 <= 2;
 END IF;
 IF (clk_tug2 = '1') AND (clk_tug2'EVENT) THEN
  IF (cont_3 = 2)  THEN
   cont_3 <= 0;
  ELSE
   cont_3 <= cont_3 + 1;
  END IF;
 END IF;
END PROCESS;
```

```
clk_tu12_1 <= '1' WHEN (cont_3 = 0) AND (clk_tug2 = '0') ELSE
              '0';

clk_tu12_2 <= '1' WHEN (cont_3 = 1) AND (clk_tug2 = '0') ELSE
              '0';

clk_tu12_3 <= '1' WHEN (cont_3 = 2) AND (clk_tug2 = '0') ELSE
              '0';
END comportamental;


USE work.pac.ALL;
ENTITY adapt_vc12_r IS
 PORT(TUG2: IN BIT_VECTOR(1 TO 8);
      ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4,
      linha_vc4: IN INTEGER:=0;
      clk_tu12, clk_tu12_local: IN BIT;
      just_p, just_n, inic_vc12, clk_vc12: BUFFER BIT;
      fase: BUFFER INTEGER:=0;
      VC12: OUT BIT_VECTOR(1 TO 8));
END adapt_vc12_r;

ARCHITECTURE comportamental OF adapt_vc12_r IS

SIGNAL V1, V2: BIT_VECTOR(1 TO 8);

SIGNAL bur_1, bur_2, bur_3, bur1_1, bur2_1, bur3_1, rel_esc_normal, rel_es
       rel_esc_justp, rel_esc_justn, rel_leit_normal, rel_leit_justp,
       rel_leit_justn, inv_I, inv_D, NDF, inic_vc: BIT;

SIGNAL cont_140, v_pont, ender_esc, ender_leit, ender_tu12: INTEGER:=0;

SIGNAL mem_elast: mem_elast_tu12;
SIGNAL nono_bit: nono_bit_tu12;

BEGIN

 ger_ender_tu12: PROCESS(coluna_vc4, clk_tu12)
 BEGIN
  IF (coluna_vc4 = 9) AND (coluna_vc4'EVENT) AND (linha_vc4 = 0) THEN
   IF (clk_tu12 = '1') THEN
    ender_tu12 <= 35;
   ELSE
    ender_tu12 <= 0;
   END IF;
  END IF;
  IF (clk_tu12 = '0') AND (clk_tu12'EVENT) THEN
   IF (ender_tu12 = 35) THEN
    ender_tu12 <= 0;
   ELSE
    ender_tu12 <= ender_tu12 + 1;
   END IF;
  END IF;
END PROCESS;


 leitura_ponteiro: PROCESS(clk_tu12)
BEGIN
 IF (clk_tu12 = '1') THEN
  IF (ender_tu12 = 0) THEN
   CASE ind_subq IS
    WHEN 0 => V1 <= TUG2;
    WHEN 1 => V2 <= TUG2;
    WHEN OTHERS => NULL;
   END CASE;
```

```
  END IF;
 END IF;
END PROCESS;


bur_1 <= '1' WHEN ender_tu12 = 0 AND ind_subq /= 2 ELSE
         '0';

bur_2 <= '1' WHEN ender_tu12 = 0 AND ind_subq = 2 ELSE
         '0';

bur_3 <= '1' WHEN ender_tu12 = 1 AND ind_subq = 2 ELSE
         '0';


rel_esc_justn <= clk_tu12 AND NOT(bur_1);

rel_esc_justp <= clk_tu12 AND NOT(bur_1) AND NOT(bur_2) AND NOT(bur_3);

rel_esc_normal <= clk_tu12 AND NOT(bur_1) AND NOT(bur_2);


 contador: PROCESS(rel_esc_normal, ender_tu12)

BEGIN
 IF (rel_esc_normal = '0') AND (rel_esc_normal'EVENT) THEN
  cont_140 <= cont_140 + 1;
 END IF;
 IF (ender_tu12 = 1) AND (ender_tu12'EVENT) THEN
  IF (ind_subq = 1) THEN
   cont_140 <= 0;
  END IF;
 END IF;
END PROCESS;



 INT_PONTEIRO : PROCESS(ender_tu12)
variable palavra, palavra_ant, palavra_ant_ant : bit_vector(1 to 16);
variable  cont_I, cont_D, cont_NDF : integer;
BEGIN
 IF (ender_tu12 = 0) THEN
  IF (ind_subq = 2) THEN
   palavra_ant_ant := palavra_ant;
   palavra_ant := palavra;
   palavra := byte_pra_palavra(V1, V2);
   cont_I := 0;
   cont_D := 0;
   FOR i in 7 to 16 LOOP
    IF (i mod 2 /= 0) THEN
     IF (palavra(i) /= palavra_ant(i)) THEN
      cont_I := cont_I + 1;
     END IF;
     IF (palavra(i+1) /= palavra_ant(i+1)) THEN
      cont_D := cont_D + 1;
     END IF;
    END IF;
   END LOOP;
   IF (inv_I = '0' and inv_D = '0') THEN
    IF (cont_I >= 3) THEN
     inv_I <= '1';
     IF (v_pont = 139) THEN
      v_pont <= 0;
     ELSE
      v_pont <= v_pont + 1;
     END IF;
```

```
    END IF;
    IF (cont_D >= 3) THEN
     inv_D <= '1';
     IF (v_pont = 0) THEN
      v_pont <= 139;
     ELSE
      v_pont <= v_pont - 1;
     END IF;
    END IF;
   ELSE
    inv_I <= '0';
    inv_D <= '0';
   END IF;
   IF (palavra = palavra_ant) and (palavra = palavra_ant_ant) THEN
    v_pont <= palavra_pra_inteiro(palavra);
   END IF;
   cont_NDF := 0;
   FOR i in 1 to 4 LOOP
    IF (palavra(i) = v_NDF(i)) THEN
     cont_NDF := cont_NDF + 1;
    END IF;
   END LOOP;
   IF (cont_NDF = 4) THEN
    NDF <= '1';
    inv_I <= '0';
    inv_D <= '0';
    v_pont <= palavra_pra_inteiro(palavra);
   ELSE
    NDF <= '0';
   END IF;
  END IF;
 END IF;
END PROCESS;

 inic_vc <= '1' WHEN cont_140 = v_pont ELSE
            '0';


rel_esc <= rel_esc_justn when inv_D = '1' ELSE
           rel_esc_justp when inv_I = '1' ELSE
           rel_esc_normal;


 CONT_END_ESCRITA : PROCESS(rel_esc)
BEGIN
 IF (rel_esc = '0') THEN
  IF (ender_esc = 7) THEN
   ender_esc <= 0;
  ELSE
   ender_esc <= ender_esc + 1;
  END IF;
 END IF;
END PROCESS;


 ESC_MEM_ELASTICA : PROCESS(rel_esc)
BEGIN
 IF (rel_esc = '1') THEN
  mem_elast(ender_esc) <= TUG2;
  nono_bit(ender_esc) <= inic_vc;
 END IF;
END PROCESS;


 LEIT_MEM_ELASTICA : PROCESS(clk_vc12)
BEGIN
 IF (clk_vc12 = '1') THEN
  VC12 <= mem_elast(ender_leit);
  inic_vc12 <= nono_bit(ender_leit);
 END IF;
END PROCESS;


bur1_l <= '1' WHEN ender_tu12_local = 0 AND ind_subq_local /= 2 ELSE
          '0';

bur2_l <= '1' WHEN ender_tu12_local = 0 AND ind_subq_local = 2 ELSE
          '0';

bur3_l <= '1' WHEN ender_tu12_local = 1 AND ind_subq_local = 2 ELSE
          '0';



rel_leit_normal <= clk_tu12_local AND NOT(bur1_l) AND NOT(bur2_l);

rel_leit_justp <= clk_tu12_local AND NOT(bur1_l) AND NOT(bur2_l)
                                                 AND NOT(bur3_l);

rel_leit_justn <= clk_tu12_local AND NOT(bur1_l);


clk_vc12 <= rel_leit_justn when just_n = '1' ELSE
            rel_leit_justp when just_p = '1' ELSE
            rel_leit_normal;


 JUSTIFICADOR : PROCESS(ind_subq_local)
BEGIN
 IF (ind_subq_local = 3) AND (ind_subq_local'EVENT) THEN
  IF (fase <= 2) THEN
   just_p <= '1';
  ELSE
   just_p <= '0';
  END IF;
  IF (fase >= 6) THEN
   just_n <= '1';
  ELSE
   just_n <= '0';
  END IF;
 END IF;
END PROCESS;



 CONT_END_LEITURA : PROCESS(clk_vc12)
BEGIN
 IF (clk_vc12 = '0') THEN
  IF (ender_leit = 7) THEN
   ender_leit <= 0;
  ELSE
   ender_leit <= ender_leit + 1;
  END IF;
 END IF;
END PROCESS;


 FASIMETRO : PROCESS(ender_esc, ender_leit)
variable v_fase : integer := 0;
BEGIN
 v_fase := ender_esc - ender_leit;
 IF (v_fase < 0) THEN
  fase <= 8 + v_fase;
 ELSE
  fase <= v_fase;
 END IF;
END PROCESS;

END comportamental;

USE WORK.pac.ALL;
ENTITY adapt_vc12_t IS
PORT(VC12: IN BIT_VECTOR(1 TO 8);
     ind_subq_local, ender_tu12_local: IN INTEGER:=0;
     clk_tu12_local, inic_vc12, just_p, just_n: IN BIT;
     TU12: OUT BIT_VECTOR(1 TO 8));
END adapt_vc12_t;

ARCHITECTURE comportamental OF adapt_vc12_t IS


SIGNAL rel_cont, NDF, chave_V1, chave_V2: BIT;
SIGNAL cont_140, v_pont: INTEGER;
SIGNAL V1, V2: BIT_VECTOR(1 TO 8);



BEGIN

rel_cont <= clk_tu12_local when ender_tu12_local /= 0  else
            '0';


CONTADOR : PROCESS(rel_cont, ender_tu12_local)
begin
 IF (ender_tu12_local = 1) AND (ender_tu12_local'EVENT) AND
                                    (ind_subq_local = 1) THEN
    cont_140 <= 0;
 END IF;
 IF (rel_cont = '0') AND (rel_cont'EVENT) THEN
  cont_140 <= cont_140 + 1;
 END IF;
END PROCESS;


 gerador_ponteiro: PROCESS(inic_vc12, ender_tu12_local)
  BEGIN
   IF (ender_tu12_local = 0) AND (ender_tu12_local'EVENT) THEN

    IF (ind_subq_local = 3) THEN

     IF (NDF = '1') THEN
      monta_ponteiro_com_NDF(V1, V2, v_pont);
     ELSE
      IF (just_p = '1') THEN
       inverte_bits_I(V1, V2);

      ELSIF (just_n = '1') THEN
       inverte_bits_D(V1, V2);

      ELSE
       monta_ponteiro(V1, V2, v_pont);
      END IF;
     END IF;


    ELSIF (ind_subq_local = 2) THEN

      IF (just_p = '1') THEN
       IF (v_pont = 139) THEN
        v_pont <= 0;
       ELSE
        v_pont <= v_pont + 1;
       END IF;
      ELSIF (just_n = '1') THEN
       IF (v_pont = 0) THEN
        v_pont <= 139;
       ELSE
        v_pont <= v_pont - 1;
       END IF;
      END IF;
    END IF;
   END IF;
   IF (inic_vc12 = '1') AND (inic_vc12'EVENT) THEN
     IF (cont_140 = v_pont) THEN
      NDF <= '0';
     ELSE
      NDF <= '1';
      v_pont <= cont_140;
     END IF;
    END IF;
  END PROCESS;


 chaves_para_V1_e_V2 : PROCESS(clk_tu12_local)
BEGIN
 IF (clk_tu12_local = '1') AND (clk_tu12_local'EVENT) THEN
  IF (ind_subq_local = 0) AND (ender_tu12_local = 0) THEN
   chave_V1 <= '1';
  ELSIF (ind_subq_local = 1) AND (ender_tu12_local = 0) THEN
   chave_V2 <= '1';
  ELSE
   chave_V1 <= '0';
   chave_V2 <= '0';
  END IF;
 END IF;
END PROCESS;


TU12 <= V1 WHEN chave_V1 = '1' ELSE
        V2 WHEN chave_V2 = '1' ELSE
        VC12;


END comportamental;


ENTITY mult_tu12 IS
 PORT(TU12_1, TU12_2, TU12_3: IN BIT_VECTOR(1 TO 8);
      cont_3_tu12: IN INTEGER;
      clk_tug2_local: IN BIT;
      TUG2: OUT BIT_VECTOR(1 TO 8));
END mult_tu12;

ARCHITECTURE comportamental OF mult_tu12 IS

BEGIN

Multiplexador_de_TU12 : PROCESS(clk_tug2_local)
BEGIN
 IF (clk_tug2_local = '1') AND (clk_tug2_local'EVENT) THEN
  CASE cont_3_tu12 IS
   WHEN 0 => TUG2 <= TU12_1;
   WHEN 1 => TUG2 <= TU12_2;
```

```
   WHEN 2 => TUG2 <= TU12_3;
   WHEN OTHERS => NULL;
  END CASE;
 END IF;
END PROCESS;

END comportamental;

ENTITY mult_tug2 IS
 PORT(TUG2_1, TUG2_2, TUG2_3, TUG2_4, TUG2_5, TUG2_6,
                                  TUG2_7:IN BIT_VECTOR(1 TO 8);
      cont_7_tug2: IN INTEGER;
      clk_tug3_local: IN BIT;
      TUG3: OUT BIT_VECTOR(1 TO 8));
END mult_tug2;

ARCHITECTURE comportamental OF mult_tug2 IS

BEGIN



 Multiplexador_de_TUG2 : PROCESS(clk_tug3_local)
BEGIN
 IF (clk_tug3_local = '1') AND (clk_tug3_local'EVENT) THEN
  CASE cont_7_tug2 IS
   WHEN 0 => TUG3 <= TUG2_1;
   WHEN 1 => TUG3 <= TUG2_2;
   WHEN 2 => TUG3 <= TUG2_3;
   WHEN 3 => TUG3 <= TUG2_4;
   WHEN 4 => TUG3 <= TUG2_5;
   WHEN 5 => TUG3 <= TUG2_6;
   WHEN 6 => TUG3 <= TUG2_7;
   WHEN OTHERS => NULL;
  END CASE;
 END IF;
END PROCESS;

ENTITY mult_tug3 IS
 PORT(TUG3_1, TUG3_2, TUG3_3: IN BIT_VECTOR(1 TO 8);
      cont_3_tug3: IN INTEGER;
      T0: IN BIT;
      carga_vc4: OUT BIT_VECTOR(1 TO 8));
END mult_tug3;

ARCHITECTURE comportamental OF mult_tug3 IS

BEGIN


Multiplexador_de_TUG3 : PROCESS(T0)
BEGIN
 IF (T0 = '1') AND (T0'EVENT) THEN
  CASE cont_3_tug3 IS
   WHEN 0 => carga_vc4 <= TUG3_1;
   WHEN 1 => carga_vc4 <= TUG3_2;
   WHEN 2 => carga_vc4 <= TUG3_3;
   WHEN OTHERS => NULL;
  END CASE;
 END IF;
END PROCESS;

END comportamental;


USE WORK.pac.ALL;

ENTITY sat IS
 PORT(VC4: IN BIT_VECTOR(1 TO 8);
      linha_local, coluna_local: IN INTEGER:=0;
      T0, inic_vc4, just_p, just_n: IN BIT;
      AU4: OUT BIT_VECTOR(1 TO 8));
END sat;

ARCHITECTURE comportamental OF sat IS

SIGNAL H1, H2: BIT_VECTOR(1 TO 8);

SIGNAL rel_cont, NDF, chave_H1, chave_H2: BIT;

SIGNAL cont_3, cont_783, v_pont: INTEGER:=0;


BEGIN



rel_cont <= T0 WHEN coluna_local > 8  ELSE
             '0';


CONTADOR : PROCESS(rel_cont, linha_local)
BEGIN

 IF (rel_cont = '0') AND (rel_cont'EVENT) THEN
  IF (linha_local = 3) AND (linha_local'EVENT) THEN
   cont_3 <= 0;
   cont_783 <= 0;
  ELSE
   IF (cont_3 = 2) THEN
    cont_3 <= 0;
    cont_783 <= cont_783 + 1;
   ELSE
    cont_3 <= cont_3 + 1;
   END IF;
  END IF;
 END IF;
END PROCESS;


 gerador_ponteiro: PROCESS(rel_cont, linha_local)
  BEGIN
   IF (linha_local = 3) AND (linha_local'EVENT) THEN
    IF (NDF = '1') THEN
     monta_ponteiro_com_NDF(H1, H2, v_pont);
    ELSE
     IF (just_p = '1') THEN
      inverte_bits_I(H1, H2);
      IF (v_pont = 782) THEN
       v_pont <= 0;
      ELSE
       v_pont <= v_pont + 1;
      END IF;
     ELSIF (just_n = '1') THEN
      inverte_bits_D(H1, H2);
      IF (v_pont = 0) THEN
       v_pont <= 782;
      ELSE
       v_pont <= v_pont - 1;
      END IF;
     ELSE
      monta_ponteiro(H1, H2, v_pont);
     END IF;
     END IF;
   END IF;
    IF (rel_cont = '0') AND (rel_cont'EVENT) THEN
    IF (inic_vc4 = '1') THEN
     IF (cont_783 = v_pont) THEN
      NDF <= '0';
     ELSE
      NDF <= '1';
      v_pont <= cont_783;
     END IF;
    END IF;
   END IF;
  END PROCESS;


 chaves_para_H1_H2 : PROCESS(T0)
BEGIN
 IF (T0 = '1') AND (T0'EVENT) THEN
  IF (linha_local = 3) THEN
   CASE coluna_local IS
    WHEN 0 => chave_H1 <= '1';
    WHEN 3 => chave_H2 <= '1';
    WHEN others => chave_H1 <= '0'; chave_H2 <= '0';
   END CASE;
  END IF;

 END IF;
END PROCESS;



AU4 <= H1 WHEN chave_H1 = '1' ELSE
       H2 WHEN chave_H2 = '1' ELSE
       VC4;


END comportamental;

USE WORK.pac_comp.ALL;
ENTITY equip_a IS
 PORT(VC12_01, VC12_02, VC12_03,VC12_04, VC12_05, VC12_06, VC12_07,
      VC12_08, VC12_09, VC12_10,VC12_11, VC12_12, VC12_13, VC12_14,
      VC12_15, VC12_16, VC12_17,VC12_18, VC12_19, VC12_20, VC12_21,
      VC12_22, VC12_23, VC12_24,VC12_25, VC12_26, VC12_27, VC12_28,
      VC12_29, VC12_30, VC12_31,VC12_32, VC12_33, VC12_34, VC12_35,
      VC12_36, VC12_37, VC12_38,VC12_39, VC12_40, VC12_41, VC12_42,
      VC12_43, VC12_44, VC12_45,VC12_46, VC12_47, VC12_48, VC12_49,
      VC12_50, VC12_51, VC12_52,VC12_53, VC12_54, VC12_55, VC12_56,
      VC12_57, VC12_58, VC12_59,VC12_60, VC12_61, VC12_62, VC12_63
                                            : IN BIT_VECTOR(1 TO 8);
      Ta : IN BIT;
      inic_vc12_local, clk_tu12_local: BUFFER BIT;
      coluna_local, linha_local, ender_tu12_local: BUFFER INTEGER:=0;
      AU4: OUT BIT_VECTOR(1 TO 8));
END equip_a;

ARCHITECTURE estrutural OF equip_a IS

FOR comp_control_local: cp_control_local USE ENTITY WORK.control_local;
FOR comp_adapt_vc12_t_01, comp_adapt_vc12_t_02, comp_adapt_vc12_t_03,
    comp_adapt_vc12_t_04, comp_adapt_vc12_t_05, comp_adapt_vc12_t_06,
    comp_adapt_vc12_t_07, comp_adapt_vc12_t_08, comp_adapt_vc12_t_09,
    comp_adapt_vc12_t_10, comp_adapt_vc12_t_11, comp_adapt_vc12_t_12,
    comp_adapt_vc12_t_13, comp_adapt_vc12_t_14, comp_adapt_vc12_t_15,
    comp_adapt_vc12_t_16, comp_adapt_vc12_t_17, comp_adapt_vc12_t_18,
    comp_adapt_vc12_t_19, comp_adapt_vc12_t_20, comp_adapt_vc12_t_21,

    comp_adapt_vc12_t_22, comp_adapt_vc12_t_23, comp_adapt_vc12_t_24,
    comp_adapt_vc12_t_25, comp_adapt_vc12_t_26, comp_adapt_vc12_t_27,
    comp_adapt_vc12_t_28, comp_adapt_vc12_t_29, comp_adapt_vc12_t_30,
    comp_adapt_vc12_t_31, comp_adapt_vc12_t_32, comp_adapt_vc12_t_33,
    comp_adapt_vc12_t_34, comp_adapt_vc12_t_35, comp_adapt_vc12_t_36,
    comp_adapt_vc12_t_37, comp_adapt_vc12_t_38, comp_adapt_vc12_t_39,
    comp_adapt_vc12_t_40, comp_adapt_vc12_t_41, comp_adapt_vc12_t_42,
    comp_adapt_vc12_t_43, comp_adapt_vc12_t_44, comp_adapt_vc12_t_45,
    comp_adapt_vc12_t_46, comp_adapt_vc12_t_47, comp_adapt_vc12_t_48,
    comp_adapt_vc12_t_49, comp_adapt_vc12_t_50, comp_adapt_vc12_t_51,
    comp_adapt_vc12_t_52, comp_adapt_vc12_t_53, comp_adapt_vc12_t_54,
    comp_adapt_vc12_t_55, comp_adapt_vc12_t_56, comp_adapt_vc12_t_57,
    comp_adapt_vc12_t_58, comp_adapt_vc12_t_59, comp_adapt_vc12_t_60,
    comp_adapt_vc12_t_61, comp_adapt_vc12_t_62, comp_adapt_vc12_t_63
                         : cp_adapt_vc12_t USE ENTITY WORK.adapt_vc12_t;

FOR comp_mult_tu12_01, comp_mult_tu12_02, comp_mult_tu12_03,
    comp_mult_tu12_04, comp_mult_tu12_05, comp_mult_tu12_06,
    comp_mult_tu12_07, comp_mult_tu12_08, comp_mult_tu12_09,
    comp_mult_tu12_10, comp_mult_tu12_11, comp_mult_tu12_12,
    comp_mult_tu12_13, comp_mult_tu12_14, comp_mult_tu12_15,
    comp_mult_tu12_16, comp_mult_tu12_17, comp_mult_tu12_18,
    comp_mult_tu12_19, comp_mult_tu12_20, comp_mult_tu12_21
                         : cp_mult_tu12 USE ENTITY WORK.mult_tu12;

FOR comp_mult_tug2_1, comp_mult_tug2_2, comp_mult_tug2_3
                         : cp_mult_tug2 USE ENTITY WORK.mult_tug2;

FOR comp_mult_tug3 : cp_mult_tug3 USE ENTITY WORK.mult_tug3;
FOR comp_insere_h4: cp_insere_h4 USE ENTITY WORK.insere_h4;
FOR comp_sat: cp_sat USE ENTITY WORK.sat;



SIGNAL inic_vc4_local, clk_tug3_local, clk_tug2_local,
       tu12_just_p, tu12_just_n, au4_just_p, au4_just_n: BIT;



SIGNAL  ind_subq_local, cont_3_tug3,
         cont_7_tug2, cont_3_tu12: INTEGER:=0;

SIGNAL TU12_01, TU12_02, TU12_03, TU12_04, TU12_05, TU12_06, TU12_07,
       TU12_08, TU12_09, TU12_10, TU12_11, TU12_12, TU12_13, TU12_14,
       TU12_15, TU12_16, TU12_17, TU12_18, TU12_19, TU12_20, TU12_21,
       TU12_22, TU12_23, TU12_24, TU12_25, TU12_26, TU12_27, TU12_28,
       TU12_29, TU12_30, TU12_31, TU12_32, TU12_33, TU12_34, TU12_35,
       TU12_36, TU12_37, TU12_38, TU12_39, TU12_40, TU12_41, TU12_42,
       TU12_43, TU12_44, TU12_45, TU12_46, TU12_47, TU12_48, TU12_49,
       TU12_50, TU12_51, TU12_52, TU12_53, TU12_54, TU12_55, TU12_56,
       TU12_57, TU12_58, TU12_59, TU12_60, TU12_61, TU12_62, TU12_63,


       TUG2_01, TUG2_02, TUG2_03, TUG2_04, TUG2_05, TUG2_06, TUG2_07,
       TUG2_08, TUG2_09, TUG2_10, TUG2_11, TUG2_12, TUG2_13, TUG2_14,
       TUG2_15, TUG2_16, TUG2_17, TUG2_18, TUG2_19, TUG2_20, TUG2_21,

       TUG3_1, TUG3_2, TUG3_3, carga_vc4, VC4 : BIT_VECTOR(1 TO 8);
BEGIN



 comp_control_local : cp_control_local
  PORT MAP(Ta,
          linha_local, coluna_local, ind_subq_local, cont_3_tug3,
          cont_7_tug2, cont_3_tu12, ender_tu12_local,
```

```
          inic_vc4_local, clk_tug3_local, clk_tug2_local, clk_tu12_local,
          inic_vc12_local);



comp_adapt_vc12_t_01: cp_adapt_vc12_t
 PORT MAP(VC12_01,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_01);

comp_adapt_vc12_t_02: cp_adapt_vc12_t
 PORT MAP(VC12_02,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_02);

comp_adapt_vc12_t_03: cp_adapt_vc12_t
 PORT MAP(VC12_03,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_03);

comp_adapt_vc12_t_04: cp_adapt_vc12_t
 PORT MAP(VC12_04,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_04);

comp_adapt_vc12_t_05: cp_adapt_vc12_t
 PORT MAP(VC12_05,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_05);

comp_adapt_vc12_t_06: cp_adapt_vc12_t
 PORT MAP(VC12_06,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_06);

comp_adapt_vc12_t_07: cp_adapt_vc12_t
 PORT MAP(VC12_07,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_07);

comp_adapt_vc12_t_08: cp_adapt_vc12_t
 PORT MAP(VC12_08,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_08);

comp_adapt_vc12_t_09: cp_adapt_vc12_t
 PORT MAP(VC12_09,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_09);

comp_adapt_vc12_t_10: cp_adapt_vc12_t
 PORT MAP(VC12_10,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_10);
```

```
 PORT MAP(VC12_22,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_22);

comp_adapt_vc12_t_23: cp_adapt_vc12_t
 PORT MAP(VC12_23,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_23);

comp_adapt_vc12_t_24: cp_adapt_vc12_t
 PORT MAP(VC12_24,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_24);

comp_adapt_vc12_t_25: cp_adapt_vc12_t
 PORT MAP(VC12_25,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_25);

comp_adapt_vc12_t_26: cp_adapt_vc12_t
 PORT MAP(VC12_26,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_26);

comp_adapt_vc12_t_27: cp_adapt_vc12_t
 PORT MAP(VC12_27,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_27);

comp_adapt_vc12_t_28: cp_adapt_vc12_t
 PORT MAP(VC12_28,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_28);

comp_adapt_vc12_t_29: cp_adapt_vc12_t
 PORT MAP(VC12_29,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_29);

comp_adapt_vc12_t_30: cp_adapt_vc12_t
 PORT MAP(VC12_30,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_30);

comp_adapt_vc12_t_31: cp_adapt_vc12_t
 PORT MAP(VC12_31,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_31);

comp_adapt_vc12_t_32: cp_adapt_vc12_t
 PORT MAP(VC12_32,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_32);

comp_adapt_vc12_t_33: cp_adapt_vc12_t
```

```
comp_adapt_vc12_t_11: cp_adapt_vc12_t
 PORT MAP(VC12_11,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_11);

comp_adapt_vc12_t_12: cp_adapt_vc12_t
 PORT MAP(VC12_12,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_12);

comp_adapt_vc12_t_13: cp_adapt_vc12_t
 PORT MAP(VC12_13,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_13);

comp_adapt_vc12_t_14: cp_adapt_vc12_t
 PORT MAP(VC12_14,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_14);

comp_adapt_vc12_t_15: cp_adapt_vc12_t
 PORT MAP(VC12_15,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_15);

comp_adapt_vc12_t_16: cp_adapt_vc12_t
 PORT MAP(VC12_16,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_16);

comp_adapt_vc12_t_17: cp_adapt_vc12_t
 PORT MAP(VC12_17,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_17);

comp_adapt_vc12_t_18: cp_adapt_vc12_t
 PORT MAP(VC12_18,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_18);

comp_adapt_vc12_t_19: cp_adapt_vc12_t
 PORT MAP(VC12_19,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_19);

comp_adapt_vc12_t_20: cp_adapt_vc12_t
 PORT MAP(VC12_20,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_20);

comp_adapt_vc12_t_21: cp_adapt_vc12_t
 PORT MAP(VC12_21,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_21);
comp_adapt_vc12_t_22: cp_adapt_vc12_t
```

```
 PORT MAP(VC12_33,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_33);

comp_adapt_vc12_t_34: cp_adapt_vc12_t
 PORT MAP(VC12_34,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_34);

comp_adapt_vc12_t_35: cp_adapt_vc12_t
 PORT MAP(VC12_35,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_35);

comp_adapt_vc12_t_36: cp_adapt_vc12_t
 PORT MAP(VC12_36,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_36);

comp_adapt_vc12_t_37: cp_adapt_vc12_t
 PORT MAP(VC12_37,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_37);
comp_adapt_vc12_t_38: cp_adapt_vc12_t
 PORT MAP(VC12_38,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_38);

comp_adapt_vc12_t_39: cp_adapt_vc12_t
 PORT MAP(VC12_39,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_39);

comp_adapt_vc12_t_40: cp_adapt_vc12_t
 PORT MAP(VC12_40,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_40);

comp_adapt_vc12_t_41: cp_adapt_vc12_t
 PORT MAP(VC12_41,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_41);

comp_adapt_vc12_t_42: cp_adapt_vc12_t
 PORT MAP(VC12_42,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_42);

comp_adapt_vc12_t_43: cp_adapt_vc12_t
 PORT MAP(VC12_43,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_43);

comp_adapt_vc12_t_44: cp_adapt_vc12_t
 PORT MAP(VC12_44,
```

```
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_44);

comp_adapt_vc12_t_45: cp_adapt_vc12_t
 PORT MAP(VC12_45,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_45);

comp_adapt_vc12_t_46: cp_adapt_vc12_t
 PORT MAP(VC12_46,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_46);

comp_adapt_vc12_t_47: cp_adapt_vc12_t
 PORT MAP(VC12_47,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_47);

comp_adapt_vc12_t_48: cp_adapt_vc12_t
 PORT MAP(VC12_48,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_48);

comp_adapt_vc12_t_49: cp_adapt_vc12_t
 PORT MAP(VC12_49,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_49);

comp_adapt_vc12_t_50: cp_adapt_vc12_t
 PORT MAP(VC12_50,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_50);

comp_adapt_vc12_t_51: cp_adapt_vc12_t
 PORT MAP(VC12_51,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_51);

comp_adapt_vc12_t_52: cp_adapt_vc12_t
 PORT MAP(VC12_52,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_52);

comp_adapt_vc12_t_53: cp_adapt_vc12_t
 PORT MAP(VC12_53,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_53);


comp_adapt_vc12_t_54: cp_adapt_vc12_t
 PORT MAP(VC12_54,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_54);

comp_adapt_vc12_t_55: cp_adapt_vc12_t
```

```
          clk_tug2_local,
          TUG2_02);

comp_mult_tu12_03: cp_mult_tu12
 PORT MAP(TU12_07, TU12_08, TU12_09,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_03);

comp_mult_tu12_04: cp_mult_tu12
 PORT MAP(TU12_10, TU12_11, TU12_12,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_04);

comp_mult_tu12_05: cp_mult_tu12
 PORT MAP(TU12_13, TU12_14, TU12_15,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_05);

comp_mult_tu12_06: cp_mult_tu12
 PORT MAP(TU12_16, TU12_17, TU12_18,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_06);

comp_mult_tu12_07: cp_mult_tu12
 PORT MAP(TU12_19, TU12_20, TU12_21,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_07);

comp_mult_tu12_08: cp_mult_tu12
 PORT MAP(TU12_22, TU12_23, TU12_24,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_08);

comp_mult_tu12_09: cp_mult_tu12
 PORT MAP(TU12_25, TU12_26, TU12_27,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_09);

comp_mult_tu12_10: cp_mult_tu12
 PORT MAP(TU12_28, TU12_29, TU12_30,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_10);

comp_mult_tu12_11: cp_mult_tu12
 PORT MAP(TU12_31, TU12_32, TU12_33,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_11);

comp_mult_tu12_12: cp_mult_tu12
 PORT MAP(TU12_34, TU12_35, TU12_36,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_12);

comp_mult_tu12_13: cp_mult_tu12
 PORT MAP(TU12_37, TU12_38, TU12_39,
          cont_3_tu12,
```

```
 PORT MAP(VC12_55,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_55);

comp_adapt_vc12_t_56: cp_adapt_vc12_t
 PORT MAP(VC12_56,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_56);


comp_adapt_vc12_t_57: cp_adapt_vc12_t
 PORT MAP(VC12_57,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_57);

comp_adapt_vc12_t_58: cp_adapt_vc12_t
 PORT MAP(VC12_58,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_58);

comp_adapt_vc12_t_59: cp_adapt_vc12_t
 PORT MAP(VC12_59,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_59);

comp_adapt_vc12_t_60: cp_adapt_vc12_t
 PORT MAP(VC12_60,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_60);

comp_adapt_vc12_t_61: cp_adapt_vc12_t
 PORT MAP(VC12_61,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_61);

comp_adapt_vc12_t_62: cp_adapt_vc12_t
 PORT MAP(VC12_62,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_62);


comp_adapt_vc12_t_63: cp_adapt_vc12_t
 PORT MAP(VC12_63,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_local, tu12_just_p, tu12_just_n,
          TU12_63);



comp_mult_tu12_01: cp_mult_tu12
 PORT MAP(TU12_01, TU12_02, TU12_03,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_01);

comp_mult_tu12_02: cp_mult_tu12
 PORT MAP(TU12_04, TU12_05, TU12_06,
          cont_3_tu12,
```

```
          clk_tug2_local,
          TUG2_13);

comp_mult_tu12_14: cp_mult_tu12
 PORT MAP(TU12_40, TU12_41, TU12_42,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_14);

comp_mult_tu12_15: cp_mult_tu12
 PORT MAP(TU12_43, TU12_44, TU12_45,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_15);

comp_mult_tu12_16: cp_mult_tu12
 PORT MAP(TU12_46, TU12_47, TU12_48,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_16);

comp_mult_tu12_17: cp_mult_tu12
 PORT MAP(TU12_49, TU12_50, TU12_51,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_17);

comp_mult_tu12_18: cp_mult_tu12
 PORT MAP(TU12_52, TU12_53, TU12_54,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_18);

comp_mult_tu12_19: cp_mult_tu12
 PORT MAP(TU12_55, TU12_56, TU12_57,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_19);

comp_mult_tu12_20: cp_mult_tu12
 PORT MAP(TU12_58, TU12_59, TU12_60,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_20);

comp_mult_tu12_21: cp_mult_tu12
 PORT MAP(TU12_61, TU12_62, TU12_63,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_21);


comp_mult_tug2_01: cp_mult_tug2
 PORT MAP(TUG2_01, TUG2_02, TUG2_03, TUG2_04, TUG2_05, TUG2_06, TUG2_07,
          cont_7_tug2,
          clk_tug3_local,
          TUG3_1);

comp_mult_tug2_02: cp_mult_tug2
 PORT MAP(TUG2_08, TUG2_09, TUG2_10, TUG2_11, TUG2_12, TUG2_13, TUG2_14,
          cont_7_tug2,
          clk_tug3_local,
          TUG3_2);

comp_mult_tug2_03: cp_mult_tug2
 PORT MAP(TUG2_15, TUG2_16, TUG2_17, TUG2_18, TUG2_19, TUG2_20, TUG2_21,
```

```
        cont_7_tug2,
        clk_tug3_local,
        TUG3_3);

comp_mult_tug3: cp_mult_tug3
     PORT MAP(TUG3_1, TUG3_2, TUG3_3,
              cont_3_tug3,
              Ta,
              carga_vc4);

comp_insere_h4: cp_insere_h4
 PORT MAP(carga_vc4,
          linha_local, coluna_local, ind_subq_local,
          Ta,
          VC4);


comp_sat: cp_sat PORT MAP(VC4,
                          linha_local, coluna_local,
                          Ta, inic_vc4_local, au4_just_p, au4_just_n,
                          AU4);



END estrutural;


USE WORK.pac_comp.ALL;
ENTITY equip_b IS
 PORT(STM1: IN BIT_VECTOR(1 TO 8);
      Tb, T1 : IN BIT;
      coluna, linha :IN INTEGER:= 0;
      coluna_local, linha_local, fase: BUFFER INTEGER:=0;
      au4_just_p, au4_just_n : BUFFER BIT;
      AU4: OUT BIT_VECTOR(1 TO 8));
END equip_b;

ARCHITECTURE estrutural OF equip_b IS

FOR comp_control_local: cp_control_local USE ENTITY WORK.control_local;

FOR comp_adapt_vc12_t_01, comp_adapt_vc12_t_02, comp_adapt_vc12_t_03,
    comp_adapt_vc12_t_04, comp_adapt_vc12_t_05, comp_adapt_vc12_t_06,
    comp_adapt_vc12_t_07, comp_adapt_vc12_t_08, comp_adapt_vc12_t_09,
    comp_adapt_vc12_t_10, comp_adapt_vc12_t_11, comp_adapt_vc12_t_12,
    comp_adapt_vc12_t_13, comp_adapt_vc12_t_14, comp_adapt_vc12_t_15,
    comp_adapt_vc12_t_16, comp_adapt_vc12_t_17, comp_adapt_vc12_t_18,
    comp_adapt_vc12_t_19, comp_adapt_vc12_t_20, comp_adapt_vc12_t_21,
    comp_adapt_vc12_t_22, comp_adapt_vc12_t_23, comp_adapt_vc12_t_24,
    comp_adapt_vc12_t_25, comp_adapt_vc12_t_26, comp_adapt_vc12_t_27,
    comp_adapt_vc12_t_28, comp_adapt_vc12_t_29, comp_adapt_vc12_t_30,
    comp_adapt_vc12_t_31, comp_adapt_vc12_t_32, comp_adapt_vc12_t_33,
    comp_adapt_vc12_t_34, comp_adapt_vc12_t_35, comp_adapt_vc12_t_36,
    comp_adapt_vc12_t_37, comp_adapt_vc12_t_38, comp_adapt_vc12_t_39,
    comp_adapt_vc12_t_40, comp_adapt_vc12_t_41, comp_adapt_vc12_t_42,
    comp_adapt_vc12_t_43, comp_adapt_vc12_t_44, comp_adapt_vc12_t_45,
    comp_adapt_vc12_t_46, comp_adapt_vc12_t_47, comp_adapt_vc12_t_48,
    comp_adapt_vc12_t_49, comp_adapt_vc12_t_50, comp_adapt_vc12_t_51,
    comp_adapt_vc12_t_52, comp_adapt_vc12_t_53, comp_adapt_vc12_t_54,
    comp_adapt_vc12_t_55, comp_adapt_vc12_t_56, comp_adapt_vc12_t_57,
    comp_adapt_vc12_t_58, comp_adapt_vc12_t_59, comp_adapt_vc12_t_60,
    comp_adapt_vc12_t_61, comp_adapt_vc12_t_62, comp_adapt_vc12_t_63
                        : cp_adapt_vc12_t USE ENTITY WORK.adapt_vc12_t;

FOR comp_mult_tu12_01, comp_mult_tu12_02, comp_mult_tu12_03,
    comp_mult_tu12_04, comp_mult_tu12_05, comp_mult_tu12_06,
    comp_mult_tu12_07, comp_mult_tu12_08, comp_mult_tu12_09,
    comp_mult_tu12_10, comp_mult_tu12_11, comp_mult_tu12_12,
    comp_mult_tu12_13, comp_mult_tu12_14, comp_mult_tu12_15,
    comp_mult_tu12_16, comp_mult_tu12_17, comp_mult_tu12_18,
    comp_mult_tu12_19, comp_mult_tu12_20, comp_mult_tu12_21
                        : cp_mult_tu12 USE ENTITY WORK.mult_tu12;

FOR comp_mult_tug2_1, comp_mult_tug2_2, comp_mult_tug2_3
                        : cp_mult_tug2 USE ENTITY WORK.mult_tug2;


FOR comp_mult_tug3 : cp_mult_tug3 USE ENTITY WORK.mult_tug3;
FOR comp_insere_h4: cp_insere_h4 USE ENTITY WORK.insere_h4;
FOR comp_sat: cp_sat USE ENTITY WORK.sat;

FOR comp_adapt_vc12_r_01, comp_adapt_vc12_r_02, comp_adapt_vc12_r_03,
    comp_adapt_vc12_r_04, comp_adapt_vc12_r_05, comp_adapt_vc12_r_06,
    comp_adapt_vc12_r_07, comp_adapt_vc12_r_08, comp_adapt_vc12_r_09,
    comp_adapt_vc12_r_10, comp_adapt_vc12_r_11, comp_adapt_vc12_r_12,
    comp_adapt_vc12_r_13, comp_adapt_vc12_r_14, comp_adapt_vc12_r_15,
    comp_adapt_vc12_r_16, comp_adapt_vc12_r_17, comp_adapt_vc12_r_18,
    comp_adapt_vc12_r_19, comp_adapt_vc12_r_20, comp_adapt_vc12_r_21,
    comp_adapt_vc12_r_22, comp_adapt_vc12_r_23, comp_adapt_vc12_r_24,
    comp_adapt_vc12_r_25, comp_adapt_vc12_r_26, comp_adapt_vc12_r_27,
    comp_adapt_vc12_r_28, comp_adapt_vc12_r_29, comp_adapt_vc12_r_30,
    comp_adapt_vc12_r_31, comp_adapt_vc12_r_32, comp_adapt_vc12_r_33,
    comp_adapt_vc12_r_34, comp_adapt_vc12_r_35, comp_adapt_vc12_r_36,
    comp_adapt_vc12_r_37, comp_adapt_vc12_r_38, comp_adapt_vc12_r_39,
    comp_adapt_vc12_r_40, comp_adapt_vc12_r_41, comp_adapt_vc12_r_42,
    comp_adapt_vc12_r_43, comp_adapt_vc12_r_44, comp_adapt_vc12_r_45,
    comp_adapt_vc12_r_46, comp_adapt_vc12_r_47, comp_adapt_vc12_r_48,
    comp_adapt_vc12_r_49, comp_adapt_vc12_r_50, comp_adapt_vc12_r_51,
    comp_adapt_vc12_r_52, comp_adapt_vc12_r_53, comp_adapt_vc12_r_54,
    comp_adapt_vc12_r_55, comp_adapt_vc12_r_56, comp_adapt_vc12_r_57,
    comp_adapt_vc12_r_58, comp_adapt_vc12_r_59, comp_adapt_vc12_r_60,
    comp_adapt_vc12_r_61, comp_adapt_vc12_r_62, comp_adapt_vc12_r_63
                        : cp_adapt_vc12_r USE ENTITY WORK.adapt_vc12_r;



FOR comp_dmult_vc4 : cp_dmult_vc4 USE ENTITY WORK.dmult_vc4;

FOR comp_dmult_tug2_01, comp_dmult_tug2_02, comp_dmult_tug2_03,
    comp_dmult_tug2_04, comp_dmult_tug2_05, comp_dmult_tug2_06,
    comp_dmult_tug2_07, comp_dmult_tug2_08, comp_dmult_tug2_09,
    comp_dmult_tug2_10, comp_dmult_tug2_11, comp_dmult_tug2_12,
    comp_dmult_tug2_13, comp_dmult_tug2_14, comp_dmult_tug2_15,
    comp_dmult_tug2_16, comp_dmult_tug2_17, comp_dmult_tug2_18,
    comp_dmult_tug2_19, comp_dmult_tug2_20, comp_dmult_tug2_21
                        : cp_dmult_tug2 USE ENTITY WORK.dmult_tug2;


FOR comp_dmult_tug3_1, comp_dmult_tug3_2, comp_dmult_tug3_3
                        : cp_dmult_tug3 USE ENTITY WORK.dmult_tug3;

FOR comp_control_vc4: cp_control_vc4 USE ENTITY WORK.control_vc4;

FOR comp_sar: cp_sar USE ENTITY WORK.sar;

SIGNAL inic_vc4_local, clk_tug3_local, clk_tug2_local, clk_tu12_local,
       inic_vc12_local, au4_just_p_terra, au4_just_n_terra,
       inic_vc4, clk_vc4,

       clk_tug3_1_d, clk_tug3_2_d, clk_tug3_3_d,

       clk_tug2_01_d, clk_tug2_02_d, clk_tug2_03_d,
       clk_tug2_04_d, clk_tug2_05_d, clk_tug2_06_d,
       clk_tug2_07_d, clk_tug2_08_d, clk_tug2_09_d,
       clk_tug2_10_d, clk_tug2_11_d, clk_tug2_12_d,
       clk_tug2_13_d, clk_tug2_14_d, clk_tug2_15_d,
       clk_tug2_16_d, clk_tug2_17_d, clk_tug2_18_d,
       clk_tug2_19_d, clk_tug2_20_d, clk_tug2_21_d,


       clk_tu12_01_d, clk_tu12_02_d, clk_tu12_03_d,
       clk_tu12_04_d, clk_tu12_05_d, clk_tu12_06_d,
       clk_tu12_07_d, clk_tu12_08_d, clk_tu12_09_d,
       clk_tu12_10_d, clk_tu12_11_d, clk_tu12_12_d,
       clk_tu12_13_d, clk_tu12_14_d, clk_tu12_15_d,
       clk_tu12_16_d, clk_tu12_17_d, clk_tu12_18_d,
       clk_tu12_19_d, clk_tu12_20_d, clk_tu12_21_d,
       clk_tu12_22_d, clk_tu12_23_d, clk_tu12_24_d,
       clk_tu12_25_d, clk_tu12_26_d, clk_tu12_27_d,
       clk_tu12_28_d, clk_tu12_29_d, clk_tu12_30_d,
       clk_tu12_31_d, clk_tu12_32_d, clk_tu12_33_d,
       clk_tu12_34_d, clk_tu12_35_d, clk_tu12_36_d,
       clk_tu12_37_d, clk_tu12_38_d, clk_tu12_39_d,
       clk_tu12_40_d, clk_tu12_41_d, clk_tu12_42_d,
       clk_tu12_43_d, clk_tu12_44_d, clk_tu12_45_d,
       clk_tu12_46_d, clk_tu12_47_d, clk_tu12_48_d,
       clk_tu12_49_d, clk_tu12_50_d, clk_tu12_51_d,
       clk_tu12_52_d, clk_tu12_53_d, clk_tu12_54_d,
       clk_tu12_55_d, clk_tu12_56_d, clk_tu12_57_d,
       clk_tu12_58_d, clk_tu12_59_d, clk_tu12_60_d,
       clk_tu12_61_d, clk_tu12_62_d, clk_tu12_63_d,
       tu12_just_p_01, tu12_just_n_01, inic_vc12_01, clk_vc12_01,
       tu12_just_p_02, tu12_just_n_02, inic_vc12_02, clk_vc12_02,
       tu12_just_p_03, tu12_just_n_03, inic_vc12_03, clk_vc12_03,
       tu12_just_p_04, tu12_just_n_04, inic_vc12_04, clk_vc12_04,
       tu12_just_p_05, tu12_just_n_05, inic_vc12_05, clk_vc12_05,
       tu12_just_p_06, tu12_just_n_06, inic_vc12_06, clk_vc12_06,
       tu12_just_p_07, tu12_just_n_07, inic_vc12_07, clk_vc12_07,
       tu12_just_p_08, tu12_just_n_08, inic_vc12_08, clk_vc12_08,
       tu12_just_p_09, tu12_just_n_09, inic_vc12_09, clk_vc12_09,
       tu12_just_p_10, tu12_just_n_10, inic_vc12_10, clk_vc12_10,
       tu12_just_p_11, tu12_just_n_11, inic_vc12_11, clk_vc12_11,
       tu12_just_p_12, tu12_just_n_12, inic_vc12_12, clk_vc12_12,
       tu12_just_p_13, tu12_just_n_13, inic_vc12_13, clk_vc12_13,
       tu12_just_p_14, tu12_just_n_14, inic_vc12_14, clk_vc12_14,
       tu12_just_p_15, tu12_just_n_15, inic_vc12_15, clk_vc12_15,
       tu12_just_p_16, tu12_just_n_16, inic_vc12_16, clk_vc12_16,
       tu12_just_p_17, tu12_just_n_17, inic_vc12_17, clk_vc12_17,
       tu12_just_p_18, tu12_just_n_18, inic_vc12_18, clk_vc12_18,
       tu12_just_p_19, tu12_just_n_19, inic_vc12_19, clk_vc12_19,
       tu12_just_p_20, tu12_just_n_20, inic_vc12_20, clk_vc12_20,
       tu12_just_p_21, tu12_just_n_21, inic_vc12_21, clk_vc12_21,
       tu12_just_p_22, tu12_just_n_22, inic_vc12_22, clk_vc12_22,
       tu12_just_p_23, tu12_just_n_23, inic_vc12_23, clk_vc12_23,
       tu12_just_p_24, tu12_just_n_24, inic_vc12_24, clk_vc12_24,
       tu12_just_p_25, tu12_just_n_25, inic_vc12_25, clk_vc12_25,
       tu12_just_p_26, tu12_just_n_26, inic_vc12_26, clk_vc12_26,
       tu12_just_p_27, tu12_just_n_27, inic_vc12_27, clk_vc12_27,
       tu12_just_p_28, tu12_just_n_28, inic_vc12_28, clk_vc12_28,
       tu12_just_p_29, tu12_just_n_29, inic_vc12_29, clk_vc12_29,
       tu12_just_p_30, tu12_just_n_30, inic_vc12_30, clk_vc12_30,
       tu12_just_p_31, tu12_just_n_31, inic_vc12_31, clk_vc12_31,
       tu12_just_p_32, tu12_just_n_32, inic_vc12_32, clk_vc12_32,
       tu12_just_p_33, tu12_just_n_33, inic_vc12_33, clk_vc12_33,
       tu12_just_p_34, tu12_just_n_34, inic_vc12_34, clk_vc12_34,
       tu12_just_p_35, tu12_just_n_35, inic_vc12_35, clk_vc12_35,
       tu12_just_p_36, tu12_just_n_36, inic_vc12_36, clk_vc12_36,
       tu12_just_p_37, tu12_just_n_37, inic_vc12_37, clk_vc12_37,
       tu12_just_p_38, tu12_just_n_38, inic_vc12_38, clk_vc12_38,
       tu12_just_p_39, tu12_just_n_39, inic_vc12_39, clk_vc12_39,
       tu12_just_p_40, tu12_just_n_40, inic_vc12_40, clk_vc12_40,
       tu12_just_p_41, tu12_just_n_41, inic_vc12_41, clk_vc12_41,
       tu12_just_p_42, tu12_just_n_42, inic_vc12_42, clk_vc12_42,
       tu12_just_p_43, tu12_just_n_43, inic_vc12_43, clk_vc12_43,
       tu12_just_p_44, tu12_just_n_44, inic_vc12_44, clk_vc12_44,
       tu12_just_p_45, tu12_just_n_45, inic_vc12_45, clk_vc12_45,
       tu12_just_p_46, tu12_just_n_46, inic_vc12_46, clk_vc12_46,
       tu12_just_p_47, tu12_just_n_47, inic_vc12_47, clk_vc12_47,
       tu12_just_p_48, tu12_just_n_48, inic_vc12_48, clk_vc12_48,
       tu12_just_p_49, tu12_just_n_49, inic_vc12_49, clk_vc12_49,
       tu12_just_p_50, tu12_just_n_50, inic_vc12_50, clk_vc12_50,
       tu12_just_p_51, tu12_just_n_51, inic_vc12_51, clk_vc12_51,
       tu12_just_p_52, tu12_just_n_52, inic_vc12_52, clk_vc12_52,
       tu12_just_p_53, tu12_just_n_53, inic_vc12_53, clk_vc12_53,
       tu12_just_p_54, tu12_just_n_54, inic_vc12_54, clk_vc12_54,
       tu12_just_p_55, tu12_just_n_55, inic_vc12_55, clk_vc12_55,
       tu12_just_p_56, tu12_just_n_56, inic_vc12_56, clk_vc12_56,
       tu12_just_p_57, tu12_just_n_57, inic_vc12_57, clk_vc12_57,
       tu12_just_p_58, tu12_just_n_58, inic_vc12_58, clk_vc12_58,
       tu12_just_p_59, tu12_just_n_59, inic_vc12_59, clk_vc12_59,
       tu12_just_p_60, tu12_just_n_60, inic_vc12_60, clk_vc12_60,
       tu12_just_p_61, tu12_just_n_61, inic_vc12_61, clk_vc12_61,
       tu12_just_p_62, tu12_just_n_62, inic_vc12_62, clk_vc12_62,
       tu12_just_p_63, tu12_just_n_63, inic_vc12_63, clk_vc12_63 : BIT;


SIGNAL  ind_subq_local, cont_3_tug3, cont_7_tug2, cont_3_tu12, ender_tu12_loc
        linha_vc4, coluna_vc4, ind_subq,

        fase01, fase02, fase03, fase04, fase05, fase06, fase07,
        fase08, fase09, fase10, fase11, fase12, fase13, fase14,
        fase15, fase16, fase17, fase18, fase19, fase20, fase21,
        fase22, fase23, fase24, fase25, fase26, fase27, fase28,
        fase29, fase30, fase31, fase32, fase33, fase34, fase35,
        fase36, fase37, fase38, fase39, fase40, fase41, fase42,
        fase43, fase44, fase45, fase46, fase47, fase48, fase49,
        fase50, fase51, fase52, fase53, fase54, fase55, fase56,
        fase57, fase58, fase59, fase60, fase61, fase62, fase63: INTEGER:=0;

SIGNAL VC12_01, VC12_02, VC12_03,VC12_04, VC12_05, VC12_06, VC12_07,
       VC12_08, VC12_09, VC12_10,VC12_11, VC12_12, VC12_13, VC12_14,
       VC12_15, VC12_16, VC12_17,VC12_18, VC12_19, VC12_20, VC12_21,
       VC12_22, VC12_23, VC12_24,VC12_25, VC12_26, VC12_27, VC12_28,
       VC12_29, VC12_30, VC12_31,VC12_32, VC12_33, VC12_34, VC12_35,
       VC12_36, VC12_37, VC12_38,VC12_39, VC12_40, VC12_41, VC12_42,
       VC12_43, VC12_44, VC12_45,VC12_46, VC12_47, VC12_48, VC12_49,
       VC12_50, VC12_51, VC12_52,VC12_53, VC12_54, VC12_55, VC12_56,
       WC12_57, VC12_58, VC12_59,VC12_60, VC12_61, VC12_62, VC12_63,

       TU12_01, TU12_02, TU12_03, TU12_04, TU12_05, TU12_06, TU12_07,
       TU12_08, TU12_09, TU12_10, TU12_11, TU12_12, TU12_13, TU12_14,
       TU12_15, TU12_16, TU12_17, TU12_18, TU12_19, TU12_20, TU12_21,
       TU12_22, TU12_23, TU12_24, TU12_25, TU12_26, TU12_27, TU12_28,
       TU12_29, TU12_30, TU12_31, TU12_32, TU12_33, TU12_34, TU12_35,
       TU12_36, TU12_37, TU12_38, TU12_39, TU12_40, TU12_41, TU12_42,
       TU12_43, TU12_44, TU12_45, TU12_46, TU12_47, TU12_48, TU12_49,
       TU12_50, TU12_51, TU12_52, TU12_53, TU12_54, TU12_55, TU12_56,
       TU12_57, TU12_58, TU12_59, TU12_60, TU12_61, TU12_62, TU12_63,

       TUG2_01, TUG2_02, TUG2_03, TUG2_04, TUG2_05, TUG2_06, TUG2_07,
       TUG2_08, TUG2_09, TUG2_10, TUG2_11, TUG2_12, TUG2_13, TUG2_14,
       TUG2_15, TUG2_16, TUG2_17, TUG2_18, TUG2_19, TUG2_20, TUG2_21,
```

```
      TUG2_1_d, TUG2_2_d, TUG2_3_d, TUG2_4_d, TUG2_5_d, TUG2_6_d, TUG2_7_d,
      TUG3_1, TUG3_2, TUG3_3,
      TUG3_1_d, TUG3_2_d, TUG3_3_d,
      carga_vc4, VC4, VC4_d : BIT_VECTOR(1 TO 8);

BEGIN


 comp_control_local : cp_control_local
  PORT MAP(Tb,
           linha_local, coluna_local, ind_subq_local, cont_3_tug3,
           cont_7_tug2, cont_3_tu12, ender_tu12_local,
           inic_vc4_local, clk_tug3_local, clk_tug2_local, clk_tu12_local,
           inic_vc12_local);

 comp_sar: cp_sar
  PORT MAP (STM1,
            linha, coluna, linha_local, coluna_local,
            Tb, T1,
            au4_just_p, au4_just_n, inic_vc4, clk_vc4,
            fase,
            VC4_d);

 comp_control_vc4: cp_control_vc4
  PORT MAP(VC4_d,
           inic_vc4, clk_vc4,
           linha_vc4, coluna_vc4,
           ind_subq);


 comp_dmult_vc4: cp_dmult_vc4
  PORT MAP(VC4_d,
           clk_vc4, inic_vc4,
           coluna_vc4,
           TUG3_1_d, TUG3_2_d, TUG3_3_d,
           clk_tug3_1_d, clk_tug3_2_d, clk_tug3_3_d);

 comp_dmult_tug3_1: cp_dmult_tug3
   PORT MAP(TUG3_1_d,
            coluna_vc4,
            clk_tug3_1_d,
            TUG2_01_d, TUG2_02_d, TUG2_03_d, TUG2_04_d, TUG2_05_d, TUG2_06_d,
            TUG2_07_d,
            clk_tug2_01_d, clk_tug2_02_d, clk_tug2_03_d, clk_tug2_04_d,
            clk_tug2_05_d, clk_tug2_06_d, clk_tug2_07_d);

 comp_dmult_tug3_2: cp_dmult_tug3
   PORT MAP(TUG3_1_d,
            coluna_vc4,
            clk_tug3_2_d,
            TUG2_08_d, TUG2_09_d, TUG2_10_d, TUG2_11_d, TUG2_12_d, TUG2_13_d,
            TUG2_14_d,
            clk_tug2_08_d, clk_tug2_09_d, clk_tug2_10_d, clk_tug2_11_d,
            clk_tug2_12_d, clk_tug2_13_d, clk_tug2_14_d);

 comp_dmult_tug3_3: cp_dmult_tug3
   PORT MAP(TUG3_1_d,
            coluna_vc4,
            clk_tug3_3_d,
            TUG2_15_d, TUG2_16_d, TUG2_17_d, TUG2_18_d, TUG2_19_d, TUG2_20_d,
            TUG2_21_d,
            clk_tug2_15_d, clk_tug2_16_d, clk_tug2_17_d, clk_tug2_18_d,
            clk_tug2_19_d, clk_tug2_20_d, clk_tug2_21_d);

 comp_dmult_tug2_01: cp_dmult_tug2
  PORT MAP(clk_tug2_01_d, inic_vc4,
           clk_tu12_01_d, clk_tu12_02_d, clk_tu12_03_d);

 comp_dmult_tug2_02: cp_dmult_tug2
  PORT MAP(clk_tug2_02_d, inic_vc4,
           clk_tu12_04_d, clk_tu12_05_d, clk_tu12_06_d);

 comp_dmult_tug2_03: cp_dmult_tug2
  PORT MAP(clk_tug2_03_d, inic_vc4,
           clk_tu12_07_d, clk_tu12_08_d, clk_tu12_09_d);

 comp_dmult_tug2_04: cp_dmult_tug2
  PORT MAP(clk_tug2_04_d, inic_vc4,
           clk_tu12_10_d, clk_tu12_11_d, clk_tu12_12_d);

 comp_dmult_tug2_05: cp_dmult_tug2
  PORT MAP(clk_tug2_05_d, inic_vc4,
           clk_tu12_13_d, clk_tu12_14_d, clk_tu12_15_d);

 comp_dmult_tug2_06: cp_dmult_tug2
  PORT MAP(clk_tug2_06_d, inic_vc4,
           clk_tu12_16_d, clk_tu12_17_d, clk_tu12_18_d);

 comp_dmult_tug2_07: cp_dmult_tug2
  PORT MAP(clk_tug2_07_d, inic_vc4,
           clk_tu12_19_d, clk_tu12_20_d, clk_tu12_21_d);

 comp_dmult_tug2_08: cp_dmult_tug2
  PORT MAP(clk_tug2_08_d, inic_vc4,
           clk_tu12_22_d, clk_tu12_23_d, clk_tu12_24_d);

 comp_dmult_tug2_09: cp_dmult_tug2
  PORT MAP(clk_tug2_09_d, inic_vc4,
           clk_tu12_25_d, clk_tu12_26_d, clk_tu12_27_d);

 comp_dmult_tug2_10: cp_dmult_tug2
  PORT MAP(clk_tug2_10_d, inic_vc4,
           clk_tu12_28_d, clk_tu12_29_d, clk_tu12_30_d);

 comp_dmult_tug2_11: cp_dmult_tug2
  PORT MAP(clk_tug2_11_d, inic_vc4,
           clk_tu12_31_d, clk_tu12_32_d, clk_tu12_33_d);

 comp_dmult_tug2_12: cp_dmult_tug2
  PORT MAP(clk_tug2_12_d, inic_vc4,
           clk_tu12_34_d, clk_tu12_35_d, clk_tu12_36_d);

 comp_dmult_tug2_13: cp_dmult_tug2
  PORT MAP(clk_tug2_13_d, inic_vc4,
           clk_tu12_37_d, clk_tu12_38_d, clk_tu12_39_d);

 comp_dmult_tug2_14: cp_dmult_tug2
  PORT MAP(clk_tug2_14_d, inic_vc4,
           clk_tu12_40_d, clk_tu12_41_d, clk_tu12_42_d);

 comp_dmult_tug2_15: cp_dmult_tug2
  PORT MAP(clk_tug2_15_d, inic_vc4,
           clk_tu12_43_d, clk_tu12_44_d, clk_tu12_45_d);

 comp_dmult_tug2_16: cp_dmult_tug2
  PORT MAP(clk_tug2_16_d, inic_vc4,
           clk_tu12_46_d, clk_tu12_47_d, clk_tu12_48_d);

 comp_dmult_tug2_17: cp_dmult_tug2
  PORT MAP(clk_tug2_17_d, inic_vc4,
           clk_tu12_49_d, clk_tu12_50_d, clk_tu12_51_d);

 comp_dmult_tug2_18: cp_dmult_tug2
  PORT MAP(clk_tug2_18_d, inic_vc4,
           clk_tu12_52_d, clk_tu12_53_d, clk_tu12_54_d);

 comp_dmult_tug2_19: cp_dmult_tug2
  PORT MAP(clk_tug2_19_d, inic_vc4,
           clk_tu12_55_d, clk_tu12_56_d, clk_tu12_57_d);

 comp_dmult_tug2_20: cp_dmult_tug2
  PORT MAP(clk_tug2_20_d, inic_vc4,
           clk_tu12_58_d, clk_tu12_59_d, clk_tu12_60_d);

 comp_dmult_tug2_21: cp_dmult_tug2
  PORT MAP(clk_tug2_21_d, inic_vc4,
           clk_tu12_61_d, clk_tu12_62_d, clk_tu12_63_d);


 comp_adapt_vc12_r_01: cp_adapt_vc12_r
  PORT MAP(TUG2_01_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_01_d, clk_tu12_local,
           tu12_just_p_01, tu12_just_n_01, inic_vc12_01, clk_vc12_01,
           fase01,
           VC12_01);

 comp_adapt_vc12_r_02: cp_adapt_vc12_r
  PORT MAP(TUG2_01_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_02_d, clk_tu12_local,
           tu12_just_p_02, tu12_just_n_02, inic_vc12_02, clk_vc12_02,
           fase02,
           VC12_02);

 comp_adapt_vc12_r_03: cp_adapt_vc12_r
  PORT MAP(TUG2_01_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_03_d, clk_tu12_local,
           tu12_just_p_03, tu12_just_n_03, inic_vc12_03, clk_vc12_03,
           fase03,
           VC12_03);

 comp_adapt_vc12_r_04: cp_adapt_vc12_r
  PORT MAP(TUG2_02_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_04_d, clk_tu12_local,
           tu12_just_p_04, tu12_just_n_04, inic_vc12_04, clk_vc12_04,
           fase04,
           VC12_04);

 comp_adapt_vc12_r_05: cp_adapt_vc12_r
  PORT MAP(TUG2_02_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_05_d, clk_tu12_local,
           tu12_just_p_05, tu12_just_n_05, inic_vc12_05, clk_vc12_05,
           fase05,
           VC12_05);

 comp_adapt_vc12_r_06: cp_adapt_vc12_r
  PORT MAP(TUG2_02_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_06_d, clk_tu12_local,
           tu12_just_p_06, tu12_just_n_06, inic_vc12_06, clk_vc12_06,
           fase06,
           VC12_06);

 comp_adapt_vc12_r_07: cp_adapt_vc12_r
  PORT MAP(TUG2_03_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_0731_d, clk_tu12_local,
           tu12_just_p_07, tu12_just_n_07, inic_vc12_07, clk_vc12_07,
           fase07,
           VC12_07);

 comp_adapt_vc12_r_08: cp_adapt_vc12_r
  PORT MAP(TUG2_03_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_08_d, clk_tu12_local,
           tu12_just_p_08, tu12_just_n_08, inic_vc12_08, clk_vc12_08,
           fase08,
           VC12_08);

 comp_adapt_vc12_r_09: cp_adapt_vc12_r
  PORT MAP(TUG2_03_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_09_d, clk_tu12_local,
           tu12_just_p_09, tu12_just_n_09, inic_vc12_09, clk_vc12_09,
           fase09,
           VC12_09);

 comp_adapt_vc12_r_10: cp_adapt_vc12_r
  PORT MAP(TUG2_04_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_1041_d, clk_tu12_local,
           tu12_just_p_10, tu12_just_n_10, inic_vc12_10, clk_vc12_10,
           fase10,
           VC12_10);

 comp_adapt_vc12_r_11: cp_adapt_vc12_r
  PORT MAP(TUG2_04_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_11_d, clk_tu12_local,
           tu12_just_p_11, tu12_just_n_11, inic_vc12_11, clk_vc12_11,
           fase11,
           VC12_11);

 comp_adapt_vc12_r_12: cp_adapt_vc12_r
  PORT MAP(TUG2_04_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_12_d, clk_tu12_local,
           tu12_just_p_12, tu12_just_n_12, inic_vc12_12, clk_vc12_12,
           fase12,
           VC12_12);


 comp_adapt_vc12_r_13: cp_adapt_vc12_r
  PORT MAP(TUG2_05_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_13_d, clk_tu12_local,
           tu12_just_p_13, tu12_just_n_13, inic_vc12_13, clk_vc12_13,
           fase13,
           VC12_13);

 comp_adapt_vc12_r_14: cp_adapt_vc12_r
  PORT MAP(TUG2_05_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_14_d, clk_tu12_local,
           tu12_just_p_14, tu12_just_n_14, inic_vc12_14, clk_vc12_14,
           fase14,
           VC12_14);

 comp_adapt_vc12_r_15: cp_adapt_vc12_r
  PORT MAP(TUG2_05_d,
```

```
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_15_d, clk_tu12_local,
          tu12_just_p_15, tu12_just_n_15, inic_vc12_15, clk_vc12_15,
          fase15,
          VC12_15);

 comp_adapt_vc12_r_16: cp_adapt_vc12_r
  PORT MAP(TUG2_06_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_16_d, clk_tu12_local,
          tu12_just_p_16, tu12_just_n_16, inic_vc12_16, clk_vc12_16,
          fase16,
          VC12_16);

comp_adapt_vc12_r_17: cp_adapt_vc12_r
  PORT MAP(TUG2_06_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_17_d, clk_tu12_local,
          tu12_just_p_17, tu12_just_n_17, inic_vc12_17, clk_vc12_17,
          fase17,
          VC12_17);

comp_adapt_vc12_r_18: cp_adapt_vc12_r
  PORT MAP(TUG2_06_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_18_d, clk_tu12_local,
          tu12_just_p_18, tu12_just_n_18, inic_vc12_18, clk_vc12_18,
          fase18,
          VC12_18);

 comp_adapt_vc12_r_19: cp_adapt_vc12_r
  PORT MAP(TUG2_07_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_19_d, clk_tu12_local,
          tu12_just_p_19, tu12_just_n_19, inic_vc12_19, clk_vc12_19,
          fase19,
          VC12_19);

comp_adapt_vc12_r_20: cp_adapt_vc12_r
  PORT MAP(TUG2_07_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_20_d, clk_tu12_local,
          tu12_just_p_20, tu12_just_n_20, inic_vc12_20, clk_vc12_20,
          fase20,
          VC12_20);

comp_adapt_vc12_r_21: cp_adapt_vc12_r
  PORT MAP(TUG2_07_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_21_d, clk_tu12_local,
          tu12_just_p_21, tu12_just_n_21, inic_vc12_21, clk_vc12_21,
          fase21,
          VC12_21);

comp_adapt_vc12_r_22: cp_adapt_vc12_r
  PORT MAP(TUG2_08_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_22_d, clk_tu12_local,
          tu12_just_p_22, tu12_just_n_22, inic_vc12_22, clk_vc12_22,
          fase22,
          VC12_22);

comp_adapt_vc12_r_23: cp_adapt_vc12_r
  PORT MAP(TUG2_08_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_23_d, clk_tu12_local,
          tu12_just_p_23, tu12_just_n_23, inic_vc12_23, clk_vc12_23,
          fase23,
          VC12_23);

 comp_adapt_vc12_r_24: cp_adapt_vc12_r
  PORT MAP(TUG2_08_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_24_d, clk_tu12_local,
          tu12_just_p_24, tu12_just_n_24, inic_vc12_24, clk_vc12_24,
          fase24,
          VC12_24);

comp_adapt_vc12_r_25: cp_adapt_vc12_r
  PORT MAP(TUG2_09_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_25_d, clk_tu12_local,
          tu12_just_p_25, tu12_just_n_25, inic_vc12_25, clk_vc12_25,
          fase25,
          VC12_25);

comp_adapt_vc12_r_26: cp_adapt_vc12_r
  PORT MAP(TUG2_09_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_26_d, clk_tu12_local,
          tu12_just_p_26, tu12_just_n_26, inic_vc12_26, clk_vc12_26,
          fase26,
          VC12_26);

 comp_adapt_vc12_r_27: cp_adapt_vc12_r
  PORT MAP(TUG2_09_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_2731_d, clk_tu12_local,
          tu12_just_p_27, tu12_just_n_27, inic_vc12_27, clk_vc12_27,
          fase27,
          VC12_27);

comp_adapt_vc12_r_28: cp_adapt_vc12_r
  PORT MAP(TUG2_10_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_28_d, clk_tu12_local,
          tu12_just_p_28, tu12_just_n_28, inic_vc12_28, clk_vc12_28,
          fase28,
          VC12_28);

comp_adapt_vc12_r_29: cp_adapt_vc12_r
  PORT MAP(TUG2_10_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_29_d, clk_tu12_local,
          tu12_just_p_29, tu12_just_n_29, inic_vc12_29, clk_vc12_29,
          fase29,
          VC12_29);

 comp_adapt_vc12_r_30: cp_adapt_vc12_r
  PORT MAP(TUG2_10_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_3041_d, clk_tu12_local,
          tu12_just_p_30, tu12_just_n_30, inic_vc12_30, clk_vc12_30,
          fase30,
          VC12_30);

comp_adapt_vc12_r_31: cp_adapt_vc12_r
  PORT MAP(TUG2_11_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_31_d, clk_tu12_local,
          tu12_just_p_31, tu12_just_n_31, inic_vc12_31, clk_vc12_31,
          fase31,
          VC12_31);

comp_adapt_vc12_r_32: cp_adapt_vc12_r
  PORT MAP(TUG2_11_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_32_d, clk_tu12_local,
          tu12_just_p_32, tu12_just_n_32, inic_vc12_32, clk_vc12_32,
          fase32,
          VC12_32);


 comp_adapt_vc12_r_33: cp_adapt_vc12_r
  PORT MAP(TUG2_11_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_33_d, clk_tu12_local,
          tu12_just_p_33, tu12_just_n_33, inic_vc12_33, clk_vc12_33,
          fase33,
          VC12_33);

comp_adapt_vc12_r_34: cp_adapt_vc12_r
  PORT MAP(TUG2_12_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_34_d, clk_tu12_local,
          tu12_just_p_34, tu12_just_n_34, inic_vc12_34, clk_vc12_34,
          fase34,
          VC12_34);

comp_adapt_vc12_r_35: cp_adapt_vc12_r
  PORT MAP(TUG2_12_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_35_d, clk_tu12_local,
          tu12_just_p_35, tu12_just_n_35, inic_vc12_35, clk_vc12_35,
          fase35,
          VC12_35);

 comp_adapt_vc12_r_36: cp_adapt_vc12_r
  PORT MAP(TUG2_12_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_36_d, clk_tu12_local,
          tu12_just_p_36, tu12_just_n_36, inic_vc12_36, clk_vc12_36,
          fase36,
          VC12_36);

comp_adapt_vc12_r_37: cp_adapt_vc12_r
  PORT MAP(TUG2_13_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_37_d, clk_tu12_local,
          tu12_just_p_37, tu12_just_n_37, inic_vc12_37, clk_vc12_37,
          fase37,
          VC12_37);

comp_adapt_vc12_r_38: cp_adapt_vc12_r
  PORT MAP(TUG2_13_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_38_d, clk_tu12_local,
          tu12_just_p_38, tu12_just_n_38, inic_vc12_38, clk_vc12_38,
          fase38,
          VC12_38);

 comp_adapt_vc12_r_39: cp_adapt_vc12_r
  PORT MAP(TUG2_13_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_39_d, clk_tu12_local,
          tu12_just_p_39, tu12_just_n_39, inic_vc12_39, clk_vc12_39,
          fase39,
          VC12_39);

comp_adapt_vc12_r_40: cp_adapt_vc12_r
  PORT MAP(TUG2_14_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_40_d, clk_tu12_local,
          tu12_just_p_40, tu12_just_n_40, inic_vc12_40, clk_vc12_40,
          fase40,
          VC12_40);

comp_adapt_vc12_r_41: cp_adapt_vc12_r
  PORT MAP(TUG2_14_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_41_d, clk_tu12_local,
          tu12_just_p_41, tu12_just_n_41, inic_vc12_41, clk_vc12_41,
          fase41,
          VC12_41);

comp_adapt_vc12_r_42: cp_adapt_vc12_r
  PORT MAP(TUG2_14_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_42_d, clk_tu12_local,
          tu12_just_p_42, tu12_just_n_42, inic_vc12_42, clk_vc12_42,
          fase42,
          VC12_42);

comp_adapt_vc12_r_43: cp_adapt_vc12_r
  PORT MAP(TUG2_15_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_43_d, clk_tu12_local,
          tu12_just_p_43, tu12_just_n_43, inic_vc12_43, clk_vc12_43,
          fase43,
          VC12_43);

 comp_adapt_vc12_r_44: cp_adapt_vc12_r
  PORT MAP(TUG2_15_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_44_d, clk_tu12_local,
          tu12_just_p_44, tu12_just_n_44, inic_vc12_44, clk_vc12_44,
          fase44,
          VC12_44);

comp_adapt_vc12_r_45: cp_adapt_vc12_r
  PORT MAP(TUG2_15_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_45_d, clk_tu12_local,
          tu12_just_p_45, tu12_just_n_45, inic_vc12_45, clk_vc12_45,
          fase45,
          VC12_45);

comp_adapt_vc12_r_46: cp_adapt_vc12_r
  PORT MAP(TUG2_16_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_46_d, clk_tu12_local,
          tu12_just_p_46, tu12_just_n_46, inic_vc12_46, clk_vc12_46,
          fase46,
          VC12_46);

 comp_adapt_vc12_r_47: cp_adapt_vc12_r
  PORT MAP(TUG2_16_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_4731_d, clk_tu12_local,
          tu12_just_p_47, tu12_just_n_47, inic_vc12_47, clk_vc12_47,
          fase47,
          VC12_47);

comp_adapt_vc12_r_48: cp_adapt_vc12_r
```

```
  PORT MAP(TUG2_16_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_48_d, clk_tu12_local,
           tu12_just_p_48, tu12_just_n_48, inic_vc12_48, clk_vc12_48,
           fase48,
           VC12_48);

comp_adapt_vc12_r_49: cp_adapt_vc12_r
  PORT MAP(TUG2_17_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_49_d, clk_tu12_local,
           tu12_just_p_49, tu12_just_n_49, inic_vc12_49, clk_vc12_49,
           fase49,
           VC12_49);

comp_adapt_vc12_r_50: cp_adapt_vc12_r
  PORT MAP(TUG2_17_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_5041_d, clk_tu12_local,
           tu12_just_p_50, tu12_just_n_50, inic_vc12_50, clk_vc12_50,
           fase50,
           VC12_50);

comp_adapt_vc12_r_51: cp_adapt_vc12_r
  PORT MAP(TUG2_17_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_51_d, clk_tu12_local,
           tu12_just_p_51, tu12_just_n_51, inic_vc12_51, clk_vc12_51,
           fase51,
           VC12_51);

comp_adapt_vc12_r_52: cp_adapt_vc12_r
  PORT MAP(TUG2_18_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_52_d, clk_tu12_local,
           tu12_just_p_52, tu12_just_n_52, inic_vc12_52, clk_vc12_52,
           fase52,
           VC12_52);


comp_adapt_vc12_r_53: cp_adapt_vc12_r
  PORT MAP(TUG2_18_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_53_d, clk_tu12_local,
           tu12_just_p_53, tu12_just_n_53, inic_vc12_53, clk_vc12_53,
           fase53,
           VC12_53);

comp_adapt_vc12_r_54: cp_adapt_vc12_r
  PORT MAP(TUG2_18_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_54_d, clk_tu12_local,
           tu12_just_p_54, tu12_just_n_54, inic_vc12_54, clk_vc12_54,
           fase54,
           VC12_54);

comp_adapt_vc12_r_55: cp_adapt_vc12_r
  PORT MAP(TUG2_19_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_55_d, clk_tu12_local,
           tu12_just_p_55, tu12_just_n_55, inic_vc12_55, clk_vc12_55,
           fase55,
           VC12_55);

comp_adapt_vc12_r_56: cp_adapt_vc12_r
  PORT MAP(TUG2_19_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_56_d, clk_tu12_local,
           tu12_just_p_56, tu12_just_n_56, inic_vc12_56, clk_vc12_56,
           fase56,
           VC12_56);

comp_adapt_vc12_r_57: cp_adapt_vc12_r
  PORT MAP(TUG2_19_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_57_d, clk_tu12_local,
           tu12_just_p_57, tu12_just_n_57, inic_vc12_57, clk_vc12_57,
           fase57,
           VC12_57);

comp_adapt_vc12_r_58: cp_adapt_vc12_r
  PORT MAP(TUG2_20_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_58_d, clk_tu12_local,
           tu12_just_p_58, tu12_just_n_58, inic_vc12_58, clk_vc12_58,
           fase58,
           VC12_58);

comp_adapt_vc12_r_59: cp_adapt_vc12_r
  PORT MAP(TUG2_20_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_59_d, clk_tu12_local,
           tu12_just_p_59, tu12_just_n_59, inic_vc12_59, clk_vc12_59,
           fase59,
           VC12_59);

comp_adapt_vc12_r_60: cp_adapt_vc12_r
  PORT MAP(TUG2_20_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_60_d, clk_tu12_local,
           tu12_just_p_60, tu12_just_n_60, inic_vc12_60, clk_vc12_60,
           fase60,
           VC12_60);

comp_adapt_vc12_r_61: cp_adapt_vc12_r
  PORT MAP(TUG2_21_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_61_d, clk_tu12_local,
           tu12_just_p_61, tu12_just_n_61, inic_vc12_61, clk_vc12_61,
           fase61,
           VC12_61);

comp_adapt_vc12_r_62: cp_adapt_vc12_r
  PORT MAP(TUG2_21_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_62_d, clk_tu12_local,
           tu12_just_p_62, tu12_just_n_62, inic_vc12_62, clk_vc12_62,
           fase62,
           VC12_62);

comp_adapt_vc12_r_63: cp_adapt_vc12_r
  PORT MAP(TUG2_21_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_63_d, clk_tu12_local,
           tu12_just_p_63, tu12_just_n_63, inic_vc12_63, clk_vc12_63,
           fase63,
           VC12_63);


comp_adapt_vc12_t_01: cp_adapt_vc12_t
 PORT MAP(VC12_01,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_01, tu12_just_p_01, tu12_just_n_01,
          TU12_01);

comp_adapt_vc12_t_02: cp_adapt_vc12_t
 PORT MAP(VC12_02,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_02, tu12_just_p_02, tu12_just_n_02,
          TU12_02);

comp_adapt_vc12_t_03: cp_adapt_vc12_t
 PORT MAP(VC12_03,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_03, tu12_just_p_03, tu12_just_n_03,
          TU12_03);

comp_adapt_vc12_t_04: cp_adapt_vc12_t
 PORT MAP(VC12_04,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_04, tu12_just_p_04, tu12_just_n_04,
          TU12_04);

comp_adapt_vc12_t_05: cp_adapt_vc12_t
 PORT MAP(VC12_05,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_05, tu12_just_p_05, tu12_just_n_05,
          TU12_05);

comp_adapt_vc12_t_06: cp_adapt_vc12_t
 PORT MAP(VC12_06,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_06, tu12_just_p_06, tu12_just_n_06,
          TU12_06);

comp_adapt_vc12_t_07: cp_adapt_vc12_t
 PORT MAP(VC12_07,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_07, tu12_just_p_07, tu12_just_n_07,
          TU12_07);

comp_adapt_vc12_t_08: cp_adapt_vc12_t
 PORT MAP(VC12_08,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_08, tu12_just_p_08, tu12_just_n_08,
          TU12_08);

comp_adapt_vc12_t_09: cp_adapt_vc12_t
 PORT MAP(VC12_09,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_09, tu12_just_p_09, tu12_just_n_09,
          TU12_09);

comp_adapt_vc12_t_10: cp_adapt_vc12_t
 PORT MAP(VC12_10,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_10, tu12_just_p_10, tu12_just_n_10,
          TU12_10);

comp_adapt_vc12_t_11: cp_adapt_vc12_t
 PORT MAP(VC12_11,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_11, tu12_just_p_11, tu12_just_n_11,
          TU12_11);

comp_adapt_vc12_t_12: cp_adapt_vc12_t
 PORT MAP(VC12_12,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_12, tu12_just_p_12, tu12_just_n_12,
          TU12_12);

comp_adapt_vc12_t_13: cp_adapt_vc12_t
 PORT MAP(VC12_13,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_13, tu12_just_p_13, tu12_just_n_13,
          TU12_13);

comp_adapt_vc12_t_14: cp_adapt_vc12_t
 PORT MAP(VC12_14,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_14, tu12_just_p_14, tu12_just_n_14,
          TU12_14);

comp_adapt_vc12_t_15: cp_adapt_vc12_t
 PORT MAP(VC12_15,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_15, tu12_just_p_15, tu12_just_n_15,
          TU12_15);

comp_adapt_vc12_t_16: cp_adapt_vc12_t
 PORT MAP(VC12_16,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_16, tu12_just_p_16, tu12_just_n_16,
          TU12_16);

comp_adapt_vc12_t_17: cp_adapt_vc12_t
 PORT MAP(VC12_17,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_17, tu12_just_p_17, tu12_just_n_17,
          TU12_17);

comp_adapt_vc12_t_18: cp_adapt_vc12_t
 PORT MAP(VC12_18,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_18, tu12_just_p_18, tu12_just_n_18,
          TU12_18);

comp_adapt_vc12_t_19: cp_adapt_vc12_t
 PORT MAP(VC12_19,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_19, tu12_just_p_19, tu12_just_n_19,
          TU12_19);

comp_adapt_vc12_t_20: cp_adapt_vc12_t
 PORT MAP(VC12_20,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_20, tu12_just_p_20, tu12_just_n_20,
          TU12_20);

comp_adapt_vc12_t_21: cp_adapt_vc12_t
 PORT MAP(VC12_21,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_21, tu12_just_p_21, tu12_just_n_21,
          TU12_21);
comp_adapt_vc12_t_22: cp_adapt_vc12_t
 PORT MAP(VC12_22,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_22, tu12_just_p_22, tu12_just_n_22,
          TU12_22);

comp_adapt_vc12_t_23: cp_adapt_vc12_t
 PORT MAP(VC12_23,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_23, tu12_just_p_23, tu12_just_n_23,
```

```
          TU12_23);

comp_adapt_vc12_t_24: cp_adapt_vc12_t
 PORT MAP(VC12_24,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_24, tu12_just_p_24, tu12_just_n_24,
          TU12_24);

comp_adapt_vc12_t_25: cp_adapt_vc12_t
 PORT MAP(VC12_25,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_25, tu12_just_p_25, tu12_just_n_25,
          TU12_25);

comp_adapt_vc12_t_26: cp_adapt_vc12_t
 PORT MAP(VC12_26,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_26, tu12_just_p_26, tu12_just_n_26,
          TU12_26);

comp_adapt_vc12_t_27: cp_adapt_vc12_t
 PORT MAP(VC12_27,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_27, tu12_just_p_27, tu12_just_n_27,
          TU12_27);

comp_adapt_vc12_t_28: cp_adapt_vc12_t
 PORT MAP(VC12_28,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_28, tu12_just_p_28, tu12_just_n_28,
          TU12_28);

comp_adapt_vc12_t_29: cp_adapt_vc12_t
 PORT MAP(VC12_29,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_29, tu12_just_p_29, tu12_just_n_29,
          TU12_29);

comp_adapt_vc12_t_30: cp_adapt_vc12_t
 PORT MAP(VC12_30,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_30, tu12_just_p_30, tu12_just_n_30,
          TU12_30);

comp_adapt_vc12_t_31: cp_adapt_vc12_t
 PORT MAP(VC12_31,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_31, tu12_just_p_31, tu12_just_n_31,
          TU12_31);

comp_adapt_vc12_t_32: cp_adapt_vc12_t
 PORT MAP(VC12_32,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_32, tu12_just_p_32, tu12_just_n_32,
          TU12_32);

comp_adapt_vc12_t_33: cp_adapt_vc12_t
 PORT MAP(VC12_33,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_33, tu12_just_p_33, tu12_just_n_33,
          TU12_33);

comp_adapt_vc12_t_34: cp_adapt_vc12_t
 PORT MAP(VC12_34,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_34, tu12_just_p_34, tu12_just_n_34,
          TU12_34);

comp_adapt_vc12_t_35: cp_adapt_vc12_t
 PORT MAP(VC12_35,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_35, tu12_just_p_35, tu12_just_n_35,
          TU12_35);

comp_adapt_vc12_t_36: cp_adapt_vc12_t
 PORT MAP(VC12_36,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_36, tu12_just_p_36, tu12_just_n_36,
          TU12_36);

comp_adapt_vc12_t_37: cp_adapt_vc12_t
 PORT MAP(VC12_37,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_37, tu12_just_p_37, tu12_just_n_37,
          TU12_37);
comp_adapt_vc12_t_38: cp_adapt_vc12_t
 PORT MAP(VC12_38,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_38, tu12_just_p_38, tu12_just_n_38,
          TU12_38);

comp_adapt_vc12_t_39: cp_adapt_vc12_t
 PORT MAP(VC12_39,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_39, tu12_just_p_39, tu12_just_n_39,
          TU12_39);

comp_adapt_vc12_t_40: cp_adapt_vc12_t
 PORT MAP(VC12_40,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_40, tu12_just_p_40, tu12_just_n_40,
          TU12_40);

comp_adapt_vc12_t_41: cp_adapt_vc12_t
 PORT MAP(VC12_41,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_41, tu12_just_p_41, tu12_just_n_41,
          TU12_41);

comp_adapt_vc12_t_42: cp_adapt_vc12_t
 PORT MAP(VC12_42,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_42, tu12_just_p_42, tu12_just_n_42,
          TU12_42);

comp_adapt_vc12_t_43: cp_adapt_vc12_t
 PORT MAP(VC12_43,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_43, tu12_just_p_43, tu12_just_n_43,
          TU12_43);

comp_adapt_vc12_t_44: cp_adapt_vc12_t
 PORT MAP(VC12_44,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_44, tu12_just_p_44, tu12_just_n_44,
          TU12_44);

comp_adapt_vc12_t_45: cp_adapt_vc12_t
 PORT MAP(VC12_45,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_45, tu12_just_p_45, tu12_just_n_45,
          TU12_45);

comp_adapt_vc12_t_46: cp_adapt_vc12_t
 PORT MAP(VC12_46,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_46, tu12_just_p_46, tu12_just_n_46,
          TU12_46);

comp_adapt_vc12_t_47: cp_adapt_vc12_t
 PORT MAP(VC12_47,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_47, tu12_just_p_47, tu12_just_n_47,
          TU12_47);

comp_adapt_vc12_t_48: cp_adapt_vc12_t
 PORT MAP(VC12_48,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_48, tu12_just_p_48, tu12_just_n_48,
          TU12_48);

comp_adapt_vc12_t_49: cp_adapt_vc12_t
 PORT MAP(VC12_49,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_49, tu12_just_p_49, tu12_just_n_49,
          TU12_49);

comp_adapt_vc12_t_50: cp_adapt_vc12_t
 PORT MAP(VC12_50,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_50, tu12_just_p_50, tu12_just_n_50,
          TU12_50);

comp_adapt_vc12_t_51: cp_adapt_vc12_t
 PORT MAP(VC12_51,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_51, tu12_just_p_51, tu12_just_n_51,
          TU12_51);

comp_adapt_vc12_t_52: cp_adapt_vc12_t
 PORT MAP(VC12_52,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_52, tu12_just_p_52, tu12_just_n_52,
          TU12_52);

comp_adapt_vc12_t_53: cp_adapt_vc12_t
 PORT MAP(VC12_53,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_53, tu12_just_p_53, tu12_just_n_53,
          TU12_53);

comp_adapt_vc12_t_54: cp_adapt_vc12_t
 PORT MAP(VC12_54,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_54, tu12_just_p_54, tu12_just_n_54,
          TU12_54);

comp_adapt_vc12_t_55: cp_adapt_vc12_t
 PORT MAP(VC12_55,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_55, tu12_just_p_55, tu12_just_n_55,
          TU12_55);

comp_adapt_vc12_t_56: cp_adapt_vc12_t
 PORT MAP(VC12_56,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_56, tu12_just_p_56, tu12_just_n_56,
          TU12_56);

comp_adapt_vc12_t_57: cp_adapt_vc12_t
 PORT MAP(VC12_57,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_57, tu12_just_p_57, tu12_just_n_57,
          TU12_57);

comp_adapt_vc12_t_58: cp_adapt_vc12_t
 PORT MAP(VC12_58,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_58, tu12_just_p_58, tu12_just_n_58,
          TU12_58);

comp_adapt_vc12_t_59: cp_adapt_vc12_t
 PORT MAP(VC12_59,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_59, tu12_just_p_59, tu12_just_n_59,
          TU12_59);

comp_adapt_vc12_t_60: cp_adapt_vc12_t
 PORT MAP(VC12_60,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_60, tu12_just_p_60, tu12_just_n_60,
          TU12_60);

comp_adapt_vc12_t_61: cp_adapt_vc12_t
 PORT MAP(VC12_61,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_61, tu12_just_p_61, tu12_just_n_61,
          TU12_61);

comp_adapt_vc12_t_62: cp_adapt_vc12_t
 PORT MAP(VC12_62,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_62, tu12_just_p_62, tu12_just_n_62,
          TU12_62);

comp_adapt_vc12_t_63: cp_adapt_vc12_t
 PORT MAP(VC12_63,
          ind_subq_local, ender_tu12_local,
          clk_tu12_local, inic_vc12_63, tu12_just_p_63, tu12_just_n_63,
          TU12_63);

comp_mult_tu12_01: cp_mult_tu12
 PORT MAP(TU12_01, TU12_02, TU12_03,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_01);

comp_mult_tu12_02: cp_mult_tu12
 PORT MAP(TU12_04, TU12_05, TU12_06,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_02);

comp_mult_tu12_03: cp_mult_tu12
 PORT MAP(TU12_07, TU12_08, TU12_09,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_03);

comp_mult_tu12_04: cp_mult_tu12
 PORT MAP(TU12_10, TU12_11, TU12_12,
```

```
          cont_3_tu12,
          clk_tug2_local,
          TUG2_04);

comp_mult_tu12_05: cp_mult_tu12
 PORT MAP(TU12_13, TU12_14, TU12_15,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_05);

comp_mult_tu12_06: cp_mult_tu12
 PORT MAP(TU12_16, TU12_17, TU12_18,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_06);

comp_mult_tu12_07: cp_mult_tu12
 PORT MAP(TU12_19, TU12_20, TU12_21,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_07);

comp_mult_tu12_08: cp_mult_tu12
 PORT MAP(TU12_22, TU12_23, TU12_24,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_08);

comp_mult_tu12_09: cp_mult_tu12
 PORT MAP(TU12_25, TU12_26, TU12_27,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_09);

comp_mult_tu12_10: cp_mult_tu12
 PORT MAP(TU12_28, TU12_29, TU12_30,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_10);

comp_mult_tu12_11: cp_mult_tu12
 PORT MAP(TU12_31, TU12_32, TU12_33,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_11);

comp_mult_tu12_12: cp_mult_tu12
 PORT MAP(TU12_34, TU12_35, TU12_36,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_12);

comp_mult_tu12_13: cp_mult_tu12
 PORT MAP(TU12_37, TU12_38, TU12_39,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_13);

comp_mult_tu12_14: cp_mult_tu12
 PORT MAP(TU12_40, TU12_41, TU12_42,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_14);

comp_mult_tu12_15: cp_mult_tu12
 PORT MAP(TU12_43, TU12_44, TU12_45,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_15);

comp_mult_tu12_16: cp_mult_tu12
 PORT MAP(TU12_46, TU12_47, TU12_48,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_16);

comp_mult_tu12_17: cp_mult_tu12
 PORT MAP(TU12_49, TU12_50, TU12_51,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_17);

comp_mult_tu12_18: cp_mult_tu12
 PORT MAP(TU12_52, TU12_53, TU12_54,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_18);

comp_mult_tu12_19: cp_mult_tu12
 PORT MAP(TU12_55, TU12_56, TU12_57,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_19);

comp_mult_tu12_20: cp_mult_tu12
 PORT MAP(TU12_58, TU12_59, TU12_60,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_20);

comp_mult_tu12_21: cp_mult_tu12
 PORT MAP(TU12_61, TU12_62, TU12_63,
          cont_3_tu12,
          clk_tug2_local,
          TUG2_21);



comp_mult_tug2_01: cp_mult_tug2
 PORT MAP(TUG2_01, TUG2_02, TUG2_03, TUG2_04, TUG2_05, TUG2_06, TUG2_07,
          cont_7_tug2,
          clk_tug3_local,
          TUG3_1);

comp_mult_tug2_02: cp_mult_tug2
 PORT MAP(TUG2_08, TUG2_09, TUG2_10, TUG2_11, TUG2_12, TUG2_13, TUG2_14,
          cont_7_tug2,
          clk_tug3_local,
          TUG3_2);

comp_mult_tug2_03: cp_mult_tug2
 PORT MAP(TUG2_15, TUG2_16, TUG2_17, TUG2_18, TUG2_19, TUG2_20, TUG2_21,
          cont_7_tug2,
          clk_tug3_local,
          TUG3_3);



 comp_mult_tug3: cp_mult_tug3
      PORT MAP(TUG3_1, TUG3_2, TUG3_3,
               cont_3_tug3,
               Tb,
               carga_vc4);

 comp_insere_h4: cp_insere_h4
  PORT MAP(carga_vc4,
           linha_local, coluna_local, ind_subq_local,
           Tb,
           VC4);


 comp_sat: cp_sat PORT MAP(VC4,
                           linha_local, coluna_local,
                           Tb, inic_vc4_local, au4_just_p_terra,
                           au4_just_n_terra,
                           AU4);

END estrutural;

USE WORK.pac_comp.ALL;
ENTITY equip_c IS
 PORT(STM1: IN BIT_VECTOR(1 TO 8);
      Tc, T1 : IN BIT;
      coluna, linha :IN INTEGER:= 0;
      coluna_local, linha_local, fase: BUFFER INTEGER:=0;
      au4_just_p, au4_just_n : BUFFER BIT;
      AU4: OUT BIT_VECTOR(1 TO 8));
END equip_c;

ARCHITECTURE estrutural OF equip_c IS

FOR comp_control_local: cp_control_local USE ENTITY WORK.control_local;
FOR comp_sat: cp_sat USE ENTITY WORK.sat;
FOR comp_sar: cp_sar USE ENTITY WORK.sar;

SIGNAL inic_vc4_local, clk_tug3_local, clk_tug2_local, clk_tu12_local,
       inic_vc12_local,inic_vc4, clk_vc4: BIT;


SIGNAL  ind_subq_local, cont_3_tug3, cont_7_tug2, cont_3_tu12, ender_tu12_local

SIGNAL  VC4 : BIT_VECTOR(1 TO 8);

BEGIN


 comp_control_local : cp_control_local
  PORT MAP(Tc,
           linha_local, coluna_local, ind_subq_local, cont_3_tug3,
           cont_7_tug2, cont_3_tu12, ender_tu12_local,
           inic_vc4_local, clk_tug3_local, clk_tug2_local, clk_tu12_local,
           inic_vc12_local);



 comp_sar: cp_sar
  PORT MAP (STM1,
            linha, coluna, linha_local, coluna_local,
            Tc, T1,
            au4_just_p, au4_just_n, inic_vc4, clk_vc4,
            fase,
            VC4);
```

```
comp_sat: cp_sat PORT MAP(VC4,
                          linha_local, coluna_local,
                          Tc, inic_vc4, au4_just_p, au4_just_n,
                          AU4);

END estrutural;


USE WORK.pac_comp.ALL;
ENTITY equip_d IS
 PORT(STM1: IN BIT_VECTOR(1 TO 8);
      Td, T1 : IN BIT;
      coluna, linha :IN INTEGER:= 0;
       tu12_just_p_01, tu12_just_n_01, inic_vc12_01, clk_vc12_01,
       tu12_just_p_02, tu12_just_n_02, inic_vc12_02, clk_vc12_02,
       tu12_just_p_03, tu12_just_n_03, inic_vc12_03, clk_vc12_03,
       tu12_just_p_04, tu12_just_n_04, inic_vc12_04, clk_vc12_04,
       tu12_just_p_05, tu12_just_n_05, inic_vc12_05, clk_vc12_05,
       tu12_just_p_06, tu12_just_n_06, inic_vc12_06, clk_vc12_06,
       tu12_just_p_07, tu12_just_n_07, inic_vc12_07, clk_vc12_07,
       tu12_just_p_08, tu12_just_n_08, inic_vc12_08, clk_vc12_08,
       tu12_just_p_09, tu12_just_n_09, inic_vc12_09, clk_vc12_09,
       tu12_just_p_10, tu12_just_n_10, inic_vc12_10, clk_vc12_10,
       tu12_just_p_11, tu12_just_n_11, inic_vc12_11, clk_vc12_11,
       tu12_just_p_12, tu12_just_n_12, inic_vc12_12, clk_vc12_12,
       tu12_just_p_13, tu12_just_n_13, inic_vc12_13, clk_vc12_13,
       tu12_just_p_14, tu12_just_n_14, inic_vc12_14, clk_vc12_14,
       tu12_just_p_15, tu12_just_n_15, inic_vc12_15, clk_vc12_15,
       tu12_just_p_16, tu12_just_n_16, inic_vc12_16, clk_vc12_16,
       tu12_just_p_17, tu12_just_n_17, inic_vc12_17, clk_vc12_17,
       tu12_just_p_18, tu12_just_n_18, inic_vc12_18, clk_vc12_18,
       tu12_just_p_19, tu12_just_n_19, inic_vc12_19, clk_vc12_19,
       tu12_just_p_20, tu12_just_n_20, inic_vc12_20, clk_vc12_20,
       tu12_just_p_21, tu12_just_n_21, inic_vc12_21, clk_vc12_21,
       tu12_just_p_22, tu12_just_n_22, inic_vc12_22, clk_vc12_22,
       tu12_just_p_23, tu12_just_n_23, inic_vc12_23, clk_vc12_23,
       tu12_just_p_24, tu12_just_n_24, inic_vc12_24, clk_vc12_24,
       tu12_just_p_25, tu12_just_n_25, inic_vc12_25, clk_vc12_25,
       tu12_just_p_26, tu12_just_n_26, inic_vc12_26, clk_vc12_26,
       tu12_just_p_27, tu12_just_n_27, inic_vc12_27, clk_vc12_27,
       tu12_just_p_28, tu12_just_n_28, inic_vc12_28, clk_vc12_28,
       tu12_just_p_29, tu12_just_n_29, inic_vc12_29, clk_vc12_29,
       tu12_just_p_30, tu12_just_n_30, inic_vc12_30, clk_vc12_30,
       tu12_just_p_31, tu12_just_n_31, inic_vc12_31, clk_vc12_31,
       tu12_just_p_32, tu12_just_n_32, inic_vc12_32, clk_vc12_32,
       tu12_just_p_33, tu12_just_n_33, inic_vc12_33, clk_vc12_33,
       tu12_just_p_34, tu12_just_n_34, inic_vc12_34, clk_vc12_34,
       tu12_just_p_35, tu12_just_n_35, inic_vc12_35, clk_vc12_35,
       tu12_just_p_36, tu12_just_n_36, inic_vc12_36, clk_vc12_36,
       tu12_just_p_37, tu12_just_n_37, inic_vc12_37, clk_vc12_37,
       tu12_just_p_38, tu12_just_n_38, inic_vc12_38, clk_vc12_38,
       tu12_just_p_39, tu12_just_n_39, inic_vc12_39, clk_vc12_39,
       tu12_just_p_40, tu12_just_n_40, inic_vc12_40, clk_vc12_40,
       tu12_just_p_41, tu12_just_n_41, inic_vc12_41, clk_vc12_41,
       tu12_just_p_42, tu12_just_n_42, inic_vc12_42, clk_vc12_42,
       tu12_just_p_43, tu12_just_n_43, inic_vc12_43, clk_vc12_43,
       tu12_just_p_44, tu12_just_n_44, inic_vc12_44, clk_vc12_44,
       tu12_just_p_45, tu12_just_n_45, inic_vc12_45, clk_vc12_45,
       tu12_just_p_46, tu12_just_n_46, inic_vc12_46, clk_vc12_46,
       tu12_just_p_47, tu12_just_n_47, inic_vc12_47, clk_vc12_47,
       tu12_just_p_48, tu12_just_n_48, inic_vc12_48, clk_vc12_48,
       tu12_just_p_49, tu12_just_n_49, inic_vc12_49, clk_vc12_49,
       tu12_just_p_50, tu12_just_n_50, inic_vc12_50, clk_vc12_50,
       tu12_just_p_51, tu12_just_n_51, inic_vc12_51, clk_vc12_51,
       tu12_just_p_52, tu12_just_n_52, inic_vc12_52, clk_vc12_52,
       tu12_just_p_53, tu12_just_n_53, inic_vc12_53, clk_vc12_53,
```

```
        tu12_just_p_54, tu12_just_n_54, inic_vc12_54, clk_vc12_54,
        tu12_just_p_55, tu12_just_n_55, inic_vc12_55, clk_vc12_55,
        tu12_just_p_56, tu12_just_n_56, inic_vc12_56, clk_vc12_56,
        tu12_just_p_57, tu12_just_n_57, inic_vc12_57, clk_vc12_57,
        tu12_just_p_58, tu12_just_n_58, inic_vc12_58, clk_vc12_58,
        tu12_just_p_59, tu12_just_n_59, inic_vc12_59, clk_vc12_59,
        tu12_just_p_60, tu12_just_n_60, inic_vc12_60, clk_vc12_60,
        tu12_just_p_61, tu12_just_n_61, inic_vc12_61, clk_vc12_61,
        tu12_just_p_62, tu12_just_n_62, inic_vc12_62, clk_vc12_62,
        tu12_just_p_63, tu12_just_n_63, inic_vc12_63, clk_vc12_63: BUFFER BIT;

         fase01, fase02, fase03, fase04, fase05, fase06, fase07,
         fase08, fase09, fase10, fase11, fase12, fase13, fase14,
         fase15, fase16, fase17, fase18, fase19, fase20, fase21,
         fase22, fase23, fase24, fase25, fase26, fase27, fase28,
         fase29, fase30, fase31, fase32, fase33, fase34, fase35,
         fase36, fase37, fase38, fase39, fase40, fase41, fase42,
         fase43, fase44, fase45, fase46, fase47, fase48, fase49,
         fase50, fase51, fase52, fase53, fase54, fase55, fase56,
         fase57, fase58, fase59, fase60, fase61, fase62, fase63:
                                                BUFFER INTEGER:=0;

        VC12_01, VC12_02, VC12_03,VC12_04, VC12_05, VC12_06, VC12_07,
        VC12_08, VC12_09, VC12_10,VC12_11, VC12_12, VC12_13, VC12_14,
        VC12_15, VC12_16, VC12_17,VC12_18, VC12_19, VC12_20, VC12_21,
        VC12_22, VC12_23, VC12_24,VC12_25, VC12_26, VC12_27, VC12_28,
        VC12_29, VC12_30, VC12_31,VC12_32, VC12_33, VC12_34, VC12_35,
        VC12_36, VC12_37, VC12_38,VC12_39, VC12_40, VC12_41, VC12_42,
        VC12_43, VC12_44, VC12_45,VC12_46, VC12_47, VC12_48, VC12_49,
        VC12_50, VC12_51, VC12_52,VC12_53, VC12_54, VC12_55, VC12_56,
        VC12_57, VC12_58, VC12_59,VC12_60, VC12_61, VC12_62, VC12_63:
                                                OUT BIT_VECTOR(1 TO 8));
END equip_d;

ARCHITECTURE estrutural OF equip_d IS


SIGNAL inic_vc4_local, clk_tug3_local, clk_tug2_local, clk_tu12_local,
        inic_vc12_local, inic_vc4, clk_vc4,

        clk_tug3_1_d, clk_tug3_2_d, clk_tug3_3_d,

        clk_tug2_01_d, clk_tug2_02_d, clk_tug2_03_d,
        clk_tug2_04_d, clk_tug2_05_d, clk_tug2_06_d,
        clk_tug2_07_d, clk_tug2_08_d, clk_tug2_09_d,
        clk_tug2_10_d, clk_tug2_11_d, clk_tug2_12_d,
        clk_tug2_13_d, clk_tug2_14_d, clk_tug2_15_d,
        clk_tug2_16_d, clk_tug2_17_d, clk_tug2_18_d,
        clk_tug2_19_d, clk_tug2_20_d, clk_tug2_21_d,

        clk_tu12_01_d, clk_tu12_02_d, clk_tu12_03_d,
        clk_tu12_04_d, clk_tu12_05_d, clk_tu12_06_d,
        clk_tu12_07_d, clk_tu12_08_d, clk_tu12_09_d,
        clk_tu12_10_d, clk_tu12_11_d, clk_tu12_12_d,
        clk_tu12_13_d, clk_tu12_14_d, clk_tu12_15_d,
        clk_tu12_16_d, clk_tu12_17_d, clk_tu12_18_d,
        clk_tu12_19_d, clk_tu12_20_d, clk_tu12_21_d,
        clk_tu12_22_d, clk_tu12_23_d, clk_tu12_24_d,
        clk_tu12_25_d, clk_tu12_26_d, clk_tu12_27_d,
        clk_tu12_28_d, clk_tu12_29_d, clk_tu12_30_d,
        clk_tu12_31_d, clk_tu12_32_d, clk_tu12_33_d,
        clk_tu12_34_d, clk_tu12_35_d, clk_tu12_36_d,
        clk_tu12_37_d, clk_tu12_38_d, clk_tu12_39_d,
        clk_tu12_40_d, clk_tu12_41_d, clk_tu12_42_d,
        clk_tu12_43_d, clk_tu12_44_d, clk_tu12_45_d,
        clk_tu12_46_d, clk_tu12_47_d, clk_tu12_48_d,
        clk_tu12_49_d, clk_tu12_50_d, clk_tu12_51_d,
        clk_tu12_52_d, clk_tu12_53_d, clk_tu12_54_d,
        clk_tu12_55_d, clk_tu12_56_d, clk_tu12_57_d,
        clk_tu12_58_d, clk_tu12_59_d, clk_tu12_60_d,
        clk_tu12_61_d, clk_tu12_62_d, clk_tu12_63_d : BIT;


SIGNAL  coluna_local, linha_local, ind_subq_local, cont_3_tug3, cont_7_tug2,
        cont_3_tu12, ender_tu12_local, linha_vc4, coluna_vc4, ind_subq:
                                                        INTEGER:=0;

SIGNAL  VC4_d, TUG3_1_d, TUG3_2_d, TUG3_3_d,
        TUG2_01, TUG2_02, TUG2_03, TUG2_04, TUG2_05, TUG2_06, TUG2_07,
        TUG2_08, TUG2_09, TUG2_10, TUG2_11, TUG2_12, TUG2_13, TUG2_14,
        TUG2_15, TUG2_16, TUG2_17, TUG2_18, TUG2_19, TUG2_20, TUG2_21 :
                                                BIT_VECTOR(1 TO 8);

FOR comp_adapt_vc12_r_01, comp_adapt_vc12_r_02, comp_adapt_vc12_r_03,
    comp_adapt_vc12_r_04, comp_adapt_vc12_r_05, comp_adapt_vc12_r_06,
    comp_adapt_vc12_r_07, comp_adapt_vc12_r_08, comp_adapt_vc12_r_09,
    comp_adapt_vc12_r_10, comp_adapt_vc12_r_11, comp_adapt_vc12_r_12,
    comp_adapt_vc12_r_13, comp_adapt_vc12_r_14, comp_adapt_vc12_r_15,
    comp_adapt_vc12_r_16, comp_adapt_vc12_r_17, comp_adapt_vc12_r_18,
    comp_adapt_vc12_r_19, comp_adapt_vc12_r_20, comp_adapt_vc12_r_21,
    comp_adapt_vc12_r_22, comp_adapt_vc12_r_23, comp_adapt_vc12_r_24,
    comp_adapt_vc12_r_25, comp_adapt_vc12_r_26, comp_adapt_vc12_r_27,
    comp_adapt_vc12_r_28, comp_adapt_vc12_r_29, comp_adapt_vc12_r_30,
    comp_adapt_vc12_r_31, comp_adapt_vc12_r_32, comp_adapt_vc12_r_33,
    comp_adapt_vc12_r_34, comp_adapt_vc12_r_35, comp_adapt_vc12_r_36,
    comp_adapt_vc12_r_37, comp_adapt_vc12_r_38, comp_adapt_vc12_r_39,
    comp_adapt_vc12_r_40, comp_adapt_vc12_r_41, comp_adapt_vc12_r_42,
    comp_adapt_vc12_r_43, comp_adapt_vc12_r_44, comp_adapt_vc12_r_45,
    comp_adapt_vc12_r_46, comp_adapt_vc12_r_47, comp_adapt_vc12_r_48,
    comp_adapt_vc12_r_49, comp_adapt_vc12_r_50, comp_adapt_vc12_r_51,
    comp_adapt_vc12_r_52, comp_adapt_vc12_r_53, comp_adapt_vc12_r_54,
    comp_adapt_vc12_r_55, comp_adapt_vc12_r_56, comp_adapt_vc12_r_57,
    comp_adapt_vc12_r_58, comp_adapt_vc12_r_59, comp_adapt_vc12_r_60,
    comp_adapt_vc12_r_61, comp_adapt_vc12_r_62, comp_adapt_vc12_r_63
                    : cp_adapt_vc12_r USE ENTITY WORK.adapt_vc12_r;

FOR comp_dmult_vc4 : cp_dmult_vc4 USE ENTITY WORK.dmult_vc4;

FOR comp_dmult_tug2_01, comp_dmult_tug2_02, comp_dmult_tug2_03,
    comp_dmult_tug2_04, comp_dmult_tug2_05, comp_dmult_tug2_06,
    comp_dmult_tug2_07, comp_dmult_tug2_08, comp_dmult_tug2_09,
    comp_dmult_tug2_10, comp_dmult_tug2_11, comp_dmult_tug2_12,
    comp_dmult_tug2_13, comp_dmult_tug2_14, comp_dmult_tug2_15,
    comp_dmult_tug2_16, comp_dmult_tug2_17, comp_dmult_tug2_18,
    comp_dmult_tug2_19, comp_dmult_tug2_20, comp_dmult_tug2_21
                    : cp_dmult_tug2 USE ENTITY WORK.dmult_tug2;


FOR comp_dmult_tug3_1, comp_dmult_tug3_2, comp_dmult_tug3_3
                    : cp_dmult_tug3 USE ENTITY WORK.dmult_tug3;

FOR comp_control_vc4: cp_control_vc4 USE ENTITY WORK.control_vc4;

FOR comp_sar: cp_sar USE ENTITY WORK.sar;

FOR comp_control_local: cp_control_local USE ENTITY WORK.control_local;
```

```
BEGIN


 comp_control_local : cp_control_local
  PORT MAP(Tb,
           linha_local, coluna_local, ind_subq_local, cont_3_tug3,
           cont_7_tug2, cont_3_tu12, ender_tu12_local,
           inic_vc4_local, clk_tug3_local, clk_tug2_local, clk_tu12_local,
           inic_vc12_local);



 comp_sar: cp_sar
  PORT MAP (STM1,
           linha, coluna, linha_local, coluna_local,
           Tb, T1,
           au4_just_p, au4_just_n, inic_vc4, clk_vc4,
           fase,
           VC4_d);

 comp_control_vc4: cp_control_vc4
  PORT MAP(VC4_d,
           inic_vc4, clk_vc4,
           linha_vc4, coluna_vc4,
           ind_subq);


 comp_dmult_vc4: cp_dmult_vc4
  PORT MAP(VC4_d,
           clk_vc4, inic_vc4,
           coluna_vc4,
           TUG3_1_d, TUG3_2_d, TUG3_3_d,
           clk_tug3_1_d, clk_tug3_2_d, clk_tug3_3_d);

 comp_dmult_tug3_1: cp_dmult_tug3
  PORT MAP(TUG3_1_d,
           coluna_vc4,
           clk_tug3_1_d,
           TUG2_01_d, TUG2_02_d, TUG2_03_d, TUG2_04_d, TUG2_05_d, TUG2_06_d,
           TUG2_07_d,
           clk_tug2_01_d, clk_tug2_02_d, clk_tug2_03_d, clk_tug2_04_d,
           clk_tug2_05_d, clk_tug2_06_d, clk_tug2_07_d);

 comp_dmult_tug3_2: cp_dmult_tug3
  PORT MAP(TUG3_1_d,
           coluna_vc4,
           clk_tug3_2_d,
           TUG2_08_d, TUG2_09_d, TUG2_10_d, TUG2_11_d, TUG2_12_d, TUG2_13_d,
           TUG2_14_d,
           clk_tug2_08_d, clk_tug2_09_d, clk_tug2_10_d, clk_tug2_11_d,
           clk_tug2_12_d, clk_tug2_13_d, clk_tug2_14_d);

 comp_dmult_tug3_3: cp_dmult_tug3
  PORT MAP(TUG3_1_d,
           coluna_vc4,
           clk_tug3_3_d,
           TUG2_15_d, TUG2_16_d, TUG2_17_d, TUG2_18_d, TUG2_19_d, TUG2_20_d,
           TUG2_21_d,
           clk_tug2_15_d, clk_tug2_16_d, clk_tug2_17_d, clk_tug2_18_d,
           clk_tug2_19_d, clk_tug2_20_d, clk_tug2_21_d);

 comp_dmult_tug2_01: cp_dmult_tug2
  PORT MAP(clk_tug2_01_d, inic_vc4,
           clk_tu12_01_d, clk_tu12_02_d, clk_tu12_03_d);

 comp_dmult_tug2_02: cp_dmult_tug2
  PORT MAP(clk_tug2_02_d, inic_vc4,
           clk_tu12_04_d, clk_tu12_05_d, clk_tu12_06_d);

 comp_dmult_tug2_03: cp_dmult_tug2
  PORT MAP(clk_tug2_03_d, inic_vc4,
           clk_tu12_07_d, clk_tu12_08_d, clk_tu12_09_d);

 comp_dmult_tug2_04: cp_dmult_tug2
  PORT MAP(clk_tug2_04_d, inic_vc4,
           clk_tu12_10_d, clk_tu12_11_d, clk_tu12_12_d);

 comp_dmult_tug2_05: cp_dmult_tug2
  PORT MAP(clk_tug2_05_d, inic_vc4,
           clk_tu12_13_d, clk_tu12_14_d, clk_tu12_15_d);

 comp_dmult_tug2_06: cp_dmult_tug2
  PORT MAP(clk_tug2_06_d, inic_vc4,
           clk_tu12_16_d, clk_tu12_17_d, clk_tu12_18_d);

 comp_dmult_tug2_07: cp_dmult_tug2
  PORT MAP(clk_tug2_07_d, inic_vc4,
           clk_tu12_19_d, clk_tu12_20_d, clk_tu12_21_d);

 comp_dmult_tug2_08: cp_dmult_tug2
  PORT MAP(clk_tug2_08_d, inic_vc4,
           clk_tu12_22_d, clk_tu12_23_d, clk_tu12_24_d);

 comp_dmult_tug2_09: cp_dmult_tug2
  PORT MAP(clk_tug2_09_d, inic_vc4,
           clk_tu12_25_d, clk_tu12_26_d, clk_tu12_27_d);

 comp_dmult_tug2_10: cp_dmult_tug2
  PORT MAP(clk_tug2_10_d, inic_vc4,
           clk_tu12_28_d, clk_tu12_29_d, clk_tu12_30_d);

 comp_dmult_tug2_11: cp_dmult_tug2
  PORT MAP(clk_tug2_11_d, inic_vc4,
           clk_tu12_31_d, clk_tu12_32_d, clk_tu12_33_d);

 comp_dmult_tug2_12: cp_dmult_tug2
  PORT MAP(clk_tug2_12_d, inic_vc4,
           clk_tu12_34_d, clk_tu12_35_d, clk_tu12_36_d);

 comp_dmult_tug2_13: cp_dmult_tug2
  PORT MAP(clk_tug2_13_d, inic_vc4,
           clk_tu12_37_d, clk_tu12_38_d, clk_tu12_39_d);

 comp_dmult_tug2_14: cp_dmult_tug2
  PORT MAP(clk_tug2_14_d, inic_vc4,
           clk_tu12_40_d, clk_tu12_41_d, clk_tu12_42_d);

 comp_dmult_tug2_15: cp_dmult_tug2
  PORT MAP(clk_tug2_15_d, inic_vc4,
           clk_tu12_43_d, clk_tu12_44_d, clk_tu12_45_d);

 comp_dmult_tug2_16: cp_dmult_tug2
  PORT MAP(clk_tug2_16_d, inic_vc4,
           clk_tu12_46_d, clk_tu12_47_d, clk_tu12_48_d);

 comp_dmult_tug2_17: cp_dmult_tug2
  PORT MAP(clk_tug2_17_d, inic_vc4,
           clk_tu12_49_d, clk_tu12_50_d, clk_tu12_51_d);
```

```
comp_dmult_tug2_18: cp_dmult_tug2
 PORT MAP(clk_tug2_18_d, inic_vc4,
          clk_tu12_52_d, clk_tu12_53_d, clk_tu12_54_d);

comp_dmult_tug2_19: cp_dmult_tug2
 PORT MAP(clk_tug2_19_d, inic_vc4,
          clk_tu12_55_d, clk_tu12_56_d, clk_tu12_57_d);

comp_dmult_tug2_20: cp_dmult_tug2
 PORT MAP(clk_tug2_20_d, inic_vc4,
          clk_tu12_58_d, clk_tu12_59_d, clk_tu12_60_d);

comp_dmult_tug2_21: cp_dmult_tug2
 PORT MAP(clk_tug2_21_d, inic_vc4,
          clk_tu12_61_d, clk_tu12_62_d, clk_tu12_63_d);


comp_adapt_vc12_r_01: cp_adapt_vc12_r
 PORT MAP(TUG2_01_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_01_d, clk_tu12_local,
          tu12_just_p_01, tu12_just_n_01, inic_vc12_01, clk_vc12_01,
          fase01,
          VC12_01);

comp_adapt_vc12_r_02: cp_adapt_vc12_r
 PORT MAP(TUG2_01_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_02_d, clk_tu12_local,
          tu12_just_p_02, tu12_just_n_02, inic_vc12_02, clk_vc12_02,
          fase02,
          VC12_02);

comp_adapt_vc12_r_03: cp_adapt_vc12_r
 PORT MAP(TUG2_01_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_03_d, clk_tu12_local,
          tu12_just_p_03, tu12_just_n_03, inic_vc12_03, clk_vc12_03,
          fase03,
          VC12_03);

comp_adapt_vc12_r_04: cp_adapt_vc12_r
 PORT MAP(TUG2_02_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_04_d, clk_tu12_local,
          tu12_just_p_04, tu12_just_n_04, inic_vc12_04, clk_vc12_04,
          fase04,
          VC12_04);

comp_adapt_vc12_r_05: cp_adapt_vc12_r
 PORT MAP(TUG2_02_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_05_d, clk_tu12_local,
          tu12_just_p_05, tu12_just_n_05, inic_vc12_05, clk_vc12_05,
          fase05,
          VC12_05);

comp_adapt_vc12_r_06: cp_adapt_vc12_r
 PORT MAP(TUG2_02_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_06_d, clk_tu12_local,
          tu12_just_p_06, tu12_just_n_06, inic_vc12_06, clk_vc12_06,
          fase06,
          VC12_06);

comp_adapt_vc12_r_07: cp_adapt_vc12_r
 PORT MAP(TUG2_03_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_0731_d, clk_tu12_local,
          tu12_just_p_07, tu12_just_n_07, inic_vc12_07, clk_vc12_07,
          fase07,
          VC12_07);

comp_adapt_vc12_r_08: cp_adapt_vc12_r
 PORT MAP(TUG2_03_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_08_d, clk_tu12_local,
          tu12_just_p_08, tu12_just_n_08, inic_vc12_08, clk_vc12_08,
          fase08,
          VC12_08);

comp_adapt_vc12_r_09: cp_adapt_vc12_r
 PORT MAP(TUG2_03_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_09_d, clk_tu12_local,
          tu12_just_p_09, tu12_just_n_09, inic_vc12_09, clk_vc12_09,
          fase09,
          VC12_09);

comp_adapt_vc12_r_10: cp_adapt_vc12_r
 PORT MAP(TUG2_04_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_1041_d, clk_tu12_local,
          tu12_just_p_10, tu12_just_n_10, inic_vc12_10, clk_vc12_10,
          fase10,
          VC12_10);

comp_adapt_vc12_r_11: cp_adapt_vc12_r
 PORT MAP(TUG2_04_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_11_d, clk_tu12_local,
          tu12_just_p_11, tu12_just_n_11, inic_vc12_11, clk_vc12_11,
          fase11,
          VC12_11);

comp_adapt_vc12_r_12: cp_adapt_vc12_r
 PORT MAP(TUG2_04_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_12_d, clk_tu12_local,
          tu12_just_p_12, tu12_just_n_12, inic_vc12_12, clk_vc12_12,
          fase12,
          VC12_12);


comp_adapt_vc12_r_13: cp_adapt_vc12_r
 PORT MAP(TUG2_05_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_13_d, clk_tu12_local,
          tu12_just_p_13, tu12_just_n_13, inic_vc12_13, clk_vc12_13,
          fase13,
          VC12_13);

comp_adapt_vc12_r_14: cp_adapt_vc12_r
 PORT MAP(TUG2_05_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_14_d, clk_tu12_local,
          tu12_just_p_14, tu12_just_n_14, inic_vc12_14, clk_vc12_14,
          fase14,
          VC12_14);

comp_adapt_vc12_r_15: cp_adapt_vc12_r
 PORT MAP(TUG2_05_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_15_d, clk_tu12_local,
          tu12_just_p_15, tu12_just_n_15, inic_vc12_15, clk_vc12_15,
          fase15,
          VC12_15);

comp_adapt_vc12_r_16: cp_adapt_vc12_r
 PORT MAP(TUG2_06_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_16_d, clk_tu12_local,
          tu12_just_p_16, tu12_just_n_16, inic_vc12_16, clk_vc12_16,
          fase16,
          VC12_16);

comp_adapt_vc12_r_17: cp_adapt_vc12_r
 PORT MAP(TUG2_06_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_17_d, clk_tu12_local,
          tu12_just_p_17, tu12_just_n_17, inic_vc12_17, clk_vc12_17,
          fase17,
          VC12_17);

comp_adapt_vc12_r_18: cp_adapt_vc12_r
 PORT MAP(TUG2_06_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_18_d, clk_tu12_local,
          tu12_just_p_18, tu12_just_n_18, inic_vc12_18, clk_vc12_18,
          fase18,
          VC12_18);

comp_adapt_vc12_r_19: cp_adapt_vc12_r
 PORT MAP(TUG2_07_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_19_d, clk_tu12_local,
          tu12_just_p_19, tu12_just_n_19, inic_vc12_19, clk_vc12_19,
          fase19,
          VC12_19);

comp_adapt_vc12_r_20: cp_adapt_vc12_r
 PORT MAP(TUG2_07_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_20_d, clk_tu12_local,
          tu12_just_p_20, tu12_just_n_20, inic_vc12_20, clk_vc12_20,
          fase20,
          VC12_20);

comp_adapt_vc12_r_21: cp_adapt_vc12_r
 PORT MAP(TUG2_07_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_21_d, clk_tu12_local,
          tu12_just_p_21, tu12_just_n_21, inic_vc12_21, clk_vc12_21,
          fase21,
          VC12_21);

comp_adapt_vc12_r_22: cp_adapt_vc12_r
 PORT MAP(TUG2_08_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_22_d, clk_tu12_local,
          tu12_just_p_22, tu12_just_n_22, inic_vc12_22, clk_vc12_22,
          fase22,
          VC12_22);

comp_adapt_vc12_r_23: cp_adapt_vc12_r
 PORT MAP(TUG2_08_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_23_d, clk_tu12_local,
          tu12_just_p_23, tu12_just_n_23, inic_vc12_23, clk_vc12_23,
          fase23,
          VC12_23);

comp_adapt_vc12_r_24: cp_adapt_vc12_r
 PORT MAP(TUG2_08_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_24_d, clk_tu12_local,
          tu12_just_p_24, tu12_just_n_24, inic_vc12_24, clk_vc12_24,
          fase24,
          VC12_24);

comp_adapt_vc12_r_25: cp_adapt_vc12_r
 PORT MAP(TUG2_09_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_25_d, clk_tu12_local,
          tu12_just_p_25, tu12_just_n_25, inic_vc12_25, clk_vc12_25,
          fase25,
          VC12_25);

comp_adapt_vc12_r_26: cp_adapt_vc12_r
 PORT MAP(TUG2_09_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_26_d, clk_tu12_local,
          tu12_just_p_26, tu12_just_n_26, inic_vc12_26, clk_vc12_26,
          fase26,
          VC12_26);

comp_adapt_vc12_r_27: cp_adapt_vc12_r
 PORT MAP(TUG2_09_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_2731_d, clk_tu12_local,
          tu12_just_p_27, tu12_just_n_27, inic_vc12_27, clk_vc12_27,
          fase27,
          VC12_27);

comp_adapt_vc12_r_28: cp_adapt_vc12_r
 PORT MAP(TUG2_10_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_28_d, clk_tu12_local,
          tu12_just_p_28, tu12_just_n_28, inic_vc12_28, clk_vc12_28,
          fase28,
          VC12_28);

comp_adapt_vc12_r_29: cp_adapt_vc12_r
 PORT MAP(TUG2_10_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_29_d, clk_tu12_local,
          tu12_just_p_29, tu12_just_n_29, inic_vc12_29, clk_vc12_29,
          fase29,
          VC12_29);

comp_adapt_vc12_r_30: cp_adapt_vc12_r
 PORT MAP(TUG2_10_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_3041_d, clk_tu12_local,
          tu12_just_p_30, tu12_just_n_30, inic_vc12_30, clk_vc12_30,
          fase30,
          VC12_30);

comp_adapt_vc12_r_31: cp_adapt_vc12_r
 PORT MAP(TUG2_11_d,
          ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
          clk_tu12_31_d, clk_tu12_local,
          tu12_just_p_31, tu12_just_n_31, inic_vc12_31, clk_vc12_31,
          fase31,
```

```
            VC12_31);

comp_adapt_vc12_r_32: cp_adapt_vc12_r
  PORT MAP(TUG2_11_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_32_d, clk_tu12_local,
           tu12_just_p_32, tu12_just_n_32, inic_vc12_32, clk_vc12_32,
           fase32,
           VC12_32);


 comp_adapt_vc12_r_33: cp_adapt_vc12_r
  PORT MAP(TUG2_11_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_33_d, clk_tu12_local,
           tu12_just_p_33, tu12_just_n_33, inic_vc12_33, clk_vc12_33,
           fase33,
           VC12_33);

comp_adapt_vc12_r_34: cp_adapt_vc12_r
  PORT MAP(TUG2_12_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_34_d, clk_tu12_local,
           tu12_just_p_34, tu12_just_n_34, inic_vc12_34, clk_vc12_34,
           fase34,
           VC12_34);

comp_adapt_vc12_r_35: cp_adapt_vc12_r
  PORT MAP(TUG2_12_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_35_d, clk_tu12_local,
           tu12_just_p_35, tu12_just_n_35, inic_vc12_35, clk_vc12_35,
           fase35,
           VC12_35);

 comp_adapt_vc12_r_36: cp_adapt_vc12_r
  PORT MAP(TUG2_12_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_36_d, clk_tu12_local,
           tu12_just_p_36, tu12_just_n_36, inic_vc12_36, clk_vc12_36,
           fase36,
           VC12_36);

comp_adapt_vc12_r_37: cp_adapt_vc12_r
  PORT MAP(TUG2_13_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_37_d, clk_tu12_local,
           tu12_just_p_37, tu12_just_n_37, inic_vc12_37, clk_vc12_37,
           fase37,
           VC12_37);

comp_adapt_vc12_r_38: cp_adapt_vc12_r
  PORT MAP(TUG2_13_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_38_d, clk_tu12_local,
           tu12_just_p_38, tu12_just_n_38, inic_vc12_38, clk_vc12_38,
           fase38,
           VC12_38);

 comp_adapt_vc12_r_39: cp_adapt_vc12_r
  PORT MAP(TUG2_13_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_39_d, clk_tu12_local,
           tu12_just_p_39, tu12_just_n_39, inic_vc12_39, clk_vc12_39,
           fase39,
           VC12_39);
```

```
  PORT MAP(TUG2_16_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_48_d, clk_tu12_local,
           tu12_just_p_48, tu12_just_n_48, inic_vc12_48, clk_vc12_48,
           fase48,
           VC12_48);

comp_adapt_vc12_r_49: cp_adapt_vc12_r
  PORT MAP(TUG2_17_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_49_d, clk_tu12_local,
           tu12_just_p_49, tu12_just_n_49, inic_vc12_49, clk_vc12_49,
           fase49,
           VC12_49);

 comp_adapt_vc12_r_50: cp_adapt_vc12_r
  PORT MAP(TUG2_17_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_5041_d, clk_tu12_local,
           tu12_just_p_50, tu12_just_n_50, inic_vc12_50, clk_vc12_50,
           fase50,
           VC12_50);

comp_adapt_vc12_r_51: cp_adapt_vc12_r
  PORT MAP(TUG2_17_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_51_d, clk_tu12_local,
           tu12_just_p_51, tu12_just_n_51, inic_vc12_51, clk_vc12_51,
           fase51,
           VC12_51);

comp_adapt_vc12_r_52: cp_adapt_vc12_r
  PORT MAP(TUG2_18_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_52_d, clk_tu12_local,
           tu12_just_p_52, tu12_just_n_52, inic_vc12_52, clk_vc12_52,
           fase52,
           VC12_52);


 comp_adapt_vc12_r_53: cp_adapt_vc12_r
  PORT MAP(TUG2_18_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_53_d, clk_tu12_local,
           tu12_just_p_53, tu12_just_n_53, inic_vc12_53, clk_vc12_53,
           fase53,
           VC12_53);

comp_adapt_vc12_r_54: cp_adapt_vc12_r
  PORT MAP(TUG2_18_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_54_d, clk_tu12_local,
           tu12_just_p_54, tu12_just_n_54, inic_vc12_54, clk_vc12_54,
           fase54,
           VC12_54);

comp_adapt_vc12_r_55: cp_adapt_vc12_r
  PORT MAP(TUG2_19_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_55_d, clk_tu12_local,
           tu12_just_p_55, tu12_just_n_55, inic_vc12_55, clk_vc12_55,
           fase55,
           VC12_55);

 comp_adapt_vc12_r_56: cp_adapt_vc12_r
  PORT MAP(TUG2_19_d,
```

```
comp_adapt_vc12_r_40: cp_adapt_vc12_r
  PORT MAP(TUG2_14_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_40_d, clk_tu12_local,
           tu12_just_p_40, tu12_just_n_40, inic_vc12_40, clk_vc12_40,
           fase40,
           VC12_40);

comp_adapt_vc12_r_41: cp_adapt_vc12_r
  PORT MAP(TUG2_14_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_41_d, clk_tu12_local,
           tu12_just_p_41, tu12_just_n_41, inic_vc12_41, clk_vc12_41,
           fase41,
           VC12_41);

comp_adapt_vc12_r_42: cp_adapt_vc12_r
  PORT MAP(TUG2_14_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_42_d, clk_tu12_local,
           tu12_just_p_42, tu12_just_n_42, inic_vc12_42, clk_vc12_42,
           fase42,
           VC12_42);

comp_adapt_vc12_r_43: cp_adapt_vc12_r
  PORT MAP(TUG2_15_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_43_d, clk_tu12_local,
           tu12_just_p_43, tu12_just_n_43, inic_vc12_43, clk_vc12_43,
           fase43,
           VC12_43);

 comp_adapt_vc12_r_44: cp_adapt_vc12_r
  PORT MAP(TUG2_15_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_44_d, clk_tu12_local,
           tu12_just_p_44, tu12_just_n_44, inic_vc12_44, clk_vc12_44,
           fase44,
           VC12_44);

comp_adapt_vc12_r_45: cp_adapt_vc12_r
  PORT MAP(TUG2_15_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_45_d, clk_tu12_local,
           tu12_just_p_45, tu12_just_n_45, inic_vc12_45, clk_vc12_45,
           fase45,
           VC12_45);

comp_adapt_vc12_r_46: cp_adapt_vc12_r
  PORT MAP(TUG2_16_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_46_d, clk_tu12_local,
           tu12_just_p_46, tu12_just_n_46, inic_vc12_46, clk_vc12_46,
           fase46,
           VC12_46);

 comp_adapt_vc12_r_47: cp_adapt_vc12_r
  PORT MAP(TUG2_16_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_4731_d, clk_tu12_local,
           tu12_just_p_47, tu12_just_n_47, inic_vc12_47, clk_vc12_47,
           fase47,
           VC12_47);

comp_adapt_vc12_r_48: cp_adapt_vc12_r
```

```
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_56_d, clk_tu12_local,
           tu12_just_p_56, tu12_just_n_56, inic_vc12_56, clk_vc12_56,
           fase56,
           VC12_56);

comp_adapt_vc12_r_57: cp_adapt_vc12_r
  PORT MAP(TUG2_19_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_57_d, clk_tu12_local,
           tu12_just_p_57, tu12_just_n_57, inic_vc12_57, clk_vc12_57,
           fase57,
           VC12_57);

comp_adapt_vc12_r_58: cp_adapt_vc12_r
  PORT MAP(TUG2_20_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_58_d, clk_tu12_local,
           tu12_just_p_58, tu12_just_n_58, inic_vc12_58, clk_vc12_58,
           fase58,
           VC12_58);

 comp_adapt_vc12_r_59: cp_adapt_vc12_r
  PORT MAP(TUG2_20_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_59_d, clk_tu12_local,
           tu12_just_p_59, tu12_just_n_59, inic_vc12_59, clk_vc12_59,
           fase59,
           VC12_59);

comp_adapt_vc12_r_60: cp_adapt_vc12_r
  PORT MAP(TUG2_20_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_60_d, clk_tu12_local,
           tu12_just_p_60, tu12_just_n_60, inic_vc12_60, clk_vc12_60,
           fase60,
           VC12_60);

comp_adapt_vc12_r_61: cp_adapt_vc12_r
  PORT MAP(TUG2_21_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_61_d, clk_tu12_local,
           tu12_just_p_61, tu12_just_n_61, inic_vc12_61, clk_vc12_61,
           fase61,
           VC12_61);

comp_adapt_vc12_r_62: cp_adapt_vc12_r
  PORT MAP(TUG2_21_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_62_d, clk_tu12_local,
           tu12_just_p_62, tu12_just_n_62, inic_vc12_62, clk_vc12_62,
           fase62,
           VC12_62);

comp_adapt_vc12_r_63: cp_adapt_vc12_r
  PORT MAP(TUG2_21_d,
           ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4, linha_vc4,
           clk_tu12_63_d, clk_tu12_local,
           tu12_just_p_63, tu12_just_n_63, inic_vc12_63, clk_vc12_63,
           fase63,
           VC12_63);

END estrutural;

USE work.pac.ALL;
USE STD.TEXTIO.ALL;
```

```
ENTITY rede IS

END rede;

ARCHITECTURE estrutural OF rede IS

CONSTANT jp : string:=" justificacao positiva ";
CONSTANT jn : string:=" justificacao negativa ";
CONSTANT fs : string:=" fase = ";
CONSTANT tp : string:=" tempo = ";
CONSTANT be :string:=" byte esperado = ";
CONSTANT br :string:=" byte recebido = ";

TYPE vc12_63_gerados IS ARRAY(1 TO 63) OF BIT_VECTOR(1 TO 8);


SIGNAL tempo : real:=0.0;

SIGNAL Ta, inic_vc12_local_a, clk_tu12_local_a,
       Tb, T1_b, Tc, T1_c, Td, T1_d,

       tu12_just_p_01, tu12_just_n_01, inic_vc12_01, clk_vc12_01,
       tu12_just_p_02, tu12_just_n_02, inic_vc12_02, clk_vc12_02,
       tu12_just_p_03, tu12_just_n_03, inic_vc12_03, clk_vc12_03,
       tu12_just_p_04, tu12_just_n_04, inic_vc12_04, clk_vc12_04,
       tu12_just_p_05, tu12_just_n_05, inic_vc12_05, clk_vc12_05,
       tu12_just_p_06, tu12_just_n_06, inic_vc12_06, clk_vc12_06,
       tu12_just_p_07, tu12_just_n_07, inic_vc12_07, clk_vc12_07,
       tu12_just_p_08, tu12_just_n_08, inic_vc12_08, clk_vc12_08,
       tu12_just_p_09, tu12_just_n_09, inic_vc12_09, clk_vc12_09,
       tu12_just_p_10, tu12_just_n_10, inic_vc12_10, clk_vc12_10,
       tu12_just_p_11, tu12_just_n_11, inic_vc12_11, clk_vc12_11,
       tu12_just_p_12, tu12_just_n_12, inic_vc12_12, clk_vc12_12,
       tu12_just_p_13, tu12_just_n_13, inic_vc12_13, clk_vc12_13,
       tu12_just_p_14, tu12_just_n_14, inic_vc12_14, clk_vc12_14,
       tu12_just_p_15, tu12_just_n_15, inic_vc12_15, clk_vc12_15,
       tu12_just_p_16, tu12_just_n_16, inic_vc12_16, clk_vc12_16,
       tu12_just_p_17, tu12_just_n_17, inic_vc12_17, clk_vc12_17,
       tu12_just_p_18, tu12_just_n_18, inic_vc12_18, clk_vc12_18,
       tu12_just_p_19, tu12_just_n_19, inic_vc12_19, clk_vc12_19,
       tu12_just_p_20, tu12_just_n_20, inic_vc12_20, clk_vc12_20,
       tu12_just_p_21, tu12_just_n_21, inic_vc12_21, clk_vc12_21,
       tu12_just_p_22, tu12_just_n_22, inic_vc12_22, clk_vc12_22,
       tu12_just_p_23, tu12_just_n_23, inic_vc12_23, clk_vc12_23,
       tu12_just_p_24, tu12_just_n_24, inic_vc12_24, clk_vc12_24,
       tu12_just_p_25, tu12_just_n_25, inic_vc12_25, clk_vc12_25,
       tu12_just_p_26, tu12_just_n_26, inic_vc12_26, clk_vc12_26,
       tu12_just_p_27, tu12_just_n_27, inic_vc12_27, clk_vc12_27,
       tu12_just_p_28, tu12_just_n_28, inic_vc12_28, clk_vc12_28,
       tu12_just_p_29, tu12_just_n_29, inic_vc12_29, clk_vc12_29,
       tu12_just_p_30, tu12_just_n_30, inic_vc12_30, clk_vc12_30,
       tu12_just_p_31, tu12_just_n_31, inic_vc12_31, clk_vc12_31,
       tu12_just_p_32, tu12_just_n_32, inic_vc12_32, clk_vc12_32,
       tu12_just_p_33, tu12_just_n_33, inic_vc12_33, clk_vc12_33,
       tu12_just_p_34, tu12_just_n_34, inic_vc12_34, clk_vc12_34,
       tu12_just_p_35, tu12_just_n_35, inic_vc12_35, clk_vc12_35,
       tu12_just_p_36, tu12_just_n_36, inic_vc12_36, clk_vc12_36,
       tu12_just_p_37, tu12_just_n_37, inic_vc12_37, clk_vc12_37,
       tu12_just_p_38, tu12_just_n_38, inic_vc12_38, clk_vc12_38,
       tu12_just_p_39, tu12_just_n_39, inic_vc12_39, clk_vc12_39,
       tu12_just_p_40, tu12_just_n_40, inic_vc12_40, clk_vc12_40,
       tu12_just_p_41, tu12_just_n_41, inic_vc12_41, clk_vc12_41,
       tu12_just_p_42, tu12_just_n_42, inic_vc12_42, clk_vc12_42,
       tu12_just_p_43, tu12_just_n_43, inic_vc12_43, clk_vc12_43,
       tu12_just_p_44, tu12_just_n_44, inic_vc12_44, clk_vc12_44,
       tu12_just_p_45, tu12_just_n_45, inic_vc12_45, clk_vc12_45,
       tu12_just_p_46, tu12_just_n_46, inic_vc12_46, clk_vc12_46,
       tu12_just_p_47, tu12_just_n_47, inic_vc12_47, clk_vc12_47,
       tu12_just_p_48, tu12_just_n_48, inic_vc12_48, clk_vc12_48,
       tu12_just_p_49, tu12_just_n_49, inic_vc12_49, clk_vc12_49,
       tu12_just_p_50, tu12_just_n_50, inic_vc12_50, clk_vc12_50,
       tu12_just_p_51, tu12_just_n_51, inic_vc12_51, clk_vc12_51,
       tu12_just_p_52, tu12_just_n_52, inic_vc12_52, clk_vc12_52,
       tu12_just_p_53, tu12_just_n_53, inic_vc12_53, clk_vc12_53,
       tu12_just_p_54, tu12_just_n_54, inic_vc12_54, clk_vc12_54,
       tu12_just_p_55, tu12_just_n_55, inic_vc12_55, clk_vc12_55,
       tu12_just_p_56, tu12_just_n_56, inic_vc12_56, clk_vc12_56,
       tu12_just_p_57, tu12_just_n_57, inic_vc12_57, clk_vc12_57,
       tu12_just_p_58, tu12_just_n_58, inic_vc12_58, clk_vc12_58,
       tu12_just_p_59, tu12_just_n_59, inic_vc12_59, clk_vc12_59,
       tu12_just_p_60, tu12_just_n_60, inic_vc12_60, clk_vc12_60,
       tu12_just_p_61, tu12_just_n_61, inic_vc12_61, clk_vc12_61,
       tu12_just_p_62, tu12_just_n_62, inic_vc12_62, clk_vc12_62,
       tu12_just_p_63, tu12_just_n_63, inic_vc12_63, clk_vc12_63,

       au4_just_p_b, au4_just_n_b, au4_just_p_c, au4_just_n_c, au4_just_p_d,
                                                   au4_just_n_d : BIT;


SIGNAL coluna_local_a, linha_local_a, coluna_b, linha_b,
       coluna_local_b, linha_local_b, coluna_c, linha_c,
       coluna_local_c, linha_local_c, coluna_d, linha_d,
       ender_tu12_local_a,
        fase01, fase02, fase03, fase04, fase05, fase06, fase07,
        fase08, fase09, fase10, fase11, fase12, fase13, fase14,
        fase15, fase16, fase17, fase18, fase19, fase20, fase21,
        fase22, fase23, fase24, fase25, fase26, fase27, fase28,
        fase29, fase30, fase31, fase32, fase33, fase34, fase35,
        fase36, fase37, fase38, fase39, fase40, fase41, fase42,
        fase43, fase44, fase45, fase46, fase47, fase48, fase49,
        fase50, fase51, fase52, fase53, fase54, fase55, fase56,
        fase57, fase58, fase59, fase60, fase61, fase62, fase63,
                fase_d, fase_b, fase_c : INTEGER:=0;

SIGNAL VC12_01, VC12_02, VC12_03,VC12_04, VC12_05, VC12_06, VC12_07,
       VC12_08, VC12_09, VC12_10,VC12_11, VC12_12, VC12_13, VC12_14,
       VC12_15, VC12_16, VC12_17,VC12_18, VC12_19, VC12_20, VC12_21,
       VC12_22, VC12_23, VC12_24,VC12_25, VC12_26, VC12_27, VC12_28,
       VC12_29, VC12_30, VC12_31,VC12_32, VC12_33, VC12_34, VC12_35,
       VC12_36, VC12_37, VC12_38,VC12_39, VC12_40, VC12_41, VC12_42,
       VC12_43, VC12_44, VC12_45,VC12_46, VC12_47, VC12_48, VC12_49,
       VC12_50, VC12_51, VC12_52,VC12_53, VC12_54, VC12_55, VC12_56,
       VC12_57, VC12_58, VC12_59,VC12_60, VC12_61, VC12_62, VC12_63,

       AU4_a, STM1_b, AU4_b, STM1_c, AU4_c, STM1_d : BIT_VECTOR(1 TO 8);

SIGNAL VC12 : vc12_63_gerados;



COMPONENT cp_equip_a
 PORT((VC12_01, VC12_02, VC12_03,VC12_04, VC12_05, VC12_06, VC12_07,
       VC12_08, VC12_09, VC12_10,VC12_11, VC12_12, VC12_13, VC12_14,
       VC12_15, VC12_16, VC12_17,VC12_18, VC12_19, VC12_20, VC12_21,
       VC12_22, VC12_23, VC12_24,VC12_25, VC12_26, VC12_27, VC12_28,
       VC12_29, VC12_30, VC12_31,VC12_32, VC12_33, VC12_34, VC12_35,
       VC12_36, VC12_37, VC12_38,VC12_39, VC12_40, VC12_41, VC12_42,
       VC12_43, VC12_44, VC12_45,VC12_46, VC12_47, VC12_48, VC12_49,
       VC12_50, VC12_51, VC12_52,VC12_53, VC12_54, VC12_55, VC12_56,
       VC12_57, VC12_58, VC12_59,VC12_60, VC12_61, VC12_62, VC12_63:
                                              IN BIT_VECTOR(1 TO 8));
       Ta : IN BIT;
       inic_vc12_local, clk_tu12_local: BUFFER BIT;
       coluna_local, linha_local, ender_tu12_local: BUFFER INTEGER:=0;
       AU4: OUT BIT_VECTOR(1 TO 8));
END COMPONENT;


COMPONENT cp_equip_b
 PORT(STM1: IN BIT_VECTOR(1 TO 8);
       Tb, T1 : IN BIT;
       coluna, linha :IN INTEGER:= 0;
       coluna_local, linha_local, fase: BUFFER INTEGER:=0;
       au4_just_p, au4_just_n : BUFFER BIT;
       AU4: OUT BIT_VECTOR(1 TO 8));

END COMPONENT;

COMPONENT cp_equip_c
 PORT(STM1: IN BIT_VECTOR(1 TO 8);
       Tc, T1 : IN BIT;
       coluna, linha :IN INTEGER:= 0;
       coluna_local, linha_local, fase: BUFFER INTEGER:=0;
       au4_just_p, au4_just_n : BUFFER BIT;
       AU4: OUT BIT_VECTOR(1 TO 8));
END COMPONENT;

COMPONENT cp_equip_d
  PORT(STM1: IN BIT_VECTOR(1 TO 8);
       Td, T1 : IN BIT;
       coluna, linha :IN INTEGER:= 0;
       tu12_just_p_01, tu12_just_n_01, inic_vc12_01, clk_vc12_01,
       tu12_just_p_02, tu12_just_n_02, inic_vc12_02, clk_vc12_02,
       tu12_just_p_03, tu12_just_n_03, inic_vc12_03, clk_vc12_03,
       tu12_just_p_04, tu12_just_n_04, inic_vc12_04, clk_vc12_04,
       tu12_just_p_05, tu12_just_n_05, inic_vc12_05, clk_vc12_05,
       tu12_just_p_06, tu12_just_n_06, inic_vc12_06, clk_vc12_06,
       tu12_just_p_07, tu12_just_n_07, inic_vc12_07, clk_vc12_07,
       tu12_just_p_08, tu12_just_n_08, inic_vc12_08, clk_vc12_08,
       tu12_just_p_09, tu12_just_n_09, inic_vc12_09, clk_vc12_09,
       tu12_just_p_10, tu12_just_n_10, inic_vc12_10, clk_vc12_10,
       tu12_just_p_11, tu12_just_n_11, inic_vc12_11, clk_vc12_11,
       tu12_just_p_12, tu12_just_n_12, inic_vc12_12, clk_vc12_12,
       tu12_just_p_13, tu12_just_n_13, inic_vc12_13, clk_vc12_13,
       tu12_just_p_14, tu12_just_n_14, inic_vc12_14, clk_vc12_14,
       tu12_just_p_15, tu12_just_n_15, inic_vc12_15, clk_vc12_15,
       tu12_just_p_16, tu12_just_n_16, inic_vc12_16, clk_vc12_16,
       tu12_just_p_17, tu12_just_n_17, inic_vc12_17, clk_vc12_17,
       tu12_just_p_18, tu12_just_n_18, inic_vc12_18, clk_vc12_18,
       tu12_just_p_19, tu12_just_n_19, inic_vc12_19, clk_vc12_19,
       tu12_just_p_20, tu12_just_n_20, inic_vc12_20, clk_vc12_20,
       tu12_just_p_21, tu12_just_n_21, inic_vc12_21, clk_vc12_21,
       tu12_just_p_22, tu12_just_n_22, inic_vc12_22, clk_vc12_22,
       tu12_just_p_23, tu12_just_n_23, inic_vc12_23, clk_vc12_23,
       tu12_just_p_24, tu12_just_n_24, inic_vc12_24, clk_vc12_24,
       tu12_just_p_25, tu12_just_n_25, inic_vc12_25, clk_vc12_25,
       tu12_just_p_26, tu12_just_n_26, inic_vc12_26, clk_vc12_26,
       tu12_just_p_27, tu12_just_n_27, inic_vc12_27, clk_vc12_27,
       tu12_just_p_28, tu12_just_n_28, inic_vc12_28, clk_vc12_28,
       tu12_just_p_29, tu12_just_n_29, inic_vc12_29, clk_vc12_29,
       tu12_just_p_30, tu12_just_n_30, inic_vc12_30, clk_vc12_30,
       tu12_just_p_31, tu12_just_n_31, inic_vc12_31, clk_vc12_31,
       tu12_just_p_32, tu12_just_n_32, inic_vc12_32, clk_vc12_32,
       tu12_just_p_33, tu12_just_n_33, inic_vc12_33, clk_vc12_33,
       tu12_just_p_34, tu12_just_n_34, inic_vc12_34, clk_vc12_34,
       tu12_just_p_35, tu12_just_n_35, inic_vc12_35, clk_vc12_35,
       tu12_just_p_36, tu12_just_n_36, inic_vc12_36, clk_vc12_36,
       tu12_just_p_37, tu12_just_n_37, inic_vc12_37, clk_vc12_37,
       tu12_just_p_38, tu12_just_n_38, inic_vc12_38, clk_vc12_38,
       tu12_just_p_39, tu12_just_n_39, inic_vc12_39, clk_vc12_39,
       tu12_just_p_40, tu12_just_n_40, inic_vc12_40, clk_vc12_40,
       tu12_just_p_41, tu12_just_n_41, inic_vc12_41, clk_vc12_41,
       tu12_just_p_42, tu12_just_n_42, inic_vc12_42, clk_vc12_42,
       tu12_just_p_43, tu12_just_n_43, inic_vc12_43, clk_vc12_43,
       tu12_just_p_44, tu12_just_n_44, inic_vc12_44, clk_vc12_44,
       tu12_just_p_45, tu12_just_n_45, inic_vc12_45, clk_vc12_45,
       tu12_just_p_46, tu12_just_n_46, inic_vc12_46, clk_vc12_46,
       tu12_just_p_47, tu12_just_n_47, inic_vc12_47, clk_vc12_47,
       tu12_just_p_48, tu12_just_n_48, inic_vc12_48, clk_vc12_48,
       tu12_just_p_49, tu12_just_n_49, inic_vc12_49, clk_vc12_49,
       tu12_just_p_50, tu12_just_n_50, inic_vc12_50, clk_vc12_50,
       tu12_just_p_51, tu12_just_n_51, inic_vc12_51, clk_vc12_51,
       tu12_just_p_52, tu12_just_n_52, inic_vc12_52, clk_vc12_52,
       tu12_just_p_53, tu12_just_n_53, inic_vc12_53, clk_vc12_53,
       tu12_just_p_54, tu12_just_n_54, inic_vc12_54, clk_vc12_54,
       tu12_just_p_55, tu12_just_n_55, inic_vc12_55, clk_vc12_55,
       tu12_just_p_56, tu12_just_n_56, inic_vc12_56, clk_vc12_56,
       tu12_just_p_57, tu12_just_n_57, inic_vc12_57, clk_vc12_57,
       tu12_just_p_58, tu12_just_n_58, inic_vc12_58, clk_vc12_58,
       tu12_just_p_59, tu12_just_n_59, inic_vc12_59, clk_vc12_59,
       tu12_just_p_60, tu12_just_n_60, inic_vc12_60, clk_vc12_60,
       tu12_just_p_61, tu12_just_n_61, inic_vc12_61, clk_vc12_61,
       tu12_just_p_62, tu12_just_n_62, inic_vc12_62, clk_vc12_62,
       tu12_just_p_63, tu12_just_n_63, inic_vc12_63, clk_vc12_63,
                                       au4_just_p, au4_just_n : BUFFER BIT;
        fase01, fase02, fase03, fase04, fase05, fase06, fase07,
        fase08, fase09, fase10, fase11, fase12, fase13, fase14,
        fase15, fase16, fase17, fase18, fase19, fase20, fase21,
        fase22, fase23, fase24, fase25, fase26, fase27, fase28,
        fase29, fase30, fase31, fase32, fase33, fase34, fase35,
        fase36, fase37, fase38, fase39, fase40, fase41, fase42,
        fase43, fase44, fase45, fase46, fase47, fase48, fase49,
        fase50, fase51, fase52, fase53, fase54, fase55, fase56,
        fase57, fase58, fase59, fase60, fase61, fase62, fase63:
                                                  BUFFER INTEGER:=0;

       VC12_01, VC12_02, VC12_03,VC12_04, VC12_05, VC12_06, VC12_07,
       VC12_08, VC12_09, VC12_10,VC12_11, VC12_12, VC12_13, VC12_14,
       VC12_15, VC12_16, VC12_17,VC12_18, VC12_19, VC12_20, VC12_21,
       VC12_22, VC12_23, VC12_24,VC12_25, VC12_26, VC12_27, VC12_28,
       VC12_29, VC12_30, VC12_31,VC12_32, VC12_33, VC12_34, VC12_35,
       VC12_36, VC12_37, VC12_38,VC12_39, VC12_40, VC12_41, VC12_42,
       VC12_43, VC12_44, VC12_45,VC12_46, VC12_47, VC12_48, VC12_49,
       VC12_50, VC12_51, VC12_52,VC12_53, VC12_54, VC12_55, VC12_56,
       VC12_57, VC12_58, VC12_59,VC12_60, VC12_61, VC12_62, VC12_63:
                                              OUT BIT_VECTOR(1 TO 8));
END COMPONENT;



FOR comp_equip_a: cp_equip_a USE ENTITY WORK.equip_a;
FOR comp_equip_b: cp_equip_b USE ENTITY WORK.equip_b;
FOR comp_equip_c: cp_equip_c USE ENTITY WORK.equip_c;
FOR comp_equip_d: cp_equip_d USE ENTITY WORK.equip_d;



BEGIN
```

```
PROCESS(clk_tu12_local_a, inic_vc12_local_a)

BEGIN
 IF (clk_tu12_local_a = '1') AND (clk_tu12_local_a'EVENT) AND
                                        (ender_tu12_local_a /= 0) THEN
   FOR i IN 62 DOWNTO 1 LOOP
    VC12(i) <= VC12(i+1) AFTER 2 ns;
   END LOOP;
   VC12(63) <= VC12(1) AFTER 2 ns;
 END IF;

 IF (inic_vc12_local_a = '1') AND (inic_vc12_local_a'EVENT) THEN
   FOR i IN 1 TO 63 LOOP
    VC12(i) <= inteiro_pra_byte(i);
   END LOOP;
 END IF;
END PROCESS;



 comp_equip_a : cp_equip_a
  PORT MAP(1), VC12(2), VC12(3), VC12(4), VC12(5), VC12(6), VC12(7),
           VC12(8), VC12(9), VC12(10), VC12(11), VC12(12), VC12(13), VC12(14),
           VC12(15), VC12(16), VC12(17), VC12(18), VC12(19), VC12(20), VC12(21),
           VC12(22), VC12(23), VC12(24), VC12(25), VC12(26), VC12(27), VC12(28),
           VC12(29), VC12(30), VC12(31), VC12(32), VC12(33), VC12(34), VC12(35),
           VC12(36), VC12(37), VC12(38), VC12(39), VC12(40), VC12(41), VC12(42),
           VC12(43), VC12(44), VC12(45), VC12(46), VC12(47), VC12(48), VC12(49),
           VC12(50), VC12(51), VC12(52), VC12(53), VC12(54), VC12(55), VC12(56),
           VC12(57), VC12(58), VC12(59), VC12(60), VC12(61), VC12(62), VC12(63),
           Ta,
           inic_vc12_local_a, clk_tu12_local_a,
           coluna_local_a, linha_local_a, ender_tu12_local_a,
           AU4_a);

ger_coluna_linha_b: PROCESS(Ta)
BEGIN
 IF (Ta = '1') THEN
  coluna_b <= coluna_local_a;
  linha_b <= linha_local_a;
 END IF;
END PROCESS;

T1_b <= NOT(Ta);

STM1_b <= AU4_a;

 comp_equip_b : cp_equip_b
  PORT MAP(STM1_b,
           Tb, T1_b,
           coluna_b, linha_b,
           coluna_local_b, linha_local_b, fase_b,
           au4_just_p_b, au4_just_n_b,
           AU4_b);

ger_coluna_linha_c: PROCESS(Tb)
BEGIN
 IF (Tb = '1') THEN
  coluna_c <= coluna_local_b;
  linha_c <= linha_local_b;
 END IF;
END PROCESS;

T1_c <= NOT(Tb);

STM1_c <= AU4_b;
```

```
 comp_equip_c : cp_equip_c
  PORT MAP(STM1_c,
           Tc, T1_c,
           coluna_c, linha_c,
           coluna_local_c, linha_local_c, fase_c,
           au4_just_p_c, au4_just_n_c,
           AU4_c);

ger_coluna_linha_d: PROCESS(Tc)
BEGIN
 IF (Tc = '1') THEN
  coluna_d <= coluna_local_c;
  linha_d <= linha_local_c;
 END IF;
END PROCESS;

T1_d <= NOT(Tc);

STM1_d <= AU4_c;

 comp_equip_d : cp_equip_d
  PORT MAP(STM1_d,
           Td, T1_d,
           coluna_d, linha_d,
       tu12_just_p_01, tu12_just_n_01, inic_vc12_01, clk_vc12_01,
       tu12_just_p_02, tu12_just_n_02, inic_vc12_02, clk_vc12_02,
       tu12_just_p_03, tu12_just_n_03, inic_vc12_03, clk_vc12_03,
       tu12_just_p_04, tu12_just_n_04, inic_vc12_04, clk_vc12_04,
       tu12_just_p_05, tu12_just_n_05, inic_vc12_05, clk_vc12_05,
       tu12_just_p_06, tu12_just_n_06, inic_vc12_06, clk_vc12_06,
       tu12_just_p_07, tu12_just_n_07, inic_vc12_07, clk_vc12_07,
       tu12_just_p_08, tu12_just_n_08, inic_vc12_08, clk_vc12_08,
       tu12_just_p_09, tu12_just_n_09, inic_vc12_09, clk_vc12_09,
       tu12_just_p_10, tu12_just_n_10, inic_vc12_10, clk_vc12_10,
       tu12_just_p_11, tu12_just_n_11, inic_vc12_11, clk_vc12_11,
       tu12_just_p_12, tu12_just_n_12, inic_vc12_12, clk_vc12_12,
       tu12_just_p_13, tu12_just_n_13, inic_vc12_13, clk_vc12_13,
       tu12_just_p_14, tu12_just_n_14, inic_vc12_14, clk_vc12_14,
       tu12_just_p_15, tu12_just_n_15, inic_vc12_15, clk_vc12_15,
       tu12_just_p_16, tu12_just_n_16, inic_vc12_16, clk_vc12_16,
       tu12_just_p_17, tu12_just_n_17, inic_vc12_17, clk_vc12_17,
       tu12_just_p_18, tu12_just_n_18, inic_vc12_18, clk_vc12_18,
       tu12_just_p_19, tu12_just_n_19, inic_vc12_19, clk_vc12_19,
       tu12_just_p_20, tu12_just_n_20, inic_vc12_20, clk_vc12_20,
       tu12_just_p_21, tu12_just_n_21, inic_vc12_21, clk_vc12_21,
       tu12_just_p_22, tu12_just_n_22, inic_vc12_22, clk_vc12_22,
       tu12_just_p_23, tu12_just_n_23, inic_vc12_23, clk_vc12_23,
       tu12_just_p_24, tu12_just_n_24, inic_vc12_24, clk_vc12_24,
       tu12_just_p_25, tu12_just_n_25, inic_vc12_25, clk_vc12_25,
       tu12_just_p_26, tu12_just_n_26, inic_vc12_26, clk_vc12_26,
       tu12_just_p_27, tu12_just_n_27, inic_vc12_27, clk_vc12_27,
       tu12_just_p_28, tu12_just_n_28, inic_vc12_28, clk_vc12_28,
       tu12_just_p_29, tu12_just_n_29, inic_vc12_29, clk_vc12_29,
       tu12_just_p_30, tu12_just_n_30, inic_vc12_30, clk_vc12_30,
       tu12_just_p_31, tu12_just_n_31, inic_vc12_31, clk_vc12_31,
       tu12_just_p_32, tu12_just_n_32, inic_vc12_32, clk_vc12_32,
       tu12_just_p_33, tu12_just_n_33, inic_vc12_33, clk_vc12_33,
       tu12_just_p_34, tu12_just_n_34, inic_vc12_34, clk_vc12_34,
       tu12_just_p_35, tu12_just_n_35, inic_vc12_35, clk_vc12_35,
       tu12_just_p_36, tu12_just_n_36, inic_vc12_36, clk_vc12_36,
       tu12_just_p_37, tu12_just_n_37, inic_vc12_37, clk_vc12_37,
       tu12_just_p_38, tu12_just_n_38, inic_vc12_38, clk_vc12_38,
       tu12_just_p_39, tu12_just_n_39, inic_vc12_39, clk_vc12_39,
       tu12_just_p_40, tu12_just_n_40, inic_vc12_40, clk_vc12_40,
       tu12_just_p_41, tu12_just_n_41, inic_vc12_41, clk_vc12_41,
       tu12_just_p_42, tu12_just_n_42, inic_vc12_42, clk_vc12_42,
       tu12_just_p_43, tu12_just_n_43, inic_vc12_43, clk_vc12_43,
       tu12_just_p_44, tu12_just_n_44, inic_vc12_44, clk_vc12_44,
       tu12_just_p_45, tu12_just_n_45, inic_vc12_45, clk_vc12_45,
       tu12_just_p_46, tu12_just_n_46, inic_vc12_46, clk_vc12_46,
       tu12_just_p_47, tu12_just_n_47, inic_vc12_47, clk_vc12_47,
       tu12_just_p_48, tu12_just_n_48, inic_vc12_48, clk_vc12_48,
       tu12_just_p_49, tu12_just_n_49, inic_vc12_49, clk_vc12_49,
       tu12_just_p_50, tu12_just_n_50, inic_vc12_50, clk_vc12_50,
       tu12_just_p_51, tu12_just_n_51, inic_vc12_51, clk_vc12_51,
       tu12_just_p_52, tu12_just_n_52, inic_vc12_52, clk_vc12_52,
       tu12_just_p_53, tu12_just_n_53, inic_vc12_53, clk_vc12_53,
       tu12_just_p_54, tu12_just_n_54, inic_vc12_54, clk_vc12_54,
       tu12_just_p_55, tu12_just_n_55, inic_vc12_55, clk_vc12_55,
       tu12_just_p_56, tu12_just_n_56, inic_vc12_56, clk_vc12_56,
       tu12_just_p_57, tu12_just_n_57, inic_vc12_57, clk_vc12_57,
       tu12_just_p_58, tu12_just_n_58, inic_vc12_58, clk_vc12_58,
       tu12_just_p_59, tu12_just_n_59, inic_vc12_59, clk_vc12_59,
       tu12_just_p_60, tu12_just_n_60, inic_vc12_60, clk_vc12_60,
       tu12_just_p_61, tu12_just_n_61, inic_vc12_61, clk_vc12_61,
       tu12_just_p_62, tu12_just_n_62, inic_vc12_62, clk_vc12_62,
       tu12_just_p_63, tu12_just_n_63, inic_vc12_63, clk_vc12_63,

       au4_just_p_d, au4_just_n_d,
        fase01, fase02, fase03, fase04, fase05, fase06, fase07,
        fase08, fase09, fase10, fase11, fase12, fase13, fase14,
        fase15, fase16, fase17, fase18, fase19, fase20, fase21,
        fase22, fase23, fase24, fase25, fase26, fase27, fase28,
        fase29, fase30, fase31, fase32, fase33, fase34, fase35,
        fase36, fase37, fase38, fase39, fase40, fase41, fase42,
        fase43, fase44, fase45, fase46, fase47, fase48, fase49,
        fase50, fase51, fase52, fase53, fase54, fase55, fase56,
        fase57, fase58, fase59, fase60, fase61, fase62, fase63,
        fase_d,
       VC12_01, VC12_02, VC12_03,VC12_04, VC12_05, VC12_06, VC12_07,
       VC12_08, VC12_09, VC12_10,VC12_11, VC12_12, VC12_13, VC12_14,
       VC12_15, VC12_16, VC12_17,VC12_18, VC12_19, VC12_20, VC12_21,
       VC12_22, VC12_23, VC12_24,VC12_25, VC12_26, VC12_27, VC12_28,
       VC12_29, VC12_30, VC12_31,VC12_32, VC12_33, VC12_34, VC12_35,
       VC12_36, VC12_37, VC12_38,VC12_39, VC12_40, VC12_41, VC12_42,
       VC12_43, VC12_44, VC12_45,VC12_46, VC12_47, VC12_48, VC12_49,
       VC12_50, VC12_51, VC12_52,VC12_53, VC12_54, VC12_55, VC12_56,
       VC12_57, VC12_58, VC12_59,VC12_60, VC12_61, VC12_62, VC12_63);

relogio: PROCESS
BEGIN
 WAIT FOR 10 ns;
  tempo <= tempo + 1.0e-9;
END PROCESS;

regist_just_au4_b: PROCESS(au4_just_p_b, au4_just_n_b)
FILE data_out : text IS OUT "../regist_just_b";
VARIABLE out_line: LINE;
VARIABLE cont_p, cont_n: INTEGER:=0;
BEGIN
 IF (au4_just_p_b = '1') AND (au4_just_p_b'EVENT) THEN
  cont_p := cont_p +1;
  WRITE(out_line, cont_p);
  WRITE(out_line, jp);
  WRITE(out_line, tp);
  WRITE(out_line, tempo);
  WRITE(out_line, fs);
  WRITE(out_line, fase_b);
  WRITELINE(data_out, out_line);
 END IF;
 IF (au4_just_n_b = '1') AND (au4_just_n_b'EVENT) THEN
  cont_n := cont_n+1;
  WRITE(out_line, cont_n);
  WRITE(out_line, jn);
  WRITE(out_line, tp);
  WRITE(out_line, tempo);
  WRITE(out_line, fs);
  WRITE(out_line, fase_b);
  WRITELINE(data_out, out_line);
 END IF;
END PROCESS;

regist_just_au4_c: PROCESS(au4_just_p_c, au4_just_n_c)
FILE data_out : text IS OUT "../regist_just_c";
VARIABLE out_line: LINE;
VARIABLE cont_p, cont_n: INTEGER:=0;

BEGIN
 IF (au4_just_p_c = '1') AND (au4_just_p_c'EVENT) THEN
  cont_p := cont_p +1;
  WRITE(out_line, cont_p);
  WRITE(out_line, jp);
  WRITE(out_line, tp);
  WRITE(out_line, tempo);
  WRITE(out_line, fs);
  WRITE(out_line, fase_c);
  WRITELINE(data_out, out_line);
 END IF;
 IF (au4_just_n_c = '1') AND (au4_just_n_c'EVENT) THEN
  cont_n := cont_n+1;
  WRITE(out_line, cont_n);
  WRITE(out_line, jn);
  WRITE(out_line, tp);
  WRITE(out_line, tempo);
  WRITE(out_line, fs);
  WRITE(out_line, fase_c);
  WRITELINE(data_out, out_line);
 END IF;
END PROCESS;

regist_just_au4_d: PROCESS(au4_just_p_d, au4_just_n_d)
FILE data_out : text IS OUT "../regist_just_d";
VARIABLE out_line: LINE;
VARIABLE cont_p, cont_n: INTEGER:=0;

BEGIN
 IF (au4_just_p_d = '1') AND (au4_just_p_d'EVENT) THEN
  cont_p := cont_p +1;
  WRITE(out_line, cont_p);
  WRITE(out_line, jp);
  WRITE(out_line, tp);
  WRITE(out_line, tempo);
  WRITE(out_line, fs);
  WRITE(out_line, fase_d);
  WRITELINE(data_out, out_line);
 END IF;
 IF (au4_just_n_d = '1') AND (au4_just_n_d'EVENT) THEN
  cont_n := cont_n+1;
  WRITE(out_line, cont_n);
  WRITE(out_line, jn);
  WRITE(out_line, tp);
  WRITE(out_line, tempo);
  WRITE(out_line, fs);
  WRITE(out_line, fase_d);
  WRITELINE(data_out, out_line);
 END IF;
END PROCESS;
```

```
regist_tu12_01: PROCESS(tu12_just_p_01, tu12_just_n_01)
FILE data_out : text IS OUT "../regist_just_01";
VARIABLE out_line: LINE;
VARIABLE cont_p, cont_n: INTEGER:=0;

BEGIN
 IF (tu12_just_p_01 = '1') AND (tu12_just_p_01'EVENT) THEN
  cont_p := cont_p + 1;
  WRITE(out_line, cont_p);
  WRITE(out_line, jp);
  WRITE(out_line, tp);
  WRITE(out_line, tempo);
  WRITE(out_line, fs);
  WRITE(out_line, fase11_d);
  WRITELINE(data_out, out_line);
 END IF;
 IF (tu12_just_n_01 = '1') AND (tu12_just_n_01'EVENT) THEN
  cont_n := cont_n + 1;
  WRITE(out_line, cont_n);
  WRITE(out_line, jn);
  WRITE(out_line, tp);
  WRITE(out_line, tempo);
  WRITE(out_line, fs);
  WRITE(out_line, fase11_d);
  WRITELINE(data_out, out_line);
 END IF;
END PROCESS;


regist_dados_nao_esperados_01: PROCESS(clk_vc12_01)
FILE data_out : text IS OUT "../regist_01";
VARIABLE out_line: LINE;
VARIABLE cont: INTEGER:=0;
VARIABLE byte: BIT_VECTOR(1 TO 8);
CONSTANT be :string:=" byte esperado = ";
CONSTANT br :string:=" byte recebido = ";

BEGIN
 IF (clk_vc12_01 = '0') THEN
  IF (inic_vc12_01 = '1') THEN
   cont := 1;
  ELSE
   IF (cont = 63) THEN
    cont := 1;
   ELSE
    cont := cont+1;
   END IF;
  END IF;
  IF (VC12_01_d /= inteiro_pra_byte(cont)) THEN
   IF (tempo > 4.0e-3) THEN
    WRITE(out_line, tp);
    WRITE(out_line, tempo);
    WRITE(out_line, be);
    WRITE(out_line, byte);
    WRITE(out_line, br);
    WRITE(out_line, VC12_01_d);
    WRITELINE(data_out, out_line);
   END IF
  END IF;
 END IF;
END PROCESS;


Ta <= not(Ta) after 257.2 ns;
Tb <= not(Tb) after 257.3 ns;
```

```
Tc <= not(Tc) after 257.2 ns;
Td <= not(Td) after 257.3 ns;

END estrutural;


 PACKAGE pac IS

  CONSTANT v_NDF : bit_vector(1 to 4) := "1001";
  TYPE ender_mem IS ARRAY(0 TO 47) OF BIT_VECTOR(1 TO 8);
  TYPE ender_mem_inic_vc IS ARRAY(0 TO 47) OF BIT;
  TYPE mem_elast_tu12 IS ARRAY(0 TO 7) OF BIT_VECTOR(1 TO 8);
  TYPE nono_bit_tu12 IS ARRAY(0 TO 7) OF BIT;
  FUNCTION byte_pra_palavra(h1, h2:  IN bit_vector(1 to 8))
                            RETURN bit_vector;
  FUNCTION palavra_pra_inteiro(palav: IN bit_vector(1 to 16))
                               RETURN INTEGER;
  FUNCTION byte_pra_inteiro(byte: IN bit_vector(1 to 8))
                               RETURN INTEGER;
  FUNCTION inteiro_pra_byte(valor_int : IN INTEGER)
                              RETURN BIT_VECTOR;
  PROCEDURE inverte_bits_I( SIGNAL h1, h2 : INOUT BIT_VECTOR(1 TO 8));
  PROCEDURE inverte_bits_D( SIGNAL h1, h2 : INOUT BIT_VECTOR(1 TO 8));
  PROCEDURE monta_ponteiro(SIGNAL h1, h2 : INOUT BIT_VECTOR(1 TO 8);
                           SIGNAL valor_ponteiro : IN INTEGER);
 PROCEDURE monta_ponteiro_com_NDF(SIGNAL h1, h2 : INOUT BIT_VECTOR(1 TO 8);
                           SIGNAL contagem : IN INTEGER);

END pac;


PACKAGE BODY pac IS

FUNCTION byte_pra_inteiro(byte: IN bit_vector(1 to 8))
                               RETURN INTEGER IS
       VARIABLE valor_int, J : integer;
    BEGIN
      J := 7;
      valor_int := 0;
      FOR i IN 1 TO 8 LOOP
        IF byte(i) = '1' THEN
          valor_int := valor_int + 2**J;
        END IF;
        J := J - 1;
      END LOOP;
      RETURN valor_int;
  END byte_pra_inteiro;

FUNCTION inteiro_pra_byte(valor_int : IN INTEGER)
                              RETURN BIT_VECTOR IS
    VARIABLE v_int : INTEGER;
    VARIABLE byte : BIT_VECTOR(1 TO 8);
     BEGIN
       v_int := valor_int;
       FOR i IN 7 DOWNTO 0 LOOP
         IF v_int >= 2**i THEN
           byte(8 - i) := '1';
           v_int := v_int - 2**i;
         ELSE
           byte(8 - i) := '0';
         END IF;
       END LOOP;
       RETURN byte;
   END inteiro_pra_byte;
```

```
 FUNCTION byte_pra_palavra(h1, h2:  IN bit_vector(1 to 8))
                           RETURN bit_vector IS
     VARIABLE palav : bit_vector(1 to 16);
    BEGIN
      FOR i IN 1 TO 8 LOOP
        palav(i) := h1(i);
      END LOOP;
      FOR i IN 9 TO 16 LOOP
        palav(i) := h2(i-8);
      END LOOP;
      RETURN palav;
 END byte_pra_palavra;

 FUNCTION palavra_pra_inteiro(palav: IN bit_vector(1 to 16))
                                RETURN INTEGER IS
       VARIABLE valor_pont, J : integer;
    BEGIN
      J := 9;
      valor_pont := 0;
      FOR i IN 7 TO 16 LOOP
        IF palav(i) = '1' THEN
          valor_pont := valor_pont + 2**J;
        END IF;
        J := J - 1;
      END LOOP;
      RETURN valor_pont;
 END palavra_pra_inteiro;

 PROCEDURE inverte_bits_I( SIGNAL h1, h2 : INOUT BIT_VECTOR(1 TO 8)) IS
    BEGIN
      h1(7) <= not(h1(7));
      FOR i IN 1 TO 7 LOOP
        IF ( i MOD 2 /= 0) THEN
          h2(i) <= not(h2(i));
        END IF;
      END LOOP;
 END inverte_bits_I;

 PROCEDURE inverte_bits_D( SIGNAL h1, h2 : INOUT BIT_VECTOR(1 TO 8)) IS
    BEGIN
      h1(8) <= not(h1(8));
      FOR i IN 2 TO 8 LOOP
        IF ( i MOD 2 = 0) THEN
          h2(i) <= not(h2(i));
        END IF;
      END LOOP;
 END inverte_bits_D;

 PROCEDURE monta_ponteiro(SIGNAL h1, h2 : INOUT BIT_VECTOR(1 TO 8);
                          SIGNAL valor_ponteiro : IN INTEGER) IS
   VARIABLE v_pont : INTEGER;
    BEGIN
      v_pont := valor_ponteiro;
      FOR i IN 1 TO 4 LOOP
        h1(i) <= not(v_NDF(i));
      END LOOP;
      IF v_pont >= 512 THEN
        h1(7) <= '1';
        v_pont := v_pont - 512;
      ELSE
        h1(7) <= '0';
      END IF;
      IF v_pont >= 256 THEN
        h1(8) <= '1';
        v_pont := v_pont - 256;
```

```
      ELSE
        h1(8) <= '0';
      END IF;
      FOR i IN 7 DOWNTO 0 LOOP
        IF v_pont >= 2**i THEN
          h2(8 - i) <= '1';
          v_pont := v_pont - 2**i;
        ELSE
          h2(8 - i) <= '0';
        END IF;
      END LOOP;
 END monta_ponteiro;

PROCEDURE monta_ponteiro_com_NDF(SIGNAL h1, h2 : INOUT BIT_VECTOR(1 TO 8)
                          SIGNAL contagem : IN INTEGER) IS
  VARIABLE v_pont : INTEGER;
    BEGIN
      v_pont:= contagem;
      FOR i IN 1 TO 4 LOOP
        h1(i) <= v_NDF(i);
      END LOOP;
      IF v_pont >= 512 THEN
        h1(7) <= '1';
        v_pont := v_pont - 512;
      ELSE
        h1(7) <= '0';
      END IF;
      IF v_pont >= 256 THEN
        h1(8) <= '1';
        v_pont := v_pont - 256;
      ELSE
        h1(8) <= '0';
      END IF;
      FOR i IN 7 DOWNTO 0 LOOP
        IF v_pont >= 2**i THEN
          h2(8 - i) <= '1';
          v_pont := v_pont - 2**i;
        ELSE
          h2(8 - i) <= '0';
        END IF;
      END LOOP;
 END monta_ponteiro_com_NDF;

END pac;


PACKAGE pac_comp IS

COMPONENT cp_adapt_vc12_t
 PORT(VC12: IN BIT_VECTOR(1 TO 8);
      ind_subq_local, ender_tu12_local: IN INTEGER;
      clk_tu12_local, inic_vc12, just_p, just_n: IN BIT;
      TU12: OUT BIT_VECTOR(1 TO 8));
END COMPONENT;

COMPONENT cp_mult_tu12
 PORT(TU12_1, TU12_2, TU12_3: IN BIT_VECTOR(1 TO 8);
      cont_3_tu12: IN INTEGER;
      clk_tug2_local: IN BIT;
      TUG2: OUT BIT_VECTOR(1 TO 8));
END COMPONENT;

COMPONENT cp_mult_tug2
 PORT(TUG2_1, TUG2_2, TUG2_3, TUG2_4, TUG2_5, TUG2_6,
                                      TUG2_7:IN BIT_VECTOR(1 TO 8);
      cont_7_tug2: IN INTEGER;
```

```
      clk_tug3_local: IN BIT;
      TUG3: OUT BIT_VECTOR(1 TO 8));
END COMPONENT;

COMPONENT cp_mult_tug3
PORT(TUG3_1, TUG3_2, TUG3_3: IN BIT_VECTOR(1 TO 8);
      cont_3_tug3: IN INTEGER;
      T0: IN BIT;
      carga_vc4: OUT BIT_VECTOR(1 TO 8));
END COMPONENT;

COMPONENT cp_adapt_vc12_r
 PORT(TUG2: IN BIT_VECTOR(1 TO 8);
      ind_subq, ind_subq_local, ender_tu12_local, coluna_vc4,
                                                  linha_vc4: IN INTEGER;
      clk_tu12, clk_tu12_local: IN BIT;
      just_p, just_n, inic_vc12, clk_vc12: BUFFER BIT;
      fase: BUFFER INTEGER;
      VC12: OUT BIT_VECTOR(1 TO 8));
END COMPONENT;

COMPONENT cp_dmult_tug2
 PORT(clk_tug2, inic_vc4: IN BIT;
      clk_tu12_1, clk_tu12_2, clk_tu12_3: OUT BIT);
END COMPONENT;

COMPONENT cp_dmult_tug3
 PORT(TUG3: IN BIT_VECTOR(1 TO 8);
     coluna_vc4: IN INTEGER;
     clk_tug3: IN BIT;
     TUG2_1, TUG2_2, TUG2_3, TUG2_4, TUG2_5, TUG2_6,
     TUG2_7: OUT BIT_VECTOR(1 TO 8);
     clk_tug2_1, clk_tug2_2, clk_tug2_3, clk_tug2_4,
     clk_tug2_5, clk_tug2_6, clk_tug2_7: BUFFER BIT);
END COMPONENT;

COMPONENT cp_dmult_vc4
 PORT(VC4: IN BIT_VECTOR(1 TO 8);
     clk_vc4, inic_vc4: IN BIT;
     coluna_vc4: IN INTEGER;
     TUG3_1, TUG3_2, TUG3_3: OUT BIT_VECTOR(1 TO 8);
     clk_tug3_1, clk_tug3_2, clk_tug3_3: BUFFER BIT);
END COMPONENT;

COMPONENT cp_control_vc4
 PORT(VC4: IN BIT_VECTOR(1 TO 8);
      inic_vc4, clk_vc4: IN BIT;
      linha_vc4, coluna_vc4: BUFFER INTEGER;
      ind_subq: OUT INTEGER);
END COMPONENT;



COMPONENT cp_sar
 PORT(STM1: IN BIT_VECTOR(1 TO 8);
      linha, coluna, linha_local, coluna_local: IN INTEGER;
      T0, T1: IN BIT;
      just_p, just_n, inic_vc4, clk_vc4: BUFFER BIT;
     fase: BUFFER INTEGER;
      VC4: OUT BIT_VECTOR(1 TO 8));
END COMPONENT;

COMPONENT cp_sat
 PORT(VC4: IN BIT_VECTOR(1 TO 8);
      linha_local, coluna_local: IN INTEGER;
      T0, inic_vc4, just_p, just_n: IN BIT;
```

```
      AU4: OUT BIT_VECTOR(1 TO 8));
END COMPONENT;

COMPONENT cp_insere_h4
 PORT(carga_vc4: IN BIT_VECTOR(1 TO 8);
      linha_local, coluna_local, ind_subq_local: IN INTEGER;
      T0: IN BIT;
      VC4: OUT BIT_VECTOR(1 TO 8));
END COMPONENT;

COMPONENT cp_control_local
 PORT(T0: IN BIT;
      linha_local, coluna_local, ind_subq_local, cont_3_tug3,
      cont_7_tug2, cont_3_tu12, ender_tu12_local: BUFFER INTEGER;
      inic_vc4_local, clk_tug3_local, clk_tug2_local, clk_tu12_local,
      inic_vc12_local: BUFFER BIT);
END COMPONENT;


END pac_comp;
```