4
Realização e Caracterização do Módulo Receptor

Este capítulo tem como objetivo apresentar o desenvolvimento de um módulo de recepção capaz de alinhar e sincronizar uma seqüência de bits gerada no padrão PRBS / NRZ para posteriormente avaliar a taxa de bits errados.

4.1.
Introdução

Sabemos que em sistemas ópticos, a transmissão de uma seqüência de bits sofre um determinado retardo introduzido pela fibra e neste caso temos também um retardo adicional introduzido pelo chip SERDES. Portanto, o receptor deverá sincronizar a seqüência recebida para que não seja identificado nenhum tipo de erro no sistema.

Na implementação deste módulo receptor, a seqüência recebida é comparada à uma seqüência idêntica gerada pelo transmissor (padrão PRBS) sendo que para o sistema operar corretamente e alinhar as duas seqüências é necessário que o número de erros acumulados seja nulo ou um valor muito pequeno. Desta forma, o fator determinante para o alinhamento do sistema é a ocorrência de poucos ou nenhum erro no sistema. A análise do funcionamento deste módulo será introduzida a seguir.

4.2.
Ferramentas computacionais desenvolvidas no FPGA

No módulo receptor desenvolvido nesta dissertação implementou-se no FPGA uma lógica em hardware para cada bloco do diagrama da Figura 22 abaixo.

O conjunto desses 6 blocos realizados foi chamado de AVALIA_ERRO e cada um destes blocos será estudado e detalhado separadamente neste item 4.2.

Primeiramente em cada bloco representado na figura abaixo foi desenvolvido uma lógica programável em VHDL através do software da Xilinx ISE Project Navigator 8.2i e suas respectivas simulações foram executadas utilizando o software de simulação Modelsim sendo estas últimas apresentadas no próximo capítulo.

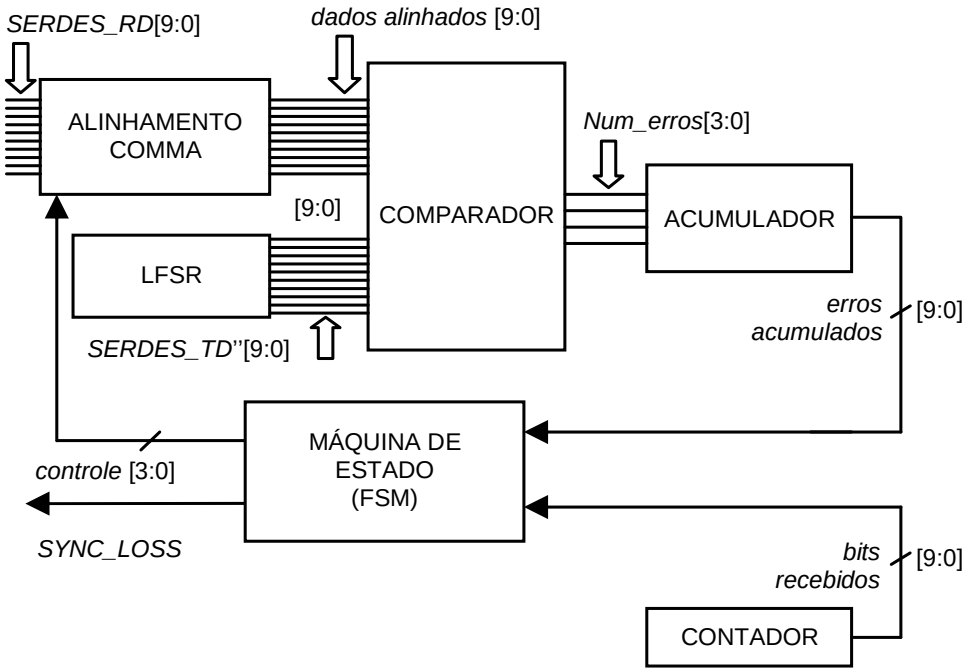

Figura 22. Diagrama de blocos representando o bloco AVALIA_ERRO utilizado para implementar o medidor de taxa de bits errados.

O diagrama da Figura 22 acima encontra-se completo com todos seus sinais representados no Apêndice C. A omissão de alguns sinais na figura acima foi apenas para simplificar e organizar melhor o diagrama de blocos AVALIA_ERRO.

No módulo receptor o SERDES funciona como deserializador convertendo dados seriais em paralelo que irão para o FPGA e é nesta interface SERDES / FPGA que dá-se início ao diagrama de blocos acima através do sinal *SERDES_RD* que é entrada do bloco ALINHAMENTO_COMMA.

4.2.1.
Bloco ALINHAMENTO_COMMA

Como mencionado anteriormente, o fator determinante para o sincronismo do sistema indicando o alinhamento correto é o número de erros gerado por cada seqüência de bits, ou seja, se o número de erros acumulados estiver muito alto o sistema não alinha caso contrário alinhará corretamente. Entenderemos melhor o funcionamento deste mecanismo de alinhamento através de sua caracterização abaixo.

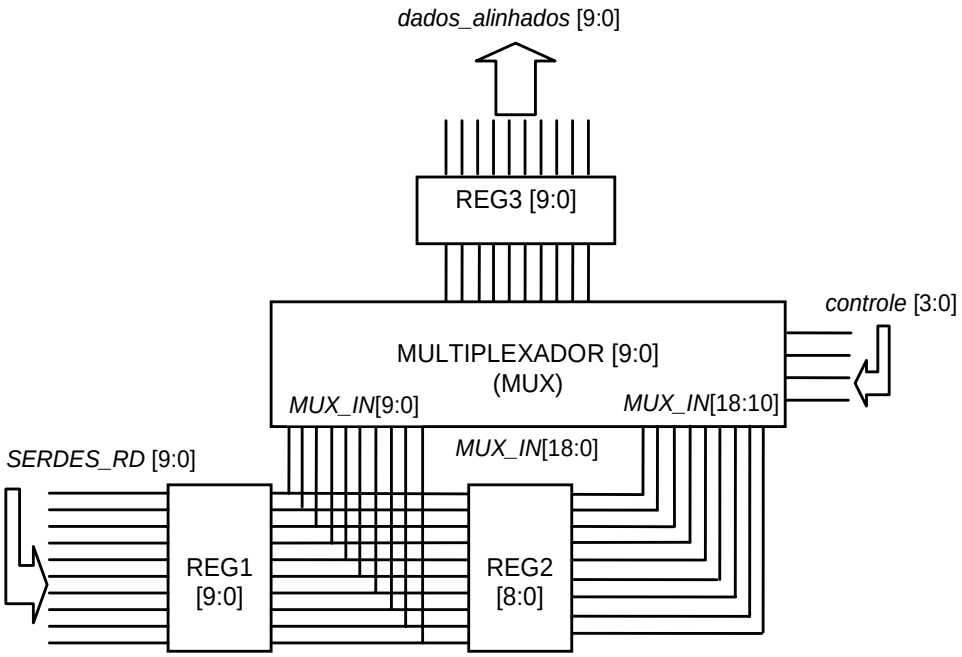

Figura 23. Diagrama de blocos referente ao bloco ALINHAMENTO_COMMA.

O diagrama de blocos da Figura 23 acima representa o bloco ALINHAMENTO_COMMA da Figura 22. Este bloco é muito importante uma vez que é o responsável por alinhar todos os bits que saem da fibra e do chip SERDES. Ele possui três portas de entrada: *relógio*, *controle* (4 bits) e *SERDES_RD* (10 bits) e uma porta de saída: *dados_alinhados* (10 bits). Para a realização deste bloco foram necessários dois registradores de 10 bits, REG1 e REG3, um registrador de 9 bits, REG2, e um multiplexador (MUX) de 10 bits que

funciona de acordo com um sinal de controle de 4 bits (*controle* [3:0]) e com um sinal de entrada do MUX chamado MUX_IN de 19 bits.

A cada transição positiva do relógio, o registro REG1 passa a obter os valores das seqüências de bits do SERDES_RD, o registro REG2 passa a obter os valores do registro REG1 sem o bit mais significativo deste, ou seja, REG2 será REG1[8:0] e o registro REG3 passa a adquirir os valores do MUX. Sem a dependência da transição de relógio, os valores do sinal MUX_IN são determinados através da concatenação de REG2 com REG1 sendo o registro REG2 o mais significativo e a seqüência *dados_alinhados* passa a obter os valores de REG3.

Foram implementados dois mecanismos de alinhamento: O primeiro deles corresponde ao alinhamento de formação de palavras que ocorre no interior do bloco ALINHAMENTO_COMMA entre as interfaces *SERDES_RD* E *dados_alinhados*. O segundo mecanismo de alinhamento corresponde ao alinhamento entre a palavra recebida *dados_alinhados* e a palavra local gerada pelo LFSR da recepção.

No primeiro mecanismo de alinhamento, o sinal *controle* determina quais 10 bits dos 19 bits do sinal MUX_IN [18:0] serão selecionados pelo bloco MULTIPLEXADOR (MUX) e de forma resumida pode-se detalhar como esses sinais operam: a partir do momento em que o *controle* for 0, o MUX irá selecionar os 10 primeiros bits menos significativos do sinal MUX_IN,ou seja, MUX_IN [9:0]. Para *controle* igual a 1 o MUX selecionará os 10 próximos bits após o primeiro bit menos significativo representando um conjunto de 10 bits mas com um shift para a esquerda, ou seja, capturando o sinal MUX_IN [10:1] e assim sucessivamente para cada *controle* diferente até o *controle* igual a 9 que equivale a MUX_IN[18:9].

Pode-se supor que várias seqüências de 10 bits como por exemplo $A_9A_8A_7A_6A_5A_4A_3A_2A_1A_0$, $B_9B_8B_7B_6B_5B_4B_3B_2B_1B_0$ etc tenham sido transmitidas. Na recepção, o chip SERDES quebra essa seqüência serial em palavras de 10 bits disserializando-as na entrada do sinal SERDES_RD. Porém, o chip SERDES pode perder alguns bits no momento da partição para formar uma palavra como por

exemplo A_9 e A_8. Desta forma, estas estarão de forma desordenada na posição SERDES_RD na entrada do registro REG1 e o bloco ALINHAMENTO-_COMMA será o responsável pelo alinhamento correto da palavra. Através da Tabela 2 abaixo torna-se mais fácil o entendimento do bloco ALINHAMENTO_COMMA onde tem-se as seqüências de entrada do multiplexador associadas a cada um dos controles. Por exemplo, MUX(0) representa MUX_IN[9:0] e é a seqüência selecionada pelo MUX quando o *controle* for 0, MUX(1) representa MUX_IN[10:1] e é a seqüência selecionada pelo MUX quando o *controle* for 1 e assim por diante até MUX(9).

Após a primeira transição de relógio, temos nas saídas dos registros REG1 e REG2 as seqüências apresentadas na primeira coluna da Tabela 2 abaixo. Até então REG2 ainda não obteve nenhum registro de seqüência, somente após a segunda transição de relógio é que a saída do registro REG1 será deslocada para a saída do registro REG2 e isto pode ser observado na segunda coluna da Tabela 2 abaixo e assim por diante para as próximas seqüências que chegarem.

	Após 1º Relógio	Após 2º Relógio	Após 3º Relógio	Após 4º Relógio
REG1	$A_{7.....}A_0B_9B_8$	$B_{7.....}B_0\,C_9C_8$	$C_{7.....}C_0D_9D_8$	$D_{7.....}D_0E_9E_8$
REG2	xxxxxxxxx	$A_{6.....}A_0B_9B_8$	$B_{6.....}B_0\,C_9C_8$	$C_{6.....}C_0D_9D_8$
MUX_IN	xxxxxxxxx$A_{7.....}A_0B_9B_8$	$A_{6.....}A_0B_{9.....}B_0C_9C_8$	$B_{6.....}B_0C_{9.....}C_0\,D_9D_8$	$C_{6.....}C_0D_{9.....}D_0\,E_9E_8$
MUX(0)	$A_{7.....}A_0B_9B_8$	$B_{7.....}B_0C_9C_8$	$C_{7.....}C_0\,D_9D_8$	$D_{7.....}D_0\,E_9E_8$
MUX(1)	xxxxxxxxxx	$B_8B_{7.....}B_0C_9$	$C_8C_{7.....}C_0\,D_9$	$D_8D_{7.....}D_0\,E_9$
MUX(2)	xxxxxxxxxx	$B_9B_{8.....}B_0$	$C_9C_{8.....}C_0$	$D_9D_{8.....}D_0$
MUX(3)	xxxxxxxxxx	$A_0B_{9.....}B_1$	$B_0C_{9.....}C_1$	$C_0D_{9.....}D_1$
MUX(4)	xxxxxxxxxx	$A_1A_0B_{9.....}B_2$	$B_1B_0C_{9.....}C_2$	$C_1C_0D_{9.....}D_2$
MUX(5)	xxxxxxxxxx	$A_2A_1A_0B_{9.....}B_3$	$B_2B_1B_0C_{9.....}C_3$	$C_2C_1C_0D_{9.....}D_3$
MUX(6)	xxxxxxxxxx	$A_{3.....}A_0B_{9.....}B_4$	$B_{3.....}B_0C_{9.....}C_4$	$C_{3.....}C_0D_{9.....}D_4$
MUX(7)	xxxxxxxxxx	$A_{4.....}A_0B_{9.....}B_5$	$B_{4.....}B_0C_{9.....}C_5$	$C_{4.....}C_0D_{9.....}D_5$
MUX(8)	xxxxxxxxxx	$A_{5.....}A_0B_{9.....}B_6$	$B_{5.....}B_0C_{9.....}C_6$	$C_{5.....}C_0D_{9.....}D_6$
MUX(9)	xxxxxxxxxx	$A_{6.....}A_0B_9B_8B_7$	$B_{6.....}B_0C_9C_8C_7$	$C_{6.....}C_0D_9D_8D_7$

Nota:
MUX(0) representa MUX_IN[9:0] e é a seqüência selecionada pelo MUX quando o controle for 0; MUX(1) representa MUX_IN[10:1] e é a seqüência selecionada pelo MUX quando o controle for 1; e assim por diante até MUX(9).

Tabela 2. Exemplo genérico do funcionamento do bloco ALINHAMENTO_COMMA.

Através do exemplo da Tabela 2 acima podemos perceber que, quando o *controle* do bloco ALINHAMENTO COMMA for igual a 2, ou seja, quando o MUX estiver selecionando a seqüência MUX(2), teremos a seqüência de 10 bits em *dados _alinhados* na ordem correta mas esta somente indicará o alinhamento correto e desejado do sistema se a seqüência de 10 bits do contador LFSR da recepção for idêntica a seqüência MUX(2).

No segundo mecanismo de alinhamento, a palavra recebida em *dados_alinhados* está alinhada com a palavra local gerada pelo LFSR da recepção, porém, elas estão defasadas. Dessa forma, para que as palavras estejam em fase, o LFSR da recepção deverá sofrer retardos no intuito de alinhar as duas palavras.

Pode-se perceber que para o sistema alinhar corretamente, os dois mecanismos de alinhamento devem ocorrer ao mesmo tempo. Caso contrário, o sistema continuará com um elevado número de erros.

4.2.2. Bloco MÁQUINA DE ESTADOS

Máquinas de Estados são estruturas lógicas utilizadas para modelar um determinado comportamento e são compostas por um conjunto de estados e um conjunto de regras de transição entre estes. As regras de transição representam as condições necessárias para haver a mudança de estados. No caso desta Dissertação temos uma Máquina de Estados Finita (*FSM – Finite State Machine*) [15], pois seu número de estados é finito. Geralmente, utiliza-se dois blocos de lógica combinacional para aplicações em hardware. O primeiro bloco determina os estados de transição e o segundo determina as saídas da Máquina de Estados.

A Máquina de Estados é considerada o bloco mais importante do módulo receptor juntamente com o bloco ALINHAMENTO_COMMA e esta possui 10 estados de `alinhamento_comma` indo desde o estado `alinhamento_comma0` (`al_cm0`) ao `alinhamento_comma9` (`al_cm9`), possui um estado de troca do contador LFSR chamado `change_lfsr` e por fim um estado de travamento chamado `lock_state`.

Este bloco apresenta quatro portas de entrada: *relógio*, *reset*, *erros_acumulados* (10 bits) e *bits_recebidos* (10 bits) e oito portas de saída: *controle* (4 bits), *SYNC_LOSS*, *reset_lfsr*, *reset_acumulador*, *reset_contador*, *enable_lfsr10*[8], *enable_acumulador*, *enable_contador*.

O mecanismo de alinhamento referente ao bloco ALINHAMENTO COMMA do item anterior é implementado para os 10 estados de `alinhamento_comma` da máquina de estados sendo que cada sinal de controle gerado pela máquina de estados corresponde a um determinado estado de `alinhamento_comma`, ou seja, quando o *controle* for igual a '0', quer dizer que estamos no estado `alinhamento_comma0`, quando o *controle* for igual a '1', estamos no estado `alinhamento_comma1` e assim por diante até o *controle* igual a '9'. Após estes 10 estados, têm-se um estado de apenas uma transição chamado `change_lfsr` onde o contador LFSR da recepção sofre um retardo de 1 relógio a cada vez que passa por este estado sendo o retardo justificado pelo fato da porta de saída *enable_lfsr10* da máquina ser desabilitada uma vez que é gerada por ela. Neste estado `change_lfsr` o controle é igual a 'A' e tem-se também o contador e o acumulador sendo resetados.

Se após os 10 estados de `alinhamento_comma` e o estado `change_lfsr` o sistema não alinhar, a varredura por estes 11 estados continuará ocorrendo e se repetirá até ser detectado o estado `alinhamento_comma` onde os erros acumulados sejam zero. Neste caso onde os erros acumulados são nulos, sabemos o estado `alinhamento_comma` responsável pelo alinhamento correto do sistema e somente após varrer os 1000 bits deste último estado de alinhamento, é que a máquina de estados será travada no estado `lock_state`. É importante destacar que cada estado `alinhamento_comma` é composto por 100 palavras de 10 bits e como temos 10 estados de alinhamento e sabendo também que o contador LFSR pode implementar 1023 palavras podemos concluir que na pior

[8] As portas *enable_lfsr10*, *enable_acumulador* e *enable_contador* , uma vez ativadas, habilitam o funcionamento do contador LFSR, acumulador e contador, respectivamente.

das hipóteses temos que varrer 10 milhões de bits para alinhar corretamente o sistema.

Utilizaram-se dois sinais determinantes na transição de estados da máquina sendo eles *end_count* e *lock* como podemos visualizar através da Figura 24 abaixo. O sinal *end_count* é ativado na última ou centésimo transição de relógio de cada um dos estados `alinhamento_comma` e `lock_state`, ou seja, é habilitado quando a porta de entrada *bits_recebidos* for igual a 100_{dec} ou 64_{hex} representando o final de cada estado e é nesse momento que o contador e o acumulador são resetados. Já o sinal *lock* é ativado quando a porta de entrada *erros_acumulados* for nula ou então se pode também acionar o *lock* através da especificação de um determinado número mínimo de erros aceitável no processo de programação.

Caso os sinais *end_count* e *lock* fossem iguais a 1, a máquina de estados deslocava-se de algum estado de `alinhamento_comma` para o estado `lock_state` permanecendo travada neste estado. Já quando o sinal *end_count* fosse igual a 1 e o *lock* fosse igual a 0, havia um deslocamento de algum estado `alinhamento_comma` para algum outro estado `alinhamento_comma` consecutivo ou então para o estado `change_lfsr` caso o estado anterior fosse `alinhamento_comma9` (`al_cm9`).

A máquina de estados possui uma das portas de saída chamada *SYNC_LOSS*[9] responsável por indicar a perda de sincronismo do sistema. Esta porta somente exibirá o valor zero quando estivermos no estado `lock_state` representando então a sincronização correta do sistema, caso contrário permanecerá ativada indicando a ausência de sincronismo do mesmo.

[9] O sinal *SYNC_LOSS* utilizado nesta Dissertação opera de forma similar ao presente em alguns dispositivos de medição de erro.

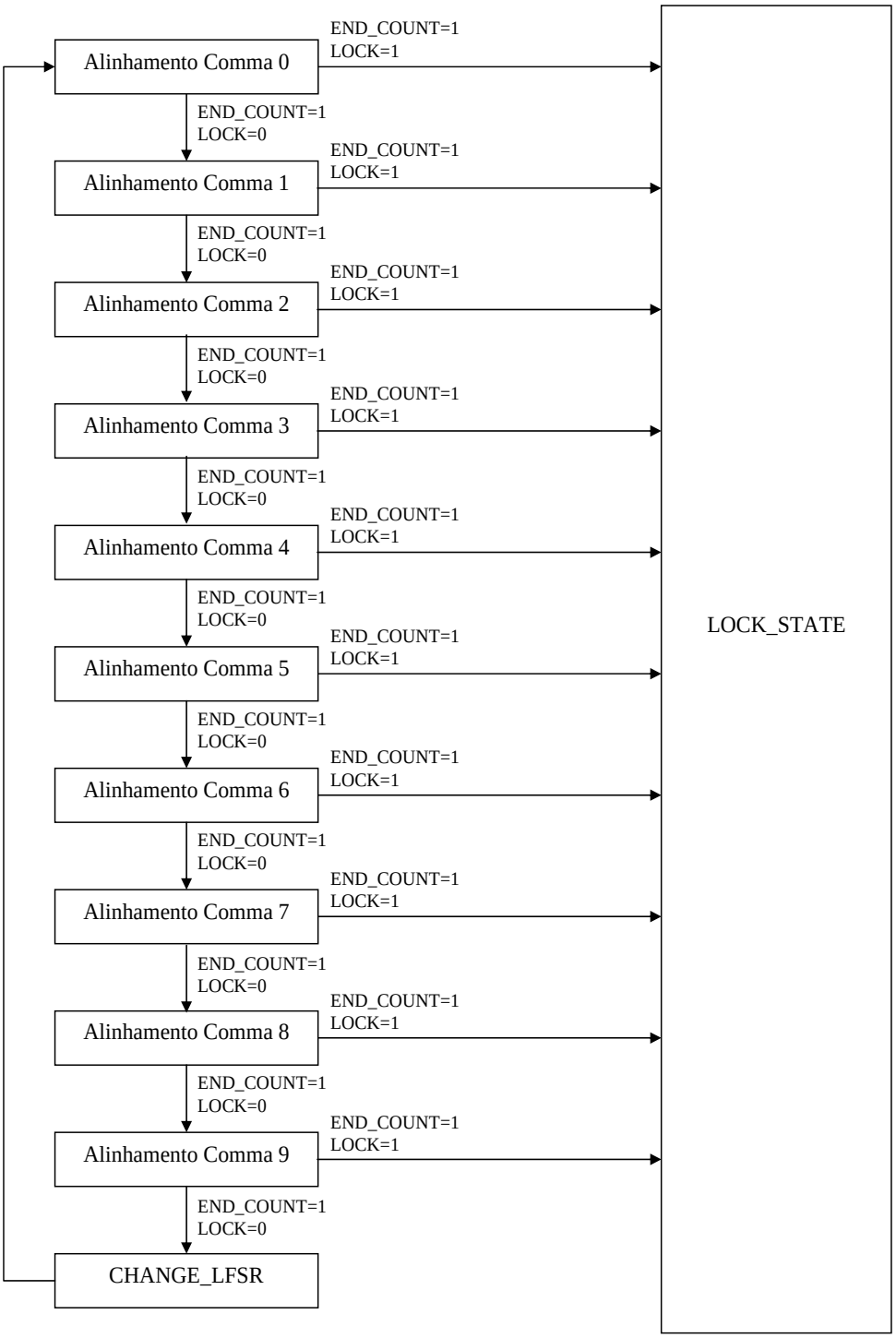

Figura 24. Diagrama de estados da Máquina de Estados.

4.2.3.
Bloco LFSR

O bloco LFSR da recepção apresenta três portas de entrada: *relógio*, *reset_lfsr* e *enable_lfsr10* e uma porta de saída: *SERDES_TD”* (10 bits). Este bloco LFSR da recepção é idêntico ao bloco LFSR da transmissão sendo a única diferença entre eles o retardo no contador LFSR da recepção. A seqüência *SERDES_TD’’* é a mesma seqüência *SERDES_TD* da transmissão apenas para os primeiros 10 estados de `alinhamento_comma`. Como já foi dito, após estes 10 estados têm-se um estado de apenas uma transição chamado `change_lfsr` onde o contador LFSR da recepção sofre um retardo de 1 relógio a cada vez que passa por este estado. Sendo assim, como é a Máquina de Estados quem gera o sinal *enable_lfsr10* que vai para o contador LFSR, é ela quem desabilita sua porta de saída *enable_lfsr10* no estado `change_lfsr` justificando o retardo neste estado. Dessa forma, a varredura pelo próximo conjunto dos 10 estados `alinhamento_comma` será feita e após o segundo estado `change_lfsr`, verificaremos um atraso de 2 relógios em relação à seqüência *SERDES_TD* original e assim sucessivamente.

4.2.4.
Bloco COMPARADOR

O bloco COMPARADOR apresenta três portas de entrada: *relógio*, *dados_alinhados* (10 bits) e *SERDES_TD”* (10 bits) e uma porta de saída: *Num_erros* (4 bits). Este bloco tem como objetivo comparar as duas seqüências de 10 bits e para cada comparação realizada entre elas tem-se na porta de saída *Num_erros* o número de erros obtidos. Conforme mencionado no capítulo 2 item 2.3, a comparação é feita através da utilização de uma porta OU Exclusivo onde as seqüências *dados_alinhados* e *SERDES_TD”* deverão ser idênticas bit a bit . Se estas seqüências forem iguais teremos como resultado o valor 0 na porta de saída *Num_erros* não havendo nenhum erro no sistema. Caso contrário, teremos o valor 1 na porta de saída deste bloco indicando que um dos dez bits foi identificado incorretamente. O número de bits distintos após a comparação das duas seqüências é variável e será visualizado na porta de saída do bloco

COMPARADOR. Por exemplo, se forem identificados quatro bits diferentes após a comparação das seqüências de 10 bits, teremos o valor quatro na porta *Num_erros*. implicando na existência de quatro erros no receptor. Estas comparações são sempre realizadas para conjuntos de 10 bits de ambas as seqüências sendo que esses erros obtidos serão somados e acumulados no bloco ACUMULADOR que será descrito a seguir.

4.2.5. Bloco ACUMULADOR

Este bloco possui quatro portas de entrada: *relógio*, *reset_acumulador*, *enable_acumulador* e *Num-erros* (4 bits) e uma porta de saída: *erros_acumulados* (10 bits). O objetivo deste bloco é acumular os erros provenientes de todas as comparações entre as seqüências de 10 bits, de forma que a quantidade de erros na porta de saída *erros_acumulados* numa transição qualquer é determinada pela quantidade de erros na porta de saída *erros_acumulados* na transição anterior a esta acrescida da quantidade de erros na porta de entrada *Num_erros* também da transição anterior. Para que este bloco funcione corretamente acumulando os erros do sistema, a porta de entrada *enable_acumulador* deverá estar sempre habilitada exceto no início de cada estado para não acumular os erros provenientes do estado anterior.

4.2.6. Bloco CONTADOR

Por último, têm-se o bloco CONTADOR apresentando três portas de entrada: *relógio*, *reset_contador* e *enable_contador* e uma porta de saída: *bits_recebidos* (10 bits). Este bloco tem como objetivo realizar a contagem dos bits recebidos em cada estado de `alinhamento_comma` e `lock_state` garantindo que cada um destes estados recebam 1000 bits (100 palavras de 10 bits), com a porta de saída *bits_recebidos* contando desde 0 a 100. Como já dito anteriormente, na centésima palavra, as portas de entrada *reset_contador* e *reset_acumulador* do contador e acumulador respectivamente, são habilitadas

indicando o final de um estado e o início de outro com esse mesmo processo de contagem.

4.3. Conclusão

Ao longo deste capítulo introduziu-se um conjunto de ferramentas computacionais necessárias para o desenvolvimento deste módulo receptor. Em seguida, detalharam-se os blocos desenvolvidos destacando suas características e por fim suas funcionalidades associadas ao hardware foram discutidas.