UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

RENATO UBIRATAN REIS MÔCHO

# Circuitos Assíncronos na Plataforma FPGA

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Prof. Dr. André Inácio Reis
Orientador

Porto Alegre, agosto de 2006.

**CIP – CATALOGAÇÃO NA PUBLICAÇÃO**

Môcho, Renato U. Reis

Circuitos Assíncronos na Plataforma FPGA/ Renato U. R. Môcho – Porto Alegre: Programa de Pós-Graduação em Computação, 2006.

132 f.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2006. Orientador: André Inácio Reis.

1. Circuitos síncronos. 2. Circuitos assíncronos. 3. FPGA. 4. DIMS. 5. NCL. I. Reis, André Inácio. II. Título.

# AGRADECIMENTOS

Agradeço a minha mãe e minha avó, Renilce de Oliveira Reis e Nilce de Oliveira Reis, pelo carinho e pela fé em mim.

Meu muito obrigado ao meu orientador, André Inácio Reis que, com muita sabedoria e amizade, desempenhou papel fundamental na realização deste trabalho.

Agradecimento especial ao professor Renato P. Ribas pelo auxílio com os circuitos que foram implementados e o texto da dissertação.

E a Deus, que sem o consentimento, nenhum de nós estaria aqui.

# SUMÁRIO

# LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| CPLD | Complex Programmable Logic Devices |
| DIMS | Delay Insensitive Minterm Synthesis |
| FPGA | Field Programmable Gate Arrays |
| LUT | Look-up Table |
| NCL | NULL Convention Logic |
| PLA | Programmable Logic Array |
| PLD | Programmable Logic Devices |
| VHDL | Very Hardware Description Language |

# LISTA DE FIGURAS

# LISTA DE TABELAS

# RESUMO

Os circuitos digitais cada vez mais são exigidos quanto ao desempenho e modularidade nos processos dos dias atuais. Para resolver estes processos, o comércio utiliza largamente circuitos digitais síncronos, que se baseiam no controle do sincronismo através de um relógio central. Esses circuitos, apesar de serem de fácil implementação e terem uma metodologia já conhecida, apresentam limitações quando se considera a distribuição dos sinais de sincronismo, a interferência do meio e os possíveis atrasos. Os circuitos assíncronos apresentam uma solução natural a essas exigências, uma vez que, possuem independência do sinal do relógio e toda sua construção é modular. Este trabalho apresenta um estudo comparativo de alguns estilos de projetos para construção de circuitos assíncronos utilizando dispositivos programados por lógica, PLDs, utilizando ferramentas de síntese lógica comerciais para circuitos síncronos. Esses circuitos assíncronos são descritos em VHDL para as células Muller, elementos M de N, registrador assíncrono, somadores e circuitos mais complexos em anel assíncrono e implementados em CPLDs e FPGAs. Os circuitos mais complexos são construídos em quatro estilos de projeto para os circuitos dos somadores: Descrição comportamental com indicação forte do sinal, DIMS, NCL e derivação a partir de circuito combinacional síncrono. Através dessa avaliação foi possível verificar as tendências do custo de elementos de programação e atrasos para realização de cálculos, frente aos circuitos síncronos similares.

**Palavras-chave:** Circuitos Síncronos, Circuitos Assíncronos, FPGA, DIMS, NCL

# ABSTRACT

This work presents a study about the implementation of asynchronous circuits on programmable devices platform. It investigates four different ways of implementing asynchronous circuits, including implementation of several different circuits in platforms provided by three different manufacturers. The implemented asynchronous circuits have a very poor performance when compared to their synchronous counterpart. However, this was expected as the platforms used were developed to be used with synchronous designs. The contributions of this work are in the following areas. First, it was described in detail how to implement VHDL code for self-timed designs. Second, different design were implemented to test the VHDL descriptions in the chosen platforms. Third, by comparing four different asynchronous styles, it is possible to find a style that is the more adequate for use in current FPGAs. Fourth, by analyzing the results obtained, it was possible to derive some conclusions on why asynchronous designs are so costly for these platforms and derive some suggestions to be used in the implementation of asynchronous FPGAs.

**Keywords:** FPGA, VHDL, asynchronous circuits, implementation

# 1 INTRODUÇÃO

Os circuitos digitais produzidos hoje demandam cada vez mais lógica, maiores freqüências e áreas cada vez menores. O projeto desses circuitos envolve custos elevados, pessoal capacitado, tecnologia e técnicas que representam novos desafios ao comércio e indústria de circuitos integrados.

Esses circuitos utilizam na sua grande maioria metodologia síncrona. Ou seja: existe um relógio central que faz o sincronismo entre todos os blocos lógicos do circuito. O período desse sinal de sincronismo é previamente definido levando-se em consideração o pior atraso de *throughput* que esses circuitos possuem, limitando o avanço da informação para blocos com atrasos menores. Por outro lado, nos circuitos assíncronos, o sincronismo é feito através de protocolos de comunicação entre blocos do circuito. Esses sinais informam aos blocos lógicos envolvidos no protocolo que o dado de informação é válido e está disponível para bloco seguinte. Uma outra característica é que esse sinal de sincronismo não é enviado para todos os blocos do circuito, limitando as operações para aqueles que estiverem realizando a comunicação naquele instante. Logo, como esses circuitos possuem diferenças acentuadas quando comparados com os circuitos síncronos, é possível enumerar as características mais importantes:

Modularidade: Esta característica é inerente nos circuitos assíncronos e uma preocupação constante dos circuitos síncronos. Nos circuitos síncronos esta preocupação se dá por causa da freqüência máxima determinada pelo bloco lógico que possui o maior atraso. Nos circuitos assíncronos esta modularidade dos blocos que compõem o circuito é inerente ao projeto pois a comunicação entre os módulos é feita através de um protocolo *handshake*, assim novas versões de um bloco sempre vão funcionar pois o protocolo de comunicação irá esperar o tempo que for necessário para os dados ficarem prontos.

Baixo consumo de potência: Nos circuitos síncronos os sinais do relógio são transmitidos para todos os blocos, independentemente de alguns operarem ou não naquele determinado momento. Já nos circuitos assíncronos, por não existir a necessidade do sinal do relógio, eles não necessitam realizar transições desnecessárias. Logo, esses circuitos são naturalmente quiescentes, evitando gastos adicionais de energia.

Baixo ruído: Nos circuitos síncronos o sinal do relógio principal pode interferir com os sinais de informação que trafeguem no mesmo, gerando ruídos por emissões eletro-magnéticas. Essas emissões, causadas pelas transições dos sinais do relógio, causam variações de consumo de correntes concentradas no tempo. Já nos circuitos assíncronos essas interferências ficam reduzidas pela ausência do sinal do relógio e assim o consumo de corrente está distribuído ao longo do tempo, gerando menos ruído.

Performance: O circuito assíncrono pode ser mais rápido que um síncrono visto que, o sinal que deverá informar ao próximo bloco de lógica que o cálculo foi efetuado, não

necessita esperar por um atraso máximo, como nos circuitos síncronos. Nos assíncronos, a partir do instante que o cálculo foi efetuado e o resultado está disponível, é enviado um sinal ao estágio seguinte avisando que essa informação já pode ser repassada. Esta avaliação é feita pelo tempo médio de atraso nos blocos combinacionais.

A partir das características apresentadas, podemos deduzir que existe a possibilidade de melhorar o desempenho e a eficiência dos sistemas digitais através do uso de circuitos assíncronos. Esses circuitos, normalmente, podem ser dimensionados através dos seguintes critérios: O modelo de atraso adotado, ou seja, como os blocos do circuito estarão dispostos e quais os possíveis caminhos que os sinais usados no protocolo de comunicação e os sinais de informação terão, o tipo de protocolo de comunicação entre os blocos lógicos, e finalmente, o formalismo para descrever o circuito que pode ser resumido no tipo de linguagem para descrição do hardware.

O desenvolvimento de circuitos FPGAs e CPLDs oferecem plataformas para uma rápida prototipagem de circuitos integrados síncronos VLSI (TODMAN, 2005). Já os circuitos assíncronos, por outro lado, não possuem uma plataforma bem definida em CPLD ou FPGA como alternativa desses projetos (SPARSO, 2001). As soluções em lógica programável específicas para assíncronos são baseadas em blocos de grande granularidade que não tem a mesma flexibilidade e grau de configuração que as LUTs baseadas em FPGA e os arrays AND-OR baseados em CPLDs possuem. Por isso, teve início uma análise de design de projeto para se estabelecer a linguagem de alto nível para descrever esse hardware, como VHDL e Verilog. Com isso, muitas arquiteturas propostas em FPGAs para circuitos assíncronos são específicas para um determinado estilo de projeto. Por exemplo, MONTAGE (PAYNE, 1996) é baseado em células arbitradoras e sincronizadoras. A pesquisa de (MEEKINGS, 1966) é baseada em Null Convention Logic, NCL descrito em (FANT, 1996). Em (TEIFEL, 2004) uma arquitetura de fluxo de dados para circuitos assíncronos é proposta. O principal inconveniente é que o projetista a partir de um determinado fluxo de dados especifica a granularidade dos blocos lógicos de forma que se tornem compatíveis com a estrutura do fluxo de dados proposto. A pesquisa de (ZAFAR, 2005) é baseada em implementações micropipelines. O trabalho de (FANG, 2005) apresenta resultados de testes para um pipeline assíncrono em FPGA. A pesquisa em (HUOT, 2005) apresenta um FPGA flexível que pode ser modificado para vários estilos de designs diferentes. Contudo, esses blocos lógicos têm um custo muito elevado pois requerem matrizes de 11x14 pontos de conexão para um bloco lógico além de duas LUTs de sete entradas.

Algumas pesquisas preferem implementar circuitos assíncronos através de design síncrono em FPGA. Por exemplo, a pesquisa feita em (BRUNVAND, 1992) mostra uma comparação entre implementações incluindo um projeto esquemático de circuito assíncrono numa topologia de circuito FPGA Actel (ACTEL, 2003). O trabalho em (NOVAK, 1994) apresenta um co-processador assíncrono parcialmente implementado em FPGA parcialmente em ASIC. Este projeto implementa uma arquitetura fluxo de dados e esta é descrita estruturalmente. A pesquisa em (ORTEGA, 2005) apresenta um projeto de uma ULA auto-temporizada em uma plataforma FPGA, mas novamente o projeto é descrito no nível esquemático não utilizando linguagem de descrição de hardware. No trabalho de (RIGAUD, 2005) existe a proposta de se implementar circuitos assíncronos usando FPGAs regulares. Contudo, o artigo limita-se a mostrar resultados para células Muller de 2 e 3 entradas. Esta pesquisa não utiliza linguagem de descrição de hardware sendo sua principal contribuição a prova informal que as células Muller são elementos livres de hazards se implementadas numa única LUT.

## 1.1 Objetivos do Trabalho

A proposta do trabalho é implementar circuitos assíncronos ou auto-temporizados através da linguagem de descrição de hardware VHDL, nas plataformas FPGA e CPLD, avaliando as possíveis diferenças entre os resultados encontrados para as principais ferramentas comerciais de síntese lógica atualmente em uso.

Para desenvolver este trabalho foi utilizado na investigação o protocolo 4 fases dual-rail. Os testes foram feitos para quatro diferentes estilos de projetos para circuitos assíncronos e através da análise dos resultados foram feitas algumas propostas sobre estruturas assíncronas, como uma LUT assíncrona baseada em FPGA.

A motivação para a realização desse trabalho surgiu da possibilidade de se investigar o comportamento das ferramentas de síntese lógica, normalmente usadas para os circuitos síncronos, para a síntese dos circuitos assíncronos. Ainda a possibilidade de investigar o comportamento de diferentes estilos de projetos, o próprio comportamento nessas ferramentas. Por fim, na conclusão são propostos caminhos que poderão ser seguidos para criação de uma estrutura adaptada para os projetos assíncronos nessas ferramentas de forma a melhorar a síntese dos circuitos assíncronos no futuro. A implementação realizada na plataforma FPGA sendo hoje aquela que apresenta a forma mais prática e rápida de síntese comercial. Esse fator foi decisivo pois representa um limitador para futuras escolhas por plataformas de síntese, uma vez que a crescente tendência em produzir circuitos digitais de baixo custo e de forma rápida pode desestimular outras a utilização de outras plataformas.

## 1.2 Organização do Trabalho

Esse trabalho está organizado em sete capítulos incluindo esta introdução. Inicialmente, no capítulo 2, são apresentados os tipos de circuitos digitais e as diferenças entre circuitos combinacionais e seqüenciais. Nos circuitos seqüenciais, são mostradas as diferenças entre os registradores que funcionam com o sinal do relógio e os registradores assíncronos. Posteriormente, são mostradas as diferenças entre circuitos síncronos e assíncronos através de um circuito em pipeline. Por fim são mostrados os efeitos dos hazards nos circuitos assíncronos.

No capítulo 3 é explicada como é feita a implementação de circuitos assíncronos. Mostra como foram descritas estruturas como os elementos de base: as células Muller, os elementos M de N e registrador assíncrono. Posteriormente, são mostrados os parâmetros necessários para se implementar circuitos assíncronos, como os tipos de codificação, indicação forte dos sinais, lógica monotônica e hazards, assim como os possíveis elementos de atraso. Por fim, são mostrados os estilos de projetos utilizados nesse trabalho como: descrição comportamental com indicação forte do sinal, DIMS, NCL e derivação a partir de circuito combinacional síncrono.

O capítulo 4 é mostrado o conceito dos circuitos programados por lógica ou simplemente PLDs. Iniciando a explicação com os elementos de programação do tipo LUT e macrocélulas e as diferentes tecnologias existentes no mercado, como FPGAs e CPLDs. Posteriormente é mostrada como é feita a síntese para os PLDs. Por fim mostra a linguagem de descrição de hardware VHDL para circuitos assíncronos, descrevendo os principais elementos utilizados no trabalho: as células Muller, elementos M de N e para o registrador assíncrono.

O capitulo 5 apresenta uma avaliação comparativa entre os circuitos combinacionais e sequenciais simulados nesse trabalho, para os diferentes estilos mostrados no capítulo 3. Nos circuitos combinacionais são mostrados resultados para os Full Adder e RCAs. Já nos circuitos sequenciais são mostrados resultados para o registrador assíncrono. Ainda nos circuitos sequenciais, é feita uma comparação quantitativa dos circuitos síncronos e assíncronos, sendo também avaliada as diferenças de performance.

O capítulo 6 mostra os resultados dos diferentes tipos de circuitos assíncronos sintetizados nesse trabalho. Os circuitos do tipo contador, do tipo divisor inteiro, do tipo divisor de resto, do tipo MMC, do tipo MDC e ainda para o cálculo da raiz quadrada. São apresentadas as tabelas de valores para FPGAs e CPLDs e ainda os percentuais de acréscimo que os resultados para uma ferramenta apresentou com relação ao menor resultado dentro dentre os diferentes estilos de projeto.

Por fim, no capítulo 7 são apresentadas as conclusões e considerações finais sobre o trabalho.

# 2 COMPARAÇÃO QUALITATIVA ENTRE CIRCUITOS SÍNCRONOS E ASSÍNCRONOS

## 2.1 Resumo

Este capítulo descreve qualitativamente as diferenças entre circuitos síncronos e assíncronos. As diferenças são feitas quanto ao comportamento, ignorando-se detalhes de implementação.

## 2.2 Introdução

Circuitos digitais podem ser classificados conforme a resposta que apresentam aos sinais de entrada. Conforme este critério, várias classificações são possíveis. Para um entendimento dos circuitos assíncronos, estudados nesta dissertação, usamos a classificação apresentada na figura 2.1.

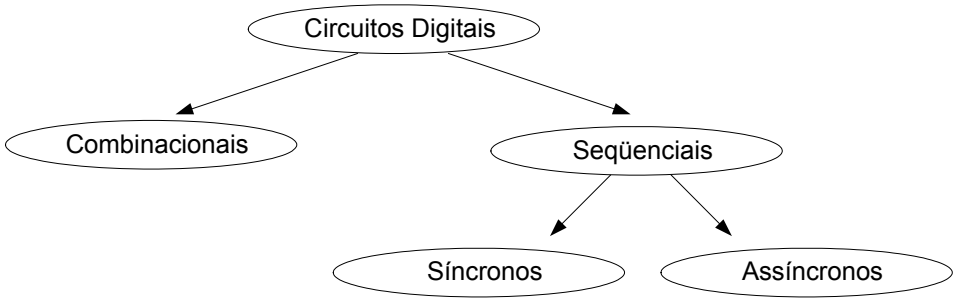



Figura 2.1: Classificação dos circuitos digitais

## 2.3 Circuitos combinacionais

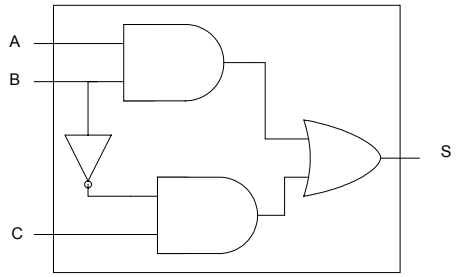
Circuitos combinacionais são aqueles cuja resposta funcional depende apenas dos valores lógicos aplicados nas entradas do circuito. Ao ser aplicado um vetor de entrada, o circuito sempre responde da mesma forma após um atraso correspondente ao tempo necessário para estabilização das saídas. Este tipo de circuito pode ter sua funcionalidade representada por meio de tabelas verdade. Uma tabela verdade associa o vetor aplicado na entrada com os valores de saída correspondentes. Na figura 2.2 pode

ser visto um exemplo de circuito combinacional com a representação da função S através de sua tabela verdade e o circuito equivalente com as portas lógicas.

Na figura é possível verificar o comportamento bem definido da saída S em função das suas entradas.Nesse circuito o início do cálculo ocorre a partir do instante que os sinais chegam nas suas entradas e o atraso total para que o resultado esteja disponível nas suas saídas é apenas a soma dos atrasos nas portas lógicas do circuito. Assim, para uma determinada entrada ABC = (0,0,1) o resultado S=1 estará disponível na saída S após os atrasos somados das portas lógicas AND, OR e o inversor. É importante notar nesses circuitos que existe um atraso desde a chegada dos sinais nas entradas e o resultado pronto apresentado na saída S. Durante esse período, a saída S poderá apresentar qualquer valor que não corresponde ao resultado correto dessas entradas.

| A | B | C | S |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

(a) Tabela Verdade




(b) Circuito Combinacional

Figura 2.2: Tabela verdade e circuito da função S

## 2.4 Circuitos Seqüenciais

Nos circuitos seqüenciais, as saídas não dependem somente dos valores presentes nas entradas. Deste modo, estes circuitos não podem ter sua funcionalidade representada por meio de tabelas verdade. Os valores presentes na saída de um circuito seqüencial dependem das entradas atuais e também da seqüência de valores de entrada aplicados anteriormente, que podem estar sendo armazenados. Este tipo de circuito possui implicitamente capacidade de memória para armazenar um estado interno que depende do conjunto de entradas aplicadas anteriormente. Esta memória pode ser regida por um sinal de relógio, que determina os instantes precisos em que os estados são armazenados na memória. Neste caso temos os circuitos síncronos, já que a operação de armazenamento dos dados é feita de modo sincronizado com um sinal de relógio. Uma outra alternativa possível se dá quando os elementos de memória são controlados através de protocolos de comunicação. Neste caso a memorização pode ser feita em qualquer instante de tempo, quando os sinais relativos ao controle do protocolo de comunicação determinam o momento do armazenamento dos dados. Teremos então, os circuitos assíncronos, já que o armazenamento de dados é feito por demanda sem estar atrelado a um sinal de temporização e sincronismo, como é o caso do sinal de relógio

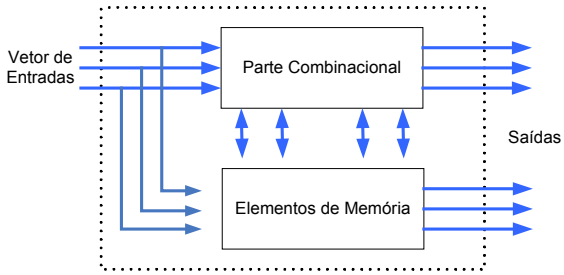
nos circuitos síncronos. A seguir são discutidas propriedades de funcionamento destes dois tipos de circuito.

Na figura 2.3 pode ser vista tabela de relacionamento e o modelo do circuito seqüencial. O circuito possui um estado atual A e um próximo estado B, indicado na primeira linha da tabela. Nessa situação uma entrada X fará com que a saída tenha resultado J. A mesma entrada X apresentada ao circuito quando o estado atual é B e o próximo estado é A fará com que a saída tenha resultado K, mostrado na segunda linha da tabela. A mudança do valor da entrada não irá alterar o resultado na saída quando o estado atual e próximo são iguais, como pode ser visto na terceira e quarta linhas. O estado atual A e próximo estado A fará com que o circuito armazene o valor de saída nesse estado apesar da entrada mudar para Y. Assim também acontece para o estado atual B e próximo estado B, armazenando o valor anterior K e o apresentado na saída.

Nesses circuitos, o tempo necessário para que o resultado esteja disponível nas suas saídas é a soma dos atrasos nas portas lógicas e dos atrasos que os sinais de estado atual e próximo estado terão até ficarem disponíveis nas entradas dessas portas.

| Entrada | Estado Atual | Próximo Estado | Saída |
|---|---|---|---|
| X | A | B | J |
| X | B | A | K |
| Y | A | A | J |
| Y | B | B | K |

(a) Tabela de Transição de Estados




(b) Circuito Seqüencial

Figura 2.3: Exemplo de circuito seqüencial.

### 2.4.1 Circuitos síncronos

Os circuitos síncronos são controlados por um sinal de relógio. Este sinal de relógio determina os tempos em que o circuito deve armazenar o estado anterior até que seja gerado o próximo estado. Para entender o funcionamento do circuito síncrono é fundamental conhecer o comportamento do seu registrador. O seu comportamento se baseia na cópia dos valores de entrada para a saída e na memorização desses valores. O que define o instante em que o registrador síncrono deverá copiar esses valores ou armazena-los é o sinal do relógio central. Na figura 2.4 é possível ver a seqüência de ações na saída de um registrador síncrono sensível à borda de subida. Quando o sinal do relógio passa de 0 para 1, o valor de entrada está autorizado a ser escrito no registrador. Nos demais casos, o registrador irá armazenar o último valor copiado.

| Relógio Central | Ação na Saída |
|---|---|
| 0 → 0 | Dado Memorizado |
| 0 → 1 | Copia o valor de entrada para a saída do registrador |
| 1 → 0 | Dado Memorizado |
| 1 → 1 | Dado Memorizado |

Figura 2.4: Descrição do registrador síncrono

Na figura 2.5 é possível verificar um modelo de anel para um circuito síncrono. O único estágio é formado por um bloco CC, que representa uma lógica combinacional, e um bloco de armazenamento que é um registrador controlado pelo sinal do relógio, descrito por REG. Considerando nesse exemplo que o bloco que representa a lógica combinacional seja um somador que realiza a operação de soma 1, ou seja, soma o valor 1 ao valor de entrada. O registrador REG é responsável pela cópia dos valores de entrada no instante em que a borda de subida do sinal do relógio é acionada. Os pontos P1 e P2 correspondem, respectivamente, aos valores de saídas do registrador e do bloco combinacional. Nesses pontos são mostrados os fluxos dos sinais, distribuídos ao longo do tempo.

Vamos partir da situação inicial de que o valor na saída de REG é P1 igual a 29 e na saída de CC é P2 igual a 30. Quando ocorrer a primeira borda de subida do sinal do relógio, o valor que estava disponível em P2 é copiado para o registrador REG e mostrado na sua saída. Nesse instante, começa o cálculo no bloco combinacional relativo à operação de soma 1 ao valor 30. Digamos que para realizar essa soma o bloco combinacional demore 10t, sendo t unidade de tempo. O cálculo deverá estar pronto antes da próxima borda de subida do sinal do relógio e durante esse período de tempo, esse valor ficará aguardando no registrador. Quando ocorrer a segunda borda de subida, o valor 31 é copiado para o registrador REG e mostrado na sua saída. O valor no ponto P1 passa a ser 31 e o bloco combinacional poderá iniciar o novo cálculo. Considerando que o valor 31 somado a 1 deva levar um tempo bem maior que a soma anterior, uma vez que irão existir vários carregamentos de bits menos significativos para os mais significativos, para só depois o resultado final ficar pronto. Vamos atribuir para essa soma um tempo de 30t. Logo, o cálculo irá se estender por um período de tempo maior que o anterior e só depois quando o resultado estiver estável e pronto é que a borda de subida deverá ser acionada. Com isso já podemos concluir que nos circuitos síncronos o período mínimo do sinal do relógio é o do maior atraso que será obtido nos cálculos desse circuito. Assim, todo e qualquer cálculo deverá ser efetuado dentro desse período mínimo e o resultado deverá estar pronto antes que a borda de subida seja acionada.

Prosseguindo com o raciocínio, na terceira borda de subida do sinal do relógio o registrador REG irá copiar o valor 32 do ponto P2 e o mostrará na sua saída, indicada pelo ponto P1. Nesse instante irá iniciar um novo cálculo pelo bloco combinacional.

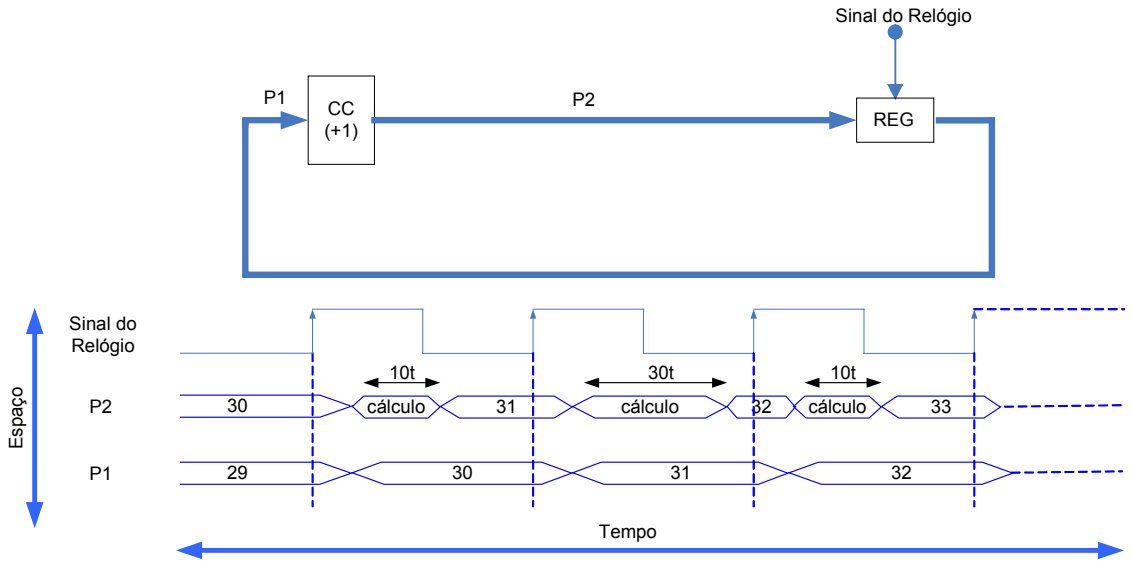



Figura 2.5: Modelo de circuito síncrono

Outra característica importante do funcionamento dos circuitos síncronos é que o único sinal que deve se manter estável durante a ativação da borda de subida é o sinal do relógio. Isto é necessário uma vez que esse sinal é que irá indicar quando o dado deverá ser copiado no registrador de cada estágio. Se por acaso esse sinal apresentar problemas durante esse processo, uma informação errada será copiada e repassada para os estágios seguintes.

### 2.4.2 Circuitos assíncronos

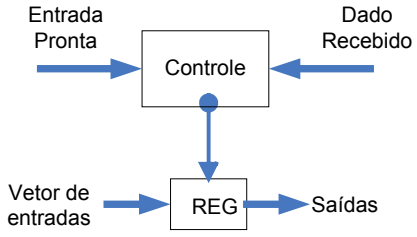
Os circuitos assíncronos são controlados através de protocolos de comunicação, sendo independentes do sinal do relógio, conforme (HAUCK, 1995) e (SPARSO, 2001). Um possível protocolo de comunicação que poderia ser utilizado nesses circuitos é baseado no envio de um sinal de fim de cálculo que avisaria que o cálculo foi realizado pelo bloco combinacional e no recebimento de um sinal que indicaria que esse resultado já poderia ser copiado no registrador e encaminhado para o estágio seguinte. Os circuitos assíncronos são implementados através de estágios onde cada um deles é composto por um bloco funcional, bloco de armazenamento e um circuito de detecção de fim de cálculo. O bloco funcional é responsável pelo cálculo propriamente dito, é um circuito combinacional onde os valores de saídas dependem exclusivamente dos valores de entrada. O bloco de armazenamento é responsável por guardar a informação oriunda do bloco funcional, é controlado por um sinal que avisa quando o resultado do bloco funcional já pode ser copiado. O circuito de detecção de fim de cálculo é responsável por verificar se os valores de saída do bloco funcional estão corretos para só assim serem copiados para o bloco de armazenamento. Para avisar que os valores de saída do bloco funcional estão corretos ele envia um sinal para o controle do circuito que se encarrega do protocolo de comunicação.

### *2.4.2.1 Registrador Assíncrono*

O registrador assíncrono tem o seu comportamento baseado no sinal que lhe é enviado pelo circuito de controle. Na figura 2.6 exemplificamos o seu funcionamento através dos sinais "entrada pronta" e "dado recebido". Esses dois sinais representam os sinais de *pronto* que são enviados pelos blocos de detecção de fim de cálculo que existem nos estágios do circuito assíncrono. No instante em que o circuito de controle recebe os sinais de "entrada pronta" igual a 0 e "dado recebido" igual a 1, significa que o circuito de controle recebeu a informação de apagar o valor armazenado, enviando para o registrador assíncrono um sinal de habilita igual a 0. Quando o circuito de controle recebe o sinal "entrada pronta" igual a 1 e "dado recebido" igual a 0, significa que o circuito de controle recebeu a informação que deve copiar os valores na entrada do registrador para a sua saída.. Já na situação em que o circuito de controle recebe o sinal "entrada pronta" e "dado recebido", ambos iguais a 0 ou ambos iguais a 1, significa que o circuito de controle recebeu a informação de que deverá manter o valor armazenado no registrador.

| Entrada Pronta | Dado recebido | Ação do controle | Valor do dado |
|---|---|---|---|
| 0 | 0 | Dado memorizado | Válido ou Vazio |
| 0 | 1 | Apagar o valor memorizado | Vazio |
| 1 | 0 | Autoriza a escrita do dado | Válido |
| 1 | 1 | Dado memorizado | Válido ou Vazio |

(a) Descrição das ações do controle




(b) Esquema do Circuito

Figura 2.6: Descrição das ações do registrador assíncrono

### *2.4.2.2 Estágio Assíncrono*

O conjunto de blocos, mostrados na figura 2.7 compõem um estágio nos circuitos assíncronos. Esse estágio é formado, obrigatoriamente, pelo bloco de armazenamento e pelo circuito de detecção de fim de cálculo, podendo ou não, ser ainda composto pelo bloco combinacional, visto que, podem existir estágios que não necessitem realizar cálculo algum. Para que um circuito assíncrono possa ter operacionalidade é necessário que o protocolo de comunicação de dados obedeça ao preceito de que irá somente enviar um sinal para apagar a informação armazenada de um registrador quando essa informação já tiver sido copiada para o registrador seguinte. Esse preceito é atendido quando o valor do sinal de pronto do estágio ao qual pertence o registrador for igual a 0

e o valor do sinal do pronto do estágio posterior ao seguinte for igual a 1, significando que a informação já foi calculada pelo bloco operacional do próximo estágio.

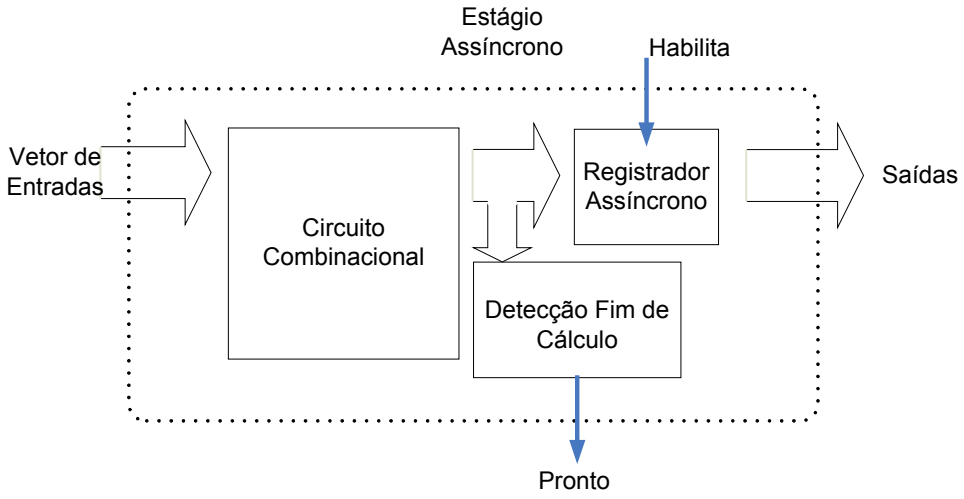



Figura 2.7: Blocos no estágio de um circuito assíncrono

### 2.4.3 Anel Assíncrono

O anel assíncrono corresponde ao conjunto de estágios distribuídos de forma que os valores de saída do último estágio correspondam aos valores de entradas do primeiro. Uma característica do anel assíncrono é que uma vez inseridos os valores de entradas não é mais possível inserir outros valores, ou seja, uma vez iniciado o seu cálculo, a quantidade de valores que circulam no anel é fixa.

Nos circuitos assíncronos existe uma limitação quanto ao número mínimo de estágios em um anel. Para que o dado possa circular pelos estágios e dar continuidade ao processo de sincronização, realizado pelo protocolo de comunicação, o anel deverá ter, além do estágio com o valor do dado de informação, um estágio em que existe a ausência de informação e um estágio em que o dado de informação foi previamente copiado. Com isso, teríamos um estágio efetivamente com o valor de informação, um estágio que possui uma cópia antiga dessa informação e um estágio vazio. Ou seja, no mínimo três estágios.

Na figura 2.8 é mostrado um anel assíncrono de três estágios. O primeiro estágio é formado pelo circuito combinacional CC que realiza a operação de soma 1 ao valor de entrada, pelo registrador REG1 responsável por copiar a informação de P1 e mostra-la na sua saída e pelo circuito de detecção de fim de cálculo, DET1 que é responsável pelo sinal que indica quando valores válidos estão presentes na saída do bloco combinacional CC. Os outros dois estágios não possuem blocos combinacionais sendo formados apenas por seus registradores e pelos circuitos de detecção de fim de cálculo. Sendo eles os circuitos REG2 e DET2, no segundo estágio e REG3 e DET3, no terceiro estágio. As saídas dos registradores REG1, REG2 e REG3 são indicadas pelos pontos P2, P3 e P4, respectivamente. A saída do bloco combinacional CC é indicada por P1. Os sinais que

indicam quando existe um valor válido nos pontos P1, P2 e P3 são PP1, PP2 e PP3, respectivamente.

Vamos partir da situação inicial em que o bloco CC acabou de realizar um cálculo, disponibilizando o resultado na sua saída indicada por P1. Consideremos que o primeiro valor seja 31 e que em todos os outros pontos exista o valor vazio ou ausência de informação. Quando o valor 31 for mostrado na entrada do circuito de detecção de fim de cálculo DET1, o mesmo irá acionar o sinal PP1, indicando que existe dado válido na entrada do registrador REG1. O circuito de controle irá receber os sinais de PP1 igual a 1 e de PP3 igual a 0. Com isso irá habilitar o registrador REG1 para escrever o valor que está disponível na sua entrada. Esse valor será mostrado na saída do registrador REG1, indicada por P2. Quando o valor 31 for mostrado na entrada do circuito de detecção de fim de cálculo DET2, o mesmo irá acionar o sinal PP2, indicando que existe dado válido na entrada do registrador REG2. O circuito de controle irá receber os sinais de PP2 igual a 1 e de PP1 igual a 1, com esses sinais o registrador armazena o último valor amostrado que é vazio. Quando o sinal PP1 for baixado pela ausência de informação em P1, o controle passará a receber os sinais de PP2 igual a 1 e de PP1 igual a 0, com isso irá habilitar o registrador REG2 para escrever o valor que está disponível na sua entrada. O registrador REG2 copia o valor 31 e o mostra na sua saída, representada por P3. Quando o valor 31 for mostrado na entrada do circuito de detecção de fim de cálculo DET3, o mesmo irá acionar o sinal PP3, indicando que existe dado válido na entrada do registrador REG3. O controle receberá o sinal de PP1 igual a 0 e de PP3 igual a 1, fazendo com que o valor que estava armazenado no registrador REG1 seja apagado. Quando o valor que era mostrado na saída do registrador REG1 for apagado, o sinal PP2 do circuito de detecção de fim de cálculo DET2 será baixado. O circuito de controle receberá o sinal de PP2 igual a 0 e PP1 igual a 0, fazendo com que o registrador REG2 armazene o valor anterior. O sinal PP3 com valor 1 juntamente com o sinal PP2 com valor 0 fará com que o circuito de controle habilite o registrador REG3 para escrever o valor que está disponível na sua entrada. O valor na entrada do registrador será copiado para a sua saída, sendo mostrado na entrada do circuito combinacional CC através do ponto P4. A partir desse instante se inicia o cálculo para que seja somado 1 ao valor na entrada de CC. Diante dessa operação de cálculo demorar mais tempo que a anterior por causa dos carregamentos que serão necessários para a soma de 1 ao valor 31, o atraso nessa operação será maior que na anterior. O protocolo de comunicação, no entanto, se adapta a essa diferença de atrasos nas operações pois o cálculo só será copiado para o registrador REG1 quando o sinal do circuito de detecção de fim de cálculo DET1 for acionado. Quando o resultado da operação de soma for copiado para o registrador REG1 e for amostrado na sua saída, o sinal PP2 será acionado e fará com que o circuito de controle apague o valor armazenado no registrador REG3, pois receberá o sinal PP3 igual a 0 e PP2 igual a 1. O anel então irá circular o novo valor, resultante da operação de soma do circuito combinacional.

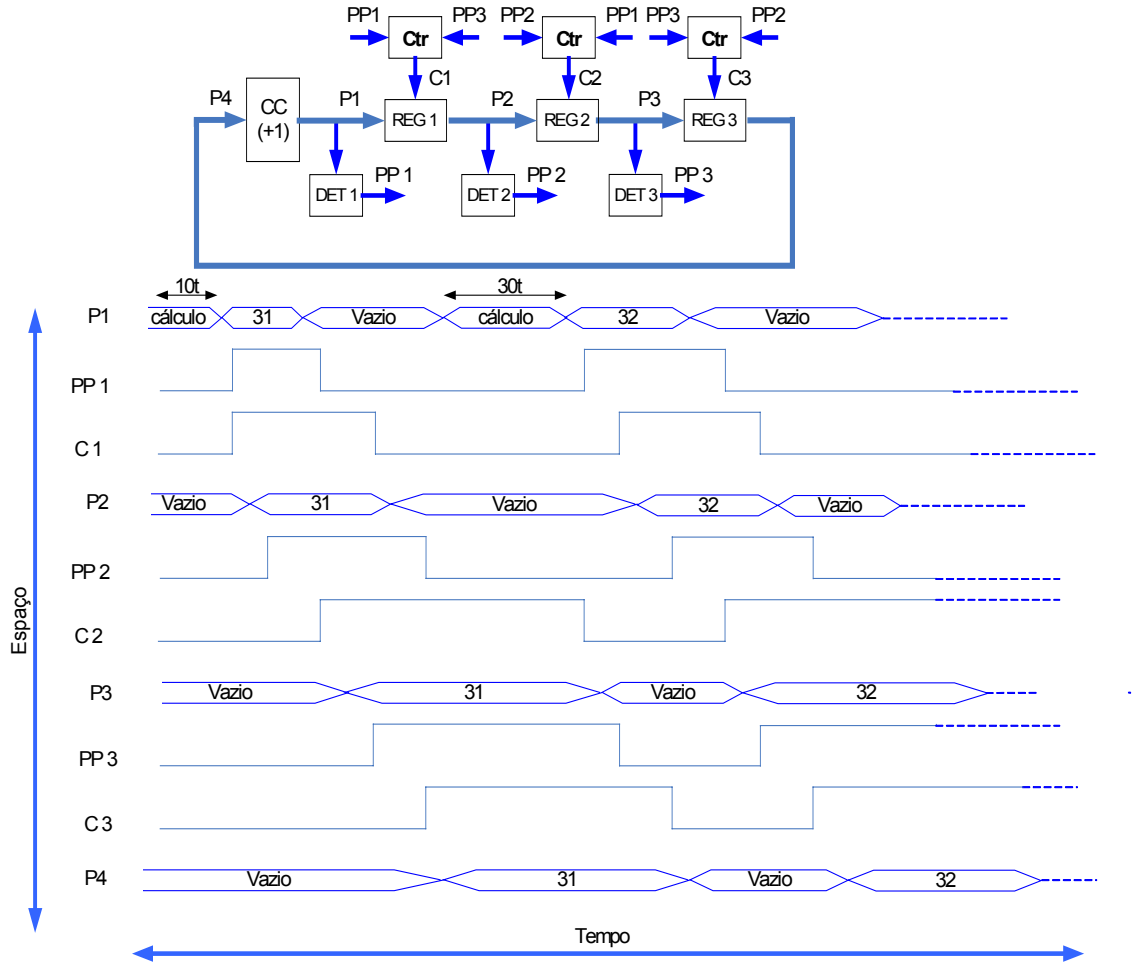



Figura 2.8: Modelo de circuito assíncrono de 3 estágios

Outra característica importante do funcionamento que difere dos circuitos síncronos é que os sinais que controlam os registradores são provenientes da comunicação dos dados nesse circuito. Esse protocolo de comunicação, conseqüentemente, deverá ter seus sinais livres de variações que possam causar erros durante o processo de escrita da informação pelos registradores. Enquanto que nos síncronos a única preocupação era em manter o sinal do relógio estável durante a ativação da borda de subida, nos assíncronos essa preocupação é com relação a qualquer sinal em qualquer instante durante o processo de comunicação da informação pelo circuito.

## 2.5 Efeito dos Hazards

Os hazards são alterações indesejáveis nos níveis dos sinais nos circuitos durante suas transições (HAUCK, 1995) e (SPARSO, 2001). Existem dois tipos de hazards nos circuitos, o estático e o dinâmico. Hazard estático pode acontecer quando ocorre uma transição de mesmo nível lógico nas entradas de um circuito e as suas saídas mostram, durante um pequeno intervalo de tempo, um nível lógico diferente dessas entradas. Já o

hazard dinâmico pode acontecer quando se tem uma mudança de níveis lógicos nas entradas e as suas saídas não passam imediatamente para esses níveis lógicos. Os dois tipos de hazards, estático e dinâmico, podem ser vistos na figura 2.9, por (a) e (b), respectivamente.

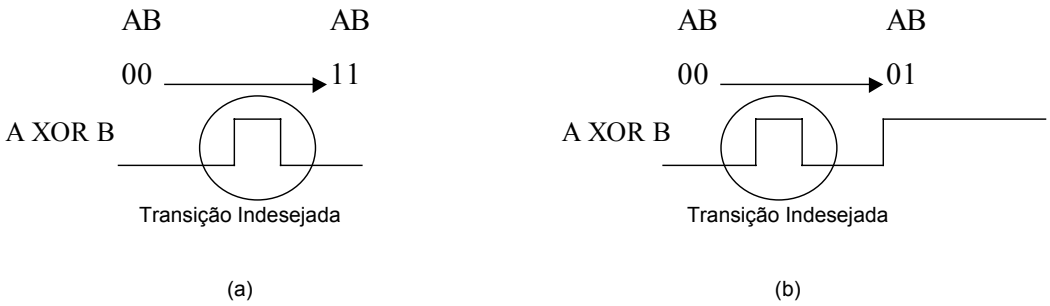



Figura 2.9: Tipos de hazards

### 2.5.1 Nos Circuitos Síncronos

Nos circuitos síncronos a única restrição para alterações indesejadas nos sinais é feita no instante da escrita da informação de entrada para a saída dos registradores. Como o sinal do relógio é o único responsável pelo sincronismo dessa informação, esse sinal deve estar livre de hazards. Ainda com relação ao sinal do relógio, uma vez que o período mínimo é sempre determinado pelo maior atraso no estágio mais lento, o tempo necessário para copiar e armazenar a informação acaba sendo suficiente. Assim, as saídas dos registradores conseguem se estabilizar até que sejam copiadas para o estágio seguinte.

### 2.5.2 Nos Circuitos Assíncronos

Nos circuitos assíncronos, o protocolo de comunicação dos dados é que define quando um registrador deverá copiar ou armazenar uma informação válida. Nesse caso, todos os sinais participam do processo de comunicação havendo a necessidade de que todos estejam livres de hazards. Como a restrição de haver variações indesejadas nos sinais é genérica, existe uma preocupação muito maior que com os circuitos síncronos. Essa preocupação tem como conseqüência uma constante verificação da validade dessas saídas em cada estágio do circuito. Essa verificação é feita através do circuito de detecção de fim de cálculo, como foi indicado nas seções anteriores. Esse circuito envia um sinal para o circuito de controle, indicando que as saídas de cada bloco funcional estão válidas e prontas para serem copiadas nos registradores de cada estágio.

## 2.6 Sumário

Nesse capítulo nós apresentamos uma visão global dos circuitos digitais com a utilização de uma classificação entre combinacionais e seqüenciais. Com a definição de

circuitos seqüenciais, introduzimos os conceitos de circuitos síncronos e assíncronos com a principal diferença entre ambos, o comportamento frente ao sinal de sincronismo dos registradores. Nos síncronos, dependem do maior atraso que um determinado estágio poderá ter para só assim, determinar o período mínimo do relógio central. Nos assíncronos são dependentes dos atrasos individualizados de cada estágio e mais os atrasos que ocorrem nos sinais de controle durante o protocolo de comunicação de dados.

# 3 IMPLEMENTAÇÃO DE CIRCUITOS ASSÍNCRONOS

## 3.1 Resumo

Este capítulo descreve os mecanismos e técnicas necessárias para implementação de circuitos assíncronos. Os conceitos foram organizados de forma que se tenha uma visão geral das características que envolvem o projeto desses circuitos.

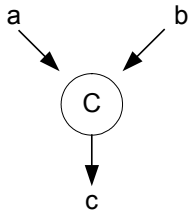
## 3.2 Elementos de base

Os circuitos assíncronos possuem elementos essenciais que compõem a sua estrutura. Eles são responsáveis pela lógica de controle e pelo armazenamento das informações que atravessam os diferentes blocos desses circuitos. Eles são compostos pelas células Muller, as células M de N e ainda os registradores assíncronos.

### 3.2.1 Célula Muller

A célula Muller ou elemento C de Muller funciona como um latch set-reset assíncrono, (HAUCK, 1995), (SPARSO, 2001) e (RIGAUD, 2002). Ele pode ser descrito da seguinte forma: Quando as entradas são 0 a saída é 0, e quando as entradas são 1 a saída é 1, para qualquer outra combinação de entradas as saídas permanecem as mesmas, ou seja, armazenam o valor anterior. Conseqüentemente, um observador vendo a saída mudar de 0 para 1 pode concluir que todas as entradas possuem valor 1 naquele instante; similarmente, um observador vendo a saída mudar de 1 para 0 poderá concluir que todas as entradas são 0 naquele momento. Na figura 3.1. pode ser visto o diagrama de estados e o símbolo usado para representar o elemento C de Muller com duas entradas. O comportamento da saída Cout na célula Muller é representado em termos das entradas a e b e do estado anterior de saída Cin pela seguinte expressão booleana. Isso se dá através da realimentação do circuito, ligando Cout ao Cin.

$$Cout = [Cin\ (a+ b)] + (a\ .\ b)$$



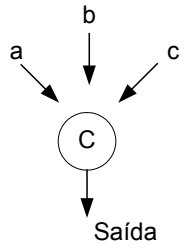

(a) Célula Muller de duas entradas

| a | b | c |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | memoriza |
| 1 | 0 | memoriza |
| 1 | 1 | 1 |

(b) Tabela de Sinais da Célula Muller de duas entradas

Figura 3.1: Símbolo utilizado e diagrama de estados da célula Muller de duas entradas

Na figura 3.2. é mostrado o símbolo e a tabela verdade para a Célula Muller de três entradas. A operação segue o mesmo princípio descrito anteriormente. Quando todas as entradas a, b e c são 0 a saída é 0, e quando as entradas a, b e c são 1 a saída é 1, para qualquer outra combinação das entradas a, b e c as saídas permanecem as mesmas.




(a) Célula Muller de três entradas

| a | b | c | saída |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | memoriza |
| 0 | 1 | 0 | memoriza |
| 0 | 1 | 1 | memoriza |
| 1 | 0 | 0 | memoriza |
| 1 | 0 | 1 | memoriza |
| 1 | 1 | 0 | memoriza |
| 1 | 1 | 1 | 1 |

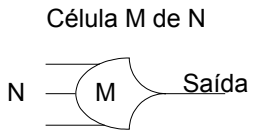
(b) Tabela de Sinais da Célula Muller de três entradas

Figura 3.2: Símbolo utilizado e diagrama de estados da célula Muller de três entradas

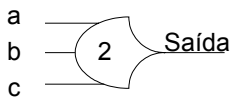
### 3.2.2 Célula M de N

A célula M de N, também chamada de *threshold gate*, (FANT, 1997) e (KUANG, 2003), apresenta um comportamento similar ao da célula Muller com a diferença que a sua saída deverá mudar de 0 para 1 quando apenas M das N entradas existentes estejam em 1. Ou seja, o seu funcionamento ocorre da seguinte maneira: Quando M entradas das N existentes estiverem em 1 a sua saída irá para 1, e quando todas as N entradas forem para 0 a sua saída irá para 0, para qualquer outra combinação de entradas a saída permanece com o valor anterior. A partir daí percebemos que essas células serão idênticas às células Muller quando M for igual a N e terão um comportamento idêntico a uma porta lógica OR quando M for igual a 1. Na figura 3.3 temos o símbolo utilizado

para descrever essas células. Para uma célula *threshold gate* M de N (DERTOUZOS, 1965), M entradas válidas para o total de N entradas existentes, sendo sempre M ≤ N.




(a) Símbolo utilizado para célula M de N



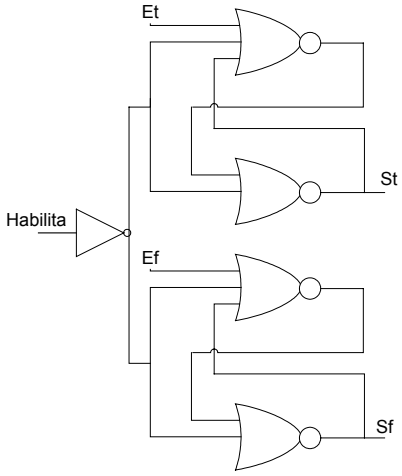

| a | b | c | saída |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | memoriza |
| 0 | 1 | 0 | memoriza |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | memoriza |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

(b) Célula 2 de 3 com sua Tabela de Sinais

Figura 3.3: Célula *threshold gate* de M=2 e N=3

### 3.2.3 Registradores assíncronos

O registrador assíncrono, mostrado na figura 3.4, é responsável por armazenar a informação entre os blocos funcionais nos circuitos assíncronos. O seu controle é feito pelo sinal de habilita que é enviado pelo protocolo de comunicação do circuito. A tabela dos sinais que caracteriza o registrador assíncrono é mostrada na figura. De acordo com essa tabela, quando o sinal de habilita for 0, todas as saídas serão 0, independentes dos sinais nas entradas. Essa situação muda quando o sinal de habilita for 1. Isto faz com que as entradas sejam relevantes para se determinar os sinais que estarão disponíveis nas saídas do registrador. Quando as duas entradas forem diferentes uma da outra, a saída St mostrará o sinal na entrada Et e a saída Sf mostrará o sinal da entrada Ef. Quando as duas entradas forem 0, as saídas irão mostrar os valores anteriores, ou seja, irão mostrar o valor que foi armazenado anteriormente pelo circuito. A situação em que habilita e as entradas Et e Ef são 1 é prevista como uma indeterminação pela tabela de sinais.




(a) Esquema do registrador assíncrono

| Habilita | Et | Ef | St | Sf |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | memoriza | memoriza |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | INVÁLIDO | |

(b) Tabela dos sinais do circuito

Figura 3.4: Representação do registrador assíncrono com tabela dos sinais

## 3.3 Circuitos combinacionais assíncronos: parâmetros

O conhecimento dos parâmetros que envolvem os circuitos assíncronos é necessário para uma visão global do funcionamento desses circuitos. Os tipos de codificação que podem ser utilizadas, os conceitos de lógica de fim de cálculo e os estilos de projetos são visto de forma resumida para introduzir o leitor no ambiente de trabalho que foi feita essa dissertação.

### 3.3.1 Codificação de dados

Nos circuitos assíncronos não existe um sinal de sincronização global como o relógio central nos circuitos síncronos. Logo, se faz necessário que os blocos que constituem esses circuitos tenham algum tipo de comunicação entre si. Essa comunicação controla o fluxo de informação entre esses blocos e gera os sinais que serão utilizados pela lógica de controle nos blocos de armazenamento, (HAUCK, 1995) e (SPARSO, 2001).

O protocolo de comunicação escolhido terá o papel do sinal de relógio nos circuitos assíncronos com a grande diferença que não irão chavear o circuito como um todo e sim apenas a parte que estiver envolvida efetivamente da computação num determinado instante de tempo.

Os blocos nos circuitos assíncronos se comunicam através dos sinais *request* e *acknowledge*. O sinal *request*, ou melhor, “estou enviando dados” diz que o bloco quer se comunicar com outro e possui dados prontos para serem enviados e o sinal de *acknowledge*, ou melhor, “dado recebido” diz que o bloco recebeu os dados que foram enviados.

Nesses protocolos de comunicação, os dados podem estar codificados em *single* ou *dual-rail*. Na codificação *single-rail* cada bit de informação trafega por apenas um único caminho, por outro lado, na codificação *dual-rail* cada bit de informação trafega por dois caminhos distintos. Há ainda a codificação que utiliza o reset como espaçador, ou seja, é enviado o dado válido e um dado vazio.

### *3.3.1.1 Single Rail*

Nesse tipo de codificação o transmissor gera um sinal avisando que está enviando o dado e o receptor gera um sinal dizendo que recebeu o dado que lhe foi enviado. Um esquema que representa a comunicação single rail de dados de n bits é mostrado na figura 3.5. A codificação single rail pode ser feita em duas fases ou quatro fases. Em duas fases ocorre que o transmissor gera o sinal estou enviando os dados e após o receptor ter recebido este dado, envia um outro correspondente a dado recebido para o transmissor. Para o de quatro fases, o transmissor envia um sinal dizendo que está pronto para enviar os dados, o receptor envia um outro sinal dizendo que pode enviar. O transmissor inicia o envio do dado de informação. Quando o receptor já tiver recebido toda a informação, o receptor envia um sinal avisando que o dado foi recebido e só assim, o transmissor poderá parar de enviar.

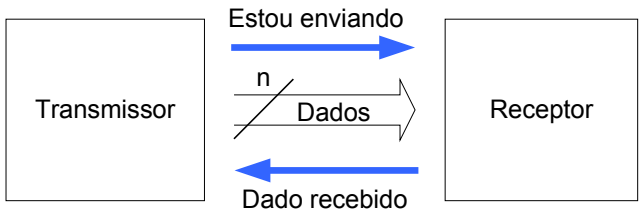
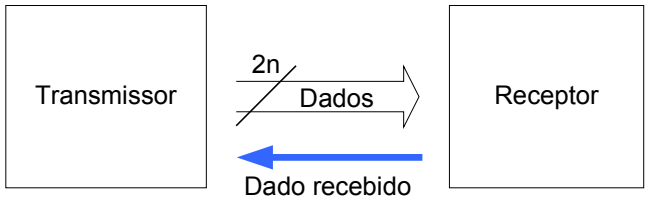



Figura 3.5: Esquema de codificação single rail

### *3.3.1.2 Dual Rail*

No esquema de codificação *dual-rail* existem dois caminhos de comunicação para representar cada bit da informação. Existem vários esquemas de codificação *dual-rail*, sendo que todos combinam codificação dos dados e os protocolos de sinalização. Não existe um sinal explícito para avisar que o transmissor está enviando os dados, como foi visto no esquema de codificação *single-rail*. São usados (2n+1) caminhos de comunicação, como mostra a figura 3.6. Cada bit de dados X é codificado através de dois sinais binários $X_t$ e $X_f$ . Caso o X=0, temos a codificação $X_t$ =0 e $X_f$ =1. Caso o X=1, temos a codificação $X_t$ =1 e $X_f$ =0. Existe ainda a codificação $X_t$ =0 e $X_f$ =0, usada para indicar a ausência de dados e realizar a pré-descarga de blocos combinacionais antes da realização de um cálculo. Esta codificação é eficaz para a detecção de fim de cálculo, que é feita pela identificação de um valor válido nos pares de bits de saída de um bloco.




(a) Esquema dos sinais entre os blocos do circuito

| X | $X_t$ | $X_f$ |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |
| Ausente | 0 | 0 |

(b) Tabela de codificação dual rail

Figura 3.6: Esquema dos sinais entre os blocos e tabela de codificação dual rail

Para o uso da codificação dual rail mais comum a quatro fases, considera-se que cada um dos caminhos de comunicação deve passar pelo estado vazio ou pré-descarga dos blocos (ausência de dados) antes de ir para o próximo estado válido. Na primeira fase todos os n bits de dados dos dois caminhos que são enviados pelo transmissor sairão do estado vazio ou ausência de dados para o estado 0 ou 1 válido. O receptor detecta a chegada do novo grupo de dados válidos quando todos os pares dos dois caminhos de comunicação tiverem saído do estado vazio. Esta detecção implícita substitui o sinal de estou enviando explícito que existia na codificação single rail. A segunda fase consiste no envio do sinal de dado pronto do receptor para o transmissor, informando que o dado foi recebido. Na terceira fase, uma vez recebido o sinal de dado pronto pelo transmissor, esse inicia a passagem de todos os pares de bits dos dois caminhos de comunicação do estado válido para o estado vazio, realizando a pré-descarga dos blocos. Já na quarta fase, o receptor abaixa o sinal de dado pronto, uma vez que deixa de receber os pares de bits de dados válidos.

#### *3.3.1.3 Reset como espaçador (dado válido vs dado vazio)*

Nesse tipo de codificação teremos a utilização da ausência de informação ou vazio para separar os dados de informação na codificação dual rail. A detecção de pares de bits válidos nos dois caminhos de comunicação codificados faz com que seja enviado pelo receptor um sinal de dado recebido para o transmissor. Já a ausência de informação ou todos os pares de bits dos dois caminhos no estado vazio fará com que o receptor deixe de enviar o sinal de dado recebido.

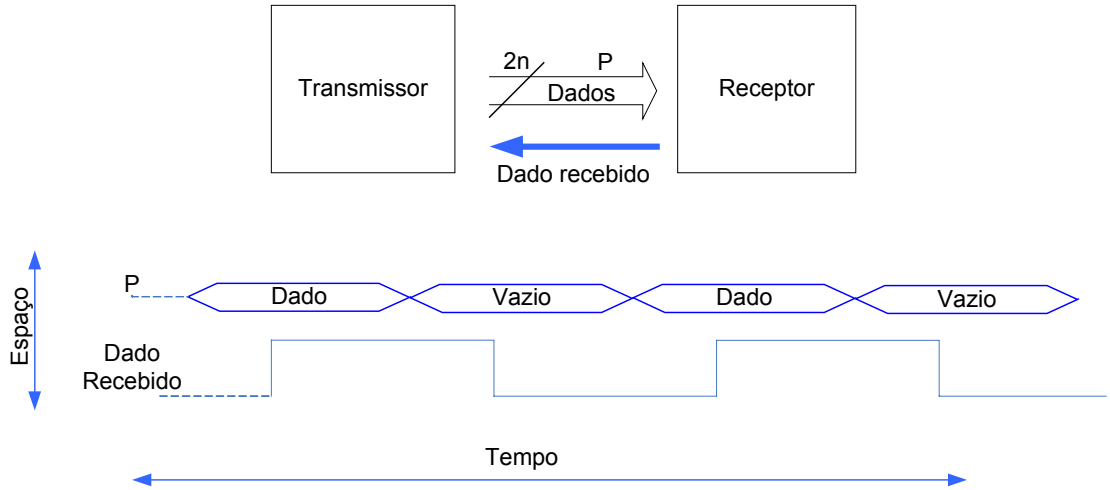



Figura 3.7: Esquema da codificação reset como espaçador

### 3.3.2 Lógica de fim de cálculo

Uma vez que não existe sinal de relógio principal nos circuitos assíncronos, existe a necessidade de que os blocos funcionais ou aqueles que realizam os cálculos no circuito avisem aos demais blocos que terminaram esse cálculo, (SPARSO, 2001). Dos muitos métodos de detecção de fim de cálculo existentes o de nosso interesse é o de verificação da validade dos dados contidos no barramento num determinado instante de tempo.

#### *3.3.2.1 Elemento de atraso*

Esse método para indicar que o cálculo foi realizado consiste na estimativa do tempo utilizado por um bloco combinacional ao realizar uma determinada operação de cálculo no circuito. Na figura 3.8 é mostrado como é possível indicar o fim de cálculo através de elementos de atraso. A informação é levada ao bloco combinacional através dos vetores de entradas. Nesse instante, é gerado um sinal de estou enviando para o bloco seguinte. Esse sinal irá atravessar um conjunto de inversores em série ou uma cadeia de buffers que lhe atribuirão um atraso t2. Quando os vetores de entradas são colocados a disposição do bloco combinacional é iniciado o cálculo e após o tempo t1 é colocado o resultado na saída desse bloco. O projetista deve então dimensionar o atraso para que o sinal de estou enviando, ao atravessar a cadeia de inversores, seja maior que o atraso que o bloco combinacional levará para calcular o resultado da operação, ou seja, que t2 seja maior que t1. Assim, o bloco seguinte poderá receber o sinal de Pronto relativo ao fim de cálculo no bloco combinacional.

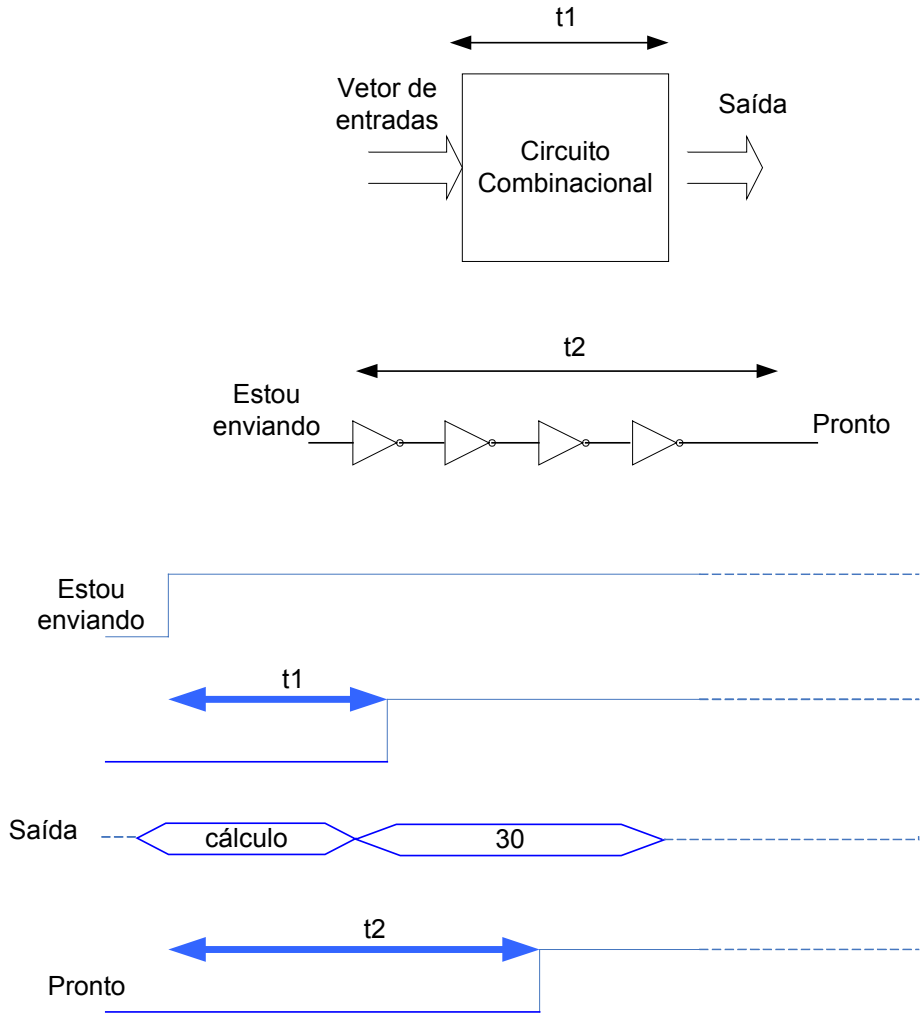



Figura 3.8: lógica de fim de cálculo com elementos de atraso

### *3.3.2.2 Detecção de dado válido em dual rail*

Esse método se baseia na detecção de fim de cálculo a partir da verificação se todos os n bits de dados dos dois caminhos de comunicação são válidos num determinado instante. Na figura 3.9. é mostrado um esquema do circuito combinacional com o circuito de detecção de fim de cálculo utilizado. O resultado é levado ao circuito de detecção de fim de cálculo, uma vez verificada a validade dos pares de bits nos dois caminhos de comunicação, irá gerar o sinal de Pronto.

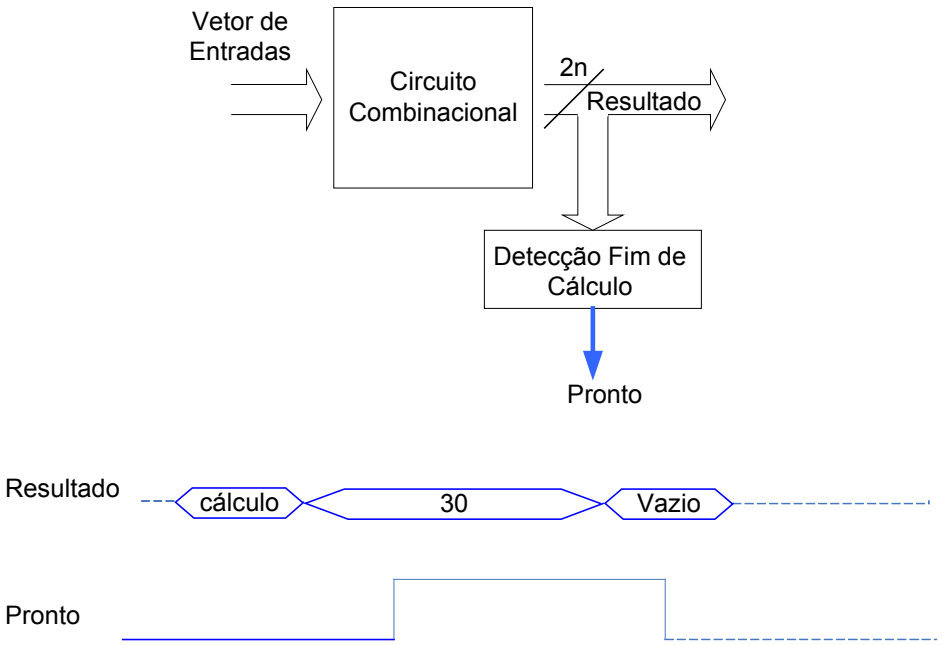



Figura 3.9: Detecção de fim de cálculo através de codificação dual rail

### 3.3.3 Lógica monotônica e hazards

Nos circuitos seqüenciais, existem diferentes blocos por onde passam as informações dos dados. Os sinais podem percorrer diferentes caminhos até chegarem às entradas de um determinado bloco, a fim de realizar uma determinada operação de cálculo. Nos circuitos síncronos os atrasos de diferentes caminhos são compensados pelo período mínimo de sinal do relógio. Esse período é projetado para que se leve em consideração o atraso necessário até que as saídas desses blocos combinacionais se estabilizem e que seja armazenado um resultado válido pelo registrador.

Nos circuitos assíncronos não existe o sinal de relógio para determinar o tempo que essas saídas serão copiadas para o registrador. Logo, alguma informação incorreta poderia ser armazenada até que as saídas do circuito que realiza esses cálculos se estabilizem. Na figura 3.10 (a) é possível verificar a ocorrência de hazards devido ao atraso que uma entrada poderia sofrer ao chegar num circuito combinacional, (SPARSO, 2001) e (HAUCK, 1995). O atraso t, devido ao inversor colocado na entrada B, provoca um hazard na saída S do circuito.

Esse problema poderia ser eliminado através da substituição por portas monotônicas. As portas monotônicas não utilizam inversores, sendo determinadas apenas pelas portas AND e OR no circuito. Através da lógica dual rail é possível substituir as entradas e saídas do circuito mostrado na figura 3.10 (a) pelos circuitos equivalentes codificados em dual rail na figura 3.10.(b). Esses circuitos garantem que somente serão produzidas saídas válidas quando todas as entradas forem válidas.

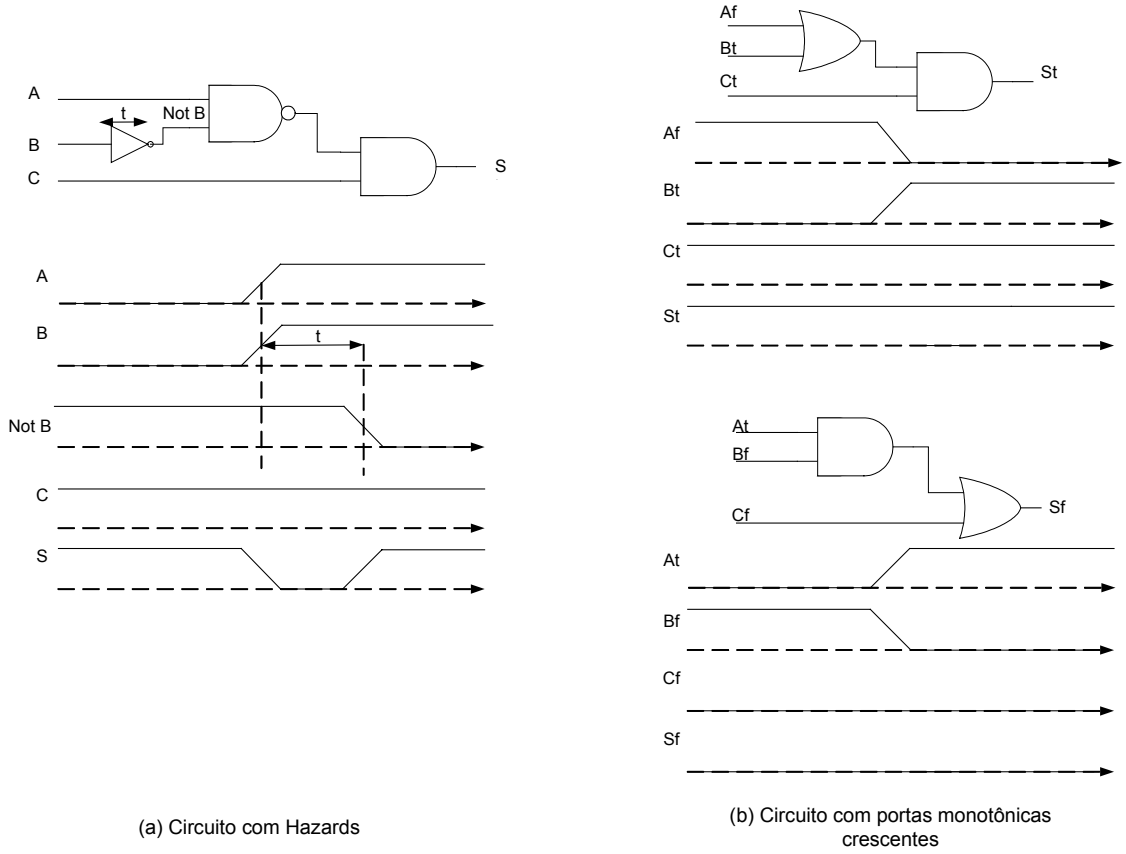



Figura 3.10: Exemplo de circuito com hazards e com portas monotônicas crescentes

### 3.3.4 Princípio da indicação forte

A indicação forte dos sinais em um circuito pode ser resumida como a possibilidade de se garantir validade das suas saídas após ter recebido todas as entradas válidas, (SPARSO, 2001). Na figura 3.11, é possível verificar como funciona um circuito que possui indicação forte dos sinais. Para uma determinada entrada de n bits, cada bit de sinal poderá ser amostrado num diferente intervalo de tempo, sendo possível o aparecimento de sinais inválidos. Com a técnica de indicação forte de sinal, a saída do circuito somente se tornará válida e disponibilizada para o bloco seguinte, quando todas as entradas se tornarem válidas. O mesmo procedimento ocorrerá quando as entradas começarem a ser zeradas. Até que todas as entradas sejam zeradas, a saída desse circuito mostrará o valor armazenado da última entrada válida. Isso garante que as entradas estejam todas zeradas quando as saídas começarem a mudar para zero. Na figura, podemos acompanhar a seqüência de mudanças que ocorrem até que as saídas deixem o estado vazio ou ausência de dados para o próximo estado válido. Em (1) os bits de entrada começam a mudar do valor vazio para o valor válido, repare que durante esse processo as saídas permanecem no estado vazio, não ocorrendo mudança alguma. Em (2) uma vez que todas as entradas estão válidas pode iniciar a mudança nas saídas. Em (3) essa mudança de estado vazio para o estado válido começa a ocorrer levando os bits de saída a iniciarem gradativamente a mudança para a saída válida. Em (4) é mostrada a passagem dos bits de entrada válida para a entrada vazia, os bits de entrada começam a

mudar o valor de entrada válida para entrada vazia, novamente a saída permanece no estado válido não sofrendo qualquer mudança. Em (5) uma vez que todas as entradas estão vazias já é possível iniciar a mudança das saídas. Essa mudança ocorre em (6), onde os bits de saída iniciam gradativamente a mudança do estado válido para o vazio.

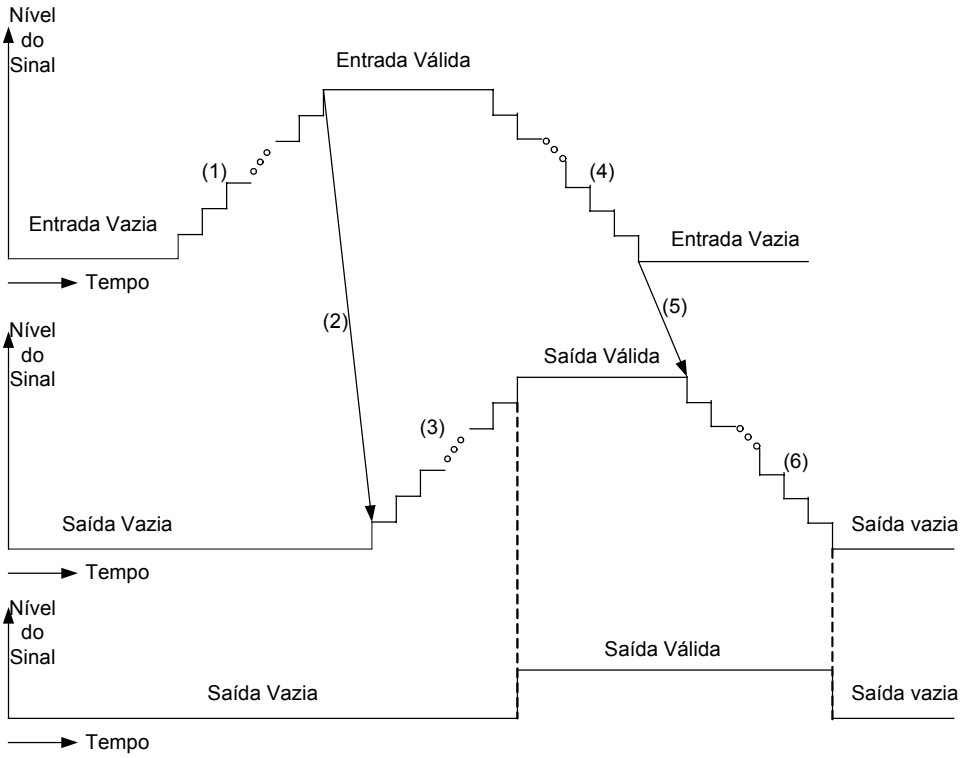



Figura 3.11: Exemplo de indicação forte dos sinais

### 3.3.5 Relacionando os conceitos

Através de associações entre os conceitos apresentados anteriormente é possível se chegar a implementações em hardware que sejam livre de hazards e obedeçam ao princípio da indicação forte. Na seção 3.3.3 foi ilustrado um exemplo de lógica monotônica livre de hazards. É possível perceber no entanto que a derivação deste circuito monotônico, só foi possível porque se passou de uma implementação single rail para uma implementação dual-rail. Note que uma lógica monotônica positiva é livre de hazards para transições começando a partir do vetor de entradas que contém todas as entradas zeradas. Assim, o uso do espaçador com vetores nulos, também contribui na implementação de lógica livre de hazards, já que isto garante que todas as transições irão sempre se iniciar com todas as entradas iguais a zero. O uso de lógica dual rail permite também a detecção de fim de cálculo, já que é possível saber quando o valor de saída representa um dado válido. A associação destes conceitos permite a produção de circuitos livres de hazards com detecção de fim de cálculo. O uso de células Muller permite a implementação do princípio de indicação forte. Para isto é usada uma lógica adicional que permite tirar proveito de uma célula Muller que é zerada quando todas as entradas primárias estão em zero, garantindo a indicação forte da saída em zero. Do

mesmo modo esta célula Muller pode ser usada para indicação forte, colocando na célula Muller próxima a saída um sinal que só atinge o valor um quando todas as entradas forem válidas. Assim, a indicação forte de saída válida pode também ser implementada. Detalhes dessa implementação serão discutidos na seção 3.5.

## 3.4 Controle de um registrador assíncrono

No capítulo 2 foi apresentado o registrador assíncrono e os sinais que são responsáveis pelo seu controle. Também foi mostrado o funcionamento de um anel assíncrono de 3 estágios utilizando esses mesmos registradores. Nesse capítulo, será mostrado o circuito de controle do registrador e o elemento de base necessário para que o mesmo obtenha os sinais desejados.

O controle do registrador assíncrono é realizado pelo elemento de base chamado célula Muller, mostrado na figura 3.1. Utilizando uma célula Muller de duas entradas é possível obter o sinal de Habilita que irá controlar o registrador. Esse elemento de base recebe como entradas o sinal de Pronto do circuito de detecção de fim de cálculo na saída do bloco combinacional que irá realizar o cálculo a ser armazenado pelo registrador e de um sinal de Pronto do circuito de detecção de fim de cálculo posterior ao seguinte no circuito assíncrono. Quando o sinal de Pronto for 1 e Pronto posterior ao seguinte for 0, o sinal de Habilita será 1 o que fará com que o registrador armazene a informação na sua entrada. Para Pronto igual a 0 e Pronto posterior ao seguinte igual a 1, o registrador irá apagar a informação que havia sido armazenada. Para qualquer outra combinação de Sinais Pronto e Pronto posterior ao seguinte, o registrador mantém a informação armazenada.

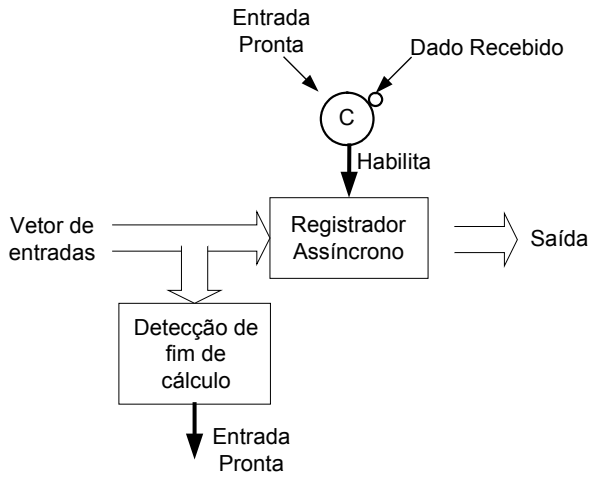



Figura 3.12: Controle realizado num registrador assíncrono

## 3.5 Estilos de projeto

Existem diferentes formas de se projetar circuitos assíncronos e uma preocupação em comum, evitar a ocorrência de hazards. Todos os estilos de projeto possuem um critério para evitar esses sinais indesejáveis utilizando conceitos anteriormente

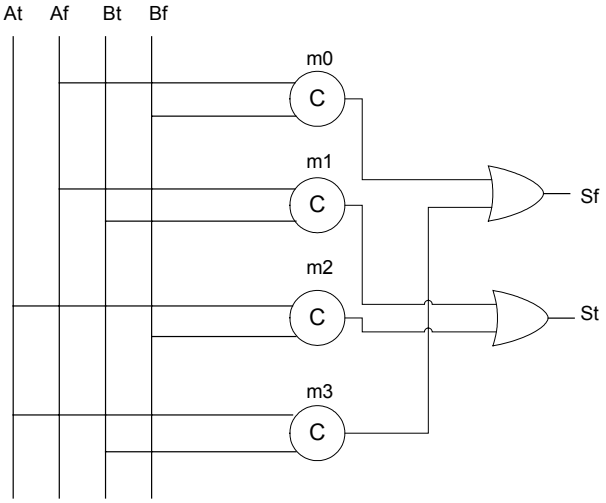
explicados como, por exemplo, detecção de dado válido em dual rail, portas monotônicas crescentes e indicação forte dos sinais.

### 3.5.1 DIMS

Os circuitos em DIMS – Delay Insensitive Minterm Synthesis são baseados em mintermos onde cada mintermo é organizado por células Muller, (MEEKINS, 2002). A figura 3.13 ilustra um exemplo de implementação da porta lógica XOR de duas entradas, codificadas em dual-rail, através da técnica DIMS. Essa técnica forma circuitos insensíveis aos atrasos nos sinais de entrada, evitando dessa forma a propagação de valores inválidos nas suas saídas. A tabela verdade mostrada na figura indica três estados fundamentais para as saídas do circuito: (1) Entradas zeradas produzem saídas zeradas. (2) Valores intermediários não alteram as saídas. (3) Valores válidos produzem saídas válidas.

| A | B | Saída | |
|---|---|---|---|
| Vazio | Vazio | 0 | |
| Valores Inválidos | | memoriza | |
| 0 | 0 | 0 | m0 |
| 0 | 1 | 1 | m1 |
| 1 | 0 | 1 | m2 |
| 1 | 1 | 0 | m3 |

(a) Tabela dos sinais para porta lógica XOR




(b) Porta lógica XOR através da técnica DIMS

Figura 3.13: Porta lógica XOR através da técnica DIMS

A técnica DIMS evita os possíveis hazards nos circuitos assíncronos por utilizar uma característica fundamental da codificação dual-rail: de que todas as transições dos sinais sempre irão partir de valores zerados. Com isso, as transições possíveis deverão partir sempre dos valores A igual a 0 e B igual a 0, tanto para os valores codificados $A_t$ e $A_f$, quanto para $B_t$ e $B_f$. As células Muller também auxiliam para se evitar o aparecimento de hazards, pois possuem a característica de armazenar a última informação válida que lhe foi apresentada. Logo, se houver hazard que apareça em suas entradas, não irá modificar a informação de saída dessa célula.

Essa garantia de que entradas zeradas resultem em saídas zeradas é uma característica de circuitos que possuem indicação forte dos sinais. Como essas células Muller recebem todas as combinações de entradas do circuito, somente irão para zero quando todas essas combinações estiverem em zero.

### 3.5.2 NCL

Os circuitos *Null Convention Logic* utilizam os elementos de base células M de N, vistos anteriormente, ou conhecidas como threshold gates (FANT, 1997) e (KUANG, 2003), ou seja, para um circuito com célula M de N, será necessário possuir M das N entradas válidas para que tenha uma saída válida. Esse valor de M será menor ou igual a N, sendo que se for igual a 1, o circuito NCL terá comportamento de uma porta lógica OR e se for igual a N, terá o comportamento de uma célula Muller. Para se ter uma saída zerada é necessário que todas as suas N entradas sejam zeradas.

Na figura 3.14 é mostrado um exemplo de um circuito full adder em NCL. Esse exemplo foi extraído de (FANT, 1997) e não traz qualquer explicação de como se transforma um circuito em DIMS para um circuito otimizado, utilizando as células M de N para NCL.

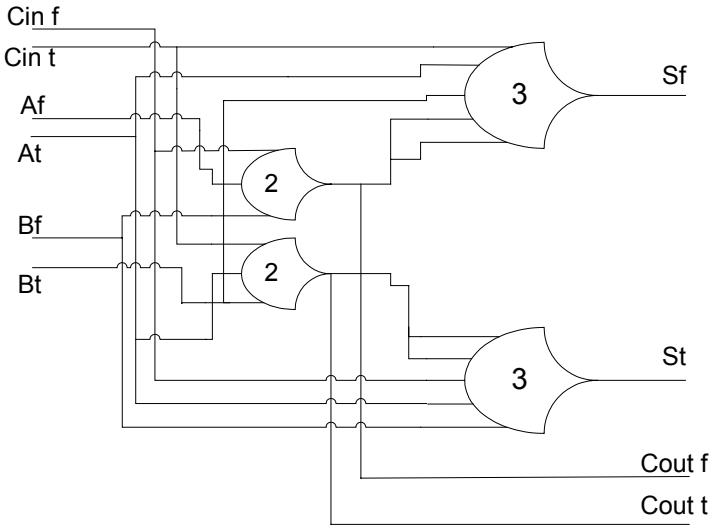



Figura 3.14: Exemplo de um circuito Full Adder em NCL

Esses circuitos possuem indicação forte dos sinais uma vez que as M das N entradas devem estar válidas antes que se sejam geradas saídas válidas. O mesmo ocorre com entradas zeradas. Todas as N entradas deverão ser zeradas para só então serem produzidas saídas zeradas. Um detalhe importante desses circuitos é que por utilizarem uma versão otimizada da Célula Muller, possuem uma maior velocidade para gerar saídas válidas uma vez que necessitam verificar apenas M das N entradas, enquanto que nos circuitos em DIMS, todas as N entradas na Célula Muller deverão ser verificadas.

### 3.5.3 Derivação a partir de circuito combinacional síncrono

Essa técnica utiliza a codificação dual-rail para os sinais do circuito, sendo composta por três módulos principais: O bloco combinacional propriamente dito, codificado em dual-rail; um circuito que fará a verificação se todos os sinais de entrada possuem valores zerados, composto por uma porta lógica OR e um circuito que fará a verificação se todos os sinais de entrada estão válidos, sendo formado por portas AND e uma porta OR. Além desses circuitos auxiliares, existem portas AND e células Muller. Essa lógica é necessária por utilizar os sinais resultantes da verificação das entradas válidas e zeradas e avaliar a indicação forte desses sinais, conforme a figura 3.15.

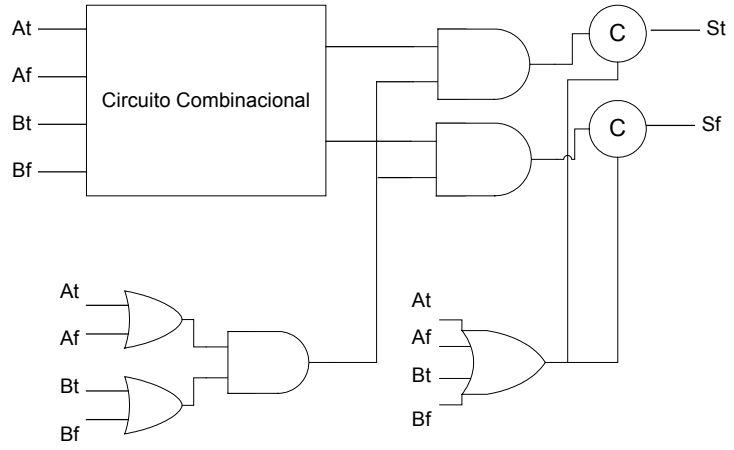



Figura 3.15: Circuito através da técnica fim de cálculo com indicação forte

Isso ocorre na técnica de fim de cálculo com indicação forte através dos circuitos auxiliares, mostrados na figura 3.16 e figura 3.17. O circuito da figura 3.16 detecta se todas as entradas são válidas. Ele é composto pelas portas lógicas AND e OR, que verificarão se os sinais de entrada estão válidos, sendo levados para as portas AND, conforme P2. Essas portas lógicas AND receberão as saídas codificadas do circuito combinacional, conforme P1. Enquanto as entradas não forem válidas, o sinal resultante dessa AND será zero, as saídas dessas portas AND são mostradas em P3.

A figura 3.17 mostra o circuito responsável pela verificação se as entradas foram zeradas. Ele é composto por uma porta lógica OR que leva o inverso desse sinal até uma Célula Muller de duas entradas, mostrado em P4. Essa célula Muller recebe a saída da porta lógica AND, conforme foi indicado no ponto P3. Ela irá funcionar como um bloco de armazenagem da informação válida do circuito. Quando as entradas começarem a ser zeradas, essa informação passará pelo circuito combinacional e pelas portas lógicas AND, para então chegar nas células Muller. No entanto, a saída da célula Muller somente será zero quando todas as entradas forem zero.

Através dessa explicação pode-se deduzir que esse circuito funciona de maneira a ter uma indicação forte dos sinais. Apenas quando todas as entradas forem válidas, irá produzir saídas válidas e somente quando todas as entradas forem zeradas, irá produzir saídas zeradas.

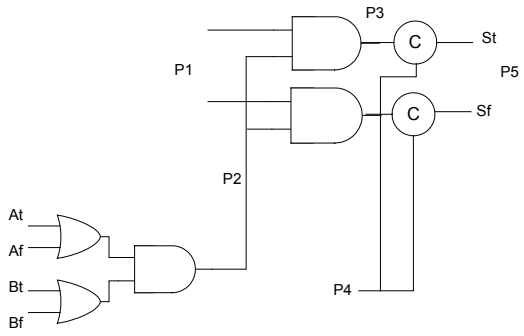



Figura 3.16: Módulo que avalia as entradas válidas.

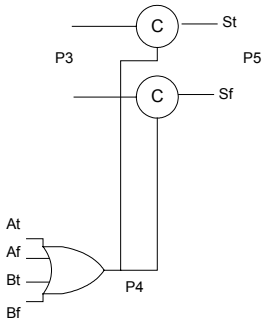



Figura 3.17: Módulo que avalia as entradas zeradas.

### 3.5.4 Descrição comportamental com indicação forte

Outro modo de descrever um circuito assíncrono é fazer uma descrição puramente comportamental, sem definir a estrutura do circuito. Isso pode ser feito em linguagem VHDL usando processos e atribuições condicionais para garantir o princípio da indicação forte. Detalhes desse método serão discutidos no próximo capítulo, quando será apresentada a linguagem VHDL, bem como plataformas baseadas em dispositivos lógicos programáveis.

## 3.6 Sumário

Nesse capítulo foi apresentada a metodologia utilizada na implementação de circuitos assíncronos. Os elementos de base utilizados como as células Muller e células M de N, o registrador assíncrono e os principais parâmetros relacionados com o protocolo de comunicação. Além disso, foram vistas técnicas de detecção de fim de cálculo, através de elementos de atraso e por dado válido em dual rail. O conceito de lógica monotônica e o princípio de indicação forte dos sinais, fundamentais para que se entenda por que esses circuitos são hazard-free.

Finalmente foi falado do tipo de controle utilizado nos registradores dos circuitos assíncronos, essencialmente realizado através de células Muller com os sinais de entradas prontas e dado recebido, como foi visto na seção da detecção de dado válido através de dual-rail e os estilos de projeto que foram feitos para esse trabalho.

# 4 DISPOSITIVOS LÓGICOS PROGRAMÁVEIS

## 4.1 Resumo

Este capítulo apresenta a base de todo o trabalho de síntese dos circuitos dessa dissertação. Mostra uma visão geral dos Dispositivos Lógicos Programáveis ou PLDs com o foco maior para os do tipo FPGA com os elementos de programação Look-up Tables e conceitos importantes como volatilidade e reprogramabilidade. Além disso, mostra a linguagem de programação VHDL em que foram descritos os principais elementos utilizados, como as células Muller, elementos M de N e o registrador assíncrono.

## 4.2 Introdução

Um PLD é um circuito integrado que permite a programação pelo usuário para executar uma determinada função lógica (BROWN, 2005). A principal vantagem que esses dispositivos apresentam é a rapidez de poder implementar um projeto de aplicação específica. No entanto, esses dispositivos apresentam uma falta de flexibilidade de projeto por fazerem parte de uma arquitetura pré-fabricada.

## 4.3 Dispositivos Lógicos Programáveis

Como foi dito, os PLDs podem ser configurados pelo usuário e isso é possível devido ao fato de seus componentes serem formados em suas arquiteturas por elementos de programação. Esses elementos servem para definir a funcionalidade das células lógicas configuráveis assim como para a distribuição dos sinais internos no circuito integrado. Esse texto irá tratar dos elementos de programação mais comuns, como: fusível, antifusível, célula SRAM e flash.

### 4.3.1 Critérios de classificação

As primeiras arquiteturas de PLDs são baseadas no conceito de que toda a função, combinacional ou seqüencial, poderia ser representada sob a forma de soma de produtos. Logo, essas arquiteturas foram constituídas basicamente por um array ou matriz de portas lógicas AND (array AND), onde eram conectadas as entradas da função lógica e um array de portas lógicas OR (array OR) por onde poderiam ser obtidas as saídas dessa função.

Essa combinação das entradas no array AND, gerando os termos produtos, que posteriormente eram enviados para o array OR, onde seriam somados, resultaria na solução da função lógica que se desejaria implementar.

Estes arrays de portas lógicas AND ou OR podiam ser programáveis ou possuírem lógica fixa. Logo, eram usados diferentes tipos de elementos de programação para realizar essa programação, sendo os primeiros dos tipos fusíveis e antifusíveis.

Esses componentes das primeiras arquiteturas permitiam a implementação de circuitos com baixa complexidade, em torno de 1000 portas equivalentes.

As arquiteturas mais recentes de PLDs, como os CPLDs e FPGAs, surgiram para aumentar a capacidade de implementação lógica destes componentes. Nas atuais gerações, os princípios de configuração diferem do simples uso de equações na forma de soma-de-produtos para uma descrição do funcionamento dos circuitos, tornando a implementação de circuitos com mais de 1000 portas equivalentes eficiente.

A tabela 4.1 mostra uma forma de classificação para os tipos de elementos de programação e os tipos de arquiteturas dos dispositivos programáveis. Ainda descreve como são os arrays de AND e OR desses elementos e quais elementos compõem as arquiteturas dos dispositivos.

Tabela 4.1: Divisão dos tipos de dispositivos lógicos programáveis

**DISPOSTIVOS LÓGICOS PROGRAMÁVEIS**

| Arquiteturas dos elementos de programação | | | | Arquiteturas dos dispositivos | |
|---|---|---|---|---|---|
| **PROM** | **PAL** | **PLA** | | | |
| Array AND fixo | Array AND programável | Array AND programável | **LUT** | **CPLD** (Vários elementos do tipo PROM, PAL ou PLA) | **FPGA** (Vários elementos do tipo LUT) |
| Array OR programável | Array OR fixo | Array OR programável | | | |

### 4.3.2 Volatilidade

Os elementos de programação são responsáveis por individualizar os dispositivos programáveis através de lógica. São os responsáveis por dizer quais os pontos serão fundamentais para a organização dos caminhos de roteamento e até mesmo na definição das funções utilizadas nos blocos lógicos. Além das características elétricas e físicas existem, porém, duas características que são de grande importância na determinação das vantagens que algumas famílias de componentes poderão ter em relação a outras: a volatilidade e a reprogramabilidade.

A volatilidade é a característica que os elementos de programação possuem de perder a sua configuração quando o sinal de alimentação não está mais presente. Isso mostra a dependência que esses elementos possuem com relação a esse sinal. Já os elementos reprogramáveis possuem a possibilidade de reconfiguração de sua lógica. Essa propriedade é uma forma a mantê-lo protegido contra uma eventual ausência do sinal de alimentação ou de erros de programação que possam ocorrer.

Os mais comumente conhecidos, dentre os vários tipos de elementos de programação são: fusíveis, antifusíveis, células SRAM e Flash. Os fusíveis são elementos não-voláteis, não reprogramáveis e ocupam pequena área na pastilha de silício. Esses elementos são fabricados com tecnologia bipolar, sendo a integração de seus transistores bastante complicada, por esse motivo não acompanharam a evolução dos componentes programáveis. Os antifusíveis possuem o funcionamento inverso dos fusíveis com a diferença de que são fabricados com tecnologia MOS.

As células SRAM ou *Static Random Access Memory* são usadas para controlar transistores de passagem, *transmission gates* e multiplexadores. São dispositivos formados por dois inversores realimentados, sendo portanto voláteis e facilmente reprogramáveis. Já as memórias Flash integram uma evolução das memórias chamadas $E^2$PROM, sendo memórias do tipo ROM que permite ser apagada e reprogramada eletricamente. Na tabela 4.2 é apresentado um pequeno resumo das características dos elementos de programação.

Tabela 4.2: Resumo das características dos elementos de programação

| **Elemento de Programação** | **Volatilidade** | **Reprogramabilidade** |
|---|---|---|
| Fusíveis | Não-voláteis | Não reprogramáveis |
| Antifusíveis | Não-voláteis | Não reprogramáveis |
| SRAM | Voláteis | Reprogramáveis |
| Flash | Não-voláteis | Reprogramáveis |

### 4.3.3 Granularidade e organização interna

A granularidade diz respeito a organização interna do dispositivo lógico-programável, principalmente relacionado ao tipo de elemento de programação usado. É dito que um dispositivo tem uma granularidade maior quando é composto de muitas instâncias de um elemento de programação com grande flexibilidade para implementar pequenas funções lógicas. De maneira similar dispositivo é considerado de menor granularidade quando é composto de elementos de programação maiores, com capacidade de implementar funções lógicas maiores. Obviamente o número de elementos de programação em um dispositivo de menor granularidade é também menor devido a seu tamanho. Quanto à organização interna, os dispositivos lógicos programáveis atuais se dividem em CPLDs e FPGAs, conforme será visto a seguir.

Na figura 4.1 é mostrado um gráfico dos tipos de PLDs e os elementos de programação associados a alguns deles. O enfoque maior foi para os fabricantes de ferramentas comerciais de síntese de PLDs utilizados no trabalho dessa dissertação. É possível verificar que os PLDs de primeiras arquiteturas não são mais empregados pelos fabricantes. Os fusíveis foram incluídos nesse gráfico apenas para classificação. Atualmente, não são mais utilizados elementos de programação baseados em fusíveis.

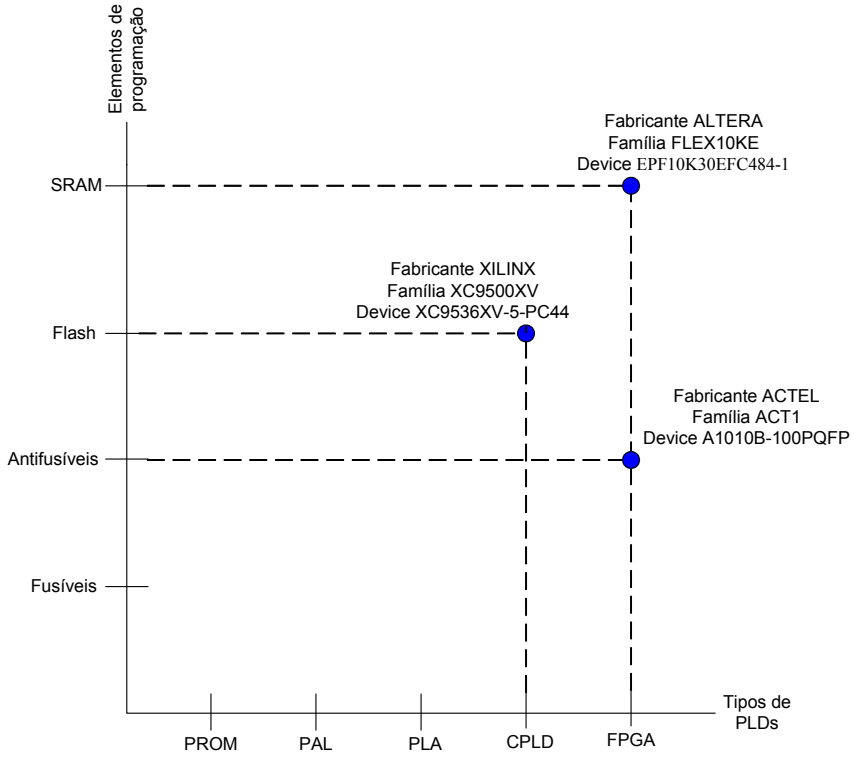



Figura 4.1: Gráfico dos tipos de PLDs e elementos de programação

### 4.3.4 CPLDs

Os Complex PLDs ou simplesmente CPLDs possuem o princípio de funcionamento com origem na arquiteturas dos PALs (arrays AND programáveis e arrays OR fixos). A diferença para essas estruturas mais antigas é que ao invés dos CPLDs possuírem arrays AND programáveis, existem vários sub-circuitos com arquiteturas e capacidade semelhantes a um PAL, que são interconectados por um array programável. Assim, os sub-circuitos podem funcionar de forma independente ou serem interligados dentro do próprio componente.

Os circuitos do tipo CPLDs podem ainda utilizar as arquiteturas PLAs ou PROM, como vimos anteriormente. Para esses tipos, os vários sub-circuitos terão arquitetura e capacidade semelhantes ao PLA ou PROM, respectivamente.

#### *4.3.4.1 PROM*

Este componente foi um dos primeiros PLDs desenvolvido em meados dos anos 70, podendo ser usado eficientemente como memória de alta velocidade por substituir circuitos lógicos combinacionais ou seqüenciais. Podendo também ser chamado de PLE (*Programmable Logic Element*), este componente apresenta um array AND fixo, de forma a apresentar todas as combinações de termos produtos possíveis, com base nas variáveis de entrada. Isso determina que estes componentes são bem empregados em projetos que requeiram um grande número de combinações de entradas e de termos produto por saída, como acontece com lógicas aleatórias. As saídas são as somas dos produtos definidas pelo usuário no array OR programável.

Os PLEs tendem a desaparecer do mercado uma vez que a implementação de lógicas aleatórias de baixa complexidade é feita com maior eficiência por arquitetura PAL.

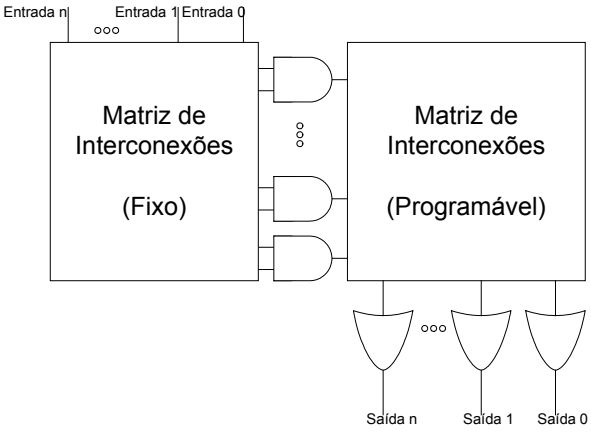



Figura 4.2: Modelo simplificado de uma PROM

#### *4.3.4.2 PALs*

Os Programmable Array Logic ou PALs são os componentes de primeira geração de maior sucesso entre os PLDs. São destinados a aplicações de baixa complexidade pois são os mais eficientes na utilização da área da pastilha, velocidade de funcionamento e custos de projeto e implementação.

O fato de o array AND ser programável torna possível para o componente ter muitas entradas, enquanto o array OR fixo o mantém pequeno e rápido, porém com número limitado de termos produto na saída, como é mostrado na figura 4.3.

Dependendo da tecnologia em que forem construídos os PALs (TTL, CMOS e ECL), o array AND destes podem ser reprogramáveis ou não.

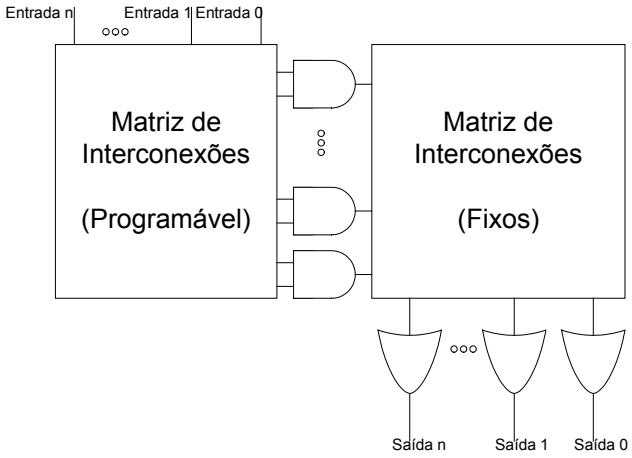



Figura 4.3: Modelo simplificado de um PAL

#### *4.3.4.3 PLAs*

Este componente permite maior flexibilidade na implementação de soma de produtos, uma vez que ambos os arrays AND e OR são programáveis, como visto na figura 4.4, permitindo a implementação de máquinas de estado maiores.

Como neste caso o array AND é programável, não é necessário usarmos todas as combinações possíveis das entradas, como na arquitetura PROM. Isso nos possibilita a implementação de lógicas que tenham um número restrito por simplificações para os termos produto.

Assim como os PROMs os PLAs programáveis pelo usuário tendem a perder mercado para os PALs devido a maior eficiência destes para as implementações com menos de 1000 portas equivalentes.

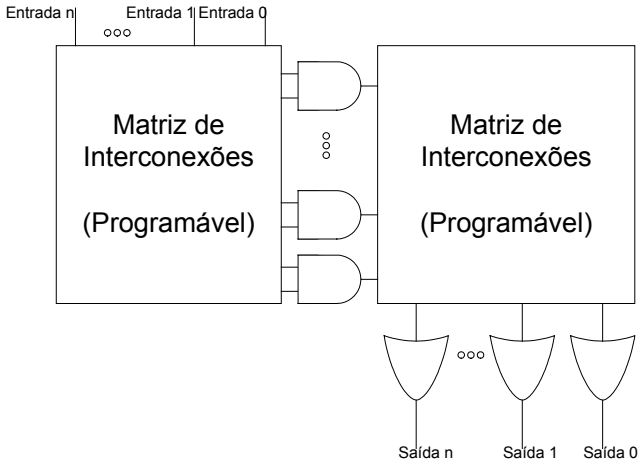



Figura 4.4: Modelo simplificado de um PLA

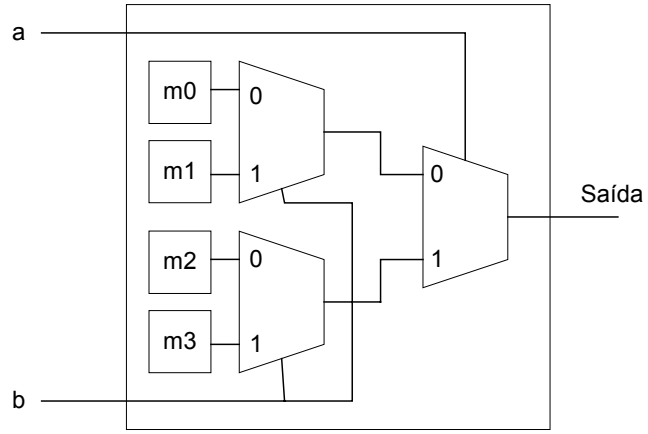
### 4.3.5 FPGAs

Field programmable gate arrays (FPGAs) são muito utilizados nos dias de hoje, tornando-se uma forma dominante em dispositivos programáveis. Em comparação com os arrays programados por lógica, os FPGAs podem ser utilizados para um número maior de funções. Além disso, os FPGAs possuem recursos suficientes para a implementação de sistemas maiores e mais complexos. Outra diferença dos arrays programados por lógica é a possibilidade de utilização de estruturas SRAM ao invés de fusíveis. A utilização de SRAM em FPGAs possibilita que esses circuitos trabalhem de forma similar a um software tornando sua programação mais acessível e aumentando a sua popularidade.

#### *4.3.5.1 O conceito de Look-up Tables*

São estruturas utilizadas nos elementos de programação dos FPGAs na síntese de circuitos. As Look-up Tables podem ser resumidas como pequenas memórias que armazenam os valores de uma tabela verdade da função que se deseja implementar. As entradas dessa função irão servir como endereço para a leitura dos valores armazenados nessa memória. A figura 4.5 mostra a visão interna de uma estrutura Look-up Table com a tabela verdade da função XOR. Nessa estrutura, existem os elementos de memória m0, m1, m2 e m3. Esses elementos poderão ser programados de acordo com a

função que se deseja implementar. Assim, se temos uma função XOR a ser implementada, os seus valores de saída deverão ser programados nesses elementos. As entradas a e b irão selecionar qual dos elementos programados deverá ser apresentado na saída dessa LUT. Por exemplo, m0=0, m1=1, m2=1 e m3=0, caso as entradas sejam a=0 e b=1, o elemento selecionado para ser mostrado na saída da LUT será o m1 igual a 1.




(a) Visão interna da estrutura de uma Look-up Table de 2 entradas

| a | b | Saída |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

(b) Tabela Verdade da função XOR

Figura 4.5: Visão interna da LUT e a tabela de sinais da função XOR

*4.3.5.2 Um exemplo de célula de base*

Um elemento de programação, utilizado em FPGA, não é apenas formado por LUT na sua estrutura. Na figura 4.6 é mostrada a estrutura básica de um elemento de programação genérico. Além da LUT, existe um elemento de armazenamento controlado pelo sinal do relógio principal e um MUX para selecionar entre a saída da LUT ou a saída do registrador. Nos circuitos assíncronos, o elemento de armazenamento controlado pelo relógio principal não é usado. Assim, teremos apenas a LUT e o MUX em atividade, permanecendo o registrador inativo nesses circuitos. O resultado da LUT é levado diretamente para a saída do elemento de programação.

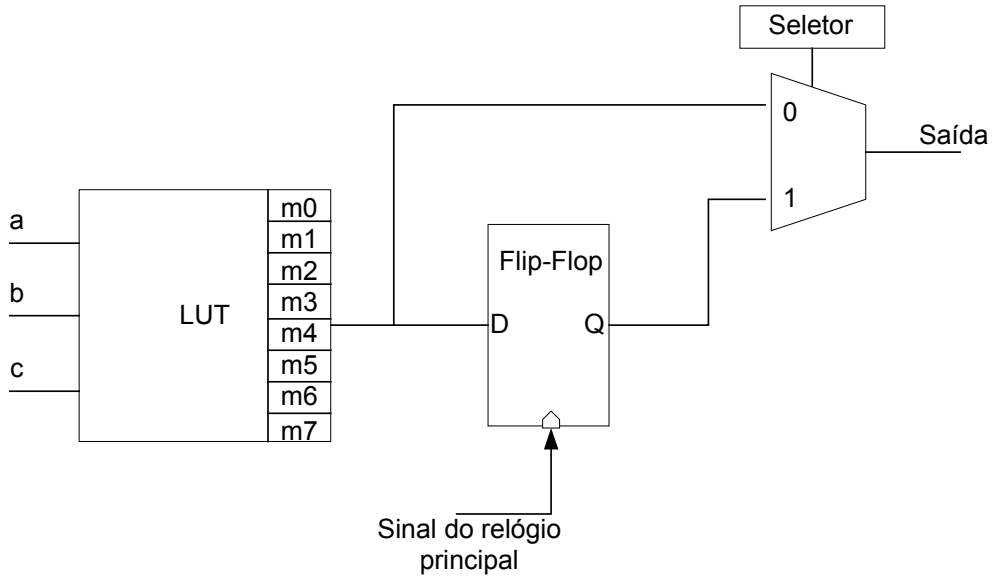



Figura 4.6: Elemento de programação para FPGA genérico

Na figura 4.7. é possível exemplificar a utilização desse elemento de programação em FPGA para Célula Muller de duas entradas. Esse elemento terá uma realimentação da saída Cout como uma das entradas da LUT, além das entradas a e b. Nota-se que o registrador não é utilizado e o MUX é permanentemente selecionado em 0. A conseqüência disso é que o resultado da LUT é diretamente levado para a saída Cout.

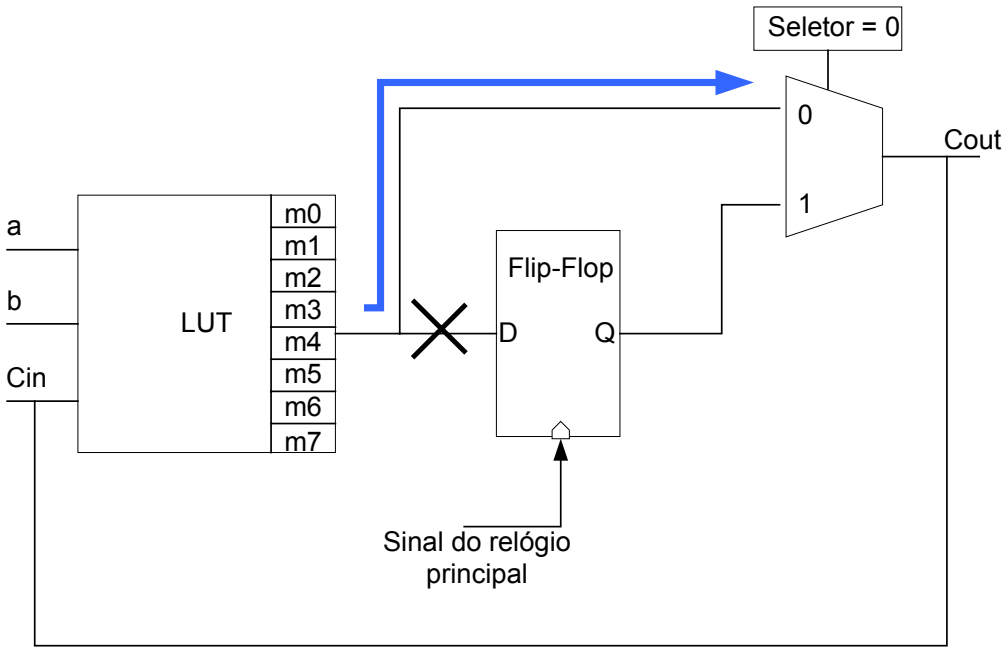



Figura 4.7: Elemento de programação para Célula Muller de duas entradas

## 4.4 Síntese voltada a Dispositivos Programáveis

Nesse trabalho foram feitas sínteses em CPLD e FPGA através do uso de ferramentas comerciais. No entanto, foi voltada objetivamente para plataforma FPGA. Foram utilizadas ferramentas Quartus II e Max+Plus II v.10.2 do fabricante ALTERA Corporation (ALTERA, 2004), ISE v. 7 do fabricante XILINX (XILINX, 2003) e Libero IDE v 5.2 do fabricante ACTEL (ACTEL, 2003) para a síntese dos circuitos. A linguagem utilizada para descrever os circuitos que foram sintetizados por essas ferramentas foi o VHDL. A seguir mostraremos o enfoque da linguagem VHDL para os circuitos assíncronos e como foi feito a descrição para os principais elementos utilizados nessa dissertação, como as células Muller, elementos M de N e registrador assíncrono.

### 4.4.1 Linguagem de programação VHDL

Para realizar a síntese dos circuitos através das ferramentas comerciais é necessária a descrição do hardware desejado através de uma linguagem própria, que a ferramenta entenda. Essas ferramentas entendem as linguagens Verilog ou VHDL para descrever os hardwares que deverão ser sintetizados.

Nesse trabalho, todos os circuitos foram descritos através da linguagem VHDL conforme o tipo de implementação escolhida. Foi necessário, contudo, uma maior preocupação no momento que esse código foi escrito com relação a detalhes essenciais que poderiam impossibilitar o funcionamento dos circuitos assíncrono. O código foi escrito observando os conceitos de codificação dual-rail para os dados, lógica monotônica e indicação forte dos sinais, evitando assim os possíveis hazards que poderiam ocorrer a partir de um projeto menos elaborado de um circuito síncrono.

### 4.4.2 Descrição de elementos assíncronos em VHDL

Esta seção descreve os códigos, escritos em VHDL, utilizados para as células Muller, elementos M de N e o registrador assíncrono. Os códigos apresentam características particulares para cada aplicação visto que são implementados por ferramentas que realizam a síntese de circuitos síncronos, como foi visto, a Look-up Table possui um registrador controlado pelo sinal do relógio principal que não é utilizado pelos circuitos assíncronos. Logo, a descrição desse código deverá ser feita de forma que substitua os possíveis arranjos causados pela presença do relógio principal por um projeto de arquitetura mais aprimorado.

#### *4.4.2.1 Célula Muller*

Os códigos em VHDL, utilizados na síntese da célula Muller de 2 entradas estão mostrados no Apêndice A. Foram implementados códigos VHDL comportamental (a ferramenta distribui os sinais de entrada de acordo com o comportamento que o circuito deverá ter) e a forma estrutural (o projetista é que determina quais as funções que a ferramenta deverá utilizar para realizar a síntese do circuito). É possível verificar nesses códigos a necessidade de realimentação do sinal Cout para uma das entradas. Nos códigos comportamental e estrutural esse sinal é definido como ctemp.

No Apêndice B é mostrado o código para célula Muller de 3 entradas. Não há muita diferença para o código mostrado no Apêndice A a não ser pela inclusão de mais uma entrada na verificação condicional para o código VHDL comportamental e de mais uma estrutura de portas AND e OR nas equações definidas no componente MODCC3.vhd.

Neste trabalho também foi feito uma modificação da célula Muller de 3 entradas. Essa nova célula foi construída a partir da composição de duas outras células Muller de duas entradas, como mostra a figura 4.8. Essa nova célula, que passou a ser chamada célula Muller 2x2 entradas é formada por duas outras células Muller onde a saída de uma célula Muller de 2 entradas é uma entrada da segunda célula Muller.

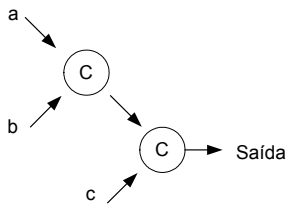



Figura 4.8: Esquema da célula Muller 2x2

Os códigos comportamental e estrutural que foram desenvolvidos para essa estrutura estão mostrados no Apêndice C. Eles foram feitos, utilizando os códigos já prontos das células Muller de 2 entradas, tanto o VHDL comportamental quanto o VHDL estrutural, através dos componentes CELULAC_COMPORTAMENTAL e CELULAC_ESTRUTURAL.

Nesse trabalho, além da variação da célula de três entradas, foram feitas variações para as de 4, 5 e de 6 entradas. A única restrição que foi mantida é que essas novas estruturas tivessem até dois níveis de células, ou seja, a partir da célula Muller original fossem construídas estruturas as quais todos os sinais de entrada passagem por, no máximo, duas células Muller. Essa restrição se faz necessária por causa do aumento do atraso que a composição de várias células Muller poderia provocar nas novas estruturas.

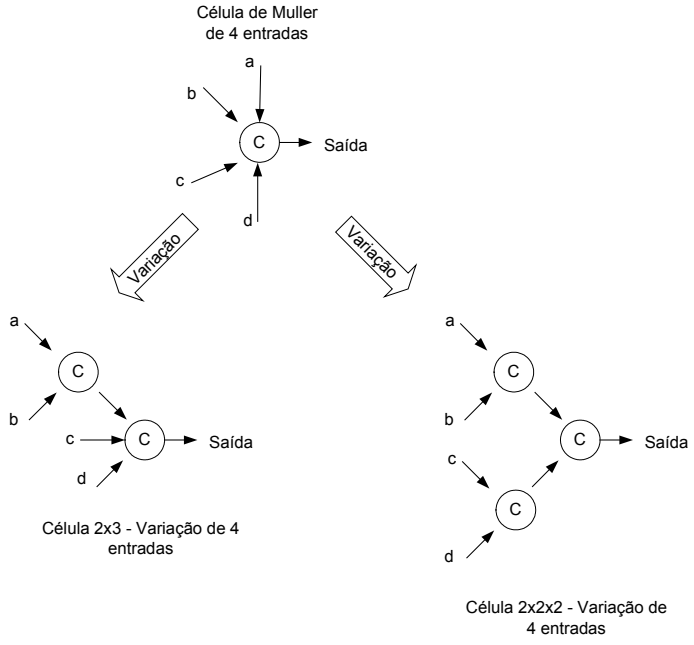



Figura 4.9: Variações da célula Muller de 4 entradas

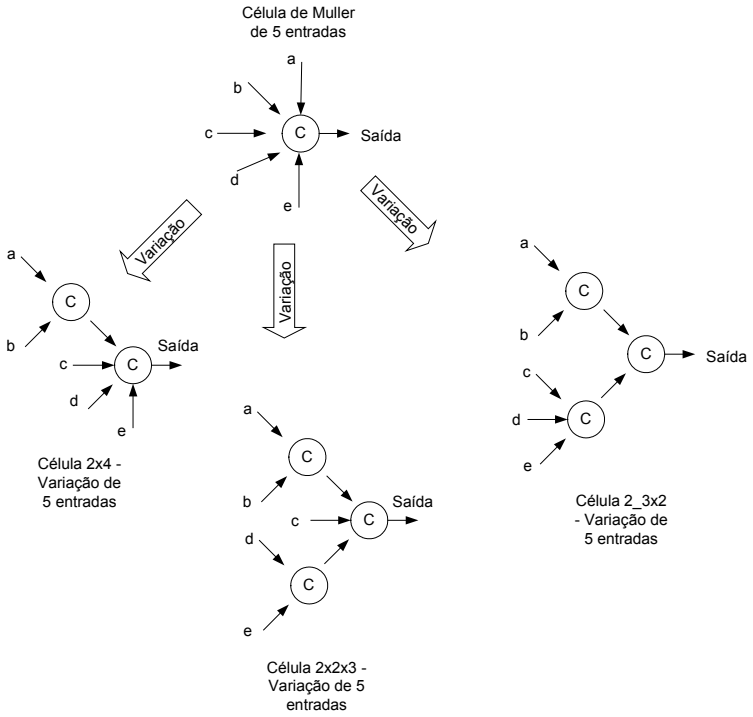



Figura 4.10: Variação da célula Muller de 5 entradas

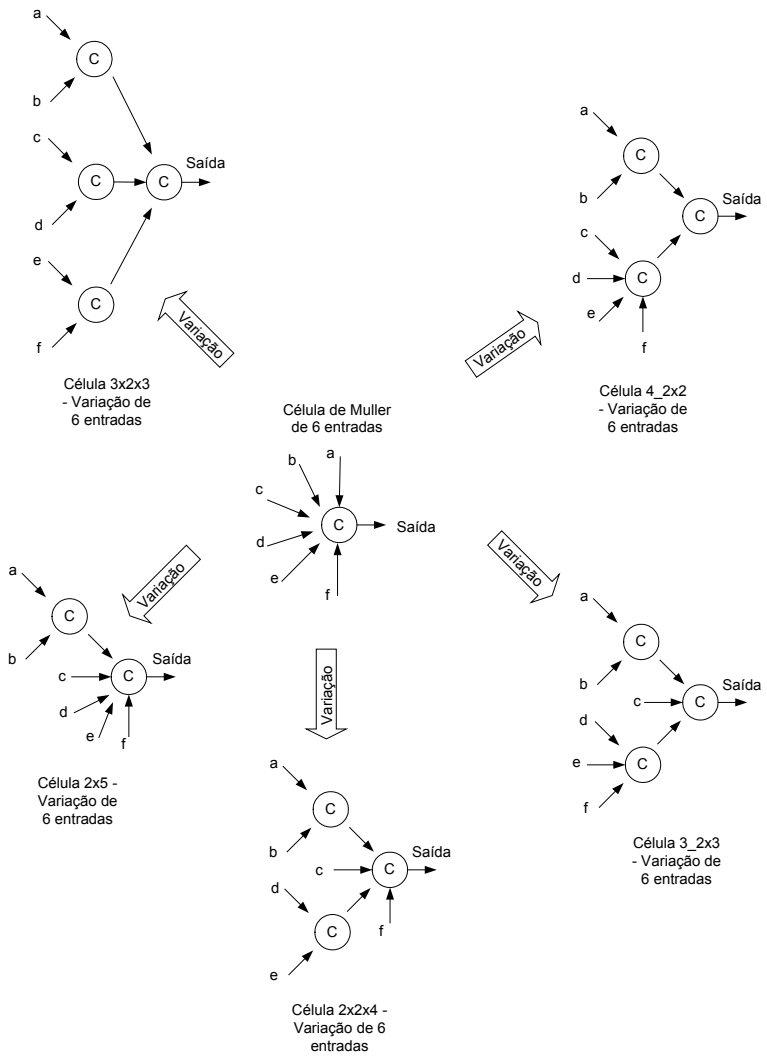



Figura 4.11: Variação da célula Muller de 6 entradas

Resultados interessantes foram obtidos da implementação dessas novas estruturas com relação às células Muller originais de 3, 4, 5 e 6 entradas, mostradas nas figuras 4.9, 4.10 e 4.11. Nas tabelas 4.3, 4.4 e 4.5 são apresentados os resultados para células lógicas e atraso nas ferramentas de síntese lógica dos fabricantes ALTERA, XILINX e ACTEL, respectivamente. Através dessas tabelas é possível verificar uma tendência comum de duas dessas ferramentas. Nas ferramentas dos fabricantes ALTERA e XILINX as estruturas feitas com código VHDL comportamental não tiveram vantagens com relação àquelas feitas com código VHDL estrutural, como a economia de células lógicas e até diminuição do atraso. Já na ferramenta ACTEL ocorreu justamente o contrário, houve uma maior economia de células lógica e diminuição do atraso quando foram implementados os códigos VHDL comportamentais em relação aos códigos VHDL estruturais. Uma explicação razoável para essa diferença de tendência entre as ferramentas é que por se tratar de diferentes fabricantes, eles possuem diferentes modos de tratar o código que o projetista descreve para ser implementado por eles.

Na tabela 4.3, para o código VHDL estrutural, existem algumas estruturas que utilizam menos LUTs que as células Muller originais. A célula Muller de 2x3 entradas utiliza 2 LUTs em comparação com as 3 LUTs utilizadas para a célula Muller de 4

entradas e ainda a célula Muller de 3_2x3 entradas utiliza apenas 3 LUTs em comparação com a sua célula Muller original de 6 entradas que utiliza 4 LUTs. Nessas estruturas citadas como exemplo houve diminuição do atraso verificado.

Tabela 4.3: Valores de Células Lógicas e Atrasos para Células Muller para Fabricante ALTERA – CPLD (Família MAX 7000AE) e FPGA (Família FLEX10KE)

| Células Muller | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Número de entradas da Célula Muller | Comportamental | | | | Estrutural | | | |
| | FPGA | | CPLD | | FPGA | | CPLD | |
| | Número de células (LUTs) | Atraso (ns) | Número de macrocélulas | Atraso (ns) | Número de células (LUTs) | Atraso (ns) | Número de macrocélulas | Atraso (ns) |
| 2 | 2 | 8,200 | 2 | 7,600 | 1 | 7,300 | 1 | 4,500 |
| 3 | 2 | 8,100 | 2 | 7,600 | 1 | 7,200 | 1 | 4,500 |
| 2X2 | 4 | 10,100 | 4 | 13,800 | 2 | 8,100 | 2 | 7,600 |
| 4 | 2 | 8,100 | 2 | 7,600 | 3 | 8,300 | 1 | 4,500 |
| 2x3 | 4 | 10,000 | 4 | 13,800 | 2 | 8,200 | 2 | 7,600 |
| 2x2x2 | 6 | 10,000 | 6 | 13,800 | 3 | 8,200 | 3 | 7,600 |
| 5 | 3 | 9,300 | 2 | 7,600 | 3 | 8,400 | 2 | 4,900 |
| 2x4 | 4 | 10,200 | 4 | 13,800 | 4 | 9,500 | 2 | 7,600 |
| 2x2x3 | 6 | 10,100 | 6 | 13,800 | 3 | 8,300 | 3 | 7,600 |
| 2_3x2 | 6 | 10,100 | 6 | 13,800 | 3 | 8,300 | 3 | 7,600 |
| 6 | 3 | 9,300 | 2 | 7,600 | 4 | 9,500 | 2 | 4,900 |
| 2x5 | 5 | 11,200 | 4 | 13,800 | 4 | 9,400 | 3 | 7,600 |
| 2x2x4 | 6 | 10,100 | 6 | 13,800 | 5 | 9,400 | 3 | 7,600 |
| 3x2x3 | 8 | 10,000 | 8 | 13,800 | 4 | 8,200 | 4 | 7,600 |
| 3_2x3 | 6 | 10,000 | 6 | 13,800 | 3 | 8,200 | 3 | 7,600 |
| 4_2x2 | 6 | 10,100 | 6 | 13,800 | 5 | 9,200 | 3 | 7,600 |

Na tabela 4.4 temos os valores das células lógicas e atraso para as células Muller extraídos através da ferramenta de síntese lógica do fabricante XILINX. Para algumas estruturas feitas através de VHDL estrutural foi possível avaliar um consumo menor de LUTs e diminuição do atraso. Podemos utilizar a célula Muller 3_2x3 entradas novamente como exemplo. Houve a diminuição de 5 LUTs para 3 LUTs em comparação com a sua célula Muller original de 6 entradas e ainda uma diminuição do atraso.

Tabela 4.4: Valores de Células Lógicas e Atrasos para Células Muller para Fabricante XILINX – CPLD (Família XC9500XV) e FPGA (Família SPARTAN2)

| Células Muller | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Número de entradas da Célula Muller | Comportamental | | | | Estrutural | | | |
| | FPGA | | CPLD | | FPGA | | CPLD | |
| | Número de células (LUTs) | Atraso (ns) | Número de macrocélulas | Atraso (ns) | Número de células (LUTs) | Atraso (ns) | Número de macrocélulas | Atraso (ns) |
| 2 | 1 | 8,128 | 1 | NA | 1 | 9,624 | 2 | 7,500 |
| 3 | 2 | 8,128 | 1 | NA | 1 | 9,624 | 2 | 7,500 |
| 2X2 | 2 | 8,128 | 2 | NA | 2 | 11,617 | 3 | 7,500 |
| 4 | 2 | 8,128 | 1 | NA | 2 | 10,329 | 2 | 7,500 |
| 2x3 | 3 | 8,128 | 2 | NA | 2 | 11,617 | 3 | 10,000 |
| 2x2x2 | 3 | 8,128 | 3 | NA | 3 | 11,617 | 4 | 10,000 |
| 5 | 2 | 8,128 | 1 | NA | 3 | 11,617 | 2 | 8,200 |
| 2x4 | 3 | 8,128 | 2 | NA | 3 | 12,272 | 3 | 10,000 |
| 2x2x3 | 4 | 8,128 | 3 | NA | 3 | 11,617 | 4 | 10,000 |
| 2_3x2 | 4 | 8,128 | 3 | NA | 3 | 11,617 | 4 | 10,000 |
| 6 | 3 | 8,128 | 1 | NA | 5 | 13,560 | 2 | 8,200 |
| 2x5 | 3 | 8,128 | 2 | NA | 4 | 9,989 | 3 | 10,700 |
| 2x2x4 | 4 | 8,128 | 3 | NA | 4 | 12,272 | 4 | 10,000 |
| 3x2x3 | 5 | 8,128 | 4 | NA | 4 | 11,617 | 5 | 10,000 |
| 3_2x3 | 5 | 8,128 | 3 | NA | 3 | 11,617 | 4 | 10,000 |
| 4_2x2 | 4 | 8,128 | 3 | NA | 4 | 12,322 | 4 | 10,000 |

(NA) valores não avaliados pela ferramenta.

Na tabela 4.5 podemos verificar a tendência do código VHDL comportamental ter resultados melhores que o código VHDL estrutural. Pode ser dado o exemplo da célula Muller 2x2x3 entradas que consumiu 5 LUTs a menos que a célula original de 5 entradas ou ainda para a célula Muller de 3x2x3 entradas que consumiu 6 LUTs a menos com relação à célula original de 6 entradas.

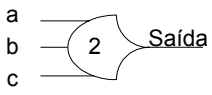
Tabela 4.5: Valores de Células Lógicas e Atrasos para Células Muller para Fabricante ACTEL – FPGA Antifusível (Família ACT1) e FPGA SRAM (Família 500K)

| Células Muller | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Número de entradas da Célula Muller | Comportamental | | | | Estrutural | | | |
| | FPGA Antifusível | | FPGA SRAM | | FPGA Antifusível | | FPGA SRAM | |
| | Número de células (LUTs) | Atraso (ns) | Número de células (LUTs) | Atraso (ns) | Número de células (LUTs) | Atraso (ns) | Número de células (LUTs) | Atraso (ns) |
| 2 | 2 | 7,465 | 2 | 7,462 | 5 | NA | 2 | NA |
| 3 | 6 | 7,465 | 4 | 7,042 | 8 | NA | 4 | NA |
| 2X2 | 4 | 7,465 | 4 | 7,462 | 10 | NA | 4 | NA |
| 4 | 9 | 7,465 | 4 | 7,462 | 12 | NA | 5 | NA |
| 2x3 | 8 | 7,465 | 6 | 7,462 | 13 | NA | 6 | NA |
| 2x2x2 | 6 | 7,465 | 6 | 7,462 | 14 | NA | 6 | NA |
| 5 | 15 | 7,465 | 6 | 7,042 | 15 | NA | 7 | NA |
| 2x4 | 11 | 7,465 | 6 | 7,462 | 17 | NA | 7 | NA |
| 2x2x3 | 10 | 7,465 | 8 | 7,462 | 18 | NA | 8 | NA |
| 2_3x2 | 10 | 7,465 | 8 | 7,042 | 18 | NA | 8 | NA |
| 6 | 18 | 7,465 | 6 | 7,462 | 18 | NA | 8 | NA |
| 2x5 | 17 | 7,465 | 8 | 7,462 | 20 | NA | 9 | NA |
| 2x2x4 | 13 | 7,465 | 8 | 7,462 | 21 | NA | 9 | NA |
| 3x2x3 | 12 | 7,465 | 10 | 7,462 | 23 | NA | 10 | NA |
| 3_2x3 | 14 | 7,465 | 10 | 7,042 | 21 | NA | 10 | NA |
| 4_2x2 | 13 | 7,465 | 8 | 7,462 | 21 | NA | 9 | NA |

(NA) valores não avaliados pela ferramenta.

#### *4.4.2.2 Célula M de N*

O código utilizado para descrever o elemento 2 de 3 está mostrado na figura 4.12. O comportamento do elemento 2 de 3 é passado para a ferramenta. Como foi explicado, esse elemento terá uma saída 1 quando duas entradas estiverem em 1 e terá uma saída 0 quando todas as três entradas forem 0. Para qualquer outra variação de entradas, a saída armazena o valor anterior.



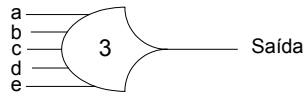

(a) Símbolo para célula 2 de 3

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY ncl_3_2 IS
PORT ( e3, e2, e1       : IN std_logic;
       s                : OUT std_logic);
END ncl_3_2;

ARCHITECTURE comportamental OF ncl_3_2 IS
SIGNAL s_temp: std_logic;
SIGNAL aux: std_logic_vector (2 downto 0);

BEGIN
aux <= e1 & e2 & e3;

s_temp <=   '0' when aux = "000" else
            '1' when aux = "110" else
            '1' when aux = "101" else
            '1' when aux = "011" else
            '1' when aux = "111" else
            s_temp;
s <= s_temp;
END comportamental;
```

(b) Código em VHDL utilizado para célula 2 de 3 comportamental

Figura 4.12: Símbolo e código VHDL comportamental do elemento 2 de 3

Na figura 4.13 é mostrado o código VHDL comportamental do elemento 3 de 5. Como foi dito, o projetista descreve o comportamento que essa estrutura deverá ter para que a ferramenta realize a síntese lógica. Nesse elemento, teremos a saída em 1 quando três entradas forem 1 e a saída 0 quando todas as cinco entradas forem 0. Para qualquer outra combinação, a saída deverá armazenar o valor anterior.




(a) Símbolo para célula 3 de 5

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY ncl_5_3 IS
PORT ( e5, e4, e3, e2, e1     : IN std_logic;
       s                      : OUT std_logic);
END ncl_5_3;

ARCHITECTURE comportamental OF ncl_5_3 IS
SIGNAL s_temp: std_logic;
SIGNAL aux: std_logic_vector (4 downto 0);
BEGIN
aux <= e1 & e2 & e3 & e4 & e5;

s_temp <=   '0' when aux = "00000" else
            '1' when aux = "00111" else
            '1' when aux = "01110" else
            '1' when aux = "01101" else
            '1' when aux = "01011" else
            '1' when aux = "10011" else
            '1' when aux = "10101" else
            '1' when aux = "10110" else
            '1' when aux = "11001" else
            '1' when aux = "11010" else
            '1' when aux = "11100" else
            '1' when aux = "11111" else
            s_temp;
s <= s_temp;
END comportamental;
```

(b) Código em VHDL utilizado para célula 3 de 5 comportamental

Figura 4.13: Símbolo e código VHDL comportamental do elemento 3 de 5

#### *4.4.2.3 Registradores Assíncronos*

A estrutura utilizada para modelar os códigos VHDL comportamental e estrutural do registrador assíncrono está na figura 4.14 e os códigos utilizados para implementar os circuitos estão no Apêndice D. O circuito do registrador é formado por cinco inversores e quatro portas lógicas OR. Ele recebe os sinais de Habilita, Et e Ef e produz as saídas St e Sf Os sinais Et, Ef assim como St e Sf representam os sinais de entrada e saída, respectivamente, codificados em dual-rail, conforme foi visto no capítulo 3. No Apêndice D são mostrados os códigos VHDL para o circuito. O código VHDL comportamental pode ser exemplificado pelo funcionamento do registrador assíncrono. O funcionamento do circuito registrador é feito através da verificação do sinal de Habilita. Se este sinal for zero, as saídas St e Sf serão zero. Caso contrário, verifica se as entradas Et e Ef são zero. Se essas entradas forem zero, repete nas saídas St e Sf as últimas entradas válidas, caso contrário, atualiza os valores de St e Sf com novas entradas. Para o código VHDL estrutural o projetista deve apenas montar a estrutura que foi planejada, nesse código teremos os sinais de Habilita, Et e Ef sendo organizados de acordo com a estrutura lógica mostrada na figura 3.4 do registrador assíncrono.

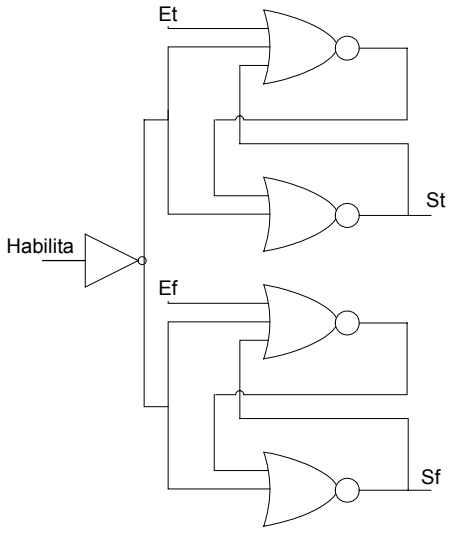



Figura 4.14: Esquema do registrador VHDL

#### *4.4.2.4 Circuito comportamental com indicação forte*

No trabalho foi descrito um código VHDL de um somador com indicação forte. No Apêndice E é mostrado o código comportamental de um somador. O projetista transmite as condições que deseja que seu circuito seja feito e a ferramenta determina como será essa lógica. O circuito descrito através do código VHDL possui indicação forte pois na seqüência de ações que são transmitidas à ferramenta existe a verificação se todas as entradas já estão disponíveis para o circuito do somador e também verifica se todas as entradas estão zeradas, ou seja, só irá gerar saídas válidas quando as entradas tiverem válidas e só irá gerar saídas zeradas quando todas as entradas estiverem zeradas.

## 4.5 Hazards e Dispositivos Programáveis

Esta seção explica como dispositivos lógicos programáveis, concebidos para o projeto de circuitos síncronos, podem ser usados no projeto de circuitos livres de hazards necessários para a implementação de circuitos assíncronos. Os mecanismos variam conforme o tipo de lógica e são descritos abaixo.

### 4.5.1 DIMS e hazards

No uso de lógica do tipo DIMS, tanto a propriedade de indicação forte quanto a propriedade de circuito livre de hazard se baseiam no uso extensivo de células Muller. Uma vez que a célula Muller pode ser implementada em VHDL, para síntese em plataforma FPGA, então as propriedades da lógica DIMS estão garantidas através do uso de instâncias de células Muller para compor os circuitos.

### 4.5.2 NCL e hazards

De modo similar à lógica do tipo DIMS, na lógica NCL tanto a propriedade de indicação forte quanto a propriedade de circuito livre de hazards se baseiam no uso extensivo de células M de N. Uma vez que todas as variações da célula de M de N podem ser implementadas em VHDL, para síntese em plataforma FPGA, então as propriedades da lógica DIMS estão garantidas através do uso de instâncias de células M de N para compor os circuitos.

### 4.5.3 Derivação a partir de circuito combinacional síncrono e hazards

A derivação de um circuito assíncrono em dual rail a partir do circuito combinacional síncrono em single rail é feita usando lógica monotônica positiva. Isto garante que transições a partir do valor nulo (todos os bits iguais a 0), usado como espaçador, serão livres de hazards, conforme discutido na seção 3.3.3. As células Muller podem ser usadas nas saídas do circuito, de modo a garantir o princípio da indicação forte, conforme discutido nas seções 3.3.5 e 3.5.3.

O uso de lógica monotônica positiva garante transições livres de hazard de modo razoavelmente independente da implementação. No caso de uma LUT com FPGAs, o que é implementado é uma tabela verdade com a função (monotônica positiva) desejada. Esta implementação é livre de hazards, pois a ordem em que as linhas da memória (tabela verdade) são acessadas garante que a implementação seja livre de hazards. Segundo a propriedade das funções monotônicas, uma vez que uma linha da memória com a saída igual ao valor lógico um for atingida, as próximas transições levarão também a linhas com o valor lógico um, levando a uma implementação monotônica e livre de hazard.

### 4.5.4 Descrição comportamental com indicação forte e hazards

Neste tipo de circuito, não há garantias de que a saída seja livre de hazards. Porém a indicação forte é garantida pela descrição comportamental. Porém, como a indicação forte é garantida através da memorização das saídas enquanto as entradas estiverem mudando, as saídas ficam estáveis durante a mudança dos sinais. Esta característica acaba garantindo na prática um comportamento livre de hazards.

## 4.6 Sumário

Nesse capítulo foi mostrada uma rápida introdução sobre PLDs e as principais arquiteturas atualmente utilizadas para a sua síntese lógica. Um enfoque maior foi feito

para a plataforma FPGA através de seus elementos de programação, as LUTs, utilizadas no processo de síntese lógica. Foi explicado o conceito de volatilidade e granularidade de elementos de programação assim como a característica que outros elementos possuem de serem reprogramados. Com essa pequena introdução sobre a plataforma FPGA e características de elementos de programação, direcionamos os trabalhos dessa dissertação para a síntese lógica voltada para FPGA, através da utilização de ferramentas de síntese, utilizadas comercialmente, para códigos descritos em VHDL. Os códigos em VHDL utilizados para os elementos de base como células Muller, elementos M de N e registrador assíncrono são mostrados na seqüência. Essa seção encerra com uma descrição de como os conceitos de lógica hazards e os dispositivos programáveis estão integrados e interligados nesse trabalho.

# 5 COMPARAÇÃO EM ÁREA ENTRE CIRCUITOS SÍNCRONOS E ASSÍNCRONOS

## 5.1 Resumo

Este capítulo descreve quantitativamente as diferenças entre circuitos síncronos e assíncronos. As diferenças são feitas quanto ao comportamento, ignorando-se detalhes de implementação.

## 5.2 Introdução

A avaliação dos circuitos assíncronos frente aos circuitos síncronos não é uma tarefa tão simples. Essa seção tem por objetivo avaliar separadamente as diferentes situações que podem ser encontradas nos circuitos síncronos e assíncronos e através delas, extrair os atrasos de *throughput* e latência desses circuitos e compara-los. Posteriormente, se faz uma comparação dos circuitos assíncronos com diferentes registradores e se verifica as conseqüências que isso pode trazer com relação à performance desses circuitos.

## 5.3 Comparação da parte Combinacional

Nesta comparação tentamos estabelecer o custo de implementação de diferentes partes combinacionais quando adaptadas ao comportamento assíncrono.

### 5.3.1 Full Adder

Esta seção descreve o comportamento de um full-adder.

#### *5.3.1.1 Descrição*

Os códigos dos somadores foram descritos de acordo com os estilos mostrados no capítulo 3: Do tipo DIMS, do tipo NCL, através de derivação de circuito combinacional síncrono e do tipo Comportamental com indicação forte do sinal. Ainda foram simulados os códigos de somadores do tipo comportamental e estrutural, como mostra o Apêndice F. Através desses valores é possível comparar o crescimento do consumo de elementos de programação que esses estilos de projeto tiveram.

### *5.3.1.2 Resultados*

As tabelas 5.1, 5.2 e 5.3 mostram os resultados para os fabricantes ALTERA, XILINX e ACTEL, respectivamente. Com relação aos valores comportamental e estrutural dos somadores dos tipos padrão, vistos no Apêndice F. Para o fabricante ALTERA, os códigos comportamental e estrutural tiveram um custo de 2LUTs e atraso 7,0ns para FPGA e de 2 macrocélulas e 4,5ns para CPLD. Para o fabricante XILINX, os códigos comportamental e estrutural tiveram um custo de 2LUTs e atraso 9,62ns para FPGA e de 2 macrocélulas e 5,0ns para CPLD. Para fabricante ACTEL, os códigos comportamental e estrutural tiveram um custo de 7LUTs e atraso 40,0ns (o estrutural) e 28,5ns (o comportamental) para FPGA antifusível e de 4 células com atraso de 4,2ns (o estrutural) e 4,15 (o comportamental) para FPGA SRAM

Tabela 5.1: Valores de Células Lógicas e Atrasos para full adder para Fabricante ALTERA – CPLD (Família MAX 7000AE) e FPGA (Família FLEX10KE)

| Full Adder | | | | |
|---|---|---|---|---|
| Tipo de Anel | FPGA | | CPLD | |
| | Número de células (LUTs) | Atraso (ns) | Número de macrocélulas | Atraso (ns) |
| Comportamental com indicação forte | 17 | 11,600 | 10 | 8,000 |
| DIMS | 12 | 9,300 | 12 | 7,600 |
| NCL | 10 | 10,800 | 8 | 13,800 |
| Derivação de lógica | 17 | 11,900 | 8 | 4,900 |
| Full Adder padrão do tipo estrutural | 2 | 7,0 | 2 | 4,5 |
| Full Adder padrão do tipo comportamental | 2 | 7,0 | 2 | 4,5 |

Tabela 5.2: Valores de Células Lógicas e Atrasos para full adder para Fabricante XILINX – CPLD (Família XC9500XV) e FPGA (Família SPARTAN2)

| Full Adder | | | | |
|---|---|---|---|---|
| Tipo de Anel | FPGA | | CPLD | |
| | Número de células (LUTs) | Atraso (ns) | Número de macrocélulas | Atraso (ns) |
| Comportamental com indicação forte | 13 | Não encontrado | 4 | 6,000 |
| DIMS | 12 | 12,017 | 12 | 7,500 |
| NCL | 8 | Não encontrado | 8 | - |
| Derivação de lógica | 13 | 14,130 | 8 | 7,500 |
| Full Adder padrão do tipo estrutural | 2 | 9,624 | 2 | 5,0 |
| Full Adder padrão do tipo comportamental | 2 | 9,624 | 2 | 5,0 |

Tabela 5.3: Valores de Células Lógicas e Atrasos para full adder para Fabricante ACTEL – FPGA Antifusível (Família ACT1) e FPGA SRAM (Família 500K)

| Full Adder | | | | |
|---|---|---|---|---|
| Tipo de Anel | FPGA Antifusível | | FPGA SRAM | |
| | Número de células (LUTs) | Atraso (ns) | Número de células (área) | Atraso (ns) |
| Comportamental com indicação forte | 42 | 61,100 | 27 | 1,602 |
| DIMS | 82 | Não encontrado | 34 | Não encontrado |
| NCL | 64 | 82,500 | 34 | Não encontrado |
| Derivação de lógica | 50 | Não encontrado | 20 | Não encontrado |
| Full Adder padrão do tipo estrutural | 7 | 40,0 | 4 | 4,228 |
| Full Adder padrão do tipo comportamental | 7 | 28,5 | 4 | 4,154 |

### 5.3.2 RCA

Esta seção descreve os custos de implementação de um ripple carry adder.

#### *5.3.2.1 Descrição*

Ripple Carry Adder, ou simplesmente RCAs. São circuitos que se utilizam da propagação dos carry out´s dos somadores para implementar circuitos de n bits. A implementação desses circuitos foi feita utilizando a saída do carry out antecessor como entrada do carry in sucessor. Com isso, foi possível verificar o fluxo de dados e os atrasos desses circuitos uma vez que o sinal de carry deve percorrer todos os n somadores para chegar até o carry out do somador mais significativo. Nesses resultados foram feitas simulações com RCAs de 4, 8, 16 e 32 bits.

#### *5.3.2.2 Resultados*

As tabelas 5.4, 5.5 e 5.6 mostram os resultados para os fabricantes ALTERA, XILINX e ACTEL, respectivamente. Na tabela 5.4 da ALTERA, foi possível comparar com os resultados encontrados dos somadores estrutural e comportamental os demais estilos de somadores apresentados no capítulo 3. Existe um grande aumento no consumo de células lógicas por parte desses somadores. O estilo que teve um melhor custo comparativo entre os quatro tipos frente aos somadores full adder padrão foi o NCL para FPGAs e derivação de lógica para CPLDs e o que teve pior custo foi o derivação de lógica seguido do comportamental com indicação forte do sinal para os FPGAs e DIMS para CPLDs. Na tabela 5.5 para fabricante XILINX, o melhor custo comparativo foi para o tipo NCL para FPGAs e derivação de lógica para os CPLDs e o pior custo foi para derivação de lógica para FPGAs e DIMS para CPLDs. Na tabela 5.6 do fabricante ACTEL, temos o melhor custo para derivação de lógica tanto para os antifusíveis quanto os SRAM e o pior custo para os do tipo NCL tanto para os antifusíveis quanto para os SRAM.

Tabela 5.4: Valores de Células Lógicas para o RCA para Fabricante ALTERA – CPLD (Família MAX 7000AE) e FPGA (Família FLEX10KE)

| RCA | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de somadores | 4 bits | | 8bits | | 16bits | | 32bits | |
| | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD |
| | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) |
| Comporta mental com indicação forte do sinal | 62 | 41 | 122 | 85 | 241 | 178 | 485 | 354 |
| Por DIMS | 48 | 73 | 100 | 134 | 205 | 254 | 416 | - |
| Por NCL | 45 | 36 | 90 | 76 | 182 | 161 | 369 | 384 |
| Derivação de lógica | 63 | 32 | 132 | 71 | 273 | 161 | 552 | 315 |
| Somador Full Adder comporta mental | 8 | 11 | 16 | 20 | 32 | 40 | 64 | 83 |
| Somador Full Adder Estrutural | 11 | 10 | 23 | 30 | 47 | 53 | 95 | 108 |

(*) *macrocell* representa a quantidade de macrocélulas em CPLDs

(-) valor não informado pela ferramenta

Tabela 5.5: Valores de Células Lógicas para o RCA para Fabricante XILINX – CPLD (Família XC9500XV) e FPGA (Família SPARTAN2)

| RCA | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de somadores | 4 bits | | 8bits | | 16bits | | 32bits | |
| | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD |
| | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell | Número de células (LUTs) | Número de macrocell (*) |
| Comporta mental com indicação forte do sinal | 58 | 31 | 110 | 58 | 222 | 88 | 446 | - |
| Por DIMS | 57 | 42 | 106 | 82 | 210 | 163 | 418 | - |
| Por NCL | 41 | 33 | 75 | 67 | 151 | 131 | 303 | - |
| Derivação de lógica | 58 | 26 | 111 | 53 | 227 | 100 | 459 | - |
| Somador Full Adder comporta mental | 9 | 8 | 17 | 20 | 33 | 45 | 65 | 92 |
| Somador Full Adder Estrutural | 9 | 8 | 17 | 20 | 33 | 45 | 65 | 90 |

(*) *macrocell* representa a quantidade de macrocélulas em CPLDs

(-) valor não informado pela ferramenta

Tabela 5.6: Valores de Células Lógicas para o RCA para Fabricante ACTEL – FPGA Antifusível (Família AXCELERATOR) e FPGA SRAM (Família 500K)

| RCA | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de somadores | 4 bits | | 8bits | | 16bits | | 32bits | |
| | FPGA/AF | FPGA/SR | FPGA/AF | FPGA/SR | FPGA/AF | FPGA/SR | FPGA/AF | FPGA/SR |
| | Número de células (LUTs) | Número de células (A) | Número de células (LUTs) | Número de células (A) | Número de células (LUTs) | Número de células (A) | Número de células (LUTs) | Número de células (A) |
| Comportamental com indicação forte do sinal | 80 | 110 | 170 | 231 | 355 | 474 | 751 | 959 |
| Por DIMS | 71 | 140 | 151 | 289 | 362 | 588 | 774 | 1185 |
| Por NCL | 147 | 145 | 251 | 296 | 653 | 600 | 1229 | 1210 |
| Derivação de lógica | 64 | 86 | 135 | 183 | 285 | 375 | 582 | 756 |
| Somador Full Adder comportamental | 17 | 20 | 65 | 74 | 40 | 192 | 108 | 407 |
| Somador Full Adder Estrutural | 16 | 16 | 47 | 32 | 117 | 64 | 274 | 128 |

(A): A ferramenta informa os resultados através da descrição *Área*

## 5.4 Comparação da parte seqüencial: registradores

Esta seção descreve os custos de implementação de registradores assíncronos, quando descritos em VHDL para implementação em CPLDs e FPGAs.

### 5.4.1 Descrição

A descrição do registrador assíncrono foi feita no capítulo 3 desse trabalho. Além desse circuito foram simulados códigos em VHDL do Flip-Flop e Latch que a própria ferramenta possuía na sua biblioteca. Com isso foi possível simular em cada fabricante esses códigos e comparar os resultados com relação ao consumo de elementos de programação.

### 5.4.2 Resultados

As tabelas 5.7, 5.8 e 5.9 mostram os resultados para os fabricantes ALTERA, XILINX e ACTEL, respectivamente. Na tabela 5.7 da ALTERA, foi possível verificar que tanto o registrador com código comportamental quanto o de código estrutural apresentaram regularidade com relação ao crescimento do número de células lógicas quando se aumentava os bits de entrada. Houve vantagem com relação aos do tipo estrutural com relação à arquitetura FPGA, com menor custo de células lógicas em relação aos códigos comportamentais. Também foi feita comparação com o latch padrão e flip flip obtidos da biblioteca da ferramenta da ALTERA. Com relação a esses circuitos foi possível verificar o aumento de custo com relação aos registradores assíncronos.

Tabela 5.7: Valores de Células Lógicas e Atrasos para registrador assíncrono para Fabricante ALTERA – CPLD (Família MAX 7000AE) e FPGA (Família FLEX10KE)

| Registrador assíncrono | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Número de bits de entrada | Comportamental | | | | Estrutural | | | |
| | FPGA | | CPLD | | FPGA | | CPLD | |
| | Número de células (LUTs) | Atraso (ns) | Número de macrocélulas | Atraso (ns) | Número de células (LUTs) | Atraso (ns) | Número de macrocélulas | Atraso (ns) |
| 1bit | 3 | 8,100 | 2 | 12,000 | 2 | 7,000 | 2 | 12,000 |
| 4bits | 12 | 12,000 | 8 | 12,100 | 8 | 10,900 | 8 | 12,100 |
| 8bits | 24 | 12,400 | 16 | 12,200 | 16 | 10,800 | 16 | 12,200 |
| 16bits | 48 | 12,800 | 34 | 21,800 | 32 | 11,200 | 32 | 12,500 |
| 32bits | 96 | 12,500 | 64 | 17,300 | 64 | 11,000 | 64 | 12,900 |

No mesmo fabricante ALTERA o Latch padrão da ferramenta teve um custo de 1 LUT com atraso médio de 7,3ns para FPGAs e custo de 1 macrocélula com atraso de 12,0ns para CPLDs. Já o Flip Flop da ferramenta teve um custo de 1 LUT com atraso de 6,3ns para FPGAs e custo de 1 macrocélula com atraso de 9,1ns para CPLDs.

Na tabela 5.8 também verificamos certa regularidade para a ferramenta do fabricante XILINX. Nessa ferramenta, o registrador de 1 bit obteve o custo de 1 LUT na versão comportamental do código VHDL, equivalente até aos custos do Latch padrão e flip flop obtidos nas bibliotecas e sintetizados nessa ferramenta.

Tabela 5.8: Valores de Células Lógicas e Atrasos para registrador assíncrono para Fabricante XILINX – CPLD (Família XC9500XV) e FPGA (Família SPARTAN2)

| Registrador assíncrono | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Número de bits de entrada | Comportamental | | | | Estrutural | | | |
| | FPGA | | CPLD | | FPGA | | CPLD | |
| | Número de células (LUTs) | Atraso (ns) | Número de macrocélulas | Atraso (ns) | Número de células (LUTs) | Atraso (ns) | Número de macrocélulas | Atraso (ns) |
| 1bit | 1 | - | 2 | 6,000 | 4 | 10,223 | 2 | 6,000 |
| 4bits | 4 | - | 8 | 6,000 | 16 | 11,303 | 8 | 6,000 |
| 8bits | 8 | - | 16 | 6,000 | 32 | 12,203 | 16 | 6,000 |
| 16bits | 16 | - | 32 | 6,000 | 64 | 13,643 | 32 | 6,000 |
| 32bits | 32 | - | 64 | 6,000 | 128 | 19,097 | 64 | 6,000 |

(-) Não foi determinado pela ferramenta

No mesmo fabricante XILINX o Latch padrão da ferramenta teve um custo de 1 LUT com atraso não determinado para FPGAs e custo de 1 macrocélula com atraso não determinado para CPLDs. Já o Flip Flop da ferramenta teve um custo de 1 LUT com atraso não determinado para FPGAs e custo de 1 macrocélula com atraso de 5,2ns para CPLDs.

Para o fabricante ACTEL, temos a síntese com elementos de programação do tipo antifusível e SRAM. Houve um crescimento aproximadamente regular nos FPGAs antifusíveis e SRAM.

Tabela 5.9: Valores de Células Lógicas e Atrasos para registrador assíncrono para Fabricante ACTEL – FPGA Antifusível (Família ACT1) e FPGA SRAM (Família 500K)

| Registrador assíncrono | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Número de bits de entrada | Comportamental | | | | Estrutural | | | |
| | FPGA antifusível | | FPGA SRAM | | FPGA antifusível | | FPGA SRAM | |
| | Número de células (LUTs) | Atraso (ns) | Número de células (área) | Atraso (ns) | Número de células (LUTs) | Atraso (ns) | Número de células (área) | Atraso (ns) |
| 1bit | 10 | 9,840 | 10 | NA | 2 | NA | 2 | NA |
| 4bits | 42 | 14,840 | 41 | NA | 8 | NA | 8 | NA |
| 8bits | 85 | 17,040 | 82 | NA | 17 | NA | 17 | NA |
| 16bits | 170 | 18,240 | 166 | NA | 36 | NA | 34 | NA |
| 32bits | 341 | 18,240 | 331 | NA | 70 | NA | 70 | NA |

NA = Não fornecido pela ferramenta

No mesmo fabricante ACTEL o Latch padrão da ferramenta teve um custo de 1 LUT com atraso médio de 4,47ns para FPGAs antifusivel e custo de 1 célula de área com atraso médio de 2,37ns para FPGAs SRAM. Já o Flip Flop da ferramenta teve um custo de 1 LUT com atraso médio de 4,47ns para FPGAs antifusivel e custo de 1 célula de área com atraso de 2,38ns para FPGAs SRAM.

## 5.5 Comparação da parte seqüencial: anel completo

Esta seção mostra uma análise do custo de implementação de circuitos síncronos e assíncronos, quando implementado em CPLDs ou em FPGAs. A avaliação é feita de modo a se medir o efeito da introdução de circuitos combinacionais e a divisão em vários estágios.

Esta análise foi feita através de um método empírico para avaliar o pior atraso e atrasos dos registradores nos circuitos síncronos e avaliar o atraso médio e atrasos das partes operacionais nos circuitos assíncronos.

### 5.5.1 Avaliação de circuitos síncronos

Nessa seção iremos avaliar a influência que os circuitos combinacionais possuem nos circuitos síncronos. Os circuitos combinacionais envolvidos serão CC1, CC2 e CC3. Cada um deles possui um atraso máximo e um atraso médio para realizar o cálculo de suas entradas. A nomenclatura utilizada para representar esses atrasos será: ACALC_M para o atraso máximo que o circuito combinacional terá ao realizar o cálculo dos valores de entrada e ACALC_m o correspondente atraso médio para realização de um cálculo desse circuito. AREG será o atraso que o registrador terá ao armazenar e disponibilizar os valores armazenados em suas saídas.

#### *5.5.1.1 Circuitos síncronos com um estágio*

O circuito com apenas um circuito combinacional, conforme a figura 5.1, é o anel síncrono formado por apenas um registrador controlado pelo sinal do relógio. O atraso para que um sinal de informação saia da posição inicial, chegue em P1 e atinja P2 será o atraso de latência do circuito. O valor representado ao realizar um cálculo e ser gravado e disponibilizado na saída do registrador, será o atraso de *throughput* do circuito. Logo, nesse circuito o atraso de latência será igual ao atraso de *throughput*, pois ambos são atrasos para que a informação saia da posição inicial e chegue a P2. Assim, o atraso de latência e *throughput* será ACALC_M+AREG.

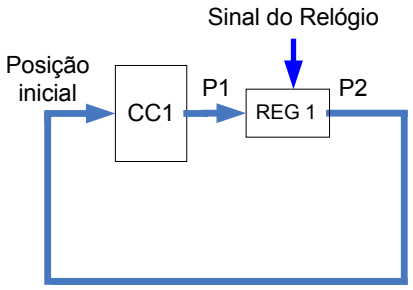



Figura 5.1: Anel síncrono com 1 circuito combinacional

### *5.5.1.2 Circuitos síncronos com dois estágios combinacionais*

O segundo caso é baseado num anel com dois circuitos combinacionais, chamados CC2, mostrados na figura 5.2. Esses dois circuitos juntos realizam o cálculo de CC1, ou seja, a função lógica de CC1 é decomposta em duas outras funções mais simples, iguais a CC2. Conforme a figura 5.2 o anel síncrono é formado por dois registradores controlados pelo sinal do relógio. O atraso para que um sinal de informação saia da posição inicial e atinja o ponto P4 será a soma dos atrasos máximos nos dois circuitos combinacionais CC2 e ainda os atrasos nos registradores. Já o atraso que o circuito possui para que a informação saia da posição inicial e chegue a P2 será o atraso de *throughput*. Logo, o atraso de latência será o dobro do atraso de *throughput*. O atraso de latência será 2ACALC_M+2AREG e o atraso de *throughput* será ACALC_M+AREG.

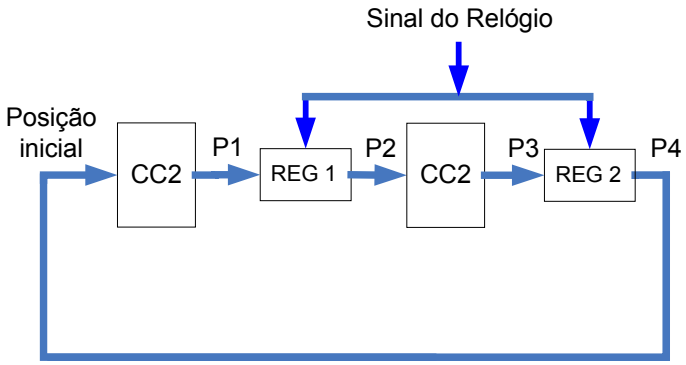



Figura 5.2: Anel síncrono com 2 circuitos combinacionais

### *5.5.1.3 Circuitos síncronos com três estágios combinacionais*

Conforme a figura 5.3 o anel síncrono é formado por 3 registradores controlados pelo sinal do relógio. O atraso para que um sinal de informação saia da posição inicial e atinja o ponto P6 será a soma dos atrasos máximos nos três circuitos combinacionais CC3 com os atrasos dos registradores. Logo, o atraso de latência desse circuito será o triplo do atraso de *throughput*. O atraso de latência será 3ACALC_M+3AREG e o atraso de *throughput* será ACALC_M+AREG.

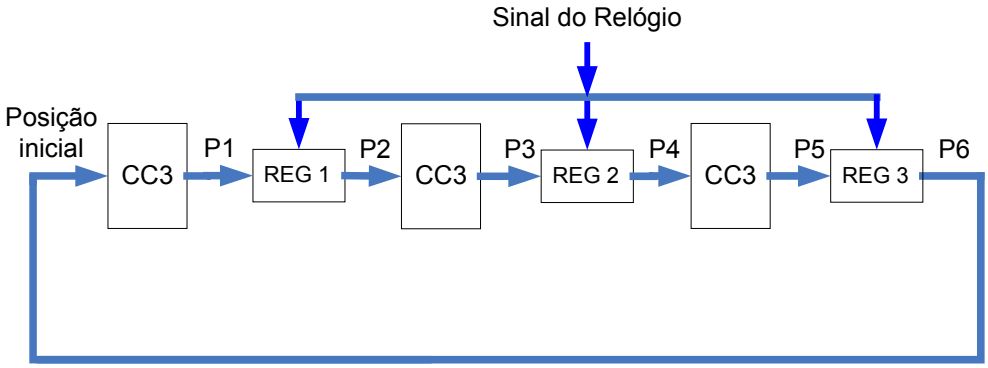



Figura 5.3: Anel síncrono com 3 circuitos combinacionais

### 5.5.2 Avaliação de circuitos assíncronos

Já nos circuitos assíncrono não ocorre interferência do sinal do relógio. Os atrasos nos circuitos combinacionais serão sempre os atrasos médios visto que uma vez que o cálculo esteja pronto, o registrador que o armazena não tem que esperar que o sinal do relógio o faça armazenar o resultado (WILLIAMS, 1994).

A seguir serão mostrados exemplos similares aos utilizados pelos anéis síncronos. Nesses circuitos teremos o número mínimo de três registradores pelos motivos apresentados na seção 2.4.3. Iremos representar além dos atrasos dos circuitos combinacionais e registradores os atrasos relativos aos circuitos de detecção de fim de cálculo e os circuitos de controle. É interessante notar que além desses atrasos, iremos considerar o atraso que o registrador assíncrono utiliza ao zerar as suas saídas.

### 5.5.3 Circuitos assíncronos com um circuito combinacional

Na figura 5.4 é mostrado um anel assíncrono com três registradores e o circuito combinacional CC1. Vamos considerar para efeito de cálculo da soma dos atrasos nesse circuito o tempo médio do atraso no circuito CC1.

Para se calcular o atraso total que um dado de informação terá para sair da posição inicial e chegar até o ponto P4 teremos que analisar o caminho que essa informação fará e os sinais de controle que serão gerados durante esse percurso. Para que a informação saia da posição inicial e chegue a P1, deverá percorrer o circuito CC1. O tempo utilizado para realizar o cálculo nesse circuito será o valor do tempo médio, representado por ACALC_m. Assim que a informação esteja disponível em P1 o circuito de detecção de fim de cálculo irá gerar o sinal PP1. Vamos considerar que esse circuito utilize um atraso ADET, que irá representar o atraso no circuito de detecção de fim de cálculo. Assim que o sinal PP1 é ativado, será levado para o circuito Ctr1, onde já está presente o sinal PP3, inicialmente em 0. Considerando que o circuito Ctr1 utilize um tempo de ACTR para gerar o sinal C1e ainda que o registrador utilize ASU+AREG, atrasos de *setup* e do registrador, para armazenar e disponibilizar esses valores nas suas saídas, teremos um total de ADET+ACTR+ASU+AREG para que o dado de informação saia de P1 e atinja P2. Uma vez em P2, o circuito de detecção de fim de cálculo DET2 irá verificar a informação e deverá gerar o sinal PP2, logo, os mesmos ADET devem ser gastos para que esse sinal seja gerado. O sinal de PP2 irá para o circuito de controle Ctr2, onde o sinal PP1 está em 0. Considerando o mesmo atraso para o Ctr2, representado por ACTR para que esse circuito produza o sinal C2 teremos novamente ADET+ACTR+ASU+AREG para que o sinal saia de P2 e chegue a P3. Uma vez com o dado em P3, o sinal de informação será detectado por DET3 que deverá gerar o sinal PP3 após os mesmos ADET. Nesse momento ocorre um evento que vai zerar o registrador REG1 do anel assíncrono. Quando o sinal PP3 é ativado, ele será levado também para o circuito de controle Ctr1 para zerar o valor do registrador REG1. Para essa operação podemos atribuir um atraso de AZERO, relativo ao tempo que esse circuito utiliza para zerar as suas saídas. Logo, devemos somar ao valor de ADET+ACTR+ASU+AREG gastos para que o dado saia de P2 e chegue a P3 o valor de ACTR+AZERO, relativos ao tempo que o sinal PP3 irá levar para acionar o sinal C1 do controle e zerar o conteúdo do registrador REG1. Seguindo o raciocínio, de P3 para P4, o sinal PP3 já foi gerado pelo circuito DET3, esse sinal será levado para o circuito de controle Ctr3 que já possui o sinal PP2 pronto. Considerando que o Ctr3 demore ACTR para gerar o sinal C3, teremos para esse caminho a soma dos atrasos de DET3, Ctr3 e do registrador, totalizando ADET+ACTR+ASU+AREG. No ponto P4 para que o circuito possa realimentar o valor para o circuito combinacional e realizar um novo ciclo, o valor armazenado no registrador REG2 também deverá ser zerado. Para que isso

ocorra, devemos levar em consideração o sinal PP1 no circuito de controle Ctr2, adicionando ACTR+AZERO do atraso do circuito de controle aos atrasos nesse caminho da posição inicial até P4. Totalizando os atrasos em cada posição no anel da figura 5.4 teremos o seguinte: ACALC pelo CC1, ADET+ACTR+ASU+AREG de P1 para P2, ADET+ACTR+ASU+AREG+ACTR+AZERO de P2 para P3 e ADET+ACTR+ASU+AREG+ACTR+AZERO de P3 para P4. O atraso de latência será ACALC+.3ADET+5ACTR+3ASU+3AREG+2AZERO. O maior atraso de *throughput* será ACALC+ADET+2ACTR+ASU+AREG+AZERO.

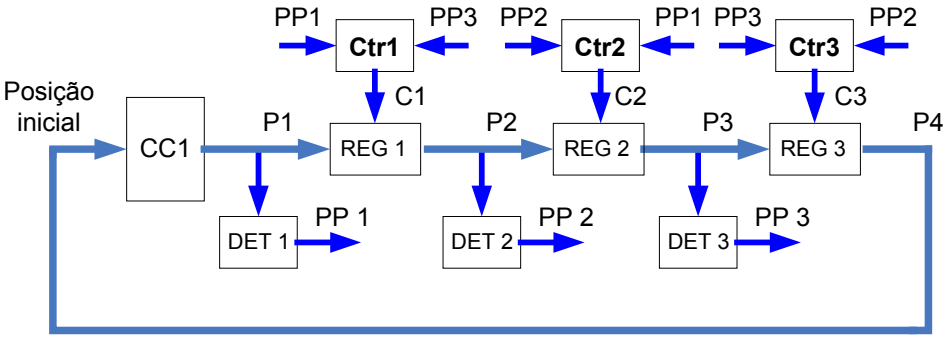



Figura 5.4: Anel assíncrono com 1 circuito combinacional

### *5.5.3.1 Circuitos assíncronos com dois circuitos combinacionais*

Na figura 5.5 é mostrado um anel assíncrono com três registradores e dois circuitos combinacionais CC2. Vamos considerar para efeito de cálculo o atraso médio em cada um dos circuitos CC2.

Vamos analisar esse anel da mesma forma que foi feita para o anel com apenas um circuito combinacional. Para calcular o atraso total que um dado de informação terá para sair da posição inicial e chegar até o ponto P5 vamos analisar ponto a ponto o caminho desse dado de informação e os sinais de controle que deverão estar envolvidos nos respectivos caminhos. Para que a informação saia da posição inicial e chegue a P1, deverá percorrer o primeiro circuito CC2. O tempo consumido para realizar o cálculo nesse circuito será o valor do tempo médio, equivalente a ACALC_m. Assim que a informação esteja disponível em P1 o circuito de detecção de fim de cálculo irá gerar o sinal PP1. Considerando o mesmo consumo que foi estimado para o anel com apenas um circuito combinacional, teremos ADET para gerar o sinal PP1. Assim que o sinal PP1 é ativado, será levado para o circuito Ctr1, onde já está presente o sinal PP3, inicialmente em 0. Considerando que o circuito Ctr1 demore ACTR para gerar o sinal C1 e ainda que o registrador demore os mesmos ASU+AREG para armazenar e disponibilizar a informação nas suas saídas, teremos o total de ADET+ACTR+ASU+AREG para que o dado de informação saia de P1 e atinja P2. Uma vez em P2, esse valor é enviado para o segundo circuito combinacional CC2 que irá demorar os mesmos ACALC_m para calcular o valor que estará disponível em P3. Uma vez em P3, o circuito de detecção de fim de cálculo DET2 irá verificar a informação e deverá gerar o sinal PP2, logo, os mesmos ADET devem ser utilizados para que esse sinal seja gerado. O sinal de PP2 irá para o circuito de controle Ctr2, onde o sinal PP1 está em 0. Considerando os mesmos ACTR para que esse circuito produza o sinal C2 teremos novamente ADET+ACTR+ASU+AREG para que o sinal saia de P3 e chegue a P4. Uma vez que o sinal de informação seja detectado por DET3 que deverá

gerar o sinal PP3 após os mesmos ADET. Quando o sinal PP3 é ativado, ele será levado também para o circuito de controle Ctr1 para zerar o valor do registrador REG1. Logo, será gerado o atraso de ACTR+AZERO, relativos ao tempo que o sinal PP3 irá levar para zerar o conteúdo do registrador REG1. Seguindo o raciocínio, de P4 para P5 a informação já foi detectada por DET3 e já foi gerado o sinal PP3. Esse sinal será levado para o circuito de controle Ctr3 que já possui o sinal PP2 pronto. Considerando que o Ctr3 demore ACTR para gerar o sinal C3, teremos para esse caminho a soma dos atrasos de DET3 e Ctr3, totalizando ADET+ACTR. A mesma consideração é feita quando o dado de informação chega a P5. Para que o anel possa realimentar o valor para o circuito combinacional e realizar um novo ciclo, o valor armazenado no registrador REG2 também deverá ser zerado. Para que isso ocorra, devemos levar em consideração o sinal PP1 no circuito de controle Ctr2, adicionando ACTR+AZERO aos atrasos nesse primeiro ciclo do dado de informação. Totalizando os atrasos em cada posição no anel da figura 5.5 teremos o seguinte atraso de latência: ACALC_m pelo primeiro CC2, ADET+ACTR+ASU+AREG de P1 para P2, ACALC_m pelo segundo CC2 de P2 para P3, ADET+ACTR+ASU+AREG+ACTR+AZERO de P3 para P4 e ADET+ACTR+ASU+AREG+ACTR+AZERO de P4 para P5. Totalizando 2ACALC_m+3ADET+5ACTR+3ASU+3AREG+2AZERO. O maior atraso de *throughput* será ACALC_m+ADET+2ACTR+ASU+AREG+AZERO.

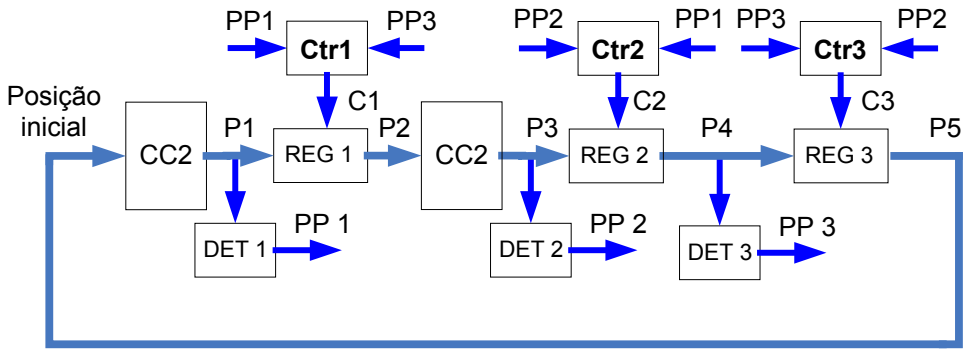



Figura 5.5: Anel assíncrono com 2 circuitos combinacionais

### *5.5.3.2 Circuitos assíncronos com três circuitos combinacionais*

Na figura 5.6 é mostrado um anel assíncrono com três registradores e três circuitos combinacionais CC3. Vamos considerar para efeito de cálculo o atraso médio em cada um dos circuitos CC3.

A análise será da mesma forma que os outros dois anéis assíncronos anteriores. Para calcular o atraso total que um dado de informação terá para sair da posição inicial e chegar até o ponto P6 vamos analisar ponto a ponto o caminho desse dado de informação e os sinais de controle que deverão estar envolvidos nos respectivos caminhos. Para que a informação saia da posição inicial e chegue a P1, deverá percorrer o primeiro circuito CC3. O tempo consumido para realizar o cálculo nesse circuito será o valor do tempo médio, equivalente a ACALC_m. Assim que a informação esteja disponível em P1 o circuito de detecção de fim de cálculo irá gerar o sinal PP1. Considerando ADET o atraso para gerar o sinal PP1. Assim que o sinal PP1 é ativado, será levado para o circuito Ctr1, onde já está presente o sinal PP3, inicialmente em 0. Considerando que o circuito Ctr1 demore ACTR para gerar o sinal C1 e ainda somando o atraso do registrador, teremos um total de ADET+ACTR+ASU+AREG para que a

informação saia de P1 e atinja P2. Uma vez em P2, esse valor é enviado para o segundo circuito combinacional CC3 que irá demorar os mesmos ACALC_m para calcular o valor que estará disponível em P3. Uma vez em P3, o circuito de detecção de fim de cálculo DET2 irá verificar a informação e deverá gerar o sinal PP2, logo, os mesmos ADET devem ser gastos para que esse sinal seja gerado. O sinal de PP2 irá para o circuito de controle Ctr2, onde o sinal PP1 está em 0. Considerando os mesmos ACTR para que esse circuito produza o sinal C2 e somando o atraso do registrador teremos novamente ADET+ACTR+ASU+AREG para que a informação saia de P3 e chegue a P4. Uma vez em P4, a informação ficará novamente disponível para o circuito combinacional CC3 que irá demorar os mesmos ACALC_m para calcular o valor que estará disponível em P5.

A partir do instante que a informação é detectada por DET3, este deverá gerar o sinal PP3 após os mesmos ADET de atraso deste circuito. Nesse momento deverá ser considerado o atraso que o circuito Ctr1 utiliza para zerar o REG1. Logo, devemos somar ao valor de ADET+ACTR+ASU+AREG o valor ACTR+AZERO, relativos ao tempo que o circuito Ctr1 irá levar para zerar o conteúdo do registrador REG1 e o atraso que este mesmo registrador leva para zerar seus valores nas suas saídas. Seguindo o raciocínio, de P5 para P6 a informação já foi detectada por DET3 e já foi gerado o sinal PP3. Esse sinal será levado para o circuito de controle Ctr3 que já possui o sinal PP2 pronto. Considerando que o Ctr3 demore ACTR para gerar o sinal C3, teremos para esse caminho a soma dos atrasos de DET3, Ctr3 e do registrador, totalizando ADET+ACTR+ASU+AREG. Para que o circuito possa realimentar o valor para o circuito combinacional e realizar um novo ciclo, o valor armazenado no registrador REG2 também deverá ser zerado. Para que isso ocorra, devemos levar em consideração o sinal PP1 no circuito de controle Ctr2, adicionando ACTR+AZERO aos atrasos nesse último estágio para o dado de informação. Totalizando os atrasos em cada posição no anel da figura 5.6 teremos o seguinte: ACALC_m pelo primeiro CC3, ADET+ACTR+ASU+AREG de P1 para P2, ACALC_m pelo segundo CC3 de P2 para P3, ADET+ACTR+ASU+AREG+ACTR+AZERO de P3 para P4, ACALC_m pelo terceiro CC3 de P4 para P5 e ADET+ACTR+ASU+AREG+ACTR+AZERO de P5 para P6. A latência total será de 3ACALC+3ADET+5ACTR+3ASU+3AREG+2AZERO. O maior atraso de *throughput* será ACALC+ADET+2ACTR+ASU+AREG+AZERO.

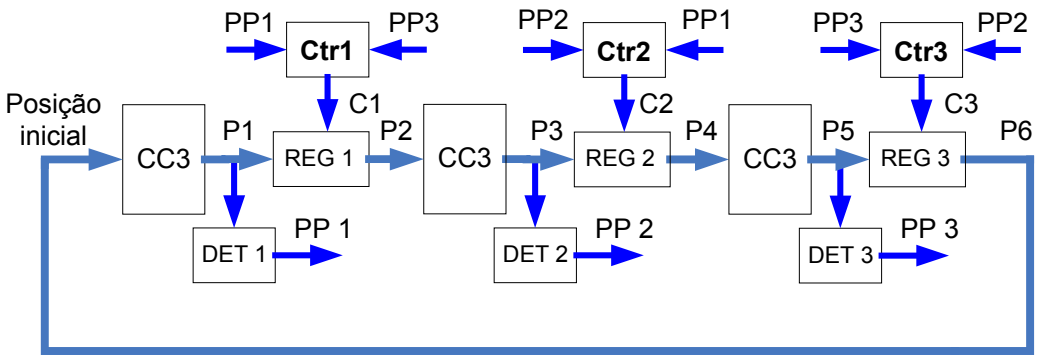



Figura 5.6: Anel assíncrono com 3 circuitos combinacionais

Na tabela 5.10 são concentrados os valores para atrasos de latência e *throughput*. É possível comparar os resultados dos circuitos síncronos e assíncronos através dessa tabela.

Tabela 5.10: Valores de atrasos de latência e throughput para circuitos síncronos e assíncronos

| Tipo de Circuito | Atraso de latência | Atraso de throughput |
|---|---|---|
| Circuito síncrono com um circuito combinacional | ACALC_M+AREG | ACALC_M+AREG |
| Circuito síncrono com dois circuitos combinacionais | 2ACALC_M+2AREG | ACALC_M+AREG |
| Circuito síncrono com três circuitos combinacionais | 3ACALC_M+3AREG | ACALC_M+AREG |
| Circuito assíncrono com um circuito combinacional | ACALC_m+.3ADET+5ACTR+3ASU+3AREG+2AZERO | ACALC_m+ADET+2ACTR+ASU+AREG+AZERO |
| Circuito assíncrono com dois circuitos combinacionais | 2ACALC_m+3ADET+5ACTR+3ASU+3AREG+2AZERO | ACALC_m+ADET+2ACTR+ASU+AREG+AZERO |
| Circuito assíncrono com três circuitos combinacionais | 3ACALC_m+3ADET+5ACTR+3ASU+3AREG+2AZERO | ACALC_m+ADET+2ACTR+ASU+AREG+AZERO |

### 5.5.4 Avaliação quanto ao aumento no número de registradores

Nessa seção iremos verificar como o aumento no número de registradores pode influenciar na melhoria do desempenho dos anéis assíncronos.

Como vimos anteriormente, um anel assíncrono necessita de no mínimo 3 registradores para compor a sua arquitetura, sendo possível aumentar esse número para 4, 5, 6 ou até mais registradores. Quando se aumenta o número de registradores cria-se no anel assíncrono a possibilidade de mais de um valor de informação estar circulando ao mesmo tempo num mesmo ciclo. No entanto, como foi possível verificar anteriormente, também aumenta a quantidade de caminhos e sinais de controle, acarretando um aumento de atrasos até que a informação complete esse ciclo.

Para podermos analisar os efeitos que terão os anéis quando se aumenta o número de registradores, tomaremos como base o anel com três circuitos combinacionais CC3, visto na seção 5.5.3.2. A esse circuito iremos acrescentar um registrador, como pode ser visto na figura 5.7. Esse anel de quatro registradores poderá ter até duas informações de dados circulando por ciclo. Para que seja feita uma análise comparativa com os outros anéis de três registradores, iremos calcular o atraso total para que esses dois dados de informação saiam da posição inicial e cheguem até P7. Vamos calcular os atrasos de latência e *throughput* desse circuito e depois compara-la ao anel de 3 registradores. Iniciando a análise, teremos que calcular o atraso total que um dado de informação terá ao percorrer o anel assíncrono de quatro registradores da posição inicial até a posição P7. Da posição inicial até P1 ele percorre CC3 que possui atraso ACALC_m, como foi visto. Em P1 é detectado pelo circuito DET1 que envia um sinal para Ctr1 e este irá gerar o sinal C1. Este atraso ainda será somado ao atraso do registrador, logo o atraso de P1 até P2 será de ADET+ACTR+ASU+AREG. Em P2 a informação passará novamente por CC3 que demorará ACALC_m para disponibilizar o resultado para P3. Com o resultado pronto em P3, o circuito de detecção de fim de cálculo DET2 irá gerar o sinal PP2 para o circuito Ctr2, que irá gerar o sinal C2 para o registrador. Logo após ADET+ACTR+ASU+AREG a informação estará em P4. Uma vez que o valor de informação chegue a P4, irá atravessar novamente CC3 com atraso de ACALC_m. Quando a informação chegar em P5 o circuito DET3 deverá gerar o sinal PP3 que irá para Ctr3 e também para Ctr1, onde deverá zerar o registrador REG1, atribui-se então um atraso de ACTR para o circuito de controle e AZERO para zerar o registrador. O

circuito de controle Ctr3 ao receber PP3 irá demorar ACTR para gerar o sinal C3. Uma vez que o valor de informação esteja em P6 o circuito DET4 irá gerar o sinal PP4, esse sinal além de ir para o circuito de controle Ctr4 deverá ir para o circuito de controle Ctr2 o qual deverá zerar o valor do registrador REG2.

Nesse instante podemos ter a situação de apenas um dado circulando no anel de quatro registradores e deduzir os atrasos de latência e *throughput* conforme os casos anteriores ou inserir mais um novo dado no anel de quatro registradores e realizar um novo cálculo para avaliar a latência e *throughput*.

Se considerarmos apenas um dado de informação circulando no anel, teremos latência igual a 3ACALC_m+4ADET+7ACTR+4ASU+4AREG+3AZERO e atraso de *throughput* igual a ACALC_m+ADET+2ACTR+ASU+AREG+AZERO.

Se inserirmos um novo dado no anel quando REG2 é zerado, é possível efetuar a avaliação da latência e atraso de *throughput* com dois dados de informação circulando no anel. Para inserir o segundo dado de informação, este passará antes pelo primeiro circuito combinacional CC3, após um atraso ACALC_m. Em P1 é detectado pelo circuito DET1 que envia um sinal para Ctr1 e este irá gerar o sinal C1. Este atraso ainda será somado ao atraso do registrador, logo o atraso de P1 até P2 será de ADET+ACTR+ASU+AREG. Em P2 essa segunda informação passará pelo segundo CC3 que terá um atraso ACALC_m para disponibilizar o resultado para P3. Com o resultado pronto em P3, o circuito de detecção de fim de cálculo DET2 irá gerar o sinal PP2 para o circuito Ctr2, que irá gerar o sinal C2 para o registrador. Logo após ADET+ACTR+ASU+AREG a informação estará em P4. Uma vez que o valor de informação chegue a P4, irá atravessar novamente CC3 com atraso de ACALC_m. Quando essa segunda informação chegar em P5 o circuito DET3 deverá gerar o sinal PP3 que irá para Ctr3 e também para Ctr1, onde deverá zerar o registrador REG1, atribui-se então um atraso de ACTR para o circuito de controle e AZERO para zerar o registrador. Seguindo o raciocínio, o primeiro dado de informação que se encontra em REG4 deverá circular nesse instante. Para que essa primeira informação seja armazenada no primeiro registrador, vamos considerar que se utilize ACALC_m+ADET+ACTR+ASU+AREG. A próxima ação deste anel de quatro estágios seria apagar a informação do registrador REG4. utilizando um atraso ACTR+AZERO. Após isso, se considerarmos somente a segunda informação circulando nesse anel até ser mostrada na saída do registrador REG4, teremos a latência e atraso de *throughput* quando duas informações circulam ao mesmo tempo no anel assíncrono. Assim, a segunda informação se encontra em REG2, deverá utilizar um atraso ACALC_m para atravessar o terceiro CC3. Após isso, será detectado pelo DET3 e gravada no REG3, utilizando um atraso de ADET+ACTR+ASU+AREG para realizar essa ação. Quando a informação for detectada por DET4, deverá ser apagado o conteúdo do segundo registrador, utilizando um atraso de ACTR+AZERO. E finalmente, quando a informação se encontrar em REG3, será detectado pelo DET4 e gravada no REG4, utilizando um atraso de ADET+ACTR+ASU+AREG para realizar essa ação. Quando a informação for gravada em REG4 a próxima ação será apagar o conteúdo do registrador REG3 para que seja possível circular no anel de quatro estágios, logo, deverá ser utilizado um atraso de ACTR+AZERO. Totalizando os atrasos em cada posição no anel da figura 5.7 teremos o seguinte: Para a latência 4ACALC_m+5ADET+9ACTR+5ASU+5AREG+4AZERO e atraso de *throughput* ACALC_m+2ADET+4ACTR+2ASU+2AREG+2AZERO.

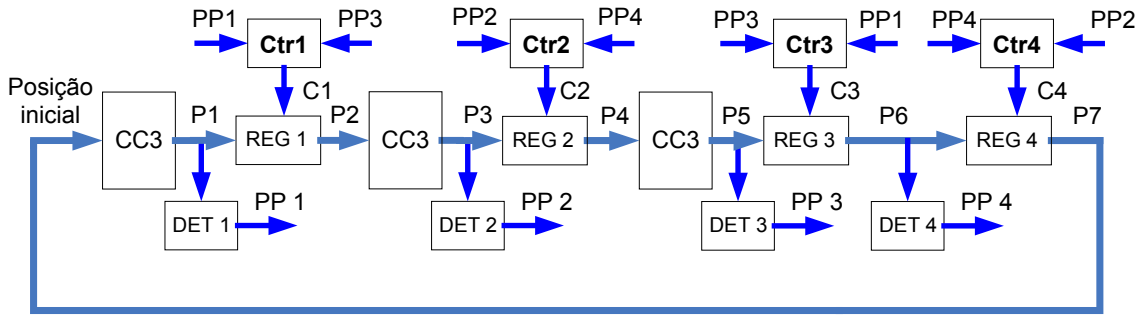



Figura 5.7: Anel assíncrono de 4 registradores - Estágios agrupados

Para um circuito de cinco registradores é possível fazer uma análise semelhante. Vamos considerar um circuito de com cinco registradores e três estágios combinacionais CC3 ao longo dele. A figura 5.8 representa um anel assíncrono de cinco registradores com três circuitos combinacionais. A análise será similar àquela feita para o circuito com quatro registradores. A única diferença é que o anel possui mais um circuito DET5 que irá gerar um sinal PP5 e um circuito de controle Ctr5, que deve gerar um sinal C5 para o REG5. Somando os atrasos para que apenas um dado de informação passe por esse circuito, teremos: 3ACALC_m+5ADET+9ACTR+5ASU+5AREG+4AZERO. E atraso de throughput igual a ACALC_m+ADET+2ACTR+ASU+AREG+AZERO.

Conforme foi feito no anel de quatro registradores, se fizermos a situação de que dois dados de informação circulem no anel ao mesmo tempo, teremos a seguinte análise: Inserindo o segundo dado de informação, este passará antes pelo primeiro circuito combinacional CC3, após um atraso ACALC_m. Em P1 é detectado pelo circuito DET1 que envia um sinal para Ctr1 e este irá gerar o sinal C1. Este atraso ainda será somado ao atraso do registrador, logo o atraso de P1 até P2 será de ADET+ACTR+ASU+AREG. Em P2 essa segunda informação passará pelo segundo CC3 que terá um atraso ACALC_m para disponibilizar o resultado para P3. Com o resultado pronto em P3, o circuito de detecção de fim de cálculo DET2 irá gerar o sinal PP2 para o circuito Ctr2, que irá gerar o sinal C2 para o registrador. Logo após ADET+ACTR+ASU+AREG a informação estará em P4. Uma vez que o valor de informação chegue a P4, irá atravessar novamente CC3 com atraso de ACALC_m. Quando essa segunda informação chegar em P5 o circuito DET3 deverá gerar o sinal PP3 que irá para Ctr3 e também para Ctr1, onde deverá zerar o registrador REG1, atribui-se então um atraso de ACTR para o circuito de controle e AZERO para zerar o registrador. Seguindo o raciocínio, o primeiro dado de informação que se encontra em REG4 deverá circular nesse instante. Para que essa primeira informação seja armazenada no primeiro registrador, vamos considerar que se utilize ACALC_m+ADET+ACTR+ASU+AREG. A próxima ação deste anel de quatro estágios seria apagar a informação do registrador REG4. utilizando um atraso ACTR+AZERO. Após isso, se considerarmos somente a segunda informação circulando nesse anel até ser mostrada na saída do registrador REG4, teremos a latência e atraso de *throughput* quando duas informações circulam ao mesmo tempo no anel assíncrono. Assim, a segunda informação se encontra em REG2, deverá utilizar um atraso ACALC_m para atravessar o terceiro CC3. Após isso, será detectado pelo DET3 e gravada no REG3, utilizando um atraso de ADET+ACTR+ASU+AREG para realizar essa ação. Quando a informação for detectada por DET4, deverá ser apagado o conteúdo do segundo registrador, utilizando um atraso de ACTR+AZERO. A seguir,

quando a informação se encontrar em REG3, será detectado pelo DET4 e gravada no REG4, utilizando um atraso de ADET+ACTR+ASU+AREG para realizar essa ação. Quando a informação for gravada em REG4 a próxima ação será apagar o conteúdo do registrador REG3, logo, deverá ser utilizado um atraso de ACTR+AZERO. Finalmente, quando a informação se encontrar em REG4, será detectado pelo DET5 e gravada no REG5, utilizando um atraso de ADET+ACTR+ASU+AREG para realizar essa ação. Quando a informação for gravada em REG5 a próxima ação será apagar o conteúdo do registrador REG3, para que seja possível circular no anel de cinco estágios, logo, deverá ser utilizado um atraso de ACTR+AZERO. Totalizando os atrasos em cada posição no anel da figura 5.8 teremos o seguinte: Para a latência 4ACALC_m+7ADET+13ACTR+7ASU+7AREG+6AZERO e atraso de *throughput* ACALC_m+3ADET+6ACTR+3ASU+3AREG+3AZERO.

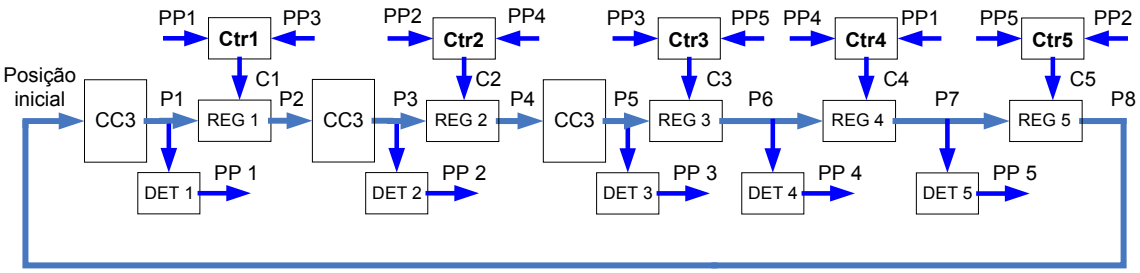



Figura 5.8: Anel assíncrono de 5 registradores

A conclusão que se pode tirar dessas avaliações é que o anel de quatro registradores possui um atraso ótimo quando possui dois dados circulando ao mesmo tempo. Além disso, foi possível perceber que o atraso de latência para que uma única informação saia da posição inicial e chegue até a posição final (no anel de quatro registradores é a posição P7 e no anel de cinco registradores é a posição P8) será sempre um múltiplo do número de registradores e da quantidade de estágios combinacionais nesse circuito. Por exemplo, sendo k igual ao número de circuitos combinacionais e n igual ao número de registradores, teremos a latência total como **(k)ACALC_m+(n)ADET+(2n-1)ACTR+(n)ASU+(n)AREG+(n-1)AZERO.**

Já para o atraso de *throughput* desses circuitos é interessante notar o seguinte. Nos circuitos em anel de quatro registradores podemos ter até dois dados circulando ao mesmo tempo num único ciclo. Nos circuitos com cinco registradores existe um atraso de *throughput* maior que no de quatro registradores, pois, possui a limitação de apenas dois dados de informação circularem em um número maior de estágios de registradores.

Na tabela 5.11 estão organizados todos os atrasos para os tipos de anéis assíncronos de quatro e cinco registradores. Notar que o atraso de *throughput* para o anel de quatro registradores com dois dados circulando é menor que o de cinco registradores com dois dados circulando.

Tabela 5.11: Valores de atrasos de latência e *throughput* para anéis assíncronos de quatro e cinco registradores

| Tipo de Circuito | Atraso de latência | Atraso de throughput |
|---|---|---|
| Anel de quatro registradores com circuitos combinacionais CC3 – 1 dado circulando | 3ACALC_m+4ADET+7ACTR+4AREG+3AZERO | ACALC_m+ADET+2ACTR+AREG+AZERO |
| Anel de quatro registradores com circuitos combinacionais CC3 – 2 dados circulando | 4ACALC_m+5ADET+9ACTR+5AREG+4AZERO | ACALC_m+2ADET+4ACTR+2AREG+2AZERO |
| Anel de cinco registradores com circuitos combinacionais CC3 – 1 dado circulando | 3ACALC_m+5ADET+9ACTR+5AREG+4AZERO | ACALC_m+ADET+2ACTR+AREG+AZERO |
| Anel de cinco registradores com circuitos combinacionais CC3 – 2 dados circulando | 4ACALC_m+7ADET+13ACTR+7AREG+6AZERO | ACALC_m+3ADET+6ACTR+3AREG+3AZERO |

## 5.6 Sumário

Nesse capítulo nós apresentamos uma comparação quantitativa entre os circuitos síncronos e os circuitos assíncronos através da comparação entre os custos de implementação das partes combinacionais como os circuitos full adder, RCA de 4, 8, 16 e 32 bits e do registrador assíncrono. Além disso, foi realizada uma comparação das partes seqüenciais nos circuitos síncronos e nos assíncronos. Esses circuitos foram avaliados com relação aos atrasos de latência e *throughput*. Os circuitos assíncronos foram avaliados também com relação ao número de estágios e a relação que estes possuem com os atrasos comentados anteriormente.

# 6 RESULTADOS

Os resultados obtidos através da síntese lógica dos circuitos utilizados nessa dissertação assim como uma breve descrição das funções serão apresentados a seguir. Os detalhes de cada tipo de implementação e algumas observações são feitas em cada subseção.

As tabelas de resultados são compostas pelo número de elementos de programação utilizados em várias ferramentas de síntese lógica. Os resultados foram obtidos para circuitos de 4, 8, 16 e 32bits com relação às tecnologias FPGAs e CPLDs. São feitas algumas observações com relação ao aumento do número de elementos de programação quando se aumenta o número de bits dos vetores de entrada e entre os diferentes fabricantes. Também é feita uma análise percentual com relação ao consumo de elementos lógicos utilizados na síntese desses circuitos, como ao número de LUTs utilizadas frente ao número de macrocélulas. Ao final de cada subseção é feita uma análise global com as observações extraídas de cada tabela e uma avaliação para cada estilo de circuito assíncrono.

## 6.1 Resumo

Nesse capítulo são apresentados os circuitos que foram implementados através da arquitetura assíncrona de um anel com três registradores. Ainda, para cada circuito, foi feita a síntese lógica através das ferramentas comerciais dos fabricantes ALTERA, XILINX e ACTEL e obtida a quantidade de elementos de programação para as tecnologias CPLD e FPGA.

## 6.2 Introdução

Durante o trabalho da dissertação foram escolhidos vários tipos de funções que poderiam ser implementadas e com isso realizar os testes com as ferramentas de síntese. A escolha dessas funções teve como fundamento realizar um gradativo aumento da complexidade das operações que poderiam ser realizadas por esses circuitos. Existia, além disso, a necessidade de demonstrar a modularidade dos circuitos assíncronos através da possibilidade de se projetar circuitos combinacionais com técnicas de implementação diferentes e posteriormente verificar qual deles utilizou uma área maior ou menor que os outros.

Os circuitos escolhidos foram contador, divisor inteiro, divisor de resto (cálculo do resto), mínimo múltiplo comum, máximo divisor comum e raiz quadrada. Os três primeiros utilizam apenas um barramento de n bits para a informação que está

circulando em cada ciclo no anel e os outros três utilizam dois barramentos de n bits para a informação nessas mesmas condições. Os circuitos foram feitos através de codificação dual-rail para 4, 8, 16 e 32 bits.

De início é apresentada a descrição de um anel realimentado, ao qual também foi feita a síntese lógica. Os parâmetros utilizados nessa síntese são iguais aos utilizados pelos demais circuitos projetados. A única diferença do anel realimentado é que não possui circuitos combinacionais ou função lógica alguma a ser realizada com os vetores de entrada.

## 6.3 Anel realimentado

### 6.3.1 Descrição

O anel realimentado consiste na organização de três registradores em série, sendo que a saída do último registrador está ligada à entrada do primeiro. O dado inicial, representado por Dado_i, na figura 6.1, é inserido através do MUX quando o sinal seletor está em 0. O dado irá circular através do anel de três registradores sem realizar qualquer cálculo, ou seja, sem atravessar qualquer circuito combinacional.

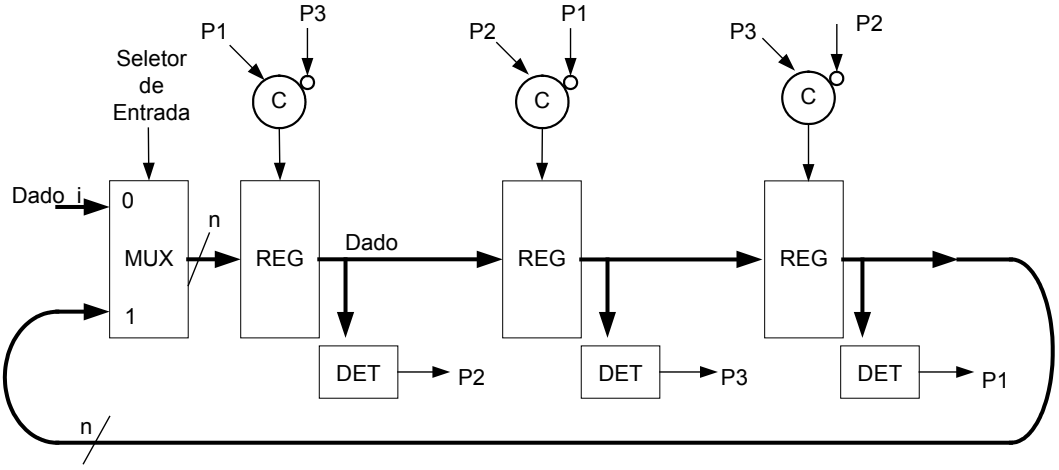



Figura 6.1: Esquema do anel realimentado

### 6.3.2 Resultados

Nas tabelas 6.1, 6.2 e 6.3 são mostrados os resultados para a síntese nas ferramentas dos fabricantes ALTERA, XILINX e ACTEL, respectivamente. Foram simulados dois tipos de anéis realimentados. O primeiro tipo apresenta apenas um único barramento para a informação que foi inserida pelo MUX de entrada. Esse único valor circula pelo anel. O segundo tipo possui dois barramentos de informação e possibilita que dois valores de informação circulem ao mesmo tempo através do anel. A diferença essencial de um para o outro é o aumento do número de registradores e do circuito de detecção de fim de cálculo.

Tabela 6.1: Valores de Células Lógicas para o Anel realimentado para Fabricante ALTERA – CPLD (Família MAX 7000AE) e FPGA (Família FLEX10KE)

| Anel realimentado | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Tipo de Anel | 4 bits | | 8bits | | 16bits | | 32bits | |
| | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD |
| | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) |
| 1 dado circulante | 62 | 49 | 115 | 89 | 222 | 166 | 398 | 392 |
| 2 dados circulantes | 113 | 82 | 217 | 153 | 421 | 377 | 794 | 582 (**) |

(*) *Macrocell* representa o número de macrocélulas para CPLDs.

(**) A ferramenta enviou uma mensagem de erro informando que a quantidade de macrocélulas excedeu o limite do circuito.

Tabela 6.2: Valores de Células Lógicas para o Anel realimentado para Fabricante XILINX – CPLD (Família XC9500XV) e FPGA (Família SPARTAN2)

| Anel realimentado | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Tipo de Anel | 4 bits | | 8bits | | 16bits | | 32bits | |
| | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD |
| | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) |
| 1 dado circulante | 93 | 47 | 176 | 83 | 341 | 162 | 680 | 249 (**) |
| 2 dados circulantes | 165 | 81 | 309 | 152 | 616 | (**) | 1356 | (**) |

(*) *Macrocell* representa o número de macrocélulas para CPLDs.

(**) A ferramenta enviou uma mensagem de erro.

Tabela 6.3: Valores de Células Lógicas para o Anel realimentado para Fabricante ACTEL – FPGA Antifusível (Família AXCELERATOR) e FPGA SRAM (Família 500K)

| Anel realimentado | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Tipo de Anel | 4 bits | | 8bits | | 16bits | | 32bits | |
| | FPGA/AF | FPGA/SR | FPGA/AF | FPGA/SR | FPGA/AF | FPGA/SR | FPGA/AF | FPGA /SR |
| | Número de células (LUTs) | Número de células (A) | Número de células (LUTs) | Número de células (A) | Número de células (LUTs) | Número de células (A) | Número de células (LUTs) | Número de células (A) |
| 1 dado circulante | 57 | 75 | 111 | 148 | 218 | 281 | 439 | 549 |
| 2 dados circulantes | 112 | 148 | 218 | 289 | 452 | 558 | 1101 | 1091 |

(A): A ferramenta informa os resultados através da descrição *Área*

Tabela 6.4: Valores em percentuais de acréscimos de elementos de programação para os circuitos nos fabricante ALTERA – CPLD (Família MAX 7000AE) e FPGA (Família FLEX10KE)

| Anel realimentado | | | | | | |
|---|---|---|---|---|---|---|
| Tipo de Anel | FPGA | | | CPLD | | |
| | De 4 para 8bits | De 8 para 16bits | De 16 para 32bits | De 4 para 8bits | De 8 para 16bits | De 16 para 32bits |
| 1 dado | 85,48% | 93,04% | 79,28% | 81,63% | 86,52% | 136,14% |
| 2 dados | 92,04% | 94,01% | 88,60% | 86,59% | 146,41% | - |

(-) Não foi possível realizar o cálculo para percentual de aumento do circuito

Tabela 6.5: Valores em percentuais de acréscimos de elementos de programação para os circuitos nos fabricante XILINX – CPLD (Família XC9500XV) e FPGA (Família SPARTAN2)

| Anel realimentado | | | | | | |
|---|---|---|---|---|---|---|
| Tipo de Anel | FPGA | | | CPLD | | |
| | De 4 para 8bits | De 8 para 16bits | De 16 para 32bits | De 4 para 8bits | De 8 para 16bits | De 16 para 32bits |
| 1 dado | 89,25% | 93,75% | 99,41% | 76,60% | 95,18% | - |
| 2 dados | 87,27% | 99,35% | 120,13% | 87,65% | - | - |

(-) Não foi possível realizar o cálculo para percentual de aumento do circuito

Tabela 6.6: Valores em percentuais de acréscimos de elementos de programação para os circuitos nos fabricante ACTEL – FPGA Antifusível (Família AXCELERATOR) e FPGA SRAM (Família 500K)

| Anel realimentado | | | | | | |
|---|---|---|---|---|---|---|
| Tipo de Anel | FPGA Antifusível | | | FPGA SRAM | | |
| | De 4 para 8bits | De 8 para 16bits | De 16 para 32bits | De 4 para 8bits | De 8 para 16bits | De 16 para 32bits |
| 1 dado | 94,74% | 96,40% | 101,38% | 97,33% | 89,86% | 95,37% |
| 2 dados | 94,64% | 107,34% | 143,58% | 95,27% | 93,08% | 95,52% |

As tabelas 6.4, 6.5 e 6.6 mostram os percentuais de acréscimos de uma versão para a seguinte. Na versão de 4bits para a de 8bits, na versão de 8bits para a de 16bits e na versão de 16bits para a de 32bits. Esses resultados mostram os aumentos relativos quando se projeta em escala e os custos em elementos de programação decorrentes disso nessas ferramentas. Para o tipo de arquitetura podemos concluir que os CPLDs do fabricante ACTEL obteve um menor aumento relativo de área frentes aos demais fabricantes nessa arquitetura de PLD. Em relação aos FPGAs, existe a vantagem da ferramenta do fabricante ALTERA para essa arquitetura.

## 6.4 Contador

### 6.4.1 Descrição

O circuito contador realiza acréscimos sucessivos de 1 unidade de n bits ao valor inserido no anel. Da mesma forma como foi dito para o anel realimentado, o valor de entrada Dado_i, mostrado na figura 6.2 é inserido no anel através do MUX de entrada. Este valor passa através do primeiro registrador e posteriormente é mostrado na entrada do circuito somador. Esse circuito irá somar o valor 1 ao valor amostrado na sua entrada pelo registrador. Quando o valor de Dado+1 atravessar o último registrador, será levado ao MUX onde o seletor de entrada deverá ter o valor 1. Para essa condição, o valor Dado+1 será inserido novamente na entrada do primeiro registrador, realizando com isso a realimentação do anel assíncrono.

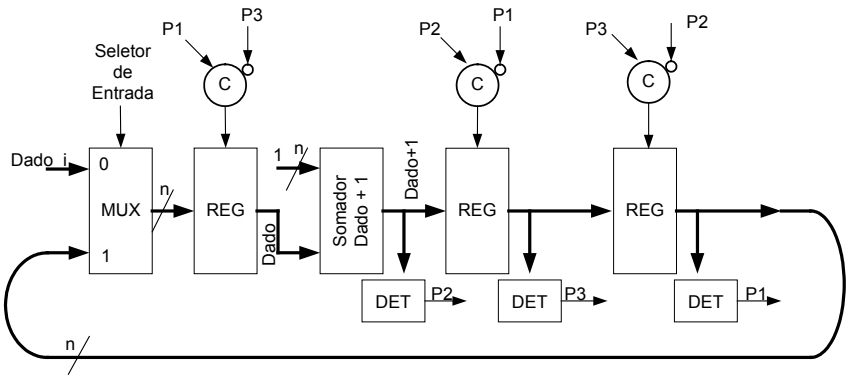



Figura 6.2: Esquema do circuito contador

### 6.4.2 Resultados

No contador assim como no anel realimentado, teremos as tabelas 6.7, 6.8 e 6.9 mostrando os resultados para a síntese nas ferramentas dos fabricantes ALTERA, XILINX e ACTEL, respectivamente. No entanto, esse circuito possui uma diferença com relação ao anterior. Nesse existe a presença de um circuito combinacional que realiza a soma+1 ao valor que vem do primeiro registrador. Logo, as tabelas de resultados vão privilegiar os estilos de projetos mostrados na seção 4 dessa dissertação como o somador do tipo comportamental com indicação forte do sinal, do tipo DIMS, do tipo NCL e ainda a derivação de circuito combinacional síncrono, ao qual é chamado de derivação de lógica nas tabelas.

Ao final são mostradas as tabelas 6.10, 6.11 e 6.12 que possuem os aumentos percentuais de elementos de programação em relação ao estilo de projeto que foi mais econômico, ou seja, aquele que consumiu menos elementos lógicos para ser criado pela ferramenta. Nessas tabelas o resultado utilizado como referência é indicado pelo percentual de 0% de acréscimo. Os outros valores dentro de cada grupo de 4, 8, 16 e de 32bits são valores percentuais acima desse.

Tabela 6.7: Valores de Células Lógicas para o Contador para Fabricante ALTERA – CPLD (Família MAX 7000AE) e FPGA (Família FLEX10KE)

| Contador | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de projeto | 4 bits | | 8bits | | 16bits | | 32bits | |
| | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD |
| | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) |
| Comportamental com indicação forte do sinal | 102 | 76 | 206 | 154 | 415 | 307 | 829 | - |
| Por DIMS | 73 | 80 | 142 | 153 | 281 | 311 | 556 | - |
| Por NCL | 91 | 75 | 177 | 155 | 351 | 310 | 698 | - |
| Derivação de lógica | 95 | 66 | 192 | 126 | 388 | 226 | 773 | - |

(*) *Macrocell* representa o número de macrocélulas para CPLDs.

(-) Não informado pela ferramenta.

Tabela 6.8: Valores de Células Lógicas para o Contador para Fabricante XILINX – CPLD (Família XC9500XV) e FPGA (Família SPARTAN2)

| Contador | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de projeto | 4 bits | | 8bits | | 16bits | | 32bits | |
| | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD |
| | Número de células (LUTs) | Número de macrocell | Número de células (LUTs) | Número de macrocell | Número de células (LUTs) | Número de macrocell | Número de células (LUTs) | Número de macrocell |
| Comportamental com indicação forte do sinal | 139 | 56 | 289 | 126 | 569 | - | 1090 | - |
| Por DIMS | 149 | 76 | 278 | 139 | 542 | - | 1077 | - |
| Por NCL | 137 | 70 | 252 | 140 | 506 | - | 1007 | - |
| Derivação de lógica | 132 | 64 | 244 | 116 | 512 | 228 | 1022 | - |

(-) Não informado pela ferramenta.

Tabela 6.9: Valores de Células Lógicas para o Contador para Fabricante ACTEL – FPGA Antifusível (Família AXCELERATOR) e FPGA SRAM (Família 500K)

| Contador | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de projetos | 4 bits | | 8bits | | 16bits | | 32bits | |
| | FPGA/ AF | FPGA/SR | FPGA/AF | FPGA/SR | FPGA/ AF | FPGA/SR | FPGA/ AF | FPGA/SR |
| | Número de células (LUTs) | Número de células (A) | Número de células (LUTs) | Número de células (A) | Número de células (LUTs) | Número de células (A) | Número de células (LUTs) | Número de células (A) |
| Comportamental com indicação forte do sinal | 110 | 130 | 235 | 278 | 478 | 552 | 953 | 1107 |
| Por DIMS | 112 | 166 | 234 | 349 | 487 | 690 | 972 | 1351 |
| Por NCL | 134 | 173 | 272 | 371 | 553 | 735 | 1109 | 1467 |
| Derivação de lógica | 91 | 120 | 190 | 256 | 386 | 506 | 1022 | 1007 |

(A): A ferramenta informa os resultados através da descrição *Area*

Com esses estilo de projeto é possível verificar que o circuito feito através de DIMS foi mais eficiente para o fabricante ALTERA em FPGAs mas menos eficiente para os seus circuitos feitos através de CPLDs. Para o fabricante XILINX teremos uma disputa entre os circuitos com os estilos NCL e derivação de lógica para os FPGAs e uma tendência de ser mais eficiente com derivação de lógica para os CPLDs. Já no fabricante ACTEL, predominantemente derivação de lógica é mais eficiente para os FPGAs e também para os seus CPLDs.

Tabela 6.10: Valores em percentuais de acréscimos de elementos de programação para o contador para fabricante ALTERA – CPLD (família MAX 7000AE) e FPGA (família FLEX10KE)

| Contador | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilo de projeto | FPGA | | | | CPLD | | | |
| | 4bits | 8bits | 16bits | 32bits | 4bits | 8bits | 16bits | 32bits |
| Comportamental com indicação forte do sinal | 39,73% | 45,07% | 47,69% | 49,10% | 15,15% | 22,22% | 35,84% | - |
| Por DIMS | 0% | 0% | 0% | 0% | 21,21% | 21,43% | 37,61% | - |
| Por NCL | 24,66% | 24,65% | 24,91% | 25,54% | 13,64% | 23,02% | 37,17% | - |
| Derivação de lógica | 30,14% | 35,21% | 38,08% | 39,03% | 0% | 0% | 0% | - |

(-) Não foi possível realizar a comparação por falta de dados.

Tabela 6.11: Valores em percentuais de acréscimos de elementos de programação para o contador para Fabricante XILINX – CPLD (família XC9500XV) e FPGA (família SPARTAN2)

| Contador | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilo de projeto | FPGA | | | | CPLD | | | |
| | 4bits | 8bits | 16bits | 32bits | 4bits | 8bits | 16bits | 32bits |
| Comportamental com indicação forte do sinal | 5,30% | 18,44% | 12,45% | 8,24% | 0% | 8,62% | - | - |
| Por DIMS | 12,88% | 13,93% | 7,11% | 6,95% | 35,71% | 19,83% | - | - |
| Por NCL | 3,79% | 3,28% | 0% | 0% | 25,00% | 20,69% | - | - |
| Derivação de lógica | 0% | 0% | 1,19% | 1,49% | 14,29% | 0% | 0% | - |

(-) Não foi possível realizar a comparação por falta de dados.

Tabela 6.12: Valores em percentuais de acréscimos de elementos de programação para o contador para fabricante ACTEL – FPGA Antifusível (família AXCELERATOR) e FPGA SRAM (família 500K)

| Contador | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilo de projeto | FPGA Antifusível | | | | FPGA SRAM | | | |
| | 4bits | 8bits | 16bits | 32bits | 4bits | 8bits | 16bits | 32bits |
| Comporta mental com indicação forte do sinal | 20,88% | 23,68% | 23,83% | 0% | 8,33% | 8,59% | 9,09% | 9,93% |
| Por DIMS | 23,08% | 23,16% | 26,17% | 1,99% | 38,33% | 36,33% | 36,36% | 34,16% |
| Por NCL | 47,25% | 43,16% | 43,26% | 16,37% | 44,17% | 44,92% | 45,26% | 45,68% |
| Derivação de lógica | 0% | 0% | 0% | 7,24% | 0% | 0% | 0% | 0% |

## 6.5 Divisor inteiro

### 6.5.1 Descrição

O circuito divisor inteiro é bem similar ao circuito contador, mostrado anteriormente, com a diferença que ao invés de realizar uma soma, será feita uma subtração e o valor subtraído será o valor do divisor, que deverá ser fornecido ao circuito do anel assíncrono mas não será utilizado como uma informação que sofrerá realimentação. Em resumo, o valor de entrada representa o dividendo de uma operação de divisão. Este valor é o Dado_i, mostrado na figura 6.3 sendo inserido no anel através do MUX de entrada. Este valor passa através do primeiro registrador e posteriormente é mostrado na entrada do circuito subtrator. Esse circuito fará a subtração do valor amostrado pelo valor do divisor. Quando o valor de Dado-divisor atravessar o último registrador, será levado ao MUX onde o seletor de entrada deverá ter o valor 1. Para essa condição, o valor Dado-divisor será inserido novamente na entrada do primeiro registrador, realizando com isso a realimentação do anel assíncrono. Essa operação será repetida até que o valor do dividendo, que no circuito é chamado de Dado, for menor que o valor do divisor. Nesse momento o circuito reconhece que a operação de divisão inteira chegou ao fim e que o número de vezes que o dado foi realimentado representa o valor do quociente dessa divisão.

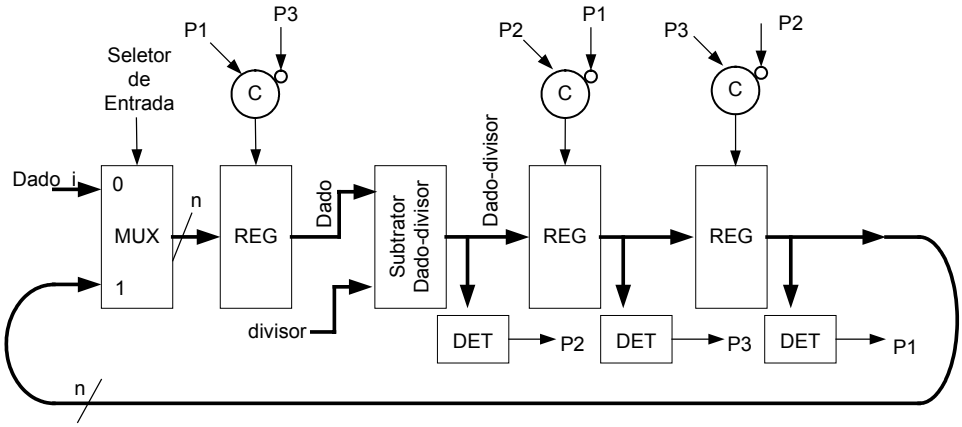



Figura 6.3: Esquema do circuito divisor inteiro

### 6.5.2 Resultados

Como no circuito contador, teremos as tabelas 6.13, 6.14 e 6.15 mostrando os resultados para a síntese nas ferramentas dos fabricantes ALTERA, XILINX e ACTEL, respectivamente. O circuito divisor inteiro utiliza o mesmo circuito combinacional que o contador, no entanto realiza uma subtração do dividendo com o divisor ao invés de uma soma+1 com o valor que lhe é entregue pelo primeiro registrador no anel. Como anteriormente, os resultados mostrados nessas tabelas vão privilegiar os estilos de projetos dos circuitos combinacionais somador do tipo comportamental com indicação forte do sinal, do tipo DIMS, do tipo NCL e ainda a derivação de circuito combinacional síncrono, chamado de derivação de lógica nas tabelas.

Ao final são mostradas as tabelas 6.16, 6.17 e 6.18 que possuem os aumentos percentuais de elementos de programação em relação ao estilo de projeto que foi mais econômico, ou seja, aquele que consumiu menos elementos lógicos para ser criado pela ferramenta. Nessas tabelas o resultado utilizado como referência está indicado pelo percentual de 0% de acréscimo. Os outros valores dentro de cada grupo de 4, 8, 16 e de 32 bits são valores percentuais acima desse.

Tabela 6.13: Valores de Células Lógicas para o Divisor inteiro para Fabricante ALTERA – CPLD (Família MAX 7000AE) e FPGA (Família FLEX10KE)

| Divisor inteiro | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de projeto | 4 bits | | 8bits | | 16bits | | 32bits | |
| | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD |
| | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) |
| Comportamental com indicação forte do sinal | 126 | 105 | 247 | 210 | 469 | 437 | 987 | - |
| Por DIMS | 102 | 113 | 208 | 175 | 418 | 376 | 837 | - |
| Por NCL | 102 | 85 | 200 | 172 | 398 | 335 | 793 | - |
| Derivação de lógica | 119 | 76 | 231 | 153 | 461 | 299 | 919 | - |

(*) *Macrocell* representa o número de macrocélulas para CPLDs.

(-) Não informado pela ferramenta.

Tabela 6.14: Valores de Células Lógicas para o Divisor inteiro para Fabricante XILINX – CPLD (Família XC9500XV) e FPGA (Família SPARTAN2)

| Divisor inteiro | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de projeto | 4 bits | | 8bits | | 16bits | | 32bits | |
| | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD |
| | Número de células (LUTs) | Número de macrocel (*) | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) |
| Comportamental com indicação forte do sinal | 217 | 76 | 406 | 151 | 781 | - | 1689 | - |
| Por DIMS | 165 | 76 | 298 | 142 | 583 | - | 1157 | - |
| Por NCL | 174 | 82 | 305 | 156 | 615 | - | NA | - |
| Derivação de lógica | 170 | 61 | 288 | 110 | 600 | - | 1199 | - |

(*) *Macrocell* representa o número de macrocélulas para CPLDs.

(-) Não informado pela ferramenta.

Tabela 6.15: Valores de Células Lógicas para o Divisor inteiro para Fabricante ACTEL – FPGA Antifusível (Família AXCELERATOR) e FPGA SRAM (Família 500K)

| Divisor inteiro | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de projeto | 4 bits | | 8bits | | 16bits | | 32bits | |
| | FPGA/AF | FPGA/SR | FPGA/AF | FPGA/SR | FPGA/AF | FPGA/SR | FPGA/AF | FPGA/SR |
| | Número de células (LUTs) | Número de células (A) | Número de células (LUTs) | Número de células (A) | Número de células (LUTs) | Número de células (A) | Número de células (LUTs) | Número de células (A) |
| Comportamental com indicação forte do sinal | 141 | 175 | 299 | 374 | 602 | 733 | 1248 | 1470 |
| Por DIMS | 137 | 203 | 299 | 429 | 619 | 856 | 1146 | 1715 |
| Por NCL | 161 | 213 | 297 | 448 | 612 | 918 | 1252 | 1776 |
| Derivação de lógica | 127 | 146 | 274 | 310 | 558 | 615 | 1168 | 1225 |

(A): A ferramenta informa os resultados através da descrição *Área.*

De acordo com a tabela 6.16, verificamos que o circuito feito através de NCL foi um pouco mais eficiente para o fabricante ALTERA em FPGAs mas menos eficiente para os seus circuitos feitos através de CPLDs. Para os circuitos feitos em CPLDs da ALTERA o estilo mais eficiente foi a derivação de lógica. Na tabela 6.17 para o fabricante XILINX teremos uma disputa entre os circuitos com os estilos DIMS e derivação de lógica para os FPGAs e uma tendência de ser mais eficiente com derivação de lógica para os CPLDs. Já na tabela 6.18, do fabricante ACTEL, predominantemente derivação de lógica ganha para os FPGAs e para os seus CPLDs (assim como foi para o circuito contador, mostrado anteriormente).

Tabela 6.16: Valores em percentuais de acréscimos de elementos de programação para o divisor inteiro para Fabricante ALTERA – CPLD (Família MAX 7000AE) e FPGA (Família FLEX10KE)

| Divisor inteiro | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilo de projeto | FPGA | | | | CPLD | | | |
| | 4bits | 8bits | 16bits | 32bits | 4bits | 8bits | 16bits | 32bits |
| Comportamental com indicação forte do sinal | 23,53% | 23,50% | 17,84% | 24,46% | 38,16% | 37,25% | 46,15% | - |
| Por DIMS | 0% | 4,00% | 5,03% | 5,55% | 48,68% | 14,38% | 25,75% | - |
| Por NCL | 0% | 0% | 0% | 0% | 11,84% | 12,42% | 12,04% | - |
| Derivação de lógica | 16,67% | 15,50% | 15,83% | 15,89% | 0% | 0% | 0% | - |

(-) A comparação não poder ser feita.

Tabela 6.17: Valores em percentuais de acréscimos de elementos de programação para o divisor inteiro para Fabricante XILINX – CPLD (Família XC9500XV) e FPGA (Família SPARTAN2)

| Divisor inteiro | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilo de projeto | FPGA | | | | CPLD | | | |
| | 4bits | 8bits | 16bits | 32bits | 4bits | 8bits | 16bits | 32bits |
| Comportamental com indicação forte do sinal | 31,52% | 40,97% | 33,96% | 45,98% | 24,59% | 37,27% | - | - |
| Por DIMS | 0% | 3,47% | 0% | 0% | 24,59% | 29,09% | - | - |
| Por NCL | 5,45% | 5,90% | 5,49% | - | 34,43% | 41,82% | - | - |
| Derivação de lógica | 3,03% | 0% | 2,92% | 3,63% | 0% | 0% | - | - |

(-) A comparação não poder ser feita.

Tabela 6.18: Valores em percentuais de acréscimos de elementos de programação para o divisor inteiro para Fabricante ACTEL – FPGA Antifusível (Família AXCELERATOR) e FPGA SRAM (Família 500K)

| Divisor inteiro | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | FPGA Antifusível | | | | FPGA SRAM | | | |
| | 4bits | 8bits | 16bits | 32bits | 4bits | 8bits | 16bits | 32bits |
| Comportamental com indicação forte do sinal | 11,02% | 9,12% | 7,89% | 8,90% | 19,86% | 20,65% | 19,19% | 20,00% |
| Por DIMS | 7,87% | 9,12% | 10,93% | 0% | 39,04% | 38,39% | 39,19% | 40,00% |
| Por NCL | 26,77% | 8,39% | 9,68% | 9,25% | 45,89% | 44,52% | 49,22% | 44,98% |
| Derivação de lógica | 0% | 0% | 0% | 1,92% | 0% | 0% | 0% | 0% |

## 6.6 Divisor de resto

### 6.6.1 Descrição

O circuito divisor de resto tem o objetivo de calcular o resto da divisão do valor de entrada pelo valor do divisor, como mostra a figura 6.4. Em resumo, o valor de entrada representa o dividendo de uma operação de divisão, chamado de Dado_i. Este valor é inserido no anel através do MUX de entrada e passa através do primeiro registrador. Em seguida, é mostrado na entrada de um circuito que realiza o produto desse valor por 2, ou seja, realiza o deslocamento dos valores de seus bits de uma posição para a esquerda. Esse valor deslocado é levado para o próximo registrador e após a sua passagem por ele, será amostrado na entrada do circuito subtrator. Esse circuito deverá subtrair o valor que veio do registrador pelo divisor. Se o valor que entregue pelo registrador for maior que o divisor, será gerado um carry out igual a 1, senão esse carry será 0. Este valor de carry out é importante pois ele seleciona as entradas do MUX intermediário. Se o valor entregue pelo registrador for menor que o divisor esse valor será repassado para o registrador seguinte pelo MUX, situação na qual o sinal seletor é igual a 0. Quando o valor da saída do MUX intermediário atravessar o último registrador, será levado ao MUX de entrada onde o seletor de entrada deverá ter o valor 1. O cálculo do resto se dá através da seqüência de carry out que são acumulados num circuito shift left, não representado nessa figura. Como condição de parada, é definido pelo usuário uma quantidade de bits que o resto dessa divisão deverá possuir. Assim, quando o valor acumulado pelo circuito shift left atingir essa quantidade pré-determinada de bits para o resto, o anel irá parar de circular.

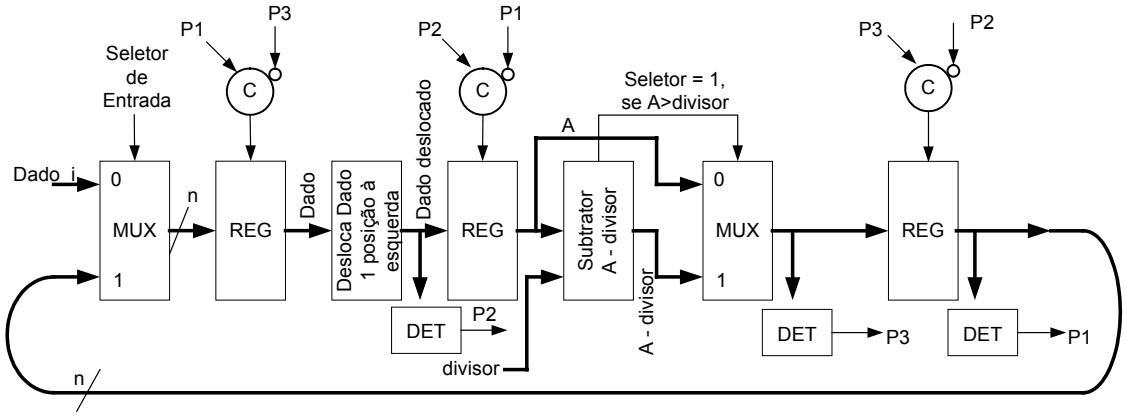



Figura 6.4: Esquema do circuito divisor de resto

### 6.6.2 Resultados

Novamente, teremos as tabelas 6.19, 6.20 e 6.21 mostrando os resultados para a síntese nas ferramentas dos fabricantes ALTERA, XILINX e ACTEL, respectivamente. O circuito divisor de resto utiliza um incremento que o diferencia do anterior, o circuito que realiza o deslocamento à esquerda, comumente chamado de *shift left*. Logo, possui um circuito que desloca de uma posição todos os bits do dado que lhe é amostrado e ainda um circuito que realiza a subtração dessa informação deslocada com o divisor. Como anteriormente, os resultados mostrados nessas tabelas vão privilegiar os estilos de projetos: somador do tipo comportamental com indicação forte do sinal, do tipo DIMS, do tipo NCL e ainda a derivação de circuito combinacional síncrono, chamado de derivação de lógica nas tabelas.

Ao final são mostradas as tabelas 6.22, 6.23 e 6.24 que representam os aumentos percentuais de elementos de programação em relação ao estilo de projeto que foi mais econômico. Ainda é utilizado como referência o percentual 0% de acréscimo. Os outros valores dentro de cada grupo de 4, 8, 16 e de 32 bits são valores percentuais acima desse.

Tabela 6.19: Valores de Células Lógicas para o Divisor de resto para Fabricante ALTERA – CPLD (Família MAX 7000AE) e FPGA (Família FLEX10KE)

| Divisor de resto | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de projeto | 4 bits | | 8bits | | 16bits | | 32bits | |
| | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD |
| | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) |
| Comportamental com indicação forte do sinal | 145 | 110 | 283 | 221 | 547 | 442 | 1134 | - |
| Por DIMS | 118 | 128 | 238 | 226 | 473 | 410 | 943 | - |
| Por NCL | 120 | 101 | 237 | 188 | 468 | 412 | 932 | - |
| Derivação de lógica | 137 | 100 | 267 | 172 | 531 | 337 | 1062 | - |

(*) *Macrocell* representa o número de macrocélulas para CPLDs.

(-) Não informado pela ferramenta.

Tabela 6.20: Valores de Células Lógicas para o Divisor de resto para Fabricante XILINX – CPLD (Família XC9500XV) e FPGA (Família SPARTAN2)

| Divisor de resto | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de projeto | 4 bits | | 8bits | | 16bits | | 32bits | |
| | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD |
| | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) |
| Comportamental com indicação forte do sinal | 220 | 79 | 434 | 154 | 809 | - | 1550 | - |
| Por DIMS | 186 | 79 | 322 | 155 | 631 | - | 1257 | - |
| Por NCL | 161 | 85 | 323 | 158 | 600 | - | - | - |
| Derivação de lógica | 184 | 63 | 327 | 112 | 644 | 232 | 1286 | - |

(*) *Macrocell* representa o número de macrocélulas para CPLDs.

(-) Não informado pela ferramenta.

Tabela 6.21: Valores de Células Lógicas para o Divisor de resto para Fabricante ACTEL – FPGA Antifusível (Família AXCELERATOR) e FPGA SRAM (Família 500K)

| Divisor de resto | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de projeto | 4 bits | | 8bits | | 16bits | | 32bits | |
| | FPGA/AF | FPGA/SR | FPGA/AF | FPGA/SR | FPGA/AF | FPGA/SR | FPGA/AF | FPGA/SR |
| | Número de células (LUTs) | Número de células (A) | Número de células (LUTs) | Número de células (A) | Número de células (LUTs) | Número de células (A) | Número de células (LUTs) | Número de células (A) |
| Comportamental com indicação forte do sinal | 149 | 208 | 310 | 429 | 726 | 848 | 1291 | 1689 |
| Por DIMS | 146 | 241 | 322 | 482 | 743 | 964 | 1303 | 1950 |
| Por NCL | 163 | 243 | 338 | 504 | 764 | 1007 | 1404 | 2060 |
| Derivação de lógica | 137 | 179 | 311 | 358 | 668 | 710 | 1412 | 1420 |

(A): A ferramenta informa os resultados através da descrição *Area*

De acordo com a tabela 6.22, verificamos que o circuito feito através de NCL teve novamente uma pequena vantagem com relação ao DIMS para o fabricante ALTERA em FPGAs mas menos eficiente para os seus circuitos feitos através em CPLDs. Para os circuitos feitos em CPLDs da ALTERA o estilo mais eficiente foi a derivação de lógica. Na tabela 6.23, para o fabricante XILINX, há uma disputa entre os circuitos com os estilos DIMS e NCL para os FPGAs e uma tendência de ser mais eficiente com derivação de lógica para os CPLDs. Já na tabela 6.24, no fabricante ACTEL, também existe uma disputa entre o estilo comportamental com indicação forte do sinal e derivação de lógica para os FPGAs e para os seus CPLDs (assim como o divisor inteiro).

Tabela 6.22: Valores em percentuais de acréscimos de elementos de programação para o divisor de resto para Fabricante ALTERA – CPLD (Família MAX 7000AE) e FPGA (Família FLEX10KE)

| Divisor de resto | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de projeto | FPGA | | | | CPLD | | | |
| | 4bits | 8bits | 16bits | 32bits | 4bits | 8bits | 16bits | 32bits |
| Comportamental com indicação forte do sinal | 22,88% | 19,41% | 16,88% | 21,67% | 10,00% | 28,49% | 31,16% | - |
| Por DIMS | 0% | 0,42% | 1,07% | 1,18% | 28,00% | 31,40% | 21,66% | - |
| Por NCL | 1,69% | 0% | 0% | 0% | 1,00% | 9,30% | 22,26% | - |
| Derivação de lógica | 16,10% | 12,66% | 13,46% | 13,95% | 0% | 0% | 0% | - |

(-) A comparação não poder ser feita

Tabela 6.23: Valores em percentuais de acréscimos de elementos de programação para o divisor de resto para Fabricante XILINX – CPLD (Família XC9500XV) e FPGA (Família SPARTAN2)

| Divisor de resto | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de projeto | FPGA | | | | CPLD | | | |
| | 4bits | 8bits | 16bits | 32bits | 4bits | 8bits | 16bits | 32bits |
| Comportamental com indicação forte do sinal | 36,65% | 34,78% | 34,83% | 23,31% | 25,40% | 37,50% | - | - |
| Por DIMS | 15,53% | 0% | 5,17% | 0% | 25,40% | 38,39% | - | - |
| Por NCL | 0% | 0,31% | 0% | - | 34,92% | 41,07% | - | - |
| Derivação de lógica | 14,29% | 1,55% | 7,33% | 2,31% | 0% | 0% | 0% | - |

(-) A comparação não poder ser feita

Tabela 6.24: Valores em percentuais de acréscimos de elementos de programação para o divisor de resto para Fabricante ACTEL – FPGA Antifusível (Família AXCELERATOR) e FPGA SRAM (Família 500K)

| Divisor de resto | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de projeto | FPGA Antifusível | | | | FPGA SRAM | | | |
| | 4bits | 8bits | 16bits | 32bits | 4bits | 8bits | 16bits | 32bits |
| Comportamental com indicação forte do sinal | 8,76% | 0% | 8,68% | 0% | 16,20% | 19,83% | 19,44% | 18,94% |
| Por DIMS | 6,57% | 3,87% | 11,23% | 0,93% | 34,64% | 34,64% | 35,77% | 37,32% |
| Por NCL | 18,98% | 9,03% | 14,37% | 8,75% | 35,75% | 40,78% | 41,83% | 45,07% |
| Derivação de lógica | 0% | 0,32% | 0% | 9,37% | 0% | 0% | 0% | 0% |

## 6.7 Mínimo Múltiplo Comum

### 6.7.1 Descrição

Esse circuito apresenta uma particularidade que o diferencia dos anteriores vistos até agora. Ele possui dois barramentos de informação no anel, como pode ser visto na figura 6.5. Os valores Ai e Bi são vetores de entrada, inicialmente inseridos no circuito através do MUX de entrada. Eles passam pelo primeiro registrador, depois pelo segundo e são levados a dois circuitos independentes. Um deles realizará a soma A+Ai, ou seja, o valor atual de A com o valor de Ai e o outro irá realizar a soma B+Bi, soma do valor atual de B com o valor de entrada Bi. As somas encontradas são levadas até um circuito que irá realizar a subtração da soma do valor A com a soma do valor B. Esse resultado irá gerar um carry out. se esse valor de carry out for 1, o circuito reconhecerá que o valor somado de A é maior que o valor somado de B. No entanto se for 0, o valor somado de A será menor que o valor somado de B. Com isso é possível utilizar o valor encontrado desse carry out como seletor ao MUX intermediário. Se SA é menor que SB, então ele irá selecionar os valores SA para ser o novo valor atual de A. Caso contrário, ele irá selecionar o valor de SB para substituir o valor de B anterior, assumindo seu novo valor atual. Após isso, os valores de A e B são levados até o terceiro registrador que os levará até uma das entradas do MUX de entrada. Nesse instante, o valor do seletor de entrada deverá estar em 1. Essa condição garante que os valores atualizados de A e B serão realimentados para dentro do anel, ocupando as entradas do primeiro registrador.

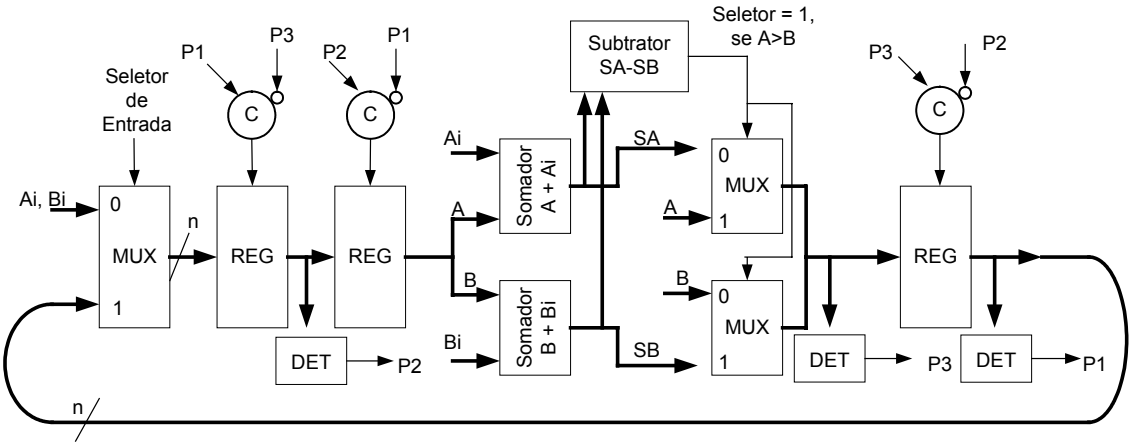



Figura 6.5: Esquema do circuito Mínimo Múltiplo Comum

### 6.7.2 Resultados

Nas tabelas 6.25, 6.26 e 6.27 estão os resultados para a síntese nas ferramentas dos fabricantes ALTERA, XILINX e ACTEL, respectivamente. O Mínimo Múltiplo Comum, comumente chamado de MMC, utiliza uma lógica bem mais complexa daquelas que foi vista até agora. Esse circuito utiliza dois somadores para os valores de A e B, separadamente e ainda utiliza um subtrator para gerar o carry out que será utilizado como seletor no MUX intermediário. Novamente, os resultados mostrados nessas tabelas deverão privilegiar os estilos de projetos dos circuitos combinacionais mostrados anteriormente. Ao final são apresentadas as tabelas 6.28, 6.29 e 6.30 que representam os aumentos percentuais de elementos de programação em relação ao estilo de projeto que foi mais econômico para cada fabricante. Ainda é utilizado como referência o percentual 0% de acréscimo. Os outros valores dentro de cada grupo de 4, 8, 16 e de 32 bits são valores percentuais acima desse valor de referência.

Tabela 6.25: Valores de Células Lógicas para o Mínimo Múltiplo Comum para Fabricante ALTERA – CPLD (Família MAX 7000AE) e FPGA (Família FLEX10KE)

| Mínimo Múltiplo Comum | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de projeto | 4 bits | | 8bits | | 16bits | | 32bits | |
| | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD |
| | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) |
| Comporta mental com indicação forte do sinal | 298 | 216 | 593 | 437 | 1204 | - | 2466 | - |
| Por DIMS | 260 | 258 | 534 | 463 | 1074 | - | 2156 | - |
| Por NCL | 248 | 180 | 494 | 373 | 979 | - | 1952 | - |
| Derivação de lógica | 282 | 164 | 573 | 323 | 1155 | - | 2318 | - |

(*) *Macrocell* representa o número de macrocélulas para CPLDs.

(-) Não informado pela ferramenta

Tabela 6.26: Valores de Células Lógicas para o Mínimo Múltiplo Comum para Fabricante XILINX – CPLD (Família XC9500XV) e FPGA (Família SPARTAN2)

| Mínimo Múltiplo Comum | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de projeto | 4 bits | | 8bits | | 16bits | | 32bits | |
| | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD |
| | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) |
| Comporta mental com indicação forte do sinal | 451 | 168 | 817 | - | 1736 | - | 3214 | - |
| Por DIMS | 390 | 204 | 708 | - | 1414 | - | 2831 | - |
| Por NCL | 344 | 183 | 672 | - | 1255 | - | 2587 | - |
| Derivação de lógica | 395 | 122 | 722 | 245 | 1452 | - | 2915 | - |

(*) *Macrocell* representa o número de macrocélulas para CPLDs.

(-) Não informado pela ferramenta.

Tabela 6.27: Valores de Células Lógicas para o Mínimo Múltiplo Comum para Fabricante ACTEL – FPGA Antifusível (Família AXCELERATOR) e FPGA SRAM (Família 500K)

| Mínimo Múltiplo Comum | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de projeto | 4 bits | | 8bits | | 16bits | | 32bits | |
| | FPGA/AF | FPGA/SR | FPGA/AF | FPGA/SR | FPGA/AF | FPGA/SR | FPGA/AF | FPGA/SR |
| | Número de células (LUTs) | Número de células (A) | Número de células (LUTs) | Número de células (A) | Número de células (LUTs) | Número de células (A) | Número de células (LUTs) | Número de células (A) |
| Comportamental com indicação forte do sinal | 353 | 457 | 674 | 937 | 1460 | 1918 | 2840 | 3814 |
| Por DIMS | 420 | 570 | 836 | 1171 | 1683 | 2403 | 3485 | 4848 |
| Por NCL | 375 | 537 | 755 | 1077 | 1569 | 2177 | 3093 | 4422 |
| Derivação de lógica | 336 | 378 | 687 | 768 | 1436 | 1545 | 3069 | 3099 |

(A): A ferramenta informa os resultados através da descrição *Área*

De acordo com a tabela 6.28, verificamos que o circuito feito através de NCL teve novamente vantagem com relação ao DIMS para o fabricante ALTERA em FPGAs mas menos eficiente para os seus circuitos feitos através de CPLDs. Para os circuitos feitos em CPLDs da ALTERA o estilo mais eficiente foi novamente a derivação de lógica. Com os resultados da tabela 6.29 do fabricante XILINX é possível ter uma definição naquela antiga disputa entre os circuitos com os estilos DIMS e NCL. Para os FPGAs o estilo que predominou em eficiência foi o NCL e para os CPLDs foi o estilo de derivação de lógica. Já na tabela 6.30, do fabricante ACTEL, também existe uma concorrência entre o comportamental com indicação forte do sinal e derivação de lógica para os FPGAs mas para os seus CPLDs, o estilo que leva mais vantagem é o de derivação de lógica (assim como o divisor inteiro e divisor de resto).

Tabela 6.28: Valores em percentuais de acréscimos de elementos de programação para o Mínimo Múltiplo Comum para Fabricante ALTERA – CPLD (Família MAX 7000AE) e FPGA (Família FLEX10KE)

| Mínimo Múltiplo Comum | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de projeto | FPGA | | | | CPLD | | | |
| | 4bits | 8bits | 16bits | 32bits | 4bits | 8bits | 16bits | 32bits |
| Comportamental com indicação forte do sinal | 20,16% | 20,04% | 22,98% | 26,33% | 31,71% | 35,29% | - | - |
| Por DIMS | 4,84% | 8,10% | 9,70% | 10,45% | 57,32% | 43,34% | - | - |
| Por NCL | 0% | 0% | 0% | 0% | 9,76% | 15,48% | - | - |
| Derivação de lógica | 13,71% | 15,99% | 17,98% | 18,75% | 0% | 0% | - | - |

(-) A comparação não poder ser feita

Tabela 6.29: Valores em percentuais de acréscimos de elementos de programação para o Mínimo Múltiplo Comum para Fabricante XILINX – CPLD (Família XC9500XV) e FPGA (Família SPARTAN2)

| Mínimo Múltiplo Comum | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de projeto | FPGA | | | | CPLD | | | |
| | 4bits | 8bits | 16bits | 32bits | 4bits | 8bits | 16bits | 32bits |
| Comportamental com indicação forte do sinal | 31,10% | 21,58% | 38,33% | 24,24% | 37,70% | - | - | - |
| Por DIMS | 13,37% | 5,36% | 12,67% | 9,43% | 67,21% | - | - | - |
| Por NCL | 0% | 0% | 0% | 0% | 50,00% | - | - | - |
| Derivação de lógica | 14,83% | 7,44% | 15,70% | 12,68% | 0% | 0% | - | - |

(-) A comparação não poder ser feita

Tabela 6.30: Valores em percentuais de acréscimos de elementos de programação para o Mínimo Múltiplo Comum para Fabricante ACTEL – FPGA Antifusível (Família AXCELERATOR) e FPGA SRAM (Família 500K)

| Mínimo Múltiplo Comum | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de projeto | FPGA Antifusível | | | | FPGA SRAM | | | |
| | 4bits | 8bits | 16bits | 32bits | 4bits | 8bits | 16bits | 32bits |
| Comportamental com indicação forte do sinal | 5,06% | 0% | 1,67% | 0% | 20,90% | 22,01% | 24,14% | 23,07% |
| Por DIMS | 25,00% | 24,04% | 17,20% | 22,71% | 50,79% | 52,47% | 55,53% | 56,94% |
| Por NCL | 11,61% | 12,02% | 9,26% | 8,91% | 42,06% | 40,23% | 40,91% | 42,69% |
| Derivação de lógica | 0% | 1,93% | 0% | 8,06% | 0% | 0% | 0% | 0% |

## 6.8 Máximo Divisor Comum

### 6.8.1 Descrição

Esse circuito recebe dois valores em suas entrada e se encarrega de calcular o mínimo múltiplo comum dos dois. Em cada ciclo do anel, mostrado na figura 6.6, ele verifica qual das duas entradas possui valor maior e subtrai o valor menor desta. No próximo ciclo o circuito irá atualizar os valores de entrada, substituindo o valor anterior por aquele que foi subtraído. No próximo ciclo, irá verificar novamente os valores atualizados subtraindo o maior valor pelo menor valor. Esse procedimento será realizado até que os dois valores sejam iguais, situação tal que o circuito armazenará esses valores e os colocará em sua saída. O Máximo Divisor Comum é formado por dois somadores que realizam subtração, independentes um do outro. Esse circuito segue a mesma configuração do MMC sendo que não utiliza um circuito extra para calcular a subtração dos valores A e B.

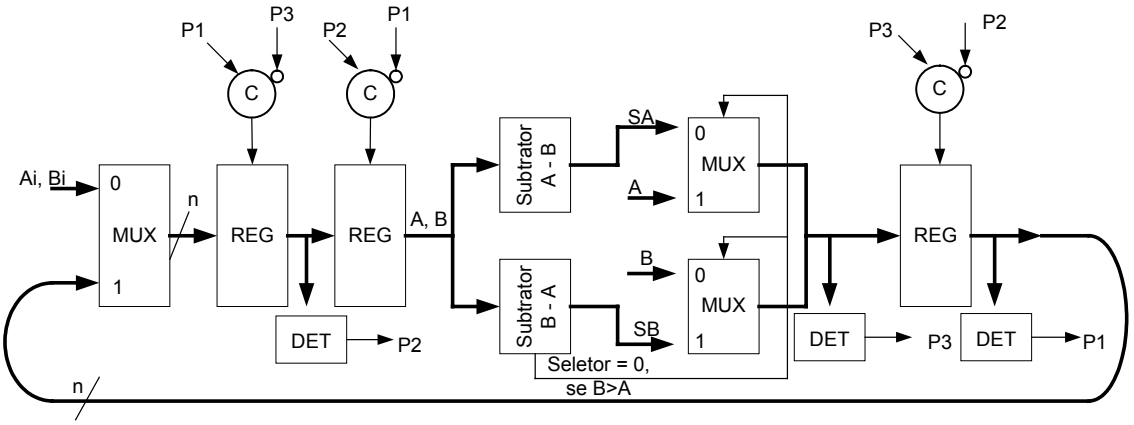



Figura 6.6: Esquema do circuito Máximo Divisor Comum

### 6.8.2 Resultados

Nas tabelas 6.31, 6.32 e 6.33 estão os resultados para a síntese nas ferramentas dos fabricantes ALTERA, XILINX e ACTEL, respectivamente. O Máximo Divisor Comum, comumente chamado de MDC, utiliza uma arquitetura similar ao circuito MMC. Esse circuito utiliza dois somadores subtratores para calcular os valores de A-B e B-A, separadamente. Não é necessário utilizar um subtrator especial para gerar o carry out, uma vez que esse valor já pode ser gerado por um dos subtratores que o próprio circuito já utiliza. Esse valor de carry out, oriundo do subtrator B-A, será levado ao MUX intermediário sob a forma de seu seletor de entradas. Novamente, os resultados mostrados nessas tabelas deverão privilegiar os estilos de projetos mostrados anteriormente. Ao final são apresentadas as tabelas 6.34, 6.35 e 6.36 que representam os aumentos percentuais de elementos de programação em relação ao estilo de projeto que foi mais econômico para cada fabricante. A referência é representada pelo percentual 0% de acréscimo. Os outros valores dentro de cada grupo de 4, 8, 16 e de 32 bits são valores percentuais acima desse valor de referência adotado.

Tabela 6.31: Valores de Células Lógicas para o Máximo Divisor Comum para Fabricante ALTERA – CPLD (Família MAX 7000AE) e FPGA (Família FLEX10KE)

| Máximo Divisor Comum | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de projeto | 4 bits | | 8bits | | 16bits | | 32bits | |
| | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD |
| | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) |
| Comportamental com indicação forte do sinal | 237 | 170 | 471 | 357 | 941 | - | 1891 | - |
| Por DIMS | 216 | 180 | 432 | 358 | 861 | - | 1720 | - |
| Por NCL | 218 | 162 | 433 | 334 | 852 | - | 1695 | - |
| Derivação de lógica | 236 | 146 | 488 | 294 | 991 | - | 2000 | - |

(*) *Macrocell* representa o número de macrocélulas para CPLDs.

(-) Não informado pela ferramenta.

Tabela 6.32: Valores de Células Lógicas para o Máximo Divisor Comum para Fabricante XILINX – CPLD (Família XC9500XV) e FPGA (Família SPARTAN2)

| Máximo Divisor Comum | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de projeto | 4 bits | | 8bits | | 16bits | | 32bits | |
| | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD |
| | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) |
| Comporta mental com indicação forte do sinal | 335 | 136 | 596 | - | 1137 | - | 2358 | - |
| Por DIMS | 289 | 167 | 549 | - | 1095 | - | 2326 | - |
| Por NCL | 262 | 152 | 508 | - | 984 | - | 2004 | - |
| Derivação de lógica | 278 | 112 | 551 | 228 | 1104 | - | 2343 | - |

(*) *Macrocell* representa o número de macrocélulas para CPLDs.

(-) Não informado pela ferramenta.

Tabela 6.33: Valores de Células Lógicas para o Máximo Divisor Comum para Fabricante ACTEL – FPGA Antifusível (Família AXCELERATOR) e FPGA SRAM (Família 500K)

| Máximo Divisor Comum | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de projeto | 4 bits | | 8bits | | 16bits | | 32bits | |
| | FPGA/ AF | FPGA/SR | FPGA/AF | FPGA/SR | FPGA/AF | FPGA/SR | FPGA/AF | FPGA/SR |
| | Número de células (LUTs) | Número de células (A) | Número de células (LUTs) | Número de células (A) | Número de células (LUTs) | Número de células (A) | Número de células (LUTs) | Número de células (A) |
| Comportamental com indicação forte do sinal | 236 | 345 | 488 | 700 | 991 | 1410 | 2144 | 2831 |
| Por DIMS | 299 | 436 | 616 | 875 | 1233 | 1755 | 2543 | 3538 |
| Por NCL | 330 | 443 | 644 | 888 | 1312 | 1756 | 2805 | 3499 |
| Derivação de lógica | 255 | 319 | 544 | 642 | 1143 | 1297 | 2442 | 2582 |

(A): A ferramenta informa os resultados através da descrição *Área*

De acordo com a tabela 6.31, verificamos que o circuito feito através de DIMS voltou a concorrer com o NCL para o fabricante ALTERA em FPGAs mas menos eficiente para os seus circuitos feitos através de CPLDs. Para os circuitos feitos em CPLDs da ALTERA o estilo mais eficiente foi novamente a derivação de lógica. Para a tabela 6.32 do fabricante XILINX, tem-se uma definição naquela antiga disputa entre os circuitos com os estilos DIMS e NCL. Para os FPGAs o estilo que predominou em eficiência foi o NCL e para os CPLDs foi o estilo de derivação de lógica. Já na tabela 6.33 do fabricante ACTEL o estilo que predominou em FPGA foi comportamental com indicação forte do sinal e para CPLD foi o estilo por derivação de lógica.

Tabela 6.34: Valores em percentuais de acréscimos de elementos de programação para o Máximo Divisor Comum para Fabricante ALTERA – CPLD (Família MAX 7000AE) e FPGA (Família FLEX10KE)

| Máximo Divisor Comum | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de projeto | FPGA | | | | CPLD | | | |
| | 4bits | 8bits | 16bits | 32bits | 4bits | 8bits | 16bits | 32bits |
| Comportamental com indicação forte do sinal | 9,72% | 9,03% | 10,45% | 11,56% | 16,44% | 21,43% | - | - |
| Por DIMS | 0% | 0% | 1,06% | 1,47% | 23,29% | 21,77% | - | - |
| Por NCL | 0,93% | 0,23% | 0% | 0% | 10,96% | 13,61% | - | - |
| Derivação de lógica | 9,26% | 12,96% | 16,31% | 17,99% | 0% | 0% | - | - |

(-) A comparação não poder ser feita

Tabela 6.35: Valores em percentuais de acréscimos de elementos de programação para o Máximo Divisor Comum para Fabricante XILINX – CPLD (Família XC9500XV) e FPGA (Família SPARTAN2)

| Máximo Divisor Comum | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de projeto | FPGA | | | | CPLD | | | |
| | 4bits | 8bits | 16bits | 32bits | 4bits | 8bits | 16bits | 32bits |
| Comportamental com indicação forte do sinal | 27,86% | 17,32% | 15,55% | 17,66% | 21,43% | - | - | - |
| Por DIMS | 10,31% | 8,07% | 11,28% | 16,07% | 49,11% | - | - | - |
| Por NCL | 0% | 0% | 0% | 0% | 35,71% | - | - | - |
| Derivação de lógica | 6,11% | 8,46% | 12,20% | 16,92% | 0% | 0% | - | - |

(-) A comparação não poder ser feita

Tabela 6.36: Valores em percentuais de acréscimos de elementos de programação para o Máximo Divisor Comum para Fabricante ACTEL – FPGA Antifusível (Família AXCELERATOR) e FPGA SRAM (Família 500K)

| Máximo Divisor Comum | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de projeto | FPGA Antifusível | | | | FPGA SRAM | | | |
| | 4bits | 8bits | 16bits | 32bits | 4bits | 8bits | 16bits | 32bits |
| Comportamental com indicação forte do sinal | 0% | 0% | 0% | 0% | 8,15% | 9,03% | 8,71% | 9,64% |
| Por DIMS | 26,69% | 26,23% | 24,42% | 18,61% | 36,68% | 36,29% | 35,31% | 37,03% |
| Por NCL | 39,83% | 31,97% | 32,39% | 30,83% | 38,87% | 38,32% | 35,91% | 35,52% |
| Derivação de lógica | 8,05% | 11,48% | 15,34% | 13,90% | 0% | 0% | 0% | 0% |

## 6.9 Raiz Quadrada

### 6.9.1 Descrição

Esse circuito calcula a raiz quadrada de um valor que lhe é inserido na entrada, mostrado na figura 6.7. Os parâmetros R=1 e S=4 são inseridos no MUX de entrada e este entrega esses valores para o primeiro registrador do anel. Passando pelo segundo registrador, o valor de R é inserido num somador que realizará a soma+1 ao valor de R. Depois disso, os dois valores, S e R+1 são levados até o terceiro registrador. Após atravessarem o terceiro registrador, o dado R será levado para um circuito que realizará o deslocamento de uma posição à esquerda de sua representação binária, ou seja, realizará o produto de R por dois. Esse novo valor, chamado de D será levado ao somador que realizará a soma S+D. Posteriormente, o resultado S+D será levado a um outro somador que fará a soma+1 desse valor. Assim quando o valor de S atualizado for enviado para as entradas do MUX de entrada, terá seu valor equivalente a S+D+1. Após isso, os valores atualizados de R e S são levados ao MUX de entrada e realimentados no primeiro registrador.

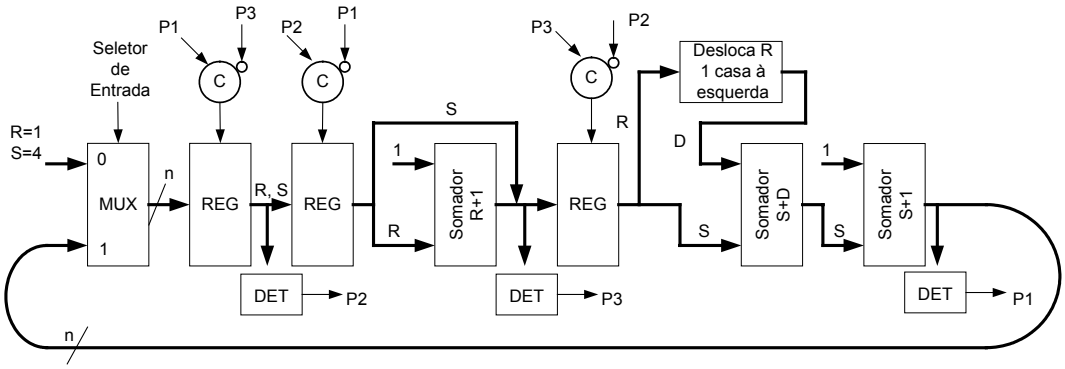



Figura 6.7: Esquema do circuito raiz quadrada

### 6.9.2 Resultados

Nas tabelas 6.37, 6.38 e 6.39 estão os resultados para a síntese nas ferramentas dos fabricantes ALTERA, XILINX e ACTEL, respectivamente. O circuito que calcula a raiz quadrada inteira utiliza um conjunto de somadores em série que o classifica como o mais complexo dentre os circuitos anteriores. Esse circuito utiliza dois soma+1, um somador de S+D e um circuito que desloca de uma posição à esquerda o valor de R. Novamente, os resultados mostrados nessas tabelas deverão privilegiar os estilos de projetos mostrados anteriormente. Ao final são apresentadas as tabelas 6.40, 6.41 e 6.42 que representa os aumentos percentuais de elementos de programação em relação ao estilo de projeto que foi mais econômico em cada fabricante. A referência é representada pelo percentual 0% de acréscimo. Os outros valores dentro de cada grupo de 4, 8, 16 e de 32 bits são valores percentuais acima desse valor de referência.

Tabela 6.37: Valores de Células Lógicas para a raiz quadrada para Fabricante ALTERA – CPLD (Família MAX 7000AE) e FPGA (Família FLEX10KE)

| Raiz quadrada | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de projeto | 4 bits | | 8bits | | 16bits | | 32bits | |
| | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD |
| | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) |
| Comporta mental com indicação forte do sinal | 203 | 180 | 434 | 362 | 939 | - | 1874 | - |
| Por DIMS | 162 | 218 | 344 | 363 | 701 | - | 1404 | - |
| Por NCL | 192 | 187 | 388 | 348 | 773 | - | 1544 | - |
| Derivação de lógica | 200 | 150 | 440 | 272 | 883 | - | 1828 | - |

(*) *Macrocell* representa o número de macrocélulas para CPLDs.

(-) Não informado pela ferramenta

Tabela 6.38: Valores de Células Lógicas para a raiz quadrada para Fabricante XILINX – CPLD (Família XC9500XV) e FPGA (Família SPARTAN2)

| Raiz quadrada | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de projeto | 4 bits | | 8bits | | 16bits | | 32bits | |
| | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD |
| | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) | Número de células (LUTs) | Número de macrocell (*) |
| Comportamental com indicação forte do sinal | 277 | 104 | 582 | - | 1241 | - | 2510 | - |
| Por DIMS | 320 | 168 | 605 | - | 1210 | - | 2418 | - |
| Por NCL | 268 | 161 | 521 | - | 1054 | - | - | - |
| Derivação de lógica | 294 | 177 | 571 | 228 | 1150 | - | 2310 | - |

(*) *Macrocell* representa o número de macrocélulas para CPLDs.

(-) Não informado pela ferramenta

Tabela 6.39: Valores de Células Lógicas para a raiz quadrada para Fabricante ACTEL – FPGA Antifusível (Família AXCELERATOR) e FPGA SRAM (Família 500K)

| Raiz quadrada | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de projeto | 4 bits | | 8bits | | 16bits | | 32bits | |
| | FPGA/AF | FPGA/SR | FPGA/AF | FPGA/SR | FPGA/AF | FPGA/SR | FPGA/AF | FPGA/SR |
| | Número de células (LUTs) | Número de células (A) | Número de células (LUTs) | Número de células (A) | Número de células (LUTs) | Número de células (A) | Número de células (LUTs) | Número de células (A) |
| Comportamental com indicação forte do sinal | 234 | 326 | 568 | 774 | 1253 | 1542 | 2765 | 3365 |
| Por DIMS | 318 | 446 | 695 | 917 | 1396 | 1860 | 2933 | 3720 |
| Por NCL | 362 | 457 | 756 | 954 | 1529 | 1944 | 3181 | 3977 |
| Derivação de lógica | 243 | 298 | 506 | 615 | 1102 | 1277 | 2200 | 2524 |

(A): A ferramenta informa os resultados através da descrição *Área*

De acordo com a tabela 6.37, verificamos que o circuito feito através do estilo DIMS venceu de seu concorrente NCL para o fabricante ALTERA em FPGAs mas menos eficiente para os seus circuitos feitos através de CPLDs. Para os circuitos feitos em CPLDs da ALTERA o estilo mais eficiente foi novamente a derivação de lógica (por sinal, em todos os circuitos visto nesse trabalho, a derivação de lógica foi mais eficiente para o fabricante ALTERA em CPLDs). Para a tabela 6.38 do fabricante XILINX tem-se o estilo NCL predominando frente ao concorrente DIMS para os FPGAs. Para os CPLDs não se pode afirmar muita coisa uma vez que a ferramenta deixou de fornecer vários resultados para uma análise mais aprofundada. Já na tabela 6.39 do fabricante ACTEL, o estilo que teve vantagem em FPGAs e para os CPLDs foi o de derivação de lógica.

Tabela 6.40: Valores em percentuais de acréscimos de elementos de programação para a raiz quadrada para Fabricante ALTERA – CPLD (Família MAX 7000AE) e FPGA (Família FLEX10KE)

| Raiz quadrada | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de projeto | FPGA | | | | CPLD | | | |
| | 4bits | 8bits | 16bits | 32bits | 4bits | 8bits | 16bits | 32bits |
| Comportamental com indicação forte do sinal | 25,31% | 26,16% | 33,95% | 33,48% | 20,00% | 33,09% | - | - |
| Por DIMS | 0% | 0% | 0% | 0% | 45,33% | 33,46% | - | - |
| Por NCL | 18,52% | 12,79% | 10,27% | 9,97% | 24,67% | 27,94% | - | - |
| Derivação de lógica | 23,46% | 27,91% | 25,96% | 30,20% | 0% | 0% | - | - |

(-) A comparação não poder ser feita

Tabela 6.41: Valores em percentuais de acréscimos de elementos de programação para a raiz quadrada para Fabricante XILINX – CPLD (Família XC9500XV) e FPGA (Família SPARTAN2)

| Raiz quadrada | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de projeto | FPGA | | | | CPLD | | | |
| | 4bits | 8bits | 16bits | 32bits | 4bits | 8bits | 16bits | 32bits |
| Comportamental com indicação forte do sinal | 3,36% | 11,71% | 17,74% | 8,66% | 0% | - | - | - |
| Por DIMS | 19,40% | 16,12% | 14,80% | 4,68% | 61,54% | - | - | - |
| Por NCL | 0% | 0% | 0% | - | 54,81% | - | - | - |
| Derivação de lógica | 9,70% | 9,60% | 9,11% | 0% | 70,19% | 0% | - | - |

(-) A comparação não poder ser feita

Tabela 6.42 Valores em percentuais de acréscimos de elementos de programação para a raiz quadrada para Fabricante ACTEL – FPGA Antifusível (Família AXCELERATOR) e FPGA SRAM (Família 500K)

| Raiz quadrada | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de projeto | FPGA Antifusível | | | | FPGA SRAM | | | |
| | 4bits | 8bits | 16bits | 32bits | 4bits | 8bits | 16bits | 32bits |
| Comportamental com indicação forte do sinal | 0% | 12,25% | 13,70% | 25,68% | 9,40% | 25,85% | 20,75% | 33,32% |
| Por DIMS | 35,90% | 37,35% | 26,68% | 33,32% | 49,66% | 49,11% | 45,65% | 47,39% |
| Por NCL | 54,70% | 49,41% | 38,75% | 44,59% | 53,36% | 55,12% | 52,23% | 57,57% |
| Derivação de lógica | 3,85% | 0% | 0% | 0% | 0% | 0% | 0% | 0% |

## 6.10 Análise de desempenho temporal

Nesse trabalho foram simulados dois circuitos para poder comparar o parâmetro *timming* com os circuitos síncronos similares. Os circuitos escolhidos foram contador e raiz quadrada. Na tabela 6.43 é possível verificar os diferentes estilos de projetos mostrados anteriormente para o circuito contador e na tabela 6.44 os diferentes estilos para o circuito raiz quadrada. No circuito contador foram feitas duas simulações para avaliação do atraso. A primeira, o incremento unitário até o valor 100 e a segunda, incremento unitário de 128 até 200.Para o circuito raiz quadrada, também foram feitas duas simulações para a avaliação do desempenho temporal. Na primeira ele executa o calcula da raiz quadrada inteira de 260 e na segunda, o cálculo da raiz quadrada inteira de 950.

Tabela 6.43: Valores de timming para circuito contador para Fabricante ALTERA – CPLD (Família MAX 7000AE) e FPGA (Família FLEX10KE)

| Contador | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de projeto para os somadores | 16 bits | | | | 32 bits | | | |
| | Para contagem de 1 a 100 | | Para contagem de 128 a 200 | | Para contagem de 1 a 100 | | Para contagem de 128 a 200 | |
| | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD |
| Comportamental com indicação forte do sinal | 12,73us | Erro ao simular | 9,23us | Erro ao simular | 18,09us | - | 13,11us | - |
| Por DIMS | 15,33us | - | 11,16us | - | 39,30us | - | 22,04us | - |
| Por NCL | 9,63us | Excede nº de células | 7,09us | Excede nº de células | 14,31,us | - | 10,53us | - |
| Derivação de lógica | 23,06us | 60,59us | 16,79us | 44,06us | 39,89us | - | 29,01us | - |
| Circuito Síncrono | 1,13us T=11.3ns | 1,42us T=17.0ns | 815,19ns T=11.3ns | 1,25us T=17.0ns | 1,58us T=15.8ns | 1,79us T=17.8ns | 1,16us T=15.8ns | 1,31us T=17.8ns |

Tabela 6.44: Valores de timming para circuito raiz quadrada para Fabricante ALTERA – CPLD (Família MAX 7000AE) e FPGA (Família FLEX10KE)

| Raiz Quadrada | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Estilos de projeto para os somadores | 16 bits | | | | 32 bits | | | |
| | Para calcula a raiz de 260 | | Para calcula a raiz de 950 | | Para calcula a raiz de 260 | | Para calcula a raiz de 950 | |
| | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD | FPGA | CPLD |
| Comportamental com indicação forte do sinal | 3,98us | - | 7,81us | - | 5,83us | - | Não simulou | - |
| Por DIMS | 4,51us | - | 8,91us | - | 7,76us | - | 15,34us | - |
| Por NCL | 2,74us | - | 5,34us | - | 3,25us | - | 6,34us | - |
| Derivação de lógica | 6,09us | - | 12,06us | - | 10,31us | - | 20,37us | |
| Circuito Síncrono | 193,03ns T=12.1ns | 637,90ns T=39.2ns | 365,71ns T=12.1ns | 1,19us T=39.2ns | 263,97ns T=16.5ns | 852,55ns T=52.6ns | 494,87ns T=16.5ns | 1,59us T=52.6ns |

Dos estilos de projetos descritos anteriormente é possível avaliar que NCL obteve um melhor desempenho comparativamente aos demais circuitos assíncronos. No entanto, comparado ao similar síncrono, esse mesmo estilo obteve desvantagem expressiva. Para o circuito contador, essa desvantagem chega a 907% da performance já no circuito para o cálculo da raiz quadrada, chega a 1463%.

## 6.11 Sumário

Foi feita uma avaliação particular para cada circuito utilizado nessa dissertação com uma descrição breve do tipo de função que realiza e a apresentação dos resultados obtidos para cada fabricante das ferramentas. As comparações foram feitas com relação aos elementos de programação de circuitos de 4, 8, 16 e 32 bits e entre as tecnologias de PLDs existentes no mercado, como CPLDs e FPGAs. Através dos resultados encontrados são obtidas algumas conclusões e direcionamentos para futuros projetos, utilizando as ferramentas de síntese comerciais para os circuitos assíncronos.

# 7 CONCLUSÃO

Neste trabalho foram mostrados alguns estilos de projetos para circuitos assíncronos e uma avaliação de ferramentas comerciais quando utilizadas para síntese desses circuitos. Foi apresentada, juntamente com a introdução da teoria assíncrona, uma avaliação da performance dos circuitos assíncronos quando comparados com seus similares síncronos.

Foi feita uma avaliação em três ferramentas comerciais, dos fabricantes ALTERA, XILINX e ACTEL, quanto a quantidade de elementos de programação, tanto em CPLDs quanto para FPGAs. Foi feita também, uma avaliação de dois circuitos em anel assíncrono, o contador e o cálculo da raiz quadrada, para se verificar o desempenho dos circuitos assíncronos quando comparados aos circuitos síncronos similares, utilizando essas ferramentas de síntese.

Os estilos de projetos, como foram mostrados nos capítulos 3 e 4, utilizam técnicas hazard-free, preservando a funcionalidade dos mesmos frente às exigências dos circuitos assíncronos. Foram avaliados os circuitos combinacionais do tipo somadores *full adder* quanto ao número de elementos de programação e atraso em suas portas, utilizando esses mesmos estilos de projeto para os circuitos assíncronos. Juntamente com esses resultados foram obtidos os resultados para os RCAs correspondentes a esses somadores.

O principal mérito deste trabalho foi avaliar os estilos de projeto de circuitos assíncronos em PLDs frente as ferramentas de síntese lógica, comercialmente usadas, para os circuitos síncronos. Essa avaliação de baseou no quantitativo de elementos de programação utilizados por essas ferramentas para construir esses circuitos e na própria funcionalidade da mesma. A funcionalidade pode ser avaliada pela consistência dos resultados obtidos, uma vez que, para alguns casos dos circuitos avaliados, determinadas ferramentas não concluíram a simulação ou não informaram valores para atrasos ou quantitativo de elementos de programação. Neste trabalho existe uma grande contribuição didática quanto ao conhecimento do funcionamento dos circuitos assíncronos. Detalhes com relação às diferenças de performance entre síncronos e assíncronos são tratadas e as próprias diferenças entre os estilos são abordadas nesse trabalho.

É importante lembrar que essa comparação não pode ser medida diretamente, avaliando simplesmente o resultado final e o tempo gasto para obtê-lo. Deve ser avaliado o conjunto do circuito, as formas de projetos e os atrasos nas informações no protocolo de controle. Os atrasos nos cálculos dos circuitos síncronos também são diferentes dos assíncronos. Nos síncronos é tratado o pior atraso, para determinar o

período mínimo do sinal do relógio. Já nos assíncronos é tratado o atraso médio que o cálculo deverá possuir.

É importante verificar que houve grande desperdício de área nos projetos dos circuitos assíncronos. Conforme foi mostrado no capítulo 5, quando se passa da codificação *single rail* para *dual rail*, o número de saídas dobra (pelo sinal de resposta e o seu correspondente sinal negado). Além, é claro, que o suporte dobra pela própria limitação da arquitetura PLD empregada. Resultando num aumento de quatro vezes, no mínimo.

Foi visto o custo adicional de se ter um registrador controlado pelo relógio central na célula lógica padrão das famílias desses fabricantes. Este elemento controlado pelo relógio central simplesmente não é utilizado na metodologia assíncrona pois não existe a influência do sinal deste relógio.

Os atrasos nesses circuitos resultantes são claros uma vez que consumindo mais LUTs no roteamento, o tempo gasto para que o sinal chegue nas saídas desse novo circuito *dual rail* é maior que nos circuitos *single rail*

A partir deste trabalho e do que foi discutido, podemos tirar algumas conclusões:

- Apesar da técnica DIMS apresentar resultados interessantes quanto ao custo de elementos de programação não teve um bom aproveitamento nas simulações quanto ao cálculo do atraso para o circuito contador e raiz quadrada. Uma possível explicação é que circuitos NCL possuem uma resposta mais rápido que os do tipo DIMS e que algumas ferramentas otimizam essa resposta mesmo que existe uma detecção prévia dos sinais de fim de cálculo no circuito somador.

- Que apesar da facilidade de construção e da regularidade dos circuitos formados a partir da técnica DIMS, os custos desta família em termos de número de elementos de programação são muito altos.

- Todas as ferramentas tiveram resultados para FPGAs mas para CPLDs algumas falharam. Inclusive quanto ao número máximo de macrocélulas que a família suporta. Foi verificado que alguns circuitos não foram gerados, alguns dos quais ainda não haviam excedido esse limite.

- É possível extrair dessas afirmações que existe a necessidade de se construir um código próprio para os elementos de base como as células Muller e os elementos M de N, que originaram os circuitos NCLs. Isso se explica, primeiramente, que para os elementos células Muller, o custo por célula pode ser otimizado caso a ferramenta possua uma biblioteca própria desse elemento e para os elementos M de N, que uma biblioteca da ferramenta já deveria prever qualquer otimização de fim de cálculo que possa ser feita de forma independente pela ferramenta de síntese.

Os trabalhos futuros nessa área poderão desenvolver as seguintes possibilidades:

1) Substitutos para a LUT:

Uma LUT própria para dual rail. Um dos mais críticos excessos de área nos circuitos assíncronos é a duplicação da lógica para implementações dual rail. O objetivo é que as entradas dobradas sejam tratadas dentro de uma única LUT dual rail. Isto é possível se forem feitas as seguintes suposições para esta LUT especial:

* ter quatro entradas dual rail;

* ter duas saídas dual rail;

* todas as saídas sejam resetadas se todas as entradas dual rail forem resetadas;

* Uma das saídas é disponibilizada quando todas as entradas dual rail possuem valores válidos;

* Se um dado válido não está presente em uma das entradas as saídas não precisam produzir dados válidos;

* A auto-temporização é garantida pela utilização de células Muller extras

2) Uma LUT M de N: Utilizando uma lógica programável para implementar células M de N, através da lógica NCL.

3) Substituir o flip-flop não utilizado em cada célula lógica:

Em cada célula lógica existem flips-flops que nunca são utilizados na lógica assíncrona. Uma possibilidade seria substituí-los por elementos C ou latches assíncronos. A melhor opção poderia ser a combinação dos dois (por exemplo, 50% dos elementos lógicos seriam elementos C de Muller e os outros 50% poderiam ser latches assíncronos). Os elementos C são úteis por implementar o controle distribuído bem como na indicação forte dos sinais de reset e fim-de-cálculo e os latches assíncronos são úteis por armazenar os elementos nos pipelines.

# REFERÊNCIAS

ACTEL. Disponível em: <www.actel.com/products/tools/sw.html>. Acesso em: mar 2003.

ALTERA. Disponível em: <http://www.altera.com/products/software/>. Acesso em: mar 2004

BROWN, S.; VRANESIC, Z. G. **Fundamentals of Digital Logic with VHDL Design**. 2nd ed. Boston: McGraw-Hill, 2005.

BURNS, S. et al. An FPGA for implementing asynchronous circuits. **IEEE Desing and Test of Computers.** [S.l.], v. 11, n. 3, p. 60-69, Autumn, 1994.

BRUNVAND, E.; MICHELL, N.; SMITH, K. A Comparison of Self-Timed Design using FPGA, CMOS and GaAs Technologies. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTER DESIGN: VLSI IN COMPUTERS AND PROCESSORS, ICCD, 1992, **Proceedings…** Los Alamitos: IEEE, 1992. p. 76-80.

DERTOUZOS, M. L. **Threshold Logic:** a Synthesis Approach. [S.l.]: The MIT Press, 1965. 256 p.

FANG, D.; TEIFEL, J.; MANOBAR, R. A High-Performance Asynchronous FPGA: Test Results. In: IEEE SYMPOSIUM ON FIELD-PROGRAMMABLE CUSTOM COMPUTING MACHINES, FCCM, 13., 2005. **Proceedings...** [S.l.]: IEEE, 2005. p. 271-272.

FANT, K.M.; BRANDT, S. A. NULL Convention Logic: a complete and consistent logic for asynchronous digital circuit synthesis. In: INTERNATIONAL CONFERENCE ON APPLICATION SPECIFIC SYSTEMS, ARCHITECTURES AND PROCESSORS, ASAP, 1996. **Proceedings…** [S.l.]: IEEE, 1996. p. 261-273

HAMBLEM, J. O.; FURMAN, M. D. **Rapid Prototyping of Digital Systems**. London: Kluwer Academic, 2000.

HAUCK, S. Asynchronous Design Methodologies: An Overview. **Proceedings of the IEEE**, [S.l.], v. 83, n 1, p. 69-93, Jan. 1995.

HUOT, N. et al. FPGA architecture for multiple-style asynchronous logic. In: IEEE AUTOMATION AND TEST IN EUROPE CONFERENCE AND EXHIBITION, DATE, 2005. **Proceedings...** [S.l.]:IEEE, 2005. p. 32-33.

KUANG, W. et al. Performance analysis and optimisation of NCL self-timed rings. **IEE Proc.–Circuits Devices Syst.**, [S.l.], v. 150, n. 3, June 2003.

MEEKINS, K.; FERGUSON, D.; BASTA, M. Delay Insensitive NCL reconfigurable Logic. In: IEEE AEROSPACE CONFERENCE, 2002. **Proceedings...**[S.l.]: IEEE, 2002, v. 4, p. 1961-1967.

NOWICK, S. M. et al. Scanning the Technology – Applications of Asynchronous Circuits. **Proceedings of the IEEE**, [S.l.], v. 87, n. 2, p. 219-222, Feb. 1999.

NOVAK, J. H.; BRUNVAND, E. Using FPGAs to Prototype a Self-timed Floating Point Co-Processor. In: IEEE CUSTOM INTEGRATED CIRCUITS CONFERENCE, CICC, 1994. **Proceedings…**[S.l.; s.n.], 1994. p. 85-88.

ORTEGA-CISNEROS, S. et al. Rapid prototyping of a self-timed ALU with FPGAs. In: IEEE INTERNATIONAL CONFERENCE ON RECONFIGURABLE COMPUTING AND FPGAS, ReConFig, 2005. **Proceedings…**[S.l.; s.n.], 2005. p. 7-14.

PAYNE, R. Asynchronous FPGA Architectures. **IEE Proc.-Comput. Digit. Tech.**, [S.l.], v. 143, n. 5, p. 282-286, Sept. 1996.

RIGAUD, J. B. et al. Implementing Asynchronous Circuits on LUT Based FPGAs. In: INTERNATIONAL CONFERENCE ON FIELD-PROGRAMMABLE LOGIC AND APPLICATIONS, FPL, 2002. **Field Programmable Logic and Applications**: Proceedings. Berlin: Springer-Verlag, 2002. p. 36-46. (Lectore Notes in Computer Science, 2438).

SPARSO, J.; FURBER, S. **Principles of Asynchronous Circuit Design – A Systems Perspective**. [S.l.]: Kluwer Academic, 2001.

TEIFEL, J.; MANOBAR, R. An Asynchronous Dataflow FPGA Architecture. **Proceedings of the IEEE**, [S.l.], v. 53, n. 11, p. 1376-1392, Nov. 2004.

TODMAN, T. J. et al. Reconfigurable computing architectures and design methods. **IEE Proc.-Comput. Digit. Tech.**, [S.l.], v. 152, n. 2, p. 193-207, Mar. 2005.

WILLIAMS, T. E. Performance of Iterative Computation in Self-Timed Rings. **Journal of VLSI Processing**, Boston, v. 7, n. 1/2, p. 17-31, Feb. 1994.

XILINX. Disponível em <www.xilinx.com/products/platform/>. Acesso em: ago. 2003.

ZAFAR, Y.; AHMED, M. A. Novel FPGA Compliant Micropipeline. **IEEE Transactions on Circuits and Systems-II** , [S.l.], v. 52, n. 9, p. 611-615, Sept. 2005.

# APÊNDICE A CÓDIGO VHDL PARA CELULA DE MULLER DE 2 ENTRADAS

Código em VHDL para célula de Muller de 2 entradas. Descrição comportamental e estrutural.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY celulac_comportamental IS
PORT (      a, b        :          IN std_logic;
            c           :          OUT std_logic);
END celulac_comp;

ARCHITECTURE instanciada OF celulac_comportamental IS
SIGNAL ctemp: std_logic;
BEGIN
PROCESS (a, b)
begin
IF (a=b) THEN
ctemp <= a;
ELSE
ctemp <= ctemp;
END IF;
c <= ctemp;
END PROCESS;

END instanciada;
```

(a) Código descrito em VHDL para célula de Muller de 2 entradas comportamental

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY celulac_estrutural IS
PORT (      a, b    :          IN std_logic;
            c       :          OUT std_logic);
END celulacestrutural;

ARCHITECTURE instanciada OF celulac_estrutural IS
SIGNAL ctemp: std_logic;

COMPONENT modcc
PORT (      a, b, cin       : IN std_logic;
            cout            : OUT std_logic);
END component;

BEGIN

INSTANCIACC: modcc PORT MAP (a, b, ctemp, ctemp);

c <= ctemp;

END instanciada;
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY modcc IS
PORT (      a, b, cin       : IN std_logic;
            cout            : OUT std_logic);
END modcc;

ARCHITECTURE equacoes OF modcc IS
BEGIN

cout <= (a or b) and (a or cin) and (b or cin);

END equacoes;
```

(b) Código descrito em VHDL para célula de Muller de 2 entradas estrutural

# APÊNDICE B CÓDIGO VHDL PARA CELULA DE MULLER DE 3 ENTRADAS

Código em VHDL para célula de Muller de 3 entradas. Descrição comportamental e estrutural.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY cc3_comportamental IS
PORT (      a, b, c     :       IN std_logic;
            s           :       OUT std_logic);
END cc3_comportamental;

ARCHITECTURE instanciada OF cc3_comportamental IS
SIGNAL stemp: std_logic;
BEGIN
PROCESS (a, b, c)
begin
IF (a=b) and (a=c) THEN
stemp <= a;
ELSE
stemp <= stemp;
END IF;
s <= stemp;
END PROCESS;

END instanciada;
```

(a) Código descrito em VHDL para célula de Muller de 3 entradas comportamental

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY cc3_estrutural IS
PORT (      a, b, c  :          IN std_logic;
            s        :          OUT std_logic);
END cc3;

ARCHITECTURE instanciada OF cc3_estrutural IS
SIGNAL stemp: std_logic;

COMPONENT modcc3
PORT (      a, b, c, srealim:   IN std_logic;
            sout        :       OUT std_logic);
END COMPONENT;

BEGIN

instanciacc: modcc3 PORT MAP(a, b, c, stemp, stemp);

s <= stemp;

END instanciada;
----------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY modcc3 IS
PORT (      a, b, c, srealim    :   IN std_logic;
            sout                :   OUT std_logic);
END modcc3;

ARCHITECTURE equacoes OF modcc3 IS
BEGIN

sout <= (a or b or c)
            and (a or srealim)
            and (b or srealim)
            and (c or srealim);

END equacoes;
```

(b) Código descrito em VHDL para célula de Muller de 3 entradas estrutural

# APÊNDICE C  CÓDIGO VHDL PARA CELULA DE MULLER DO TIPO 2X2 ENTRADAS

Código em VHDL para célula de Muller do tipo 2x2 entradas, variação da célula de Muller de 2 entradas. Descrição comportamental e estrutural.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY celula2x2_comp IS
PORT (      a, b, c  :          IN std_logic;
            Saída    :          OUT std_logic);
END celula2x2_comp;

ARCHITECTURE instanciada OF celula2x2_comp IS
SIGNAL temp, stemp: std_logic;

COMPONENT celulac_comportamental
PORT (      a, b        :       IN std_logic;
            c           :       OUT std_logic);
END COMPONENT;

BEGIN

INST_1: celulac_comportamental PORT MAP (a, b, temp);
INST_2: celulac_comportamental PORT MAP (temp, c, stemp)

Saída <= stemp;

END instanciada;
```

(b) Código descrito em VHDL para célula de Muller de 2x2 entradas comportamental

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY celula2x2_estrutural IS
PORT (      a, b, c     :       IN std_logic;
            Saída       :       OUT std_logic);
END celula2x2_estrutural;

ARCHITECTURE instanciada OF celula2x2_estrutural IS
SIGNAL temp, stemp: std_logic;

COMPONENT celulac_estrutural
PORT (      a, b        :       IN std_logic;
            c           :       OUT std_logic);
END COMPONENT;

BEGIN

INST_1: celulac_estrutural PORT MAP (a, b, temp);
INST_2: celulac_estrutural PORT MAP (temp, c, stemp);

Saída <= stemp;

END instanciada;
```

(c) Código descrito em VHDL para célula de Muller de 2x2 entradas estrutural

# APÊNDICE D CÓDIGO VHDL PARA REGISTRADOR ASSÍNCRONO

Código em VHDL para registrador assíncrono. Descrição comportamental e estrutural.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY registrador_comp IS
PORT (      Et, Ef, habilita      :      IN std_logic;
            St, Sf                :      OUT std_logic);
END registrador_comportamental;

ARCHITECTURE instanciada OF registrador_comp IS
SIGNAL st_temp, sf_temp: std_logic;
BEGIN
PROCESS (Et, Ef, habilita, st_temp, sf_temp)
BEGIN
IF (habilita='0') THEN
      st_temp <= '0';
      sf_temp <= '0';
      ELSIF (Et='0') and (Ef='0') THEN
            st_temp <= st_temp;
            sf_temp <= sf_temp;
      ELSE
      st_temp <= Et;
      sf_temp <= Ef;
END IF;
St <= st_temp;
Sf <= sf_temp;
END PROCESS;
END instanciada;
```

(a) Código em VHDL do registrador assíncrono comportamental

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY registrador_estrutural is
PORT (      Et, Ef, habilita      :      IN std_logic;
            St, Sf                :      OUT std_logic);
END registrador_estrutural;

ARCHITECTURE instanciada OF registrador_estrutural IS

SIGNAL temp_s_t, temp_s_f, temp1, temp2, nset: std_logic;

BEGIN

nset <= not habilita;
temp1 <= not(temp_s_t or Et);
temp_s_t <= not (temp1 or nset);
temp2 <= not (temp_s_f or Ef);
temp_s_f <= not (temp2 or nset);

St<= temp_s_t;
Sf<= temp_s_f;

END instanciada;
```

(b) Código em VHDL do registrador assíncrono estrutural

# APÊNDICE E CÓDIGO VHDL PARA SOMADOR COMPORTAMENTAL COM INDICAÇÃO FORTE

Código em VHDL para somador comportamental com indicação forte.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY soma_comportamental IS
PORT (At, Af, Bt, Bf, Cint, Cinf          :     IN std_logic;
      St, Sf, Coutt, Coutf                :     OUT std_logic);
END soma_comportamental;

ARCHITECTURE comportamental OF soma_comportamental IS

SIGNAL not_Af, not_Bf, not_Cinf                 : std_logic;
SIGNAL st_temp, sf_temp, cout_temp, couf_temp   : std_logic;

BEGIN
not_Af<= not(Af);
not_Bf<= not(Bf);
not_Cinf<= not(Cinf);

PROCESS (At, Af, Bt, Bf, Cint, Cinf, not_Af, not_Bf, not_Cinf)
BEGIN
      IF ((At = not_Af) and (Bt = not_Bf) and (Cint = not_Cinf)) THEN
            st_temp<=   (Cint and Af and Bf)
                        or (Cint and At and Bt)
                        or (Cinf and At and Bf)
                        or (Cinf and Af and Bt);
            sf_temp<=   (Cint and Af and Bt)
                        or (Cint and At and Bf)
                        or (Cinf and At and Bt)
                        or (Cinf and Af and Bf);
            cout_temp<= (At and Bt) or (At and Cint) or (Bt and Cint);
            couf_temp<= (Af or Bf) and (Af or Cinf) and (Bf or Cinf);
            ELSIF ((At=Af) and (Bt=Bf) and (Cint=Cinf) and (At=Bt) and (At=Cint) and (At='0'))
            THEN
            st_temp<= '0';
            sf_temp<= '0';
            cout_temp<= '0';
            couf_temp<= '0';
            ELSE
            st_temp<= st_temp;
            sf_temp<= sF_temp;
            cout_temp<= cout_temp;
            couf_temp<= couf_temp;
      END IF;
st <= st_temp;
sf <= sf_temp;
coutt <= cout_temp;
coutf <= couf_temp;
END PROCESS;
END comportamental;
```

# APÊNDICE F CÓDIGO VHDL PARA SOMADOR INTEIRO – FULL-ADDER

Código em VHDL para somador inteiro do tipo full adder. Descrição comportamental e estrutural.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY adder IS

PORT (
        a,b,cin        : IN std_logic;
        sum,cout     : OUT std_logic );
end adder;
ARCHITECTURE fulladder OF adder IS
BEGIN
    sum <= a xor b xor cin;
    cout<= (a and b) or (cin and (a or b));
END fulladder;
```

(a) Código VHDL para full adder estrutura padrãol

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY adder_comp IS
PORT (
         a,b,cin: in std_logic;
         sum,cout: out std_logic);
END adder_comp;
ARCHITECTURE fulladder OF adder_comp IS
SIGNAL sum_temp, cout_temp      :  STD_LOGIC;
BEGIN
     PROCESS (a,b,cin)
     BEGIN
        IF cin='0' THEN
                IF a='0' THEN
                sum_temp<= b;
                cout_temp<= '0';
                ELSE
                sum_temp<= not b;
                cout_temp<= b;
                END IF;
        ELSIF a='0' THEN
                sum_temp<= not b;
                cout_temp<= b;
                ELSE
                sum_temp<= b;
                cout_temp<= '1';
        END IF;
END PROCESS;
sum<= sum_temp;
cout<= cout_temp;
END fulladder;
```

(b) Código VHDL para full adder comportamental padrão