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

GIOVANI HERIBERTO SARTORI

Estudo e Implementação de Somador com Detecção de Fim de Cálculo para Circuitos Assíncronos

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

Prof. Dr. Renato Perez Ribas
Orientador

Porto Alegre, maio de 2005

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Sartori, Giovani Heriberto

Estudo e Implementação de Somador com Detecção de Fim de Cálculo para Circuitos Assíncronos / Giovami H. Sartori – Porto Alegre: Programa de Pós-Graduação em Computação, 2005.

117 f.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2005. Orientador: Renato P. Ribas.

1. Circuitos Aritméticos. 2. Arquiteturas auto-temporizadas. 3. Circuitos Assíncronos. 4. Projeto e simulação de somadores. I. Ribas, Renato P. II. Título.

AGRADECIMENTOS

Em primeiro lugar gostaria de agradecer a meus pais Waldecy e Neli. Pelo suporte, educação carinho, apoio. Por fornecerem um ambiente muito amoroso e sempre acreditarem e incentivarem este trabalho de forma a auxiliar na obtenção de mais esta conquista.

Ao meu amor Emmanuelle, pela paciência, apoio e incentivo durante os momentos mais críticos do desenrolar desta dissertação, sem os quais a conclusão da mesma teria sido muito mais difícil. Pelo amor e dedicação dispensados de forma a tornar os desafios tarefas muito mais amenas.

Ao meu orientador Renato Perez Ribas, por incentivar e acreditar neste trabalho quando muitas vezes eu mesmo já não mais acreditava. Ao programa de pós-graduação pelo apoio representado pela qualidade de ensino e infra-estrutura e ao CNPQ pelo custeamento financeiro.

A meus irmãos Robinson e Lúcio por aturarem os dias de mau humor e terem paciência para entender os dias difíceis onde nada parecia dar certo. A meus colegas de apartamento pelo apoio e ao pessoal dos laboratórios do Instituto de Informática que sempre me apoiaram e auxiliaram no desenrolar deste últimos anos.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

AND	Função lógica ‘e’.
CLA	*Carry look-ahead.*
DCVS	*Differential cascode voltage switch logic.*
DI	*Delay insensitive.*
DIMS	*Delay insensitive minterm synthesis*.
DPTL	*Differential pass transistor logic*.
ECDL	*Enable/disable CMOS differential logic*
MDC	Máximo divisor comum.
MMC	Mínimo múltiplo comum.
MO	Múltiplas saídas.
NCL	*Null convention logic*.
NOR	Função lógica ‘ou’ negado.
OR	Função lógica ‘ou’.
QDI	*Quasi delay insensitive.*
RCA	*Ripple carry adder.*
SI	*Speed-independent*.
SDP	Soma de produtos.
XOR	Função lógica ‘ou exclusivo’.

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

É contínua a procura por técnicas de construção de circuitos que ajudem a minimizar os problemas existentes no mercado de microeletrônica atual. Uma alternativa para a resolução destes problemas consiste na utilização de circuitos assíncronos.

Circuitos aritméticos são alvo de um contínuo esforço na busca de melhores resultados de desempenho e área. Em especial o somador é uma das partes constituintes desta classe de circuitos que apresenta interessante campo para pesquisas.

Este trabalho apresenta um método de avaliação de somadores implementados através do uso de famílias lógicas CMOS *dual-rail*. Esta tarefa é realizada através do uso de um circuito assíncrono que serve como base de avaliação. Este circuito obedece ao protocolo de comunicação utilizado pelos somadores e nele são desenvolvidas diversas aplicações para que seja possível avaliar o comportamento dos somadores quando expostos a diferentes padrões de vetores. Os parâmetros avaliados nas estruturas dos somadores são número de transistores, atraso e consumo de potência para topologias *carry look-ahead* e *ripple carry adders*.

Na avaliação dos somadores através de simulação elétrica são utilizadas as ferramentas Pspice e Spectre da Cadence. As tecnologias utilizadas nesta caracterização são AMI 0.5 da MOSIS e AMS 0.35. Como resultados são apresentados dados que demonstram a economia no número de transistores obtida através do uso da técnica de múltiplas saídas para o CLA, que a família DCVS geralmente apresenta os menores atrasos médios quando comparada a outras estruturas e a potencialidade de famílias NCL.

Palavras-chave: circuitos aritméticos, arquiteturas auto-temporizadas, circuitos assíncronos, projeto e simulação de somadores.

Study and Implementation of Adders with Completion Detection Targeted to Asynchronous Circuits Design

ABSTRACT

The search for construction techniques of circuits that helps to minimize the challenges that occurs in nowadays microelectronic market is continuous. An alternative to solve great part of these problems is the use of asynchronous circuits.

Arithmetic circuits are the target of a continuous effort in the pursuit of better results in terms of performance and area. Adder circuits in special compose a subset of this class of circuits that presents an interesting research field.

This work presents an evaluation method for adders that where implemented through different dual-rail logic families. This task is accomplished through the use of asynchronous circuits used as an evaluation base. The asynchronous circuits implemented obey the communication protocol adopted by the adders and implement different applications. These applications are constructed with the finality of study the adder's behavior when they are exposed to different vector patterns. The adder's evaluated parameters are the number of transistors, delay and power consumption of topologies like Carry Look-ahead and Ripple Carry Adders.

The electrical simulations were accomplished trough the use of Pspice and Cadence's Spectre cad tools. MOSIS AMI 0.5 and AMS 0.35 transistor technologies were utilized in the electrical characterization of the adders. Some of the results obtained trough this work that could be cited are: the low transistor count presented for the Multiple Outputs CLA structures, the performance advantage of the DCVS family in relation to the other families and the evaluation of NCL logic family potentiality.

Keywords: arithmetic circuits, self-timed architectures, asynchronous circuits, project and simulation of adders.

1 INTRODUÇÃO

Nos dias de hoje, o mercado de microeletrônica encontra-se em crescente necessidade de circuitos cada vez mais densos, mais complexos, com altas freqüências de operação e construídos com áreas ativas cada vez mais reduzidas. O cumprimento dessas premissas tem criado novos desafios aos projetistas de circuitos integrados.

A grande capacidade de integração VLSI tem tornado possível a solução de inúmeros problemas, mas à medida que alguns são resolvidos, novos desafios são encontrados. Justamente o aumento de freqüência de operação aliada à necessidade de alta integração e baixo consumo de potência faz com que surjam algumas limitações nas técnicas de projeto atuais que mais comumente se utilizam de tecnologia CMOS convencional.

Como é dado grande destaque à necessidade de baixo consumo de potência nos dias de hoje, passa-se a observar que esta característica configura mais uma dificuldade de projeto. Tal problema tem surgido principalmente devido à expansão de novos mercados, como o de sistemas móveis (celulares, computadores móveis, etc.), que por necessitarem de um sistema de alimentação próprio e limitado, acabam fazendo com que essa característica "baixa potência" seja mais importante que a velocidade de operação do circuito em si.

Uma possível alternativa para que seja possível a resolução de grande parte dos problemas causados pelas necessidades tecnológicas de hoje em dia é projetar os circuitos usando técnicas que façam com que estes dispositivos possam trabalhar de forma assíncrona.

Esta técnica apresenta interessantes vantagens em certos aspectos como, por exemplo, a não necessidade de um sinal de relógio global. Cada bloco funcional que compõe o circuito trabalha somente quando requisitado, ou seja, não há a necessidade de que todos os blocos estejam em perfeita sincronia para o correto funcionamento do circuito. Tem-se, portanto a eliminação do problema de distribuição de relógio para todas as partes do circuito quando o mesmo se encontra trabalhando em freqüências consideravelmente altas.

Pode-se citar também como potencial vantagem do uso de técnicas assíncronas a diminuição do consumo de potência. Esta diminuição é atingida simplesmente pelo fato de que em circuitos síncronos sempre que se têm transições do sinal de relógio a mesma é propagada pelo circuito inteiro, fazendo com que vários blocos que poderiam permanecer inativos acabem chaveando os seus transistores assim gerando consumo de energia, fato que não ocorre em circuitos assíncronos afinal os blocos só são ativados quando necessário. Outra fonte de redução de consumo de potência reside na eliminação do circuito gerador de relógio que nos sistemas atuais é responsável por uma parte considerável do consumo total do sistema.

É interessante também salientar a potencial diminuição do ruído eletromagnético emitido. Tal característica é atingida devido a novamente os blocos funcionarem somente quando requisitados, ou seja, somente determinadas áreas do circuito estão funcionando a cada momento, fazendo com que as emissões de radiação tendam a ser mais randômicas e conseqüentemente de menor amplitude.

Isto pode ser mais bem explicado da seguinte forma: quando se trabalha com circuitos síncronos em determinados momentos bem definidos o sinal de relógio varia fazendo com que aconteçam chaveamentos, ou seja, em outras palavras, nestes momentos uma grande quantidade de corrente é drenada da fonte, fazendo com que o sinal de corrente tenha o aspecto de diversos picos de consumo ao longo do tempo. Já quando se trabalha com circuitos assíncronos, onde não existe um sinal de relógio, os picos de consumo de corrente tendem a ser mais bem distribuídos pelo tempo fazendo assim com que o aspecto da função corrente em relação com o tempo passe a ter uma característica de consumo médio ao longo do tempo, onde a emissão de ruído eletromagnético é bastante reduzida.

Tem-se também a característica de cada bloco funcional poder trabalhar com o menor tempo de propagação possível, sendo este dependente apenas da combinação de suas entradas. Como neste tipo de arquitetura trabalha-se com protocolos de comunicação entre os blocos constituintes, esta possível menor latência é conseguida através do uso de sinais dedicados que servem para indicar ao protocolo quando os blocos funcionais terminaram de fazer uma computação, ou seja, sempre que um bloco termina a tarefa a ele designada, é emitido para o próximo bloco um sinal de "pronto" que indica a disponibilidade de dados válidos no barramento de comunicação.

Este tipo de característica não pode ser observado em métodos síncronos, pois o sinal de relógio que servirá para sincronizar o circuito terá sempre que ser calculado observando-se o pior caso de propagação. Esta necessidade surge devido à existência da sincronização em si, pois ela deve contemplar com precisão o maior tempo possível de computação para que haja a certeza da validade nos dados contidos no barramento, fazendo assim com que muitas vezes haja partes do circuito que ficam ociosas, esperando o próximo ciclo de relógio, quando na verdade já terminaram há algum tempo os seus cálculos.

Por ser feito de forma modular, este tipo de arquitetura acaba não dependendo tanto de análises temporais e dos aspectos tecnológicos admitidos, assim fazendo com que não seja necessário que se repense todo o circuito sempre que acontece algum avanço tecnológico de processo, facilitando conseqüentemente a migração tecnológica e o reprojeto de partes do circuito. Esta modularidade é atingida devido à existência de um protocolo de comunicação entre os blocos, protocolo este que é insensitivo a atrasos (SPARSO, 2001). Como o protocolo é independente dos atrasos, o tempo que um bloco leva para realizar um cálculo não influência em nenhuma variável global (relógio) do sistema, pois a mesma não existe, fazendo assim com que não seja necessário ajuste global devido a uma variação local.

As principais desvantagens apresentadas por esta técnica de construção de circuitos consistem na complexidade de projeto e no relativo aumento da área final dos circuitos quando estes são comparados aos gerados de forma síncrona. A complexidade muitas vezes é devida ao baixo volume de estudos e a deficiência existente na área de ferramentas de CAD que auxiliem no projeto dos sistemas. O acréscimo na área geralmente é causado pela necessidade de se adicionar circuitos dedicados ao protocolo

de comunicação existente entre os blocos, sendo que este aumento é dependente do protocolo escolhido.

Além da diminuição do consumo de potência, como citado acima, é contínua a busca por técnicas de projeto de circuitos que visem aperfeiçoar o desempenho de circuitos aritméticos, que são essenciais na construção dos mais diversos sistemas eletrônicos. Algumas dessas técnicas propostas para este tipo de circuitos consistem em métodos de construção de circuitos que funcionem de modo auto temporizado.

Estruturas aritméticas são uma classe muito importante de circuitos. As mesmas são comumente encontradas em praticamente qualquer projeto da área de microeletrônica. Uma variada gama de circuitos faz parte desta classe (como, somadores, subtratores, etc...). Elas estão principalmente presentes em unidades aritméticas lógicas (ULA).

Uma estrutura que se destaca dentre esta gama de possibilidades é o somador. Ele tem sua importância devido a ser uma unidade básica que faz parte de muitas das arquiteturas aritméticas mais complexas (como multiplicadores e filtros digitais). Devido a esta constante presença na construção de aplicações aritméticas, esta estrutura é alvo de constante pesquisa com o intuito de melhorar o seu desempenho.

Este estudo intensivo desta estrutura pode ser notado pela quantidade de modos de implementação apresentados na literatura (assim como *ripple carry adders*, *carry-look ahead adders*, *paralel prefix adders*, *carry selection adder, manchester adder*, etc...). Uma nova abordagem para a construção do somador consiste em projetá-lo de forma auto-temporizada. Esta abordagem é interessante para que seja possível fazer um estudo de como esta estrutura se comporta quando utilizada desta forma.

Um grande desafio que surge quando se trabalha com circuitos aritméticos auto temporizados se concentra na avaliação de desempenho dos mesmos. Em sistemas síncronos geralmente é possível caracterizar estas estruturas em relação a atraso através da análise da topologia do mesmo ou de ferramentas específicas que determinam o maior atraso de propagação existente. Esta informação é então utilizada como um dos fatores que influenciam no cálculo do período mínimo de relógio.

Ao se trabalhar com circuitos auto-temporizados, esta análise realizada em topologias síncronas não tem mais sentido, devido a não existência do relógio. Em circuitos assíncronos cada parte do sistema funciona somente quando requisitada e sua velocidade de operação depende principalmente da combinação dos vetores de entrada do bloco, o que faz com que seja necessária uma análise do desempenho médio do sistema ao invés da determinação do pior caso de propagação.

Esta análise se torna um desafio, pois a geração de vetores de teste e repetição de simulações para que possa ser alcançada uma medida média de desempenho do sistema acaba se tornando um trabalho muito dispendioso. Isto é devido ao número geralmente expressivo de valores que devem ser gerados para uma simulação que forneça resultados médios significativos.

1.1 Motivação

Devido aos fatos citados acima surge á motivação para iniciar-se um trabalho envolvendo tanto circuitos assíncronos como técnicas de construção de circuitos aritméticos auto temporizados. No grupo de microeletrônica já havia um estudo envolvendo técnicas de construção de circuitos dinâmicos diferenciais com múltiplas

saídas, no caso Enable/Disable CMOS differential Logic (ECDL), Differential Pass Transistor Logic (DPTL) e Differential Cascode Voltage Switch Logic (DCVS) (OSÓRIO, 2004a), (SAMPAIO, 2004), (OSÓRIO, 2004b), além do início de um grupo para estudos de circuitos assíncronos.

O operador aritmético escolhido para ser analisado é o somador. Este operador aritmético foi escolhido por ser parte constituinte de praticamente qualquer parte operativa. Devido a ser tão usual, é válido um estudo do mesmo, pois a partir da sua otimização podemos obter ganhos nos mais diversos campos de aplicação. As arquiteturas básicas das quais os somadores irão fazer parte são na verdade circuitos que desempenham alguma função aritmética pré-determinada.

A estrutura utilizada em todas as arquiteturas básicas assíncronas consiste em um anel assíncrono. Tal estrutura é muito semelhante para todas as operações aritméticas construídas, somente havendo diferenças na parte do anel que concentra a lógica propriamente dita da operação e no esquema que determina o fim da operação. Já as outras partes constituintes do anel são iguais, fazendo com que todos os circuitos tenham o mesmo sistema de inicialização, o mesmo número de registradores, multiplexadores e o mesmo sistema de controle.

As operações aritméticas escolhidas para que fossem executadas nas arquiteturas básicas assíncronas foram: contador, divisão inteira, divisão do resto, máximo divisor comum (MDC), mínimo múltiplo comum (MMC) e raiz quadrada inteira.

1.2 Objetivos

Realizar um estudo de topologias de circuitos aritméticos com o intuito de aperfeiçoar a construção dos blocos somadores utilizando-se de diversas famílias CMOS. Atividade esta que auxiliará no desenvolvimento e otimização dos blocos construídos ao longo do desenvolvimento dos trabalhos.

Avaliar a funcionalidade e características quanto à área, desempenho e consumo de potência de diversas famílias lógicas CMOS sendo que tais famílias compreendem DCVS, DPTL, ECDL, Martin, *null convention logic* (NCL) e *delay insensitive minterm synthesis* (DIMS). A avaliação das mesmas será realizada através de simulações elétricas geradas a partir de uma descrição em esquemáticos a nível de transistor.

Outro objetivo é comprovar a validade e funcionalidade da utilização da arquitetura básica assíncrona. Ao final deste trabalho é esperado que se possa obter dados e conhecimentos suficientes para que seja possível a avaliação da mesma com respeito à validade de sua utilização como ferramenta para avaliação de parâmetros de circuitos aritméticos que sejam compatíveis com a mesma.

1.3 Estrutura

A dissertação que é apresentada a seguir tem os seus temas constituintes divididos da seguinte forma:

- No capítulo dois são descritos os fundamentos de circuitos assíncronos, a teoria que demonstra sua estruturação, os protocolos de comunicação envolvidos e o conceito de “fim de cálculo”, que é o parâmetro passado ao controle pelas diferentes estruturas de famílias abordadas neste trabalho.

- No capítulo três será apresentada a arquitetura básica assíncrona em conjunto com seus componentes internos, assim como os estudos de caso, onde serão descritas topologias básicas utilizadas, algoritmos e implementações dos mesmos.
- No capítulo quatro serão apresentadas as estruturas utilizadas na construção dos somadores (DIMS, Martin, NCL, ECDL, DPTL e DCVS), uma análise dos mesmos quando construídos em uma topologia *ripple carry adder* (RCA) e os resultados e análise dos dados obtidos.
- No capítulo cinco serão apresentadas as estruturas diferenciais com múltiplas saídas (MO), a idéia que descreve deste tipo de aproximação, as estruturas diferenciais utilizadas (MODCVS, MODPTL, MOECDL) e uma análise comparativa baseando-se na utilização de uma topologia *carry-look ahead* (CLA).
- No capítulo seis é apresentada a conclusão deste trabalho.

2 CIRCUITOS ASSÍNCRONOS

Neste capítulo serão apresentadas as bases teóricas das técnicas de construção de circuitos assíncronos que serão úteis para o entendimento do trabalho aqui apresentado. No mesmo é fornecido um panorama geral do assunto sendo que alguns tópicos serão mais aprofundados devido a sua relevância para o entendimento dos procedimentos utilizados na construção deste trabalho.

2.1 Introdução

Circuitos assíncronos, assim como circuitos síncronos, se baseiam em uma estrutura em que um bloco de computação é sempre acompanhado por dois elementos de armazenagem tanto em suas entradas como saídas (ver Figura 2.1). A principal diferença reside no modo como estes elementos de armazenagem são controlados.

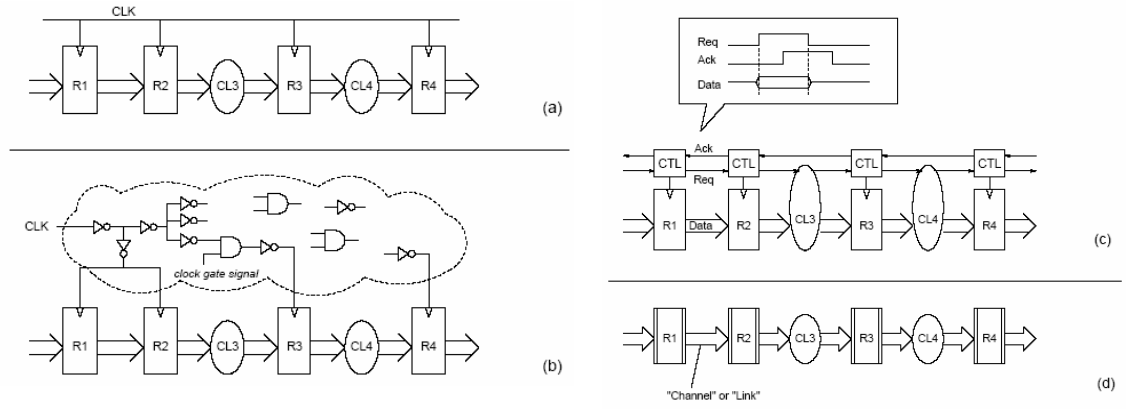


Figura 2.1 : a) Representação de um circuito síncrono genérico, b) o mesmo circuito agora adicionados circuitos necessários à construção da rede de relógio, c) um circuito assíncrono equivalente ao primeiro, d) uma representação mais abstrata do tipo "*data-flow*" do circuito assíncrono (SPARSO, 2001).

Em circuitos síncronos, tais elementos são controlados por um sinal de relógio. Quando se trabalha no domínio assíncrono, este controle é sempre baseado em um protocolo de comunicação ou *handshake* (tal protocolo é demonstrado na Figura 2.2). Como pode ser visto este *handshake* consiste em uma comunicação entre dois elementos de armazenagem, onde 'R1' em um primeiro momento avisa a 'R2' que disponibilizou dados no barramento (req = 1, ver Figura 2.2a), ao passo que o segundo responde dizendo que recebeu os dados (ack = 1, ver Figura 2.2b), então fazendo com que o primeiro indique que recebeu esta informação (req = 0, ver Figura 2.2c) de forma a fazer com que o segundo coloque ack = 0 (ver Figura 2.2d) assim fazendo com que

um novo processo de comunicação possa ser iniciado. Detalhes mais aprofundados sobre protocolos de comunicação serão apresentados mais adiante.

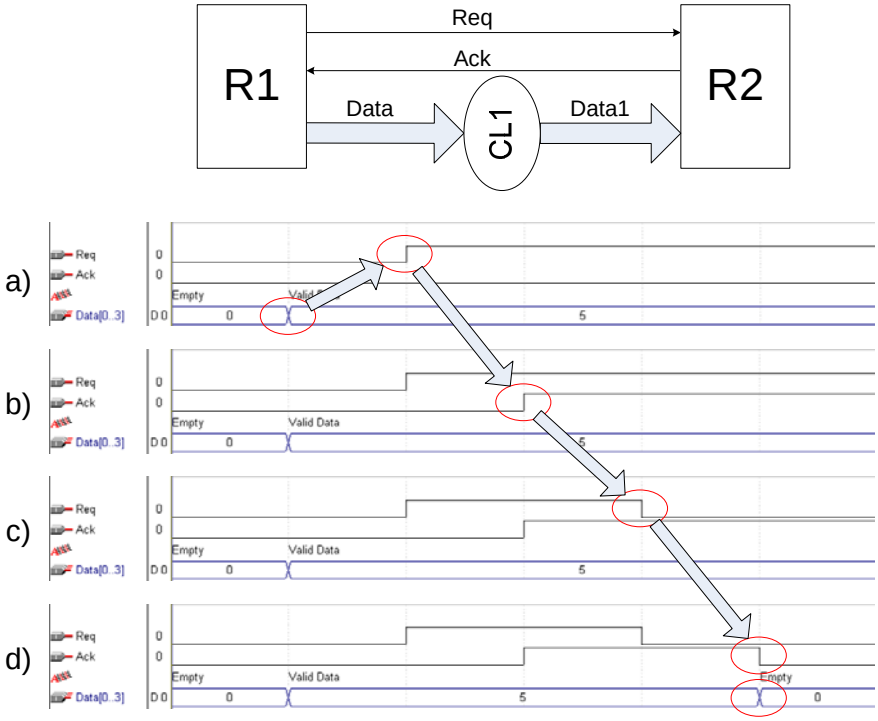


Figura 2.2 : Etapas de comunicação em uma seção de *handshake*.

Em implementações reais, o bloco de computação ‘CL1’ adiciona um atraso nos dados que foram colocados no barramento de forma a fazer com que os sinais de controle geralmente se propaguem de forma mais rápida que os dados (‘Req’ chega a ‘R2’ antes de ‘Data1’). Esta situação causa um problema, pois há um erro na indicação do sinal de controle fazendo assim com que provavelmente ‘R2’ armazene um sinal qualquer que esteja presente no barramento ao invés de ‘Data1’, o que não ocorreria se ‘CL1’ pudesse indicar a ‘R2’ que já realizou a operação de computação e os dados em suas saídas estão estáveis.

A partir deste momento, a construção de circuitos combinacionais, aqui chamados de blocos funcionais, se torna mais complicada. Em circuitos assíncronos os sinais que transitam entre os blocos constituintes do sistema e o circuito de controle devem ser válidos sempre, todo o sinal repassado ao bloco de controle significa alguma coisa, portanto sinais intermediários ou *hazards* não são permitidos fazendo com que surja a necessidade de que eles sejam *hazard free*.

Portanto surgem duas opções, a primeira é fazer com que todos os blocos funcionais do circuito sejam de fato *hazard free* e possuam a capacidade de indicar a validade dos dados em sua saída, ou seja, gerarem sinais de *request* ao final do processo de computação realizado por eles. A segunda consiste na imposição de certas restrições temporais de modo a fazer com que *hazards* sejam tolerados de forma a não comprometer o funcionamento do sistema. Ambas as opções serão explicadas nos próximos parágrafos.

Existem dois tipos de *hazards* que podem ocorrer em blocos funcionais: *hazards* estáticos e dinâmicos como demonstrado em (SPARSO, 2001). *Hazards* estáticos são variações no sinal de saída de uma porta quando na verdade este valor deveria se manter constante. Já *hazards* dinâmicos são flutuações do sinal de saída de uma porta no momento em que ocorre variação nas suas entradas.

Um exemplo de *hazard* dinâmico pode ser explicado pela seguinte situação; imagine o circuito da Figura 2.3 onde a porta 'um' tem um atraso de 0,1ns, a porta 'dois' tem um atraso de 0,3ns e a porta 'três' tem um atraso de 0,1ns e que todos os sinais de entrada, saída e intermediários estão zerados inicialmente. Se o circuito for excitado com os valores mostrados na figura em um dado momento, o valor de saída sofrerá um *hazard*, pois o sinal que deveria ter permanecido sempre em '0' permaneceu em '1' por um período de 0,2ns, tempo o suficiente para propagar uma informação errônea pelo circuito.

A outra opção na construção de blocos funcionais, como citado acima, consiste na utilização de algum artifício de engenharia para através de alguma suposição temporal garantir que os sinais estarão estáveis no momento certo de serem avaliados pelo bloco de controle e armazenados pelo próximo registrador. Um modo simples de demonstrar tal estratégia pode ser notado no trabalho sobre *Micropipelines* (SUTHERLAND, 1989).

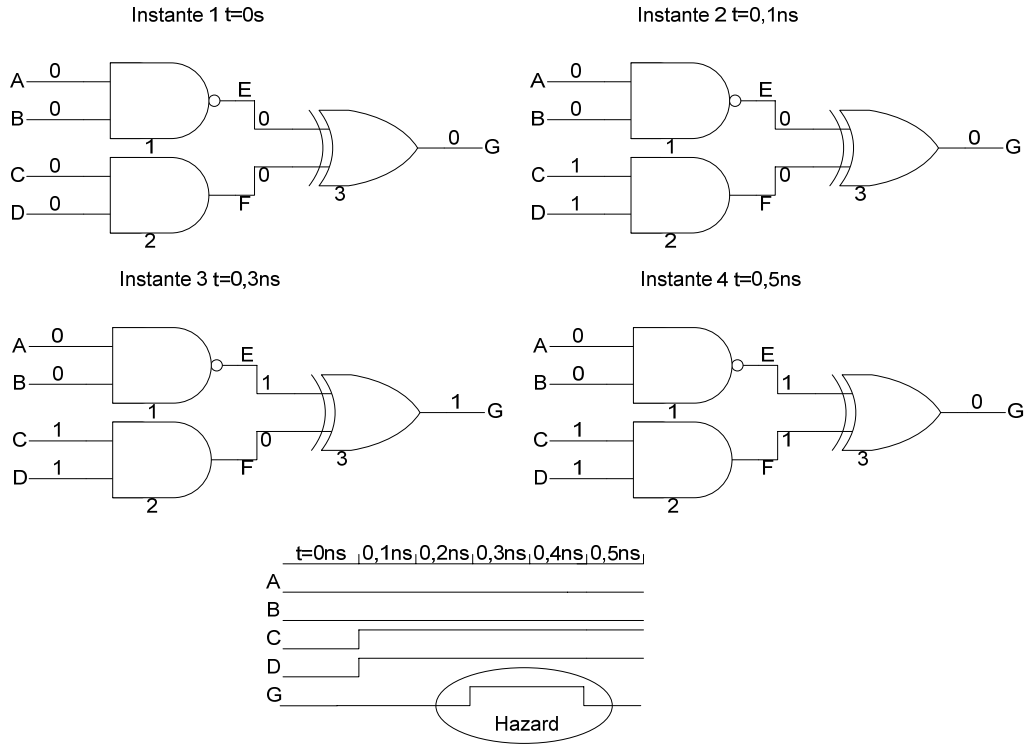


Figura 2.3 : Demonstração de *hazard* dinâmico.

Neste trabalho é apresentada uma técnica na qual é adicionado um "bloco de atraso" que faz com que o tempo de propagação do sinal de *request* seja maior ou igual ao atraso do caminho crítico do bloco funcional existente entre duas células de armazenamento vizinhas (ver Figura 2.4). Deste modo é possível admitir que ocorram

variações nas saídas do bloco funcional ‘CL1’ antes que o sinal de controle seja propagado pelo “bloco de atraso” de forma a não influenciarem no funcionamento do sistema, pois no momento em que ‘Req’ atingir ‘R2’ os dados ‘Data1’ já estarão prontos e estáveis em suas entradas.

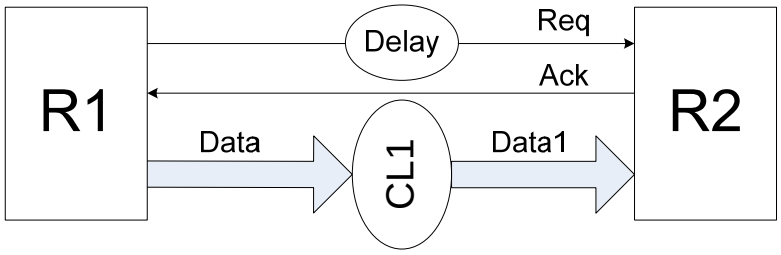


Figura 2.4 : Esquema de construção utilizando a técnica de *Micropipelines*, onde um elemento de atraso é adicionado ao sinal de *request*.

Devido a estas duas opções de tratamento dos sinais de controle, surge a classificação dos circuitos assíncronos sendo que as mesmas são: *delay insensitive* (DI), *quasi delay insensitive* (QDI) e *speed-independent* (SI). Circuitos *delay insensitive* são circuitos que independentemente de qualquer atraso existente, tanto no roteamento quanto nos blocos funcionais, funcionarão corretamente. Já circuitos *quasi delay insensitive* são circuitos que operam corretamente assumindo-se atrasos maiores que zero para os blocos funcionais e iguais à zero para o roteamento. Já circuitos que necessitem de esquemas mais avançados ou grandes restrições temporais para que operem corretamente, como no exemplo dos *Micropipelines*, são chamados simplesmente de *speed-idependent* (SUTHERLAND,1989).

2.2 Blocos funcionais

Os blocos funcionais em circuitos assíncronos são equivalentes a circuitos combinacionais em arquiteturas síncronas. A grande diferença entre os circuitos combinacionais e os blocos funcionais reside na necessidade dos blocos funcionais terem de indicar quando finalizam a operação a eles devida. O que geralmente ocorre é o bloco receber dados em canais separados, ter que sincronizá-los, processá-los e os enviar em canais separados de forma coordenada avisando ao próximo bloco que os dados disponíveis no barramento são válidos.

Blocos funcionais em circuitos assíncronos sempre estão colocados entre dois estágios de armazenamento (*latches*) sendo que para o segundo *latch*, se considerado como observador, deve ocorrer à mesma seqüência de sinais de *handshaking* com e sem a existência do bloco funcional entre eles (transparência ao protocolo). Este requisito de transparência faz com que o bloco funcional além de ter de sincronizar sinais de entrada e indicar o fim de execução da tarefa a ele submetida também não possa emitir nenhum sinal de controle, no caso *request*, até o momento em que tenha recebido o mesmo do estágio anterior e sem que esteja com todas as suas saídas válidas e estáveis.

Devido aos quesitos expostos anteriormente surgem duas classificações para os blocos em relação ao tipo de indicação que eles repassam, um bloco pode ser do tipo indicação forte ou fraco.

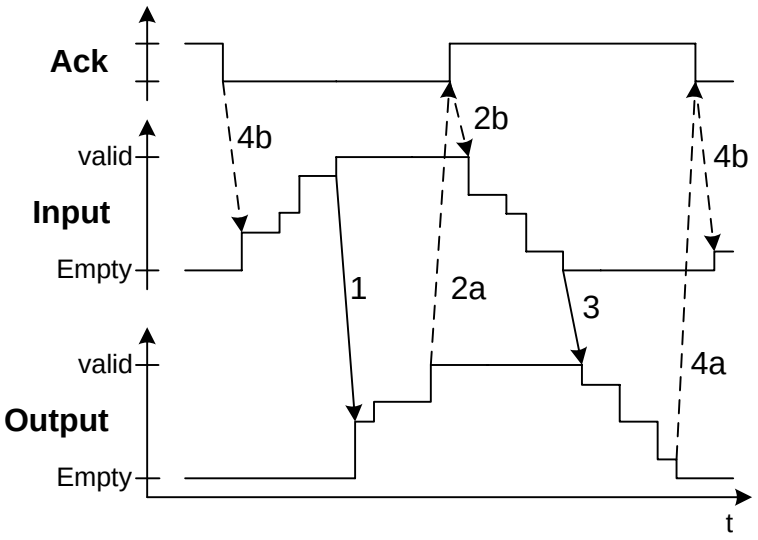


Figura 2.5 : Diagrama de sinais contendo o ordenamento dos mesmos para um bloco com indicação forte (SPARSO, 2001).

As etapas de transição apresentadas na figura são as seguintes:

1) Todas as entradas se tornam válidas, algumas saídas começam a ficar válidas.

2a) Todas as saídas ficam válidas.

2b) As entradas começam a ficar válidas.

3) As entradas ficam todas inválidas e as saídas começam a ficar inválidas.

4) Todas as saídas ficam inválidas, algumas entradas começam a ficar válidas.

Os blocos de indicação forte (ver Figura 2.5) são elementos que esperam até que todas as suas entradas estejam válidas e estáveis para que comece a execução de sua função e conseqüentemente a produção de saídas válidas, assim como ele espera até que todas as suas entradas estejam com o valor “vazio” (ausência de dados) até que comece a produzir saídas vazias.

Os blocos com indicação fraca já começam a computar e produzir saídas assim que possível, ou seja, quando algumas, mas não todas as entradas estão válidas, o mesmo ocorrendo para valores “vazios”. Apesar de produzirem saídas assim que possível, para que funcionem corretamente deve-se garantir que não sejam produzidas “todas” as saídas válidas antes que “todas” as entradas estejam válidas e estáveis como mostrado na Figura 2.6.

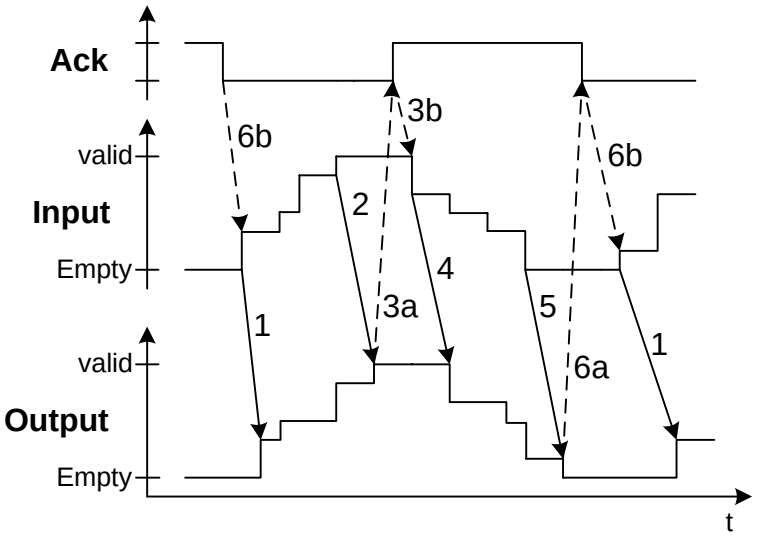


Figura 2.6 : Diagrama de sinais contendo o ordenamento dos mesmos para um bloco com indicação fraca (SPARSO, 2001).

As modificações apresentadas no diagrama da Figura 2.6:

1) Algumas entradas se tornam válidas, algumas saídas começam a ficar válidas.

2) Todas as entradas e saídas ficam válidas.

3) As entradas começam a ficar válidas.

4) Algumas saídas ficam válidas.

5) Todas as entradas ficam inválidas.

6) Todas as saídas ficam inválidas.

2.3 Fim de Cálculo em Blocos Funcionais

Como os circuitos assíncronos não têm um sinal de relógio, mas sim sinais gerados localmente que determinam o comportamento do mesmo, surge nesses circuitos o princípio de indicabilidade ou fim de cálculo. Quando dois blocos querem transmitir informações ou um bloco funcional terminou a operação a ele cabida, eles têm de ter condições de avisar ao sistema que mandaram a informação, que receberam algum dado ou que terminaram de realizar alguma operação o que implica no fim de cálculo.

Existem vários esquemas para a detecção do fim de cálculo, como *Current-Sensing Completion Detection* (CSCD) (DEAN, 94), detecção por validade de dados (SPARSO, 2001) e detecção por paridade (TRAVER, 2001). Geralmente os pontos mais usuais de amostragem deste sinal são posicionados na detecção de final de operação de um bloco funcional ou de validade dos sinais contidos em um barramento. Ambos são demonstrados na Figura 2.7 abaixo, onde mostramos a necessidade da detecção da validade dos dados no barramento precedente ao *Latch* ‘L1’ através de ‘FC1’ para que o mesmo possa então transmitir estes dados e a indicação de fim de operação executada por ‘FC2’ nas saídas do bloco funcional ‘CL1’ para que o *Latch* ‘L2’ possa armazenar os dados processados.

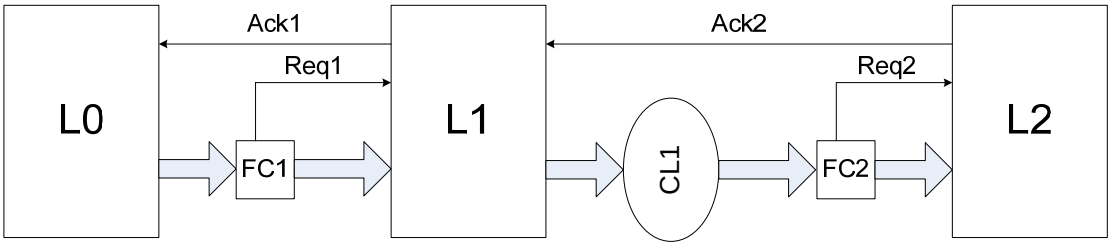


Figura 2.7 : Detecção de fim de cálculo por FC1 e FC2 em um circuito assíncrono.

2.4 Protocolos

Devido a não existência de um sinal de sincronização global entre os blocos constituintes dos circuitos construídos através de técnicas de projeto assíncrono, faz se necessário que estes mantenham alguma forma de comunicação entre si. Esta comunicação tem a finalidade de controlar o fluxo de dados entre os blocos, assim como gerar os sinais de controle dos dispositivos de armazenamento.

Em circuitos assíncronos sempre haverão protocolos de comunicação controlando o comportamento do sistema. Estes protocolos fazem o papel do sinal de relógio em circuitos síncronos, mas com a grande diferença de não chavearem o circuito como um todo, mas sim só as partes do circuito que estão participando efetivamente da computação executada.

Os blocos se comunicam através de sinais de *request* e *acknowledge*, ou seja, quando um bloco quer se comunicar com outro ele envia um sinal de *request* (o que seria o mesmo que dizer 'estou enviando dados'), ao ponto que o outro responde sinalizando com *acknowledge* (o mesmo que 'Ok, recebi os dados'), sendo que esta sinalização pode ser feita em ambos os sentidos. O exemplo acima é o exemplo de um canal do tipo *push* (mostrado anteriormente na Figura 2.2), o primeiro bloco envia os dados e o segundo avisa que os recebeu, mas podemos ter um canal do tipo *pull* onde a ordem dos sinais é invertida, ou seja, o segundo bloco pede os dados e então o primeiro os envia, fazendo com que independentemente do canal que envia dados o protocolo é obedecido.

Os principais protocolos utilizados no projeto de circuitos assíncronos são *bundled data* e *dual-rail* sendo que estes ainda se dividem em protocolos de duas e quatro fases (SPARSO, 2001). No protocolo *bundled* data a sinalização de *request* e *acknowledge* é executada através de somente um fio de comunicação para cada um dos sinais enquanto que no *dual-rail* o sinal de *request* é codificado junto com os dados. As combinações dos protocolos assim como uma explicação mais detalhada dos mesmos é apresentada a seguir.

2.4.1 Protocolo *Bundled-data*

No protocolo *bundled-data*, existem canais de comunicação próprios para dados, sinal de *request* e sinal de *acknowledge*. Isto quer dizer que além do barramento de dados existem um fio para comunicação do sinal de *request* e outro para *acknowledge*, o que é uma das principais diferenças em relação ao protocolo *dual-rail* que será apresentado a seguir.

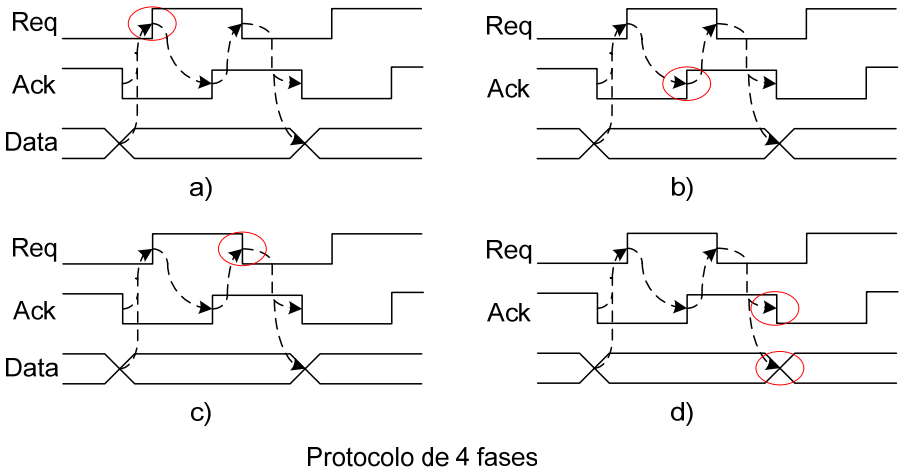


Figura 2.8 : Seqüência de passos de comunicação em um protocolo de 4 fases.

O protocolo *bundled-data* é mais usualmente encontrado em codificações de duas e quatro fases. Esta codificação em fases diz respeito ao número de ações de comunicação executados, por exemplo, quando estamos um protocolo de quatro fases é utilizado tem se primeiro o bloco emissor disponibilizando os dados e "setando" o valor do sinal *request* em '1' (ver Figura 2.8a), então o bloco receptor absorve os dados e seta o valor do sinal *acknowledge* em '1' (ver Figura 2.8b), a partir deste momento o receptor coloca o valor de *request* em zero indicando que os dados no barramento não são mais válidos (ver Figura 2.8c) fazendo com que a partir deste momento o receptor coloque o sinal de *acknowledge* em '0' (ver Figura 2.8d), o que significa que o ciclo está completo e que pode se dar início a uma nova etapa de processamento. Como pode-se notar no processo de transmissão de dados ocorreram quatro ações de comunicação.

Esta comunicação de quatro fases pode ser comparada analogicamente a um processo de aperto de mão. Um processo onde a primeira pessoa estende a sua mão, a segunda então estende a sua apertando assim a do primeiro, ao passo que o primeiro recolhe a sua mão, indicando que o segundo já pode fazer o mesmo estando então ambos prontos para um novo processo de aperto de mãos.

Já no protocolo *bundled-data* de duas fases (ver Figura 2.9), o que indicará a modificação e validade de dados nos canais será uma transição ao invés de uma mudança de nível. Esta mudança pode ocorrer tanto no sentido de '0' para '1' quanto de '1' para '0' pois ambas representarão um evento de mudança. Apesar de geralmente levarem a construção de circuitos com maior velocidade que o protocolo de quatro fases e por evitarem a necessidade de um retorno para zero entre as transições de dados, o que os torna mais eficientes em relação a tempo e energia, o protocolo de duas fases não é tão utilizado devido a ser mais complicado gerar circuitos que respondam a estímulos do tipo transição do que circuitos que respondam a nível.

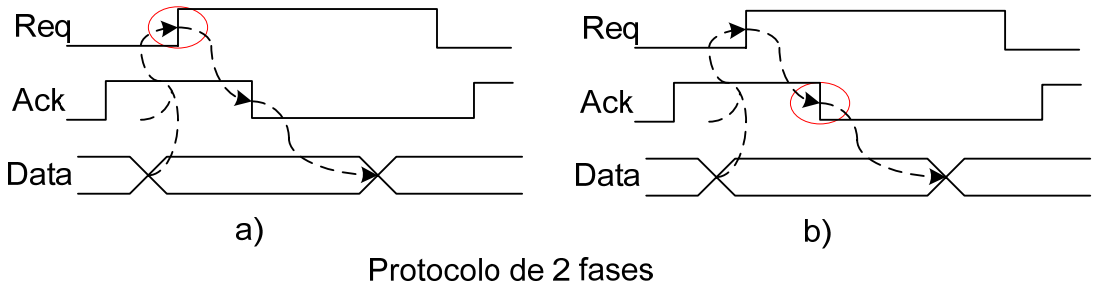


Figura 2.9 : Seqüência de eventos de comunicação em um protocolo de 2 fases.

Outro ponto que é interessante observar é que esta separação do sinal de *request* do canal de dados acaba incorrendo em um agravamento da dificuldade de projeto do sistema. Este agravamento ocorre devido ao pré-requisito de validade dos sinais durante todo o período de funcionamento do sistema.

Imagine a seguinte situação, o fio de *request* com um atraso de propagação de 0,2ns em conjunto com um canal de dados de quatro bits, onde o atraso de propagação do fio do bit 'zero' é igual a 0,1ns, do bit 'um' é 0,15ns, do bit 'dois' é 0,2ns e do bit 'três' é de 0,25ns. Agora imagine que o bloco que está enviando dados deseja que a seqüência (1,1,1,1) seja enviada então ele coloca o sinal de *request* em '1' e envia os dados. Após 0,2ns o segundo bloco recebe o sinal de *request* e então faz uma amostra do barramento de dados e os armazena, ou seja, armazena o valor (0,1,1,1) incorrendo em um erro de transmissão de dados.

Devido à possibilidade demonstrada acima é que surge a dificuldade de projeto, pois tem-se a partir dela a necessidade de fazer com que o atraso de propagação do sinal de *request* seja igual ou superior ao maior dos atrasos do barramento de dados, o que resulta em um problema muito parecido com o de redimensionamento de redes de relógio, além de demonstrar que o referido protocolo não é *delay insensitive*.

2.4.2 Protocolo *Dual-Rail* quatro fases

No protocolo *dual-rail* o sinal de *request* é codificado juntamente com os dados a serem transmitidos. Esta codificação é feita da seguinte forma, cada bit é transmitido a partir de dois fios, isto é para uma transmissão de 'n' bits são necessários '2n' fios. A cada valor lógico é dado um código composto de um valor verdadeiro 'd.t.' e um falso 'd.f.', no caso o código correspondente ao valor lógico '1' é d.t.=1 e d.f.=0, enquanto que para o valor lógico '0' é d.t.=0 e d.f.=1. Ainda existe mais um código que é chamado "vazio" correspondente aos valores d.t.=0 e d.f.=0 que é usado como sinal intermediário entre dois dados válidos.

Não são permitidas transições diretas entre dois dados válidos (valor lógico '0' ou '1') como demonstrado na Figura 2.10, ou seja, sempre existe um valor '*empty*' entre dois dados válidos. A codificação d.t.=1 e d.f.=1 não é permitida e o sinal '*empty*' significa dado inválido.

	D.t.	D.f.
Empty	0	0
Valid “0”	0	1
Valid “1”	1	0
Não Usado	1	1

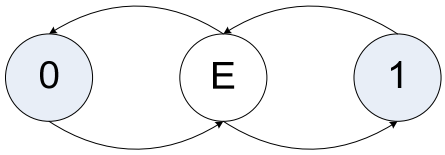


Figura 2.10 : Tabela verdade e diagrama de estados de uma transmissão entre dados válidos.

Quando se trabalha com protocolo *dual-rail*, o bloco receptor deve ser capaz de identificar o momento em que os dados em seu barramento de entrada são válidos. Como dito anteriormente, como o código d.t.=1 e d.f.=1 não é permitido no protocolo esta tarefa se simplifica pois uma simples porta OR já é o suficiente para identificar dados válidos em um barramento de 1 bit ou um dado ‘*empty*’ (princípio do fim de cálculo).

Devido a este esquema de codificação este protocolo acaba se tornando muito robusto, fazendo com que dois blocos possam se comunicar independentemente do atraso existente entre eles, ou seja, este é um protocolo *delay insensitive*.

A seqüência de eventos que ocorre em uma comunicação de um protocolo como este é a seguinte, o bloco emissor disponibiliza os dados codificados com o *request* no barramento, o bloco receptor identifica um dado válido em suas entradas e coloca o sinal de *acknowledge* em ‘1’, o emissor recebe o sinal de *acknowledge* e envia para o barramento um código ‘*empty*’, o receptor percebe este código ‘*empty*’ e então coloca o sinal de *acknowledge* em nível lógico ‘0’ estando agora ambos os blocos prontos para iniciarem um novo ciclo de comunicação (ver Figura 2.11).

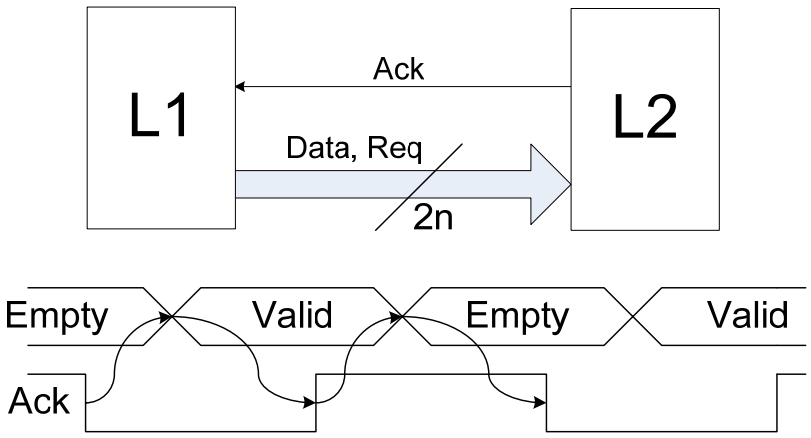


Figura 2.11 : Seqüência de comunicação em um protocolo *dual-rail* de 4 fases.

2.4.3 Protocolo *Dual-Rail* duas fases

No protocolo *dual-rail* de duas fases também são usados ‘2n’ fios para representar ‘n’ bits. A diferença em relação ao anterior é que não existe valor ‘*empty*’, apesar da existência da mesma codificação usada no protocolo anterior para representar dados válidos. O valor ‘*empty*’ não é necessário, pois agora o que indica o evento de mudança de dado válido no barramento é a transição do sinal ‘d.t’. ‘d.f.’ de ‘0,1’ para ‘1,0’ e vice versa, e não uma transição ‘*empty*’ para ‘dado válido’, ou seja, este protocolo é sensível a transições.

Um exemplo de funcionamento deste protocolo é o seguinte: inicialmente tem-se ‘d.t.’ e ‘d.f.’ em ‘1,1’ no momento em que o bloco emissor varia o valor de ‘d.f.’ passando-o de ‘1’ para ‘0’, fazendo com que o valor de ‘d.t.’ ‘d.f.’ fique em ‘1,0’. Neste momento o receptor varia o sinal de *acknowledge* indicando que um novo ciclo pode ser iniciado. Como se pode notar, ao final deste ciclo, o valor de ‘d.t.’ ‘d.f.’ é ‘1,0’ o que no protocolo anterior indicaria um valor válido igual a ‘1’, mas como este protocolo é regido por variações, e o sinal que variou foi o ‘d.f’., estamos na verdade passando um dado válido que é igual ao valor lógico ‘0’.

O que deve ser observado neste protocolo é quem variou e considerar esta variação como o evento que determina o valor lógico que está sendo passado pela operação, ou seja, se ‘d.t.’ variar, o valor lógico a ser passado é o ‘1’ ao ponto que se ‘d.f.’ variar o valor lógico a ser passado será ‘0’. Um exemplo de comunicação deste protocolo é mostrado na Figura 2.12.

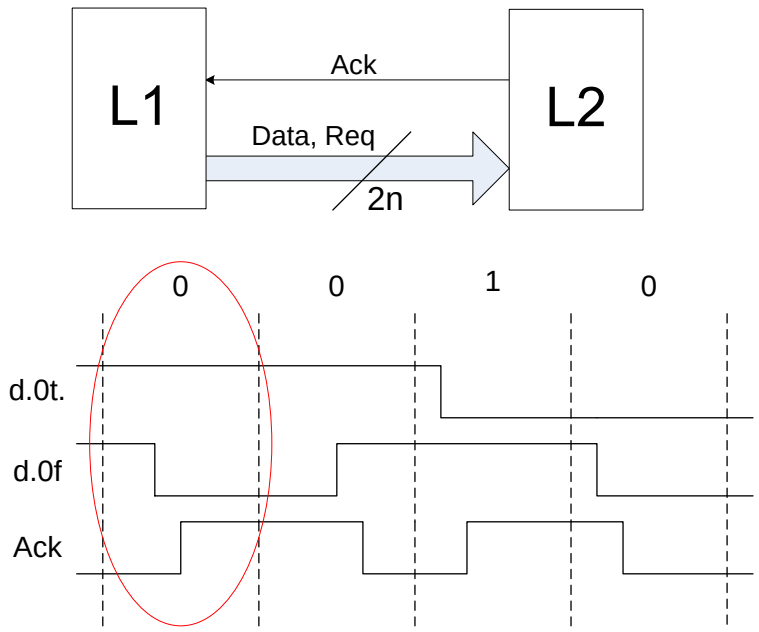


Figura 2.12 : Seqüência de comunicação em um protocolo *dual-rail* de 2 fases.

2.5 Arquiteturas Assíncronas

Neste tópico serão apresentadas as estruturas básicas utilizadas na construção de uma arquitetura assíncrona. Toda a teoria apresentada se baseia na escolha como protocolo de comunicação e esquema de validade de dados como sendo *dual-rail* de quatro fases. Esta escolha é devida a este protocolo ser mais usualmente utilizado assim

como por apresentar vantagens quanto à simplicidade e robustez de projeto e por ser compatível com as estruturas de somadores a serem avaliadas.

Todos os sistemas construídos através desta técnica obedecem a seguinte lei de formação: “Um registrador pode receber e armazenar um novo ‘*token*’ de dados de seu predecessor somente se seu sucessor já armazenou o ‘*token*’ previamente armazenado por ele”.

2.5.1 Transferência de *Tokens* em Anéis

O modo de comunicação existente entre elementos de armazenamento (*latches*) é baseado em um protocolo de transferência de *tokens*. Existem dois tipos de *token* (*token* de dado válido, *token* de dado vazio) e dado sem *token*. Quando são construídas tais estruturas há a necessidade de que se tenha no mínimo três estágios de armazenamento, isto é devido justamente ao circuito mínimo conter dois *tokens*, um válido e outro vazio mais um com um dado sem *token* necessário para que os outros dois possam se movimentar, como demonstrado a seguir.

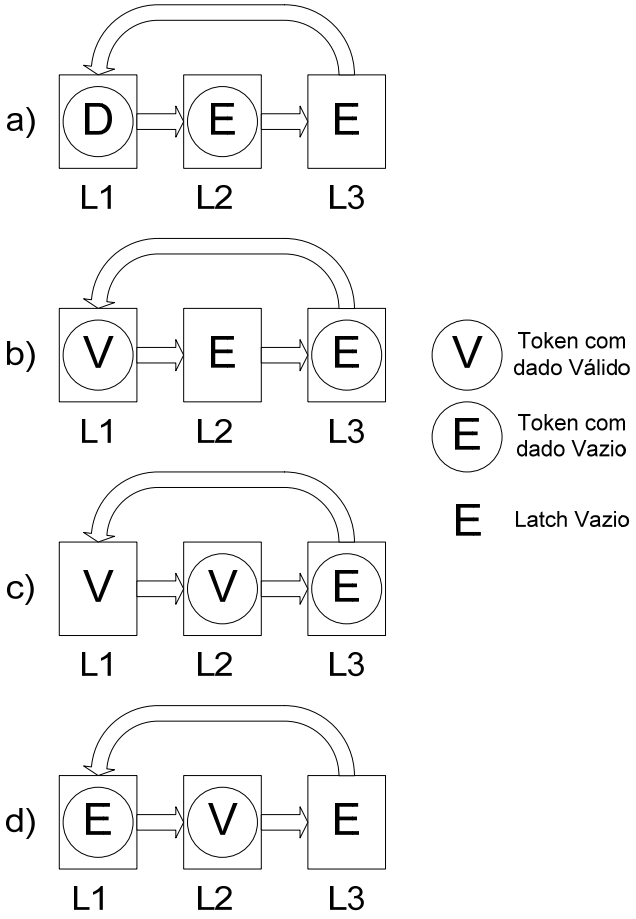


Figura 2.13 : Seqüência de transferência de *tokens* em anéis.

Os *tokens* são representados na Figura 2.13 com uma letra inserida em um círculo e o dado sem *token* por uma letra somente. Na Figura 2.13 é demonstrado o principio de que fala o parágrafo acima, nele é observado um anel simples com três

latches ‘L1’, ‘L2’ e ‘L3’ com seus respectivos *tokens* de acordo com a Figura 2.13a. A seqüência de movimentação dos dados dentro dessa estrutura será a seguinte: o *token empty* em ‘L2’ passa para o *latch* ‘L3’ que contém um espaço vazio deixando um espaço sem *token* em ‘L2’ (ver Figura 2.13b); o *token* válido de ‘L1’ passa para o *latch* ‘L2’ deixando um espaço vazio em ‘L1’ (ver Figura 2.13c); o *token empty* contido em ‘L3’ passa para o *latch* ‘L1’ deixando um espaço vazio em ‘L3’ (ver Figura 2.13d) e assim sucessivamente.

Um modo mais simples e “didático” de apresentar este princípio pode ser apresentado utilizando-se a seguinte analogia. Imagine agora um círculo formado por três crianças (criança = *latch*) que estão prestes a iniciar uma brincadeira, duas crianças seguram tortas em suas mãos, uma de chocolate e a outra de limão (chocolate = *token* válido, limão = *token empty*), sendo que a terceira está de mãos vazias (vazio = *empty*). A regra que define como eles irão brincar é a seguinte: “somente uma criança sem nada nas mãos pode receber uma torta sendo que a mesma só pode ser passada no sentido horário, a brincadeira termina depois de um determinado tempo sendo que ganha quem terminar com a torta de chocolate nas mãos”. Como se pode notar, o fluxo de transferência das tortas será idêntico ao da Figura 2.13.

Este exemplo além de demonstrar de uma forma mais simples a transferência de *tokens* (aqui representados por tortas) em um anel assíncrono também demonstra o porquê da necessidade de no mínimo três estágios. Afinal de acordo com a lei de formação da brincadeira caso existam somente duas crianças e dois tipos de torta (que é o mínimo possível de opções, *token* com e sem dado), não haverá alguém de mãos vazias para receber uma torta de seu companheiro fazendo com que a condição inicial de distribuição das mesmas não seja alterada, ou seja, trancando o anel.

2.5.2 Pipeline Muller

O *pipeline* Muller (MULLER, 59), (MULLER, 63) é um circuito formado somente por um bloco conhecido como célula Muller ou C (que nada mais é que uma porta do tipo *state holding*, ou seja, onde as saídas só são modificadas quando, neste caso existem duas entradas iguais, por exemplo, as duas entradas em zero geram saída em zero e vice versa, sendo que com outras combinações de entrada a saída mantém o estado anterior) e inversores, um exemplo de sua estrutura é mostrado na Figura 2.14. Devido a seu comportamento simétrico e, apesar de não ser óbvio para quem olha a estrutura pela primeira vez obedecer exatamente à lei de formação de anéis assíncronos descrita no início desta seção, esta estrutura acaba tornando-se o principal esquema de controle dessas estruturas.

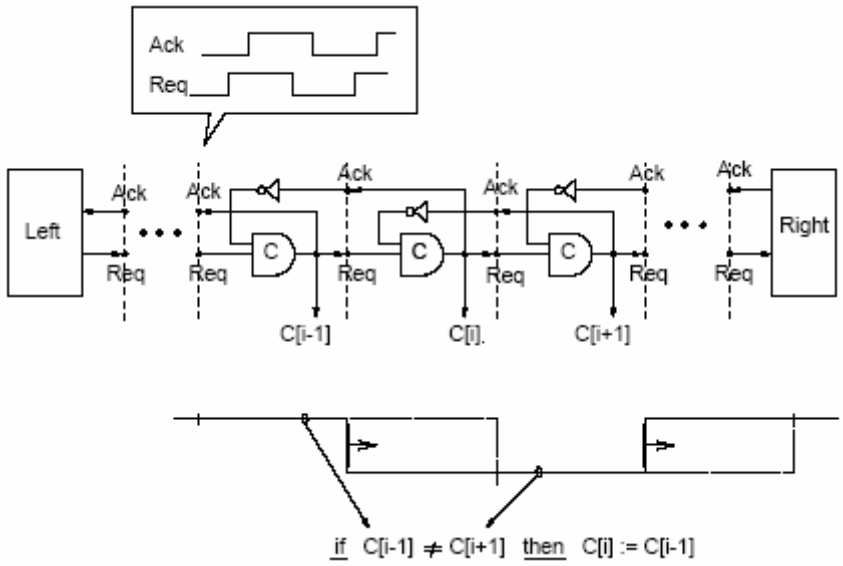


Figura 2.14 : Pipeline Muller (SPARSO, 2001).

Para que essa característica de obedecer exatamente à lei de formação de anéis assíncronos se torne mais óbvia, o comportamento detalhado de um período de comunicação é mostrado na Figura 2.15. Para a figura de exemplo o sinal de *request* pode ser considerado como o que indica a transmissão de um *token* de dados de um bloco de armazenagem (*latch*) para outro. Considerando-se o elemento 'C(2)', ele somente transmitirá um valor '1' de seu predecessor 'C(2-1)' se seu sucessor 'C(2+1)' contiver um '0' (para que ele possa armazenar um valor '{req(2)=1}', 'ack(2+1)' e 'ack(2-1)' têm que ser respectivamente iguais a '0' e '1', ou seja, 'C(2+1)=1' e 'C(2-1)=0)', assim como da mesma forma ele só armazenará um '0' de seu predecessor se seu sucessor for igual a '1', como podemos ver, dessa forma obedecendo ao princípio de formação de anéis assíncronos.

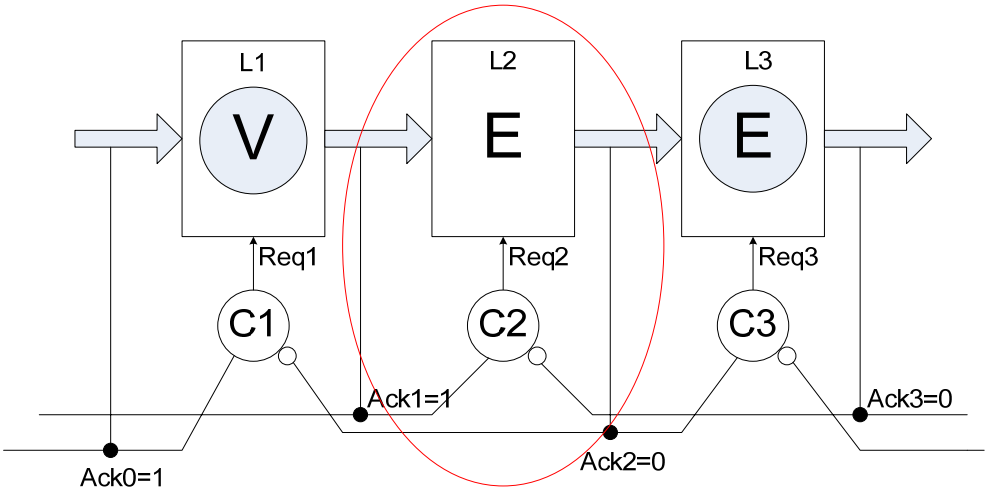


Figura 2.15 : Exemplo de transferência de dados no *pipeline* Muller.

Voltando ao exemplo de crianças e tortas, poder-se-ia considerar que as crianças estão vendadas e somente pudessem dizer as palavras ‘não’, ‘chocolate’ e ‘limão’ para que pudessem trocar as tortas entre si. Digamos que a situação inicial é a mostrada na Figura 2.15, nela substituindo-se ‘L1’ por João, ‘L2’ por Maria e ‘L3’ por Pedro, assim como o sistema de células C por uma comunicação de palavras. Caso Maria estivesse de mãos vazias e desejasse receber a torta que João tem em suas mãos, ela, de acordo com o princípio de funcionamento do *pipeline* Muller teria de ouvir a seguinte palavra de João: ‘chocolate’ o que indica que existe uma torta para ser recebida. Ao mesmo tempo, ela deveria ouvir de Pedro a palavra ‘limão’ fazendo assim com que ela esteja em um estado ‘não-limão’, o que indica que Pedro armazenou o objeto por ela antes carregado, fazendo assim com que a condição seja satisfeita e ela possa receber a torta. Ou analogamente, como diz a lei “somente receber um dado novo quando o dado previamente armazenado tiver sido absorvido pelo elemento seguinte”.

O *pipeline* Muller tem um comportamento muito parecido com um circuito do tipo FIFO, sendo que se o lado direito do sistema não responder quando nele chegar o *request* inserido pelo lado esquerdo, o *pipeline* irá encher assim interrompendo o protocolo de comunicação. Outra característica é que a velocidade com que o processo de *handshake* entre as partes constituintes é executado será definida somente pelo atraso dos blocos que compõe o circuito.

2.5.3 Anel Assíncrono

O anel assíncrono é a estrutura básica resultante da junção das idéias expostas sobre transferência de *tokens* em conjunto com a utilização do princípio do *pipeline* Muller como sistema de controle e sincronismo. Tal estrutura nada mais é do que uma estrutura do tipo *pipeline*, com no mínimo três estágios para as estruturas mais simples, com um sistema de validade de dados do tipo *token* (ver Figura 2.16).

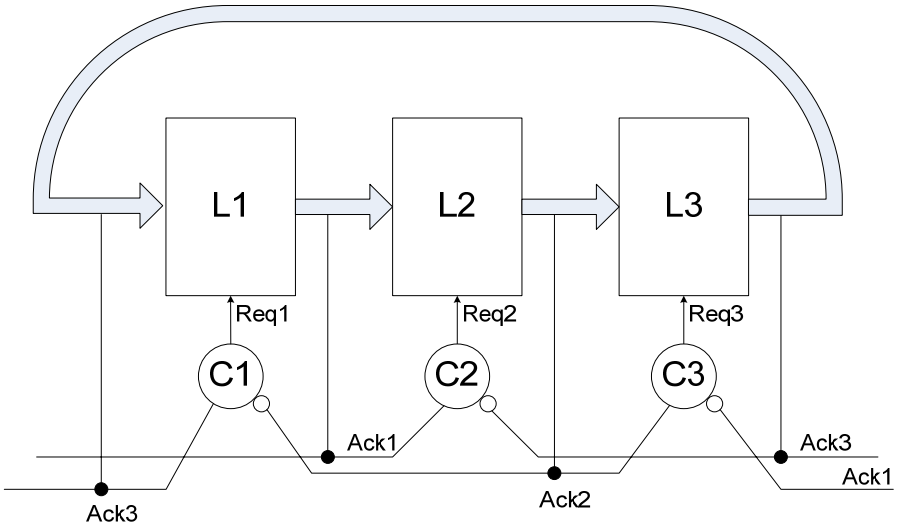


Figura 2.16 : Estrutura do anel assíncrono.

Uma etapa bastante importante na construção de um anel diz respeito ao modo como ele é inicializado. Como dito anteriormente, considerando-se um anel de três estágios, têm-se dois *tokens* e um dado vazio circulando pelo anel. Para que se possa

atingir este estado o anel deve ser incializado, ou seja, os *tokens* de alguma forma devem ser inseridos no anel.

A estrutura utilizada neste anel inicializável é uma variação da apresentada na Figura 2.16 através da adição de um multiplexador ‘M1’. Como pode ser notado em um primeiro momento todos os blocos do anel são “resetados” (Figura 2.17a), fazendo assim com que todos os sinais internos do anel se tornem zero. A partir do momento que o sinal de *reset* é desligado pode-se considerar que o circuito passe ao estado mostrado na Figura 2.17b), onde temos dois elementos com valores *empty* e um *token* com valor *empty*. Esta configuração é devida ao estado inicial do sistema de controle e como temos somente um *token* e nenhum dado válido presente no anel, ele permanece parado de forma a não haver nenhum fluxo de dados.

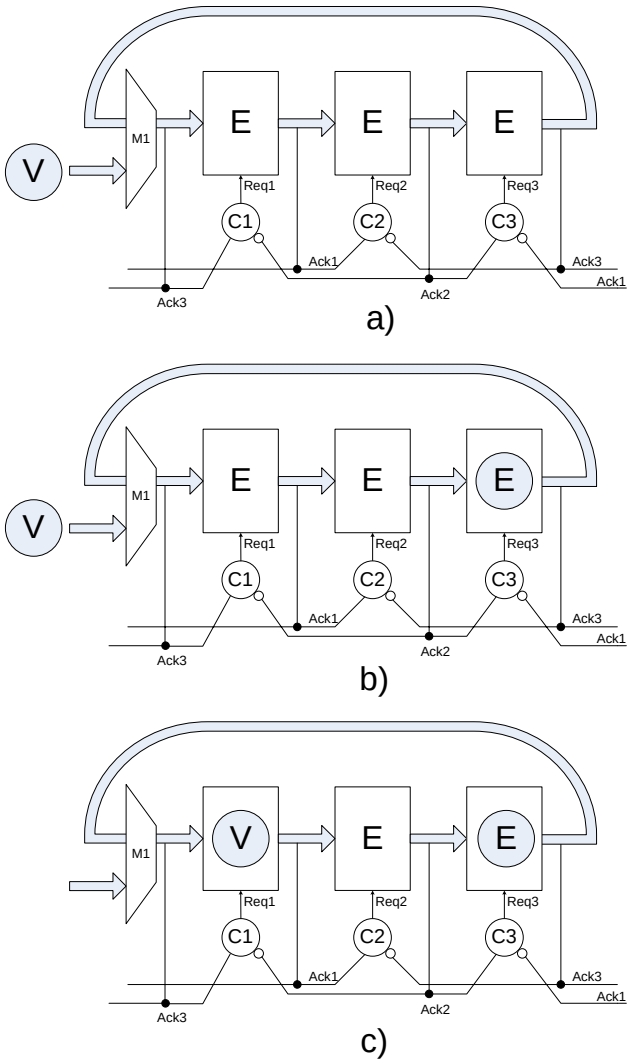


Figura 2.17 : Estrutura do anel assíncrono inicializável; a), b), c), seqüência de passos de inicialização do sistema.

A partir do momento que o sinal do controle do multiplexador ‘M1’ passa para ‘1’, o dado válido presente em suas entradas é transferido para o anel (ver Figura 2.17c). A partir deste momento o sinal de controle do multiplexador pode voltar para zero

assim fazendo com que a realimentação do anel seja restabelecida, pois o dado válido já foi inserido no anel e o processo de transferência de informação inicializado. A inicialização é explicada de uma forma mais detalhada no próximo capítulo.

3 ESTUDOS DE CASO

Neste capítulo serão apresentados os circuitos que servirão de estudo de caso para avaliação da proposta inicial de verificação de parâmetros de somadores. Em um primeiro momento é apresentada a estrutura em anel que serve de base à implementação dos estudos de caso. Em seguida são apresentados os blocos que compõem anel (*latches*, multiplexadores células C, etc...), o modo de inicialização do sistema. Por fim são apresentadas as aplicações desenvolvidas e seus algoritmos formadores.

3.1 Arquitetura Básica Assíncrona em Anel

Como dito anteriormente, todas as aplicações que servirão de estudo de caso para que seja possível avaliar os somadores compartilham de uma mesma estrutura básica. Esta estrutura básica consiste em um anel assíncrono composto por três *latches*, blocos funcionais, multiplexadores e circuito de controle.

Uma introdução ao modo de operação de uma forma genérica de tais anéis já foi apresentada no capítulo 2, sendo que agora será dado um maior enfoque na estrutura usada durante este trabalho. A forma geral do anel utilizado é apresentada na Figura 3.1.

A principal diferença em relação à estrutura apresentada no final do capítulo 2 (Figura 2.17) consiste na substituição do Multiplexador por um conjunto de *latches*, um inicializável e outro normal. O porquê da inclusão destes dois elementos é explicado a seguir no tópico sobre inicialização da estrutura básica em anel.

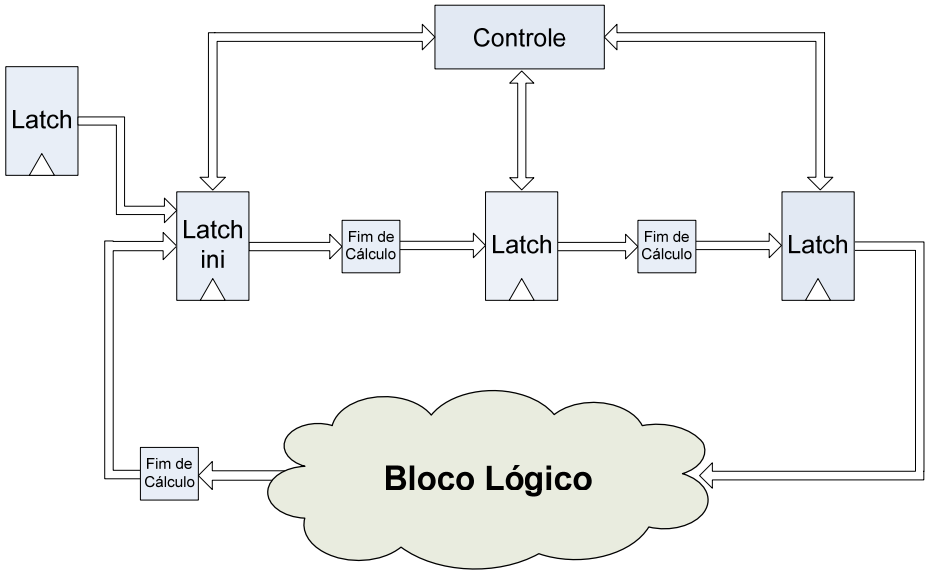


Figura 3.1 : Estrutura assíncrona básica em anel.

Como pode-se notar, esta estrutura é composta por três grupos de *latches* normais, um grupo de *latches* inicializáveis, um bloco lógico, circuitos de detecção de fim de cálculo e pelo circuito de controle. Esta estrutura é construída de forma igual para todos os estudos de caso, ou seja, todas contêm exatamente os mesmos blocos, com exceção do bloco lógico.

O bloco lógico é diferenciado, dependendo do estudo de caso em questão. Esta diferença é devida justamente a ser ele o responsável por fazer com que apesar da estrutura geral ser a mesma, seja possível fazer com que a aplicação executada seja diferenciada. As estruturas constituintes do bloco lógico serão explicadas mais detalhadamente em um próximo tópico deste capítulo.

3.2 Implementações

Neste tópico são apresentadas as estruturas que compõe o anel assíncrono com exceção dos somadores que serão apresentados nos próximos capítulos. Como mostrado no tópico anterior, os anéis são formados por um conjunto de circuitos que são necessários à implementação dos estudos de caso, como por exemplo o *latch*, células C e multiplexadores. Justamente estes dispositivos terão sua estrutura elétrica e modos de operação demonstrados.

3.2.1 Células C

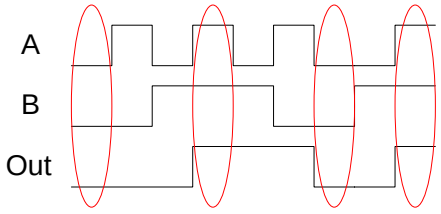
A célula C (ou elemento C ou célula Muller) representa a função lógica fundamental para a implementação do sistema de controle do anel assíncrono. Como dito anteriormente, no capítulo sobre teoria de circuitos assíncronos, esta célula é responsável pela sincronização de sinais. O funcionamento da mesma é muito semelhante a um *latch* do tipo *set/reset.*

A partir do momento em que todas as entradas apresentam o mesmo valor, todas em ‘0’ ou todas em ‘1’, a saída da célula vai para o valor correspondente a ‘0’ e ‘1’ respectivamente. Quando o valor contido nas entradas é diferente da situação apresentada acima, as saídas não se modificam mantendo o valor previamente armazenado.

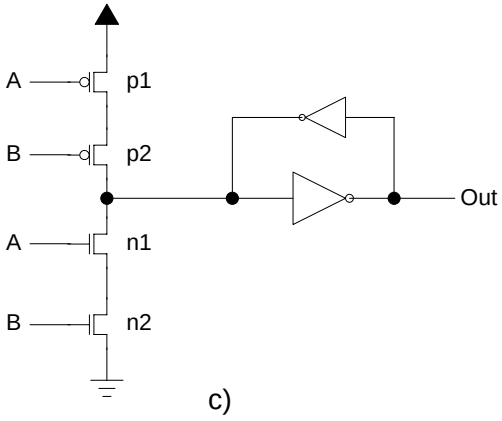
Esta estruturação da célula faz com que a mesma apresente um comportamento de histerese, onde o sinal de saída só é modificado quando as entradas da célula apresentam o mesmo valor. Caso contrário o valor de saída previamente armazenado é mantido, como demonstrado na tabela verdade da Figura 3.2a e pelas formas de onda apresentadas na Figura 3.2b.

A	B	Out
0	0	0
0	1	Não muda
1	0	Não muda
1	1	1

a)



b)



c)

Figura 3.2 : a) Tabela verdade, b) Formas de onda e c) diagrama elétrico de uma célula C.

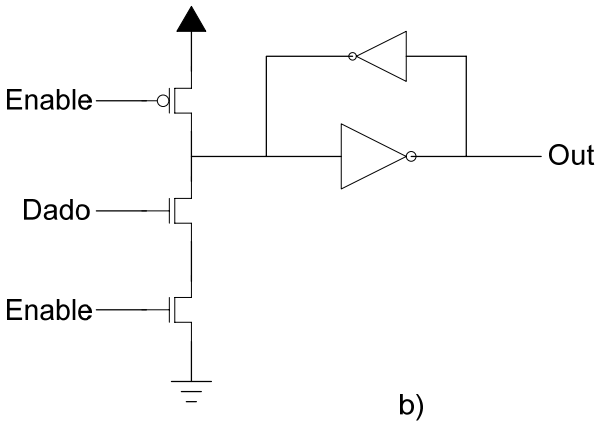
A estrutura elétrica da mesma é apresentada na Figura 3.2c e segue o modelo apresentado na referência (SPARSO, 2001), (MULLER, 59), (MULLER, 63), sendo esta uma estrutura clássica na construção de circuitos assíncronos.

3.2.2 Latches

Nas implementações realizadas neste trabalho temos dois tipos de *latch*, com e sem sinal de inicialização. A estrutura básica do *latch* sem inicialização (sinal de *set* ou *reset*) é apresentada na Figura 3.3.

Dado	Enable	Out
0	0	0
0	1	0
1	0	0
1	1	1

a)



b)

Figura 3.3 : Diagrama elétrico de um *Latch* sem inicialização, a) tabela verdade; b) circuito resultante.

Sua estrutura elétrica se assemelha muito a de uma célula C. Seu principio de funcionamento é o seguinte, quando o sinal '*Enable'* está em nível lógico '0', o transistor 'p1' passa a conduzir assim fazendo com que o valor da saída independentemente da entrada 'Dado' vai ou permanece em '0'. O transistor NMOS ligado ao sinal de "*Enable*" existe para evitar que aconteça um curto entre 'Vdd' e 'gnd' quando acontece a situação em que o valor do sinal 'Dado' vai para '1' no momento em que o sinal de '*Enable'* ainda está em '0'.

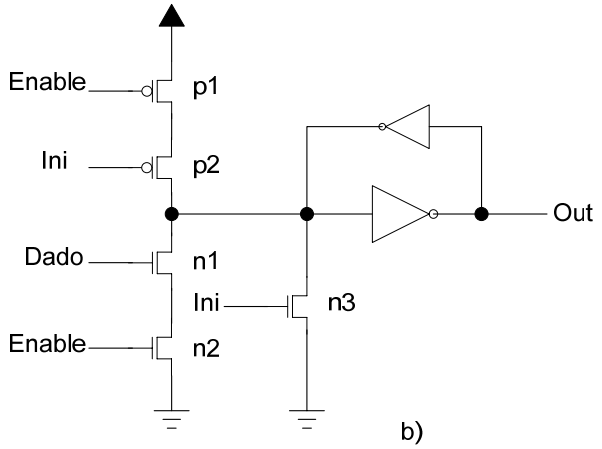
A principal diferença existente entre este *latch* e o comumente projetado em circuitos síncronos é que nele quando um sinal *enable* = 0 é recebido, a saída é "resetada" enquanto que normalmente o dado seria mantido em suas saídas. Este *latch* apresenta um funcionamento transparente para *enable* = 1 e de *reset* para *enable* = 0.

O sinal 'Dado' é representado por somente um transistor na rede do *latch*, pois o mesmo só precisa indicar uma transição para '1'. Esta condição é suficiente, pois a outra única possibilidade é permanecer em '0'. Fato este que não precisa ser indicado por 'Dado' pois uma transição para um valor 'válido' é sempre precedida por um '*empty'*, ou seja, ambas as entradas em '0'.

Já o *latch* com sinal de inicialização é praticamente idêntico ao *latch* normal, excluindo-se os transistores 'p2 e n3' (ver Figura 3.4). Seu princípio de funcionamento é igual ao anterior com a exceção da parte que diz respeito a inicialização. Esta característica pode ser demonstrada da seguinte forma: quando o sinal 'ini' do *latch* vai para '1', independentemente dos demais sinais a saída vai para '1' também (comportamento tipo '*set*'). O transistor 'p2' existe para evitar que um curto entre 'vdd' e 'gnd' aconteça na situação em que temos 'ini' em '1' e '*enable'* em '0'.

Ini	Dado	Enable	Out
1	X	X	1
0	X	0	0
0	0	1	0
0	1	1	1

a)



b)

Figura 3.4 : Diagrama elétrico de um *Latch* inicializável , a) tabela verdade; b) circuito resultante.

3.2.3 Multiplexador

O esquema elétrico do multiplexador é apresentado na Figura 3.5. Como pode ser notado, o controle do mesmo assim como as entradas obedecem ao protocolo *dual-rail*.

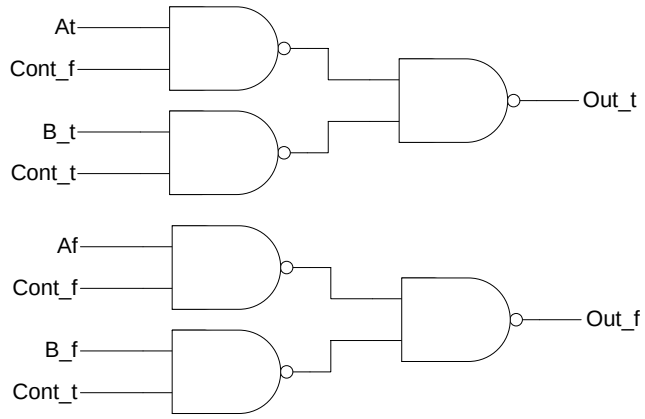


Figura 3.5 : Esquemático no nível de portas lógicas do multiplexador.

Seu princípio de funcionamento é o seguinte, quando o sinal de controle recebe cont_t = 0 e cont_f = 1 o canal 'A' passará para as saídas, já quando controle receber cont_t = 1 e cont_f = 0, o canal que será transmitido às saídas corresponde a 'B'. Quando o sinal de controle receber um dado '*empty*', ou seja, cont_t = 0 e cont_f = 0 a saída também receberá um dado '*empty*'.

3.2.4 Circuito de fim de cálculo

Os circuitos que provêm o sinal de fim de cálculo para que seja possível a comunicação assíncrona são mostrados na Figura 3.6a.

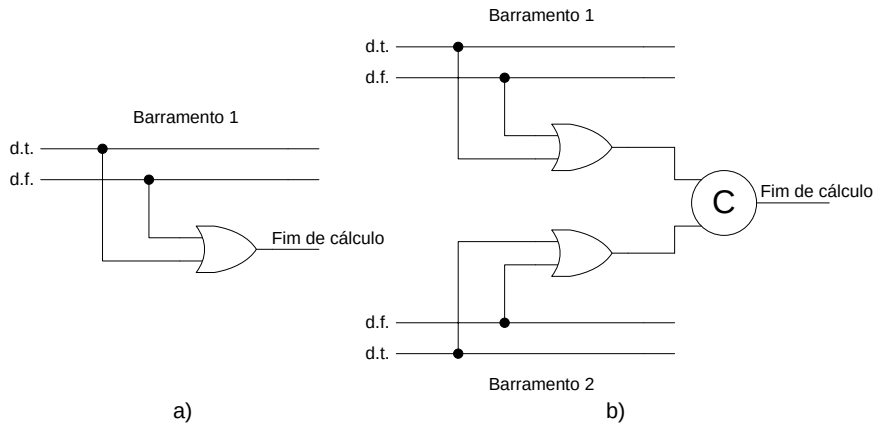


Figura 3.6 : Circuito de detecção de fim de cálculo; a) para barramento único; b) para barramentos múltiplos.

Eles consistem simplesmente em uma porta OR. A porta OR é suficiente para a indicação de dados válidos no barramento, pois como o código '1,1' é proibido (devido a especificações do protocolo *dual-rail*) a indicação se dá pela verificação do sinal '0,1' ou '1,0' no barramento para os quais ela responde "setando" sua saída para '1'. Já o sinal '0,0' indica o valor '*empty*' no barramento ao qual a porta responde "setando" '0'.

Caso sejam considerados mais de um barramento de dados, podemos utilizar uma porta OR para cada conjunto de bits de cada barramento sincronizadas por uma célula C, como indicado na Figura 3.6b. Isto se torna necessário, pois como o sistema de controle coordena ambos os barramentos a validade de sinal também tem de ser sincronizada para ambos.

3.2.5 Circuito de Controle

O circuito de controle é o bloco que coordena e sincroniza o funcionamento de todas as arquiteturas básicas assíncronas que executam um determinado estudo de caso. Neste caso é composto única e exclusivamente de células C e inversores.

Na implementação do bloco de controle foram utilizadas células C modificadas (ver Figura 3.7). Esta modificação diz respeito à inclusão de um sinal de *reset* que é implementado através dos transistores 'p3, n3'. Este sinal se faz necessário para que seja possível garantir um estado inicial conhecido do circuito, ou seja, garantir que todo o circuito contenha valores '*empty*' em todos os seus barramentos, *latches* e blocos funcionais.

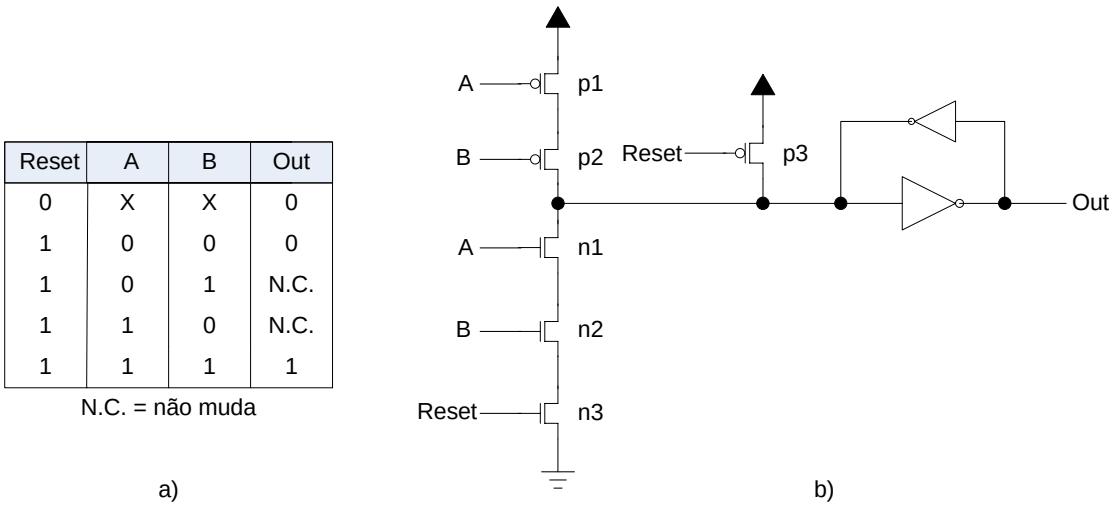


Figura 3.7 : Célula C "resetável" , a) tabela verdade; b) circuito resultante.

Este bloco terá como entradas um sinal de *reset* em conjunto com os sinais providos pelos circuitos de fim de cálculo, aqui chamados de *acknowledge* (ack). O número de sinais de *acknowledge* que chegarão ao bloco de controle serão iguais ao número de *latches* que o circuito contém. No caso como existem quatro *latches* haverá também este mesmo número de sinais de *acknowledge*. Caso o circuito tenha mais estágios de *latches*, a lógica que será apresentada nos próximos parágrafos pode ser estendida com a simples inclusão de células C e inversores.

Como saídas o circuito de controle têm os sinais de *request* (req) que são responsáveis pela indicação do modo de funcionamento do *latch*. Quando o sinal de 'req' vai para '1' o *latch* faz com que suas saídas recebam o dado contido em suas entradas (modo transparente de funcionamento) ao passo que quando o sinal é igual a '0' o *latch* é "resetado" e todas suas saídas vão para nível lógico '0' (modo de bloqueio).

A estrutura de circuito de controle utilizada neste trabalho é apresentada na Figura 3.8. Este circuito obedece à regra já apresentada no capítulo 2 que diz respeito ao anel assíncrono e postula: “Um registrador pode receber e armazenar um novo *token* de dados de seu predecessor somente se seu sucessor já armazenou o *token* previamente armazenado por ele”. Como pode ser visto o circuito de controle nada mais é que a tradução desta lei para um circuito.

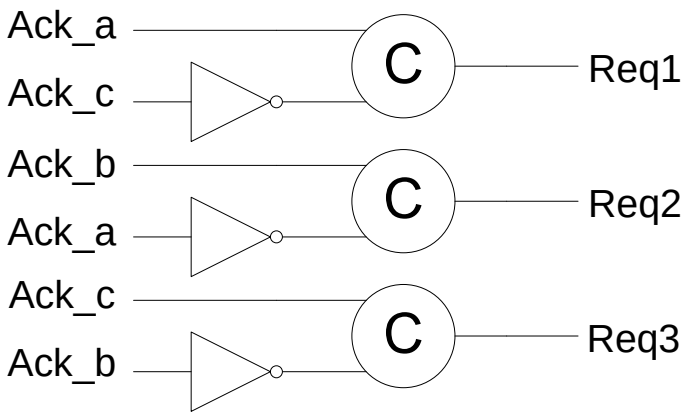


Figura 3.8 : Diagrama elétrico do circuito de controle no nível de portas lógicas.

Para que esta tradução fique mais clara poderemos pegar como exemplo a terceira célula C da Figura 3.8. Como é visto suas entradas correspondem aos sinais ‘Ack_c’ e ‘Ack_b’ e a saída gerada é o sinal ‘Req3’. O sinal ‘Ack_b’ é negado fazendo assim com que a condição necessária para que o *latch* controlado por este circuito esteja em modo transparente de funcionamento é ‘Ack_c = 1’ e ‘Ack_b = 0’ assim como que a condição para que ele esteja em modo de bloqueio é ‘Ack_b = 1’ e ‘Ack_c = 0’. Qualquer outra combinação de entradas não influenciará nas saídas devido à característica de histerese da célula C.

Como pode ser visto, a condição para modo de funcionamento transparente, ou seja a transmissão de dado, é a seguinte: o *latch* deve conter um dado válido em sua entrada (Ack_c = 1) e o seu sucessor deve conter o seu dado previamente armazenado (Ack_b = 0). Já a condição para bloqueio é caso seu sucessor já tenha recebido o dado previamente transmitido por ele, ou seja, o dado nas saídas do próximo *latch* é válido (Ack_b = 1), e o valor contido em sua entrada seja um dado vazio (Ack_c = 0) ele pode ser “resetado”. Estes dois modos apresentados sintetizam a tradução da regra citada acima.

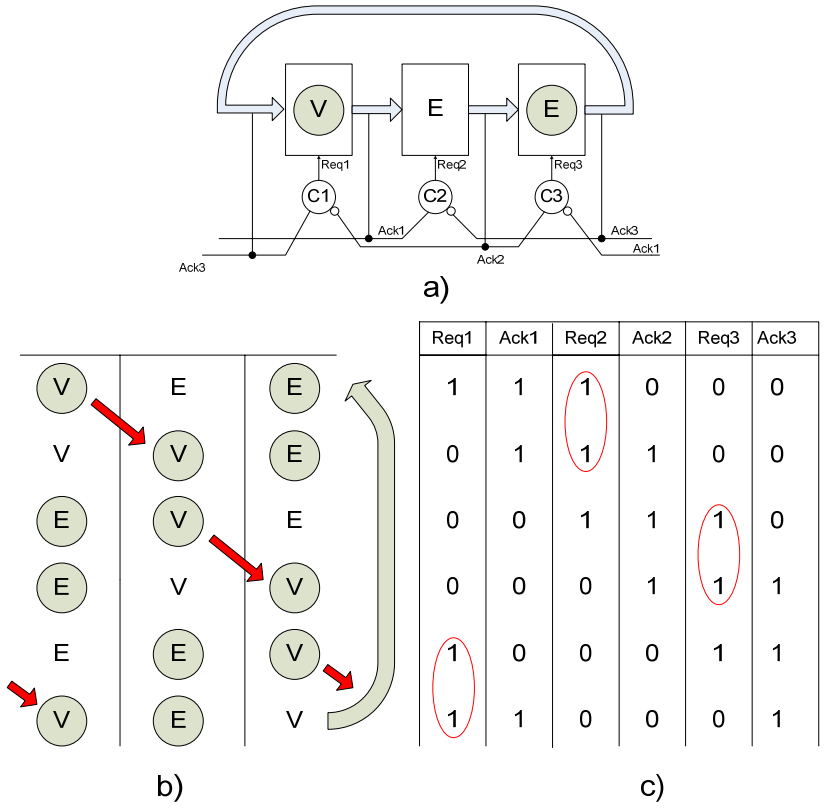


Figura 3.9 : Estrutura do anel assíncrono, diagrama de transferência de *tokens* e tabela de valores de sinais para cada etapa de comunicação.

A Figura apresentada acima (ver Figura 3.9) apresenta o fluxo de transmissão de dados gerado pelo circuito de controle construído. Este fluxo é obtido considerando-se a condição inicial em que 'L1' contém um dado válido com *token* e 'L3' contém um dado '*empty*' com *token*. 'L2' contém dado '*empty*' sem *token*. Em um primeiro momento é apresentada a estrutura que compõe o anel em conjunto com o sistema de controle (ver Figura 3.9a), em seguida um diagrama contendo o fluxo de transmissão de *tokens* (ver Figura 3.9b) e por último o valor de cada um dos sinais de controle no momento em que ocorre cada modificação de configuração do circuito (ver Figura 3.9c). As flechas no diagrama de *tokens* representam uma modificação de posição do *token* de dado válido e são correspondentes as modificações mostradas pelas elipses nos sinais de controle.

3.3 Inicialização

A inicialização do anel assíncrono é realizada através de dois sinais de controle que estão disponíveis ao usuário, o sinal de '*reset*' e o de '*set*'.

O primeiro passo na inicialização do circuito é colocar o sinal de '*reset*' em '0'. Quando esta condição é imposta ao circuito, todos os sinais de '*request*' são levados a '0' fazendo assim com que todos os *latches* modifiquem suas saídas para o valor '*empty*'. Desta forma é possível garantir que todos os sinais internos do circuito, ou seja, sinais nos barramentos, *latches* e blocos funcionais estão em nível lógico '0', conseqüentemente fazendo com que o anel fique parado esperando por um dado para processamento.

O sinal que inicia o protocolo de comunicação do anel propriamente dito é o sinal de '*set*'. Quando o sinal de '*set*' vai para '0', o *latch* 'L0' que é controlado por ele disponibilizará um dado válido no barramento. No momento em que o dado válido neste barramento é detectado pelo circuito de fim de cálculo 'FC1', o mesmo envia um sinal de '*acknowledge*' para o bloco de controle indicando assim que já existe um sinal de entrada válido no circuito. A partir deste momento o sistema de controle começa a funcionar realizando assim o protocolo de *handshake* que coordenará o funcionamento do mesmo até que seja atingido um estado de resolução do algoritmo executado pelo anel. O sinal '*set*' deve sempre ir para '0' somente após o sinal de '*reset*' ter ido para '1'.

Na Figura 3.10 é apresentada a estrutura do anel em conjunto com um diagrama contendo as formas de onda dos sinais de controle do circuito.

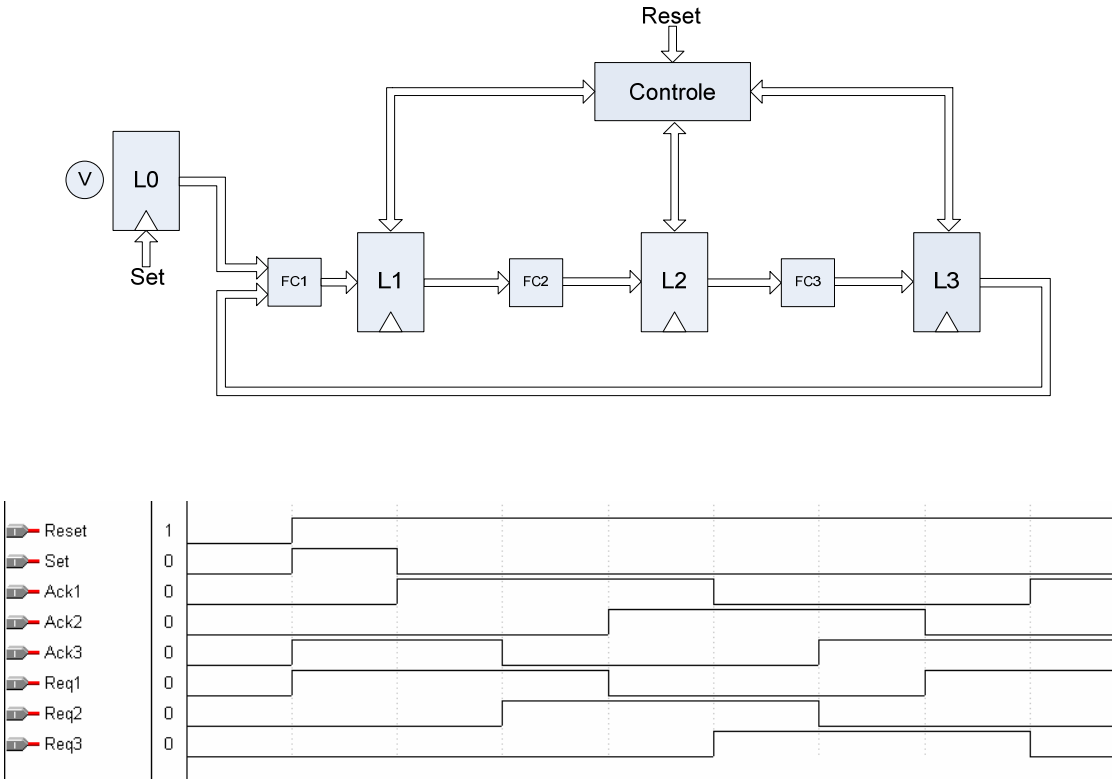


Figura 3.10 : Estrutura do anel básico e formas de onda dos sinais de controle na etapa de inicialização.

3.4 Aplicações

Nesta seção serão apresentas as aplicações propriamente ditas que foram construídas sobre anel assíncrono básico. Estas seis aplicações foram escolhidas levando em consideração primeiramente as que se adaptassem ao anel assíncrono básico com o menor número possível de modificações feitas a ele e em seguida para que fosse possível fazer um estudo comparativo do comportamento dos somadores quando submetidos a diferentes seqüências de vetores de entrada.

Em todas as aplicações houve uma perfeita junção entre a estrutura básica do anel assíncrono e o bloco lógico que faz com que haja uma diferenciação no propósito

final do circuito. Para que esta junção pudesse ser possível foram escolhidas aplicações que operassem dados através de laços de realimentação onde o próximo dado é calculado sobre o dado anterior. Esta característica da existência de realimentação nos algoritmos formadores das aplicações ficará mais nítida através da explicação dos circuitos que é dada a seguir nos próximos parágrafos.

Um primeiro esboço destas estruturas foi obtido a partir do exemplo MDC apresentado na referência (SPARSO, 2001) com modificações propostas pelo autor. As aplicações e estruturas seguintes (MMC, divisor, contador, etc...) são derivadas da estrutura da referência (SPARSO, 2001) com modificações no bloco lógico.

3.4.1 Contador

A aplicação Contador gera como resultado a variável de entrada 'a' incrementada de um valor 'n'. Para que isso seja possível, o anel assíncrono básico é modificado com a inclusão de alguns blocos como mostrado na Figura 3.11.

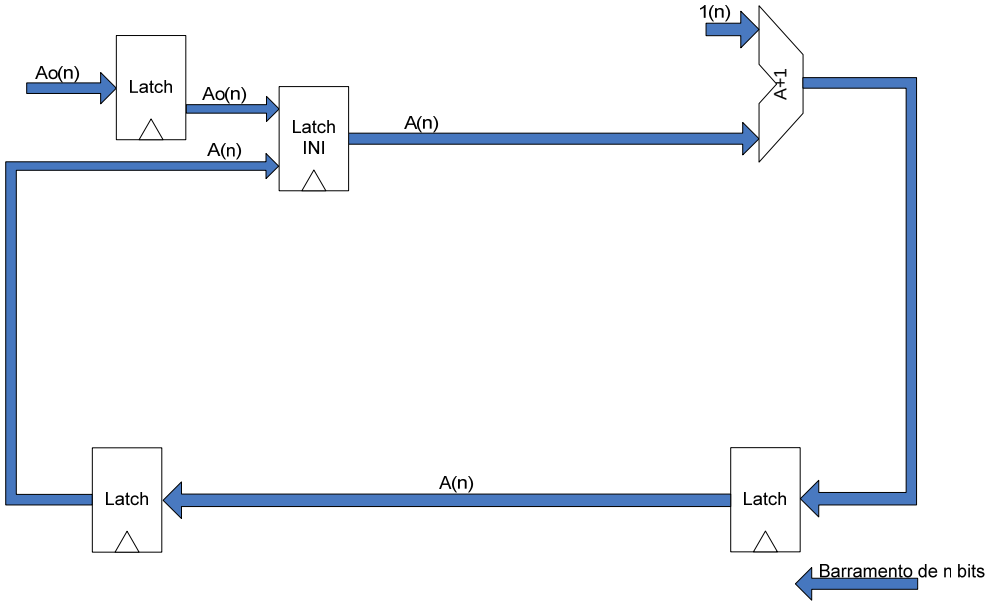


Figura 3.11 : Estrutura do anel que realiza a aplicação contador.

O bloco adicional consiste em um somador. Este bloco adicional é anexado ao circuito para que o algoritmo de formação do Contador, que consiste em:

```
- A recebe valor inicial;

- n recebe valor inicial;

- A recebe A + n;
```

possa ser executado de forma correta, ou seja:

O barramento de dados é inicializado com o dado ‘A’. A partir deste momento o somador executa a operação ‘A’ recebe ‘A + n’ sucessivamente até que o circuito seja resetado.

3.4.2 Divisor Inteiro

A aplicação Divisor Inteiro (DI) gera como resultado o divisor inteiro de dois números distintos ‘a’ e ‘b’ sendo que necessariamente ‘a’ ≥ ‘b’. Para que isso seja possível, o anel assíncrono básico é modificado com a inclusão de alguns blocos como mostrado na Figura 3.12.

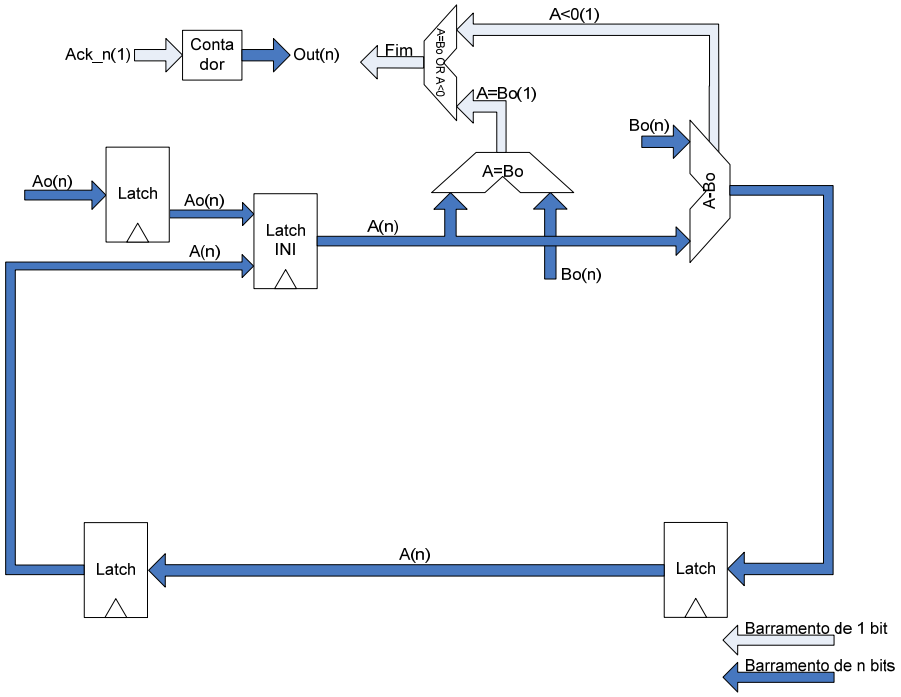


Figura 3.12 : Estrutura do anel que realiza a aplicação divisão inteira.

Estes blocos adicionais consistem em um somador, um circuito de comparação A ≤ B e um contador de ‘n’ bits, onde ‘n’ é o número de bits do circuito. Estes blocos adicionais são anexados ao circuito para que o algoritmo de formação do DI, que consiste em:

```
- Se A > B;

        - A recebe A - B

        - r recebe r + 1

- Senão;

     - Saída recebe r;
```

possa ser executado de forma correta, ou seja:

O barramento de dados é inicializado com o dado ‘A’. A partir deste momento o bloco ‘A ≤ B’ verifica se os dois valores são iguais ao mesmo tempo em que gera um pulso de relógio que incrementa o contador, caso a condição seja afirmativa, ele para o circuito e indica que o valor do contador ‘r’ é uma saída válida, caso contrário o anel segue transferindo o dado até que ele chegue ao somador. Neste ponto é executada a operação ‘A – B’.

O resultado da operação realizada pelo somador então é passada adiante e a condição ‘A ≤ B’ é verificada novamente e o ciclo se repete até que ‘A’ seja menor ou igual a ‘B’.

O resultado em si da operação é coletado nas saídas do contador. O contador é um circuito construído da forma usual através da utilização de *flip-flops* onde o sinal de relógio é substituído por um dos sinais de *acknowledge*. Como o sinal de *acknowledge* é gerado cada vez que um ciclo do anel é completado, ao contar o número de vezes que ele é fornecido ao contador é atingido o valor que é resultado da operação.

3.4.3 Divisor de Resto

A aplicação Divisor de Resto (DR) gera como resultado o resto da divisão de dois números distintos ‘a’ e ‘b’ sendo que necessariamente ‘a’ < ‘b’. Para que isso seja possível, o anel assíncrono básico é modificado com a inclusão de alguns blocos como mostrado na Figura 3.13.

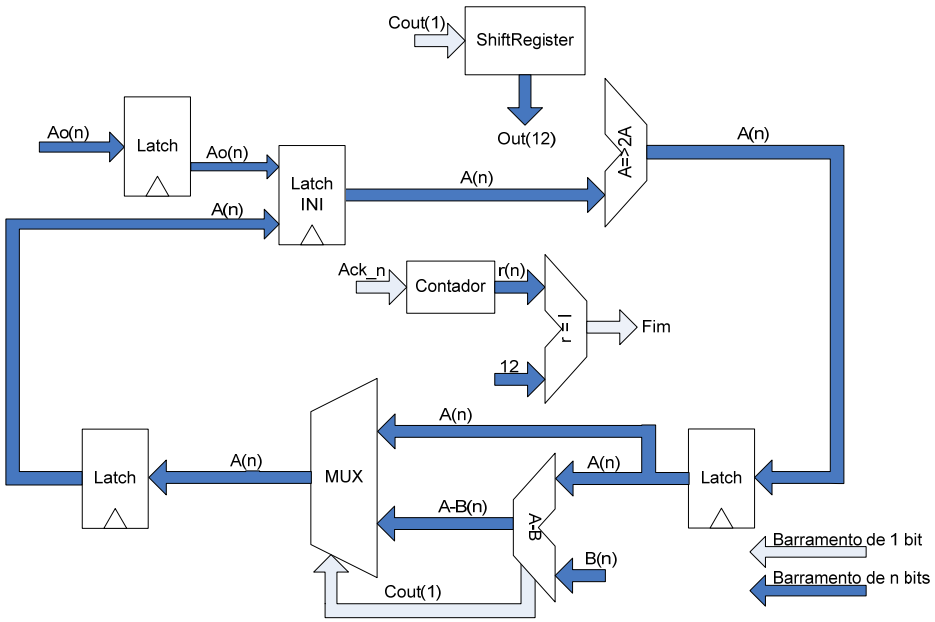


Figura 3.13 : Estrutura do anel que realiza a aplicação divisão de resto.

Estes blocos adicionais consistem em um somador, um multiplexador de duas entradas, um circuito de comparação ‘r = I’ e outro ‘A < B’, um circuito *shift left* (desloca o vetor para a esquerda), um *shift register* (registrador de deslocamento) e um contador de ‘n’ bits, onde ‘n’ é o número de bits do circuito. Estes blocos adicionais são anexados ao circuito para que o algoritmo de formação do DR, que consiste em:

```
- Se r ≠ I;

     - A recebe shift left (A)**(que é igual ao valor 2A)

     - r recebe r + 1

          - Se A > B;

               - A recebe A – B

          - Senão;

               - A recebe A

- Senão;

     Saída recebe r
```

possa ser executado de forma correta, ou seja:

O barramento de dados é inicializado com o dado 'A'. A partir deste momento o bloco 'r = I' (onde l é um valor entrado pelo usuário que indica a definição em bits do valor de saída, por exemplo, se forem amostrados 12bits na parte decimal, o valor de 'l' será igual a doze) verifica se os dois valores são iguais, caso a condição seja afirmativa, ele para o circuito e indica que o valor contido no *shift register* é uma saída válida, caso contrário o anel segue transferindo o dado até que ele chegue ao *shift left*. Neste ponto é executada a operação *shift left* (A) ao mesmo tempo em que o contador é incrementado. A partir deste ponto o somador realiza a operação 'A – B' e o resultado de *carry out* (que será o nosso fluxo de dados que forma o resultado) será armazenado no *shift register*, simultaneamente ao bloco que verifica a condição 'A < B'.

Na verdade esta é a condição que escolhe qual dos resultados passará adiante no circuito, pois o bloco 'A < B' controla o multiplexador que está colocado na saída do somador. O multiplexador tem em suas entradas o valor 'X' e o valor 'X – Y' sendo que se por acaso a condição 'A < B' for verdadeira, o multiplexador passará para o próximo bloco o valor 'A' e caso 'B'. A partir deste momento a condição 'r = I' é verificada novamente e o ciclo se repete até que 'r' seja igual a 'l'.

3.4.4 Mínimo Múltiplo Comum

A aplicação Mínimo Múltiplo Comum (MMC) gera como resultado o menor valor que é múltiplo de dois números distintos 'a' e 'b'. Para que isso seja possível, o anel assíncrono básico é modificado com a inclusão de alguns blocos como mostrado na Figura 3.14.

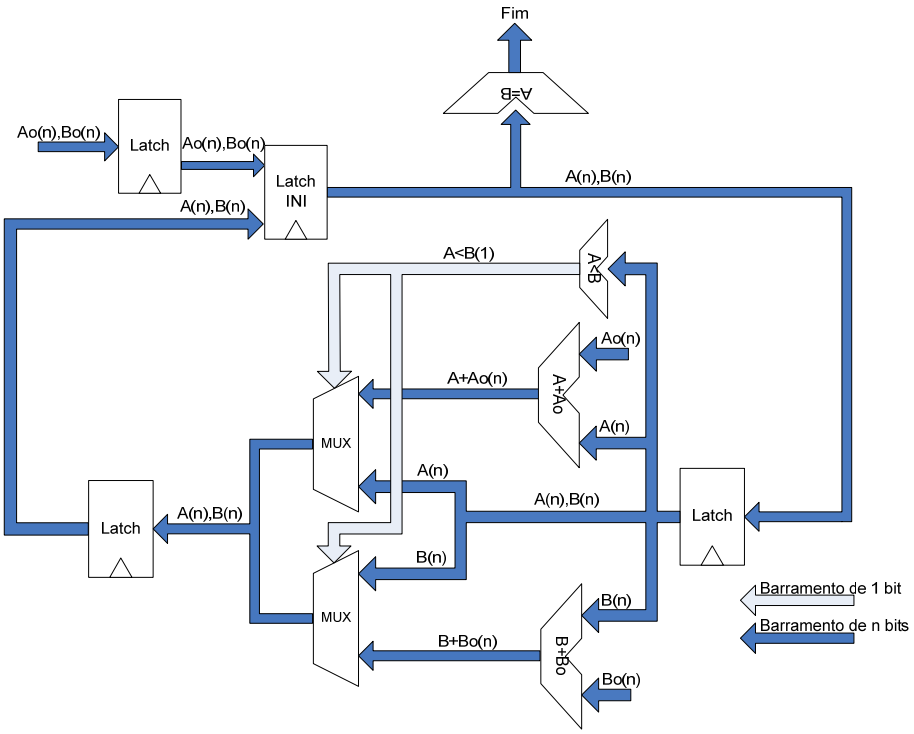


Figura 3.14 : Estrutura do anel que realiza a aplicação MMC.

Estes blocos adicionais consistem em dois somadores, dois multiplexadores de duas entradas, um circuito de comparação ‘A = B’ e outro ‘A < B’. Estes blocos adicionais são anexados ao circuito para que o algoritmo de formação do MMC, que consiste em:

```
- Se A ≠ B;
      - Se A < B;
            - A recebe A + Ao
            - B recebe B
      - Senão;
            - B recebe B + Bo
            - A recebe A
- Senão;
      - Saída (A);
```

possa ser executado de forma correta, ou seja:

Os barramentos de dados são inicializados com os dados ‘A’ e ‘B’, a partir deste momento o bloco ‘A=B’ verifica se os dois valores são iguais, caso afirmativo ele para o circuito e indica que o barramento que contem o valor ‘A’ é uma saída válida e caso contrário o anel segue transferindo os dados até que eles cheguem aos somadores. Neste ponto três tarefas são executadas simultaneamente, os dois somadores irão executar a função ‘X+Xo’ enquanto que o bloco ‘A<B’ verificará se esta condição é válida ou não.

Na verdade esta é a condição que escolhe qual dos resultados dos dois somadores passará adiante no circuito, pois o bloco ‘A<B’ controla os multiplexadores que estão colocados nas saídas dos somadores. Cada um dos dois multiplexadores tem em suas entradas o valor ‘X’ e o valor ‘X+Xo’ sendo que se por acaso a condição ‘A<B’ for verdadeira, o primeiro multiplexador passará para o próximo bloco o valor ‘A+Ao’ e o segundo multiplexador passará o valor ‘B’ e vice-versa. A partir deste momento a condição ‘A=B’ é verificada novamente e o ciclo se repete até que ‘A’ seja igual a ‘B’.

3.4.5 Máximo divisor comum

A aplicação Máximo Divisor Comum (MDC) gera como resultado o maior valor que é divisor de dois números distintos ‘a’ e ‘b’. Para que isso seja possível, o anel assíncrono básico é modificado com a inclusão de alguns blocos como mostrado na Figura 3.15.

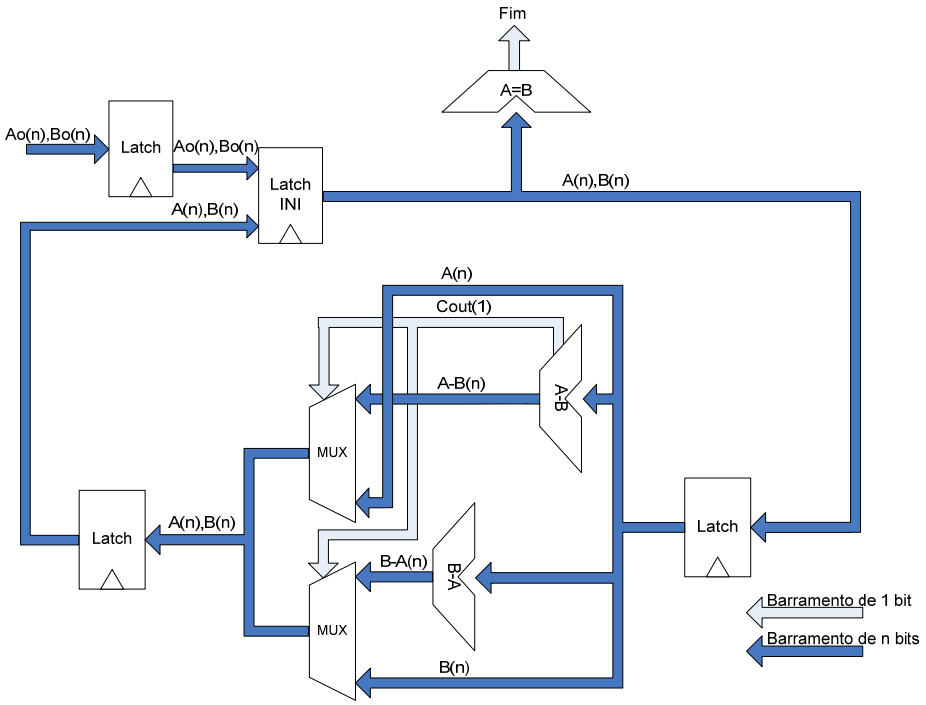


Figura 3.15 : Estrutura do anel que realiza a aplicação MDC.

Estes blocos adicionais consistem em dois somadores, dois multiplexadores de duas entradas, um circuito de comparação ‘A=B’ e outro ‘A<B’. Estes blocos adicionais são anexados ao circuito para que o algoritmo de formação do MDC, que consiste em:

```
- Se A ≠ B;
     - Se A < B;
          - A recebe A - B
          - B recebe B
     - Senão;
          - B recebe B - A
          - A recebe A
- Senão
     - Saída (A);
```

possa ser executado de forma correta, ou seja:

Os barramentos de dados são inicializados com os dados ‘A’ e ‘B’, a partir deste momento o bloco ‘A=B’ verifica se os dois valores são iguais, caso afirmativo ele pára o circuito e indica que o barramento no qual é contido o valor ‘A’ é uma saída válida e caso contrário o anel segue transferindo os dados até que eles cheguem aos somadores. Neste ponto três tarefas são executadas simultaneamente, os dois somadores irão executar a função ‘X – Y’ enquanto que o bloco ‘A<B’ verificará se esta condição é válida ou não.

Na verdade esta é a condição que escolhe qual dos resultados dos dois somadores passará adiante no circuito, pois o bloco ‘A<B’ controla os multiplexadores que estão colocados nas saídas dos somadores. Cada um dos dois multiplexadores tem em suas entradas o valor ‘X’ e o valor ‘X – Y’ sendo que se por acaso a condição ‘A<B’ for verdadeira, o primeiro multiplexador passará para o próximo bloco o valor ‘A - B’ e o segundo multiplexador passará o valor ‘B’ e vice-versa. A partir deste momento a condição ‘A=B’ é verificada novamente e o ciclo se repete até que ‘A’ seja igual a ‘B’.

3.4.6 Raiz Quadrada

A aplicação Raiz Quadrada (RQ) gera como resultado a raiz quadrada inteira de um número qualquer ‘a’. Para que isso seja possível, o anel assíncrono básico é modificado com a inclusão de alguns blocos como mostrado na Figura 3.16.

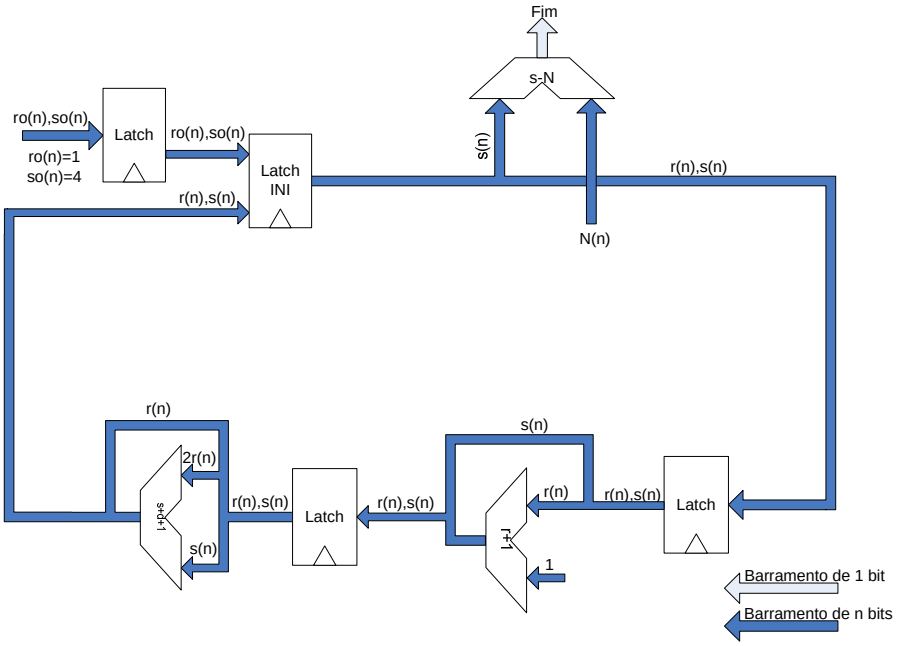


Figura 3.16 : Estrutura do anel que realiza a aplicação raiz.

Estes blocos adicionais consistem em dois somadores, um circuito de comparação ‘A ≤ S’ e um circuito de *shift left*. Estes blocos adicionais são anexados ao circuito para que o algoritmo de formação da RQ, que consiste em:

```
R recebe 1;

S recebe 4;

- Se A > S;

     - R recebe R + 1

     - D recebe shift left (R)

     - S recebe S + D + 1

- Senão;

      Saída recebe R
```

possa ser executado de forma correta, ou seja:

Os barramentos de dados são inicializados com os valores ‘1’ e ‘4’. A partir deste momento o bloco ‘A ≤ S’ verifica se esta condição é verdadeira, caso afirmativo ele pára o circuito e indica que o barramento que contém o valor ‘R’ é uma saída válida e caso contrário o anel segue transferindo os dados até que eles cheguem ao primeiro somador. Neste ponto o barramento ‘R’ receberá o valor ‘R + 1’.

Em seguida, o próximo somador será configurado com as seguintes entradas, S, D (que é igual à *shifl left* (R)) e seu *carry in* é colocado em ‘1’. A partir desta configuração ele executará a operação aritmética ‘S = S + D + 1’. Deste momento em diante a condição ‘A ≤ S’ é verificada novamente e o ciclo se repete até que ‘A’ seja menor ou igual a ‘S’.

4 Somadores *Riple Carry* (RCA)

Neste capítulo serão apresentadas algumas de famílias lógicas CMOS comumente utilizadas em projetos de blocos funcionais com detecção de fim de cálculo. Sendo tal propriedade essencial para o projeto de circuitos assíncronos. Estas famílias serão utilizadas na construção de somadores RCA (*Ripple Carry Adder*) com a finalidade de obter os dados necessários para que se possa fazer um estudo comparativo entre elas assim como gerar conhecimento para uma análise da validade da utilização das mesmas na construção de circuitos mais complexos (como, por exemplo, multiplicadores e filtros digitais).

Em um primeiro momento será discutido o funcionamento e diferenças das famílias lógicas tratadas aqui. Em seguida será apresentado o diagrama de blocos utilizado na construção dos circuitos RCA e a construção de tais somadores utilizando cada uma das estruturas CMOS. Por fim são mostrados os resultados de simulação das mesmas em conjunto com uma tabela comparativa com os resultados de todas as simulações feitas e uma análise dos mesmos.

4.1 Somador RCA

Talvez a forma mais simples de se implementar uma soma seja através da técnica RCA. O somador RCA obedece as equação mostradas abaixo e tem o diagrama de blocos apresentado na Figura 4.1. Ele nada mais é do que o "cascateamento" de 'n' *full-adders* onde 'n' é o número de bits da soma.

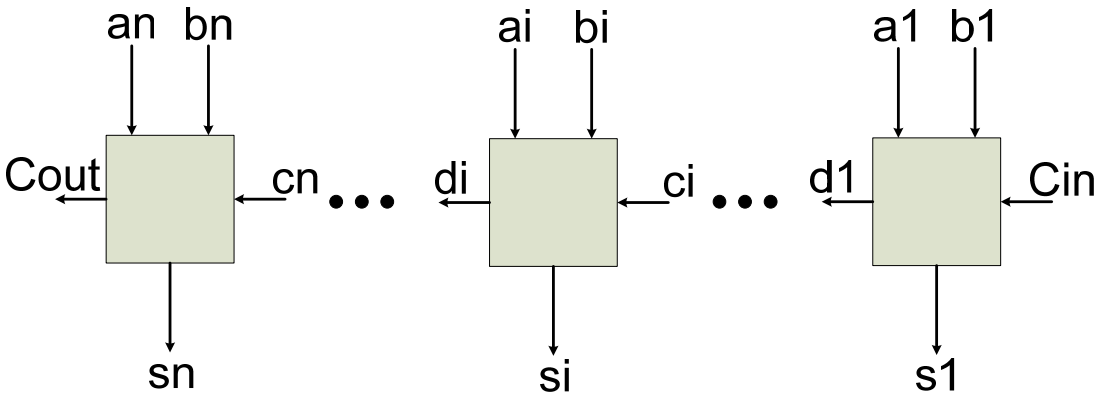


Figura 4.1 : Diagrama de blocos de um somador RCA de tamanho "n".

Equações gerais do RCA:

$s = a \oplus b \oplus c$

$d = a.b + a.c + b.c$

d = *carry-out* de cada bloco

Quando o número de bits da soma é pequeno, ele apresenta uma área relativamente pequena e um bom desempenho. A sua principal deficiência aparece quando partimos para a construção de circuitos com grande número de bits. Como esta estrutura é construída a partir do cascateamento de vários blocos de *full-adder*, acaba-se gerando uma cadeia de *carry* onde a geração do *carry* de nível 'n' depende da conclusão da computação do seu antecessor 'n-1'.

À medida que o número de bits cresce o tamanho da cadeia também cresce fazendo assim com que, no pior caso, se os vetores de entrada forem tais que todos os blocos *full-adders* tenham que propagar o sinal de *carry* até a saída, o atraso de computação será proibitivamente alto.

4.2 Estruturas lógicas CMOS

Todas as estruturas aqui apresentadas obedecem ao protocolo *dual-rail*, portanto estando em conformidade com o protocolo utilizado nas estruturas básicas assíncronas. Quatro das famílias apresentadas são estáticas (as saídas respondem imediatamente após as entradas estarem disponíveis) e consistem em: soma de produtos (SDP), *delay insensitive minterm synthesis* (DIMS) (SPARSØ, 93), (SPARSØ, 92) *null convention logic* (NCL) (FANT, 96), (SOBELMAN, 2004), (FANT, 2004) e uma estrutura CMOS *dual-rail* apresentada por Martin (MARTIN, 92) que neste trabalho será referenciada como sendo técnica "Martin". As outras três são dinâmicas (as saídas são válidas somente no período de avaliação) e são elas *differential cascode voltage switch logic* (DCVS) (CHU, 86), (HELLER, 84), *enable/disable CMOS differential logic* (ECDL) (LU, 88) e *differential pass transistor logic* (DPTL) (YANO, 89). A descrição das mesmas é apresentada a seguir.

4.2.1 Soma-de-Produtos (SDP)

A SDP é uma técnica de construção de circuitos bastante conhecida que gera circuitos que combinam um conjunto de portas lógicas AND e OR em uma estrutura parecida com uma matriz (ERCEGOVAC, 99). Os circuitos obtidos a partir desta técnica são gerados através da análise da tabela verdade da função. Por exemplo, na Figura 4.2a é apresentada a tabela verdade de uma porta XOR: para cada valor lógico '1' que é apresentado na coluna de saída é gerado um termo da equação do circuito onde um '0' representa um termo de entrada negado e um '1' um termo de entrada direto Figura 4.2b. A partir da obtenção desta equação, o circuito resultante é apresentado na Figura 4.2c.

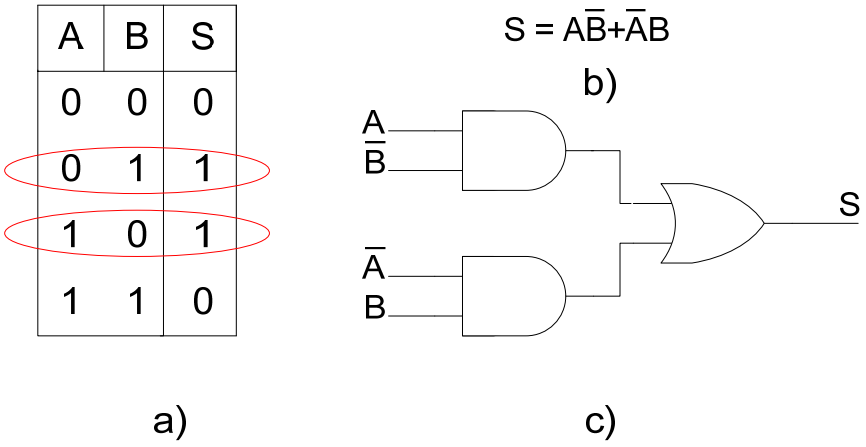


Figura 4.2 : a) Tabela verdade, b) equação gerada e c) circuito resultante do uso da técnica SDP.

Para que um circuito possa ser considerado um bloco funcional *dual-rail*, ele deve obedecer à codificação de dados. Ou seja, ter entradas e saídas diferenciais para cada bit de dado e obedecer à regra de transição entre dados que consiste na existência de um valor '*empty*' entre a transição de dois dados consecutivos. Para que isso possa ser concretizado, a tabela verdade da Figura 4.2a é modificada para a forma mostrada na Figura 4.3a, que gerará duas equações de saída, uma para o valor '0' da saída 'Sf' e uma para o '1' da mesma 'St' (ver Figura 4.3b); o circuito final gerado é apresentado na Figura 4.3c).

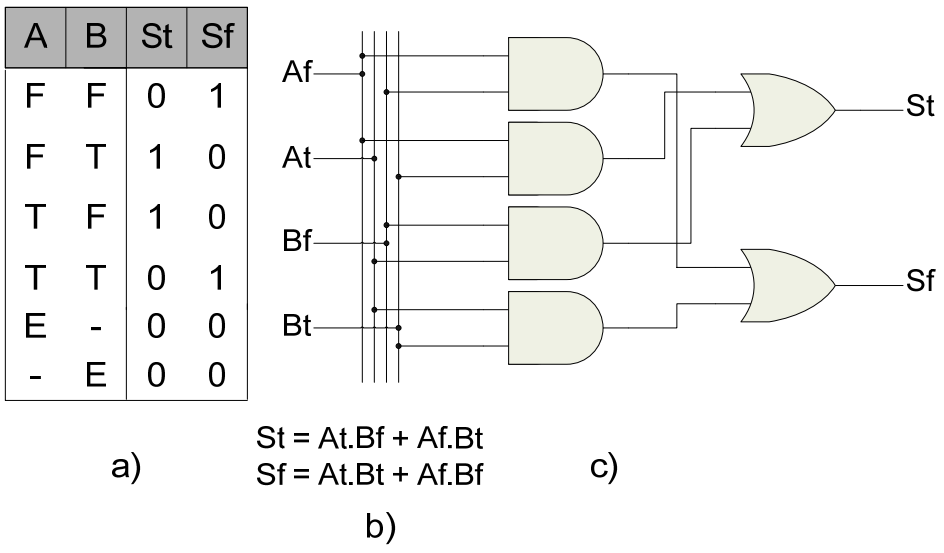


Figura 4.3 – a) Tabela verdade, b) equações *dual-rail* gerada e c) circuito gerado através da técnica SDP.

Como se pode notar, o circuito aparentemente obedece aos critérios necessários para que funcione de forma assíncrona. Para que essa condição possa ser verificada, é

feito uma análise do comportamento do mesmo quando submetido a uma seqüência de dados.

Considerando-se que o circuito está com todas as entradas, saídas e nós intermediários em '*empty*' (Figura 4.4a), se em um primeiro momento o dado 'A' for para '*true*' (At=1, Af=0) e todas as outras entradas se mantiverem em '0', as saídas não são modificadas Figura 4.4b. A partir do momento em que 'B' vai para '*false*' (Bt=0, Bf=1), as saídas são definidas com dados válidos (Figura 4.4c). O que está correto, pois de acordo com os princípios de funcionamento deste tipo de circuito assíncrono, as saídas só podem se tornar dados válidos a partir do momento em que todas as entradas contêm dados.

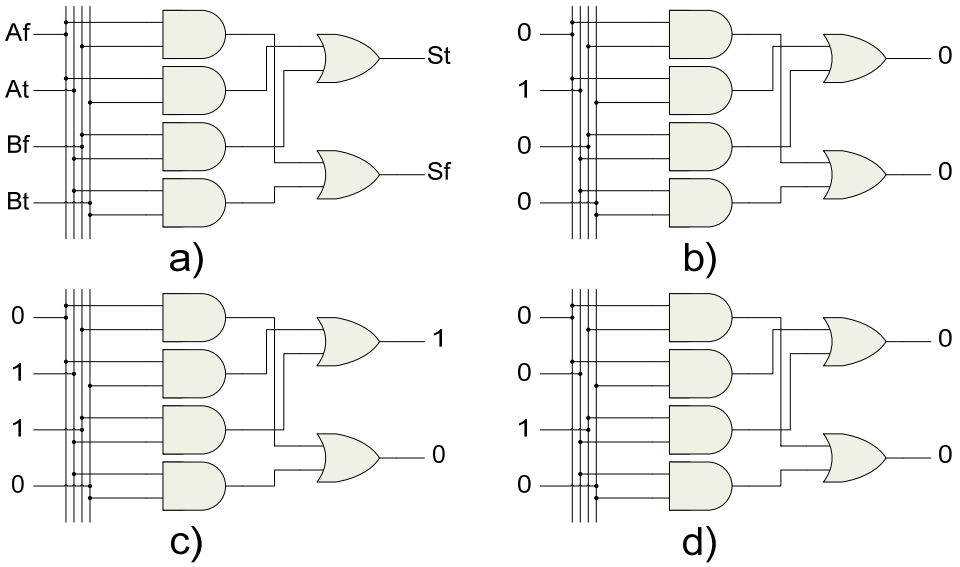


Figura 4.4 : Verificação de falha da validade da técnica SDP na construção de circuitos auto-temporizados.

Neste momento passamos à análise do comportamento do circuito quando passamos em uma transição de dado válido para inválido. A partir do momento em que, por exemplo, 'At' recebe um valor '0' em sua entrada, a saída vai para '*empty*' Figura 4.4d. Este comportamento não é válido do ponto de vista de circuitos assíncronos, pois a mesma lei que foi citada no capítulo 2 na parte sobre blocos funcionais, que diz que todas as saídas só podem ir para '*empty*' depois que todas as entradas estão neste estado, o que não acontece na situação da Figura 4.4d, pois 'B' ainda contém um dado válido.

Como esta variação de 'At' poderia ser uma variação transiente, fazendo com que após alguns instantes esta entrada voltasse para '1', o que indica para o resto do sistema que um dado novo havia chegado quando na verdade isto não havia ocorrido, teríamos a ocorrência de uma transição espúria. Como foi explicado anteriormente que circuitos assíncronos devem ser *hazard-free* esta técnica logo não serve para a construção dos mesmos.

Uma forma de evitar este problema de indicação do valor '*empty*' seria substituir as portas AND por células C. Como as células C apresentam um comportamento de histerese que faz que o estado da saída só mude para '1' quando ambas as entradas estão

em ‘1’ e para ‘0’ quando ambas estão em ‘0’ o problema citado anteriormente desaparece. Esta substituição das portas AND por células C é o principio de funcionamento da técnica DIMS que é apresentada a seguir.

4.2.2 Delay Insensitive Minterm Synthesis (DIMS)

Blocos funcionais construídos a partir da técnica DIMS (SPARSØ, 93), (SPARSØ, 92), são projetados de forma muito semelhante à construção de circuitos combinacionais através de SDP. A principal diferença reside na substituição da porta AND por uma célula C. A célula C é responsável pela sincronização dos sinais de entrada do bloco funcional e pela geração da lógica de *reset*, que é executada quando a mesma recebe um valor de entrada ‘*empty*’.

Circuitos DIMS podem ser construídos a partir do mapeamento direto de SDP. Esta característica acaba se tornando importante, pois faz com que sua construção possa ser automatizada a partir da utilização de ferramentas de CAD já existentes, com apenas pequenas modificações.

Na Figura 4.5 é apresentado o processo de construção de um bloco funcional que executa a operação lógica XOR. A partir da descrição da operação mencionada através de sua tabela verdade (ver Figura 4.5a), são verificadas as combinações de entrada que geram um valor de saída igual a ‘1’, como por exemplo, entrada A com valor ‘1’ (At=1, Af=0) e entrada B com valor ‘0’ (Bt=0, Bf=1) fazem com que saída se iguale a ‘1’, ou seja, o bit ‘St=1’ e ‘Sf=0’ (Figura 4.5a e b). A partir deste momento todas as combinações que gerem St=1 são agrupadas através de uma porta OR assim como todas que geram Sf =1 (Figura 4.5b), completando assim o processo de construção do operador lógico.

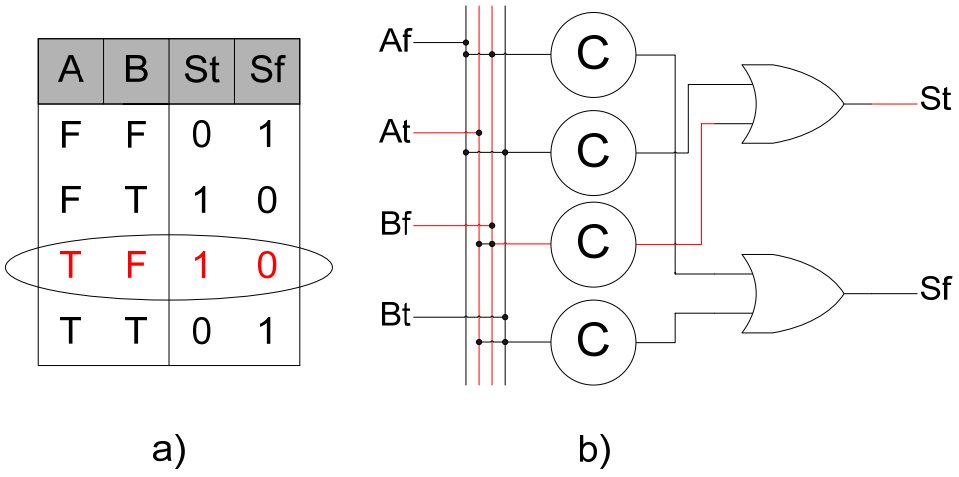


Figura 4.5 : Portas DIMS; a) Tabela verdade da função; b) Circuito gerado a partir da resolução da mesma.

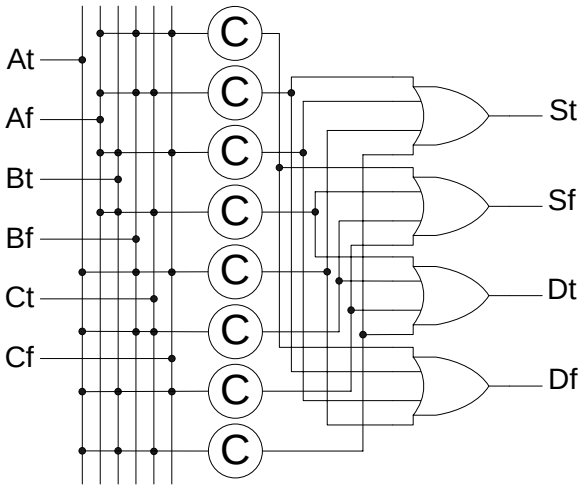
A principal vantagem dos circuitos gerados a partir desta técnica é que eles podem ser considerados praticamente *delay-insensitive*. A desvantagem é a área exagerada ocupada pelo circuito. Neste caso, para construir uma porta XOR foram utilizadas 4 células C e duas portas OR, totalizando em torno de 44 transistores, ou seja um aumento de aproximadamente 5 vezes relativo a área de uma porta XOR normal.

Para circuitos maiores a tendência é que o aumento de área relativo à versão combinacional dos circuitos seja menor, mas ainda assim da ordem de 2 a 3 vezes.

A estrutura *full-adder* DIMS é gerada diretamente através de uma técnica semelhante à utilizada em SDP. A partir da tabela verdade a equação é mapeada e a estrutura é construída com a substituição das portas AND por células C gerando o circuito apresentado na Figura 4.6.

A	B	C	St	Sf	Dt	Df
F	F	F	0	1	0	1
F	F	T	1	0	0	1
F	T	F	1	0	0	1
F	T	T	0	1	1	0
T	F	F	1	0	0	1
T	F	T	0	1	1	0
T	T	F	0	1	1	0
T	T	T	1	0	1	0

a)



b)

Figura 4.6 : a) Tabela verdade *dual-rail* de um *full-adder*; b) Circuito DIMs.

4.2.3 Null Convention Logic (NCL)

NCL é uma lógica de construção de circuitos funcionais assíncronos que foi patenteada pela empresa Theseus *Logic* (FANT, 96), (SOBELMAN, 2004), (FANT, 2004). Seu princípio de funcionamento é baseado na técnica de construção de circuitos DIMS sendo que circuitos NCL podem ser construídos a partir de mapeamento direto da mesma.

Circuitos NCL seriam iguais a DIMS caso não tivessem inserido nesta técnica novos tipos de portas lógicas. Estas novas portas lógicas são como células C que obedecem a uma condição. Uma célula C tradicional só modifica o valor de suas saídas para ‘0’ ou para ‘1’ quando todas as suas entradas estão em ‘0’ ou ‘1’ respectivamente. As novas portas inseridas em NCL são chamadas de *threshold gates*.

A condição, citada anteriormente, é que elas são modificadas a ponto de que não seja necessário que todas as entradas tenham de ir para ‘1’ para que a saída vá para ‘1’, sendo que o comportamento delas no que se refere à transição para ‘0’ continua o mesmo da célula C. Isto faz com que seja gerado uma nova gama de células to tipo ‘2 de 3’, ‘2 de 4’, ‘3 de 5’, etc. Onde, considerando-se uma notação ‘m de n’, ‘m’ significa o número de entradas que têm de ir para ‘1’ para que a saída vá para ‘1’ e ‘n’ significa o número de entradas da porta sendo que $m \leq n$ (na Figura 4.7 é apresentado o diagrama elétrico de uma porta semi-estática ‘2 de 3’).

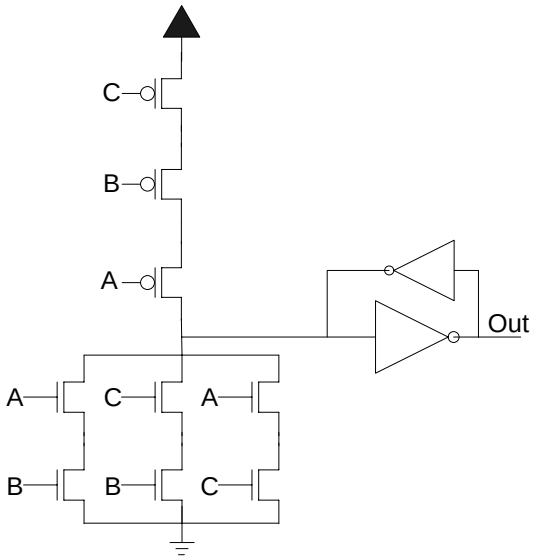
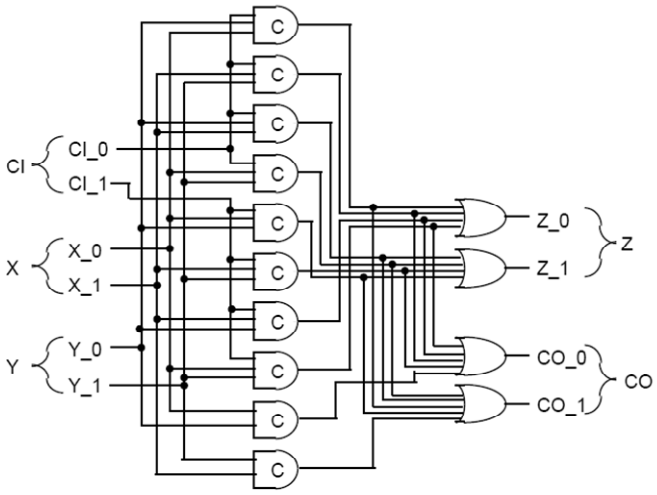


Figura 4.7 : Threshold gate 2 de 3 (FANT, 2004).

Esta nova gama de portas lógicas traz o benefício que a partir de uma descrição de uma equação qualquer, como por exemplo, a equação geral de formação de um *full-adder* para o somador RCA, que em DIMS seria mapeada da forma apresentada na Figura 4.8a, a partir de mapeamento direto para NCL é apresentada na Figura 4.8b, possa ser otimizada de forma a gerar a descrição compacta apresentada na Figura 4.8c, como proposto em (FANT, 2004).



a)

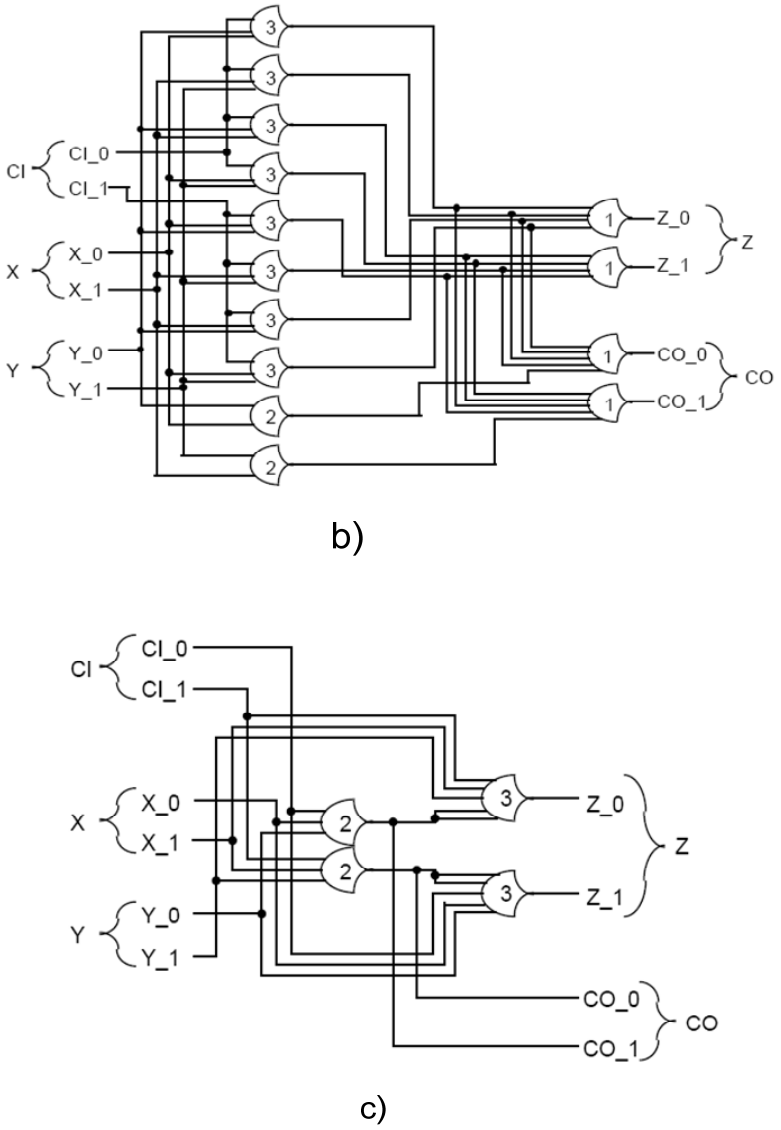


Figura 4.8 : a) *Full-adder* DIMS; b) conversão para NCL; c) circuito resultante após otimizações (FANT, 2004).

4.2.4 Martin

Blocos funcionais construídos a partir da técnica proposta por Alain Martin seguem o algoritmo descrito em seu trabalho (MARTIN, 92). Os circuitos resultantes consistem em uma árvore NMOS responsável pela avaliação da função em conjunto com uma série de transistores PMOS que implementam a lógica de *reset* da função.

A implementação do somador completo de um bit é apresentada na Figura 4.9. Como se pode ver no circuito que irá calcular o valor do *carry-out* são utilizados quatro transistores do tipo PMOS em série para um circuito de apenas duas entradas, o que acaba limitando o número de entradas para que não haja degradação de desempenho. Como os transistores do tipo PMOS são utilizados somente para geração da lógica de *reset*, uma possível solução é substituir estes transistores por somente um, que seria então controlado pelo sinal de *request* do sistema, gerando assim um circuito dinâmico muito parecido com os que se utilizam de pré-carga, como a estrutura DCVS apresentada a seguir.

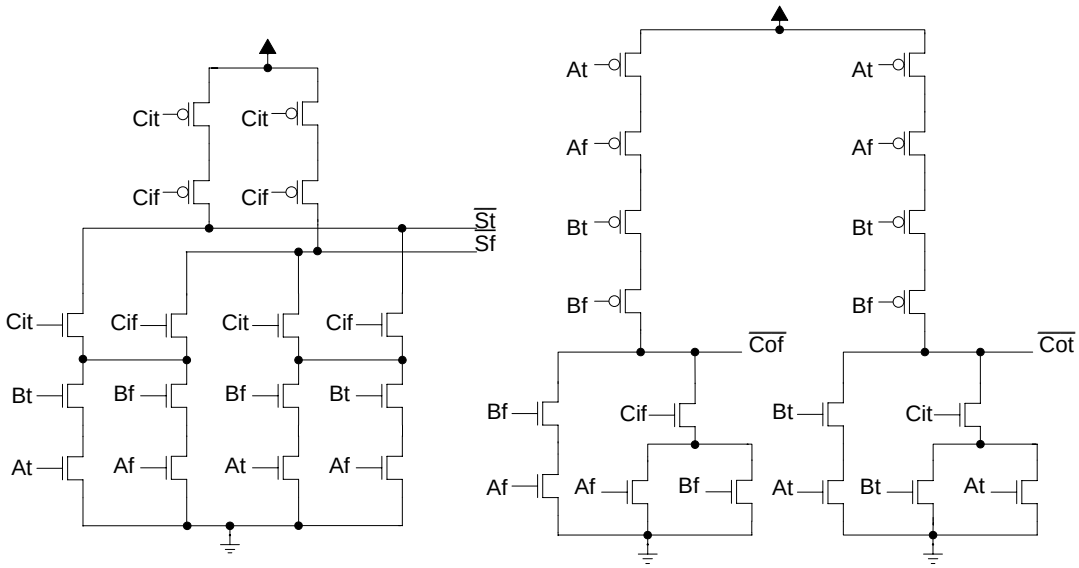


Figura 4.9 : Diagrama elétrico do full-adder gerado a partir da técnica Martin (MARTIN, 92).

4.2.5 Differential Cascode Voltage Switch Logic (DCVS)

A estrutura DCVS pode ser dividida em duas partes, o circuito de pré-carga e a árvore de decisão binária (ou árvore N), como mostrado na Figura 4.10. Este tipo de implementação de circuito lógico funciona obedecendo a duas fases distintas de funcionamento, um período de *reset* ou pré-carga e um período de avaliação, controladas por uma variável externa.

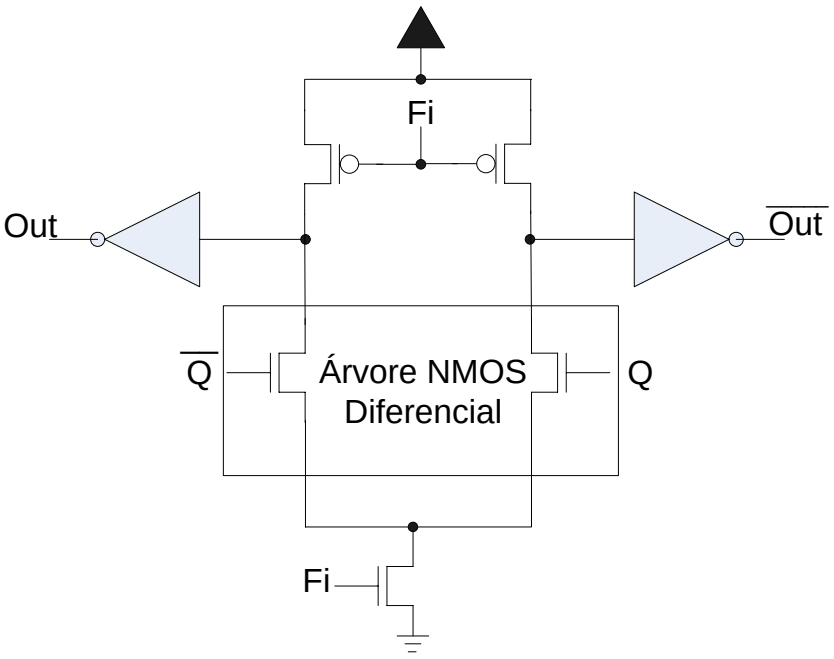


Figura 4.10 : Diagrama elétrico do circuito de pré-carga DCVS.

De acordo com (CHU, 86) e (HELLER, 84), a vantagem mais explícita desta técnica consiste na redução do número de dispositivos utilizados. Outra vantagem que pode ser citada aqui é o aumento da flexibilidade lógica obtida, principalmente quando estamos tratando de funções complexas a serem implementadas.

A árvore binária é sempre ativada no período de avaliação, ou seja, quando o circuito recebe um sinal de controle com nível lógico alto '1', sendo então responsável pela execução da função propriamente dita. A mesma é projetada de modo que: 1) quando o vetor de entrada $x=(x_1,...,x_n)$ é o valor verdadeiro da função de chaveamento Q(x), o nó Q é desconectado do *ground* ao mesmo tempo que o nó $\overline{Q}$ é conectado a *ground* através de um caminho único de condução existente na árvore; e 2) e quando a entrada $x=(x_1,...,x_n)$ for o valor falso de Q(x), o contrário acontece.

Por exemplo, a árvore NMOS apresentada na Figura 4.10 executa a função inversora (Q(x) = inversor). Se o vetor x for igual a '1', o nó Q é conectado a GND e $\overline{Q}$ desconectado, assim avaliando a função. Esta operação resulta em OUT = 0.

Um exemplo simples da construção da árvore de decisão binária pode ser ilustrado pela construção de uma porta XOR em DCVS (Figura 4.11). A funcionalidade do circuito pode ser facilmente verificada através do teste de todas as combinações de entradas possível.

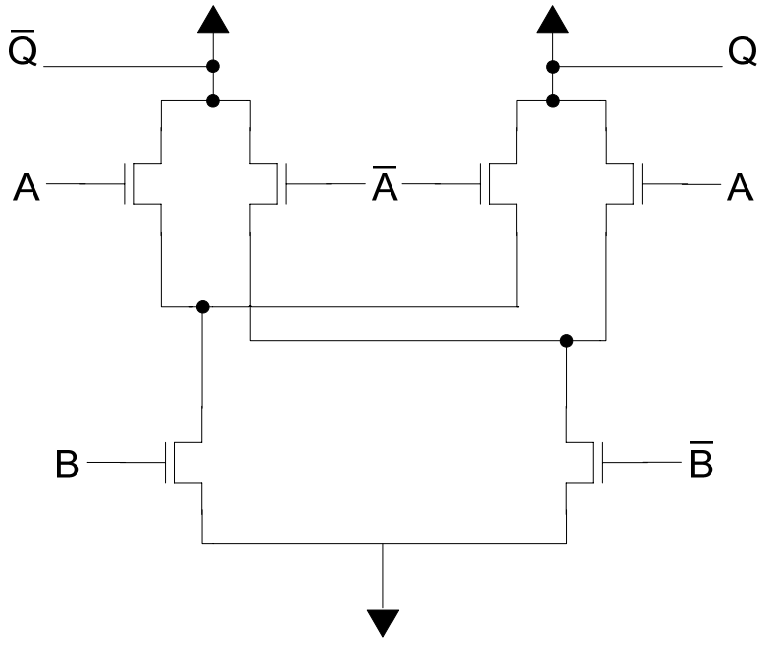


Figura 4.11 : Diagrama elétrico da árvore NMOS que representa a função XOR.

Já o circuito de pré-carga é composto de: transistores do tipo PMOS que conectam os nós Q e $\overline{Q}$ a Vdd (*pull-up*); dois inversores que estabilizam o valor de saída da função e aceleram a detecção do nó que será descarregado; e um transistor do tipo NMOS que conecta a árvore binária ao *ground*, com a finalidade de evitar curto-circuitos quando o mesmo se encontra em período de transição entre pré-carga e avaliação e vice-versa.

O princípio de funcionamento (Figura 4.12) deste circuito é o seguinte: quando o sinal de controle ‘fi’ está em nível lógico baixo ‘0’, o transistor ‘n3’ é cortado, desconectando assim qualquer caminho possível entre Vdd e gnd, os transistores ‘p1’ e ‘p2’ estão ativos assim conectando Q e $\overline{Q}$ a Vdd fazendo com que as saídas F e $\overline{F}$ permaneçam zeradas Figura 4.12a. A partir do momento em que o sinal de controle ‘fi’ vai para nível lógico alto ‘1’, o circuito entra em período de avaliação, os transistores ‘p1’ e ‘p2’ são cortados e o transistor ‘n3’ passa a conduzir, fazendo assim com que um dos nós Q ou $\overline{Q}$ sejam descarregados através da função lógica implementada pela árvore binária, dessa forma fazendo com que se tenha uma saída em nível lógico alto e outra em baixo, Figura 4.12b.

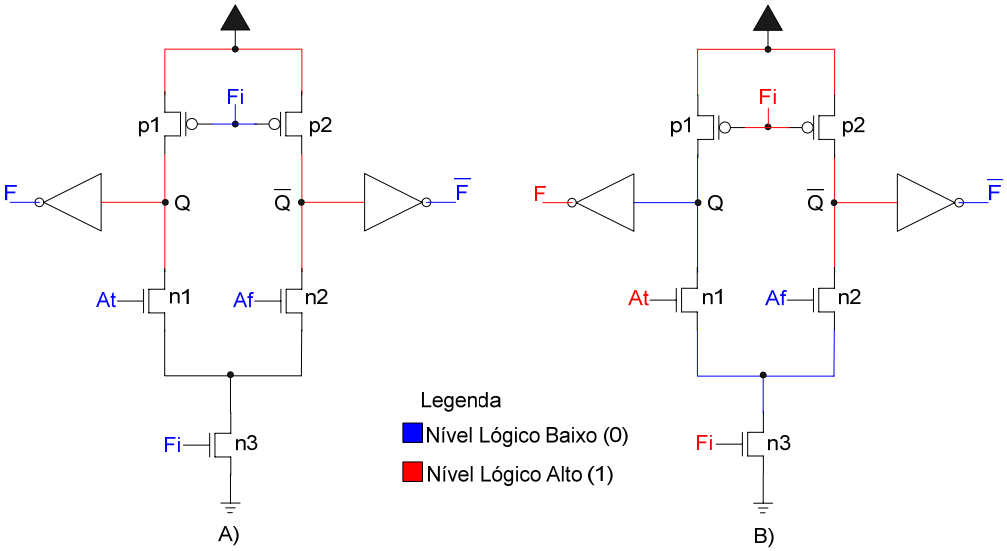


Figura 4.12 : Demonstração do princípio de funcionamento de circuitos DCVS.

A estrutura *full-adder* DCVS (NG, 96) foi gerada através da conversão da equação original do mesmo em uma árvore de transistores NMOS ligados a circuitos de pré-carga controlados por um sinal de controle ‘fi’. Na Figura 4.13 é apresentada à estrutura da árvore NMOS utilizada na construção do bloco *full-adder* que compõe os somadores projetados. Os nós negados indicados na figura assim como o nó massa são ligados ao circuito de pré-carga.

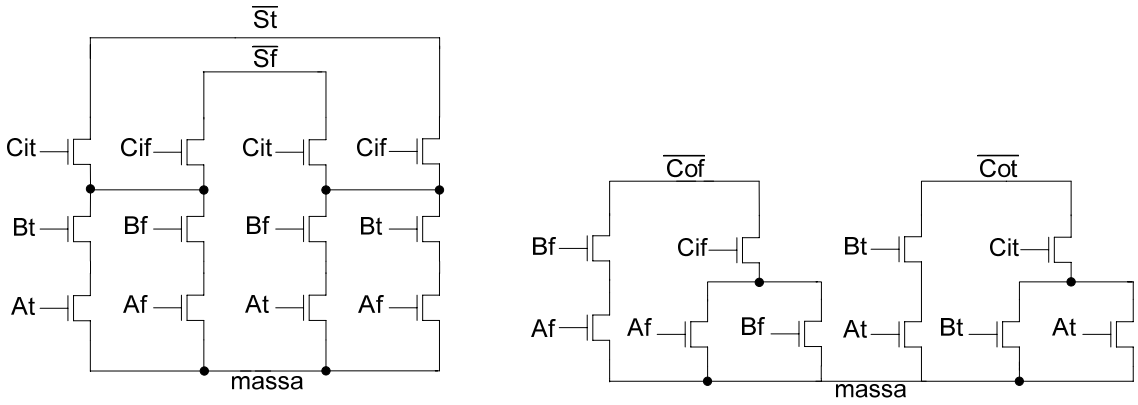


Figura 4.13 : Diagrama elétrico da árvore NMOS que descreve a função *full-adder*.

4.2.6 Enable/Disable CMOS differential Logic (ECDL)

A lógica ECDL foi primeiramente proposta por Lu em (LU, 88). Ela também consiste em uma técnica composta por dois períodos de funcionamento, avaliação e *reset* (neste caso pré-descarga), controlados por um sinal externo. Assim como nos circuitos DCVS, em ECDL temos duas partes distintas constituindo o circuito (ver Figura 4.14): a árvore binária (que no caso obedece as mesmas regras de formação e funcionamento da explicada na seção anterior (DCVS)) e por um circuito do tipo *sense-amplifier*.

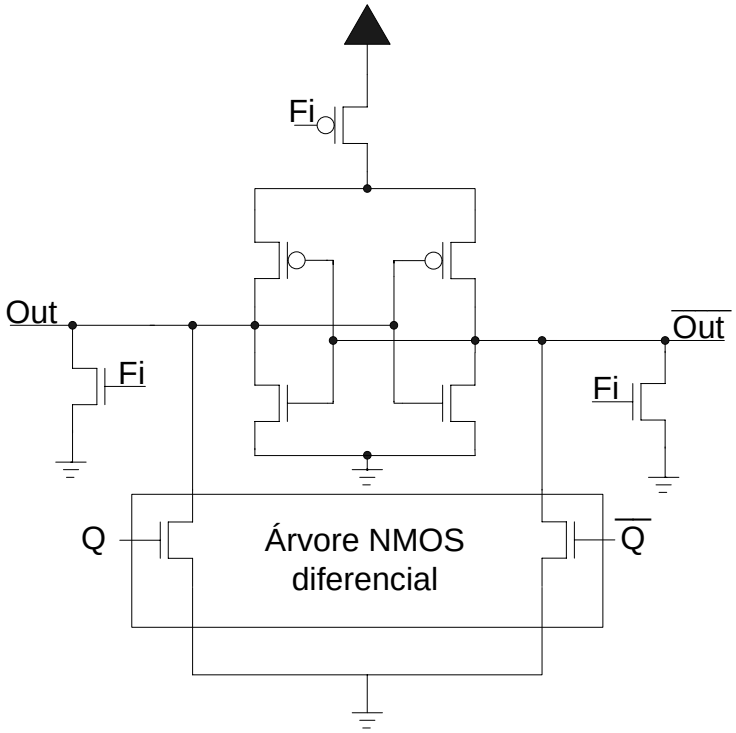


Figura 4.14 : Diagrama elétrico do circuito *sense-amplifier* ECDL.

Uma vantagem desta técnica reside na característica de que implementações ECDL são totalmente estáticas, uma vez que não existe uma condição de valor mínimo de período para o sinal de controle (LU, 88).

Circuitos construídos a partir desta técnica requerem que todas as entradas contenham valores válidos e estáveis no momento em que recebem um sinal de controle ‘fi’. Isto faz com que no momento em que exista um cascateamento de blocos ECDL se torne necessário que cada bloco gere o sinal de ‘fi’ do próximo na cadeia. Como em um cascateamento o bloco ‘n’ tem suas entradas geradas pelo bloco ‘n-1’, ele requer também que o bloco ‘n-1’ “avise” quando terminou a operação, ou seja, gere o sinal de ‘fi’ que será provido ao bloco ‘n’.

O princípio de funcionamento do *sense-amplifier* (Figura 4.15) é o seguinte: quando o sinal de controle (fi) está em nível lógico alto ‘1’, o circuito entra em modo de pré-descarga, os nós OUT e $\overline{OUT}$ são levados a gnd por ‘n1’ e ‘n2’, e os inversores

formados por ‘p2, n3’ e ‘p3, n4’ (aqui chamados por inv1 e inv2 respectivamente) são desconectados de Vdd por p1, Figura 4.15a; a partir do momento que o sinal de controle ‘fi’ vai para valor lógico baixo ‘0’, o circuito entra em período de avaliação, ‘n1’ e ‘n2’ são cortados, ‘p1’ passa a conduzir conectando assim inv1 e inv2 a Vdd (Figura 4.15b).

Quando o circuito se encontra em período de avaliação, o inv1 tentará carregar o nó OUT para Vdd assim como inv2 tentará fazer o mesmo com $\overline{OUT}$. Este impasse é resolvido devido à existência da árvore binária, pois quando todas as entradas forem avaliadas, um dos nós OUT ou $\overline{OUT}$ será conectado com gnd (Figura 4.15c), no caso considerando-se o instante em que inv1 recebe a conexão com gnd assim resolvendo o impasse (Figura 4.15d condição das variáveis após o circuito estabilizar). Justamente este “desbalanço” causado pela árvore binária que nos fornece a correta avaliação da função projetada.

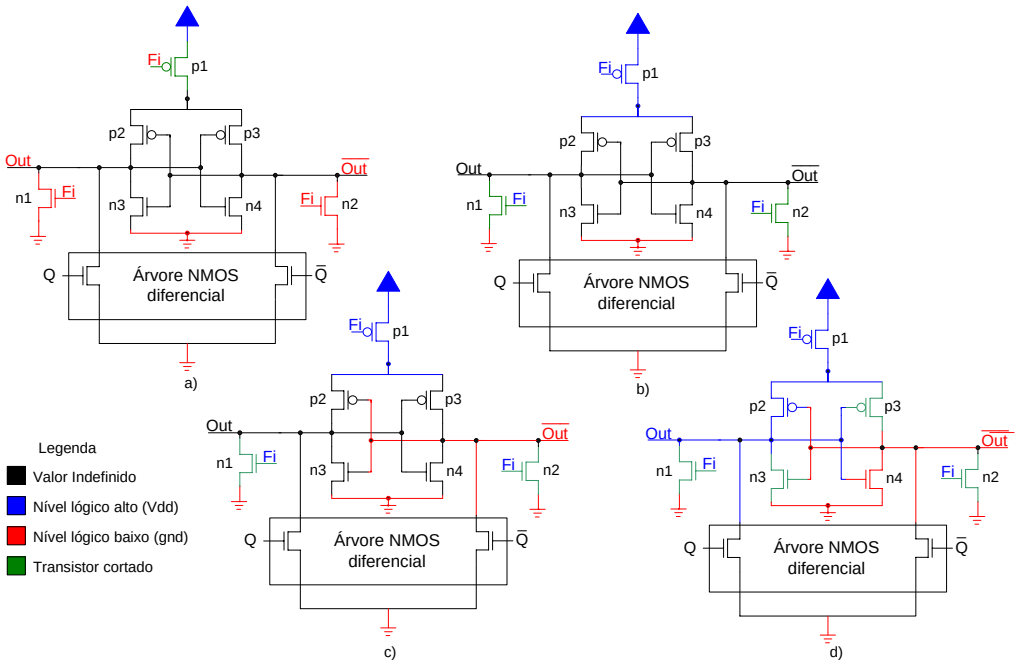


Figura 4.15 : Demonstração do princípio de funcionamento da técnica ECDL.

A estrutura *full-adder* ECDL é gerada a partir da estrutura construída em DCVS. A mesma árvore de transistores NMOS é utilizada com a diferença de que agora ela é conectada ao *sense-amplifier* e que o sinal de controle ‘fi’ externo também é gerado internamente a partir de cada bloco que constitui a cadeia. Na Figura 4.16 é apresentada à estrutura da árvore ‘n’ utilizada na construção do bloco *full-adder* que compõe os somadores construídos. Os nós negados indicados na figura são ligados ao circuito de pré-descarga.

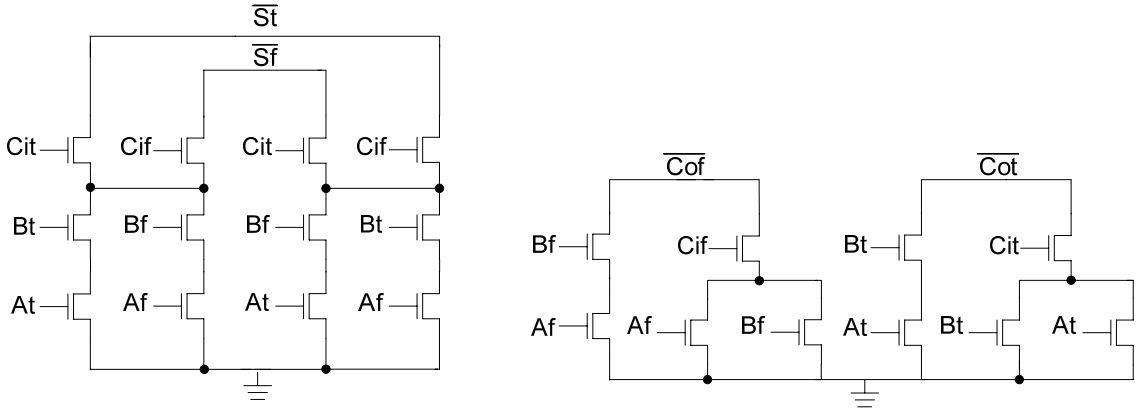


Figura 4.16 : Diagrama elétrico da árvore "n" que implementa a função full-adder.

4.2.7 Differential Pass Transistor Logic (DPTL)

Quando utilizamos à técnica DPTL de construção de circuitos, acabamos gerando assim como em DCVS e ECDL dois componentes distintos do circuito, uma rede NMOS ou PMOS que avaliará a função e um circuito de avaliação (neste caso um *sense amplifier*). A principal diferença reside no modo com que a rede que avaliará a função é construída.

Diferentemente dos casos anteriores onde tínhamos árvores binárias que avaliavam a função, agora temos uma rede formada por transistores de passagem. Na Figura 4.17a é apresentado um diagrama de blocos da referida técnica e na Figura 4.17b é mostrado o *sense amplifier* em maior detalhe.

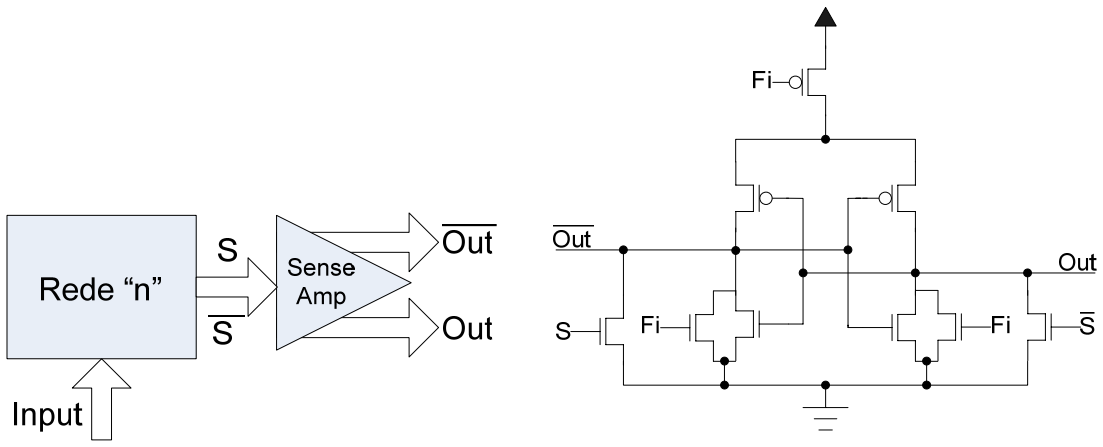


Figura 4.17 : a) Estrutura de blocos e b) Diagrama elétrico do *sense-amplifier* DPTL.

A implementação de uma simples porta formada por transistor de passagem (PTL) e sua tabela verdade são apresentados na Figura 4.18. Quando o sinal de controle é colocado em nível lógico alto, o valor da entrada é passado para a saída, caso contrário, a saída irá para alta impedância.

A	S
0	X
1	0

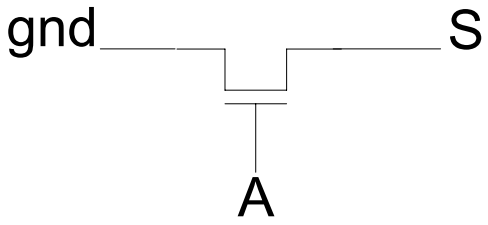


Figura 4.18 : Tabela verdade e implementação desta função em PTL.

Neste trabalho, somente foram geradas árvores para avaliação de funções que eram compostas por transistores NMOS. O objetivo desta tática de projeto é eliminar os lentos transistores do tipo ‘p’ com o intuito de que se possa usufruir das vantagens das redes de transistores de passagem NMOS (PASTERNAK, 87).

O problema de se eliminar os transistores do tipo PMOS reside na característica de que transistores tipo NMOS não transmitem eficientemente um nível lógico alto ‘1’. Para que se possa lidar com este problema, em DPTL as variáveis são codificadas de uma forma diferencial. Como os transistores do tipo ‘n’ transmitem eficientemente um nível lógico baixo ‘0’, uma das saídas diferenciais da função sempre conterá um ‘0’ ao final do período de avaliação enquanto que a outra terá um valor diferente de ‘0’. A partir deste momento o *sense amplifier* restaurará os valores de saída, pois levará para nível lógico alto à saída que contiver um valor diferente de ‘0’ deixando a outra em ‘0’.

Esta abordagem de construção de estruturas DPTL foi primeiramente proposta por (YANO, 89). Diferentemente da técnica usual de projeto, onde geralmente o elemento básico de construção de funções são as portas lógicas, em DPTL este elemento básico é formado pelo próprio transistor MOS (RADHAKRISHNAN, 85). A porta formada por este transistor obedecerá a um sinal de controle que poderá ser gerado utilizando-se tanto um nível lógico alto ‘1’ como baixo ‘0’. A escolha do nível lógico do controle dependerá somente do tipo de rede que será implementado, no caso ‘1’ para rede NMOS, devido à sua facilidade em “transmitir” o nível lógico ‘1’ e ‘0’ para rede PMOS.

De acordo com (YANO, 89) é colocado como principal vantagem desta estrutura uma grande imunidade a ruído. Já em (PASTERNAK, 91) é colocado que o modo de construção de circuitos se utilizando da técnica DPTL tem demonstrado vantagens significantes na área, desempenho e dissipação de potência no projeto de multiplexadores, máquinas de estado (PASTERNAK, 87), (PASTERNAK, 89) e circuitos aritméticos (PASTERNAK, 89), (YANO, 89).

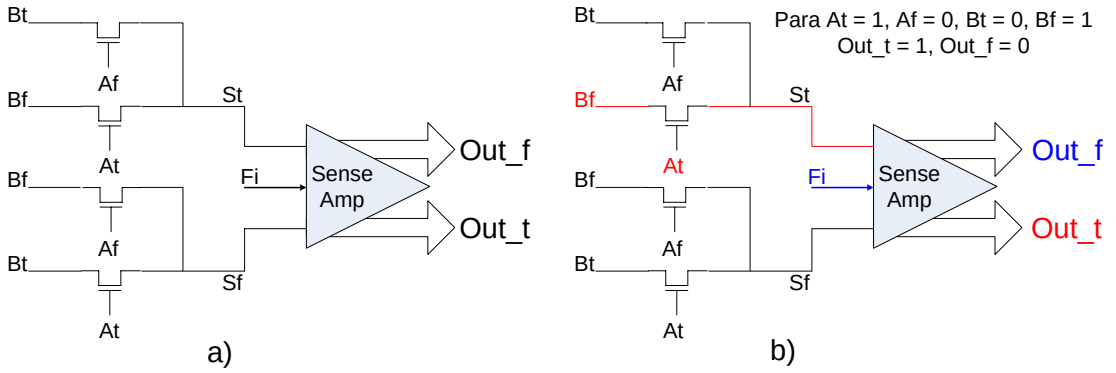


Figura 4.19 : Demonstração do princípio de funcionamento da técnica DPTL.

Este comportamento é demonstrado na Figura 4.19. Na mesma é demonstrada a construção de uma função lógica XOR. A partir do momento em que as entradas estão disponíveis e que a função foi avaliada (Figura 4.19a), o *sense amplifier* examinará as saídas e elevará para '1' a que contiver um valor diferente de '0' Figura 4.19b). Como quando projetamos circuitos DPTL também acabamos caindo numa situação com dois períodos de funcionamento, avaliação e *reset*, teremos de esperar que o sinal de controle 'fi' vá para '0' para que possa ocorrer a condição mostrada na Figura 4.19b.

A estrutura *full-adder* DPTL foi gerada através da conversão da equação original do mesmo em uma rede de transistores NMOS que é ligada ao *sense-amplifier* e controlada por um sinal de controle 'fi' externo. Sinais de controle 'fi' internos também são gerados na saída de cada bloco. Na Figura 4.20 é apresentada a estrutura da rede NMOS utilizada na construção do bloco *full-adder* que compõe os somadores construídos. Os nós negados indicados na figura são ligados ao *sense-amplifier* DPTL.

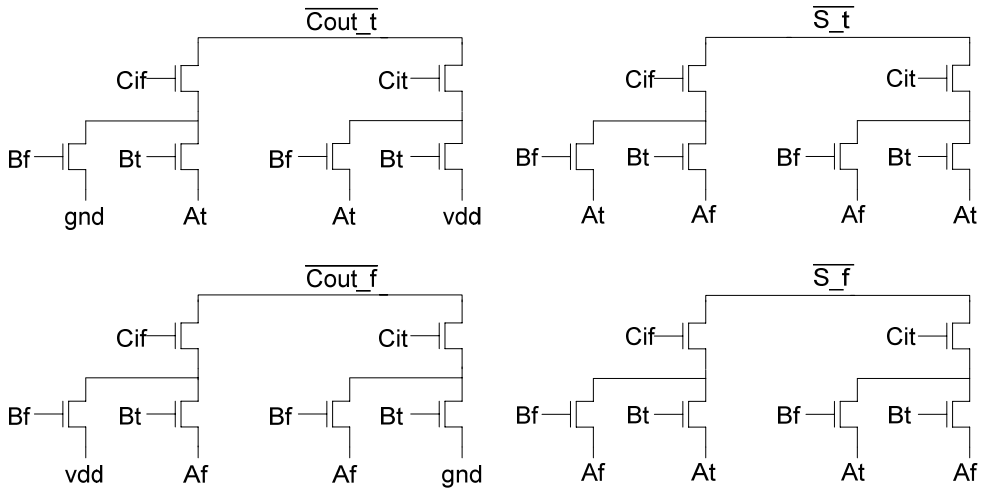


Figura 4.20 : Diagrama elétrico da rede NMOS que implementa a função *full-adder*.

4.3 Resultados de Simulação

Nesta seção serão apresentados os resultados obtidos na simulação das bases assíncronas para avaliação das estruturas somadoras. Estas simulações foram realizadas utilizando o simulador elétrico Spectre do ambiente DF II da CADENCE (CADENCE, 2003) em conjunto com o Pspice. Os processos utilizados nas simulações foram o AMI 0.6 (MOSIS, 2004) da Mosis e o AMS 0.35 (AMS, 2003) da AMS. Para o primeiro processo foram utilizados os seguintes parâmetros: tensão de alimentação equivalente a 5Volts e W e L mínimos tanto para NMOS como para PMOS iguais a 1,8 μm e 0,6 μm respectivamente. Para o segundo temos: tensão de alimentação igual a 3.3 Volts e W e L mínimos iguais a 1 μm e 0,3 μm respectivamente tanto para transistores NMOS como para PMOS.

Nesta simulação foram obtidos dados referentes à área (aqui representados pelo número de transistores de cada estrutura), atraso de propagação e consumo de potência.

Para que fosse possível coletar dados precisos em relação ao número de transistores e atraso, uma vez que o simulador apresenta dados relativos à estrutura como um todo, ou seja, arquitetura básica mais somador foi executada a seguinte operação:

- Realizar a simulação da estrutura como um todo, coletando dados relativos a número de transistores e atraso para um determinado número de ciclos do anel.

- Simular somente a estrutura do anel, de onde o somador a ser avaliado foi retirado, para o mesmo número de ciclos utilizado na simulação anterior.

- Subtrair dos resultados obtidos na primeira simulação os dados adquiridos na segunda, que são referentes somente ao anel, de forma a obter o número de transistores e o atraso de propagação dos somadores.

Para a computação dos dados de consumo de potência não é necessário realizar esta operação, pois para cada somador é adicionada uma fonte de alimentação independente onde é medido o consumo. O consumo de potência apresentado consiste no valor RMS do mesmo.

Um bloco somador do tipo RCA, pode ser considerado um bloco funcional do tipo indicação fraca. Como os dados são gerados assim que possível, não se pode afirmar com certeza qual será o último bit gerado e nele amostrar o fim de operação, pois este depende da combinação das entradas.

Esta situação pode ser mais bem entendida considerando-se a seguinte hipótese: imagine que o bit a ser amostrado para efeitos de indicação de fim de cálculo seja o *carry-out*, para a seguinte combinação de entradas (00111111+00111111). Teremos um "*kill*" (o sinal de saída *carry* do bloco é zerado) no último bit que define o valor de *carry-out*, e um "*propagate*" (o sinal *carry* de entrada é propagado para a saída do bloco) que passará por toda a cadeia até chegar ao sétimo bit, fazendo assim com que tenhamos uma indicação de fim de cálculo antes de o sétimo bit ser calculado, assim incorrendo em um erro de computação.

O meio de contornar este problema consistiu em amostrar 'n/2' bits, dessa forma gerando uma indicação um pouco mais lenta, mas bastante eficiente, pois no mínimo um '*propagate*', '*generate*' (o sinal *carry* de saída do bloco é gerado) ou '*kill*' em seqüência é gerado.

Outra observação importante é que devido à técnica SDP não obedecer a todos os requisitos para que possa ser considerada uma técnica de construção de circuitos assíncronos, como demonstrado anteriormente, a mesma é somente apresentada, e não simulada.

Os resultados apresentados na tabela consistem em número de transistores, atraso de propagação, potência consumida e a figura de mérito 'AxP' que consiste o produto entre atraso e potência. As colunas onde estes dados estão representados são divididas em duas partes, na primeira é apresentado o valor medido e na segunda é apresentado um valor percentual do mesmo tendo a família DCVS como referência (100%). Os valores de atraso e potência consumida são valores médios referentes a um ciclo de cálculo do anel.

O critério utilizado na escolha dos vetores de teste que seriam aplicados às aplicações foi o seguinte: escolher valores que dentre as possibilidades presentes para a atual largura de palavra na qual será realizada a simulação, sejam obtidos o maior número de ciclos de cálculo possíveis para a operação.

4.3.1 Resultados de simulação Divisor Inteiro (DI)

A média de resultados apresentado para o DI RCA de 4 bits foi calculado tendo por base uma simulação gerada a partir dos vetores A=15 e B=2 que totalizaram oito ciclos de cálculo do anel.

Como pode ser notado, em termos de área as estruturas DCVS, ECDL, DPTL e Martin apresentaram resultados semelhantes, ao passo que o circuito NCL apresenta uma relação de duas vezes em relação a referência (DCVS), que é devido ao número de transistores relativamente grande que compõe suas portas. O que mais se destaca em termos de número de transistores é com certeza a técnica DIMS que apresenta uma área da ordem de quatro vezes a da referência. Este fato é devido à baixa possibilidade de otimização da estrutura gerada e ao conseqüente grande número de portas utilizado em sua construção.

Tabela 4.1 : Resultados obtidos para o DI RCA 4 bits.

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	160	100,00	1,34	100,00	1,93	100,00	2,58	100,00
ECDL	158	98,75	4,55	340,19	1,82	94,30	8,28	320,80
DPTL	182	113,75	3,16	236,45	1,89	97,93	5,98	231,55
DIMS	608	380,00	12,91	965,42	2,75	142,49	35,51	1375,60
Martin	168	105,00	2,33	173,83	2,38	123,32	5,53	214,36
NCL	288	180,00	4,53	338,32	1,56	80,83	7,06	273,46

Já em relação ao atraso, a estrutura DCVS se destaca devido à baixa média atingida. Isto se deve principalmente ao sinal de pré-carga chavear todo o circuito de forma a atingir a máxima velocidade de computação. Isto pode ser notado, pois apesar das estruturas DPTL, ECDL e Martin apresentarem praticamente a mesma estrutura,

elas acabam perdendo em relação a desempenho. Isto é devido, no caso de ECDL e DPTL, à necessidade de que um bloco gere o sinal de avaliação que será transmitido ao próximo bloco, assim aumentando a profundidade lógica, e em relação a Martin, o atraso adicionado pelos vários transistores PMOS em série. Outro fato que chama a atenção é o baixo desempenho da técnica DIMS que é praticamente dez vezes mais lenta que a referência, sendo este fato devido à alta profundidade lógica da estrutura. Enquanto que para um *full-adder* DCVS temos somente uma rede NMOS avaliando cada sinal, em DIMS existem cinco portas em série para gerar cada sinal.

Quanto ao consumo de potência, a estrutura NCL apresenta o melhor resultado sendo seguido pelas estruturas dinâmicas ECDL, DCVS e DPTL que apresentam um consumo levemente superior devido à necessidade de pré-carga ou pré-descarga durante o período de *reset*. Neste resultado a técnica DIMS apresenta novamente o pior resultado devido à grande quantidade de transistores envolvidos no processo de computação da função.

Apesar da estrutura NCL apresentar o menor consumo de potência quando só este parâmetro é analisado pode-se notar que devido ao baixo atraso em conjunto com um consumo razoavelmente baixo de potência, a estrutura que apresenta melhor relação 'consumo X atraso' é a técnica DCVS. Como era de se esperar também a família DIMS apresenta a pior relação AxP sendo está na ordem de treze vezes.

Tabela 4.2 : Resultados obtidos para o DI RCA 8 bits.

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	320	100,00	1,91	100,00	2,00	100,00	3,82	100,00
ECDL	318	99,38	9,96	521,23	2,12	106,00	21,12	552,50
DPTL	366	114,38	7,50	392,38	3,01	150,50	22,58	590,53
DIMS	1216	380,00	28,21	1475,64	2,63	131,50	74,18	1940,46
Martin	336	105,00	3,43	179,67	2,13	106,50	7,32	191,35
NCL	576	180,00	7,63	399,40	1,84	92,00	14,05	367,45

A simulação realizada para o circuito DI RCA de 8 bits foi executada para os vetores de entrada A=245 e B=7 sendo que foram considerados 35 ciclos do anel no cálculo das médias de medidas.

Como pode ser notado os resultados obtidos seguem o padrão obtido na simulação de 4 bits, sendo as justificativas antes mencionadas aqui também aplicáveis. A única diferença aparente é um aumento relativo à referência nos dados coletados de consumo de potência e atraso, sendo o último devido principalmente ao aumento da cadeia de *carry*.

4.3.2 Resultados de simulação Divisor de Resto (DR)

Os vetores utilizados na simulação do DR RCA de 4 bits foram A=3 e B=7. A média dos valores foi calculada considerando-se doze ciclos de computação do anel.

Tabela 4.3 : Resultados obtidos para o DR RCA 4 bits.

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	160	100,00	1,58	100,00	1,72	100,00	2,72	100,00
ECDL	158	98,75	3,62	228,42	2,05	119,19	7,41	272,25
DPTL	182	113,75	2,18	137,37	2,09	121,51	4,55	166,92
DIMS	608	380,00	6,08	384,21	3,63	211,05	22,08	810,86
Martin	168	105,00	1,98	124,74	2,18	126,74	4,31	158,10
NCL	288	180,00	2,52	158,95	1,61	93,60	4,05	148,78

Quanto ao atraso a tendência apresentada na aplicação anterior se mantém, apesar de neste exemplo termos uma maior proximidade das famílias DPTL, Martin e NCL da referência.

A potência consumida também apresenta resultados semelhantes à aplicação anterior, mas com um leve aumento em relação à referência. Já a figura AxP mantém o padrão sendo ainda a família DCVS a melhor relação ‘atraso X potência’.

Tabela 4.4 : Resultados obtidos para o DR RCA 8 bits.

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	320	100,00	2,76	100,00	1,61	100,00	4,44	100,00
ECDL	318	99,38	10,78	391,16	1,28	79,50	13,80	310,98
DPTL	366	114,38	7,21	261,56	1,65	102,48	11,90	268,06
DIMS	1216	380,00	23,97	869,50	5,13	318,63	122,94	2770,52
Martin	336	105,00	1,53	55,44	1,60	99,38	2,45	55,10
NCL	576	180,00	2,78	100,79	1,86	115,53	5,17	116,44

Para a avaliação do DR RCA de 8 bits foram utilizados os vetores A=100 e B=23 e considerados 32 ciclos do anel. O fato mais marcante nesta aplicação é o desempenho em relação ao atraso que as famílias Martin e NCL apresentam.

Devido à relação de consumo permanecer obedecendo a mesma tendência e como houve uma modificação nos atrasos, o AxP que apresenta a melhor relação ‘consumo X potência’ nesta aplicação passa a ser a família Martin sendo que a estrutura NCL também se aproxima bastante da referência.

4.3.3 Resultados de simulação Máximo Divisor Comum (MDC)

A média de resultados apresentado para o MDC RCA de 4 bits foi calculado tendo por base uma simulação gerada a partir dos vetores A=15 e B=12 que totalizaram cinco ciclos do anel.

Tabela 4.5 : Resultados obtidos para o MDC RCA 4 bits .

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	160	100,00	0,52	100,00	1,34	100,00	0,70	100,00
ECDL	158	98,75	2,36	453,85	1,53	114,18	3,61	518,20
DPTL	182	113,75	1,34	257,69	1,66	123,51	2,22	318,27
DIMS	608	380,00	5,08	976,92	2,05	152,99	10,41	1494,55
Martin	168	105,00	0,26	50,00	1,41	105,22	0,37	52,61
NCL	288	180,00	0,64	123,08	1,05	78,36	0,67	96,44

Como pode ser notado, novamente a família Martin apresenta resultados de atraso da ordem de metade do valor referência. A potência consumida pela família NCL continua sendo a menor. Estes dois fatos, menor atraso Martin e menor consumo NCL fazem com que a relação AxP seja mais interessante nestas duas famílias do que na referência.

Tabela 4.6 : Resultados obtidos para o MDC RCA 8 bits.

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	320	100,00	3,37	100,00	2,23	100,00	7,50	100,00
ECDL	318	99,38	11,64	345,24	1,09	48,76	12,63	168,35
DPTL	366	114,38	8,95	265,39	1,23	55,28	11,00	146,71
DIMS	1216	380,00	28,12	834,12	1,81	81,35	50,89	678,54
Martin	336	105,00	2,62	77,63	2,53	113,71	6,62	88,27
NCL	576	180,00	6,50	192,95	0,89	40,00	5,79	77,18

A simulação realizada para o circuito MDC RCA de 8 bits foi executada para os vetores de entrada A=249 e B=12 sendo que foram considerados 24 ciclos do anel no cálculo das médias de medidas.

As tendências apresentadas na versão de quatro bits desta aplicação continuam se mantendo, fazendo assim com que as melhores relações AxP continuem sendo pertencentes às famílias Martin e NCL. A única diferença aparece na diminuição relativa a referência da maioria das famílias, como ECDL, DPTL e DIMS.

4.3.4 Resultados de simulação Mínimo Múltiplo Comum (MMC)

Os vetores utilizados na simulação do MMC RCA de 4 bits foram A=5 e B=3. A média dos valores foi calculada considerando-se sete ciclos de computação do anel.

Tabela 4.7 : Resultados obtidos para o MMC RCA 4 bits.

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	160	100,00	2,33	100,00	1,17	100,00	2,72	100,00
ECDL	158	98,75	4,19	179,75	1,40	120,00	5,86	215,71
DPTL	182	113,75	3,17	136,20	1,47	125,71	4,65	171,22
DIMS	608	380,00	5,16	221,47	2,75	235,43	14,16	521,41
Martin	168	105,00	1,54	66,26	1,38	118,00	2,12	78,18
NCL	288	180,00	1,94	83,44	0,88	75,43	1,71	62,93

Na aplicação MMC as famílias Martin e NCL apresentam novamente vantagens em relação à referência. Devido à estrutura NCL apresentar também o menor consumo de potência é a família que apresenta a menor relação AxP sendo seguida por Martin e DCVS.

Tabela 4.8 : Resultados obtidos para o MMC RCA 8 bits.

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	320	100,00	6,14	100,00	1,50	100,00	9,19	100,00
ECDL	318	99,38	13,15	214,21	1,52	101,34	19,94	217,08
DPTL	366	114,38	10,70	174,27	1,77	118,49	18,97	206,48
DIMS	1216	380,00	29,42	479,38	4,19	280,18	123,39	1343,12
Martin	336	105,00	4,20	68,43	1,57	104,68	6,58	71,63
NCL	576	180,00	7,77	126,63	0,96	64,37	7,49	81,51

Para a avaliação do MMC RCA de 8 bits foram utilizados os vetores A=17 e B=13 e considerados 29 ciclos do anel. O fato mais marcante nesta aplicação é o desempenho em relação ao atraso que as famílias Martin e NCL apresentam.

Nesta versão novamente as famílias Martin e NCL apresentam bom desempenhos relativos a atraso, consumo e AxP. Este bom desempenho da estrutura Martin em relação a atraso é devida provavelmente a ela apresentar um bom desempenho no cálculo do *carry out* do somador. Como pode ser notado, todas as aplicações (DR, MDC e MMC) onde esta estrutura apresentou desempenho melhor que a referência contêm um multiplexador logo em seguida ao circuito somador. Como o multiplexador é em todos os casos controlado pelo sinal de *carry out* de algum dos somadores, provavelmente o cálculo mais rápido do mesmo beneficie no desempenho médio dos sistemas.

4.3.5 Resultados de simulação Raiz

A média de resultados apresentado para o Raiz RCA de 4 bits foi calculado tendo por base uma simulação gerada a partir do vetor A=9 que totalizou cinco ciclos do anel.

Tabela 4.9 : Resultados obtidos para a Raiz RCA 4 bits.

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	160	100,00	1,66	100,00	1,80	100,00	2,99	100,00
ECDL	158	98,75	6,44	387,95	1,88	104,17	12,08	404,12
DPTL	182	113,75	3,70	222,89	2,33	129,44	8,62	288,52
DIMS	608	380,00	22,12	1332,53	1,98	110,00	43,80	1465,78
Martin	168	105,00	2,50	150,60	1,91	105,83	4,76	159,39
NCL	288	180,00	2,96	178,31	1,41	78,06	4,16	139,18

Como nesta estrutura novamente não temos um multiplexador na saída do somador, assim como no Contador e DI, a família referência DCVS volta a ter o melhor desempenho em relação a atraso e conseqüentemente a melhor relação AxP. Os outros resultados continuam mantendo as mesmas tendências.

Tabela 4.10 : Resultados obtidos para a Raiz RCA 8 bits.

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	320	100,00	7,92	100,00	2,40	100,00	11,05	100,00
ECDL	318	99,38	22,89	289,00	2,32	94,62	30,22	273,46
DPTL	366	114,38	21,04	265,55	2,60	114,34	33,55	303,63
DIMS	1216	380,00	64,56	815,06	3,70	193,55	174,32	1577,53
Martin	336	105,00	10,71	135,26	2,52	108,96	16,29	147,38
NCL	576	180,00	18,82	237,60	2,15	82,44	21,64	195,87

A simulação realizada para o circuito Raiz RCA de 8 bits foi executada para o vetor de entrada A=249 sendo que foram considerados 14 ciclos do anel no cálculo das médias dos resultados.

Aos resultados apresentados para esta aplicação podem ser observados novamente as mesmas tendências de resultados, tendo como DCVS a estrutura que apresenta melhores resultados em relação a atraso e AxP e a família NCL quando se observa a potência consumida.

4.3.6 Resultados de simulação Contador

Os resultados obtidos para a aplicação Contador são apresentados a partir de duas tabelas. A primeira contém os resultados de simulação utilizando-se a tecnologia AMI 0.6 e a segunda AMS 0.35. Este formato é utilizado para todas as versões, ou seja, 4, 8, 16 e 32 bits da mesma.

Tabela 4.11 : Resultados obtidos para o Contador RCA 4 bits (AMI).

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	144	100,00	0,86	100,00	2,63	100,00	2,25	100,00
ECDL	162	112,50	5,34	624,12	2,00	76,05	10,67	474,62
DPTL	186	129,17	4,67	545,91	1,84	69,96	8,59	381,93
DIMS	550	381,94	10,54	1232,31	2,60	98,86	27,39	1218,25
Martin	174	120,83	3,31	386,55	1,54	58,56	5,09	226,34
NCL	294	204,17	3,84	448,68	4,39	166,92	16,84	748,94

Tabela 4.12 : Resultados obtidos para o Contador RCA 4 bits (AMS).

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	144	100,00	0,55	100,00	1,14	100,00	0,63	100,00
ECDL	162	112,50	3,66	664,77	0,82	71,93	3,00	478,17
DPTL	186	129,17	3,21	584,09	0,76	66,67	2,44	389,39
DIMS	550	381,94	7,04	1280,68	1,10	96,49	7,75	1235,75
Martin	174	120,83	2,18	396,59	0,69	60,53	1,51	240,04
NCL	294	204,17	2,58	468,18	1,61	141,23	4,15	661,20

Tabela 4.13 : Resultados obtidos para o Contador RCA 8 bits (AMI).

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	288	100,00	0,95	100,00	4,43	100,00	4,21	100,00
ECDL	322	111,81	9,76	1027,63	3,16	71,33	30,85	733,03
DPTL	322	111,81	9,71	1022,37	3,17	71,56	30,79	731,58
DIMS	1094	379,86	19,36	2038,16	3,08	69,53	59,64	1417,05
Martin	342	118,75	2,30	242,11	3,25	73,36	7,48	177,62
NCL	582	202,08	4,65	489,47	7,15	161,40	33,25	790,01

Tabela 4.14 : Resultados obtidos para o Contador RCA 8 bits (AMS).

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	288	100,00	0,68	100,00	2,14	100,00	1,44	100,00
ECDL	322	111,81	7,29	1079,63	1,24	57,94	9,04	625,58
DPTL	322	111,81	7,23	1070,37	1,23	57,48	8,89	615,21
DIMS	1094	379,86	13,56	2009,26	1,35	63,08	18,31	1267,52
Martin	342	118,75	1,54	227,78	1,57	73,36	2,41	167,11
NCL	582	202,08	3,15	466,67	2,87	134,11	9,04	625,86

Tabela 4.15 : Resultados obtidos para o Contador RCA 16 bits (AMI).

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	576	100,00	1,10	100,00	8,50	100,00	9,34	100,00
ECDL	642	111,46	19,90	1811,04	3,60	42,35	71,64	767,03
DPTL	642	111,46	19,87	1808,19	3,58	42,12	71,13	761,57
DIMS	2182	378,82	39,89	3630,72	4,53	53,29	180,71	1934,96
Martin	678	117,71	2,32	211,49	6,49	76,35	15,08	161,48
NCL	1158	201,04	7,26	660,75	12,82	150,82	93,07	996,57

Tabela 4.16 : Resultados obtidos para o Contador RCA 16 bits (AMS).

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	576	100,00	0,63	100,00	3,91	100,00	2,47	100,00
ECDL	642	111,46	14,76	2338,61	1,38	35,29	20,37	825,39
DPTL	642	111,46	14,73	2332,67	1,37	35,04	20,17	817,33
DIMS	2182	378,82	27,80	4403,96	1,86	47,57	51,71	2094,98
Martin	678	117,71	1,41	222,77	2,86	73,15	4,02	162,95
NCL	1158	201,04	4,93	781,19	5,00	127,88	24,66	998,96

Tabela 4.17 : Resultados obtidos para o Contador RCA 32 bits (AMI).

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	1152	100,00	1,49	100,00	16,20	100,00	24,14	100,00
ECDL	1282	111,28	40,28	2703,02	4,31	26,60	173,59	719,14
DPTL	1282	111,28	40,27	2702,60	4,29	26,48	172,75	715,69
DIMS	4358	378,30	75,35	5057,05	5,80	35,80	437,03	1810,55
Martin	1350	117,19	2,33	156,52	11,85	73,15	27,64	114,49
NCL	2310	200,52	12,43	834,52	22,00	135,80	273,56	1133,30

Tabela 4.18 : Resultados obtidos para o Contador RCA 32 bits (AMS).

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	1152	100,00	1,13	100,00	7,10	100,00	8,05	100,00
ECDL	1282	111,28	30,71	2709,76	1,85	26,06	56,80	706,06
DPTL	1282	111,28	30,66	2705,90	1,84	25,92	56,42	701,25
DIMS	4358	378,30	53,88	4754,99	2,43	34,23	130,93	1627,41
Martin	1350	117,19	1,60	141,09	5,35	75,35	8,55	106,32
NCL	2310	200,52	8,95	789,60	8,83	124,37	79,00	982,00

A partir destes resultados aqui apresentados para as versões 4, 8, 16 e 32 bits do somador RCA, pode-se notar principalmente as desvantagens em relação a atraso de propagação que foram obtidas para as famílias ECDL, DPTL e DIMS. Para a família DIMS, este mal resultado já era esperado e como já exposto anteriormente, já havia uma tendência para o aumento de atraso das famílias ECDL e DPTL devido à necessidade de geração dos *carry* intermediários.

Levando-se em consideração o consumo de potência, é observado que as famílias DCVS e principalmente a NCL apresentam os valores mais altos. Para a família NCL isto é uma surpresa, pois a mesma apresentou valores bastante baixos para este parâmetro nas outras aplicações.

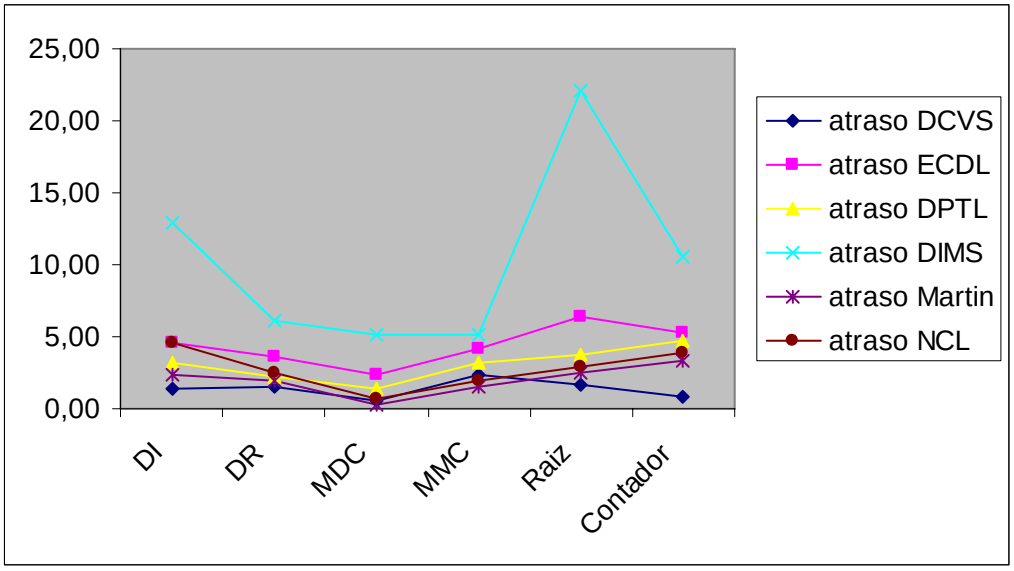
Em relação a área, as tendências se mantiveram, ou seja, a família DCVS apresentando sempre os valores mais baixos de nº de transistores utilizados.

Quanto a figura de mérito AxP pode-se perceber também o melhor desempenho da família DCVS em relação as demais.

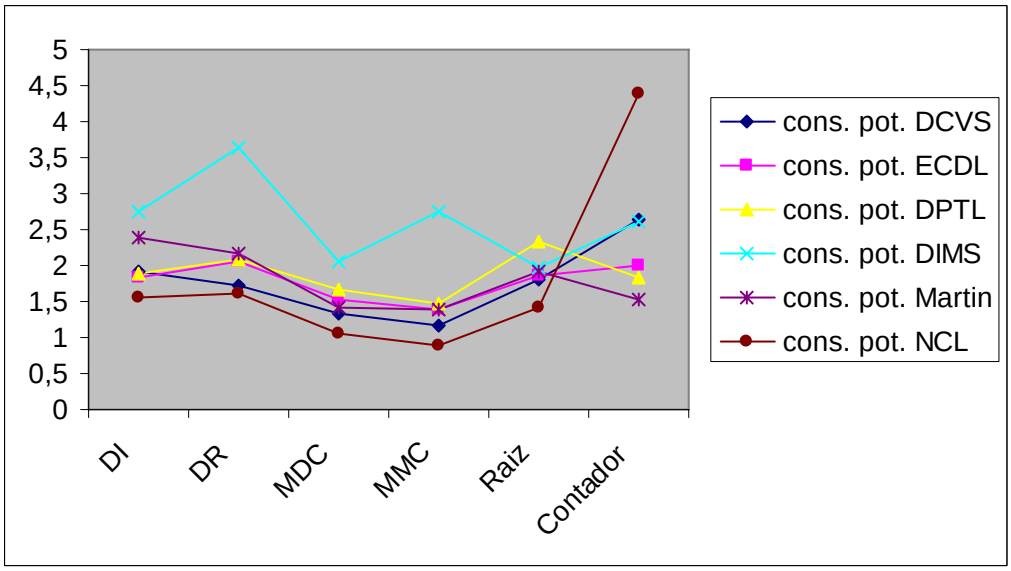
4.4 Gráficos comparativos RCA

Neste tópico será exposto o comportamento das diferentes famílias lógicas CMOS quando seus parâmetros de atraso e consumo de potência são submetidos as diferentes aplicações. Serão mostrados gráficos em que um parâmetro fixo (atraso ou consumo) são avaliados através da variação da aplicação.

Apesar do objetivo deste trabalho não consistir na avaliação em si das aplicações assíncronas, este tipo de comparação é interessante para demonstrar a dependência do atraso e consumo em relação ao padrão de vetores de teste a que ele está submetido.



a)

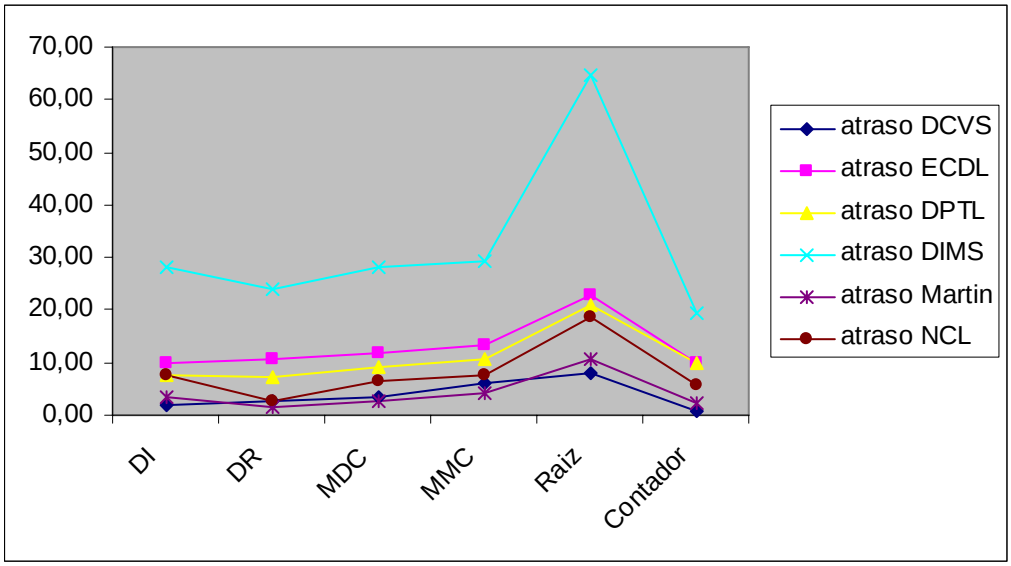


b)

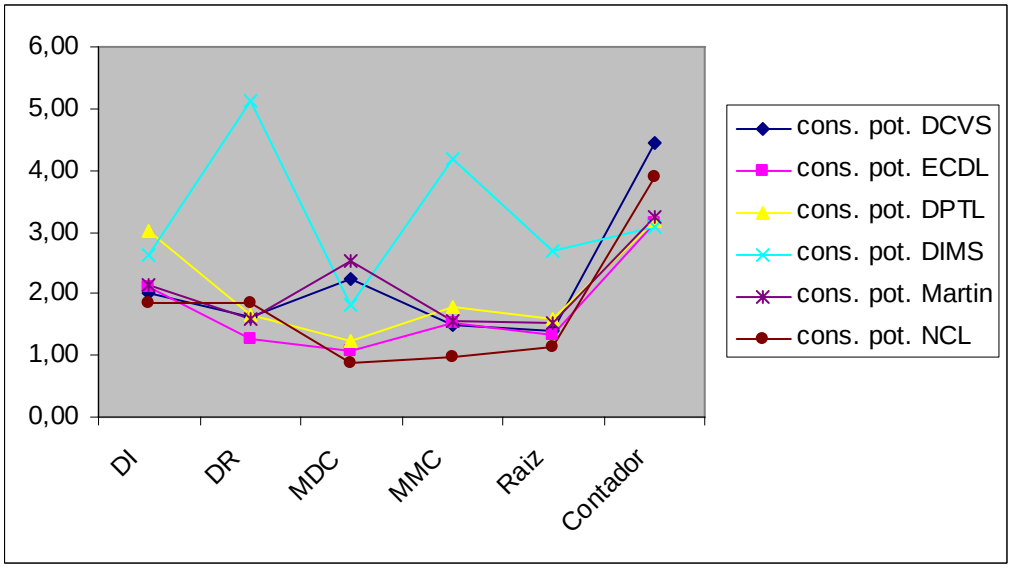
Figura 4.21 : Gráficos comparativos ‘parâmetro X família’ RCA 4 bits; a) atraso (ns) X família lógica; b) consumo (mW) X família lógica.

Como pode ser percebido, existe uma correlação interessante entre o modo com que o parâmetro se relaciona com o padrão de vetores de avaliação a ele aplicados.

A mesma tendência pode ser observada no gráfico apresentado a seguir, para as aplicações de 8 bits.



a)



b)

Figura 4.22 : Gráficos comparativos ‘parâmetro X família’ RCA 8 bits; a) atraso (ns) X família lógica; b) consumo (mW) X família lógica.

O padrão apresentado em relação a atraso tanto para as versões de 4 bits quanto de 8 claramente seguiram a mesma tendência, o mesmo ocorrendo para o consumo.

Este tipo de análise através dos gráficos é interessante, pois de uma maneira mais clara demonstra particularidades de cada família. Um exemplo disto é que através da observação do gráfico podem-se perceber quais são as estruturas que são mais penalizadas em relação a atraso, quando se observa os valores para a aplicação raiz.

Como a mesma contem dois somadores em série, através de sua análise, é possível concluir qual das famílias é mais suscetível a uma dependência de dados.

5 SOMADORES CLA

Neste capítulo será apresentado em um primeiro momento o algoritmo utilizado na construção do somador *carry look-ahead* e seu diagrama de blocos para diferentes larguras de dados de entrada (4, 8, 16 e 32 bits). Em seguida serão apresentadas as estruturas CMOS com múltiplas saídas. A seguir as estruturas CLA geradas assim como os resultados de suas simulações. E ao final será apresentada uma tabela comparativa de resultados e a análise dos mesmos.

5.1 *Carry Look-Ahead Adder* (CLA)

Estruturas com múltiplas saídas são apropriadas para a implementação de expressões lógicas com propriedades recursivas. Um bom exemplo é o somador *carry look-ahead* (CLA). Devido ao fato deste somador representar, até o presente momento, uma das melhores arquiteturas para unidades aritméticas de alto desempenho, o mesmo tem sido freqüentemente investigado para a implementação com portas lógicas de múltiplas saídas (WANG, 97), (HWANG, 89).

5.1.1 Algoritmo

No somador CLA, se C_{i-1} representa o sinal de *carry in* para o estágio 'i', e A_i e B_i os 'i-ésimos' bits dos vetores de entrada, então o sinal de *carry out* C_i pode ser expresso por (PARHAMI, 2000):

$$C_i = G_i + P_i C_{i-1},$$

onde,

$$G_i = A_i B_i \quad \text{e} \quad P_i = A_i \oplus B_i$$

são os sinais de *generate* G_i e *propagate* P_i respectivamente.

Expandindo a equação de C_i, tem-se:

$$C_i = G_i + P_i G_{i-1} + P_i P_{i-1} G_{i-2} + ... + P_i P_{i-1} ... P_1 C_0.$$

A soma é calculada por:

$$S_i = C_{i-1} \oplus A_i \oplus B_i = C_{i-1} \oplus P_i$$

Este princípio pode gerar todos os sinais de *carry* em apenas três níveis lógicos, desconsiderando restrições de *fan-out*, *fan-in* ou tamanho das portas lógicas. Caso o índice 'i' torne-se muito grande, o número de transistores aumenta demasiadamente, assim como o valor do *fan-out* para P_i e G_i.

Para a implementação em lógicas diferenciais, é necessário gerar tanto o sinal C_i, como seu complemento $\overline{C_i}$.

Para isso, faz-se necessário também a geração da equação complementar para o sinal de *carry-out*:

$$\overline{C_i} = \overline{G_i} \cdot (\overline{P_i} + \overline{C_{i-1}})$$

Entretanto, a falta de dualidade entre as árvores complementares pode ser resolvida modificando a definição de $\overline{C_i}$.

Partindo da equação inicial:

$$\overline{C_i} = \overline{G_i} \cdot (\overline{P_i} + \overline{C_{i-1}}) = \overline{G_i} \cdot (\overline{P_i} + P_i) \cdot (\overline{P_i} + \overline{C_{i-1}}) = \overline{G_i P_i} + \overline{G_i} P_i \overline{C_{i-1}}$$

sendo

$$\overline{G_i} P_i = P_i \text{ e } \overline{G_i P_i} = \overline{A_i + B_i} = N_i$$

onde N (kill) é o sinal que gera $\overline{C_i}$, tem-se:

$$\overline{C_i} = N_i + P_i N_{i-1} + P_i P_{i-1} N_{i-2} + \ldots + P_i P_{i-1} \ldots P_i C_0.$$

Pode-se observar que a equação de $\overline{C_i}$ é similar a C_i, trocando-se apenas N_i por G_i. A árvore lógica diferencial é construída de forma simétrica, usando o conceito dos sinais P, G e N. Devido ao rápido crescimento da equação que gera os sinais de *carry* C_i´s, geralmente o tamanho do termo "i" é limitado a 4.

Nos subitens a seguir será demonstrado, por diagramas de blocos, o modo como os somadores implementados neste trabalho serão estruturados. Estes diagramas conterão as estruturas para as versões de 4, 8, 16 e 32 bits dos mesmos.

5.1.2 CLA 4 bits

Como dito anteriormente, devido ao aumento gerado no tamanho das portas lógicas quando o termo 'i' cresce em demasiado, acaba-se por ter de "particionar" os somadores. Como este termo geralmente é limitado a 4, o somador *carry look-ahead* de quatro bits pode ser gerado diretamente das equações que definem este tipo de somadores.

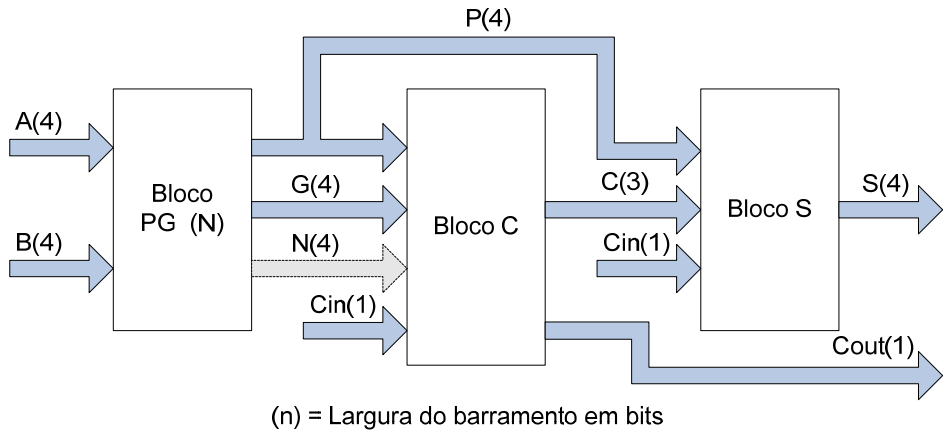


Figura 5.1 : Diagrama de blocos de um somador CLA de 4 bits.

Na Figura 5.1 é mostrado o diagrama de blocos do somador CLA de quatro bits. Como pode-se perceber ele utiliza o número mínimo de blocos necessário para implementar o algoritmo *carry look-ahead*. O bloco 'PG (N)' do circuito gera as operações $G_i = A_iB_i$, $P_i = A_i \oplus B_i$ e $N_i = \overline{A_i + B_i}$.

O bloco C será responsável pela geração dos valores de somas de *carry* intermediários, implementando as equação $C_i = G_i + P_iG_{i-1} + P_iP_{i-1}G_{i-2} + ... + P_iP_{i-1} ... P_1C_0$ e $\overline{C_i} = N_i + P_iN_{i-1} + P_i P_{i-1}N_{i-2} + ... + P_i P_{i-1}...P_iC_0$ com 'i' variando de '1' até '4'.

No último bloco do circuito (bloco S), serão geradas as somas. A equação desenvolvida será simplesmente uma XOR dos valores P_i e C_{i-1}, ou seja $S_i = C_{i-1} \oplus P_i$.

5.1.3 CLA 8 bits

Para o somador em sua versão de oito bits, é construída uma estrutura a partir de duas partes operativas de quatro bits. O diagrama de blocos é mostrado na Figura 5.2 sendo que o primeiro e segundo blocos são iguais, onde cada um é composto por um somador *carry look-ahead* de quatro bits. Os mesmos são conectados na forma de um *ripple carry adder*.

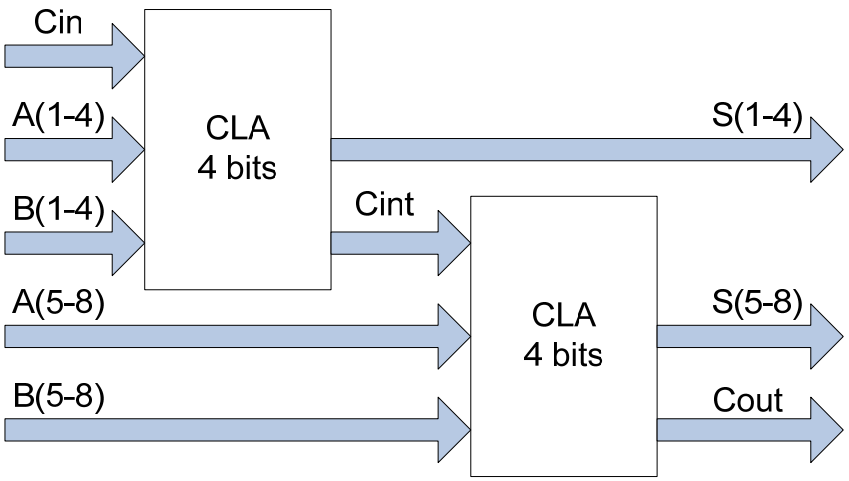


Figura 5.2 : Diagrama de blocos de um CLA 8 bits.

5.1.4 CLA 16 bits

Na Figura 5.3 segue o diagrama de blocos que constitui o somador *carry look-ahead* de 16 bits. Como podemos notar ele é constituído de cinco diferentes tipos de blocos que realizam funções distintas como demonstrado a seguir.

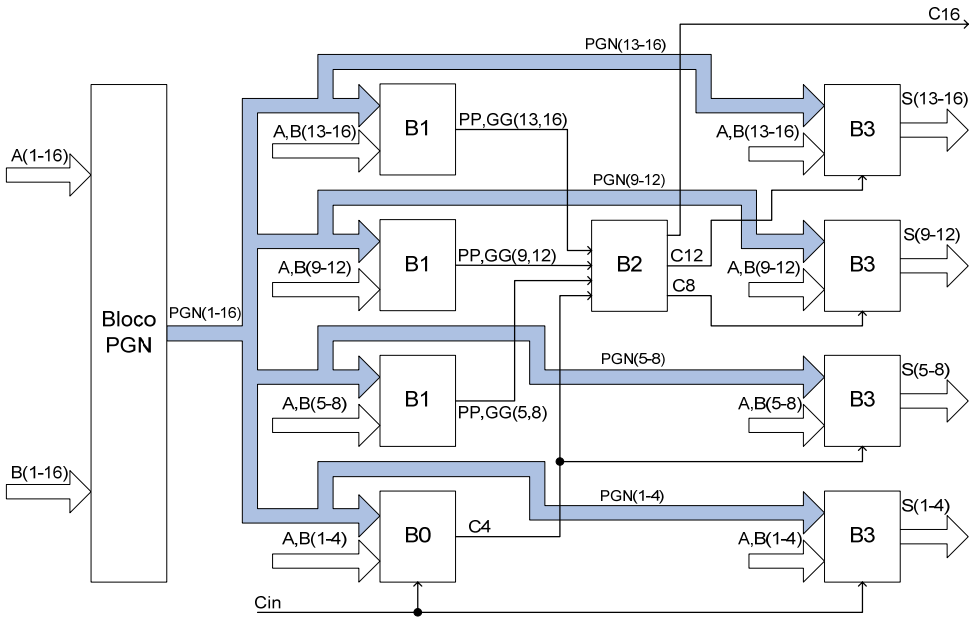


Figura 5.3 : Diagrama de blocos de um CLA 16 bits.

O bloco ‘PGN’ é responsável pela geração dos sinais *propagate, generate e kill*. Este bloco realiza as funções $G_i = A_iB_i$, $P_i = A_i \oplus B_i$ e $N_i = \overline{A_i + B_i}$ sendo que ‘i’ varia de 1 a 16 neste caso. Já o bloco ‘B0’ será incumbido da geração do sinal de *carry* resultante da operação sobre os primeiros 4 bits da seqüência. A equação implementada por ele será: $C_4 = G_4 + P_4 \cdot G_3 + P_4 \cdot P_3 \cdot G_2 + P_4 \cdot P_3 \cdot P_2 \cdot G_1 + P_4 \cdot P_3 \cdot P_2 \cdot P_1 \cdot C_0$.

No bloco ‘B1’ teremos implementadas as equações $PG_{i,j} = P_{i-3} \cdot P_{i-2} \cdot P_{i-1} \cdot P_i$ e $GG_{i,j} = G_i + P_i \cdot G_{i-1} + P_i \cdot P_{i-1} \cdot G_{i-2} + P_i \cdot P_{i-1} \cdot P_{i-2} \cdot G_{i-3}$, onde ‘i’ é o limite superior do grupo e ‘j’ é o inferior. Por exemplo caso estejamos trabalhando com o último grupo de quatro bits, ‘i’ será igual a 16 e ‘j’ igual a 13. Como pode ser visto, este bloco é responsável pela geração dos sinais *propagate* e *generate* agrupados. Tais sinais servem para gerar os sinais de *carry* intermediários antecipadamente quando estamos trabalhando com estruturas com ‘n’ maior que 13 bits.

As equação implementadas no bloco ‘B2’ serão as seguintes: $C_8 = GG_{5,8} + C_4$, $C_{12} = GG_{9,12} + PP_{9,12} \cdot C_8$ e $C_{16} = GG_{13,16} + PP_{13,16} \cdot C_{12}$. Esta equação é responsável pela geração dos sinais de *carry* intermediários.

O último bloco, bloco ‘B3’, é responsável pela geração dos resultados da soma dos dois vetores de entrada de dezesseis bits. Para que isso seja possível, ele implementa as equações para C_n, C_{n-1}, C_{n-2} e C_{n-3} de acordo com a fórmula geral apresentada no início deste capítulo ao mesmo tempo que executa a equação $S_i = P_i \oplus C_i$ para que possa determinar os valores da soma. Este processo é aplicado para cada grupo de quatro bits que compõe o somador.

5.1.5 CLA 32 bits

Na Figura 5.4 segue o diagrama de blocos que constitui o somador *carry look-ahead* de 32 bits. Como podemos notar ele é constituído de nove diferentes tipos de blocos que realizam funções distintas como demonstrado a seguir.

O bloco ‘PGN’ é idêntico ao bloco com o mesmo nome mostrado no CLA de 16 bits. O mesmo acontecendo com os blocos ‘T1’, ‘T2’, ‘T5’ e ‘T7’ que são iguais aos blocos ‘B0’, ‘B1’, ‘B2’ e ‘B3’ do CLA de 16 bits, respectivamente.

Já no bloco ‘T3’, a função implementada é a mesma do bloco ‘T2’. A diferença consiste no bloco ‘T3’ trabalhar sobre grupos de três bits ao passo que o bloco ‘T2’ trabalha sobre grupos de quatro bits.

No bloco ‘T6’ temos uma operação semelhante à executada nos blocos ‘T3’, com a diferença de agora reagruparmos os sinais de *propagate* e *generate* fornecidos por ‘T3’.

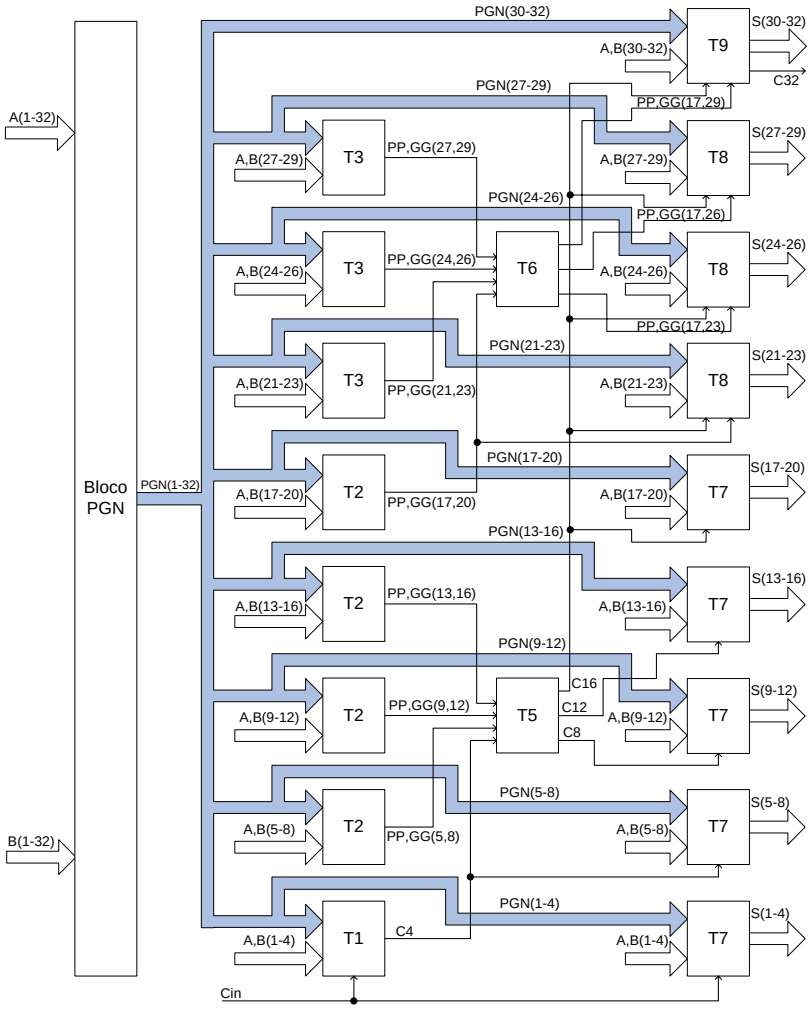


Figura 5.4 : Diagrama de blocos de um CLA 32 bits.

Esta operação é feita para que possa existir uma aceleração do processo de geração dos *carry* intermediários para índices maiores que 16. Como se pode imaginar, a mesma operação gerada para os primeiros 16 bits poderia ser repetida, mas isso faria com que tivéssemos uma dependência de C_{16} no segundo nível de blocos como mostrado em Figura 5.5a. Deste modo eliminamos esta dependência fazendo com que o sinal C_{16} só seja considerado no terceiro nível de blocos (Figura 5.5b), ou, em outras palavras, fazendo com que a profundidade lógica permaneça em quatro.

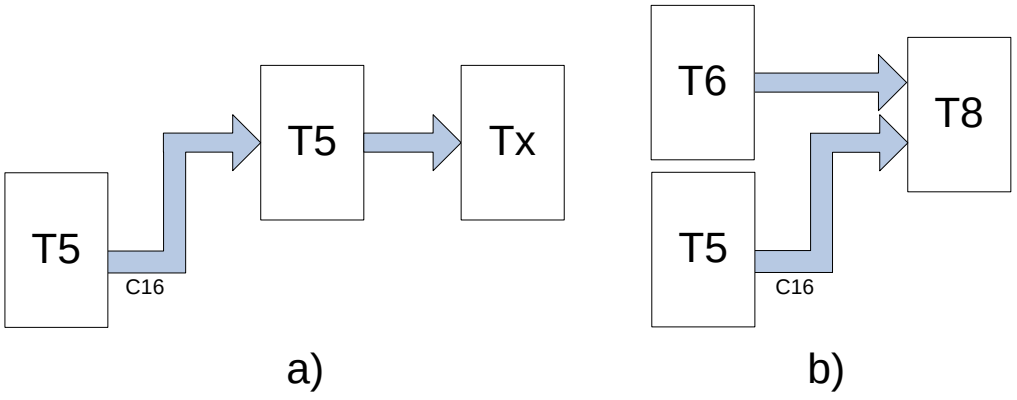


Figura 5.5 : a) profundidade lógica gerada a partir da repetição da estratégia utilizada na versão de 16bits; b) profundidade lógica obtida a partir da estrutura utilizada.

As equações geradas no bloco T6 trabalham sobre grupos de 3 bits e são:

$$PP_{i,\frac{n}{2}+1} = PP_{i,j} \cdot PP_{i-3,\frac{n}{2}-1} \text{ e } GG_{i,\frac{n}{2}-1} = GG_{i,j} + PP_{i,j} \cdot GG_{i-3,\frac{n}{2}-1}$$

Onde ‘i’ é o índice superior do grupo, ‘j’ é o inferior e ‘n’ a largura da palavra em bits do somador. Os grupos consistem nos pares de bits: i=23 j=21, i=26 j=24 e i=29 j=27.

O bloco ‘T8’ é responsável pela geração dos sinais de *carry* intermediários e pela soma propriamente dita. Nele são implementadas as equações $C_i = G_i + P_{i-1} \cdot C_{i-1}$ e $S_i = P_i \oplus C_i$ sendo que o mesmo opera sobre grupos de três bits. No bloco ‘T9’ temos praticamente a mesma estrutura da contida em ‘T8’, com uma única diferença que reside no modo como é tratado o último *carry* da cadeia.

Na Tabela 5.1 são apresentadas as equações implementadas em cada bloco do somador CLA de 32 bits:

Tabela 5.1 : Equações formadoras do CLA 32 bits.

Bloco	Funções lógicas Implementadas	
T1	$C_4 = G_4 + P_4 (G_3 + P_3 (G_2 + P_2 (G_1 + P_1C_0)))$	
T2	$G_{i,i+3} = G_{i+3} + P_{i+3} (G_{i+2} + P_{i+2} (G_{i+1} + P_{i+1}C_i))$ $P_{i,i+3} = P_i .P_{i+1} .Pi_{,i+1} .P_{i+2} .P_{i,i+2} .P_{i+3}$	
T3	$G_{i,i+2} = G_{i+2} + P_{i+2} (G_{i+1} + P_{i+1}C_i)$ $P_{i,i+2} = P_i .P_{i+1} .Pi_{,i+1} .P_{i+2}$	
T5	$C_8 = G_{5,8} + C_4$ $C_{12} = G_{9,12} + P_{9,12}C_8$ $C_{16} = G_{13,16} + P_{13,16}C_{12}$	
T6	$G_{17,23} = G_{21,23} + P_{21,23}G_{17,20}$ $G_{17,26} = G_{24,26} + P_{24,26}G_{17,23}$ $G_{17,29} = G_{27,29} + P_{27,29}G_{17,26}$	$P_{17,23} = P_{21,23} .P_{17,20}$ $P_{17,26} = P_{24,26} .P_{17,23}$ $P_{17,29} = P_{27,29} .P_{17,26}$
T7	$C_{i+1} = G_{i+1} + P_{i+1}C_i$ $C_{i+2} = G_{i+2} + P_{i+2}C_{i+1}$	$C_{i+3} = G_{i+3} + P_{i+3}C_{i+2}$ $S_i = P_i \oplus C_{i-1}$
T8	$C_i = G_{i-3,1} + P_{i-3,1}C_{i-4}$ $C_{i+1} = G_{i+1} + P_{i+1}C_i$	$C_{i+2} = G_{i+2} + P_{i+2}C_{i+1}$ $S_i = P_i \oplus C_{i-1}$
T9	$C_{29} = G_{17,29} + P_{17,29}C_{16}$ $C_{30} = G_{30} + P_{30}C_{29}$ $C_{31} = G_{31} + P_{31}C_{30}$	$C_{32} = G_{32} + P_{32}C_{31}$ $S_i = P_i \oplus C_{i-1}$

5.2 Circuitos com múltiplas saídas (MO)

Circuitos com múltiplas saídas (MO) (WANG, 97), (WANG, 89) são estruturas lógicas CMOS que se aproveitam das características recursivas e recorrentes das funções nelas projetadas para gerarem portas que desempenham mais de uma função lógica. As funções adicionais geradas na verdade são obtidas através dos nodos intermediários das redes geradas (SAMPAIO, 2004).

Devido justamente a esta característica de "compactação de tarefas", ou seja, realização em somente um elemento da função que normalmente seria gerada a partir de diversas portas lógicas, que surge a principal vantagem desta técnica, mais especificamente diminuição da área.

Através da mesma pode-se esperar também um menor atraso de propagação de sinais e baixo consumo de potência, mas ambos tem de ser avaliados mais cuidadosamente. Para que estes fatores não sejam prejudicados é necessário que se tomem cuidados de forma a não gerar caminhos com muitos transistores em série, desta forma evitando o aumento demasiado da capacitância de carga dos nodos.

As portas estudadas neste capítulo são todas constituintes de famílias CMOS derivadas de portas lógicas dinâmicas diferenciais, ou seja, as funções são geradas através de redes de transistores do tipo NMOS. Como exemplo da construção de uma porta com múltiplas saídas temos a Figura 5.6. A função escolhida para ser demonstrada

nesta figura é o cálculo dos *carry* intermediários de uma estrutura de somador do tipo *carry look-ahead*.

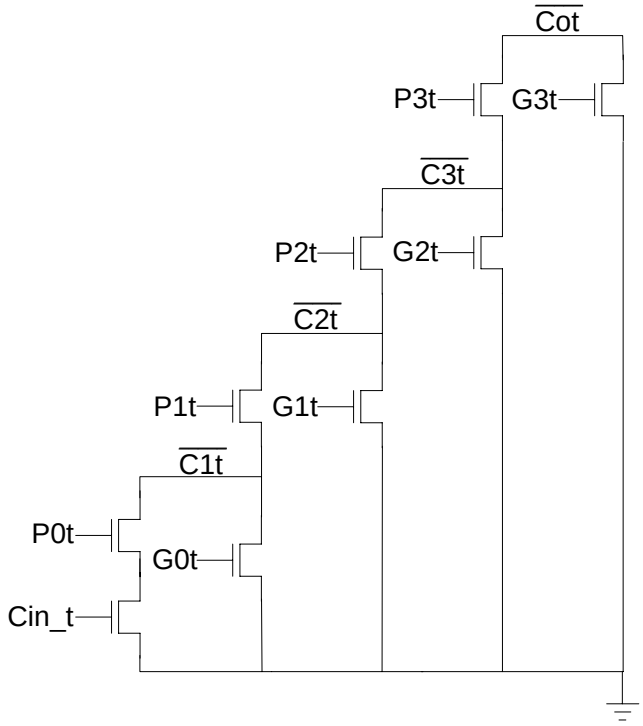


Figura 5.6 : Estrutura MO gerada a partir das equações de cálculo de *carry* intermediários para um CLA 4 bits.

A função demonstrada na Figura 5.6 se torna um bom exemplo de estruturas MO, pois apresenta um grande grau de recursividade. Este grau pode ser observado, pois cada elemento $Cout_{n+1}$ é calculado levando-se em consideração o cálculo anterior de C_n, por exemplo:

$$C_0=G_0+P_0C_{in} \text{ e } C_1=G_1+P_1C_0$$

Logo:

$$C_1=G_1+P_1(G_0+P_0C_{in})$$

Transformando-se as duas primeiras equações em portas lógicas estandares, teríamos um primeiro bloco correspondente a C_0 (Figura 5.7a) e outro a C_1 (Figura 5.7b) que poderiam ser unidos para formar um terceiro bloco (Figura 5.7c) que expressa ao mesmo tempo o termo relativo à C_1 e de onde podemos extrair o valor de C_0 através do nó interno 'X' indicado na figura. O mesmo pode-se aplicar para C_2 e C_3 sendo que ao final teremos uma única estrutura que concorrentemente expressa C_0, C_1, C_2 e C_3.

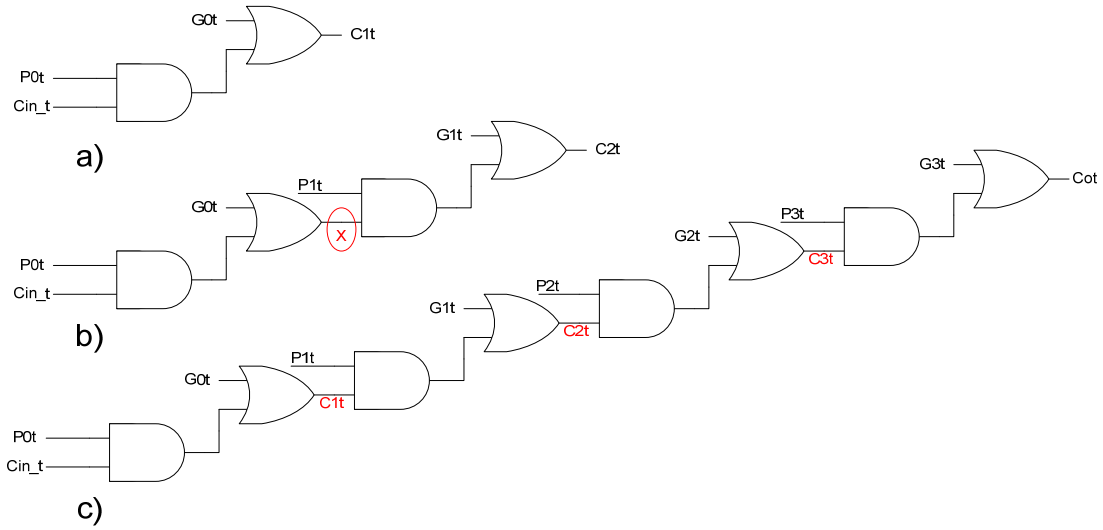


Figura 5.7 : Circuito obtido a partir das equações de cálculo dos *carry* intermediários para um CLA de 4 bits utilizando-se portas lógicas standares.

Este exemplo demonstrado na Figura 5.7 é uma forma "didática" de apresentar, através de uma forma mais usual que consiste no uso de portas lógicas, a forma de construir, por exemplo, a árvore de transistores NMOS. Se o leitor prestar atenção, através da simples transformação de todas as portas AND em dois transistores NMOS em série controlados pelas entradas da mesma, a função desta porta propriamente dita será desempenhada. O mesmo acontece com as portas OR, com a diferença de gerarem arranjos em paralelo ao invés de arranjos em série. Fazendo estas simples modificações fica claro que o circuito resultante das mesmas é o apresentado em Figura 5.6.

Os blocos construídos utilizando-se da técnica MO foram: para 4 e 8 bits o bloco de cálculo dos *carry-out* 'Bloco c'; para 16 bits os blocos 'B0', 'B1', 'B2' e 'B3'; e para 32 bits os blocos 'T1', 'T2', 'T3', 'T5', 'T6', 'T7', 'T8' e 'T9'.

5.3 Estruturas CLA Implementadas

Chegamos finalmente ao objetivo maior deste trabalho onde serão mostradas as estruturas usadas nas implementações de cada um dos tipos de somadores escolhidos para construção com múltiplas saídas. Suas construções em nível de transistor serão exemplificadas para cada um dos blocos constituintes do somador *carry look-ahead* a qual virão a pertencer (4, 8, 16 e 32 bits). Também serão apresentados os resultados de simulação dos mesmos quando submetidos à operação em conjunto com as arquiteturas básicas assíncronas.

As famílias lógicas CMOS escolhidas para serem validadas através das estruturas básicas assíncronas quando construídas utilizando-se a técnica MO são DCVS, ECDL e DPTL. O principal motivo da adoção destas famílias para a construção de circuitos MO é a facilidade com que as mesmas se adaptam às estruturas construídas com esta técnica.

A técnica SDP não é válida por seus problemas de indicação de valor “empty”. A técnica Martin apesar de apresentar bons resultados é gerada através de um algoritmo muito complexo de onde é difícil extrair a característica MO. DIMS gera circuitos com áreas muito grandes tornando a simulação muito dispendiosa em termos de tempo de processamento e espaço em disco, além de apresentar desempenho relativamente baixo se comparada a outras estruturas. E no caso da NCL não existem regras claras de otimização dos circuitos, fazendo com que os circuitos gerados em uma primeira instância também apresentem áreas grandes e atrasos significativos.

5.3.1 CLA MODCVS

A estrutura MODCVS (RUIZ, 2000), (RUIZ, 96) é construída a partir da junção de uma árvore de transistores NMOS e um circuito de pré-carga. O circuito de pré-carga foi apresentado no Capítulo 4 e a estrutura da árvore NMOS, que é construída de forma a respeitar os preceitos da técnica de múltiplas saídas, é apresentada a seguir.

Como dito anteriormente, os somadores apresentados deste ponto em diante seguem a estrutura CLA devido à sua característica de grande recursividade. Esta recursividade é explorada na construção das árvores lógicas como apresentado na Figura 5.8.

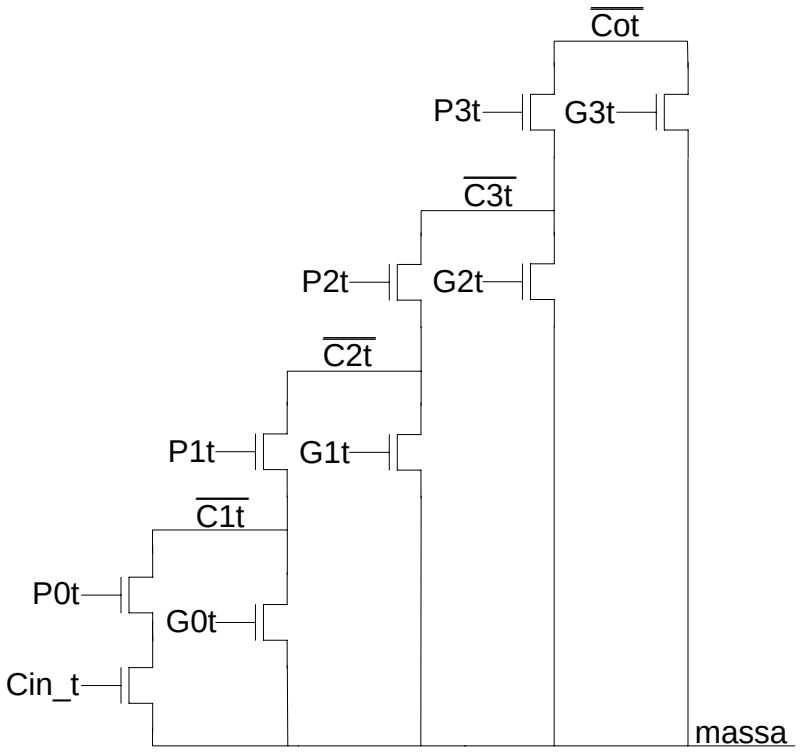


Figura 5.8 : Diagrama elétrico da árvore NMOS responsável pela implementação da função *carry-out true* de um CLA 4 bits DCVS.

Na Figura 5.8 é apresentada a estrutura elétrica correspondente à equação $C_i = G_i + P_iG_{i-1} + P_iP_{i-1}G_{i-2} + ... + P_iP_{i-1} ... P_1C_0$. Como pode ser notado, através desta única árvore podem ser extraídos os sinais de *carry* intermediários C1, C2, C3 e C4. Como

nas estruturas diferenciais sinais complementares devem ser gerados, a equação alternativa $\overline{C_i} = N_i + P_iN_{i-1} + P_i\ P_{i-1}N_{i-2} + ... + P_i\ P_{i-1}...P_iC_0$ é utilizada para gerar os valores C1, C2, C3 e C4 negados. A estrutura final gerada é apresentada na Figura 5.9, onde os nós Cxx negados são conectados aos circuitos de pré-carga.

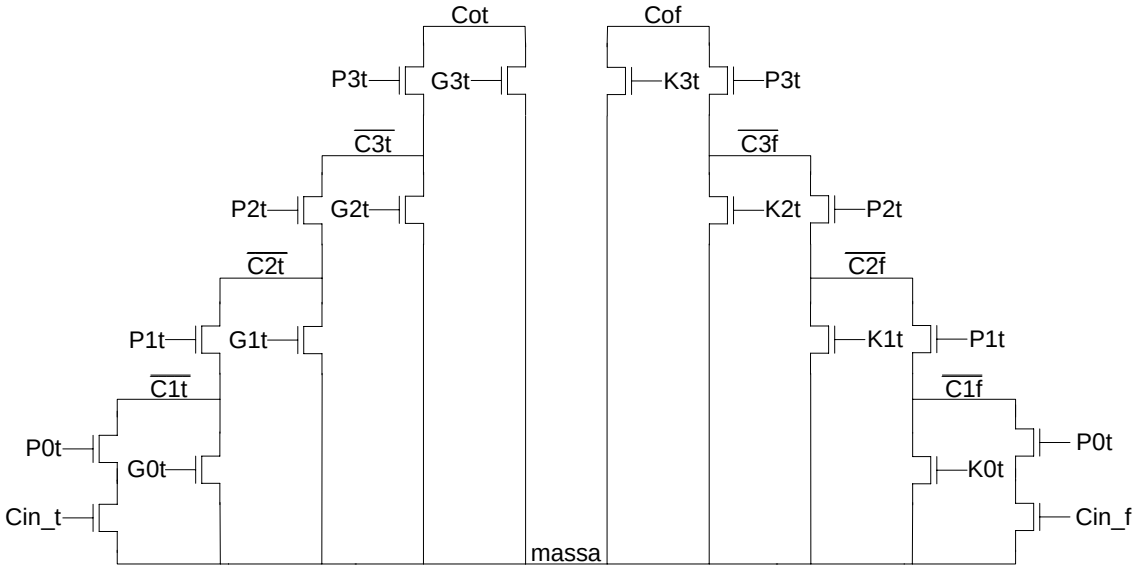


Figura 5.9 : Diagrama elétrico da árvore NMOS responsável pela implementação da função *carry-out true* e *flase* de um CLA 4 bits DCVS.

De forma semelhante são geradas as equações correspondentes às versões de 8, 16 e 32 bits que foram apresentadas anteriormente. Informações adicionais podem ser encontradas nas referências (OSÓRIO, 2004b), (RUIZ, 96), (RUIZ, 2000).

5.3.2 CLA MOECDL

Nos circuitos MOECDL (LU, 88), (OSÓRIO, 2001) a estrutura utilizada na construção da árvore lógica é igual à utilizada em MODCVS. As principais diferenças residem nesta estrutura utilizar um circuito *sense-amplifier* (apresentado no Capítulo 4) e a ausência do transistor que conecta a referência (*ground*) ao nó inferior da árvore.

As árvores que são geradas também obedecem às equações apresentadas anteriormente para as versões de 4, 8, 16 e 32 bits. Explicações mais detalhadas sobre a estrutura podem ser obtidas em (OSÓRIO, 2001).

Na Figura 5.10 é apresentada a estrutura final da árvore utilizada na computação dos *carry-outs* intermediários de um somador CLA de 4 bits. Os nós Cxx negados mostrados na figura são conectados aos circuitos de pré-descarga para que sejam gerados os sinais *carry* intermediários propriamente ditos.

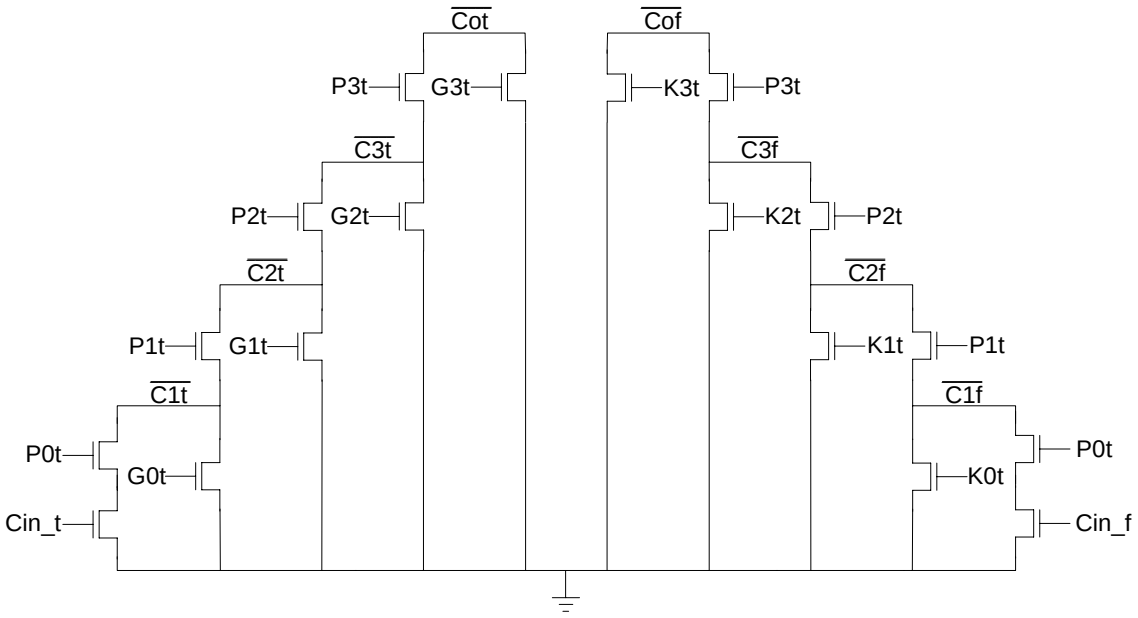


Figura 5.10 : Diagrama elétrico da árvore NMOS responsável pela implementação da função *carry-out true* e *false* de um CLA 4 bits ECDL.

5.3.3 CLA MODPTL

Diferentemente das outras famílias dinâmicas apresentadas (DCVS e ECDL), a família MODPTL (OSÓRIO, 2004b) não compartilha a mesma estrutura lógica utilizada por elas. Como apresentado no capítulo anterior, em circuitos DPTL as funções a serem implementadas são geradas por redes de transistores NMOS.

O CLA MODPTL, aqui apresentado, também obedece às equações formuladas anteriormente para geração dos blocos constituintes de sua versão em 4, 8, 16 e 32 bits. Na Figura 5.11 é apresentada a estrutura final do bloco responsável pelo cálculo dos *carry* intermediários em um CLA de 4 bits. Esta estrutura nada mais é do que a conversão da equação $C_i = G_i + P_iG_{i-1} + P_iP_{i-1}G_{i-2} + ... + P_iP_{i-1} ... P_1C_0$ em um conjunto de transistores, sendo que os demais blocos seguem a mesma metodologia de construção da rede. Os nós Cxx negados apresentados são ligados aos *sense-amplifier* de modo a gerar os sinais que serão utilizados internamente entre os blocos.

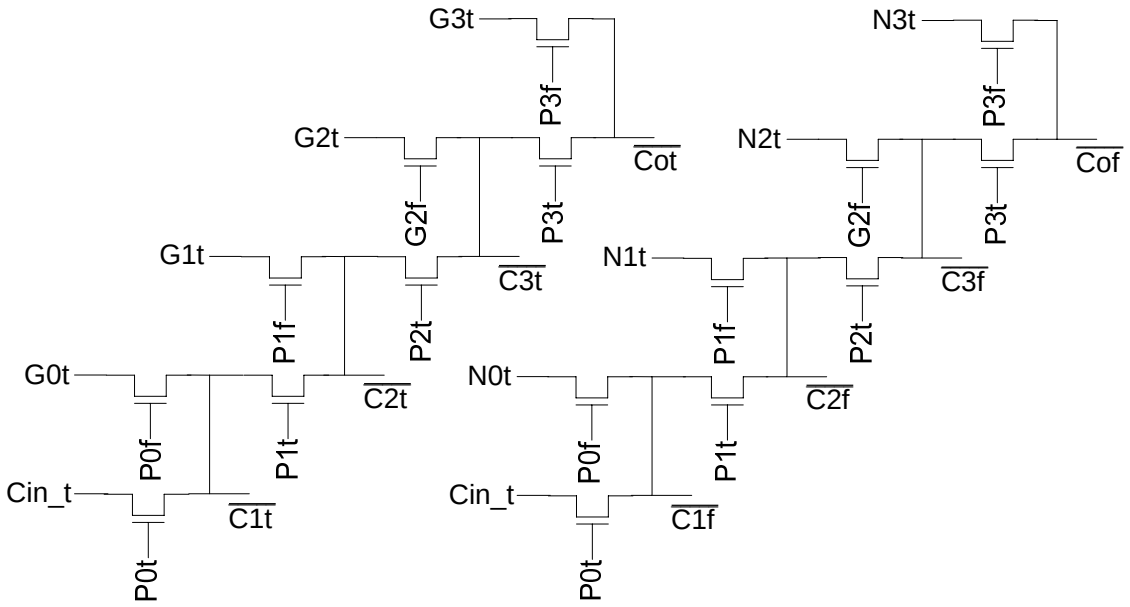


Figura 5.11 : Diagrama elétrico da rede NMOS responsável pela implementação da função *carry-out true* e *false* de um CLA 4 bits DPTL.

5.4 Resultados e Simulações

As simulações aqui apresentas levam em consideração todas as premissas estipuladas no tópico sobre simulações apresentados no capítulo anterior.

Estas simulações foram realizadas utilizando o simulador elétrico Spectre do ambiente DF II da CADENCE (CADENCE, 2003) em conjunto com o Pspice. Os processos utilizados nas simulações foram o AMI 0.6 (MOSIS, 2004) da Mosis e o AMS 0.35 (AMS, 2003) da AMS. Para o primeiro processo foram utilizados os seguintes parâmetros: tensão de alimentação equivalente a 5Volts e W e L mínimos tanto para NMOS como para PMOS iguais a 1,8 µm e 0,6 µm respectivamente. Para o segundo temos: tensão de alimentação igual a 3.3 Volts e W e L mínimos iguais a 1 µm e 0,3 µm respectivamente tanto para transistores NMOS como para PMOS.

Como nesta parte do trabalho estamos lidando com blocos CLA que são blocos funcionais que apresentam característica de indicação forte, ou seja, as saídas só vão para válido depois de todas as entradas estarem válidas, a estrutura de detecção de fim de cálculo é modificada consistindo novamente em somente uma porta OR, ou portas OR em conjunto com células C no caso da existência de mais de um barramento.

5.4.1 Resultados de simulação Divisor Inteiro (DI)

A média de resultados apresentado para o DI CLA de 4 bits foi calculado tendo por base uma simulação gerada a partir dos vetores A=15 e B=2 que totalizaram oito ciclos de cálculo.

Tabela 5.2 : Resultados obtidos para o DI CLA 4 bits.

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	171	100,00	0,41	100,00	2,26	100,00	0,93	100,00
ECDL	215	125,73	1,44	348,48	2,95	130,53	4,24	454,88
DPTL	270	157,89	2,81	681,82	2,90	128,32	8,16	874,90

De acordo com os resultados obtidos em relação à área, pode-se notar um relativo balanceamento entre as famílias DCVS e ECDL. O *overhead* apresentado no CLA ECDL é devido ao circuito de geração de próximo sinal de pré-descarga 'fi' que cada bloco tem que prover para seu sucessor, sendo este circuito uma porta NOR. Como temos três blocos que compõe o CLA teremos de ter duas portas NOR para geração de sinais internos 'fi' mais uma porta NOR para indicação de fim de processamento caso seja necessário cascatear mais de um bloco CLA de 4 bits. O *overhead* notado na família DPTL é devido também aos circuitos de geração de 'fi' em conjunto com o *sense-amplifier* que é composto por um número maior de transistores.

Quanto a atraso de propagação, pode-se notar uma grande vantagem da família DCVS em relação às demais. Este fato já era esperado, pois além de apresentar uma avaliação mais rápida da função por seus blocos lógicos, ainda não há a necessidade de um bloco gerar o sinal de controle 'fi' de seu sucessor. Como o atraso da porta NOR utilizada no cálculo do próximo sinal 'fi' é da mesma ordem de grandeza, ou até maior do que o atraso dos blocos constituintes do CLA quando trabalhamos com uma baixa quantidade de bits, temos um atraso de aproximadamente duas vezes o atraso da referência, o que pode ser confirmado pelos resultados, ECDL aproximadamente três e meio vezes e DPTL sete vezes.

Em relação ao consumo é observado um relativo equilíbrio entre as famílias consideradas sendo que, devido à grande vantagem apresentada pela família DCVS em relação ao atraso, a sua figura de mérito AxP é também a que apresenta melhor relação 'consumo X potência' com uma grande vantagem em relação às outras famílias.

Tabela 5.3 : Resultados obtidos para o DI CLA 8 bits.

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	342	100,00	1,04	100,00	5,38	100,00	5,60	100,00
ECDL	432	126,32	3,33	320,33	6,88	127,88	22,92	409,64
DPTL	542	158,48	7,77	747,25	6,80	126,39	52,85	944,48

A simulação realizada para o circuito DI CLA de 8 bits foi executada para os vetores de entrada A=245 e B=7 sendo que foram considerados 35 ciclos do anel no cálculo das médias de medidas.

Como se pode notar os resultados seguem o padrão dos apresentados para a versão de quatro bits.

5.4.2 Resultados de simulação Divisor de Resto (DR)

Os vetores utilizados na simulação do DR CLA de 4 bits foram A=3 e B=7. A média dos valores foi calculada considerando-se doze ciclos de computação no anel assíncrono.

Tabela 5.4 : Resultados obtidos para o DR CLA 4 bits.

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	171	100,00	0,92	100,00	2,66	100,00	2,44	100,00
ECDL	215	125,73	2,55	278,18	3,46	130,08	8,82	361,85
DPTL	270	157,89	2,72	296,36	3,33	125,19	9,05	371,01

Nesta aplicação, o resultado que mais chama atenção é a melhora no atraso atingido pela família DPTL e em menor escala pelo ECDL também. Como pode ser notado houve uma diminuição significativa devido à queda de aproximadamente a metade do valor apresentado na aplicação anterior. Já os outros resultados apresentam valores condizentes com os apresentados anteriormente.

Tabela 5.5 : Resultados obtidos para o DR CLA 8 bits.

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	342	100,00	2,10	100,00	4,15	100,00	8,73	100,00
ECDL	432	126,32	5,28	251,11	4,97	119,76	26,25	300,73
DPTL	542	158,48	5,98	284,25	4,00	96,39	23,91	273,98

Para a avaliação do DR CLA de 8 bits foram utilizados os vetores A=100 e B=23 e considerados 32 ciclos de cálculo no anel.

Como pode ser visto, os resultados continuam a seguir a mesma tendência, com uma ligeira diminuição do atraso DPTL. O fato mais marcante é a diminuição do consumo de potência da família DPTL fazendo com que a figura de mérito AxP da mesma seja menor que a do CLA ECDL.

5.4.3 Resultados de simulação Máximo Divisor Comum (MDC)

A média de resultados apresentado para o MDC CLA de 4 bits foi calculado tendo por base uma simulação gerada a partir dos vetores A=15 e B=12 que totalizaram cinco ciclos do anel.

Tabela 5.6 : Resultados obtidos para o MDC CLA 4 bits.

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	171	100,00	1,34	100,00	2,05	100,00	2,75	100,00
ECDL	215	125,73	1,48	110,45	2,91	141,71	4,30	156,51
DPTL	270	157,89	1,46	108,96	2,63	128,05	3,83	139,52

Neste caso, pode-se observar a repetição do padrão apresentado anteriormente, apesar de nesta aplicação ocorrer um maior equilíbrio entre as famílias lógicas para esta condição de parâmetros.

Tabela 5.7 : Resultados obtidos para o MDC CLA 8 bits.

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	342	100,00	1,44	100,00	3,37	100,00	4,86	100,00
ECDL	432	126,32	6,43	445,95	4,78	141,69	30,70	631,88
DPTL	542	158,48	10,42	722,83	3,25	96,29	33,82	696,02

A simulação realizada para o circuito MDC CLA de 8 bits foi executada para os vetores de entrada A=249 e B=12 sendo que foram considerados 24 ciclos do anel no cálculo das médias de medidas.

Os resultados apresentados para os diversos sinais medidos continuam apresentando o mesmo padrão da aplicação DI, com exceção à medida de consumo onde novamente a família DPTL apresentou o melhor desempenho.

5.4.4 Resultados de simulação Mínimo Múltiplo Comum (MMC)

Os vetores utilizados na simulação do MMC CLA de 4 bits foram A=5 e B=3. A média dos valores foi calculada considerando-se sete ciclos de computação do anel.

Tabela 5.8 : Resultados obtidos para o MMC CLA 4 bits.

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	171	100,00	2,00	100,00	1,74	100,00	3,48	100,00
ECDL	215	125,73	3,67	183,57	2,74	157,28	10,05	288,72
DPTL	270	157,89	2,87	143,57	2,39	137,55	6,87	197,48

Aqui novamente se pode observar os padrões obtidos no circuito DR, mas com uma mudança ainda mais significativa na melhora do atraso da família DPTL. Devido a esta melhora, novamente a figura de mérito AxP apresentou uma melhor relação ‘consumo X potência’ para a família DPTL em relação à ECDL.

Tabela 5.9 : Resultados obtidos para o MMC CLA 8 bits.

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	342	100,00	4,24	100,00	3,55	100,00	15,07	100,00
ECDL	432	126,32	8,12	191,31	5,64	158,87	45,80	303,94
DPTL	542	158,48	9,19	216,57	3,65	102,82	33,55	222,67

Para a avaliação do MMC CLA de 8 bits foram utilizados os vetores A=17 e B=13 e considerados 29 ciclos de cálculo no anel.

A partir dos resultados obtidos, pode-se notar um equilíbrio entre as versões de 4 e 8 bits desta aplicação. Sendo a principal diferença uma diminuição relativa no consumo da família DPTL em relação as demais.

5.4.5 Resultados de simulação Raiz

A média de resultados apresentado para o Raiz CLA de 4 bits foi calculada tendo por base uma simulação gerada a partir do vetor A=9 que totalizou cinco ciclos do anel.

Tabela 5.10 : Resultados obtidos para a Raiz CLA 4 bits.

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	171	100,00	1,18	100,00	3,05	100,00	3,60	100,00
ECDL	215	125,73	3,56	301,69	3,75	122,95	13,35	370,94
DPTL	270	157,89	3,58	303,39	3,65	119,67	13,07	363,07

Os padrões apresentados nas simulações anteriores continuam a aparecer neste caso. A referência continua com o melhor desempenho tanto em atraso como em consumo, sendo que um aumento no atraso dos somadores ECDL e DPLT é evidente.

Tabela 5.11 : Resultados obtidos para a Raiz CLA 8 bits.

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	342	100,00	1,33	100,00	2,71	100,00	3,60	100,00
ECDL	432	126,32	8,54	643,01	3,24	119,37	27,64	767,58
DPTL	542	158,48	9,90	745,16	2,49	91,88	24,65	684,67

A simulação realizada para o circuito Raiz CLA de 8 bits foi executada para o vetor de entrada A=249 e B=12 sendo que foram considerados 14 ciclos do anel no cálculo das médias dos resultados.

Devido a problemas na adaptação desta estrutura, as famílias DPTL e ECDL continuam a apresentar resultados muito pobres em relação ao desempenho.

5.4.6 Resultados de simulação Contador

Os resultados obtidos para a aplicação Contador são apresentados a partir de duas tabelas. A primeira contém os resultados de simulação utilizando-se a tecnologia AMI 0.6 e a segunda, AMS 0.35. Este formato é utilizado para todas as versões, ou seja, 4, 8, 16 e 32 bits da mesma.

Tabela 5.12 : Resultados obtidos para o Contador CLA 4 bits (AMI).

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	186	100,00	0,84	100,00	4,24	100,00	3,56	100,00
ECDL	264	141,94	4,23	503,65	5,67	133,73	23,99	673,51
DPTL	366	196,77	4,08	485,42	5,50	129,72	22,43	629,67

Tabela 5.13 : Resultados obtidos para o Contador CLA 4 bits (AMS).

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	186	100,00	0,53	100,00	1,76	100,00	0,94	100,00
ECDL	264	141,94	2,69	505,88	2,27	128,98	6,10	652,47
DPTL	366	196,77	2,60	489,41	2,23	126,70	5,80	620,11

Tabela 5.14 : Resultados obtidos para o Contador CLA 8 bits (AMI).

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	372	100,00	0,96	100,00	8,10	100,00	7,80	100,00
ECDL	522	140,32	8,12	843,90	10,40	128,40	84,47	1083,52
DPTL	726	195,16	8,02	832,86	10,00	123,46	80,16	1028,22

Tabela 5.15 : Resultados obtidos para o Contador CLA 8 bits (AMS).

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	372	100,00	0,61	100,00	3,70	100,00	2,27	100,00
ECDL	522	140,32	4,99	815,10	4,05	109,46	20,22	892,21
DPTL	726	195,16	4,98	812,24	4,00	108,11	19,90	878,10

Tabela 5.16 : Resultados obtidos para o Contador CLA 16 bits (AMI).

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	718	100,00	3,01	100,00	26,25	100,00	79,09	100,00
ECDL	1085	151,11	3,49	115,70	22,54	85,87	78,58	99,35
DPTL	1032	143,73	3,07	101,80	15,50	59,05	47,55	60,11

Tabela 5.17 : Resultados obtidos para o Contador CLA 16 bits (AMS).

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	718	100,00	2,16	100,00	10,50	100,00	22,70	100,00
ECDL	1085	151,11	2,72	125,93	8,60	81,90	23,41	103,14
DPTL	1032	143,73	2,41	111,48	5,80	55,24	13,98	61,58

Tabela 5.18 : Resultados obtidos para o Contador CLA 32 bits (AMI).

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	1486	100,00	2,80	100,00	65,39	100,00	183,40	100,00
ECDL	2287	153,90	6,19	220,60	39,85	60,94	246,56	134,44
DPTL	2135	143,67	5,49	195,80	29,60	45,27	162,55	88,63

Tabela 5.19 : Resultados obtidos para o Contador CLA 32 bits (AMS).

	Transistores		Atraso		Pot. Cons.		AxP	
Estrutura	Nº	%	ns	%	mW	%		%
DCVS	1486	100,00	2,46	100,00	23,80	100,00	58,50	100,00
ECDL	2287	153,90	4,68	190,40	16,03	67,35	75,02	128,24
DPTL	2135	143,67	4,03	164,06	10,90	45,80	43,95	75,13

O que pode principalmente ser observado através da análise dos resultados obtidos para a aplicação contador quando avaliada em suas versões de 4, 8, 16 e 32 bits é que, como dito anteriormente, as estruturas de quatro e principalmente oito bits geram circuitos somadores CLA DPTL e ECDL com desempenhos muito pobres. Como é observado, tanto o atraso como o consumo de potência de ambos são praticamente idênticos, demonstrando que além de problemas na adaptação à estrutura, provavelmente o fator dominante destes dois parâmetros é o circuito de geração de próximo ‘fi’ e não a estrutura CLA em si.

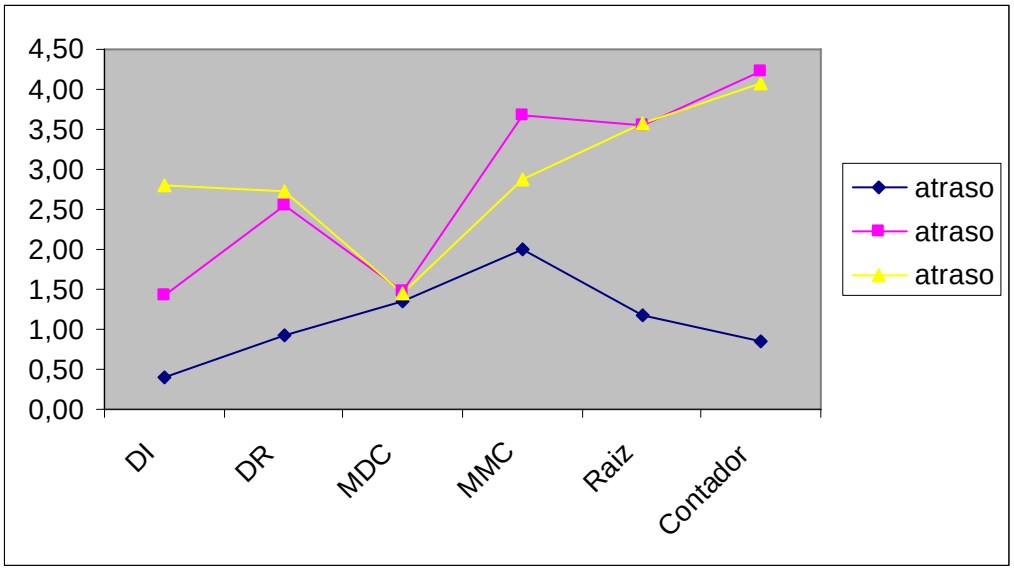
A partir da estrutura de 16 bits ocorre uma modificação na tendência observada até agora. Os parâmetros de atraso e consumo de potência tendem a se aproximar muito mais da referência DCVS. O fato que mais chama atenção a partir deste ponto é o consumo de potência da família DPTL. Este parâmetro obtém resultados tão significantes que a partir da estrutura de 16 bits é a família DPTL que apresenta as melhores relações de ‘consumo X potência’.

Este desempenho pode ser explicado principalmente pelo dimensionamento dos transistores que constituem o *sense amplifier*. Enquanto que na família DCVS é necessário um aumento considerável nas dimensões dos transistores destas estruturas para que seja atingido o ponto ótimo em relação a atraso, o aumento aplicado às famílias ECDL e DPTL para que seja atingido o mesmo é muito menor, assim gerando menor consumo de potência.

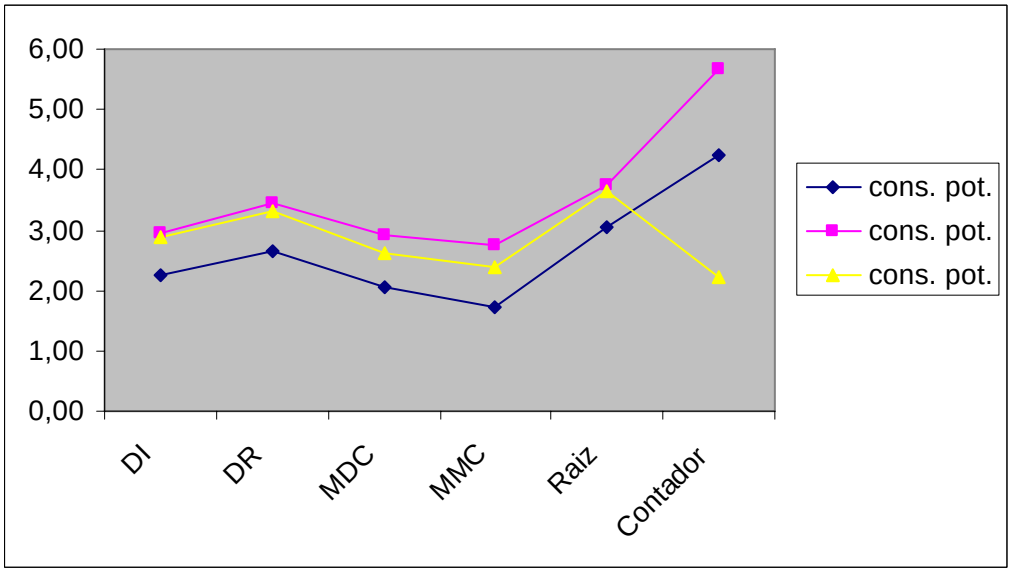
5.5 Gráficos comparativos CLA

Neste tópico será exposto o comportamento das diferentes famílias lógicas CMOS quando seus parâmetros de atraso e consumo de potência são submetidos às diferentes aplicações. Serão mostrados gráficos em que um parâmetro fixo (atraso ou consumo) são avaliados através da variação da aplicação.

Apesar do objetivo deste trabalho não consistir na avaliação em si das aplicações assíncronas, este tipo de comparação é interessante para demonstrar a dependência do atraso e consumo em relação ao padrão de vetores de teste a que ele está submetido.



a)

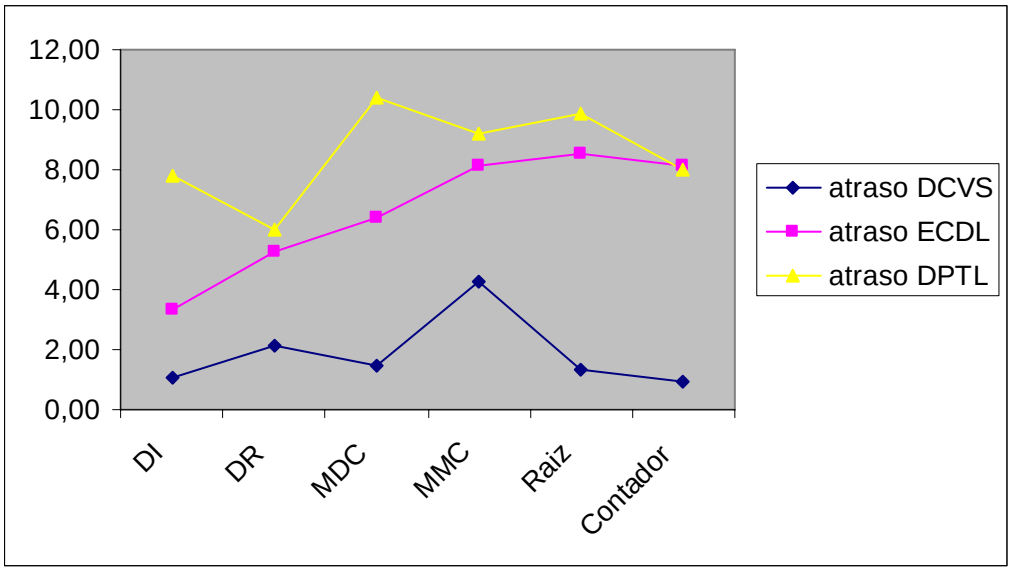


b)

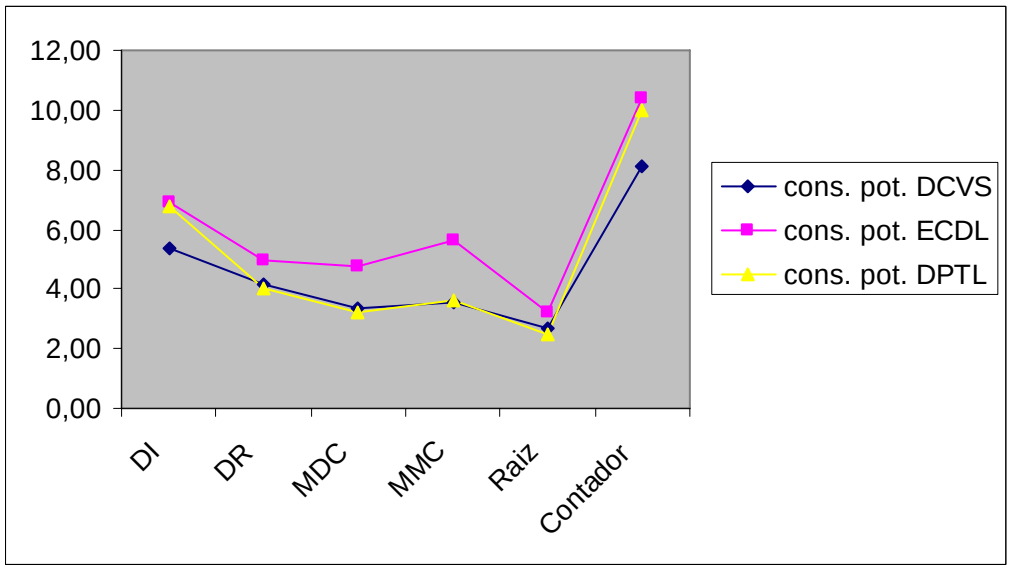
Figura 5.12 : Gráficos comparativos 'parâmetro X família' RCA 4 bits; a) atraso (ns) X família lógica; b) consumo (mW) X família lógica.

Como pode ser percebido, existe uma correlação interessante entre o modo com que o parâmetro se relaciona com o padrão de vetores de avaliação a ele aplicados.

A mesma tendência pode ser observada no gráfico apresentado a seguir, para as aplicações de 8 bits.



a)



b)

Figura 5.13 : Gráficos comparativos 'parâmetro X família' RCA 8 bits; a) atraso (ns) X família lógica; b) consumo (mW) X família lógica.

O padrão apresentado em relação a atraso difere um pouco para as versões de 4 e 8 bits. Isto pode ser decorrente do número baixo de vetores que podem ser avaliados na versão de 4 bits, sendo que na de 8, uma tendência mais concreta pode ser observada, com as demais famílias seguindo o comportamento da DCVS. Em relação a consumo a tendência em ambos se mantém. Outros gráficos contendo comparações entre famílias e parâmetros podem ser encontrados nos anexos A e B.

6 CONCLUSÃO

Neste trabalho foi apresentado um método para avaliação de desempenho de circuitos aritméticos auto-temporizados que se mostrou eficiente na obtenção dos objetivos almejados. Foram apresentados conceitos sobre a teoria assíncrona a fim de situar o leitor no contexto e introduzir os conceitos básicos necessários à compreensão do trabalho desenvolvido.

Uma metodologia foi descrita e circuitos que a desenvolvessem foram apresentados. Mais de uma aplicação utilizando-se deste método foi desenvolvida com o intuito de observar o comportamento dos somadores quando submetidos a diferentes padrões de vetores de teste.

Os circuitos aritméticos, neste caso somadores, foram apresentados para mais de uma topologia, sendo que seus diagramas de blocos e equações formadoras foram mostradas. As famílias lógicas a serem avaliadas também foram apresentadas e os circuitos a partir delas gerados foram submetidos à avaliação de diversos parâmetros, como nº de transistores, atraso de propagação e consumo de potência.

A teoria que descreve circuitos com múltiplas saídas foi exemplificada e estruturas que dela se beneficiam foram avaliadas.

O principal mérito deste trabalho é apresentar um método de avaliação de circuitos aritméticos de forma a ser possível obter características de diversas topologias e famílias quando submetidos a diferentes situações. Este tipo de avaliação é importante quando se fala de circuitos auto-temporizados uma vez que não existe muito sentido em se falar em atraso de computação ou atraso de pior caso ao trabalharmos com este tipo de circuitos, onde na verdade o que importa é a média deste valor.

A média se torna tão importante devido justamente à característica fundamental desta técnica que consiste na ausência de um sinal de relógio. Este tipo de avaliação, onde uma análise topológica ou o uso de ferramentas para que seja obtido o caminho crítico do circuito não tem um porquê de existir, há uma lacuna a ser preenchida na forma de avaliar estes circuitos de forma eficiente e de onde se possam tirar algumas conclusões sobre qual estrutura melhor se encaixa em cada caso, e é justamente neste ponto que entra a importância deste trabalho.

A partir deste trabalho podemos tirar algumas conclusões como as apresentadas a seguir:

- Apesar da família Martin apresentar resultados interessantes quando utilizada na construção de somadores RCA, seu uso na construção de circuitos CLA dificilmente exploraria a técnica MO, devido a necessidade de um grande número de transistores PMOS em série na geração da lógica de *reset*. Caso fosse implemetada da forma normal, através de diversas portas lógicas provavelmente seria muito pouco otimizada.

- A topologia utilizada na construção dos circuitos CLA 8 bits ECDL e DPTL penaliza muito estas estruturas de forma a não ser uma boa opção quando partimos para a construção utilizando estas famílias.

- Que apesar da facilidade de construção e da regularidade dos circuitos formados a partir da técnica DIMS, os custos desta família em termos de nº de transistores e atraso são demasiadamente altos.

- Para topologias CLA onde são considerados um maior número de bits (16 e 32) as famílias ECDL e DPTL passam a apresentar uma relação mais interessante de 'potência X consumo' em relação à família DCVS.

- Caso um circuito requeira que seus blocos internos, devido à função lógica neles implementados, tenha um atraso que seja muito maior que o circuito de geração de próximo 'fi', provavelmente nesta topologia valha a pena fazer um estudo mais aprofundado das lógicas ECDL e DPTL, pois as mesmas apresentarão resultados da ordem de grandeza ou superiores à família DCVS, uma vez que consomem menos potência como pode ser notado nas impementações do contador para 16 e 32 bits.

Como idéias de trabalhos futuros que surgem a partir deste trabalho temos um maior estudo da família DCVS quando aplicada a diferentes topologias de circuitos aritméticos e sistemas.

Uma verificação mais profunda da viabilidade do uso da técnica NCL na construção de sistemas assíncronos, devido aos resultados interessantes apresentados na construção de somadores RCA, a sua regularidade e a facilidade com que bibliotecas de células NCL podem ser construídas. Este estudo é válido também devido ao baixo número de regras por enquanto existente para otimização desta lógica, assim tornando interessante o seu estudo de forma a verificar tal possibilidade.

REFERÊNCIAS

AMS. 0.35 Tech Library. Disponível em: <http://www.austriamicrosystems.com>. Acesso em: 04 maio 2003.

CADENCE. CAD tools. Disponível em: <http://www.cadence.com>. Acesso em: 20 fev. 2003.

CHU, K. M.; PULFREY, D. I. Design Procedures for Differential Cascode Voltage Switch Circuits, **IEEE Journal of Solid-State Circuits**, [S.l.], v. SC-21, n. 6, p. 1082-1087, Dec. 1986.

DEAN, M. E.; DILL, D. L.; HOROWITZ, M. Self-Timed Logic Using Current-Sensing Completion Detection (CSCD), **Journal of VLSI Signal Processing**, [S.l.], v. 7, p. 7-16, 1994.

ERCEGOVAC, M.; LANG, T. **Introdução aos sistemas digitais**. [S.l.]: Bookman, 1999.

FANT, K. M.; BRANDT, S. A. NULL Convention Logic. Disponível em: <http://www.theseus.com/FramesTech.htm>. Acesso em: 15 set. 2004.

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

HELLER, L. G.; GRIFFIN, W. R. Cascode Voltage Switch Logic: A differential CMOS logic family. In: ISSCC DIG. TECH. PAPERS, 1984. **Proceedings…** [S.l.:s.n.], 1984. p. 16-17.

HWANG, I. S.; FISHER, A. L. Ultrafast Compact 32-Bit CMOS Adders in Multiple-Output Domino Logic. **IEEE Journal of Solid State Circuits**, [S.l.], v. 24, n. 2, p. 358-369, April 1989.

LU, S. Implementation of Iterative Networks with CMOS Differential Logic, **IEEE Journal of Solid State Circuits**, [S.l.], v. 23, n. 4, p. 1013-1017, Aug. 1988.

MARTIN, A. J. Asynchronous datapaths and the design of an asynchronous adder, **Formal Methods in System Design**, [S.l.], v.1, n. 1, p. 119–137, July 1992.

MOSIS. C5N Tech. Library. Disponível em: <http://www.mosis.org>. Acesso em: 18 jul. 2004.

MULLER, D. E. Asynchronous logics and application to information processing, In: SYMP. ON APPLICATION OF SWITCHING THEORY IN SPACE TECHNOLOGY, 1963. **Proceedings…** [S.l.]: Stanford University Press, 1963. p. 289–297.

MULLER, D. E.; BARTKY, W. S. A theory of asynchronous circuits. In: INTERNATIONAL SYMPOSIUM ON THE THEORY OF SWITCHING, 1959, Cambridge. **Proceedings…** [S.l.]: Harvard University Press, 1959. p. 204–243.

NG, P.; BALSARA, P. T.; STEISS, D. Performance of CMOS Differential Circuits. **IEEE Journal of Solid State Circuits**, [S.l.], v. 31, n. 6, p. 841-846, June 1996.

OSÓRIO, M. C.; REIS, A.; RIBAS, R. P. Estruturas CMOS Dinâmicas com Múltiplas Saídas, In: WORKSHOP IBERCHIP, 10., 2004, Cartagena de Índias, Colômbia. **Resumos**. Cartagena de Indias: Universidade de los Ardes, 2004b. p.103

OSÓRIO, M. C.; SAMPAIO C. A.; REIS, A.; RIBAS R. P. Enhanced 32-Bit Carry Lookahead Adder using Multiple Output Enable-Disable CMOS Differential Logic. In: *SBCCI* , 2004. **Proceedings…** [S.l.:s.n.], 2004.

PARHAMI, B. **Computer Arithmetic algorithms and hardware designs**. [S.l.]: Oxford University Press, 2000.

PASTERNAK, J. H.; SALAMA, C. A. T. Design of Submicrometer CMOS Differential Pass-Transistor Logic Circuits. **IEEE Journal of Solid State Circuits**, [S.l.], v. 26, n. 9, p. 1249-1258, Sept. 1991.

PASTERNAK, J. H.; SALAMA, C. A. T. Differential pass-transistor logic partial product generator for iterative multipliers. In: ECCTD, 1989. **Proceedings…** [S.l.:s.n.], 1989. p. 176-179.

PASTERNAK, J. H.; SALAMA, C. A. T. Optimization of submicron CMOS differential pass-transistor logic design. In: ESSCIRC, 1989. **Proceedings…** [S.l.:s.n.], 1989. p. 218-221.

PASTERNAK, J. H.; SHUBAT, A. S.; SALAMA, C. A. T. CMOS differential pass-transistor logic design. **IEEE Journal of Solid State Circuits**, [S.l.], v. 22, p. 216-222, 1987.

RADHAKRISHNAN, D.; WHITAKER, S. R.; MAKI, G. K. Formal Design Procedures for Pass Transistor Switching Circuits, **IEEE Journal of Solid State Circuits**, [S.l.], v. 20, n. 2, p. 531-536, April 1985.

RUIZ, G. A,; MANZANO, M. A. Compact 32-Bit adder in multiple-output DCVS logic for Self-timed Circuits. **IEE Proc. – Circuit Devices Syst.**, [S.l.], v. 147, n. 3, p. 183-188, June 2000.

RUIZ, G. A. Compact four bit carry look-ahead CMOS adder in multi-output DCVS Logic. **Electronic Letters**, [S.l.], v. 32, n. 17, p. 1556-1557, Aug. 1996.

RUIZ, G. A. Evaluation of Three 32-Bit CMOS Adders in DCVS Logic for Self-Timed Circuits. **IEEE Journal of Solid State Circuits**, [S.l.], v. 33, n. 4, p.604-613, April 1998.

SAMPAIO, C. A.; SILVA, R. T. V.; REIS, A.; RIBAS, R. P. Construção de Parallel-Prefix Adders com Portas MOCMOS. In: WORKSHOP IBERCHIP, 10., 2004, Cartagena de Índias, Colômbia. **Resumos**. Cartagena de Indias: Universidade de los Ardes, 2004. p.102

SOBELMAN, G. E.; FANT, K. **CMOS circuit design of threshold gates with hysteresis**. Disponível em: <http://www.theseus.com/FramesTech.htm>. Acesso em: 15 set. 2004.

SPARSØ, J.; STAUNSTRUP, J. Delay-insensitive multi-ring structures. **Integration, VLSI Journal**, [S.l.], v. 15, n.3, p. 313–340, Oct. 1993.

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

SPARSØ, J.; STAUNSTRUP, J.; DANTZER-SØRENSEN, M. Design of delay insensitive circuits using multi-ring structures, In: EUROPEAN DESIGN AUTOMATION CONFERENCE, EURO-DAC, 1992, Hamburg, Germany. **Proceedings…** Los Alamitos: IEEE Computer Society Press, 1992. p. 15-20.

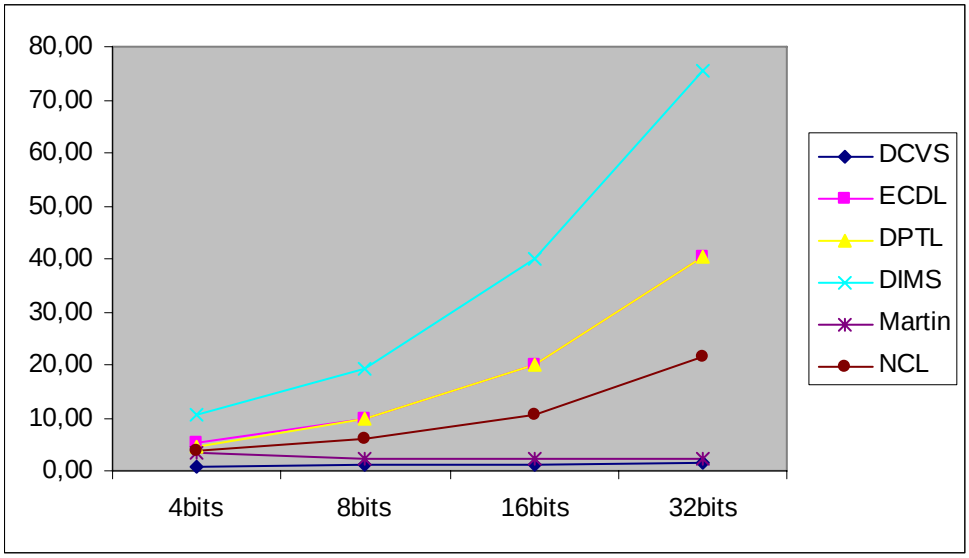
SUTHERLAND, I. E. Micropipelines. **Communications of the ACM**, New York, v.32, n.6, p. 720–738, June 1989.

TRAVER, C.; REESE, R. B.; THORNTON, M. A. Cell Designs for Self-Timed FPGAs. In: ASIC/SOC, 2001. **Proceedings…** [S.l.:s.n.], 2001.

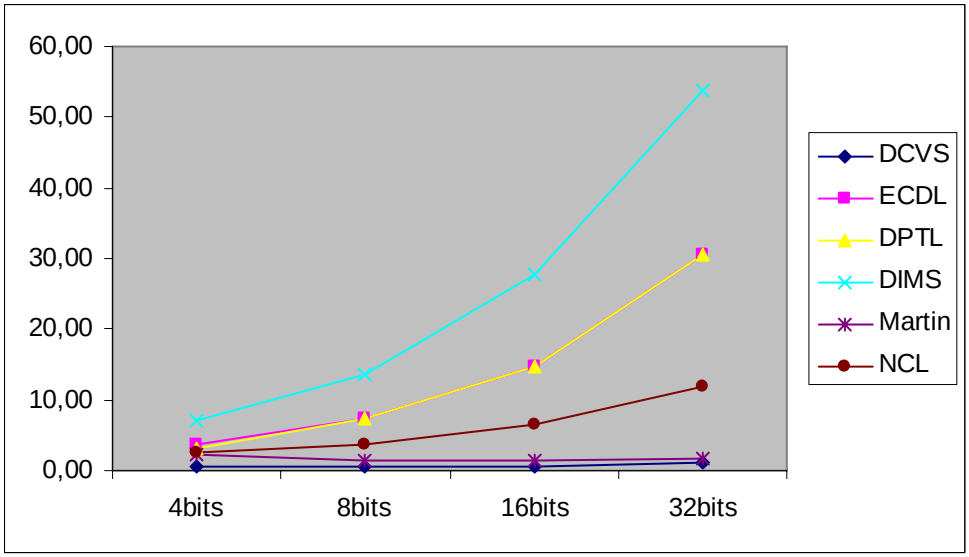
WANG, Z. et al. Fast Adders Using Enhanced Multiple-Output Domino Logic. **IEEE Journal of Solid State Circuits**, [S.l.], v. 32, n. 2, p. 206-214, Feb. 1997.

YANO, K. A 3.8ns CMOS 16 X 16 multiplier using complementary pass transistor logic. In: CICC, 1989. **Proceedings…** [S.l.:s.n.], 1989.

ANEXO A Gráficos comparativos entre RCAs

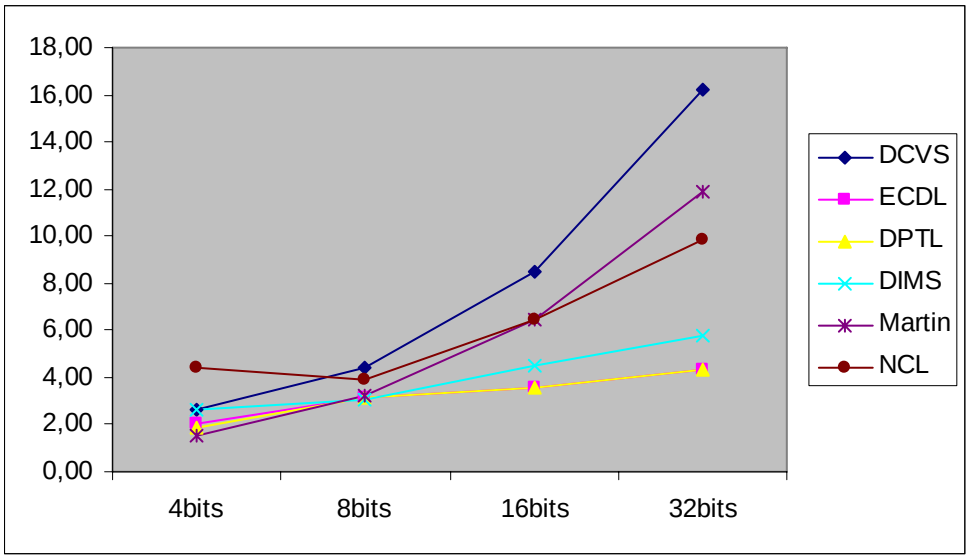


a)

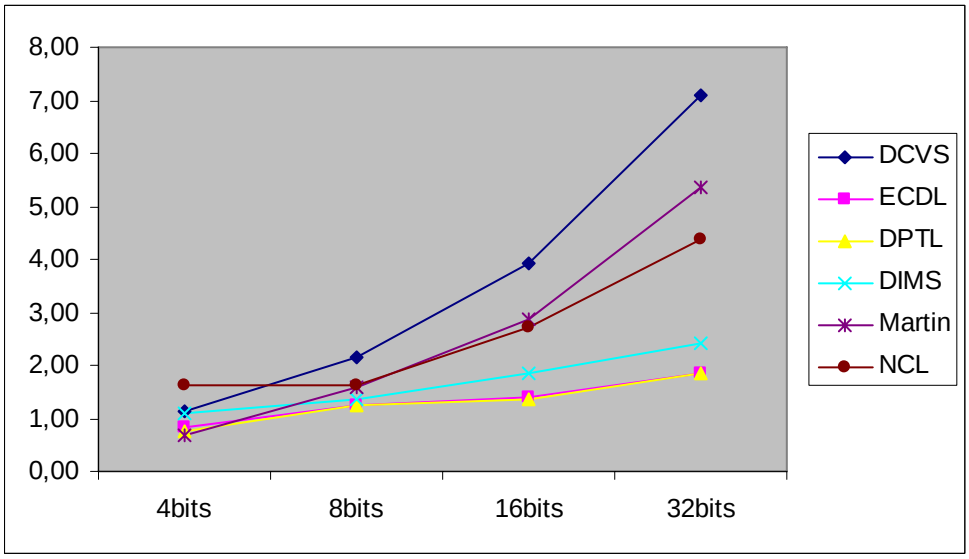


b)

Gráficos atraso X largura de palavra de somadores RCA; a) para parâmetros de processo AMI 0.6; b) processo AMS 0.35.

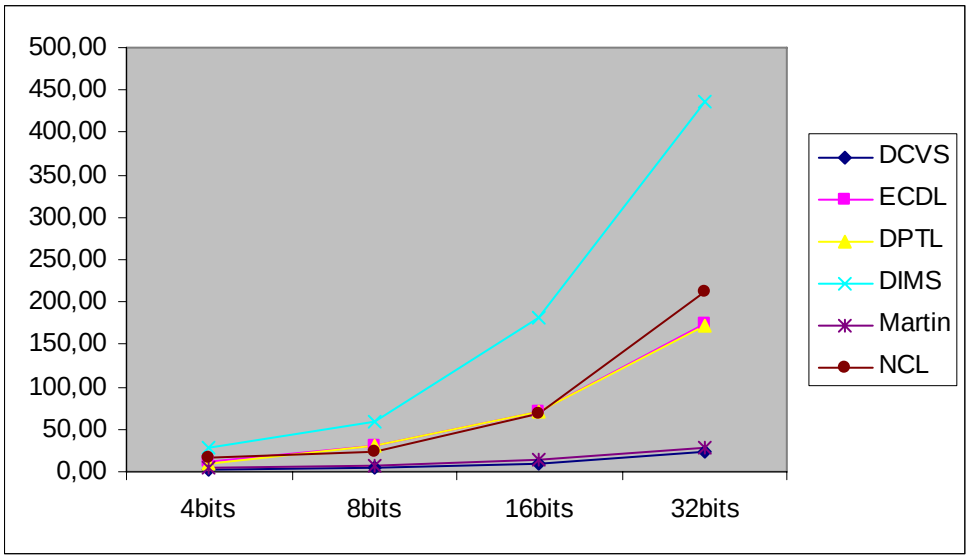


a)

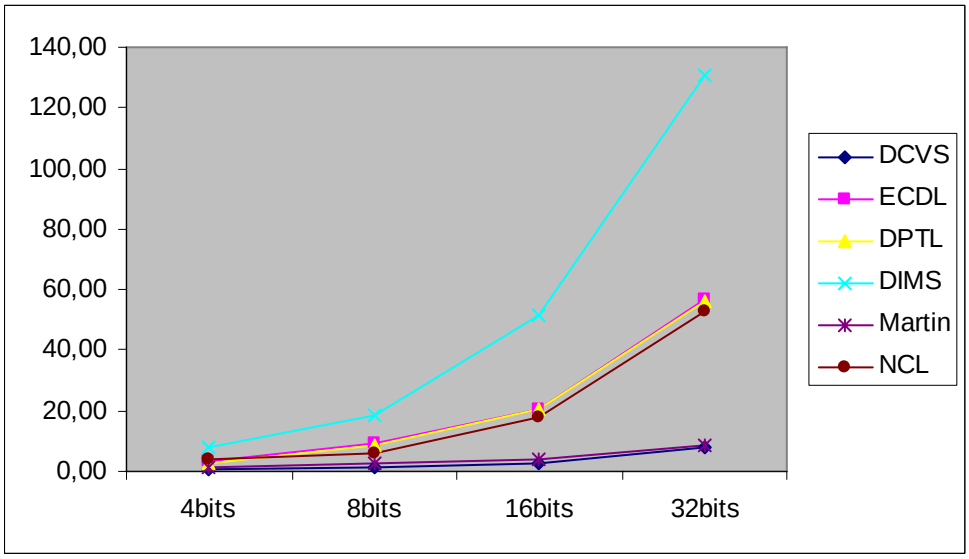


b)

Gráficos consumo X largura de palavra de somadores RCA; a) para parâmetros de processo AMI 0.6; b) processo AMS 0.35.

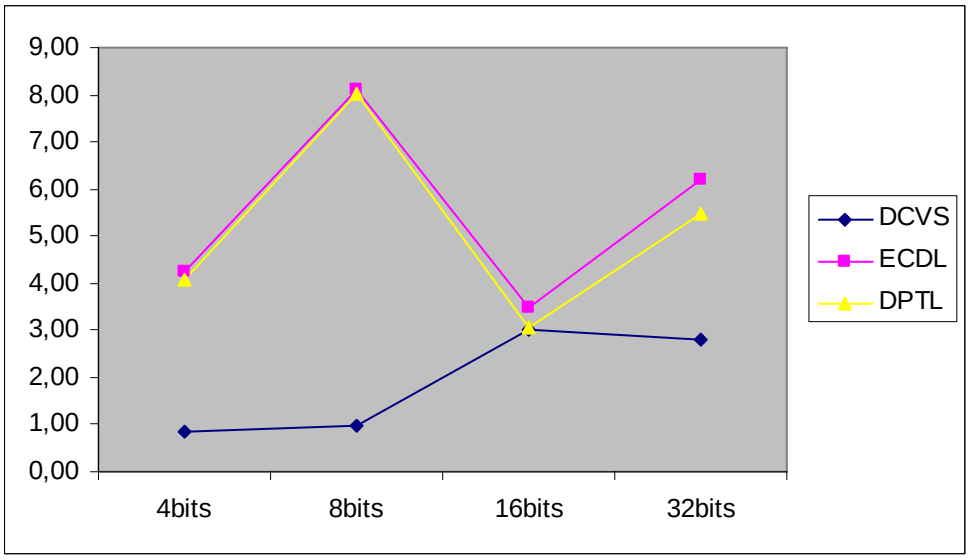


a)

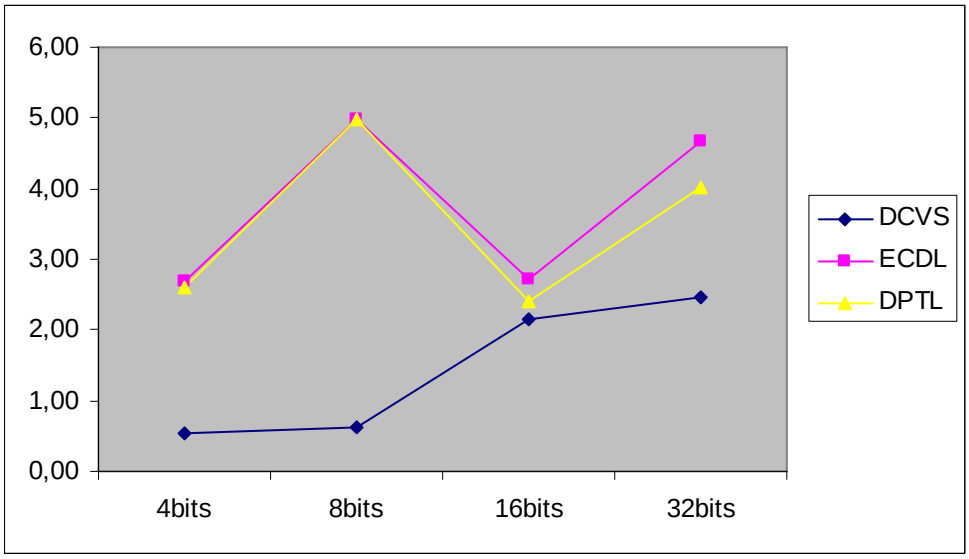


b)

Gráficos AxP X largura de palavra de somadores RCA; a) para parâmetros de processo AMI 0.6; b) processo AMS 0.35.

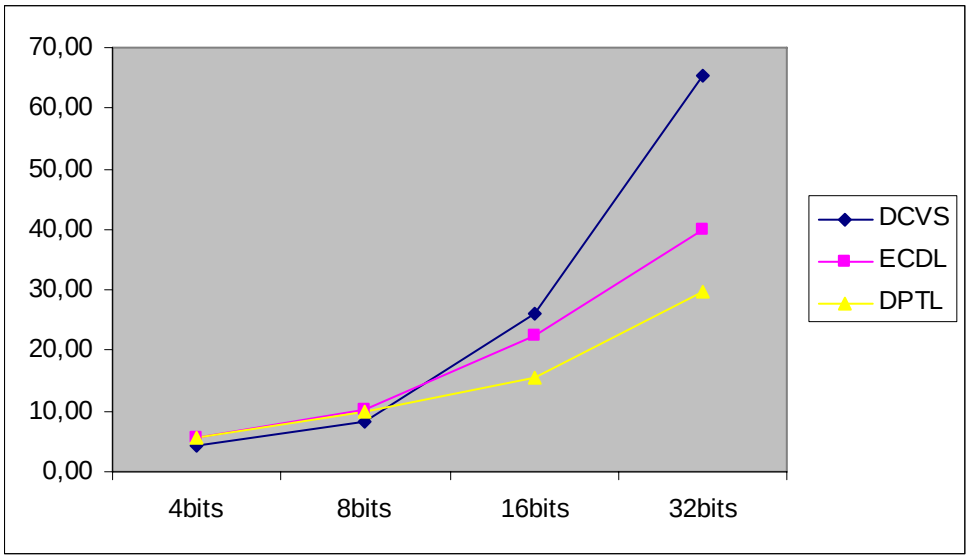


a)

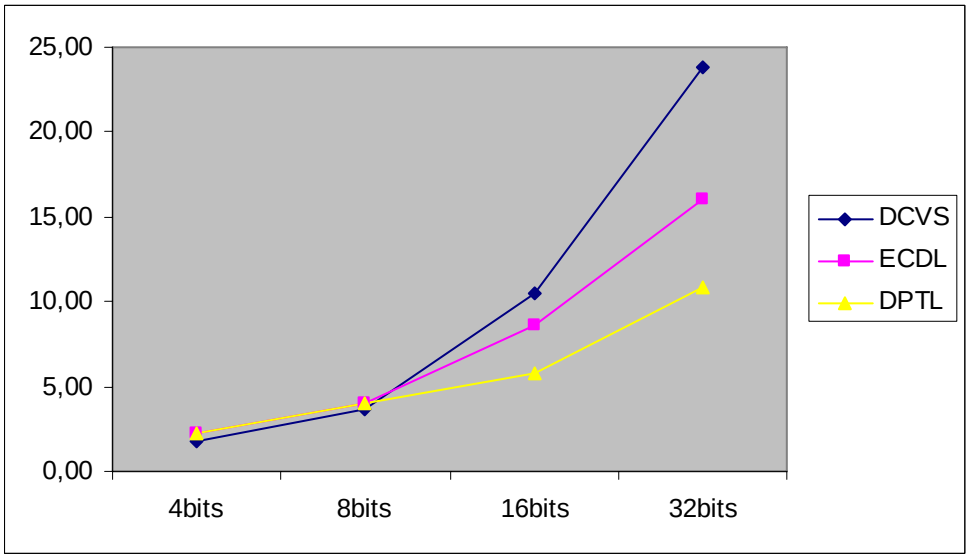


b)

Gráficos atraso X largura de palavra de somadores CLA; a) para parâmetros de processo AMI 0.6; b) processo AMS 0.35.

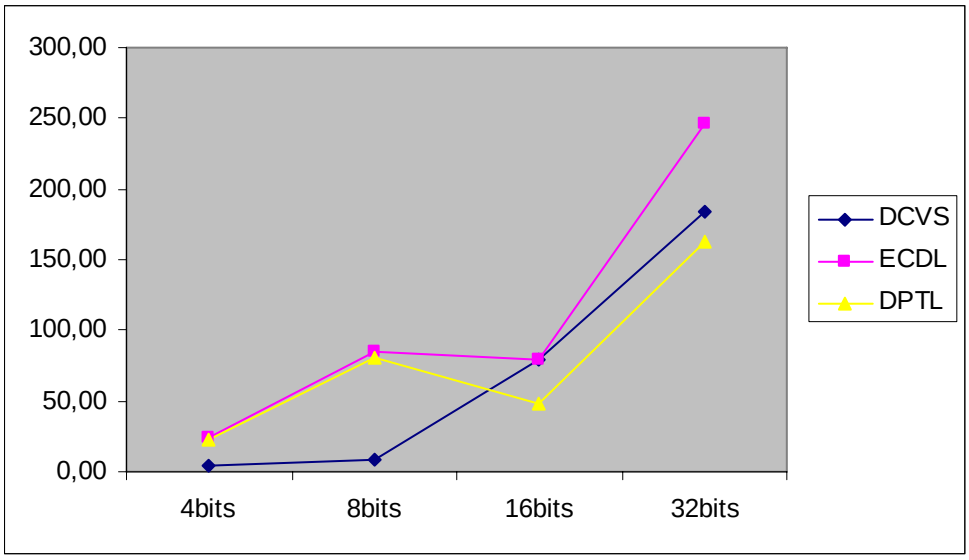


a)

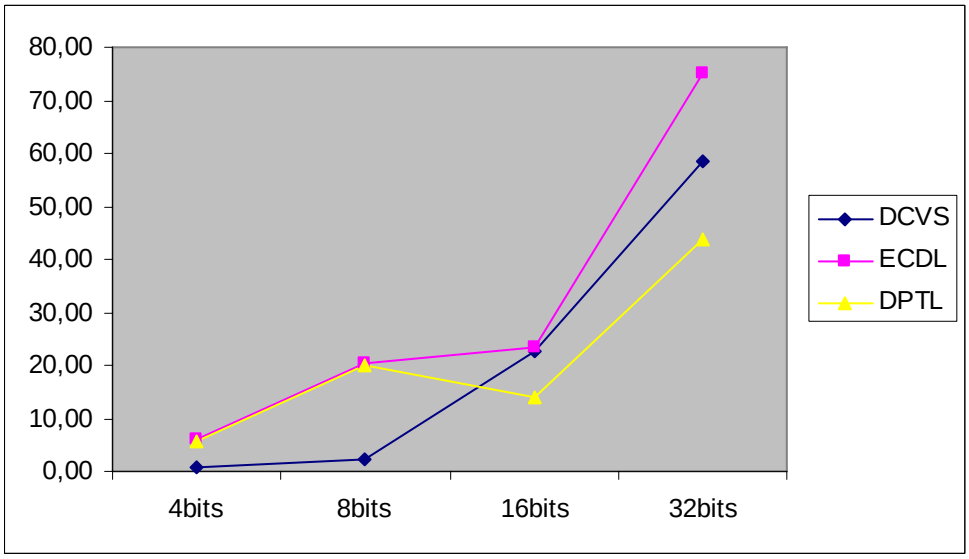


b)

Gráficos consumo X largura de palavra de somadores CLA; a) para parâmetros de processo AMI 0.6; b) processo AMS 0.35.



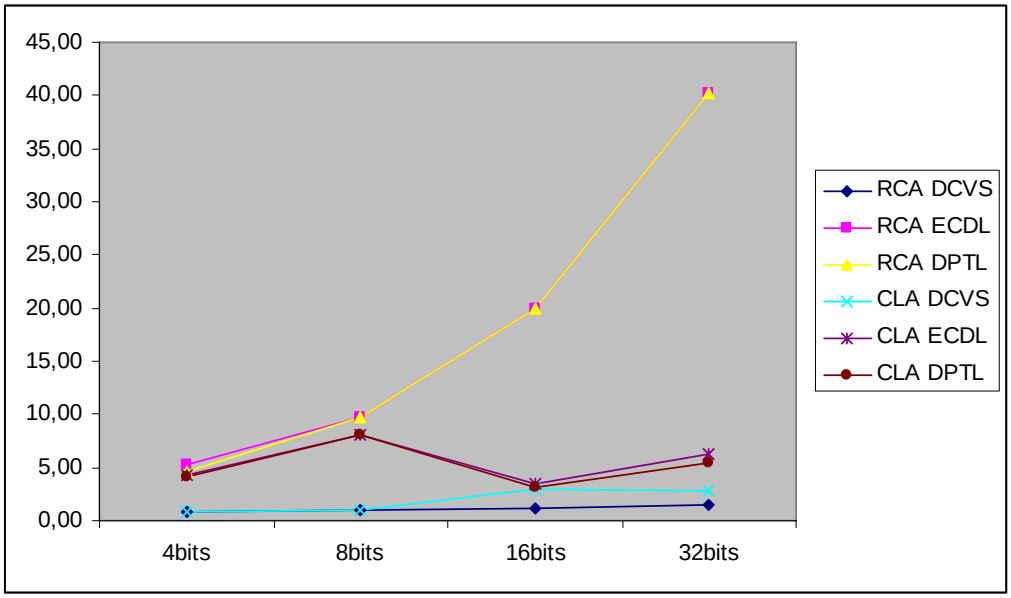
a)



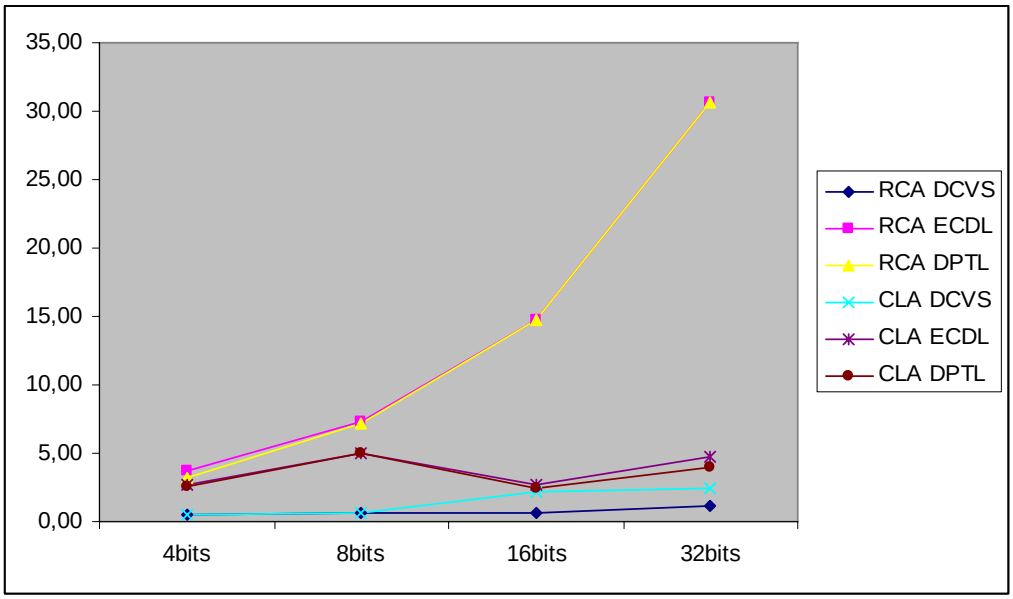
b)

Gráficos AxP X largura de palavra de somadores CLA; a) para parâmetros de processo AMI 0.6; b) processo AMS 0.35.

ANEXO B Gráficos comparativos entre CLAs

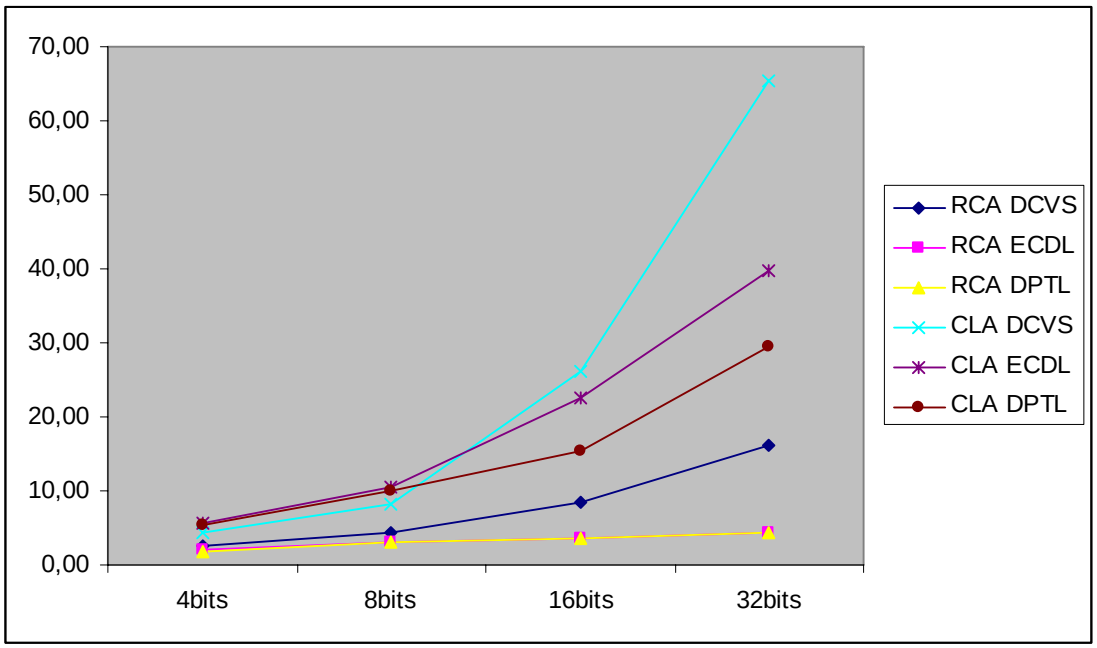


a)

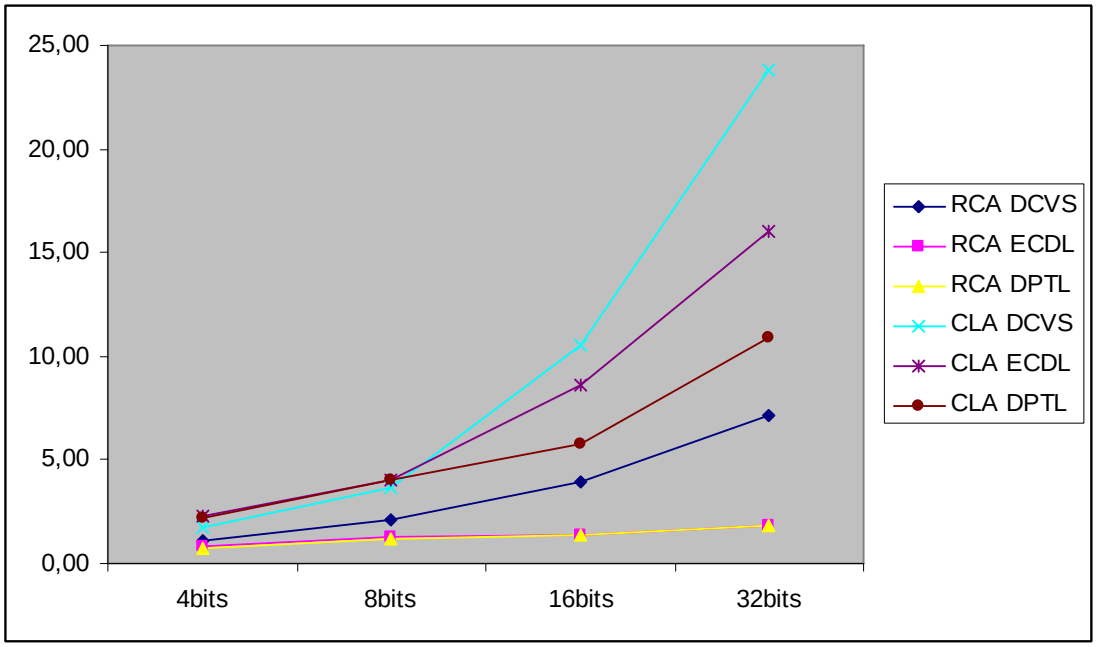


b)

Gráficos atraso X largura de palavra quando se compara as famílias DCVS, ECDL e DPTL em suas implementações RCA e CLA; a) para parâmetros de processo AMI 0.6; b) processo AMS 0.35.

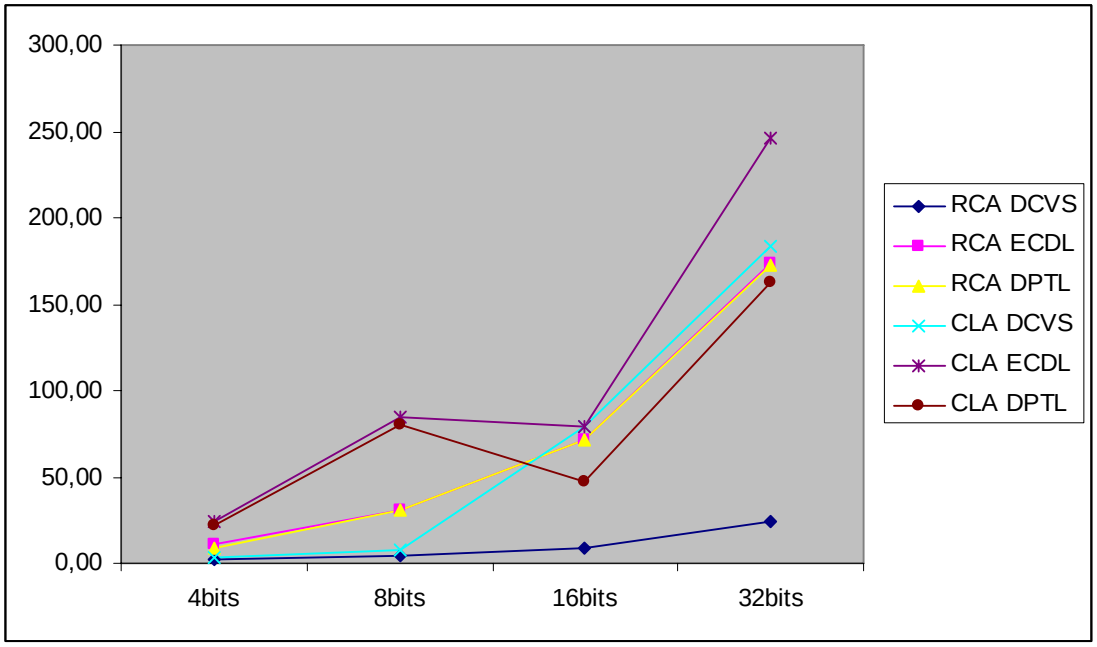


a)

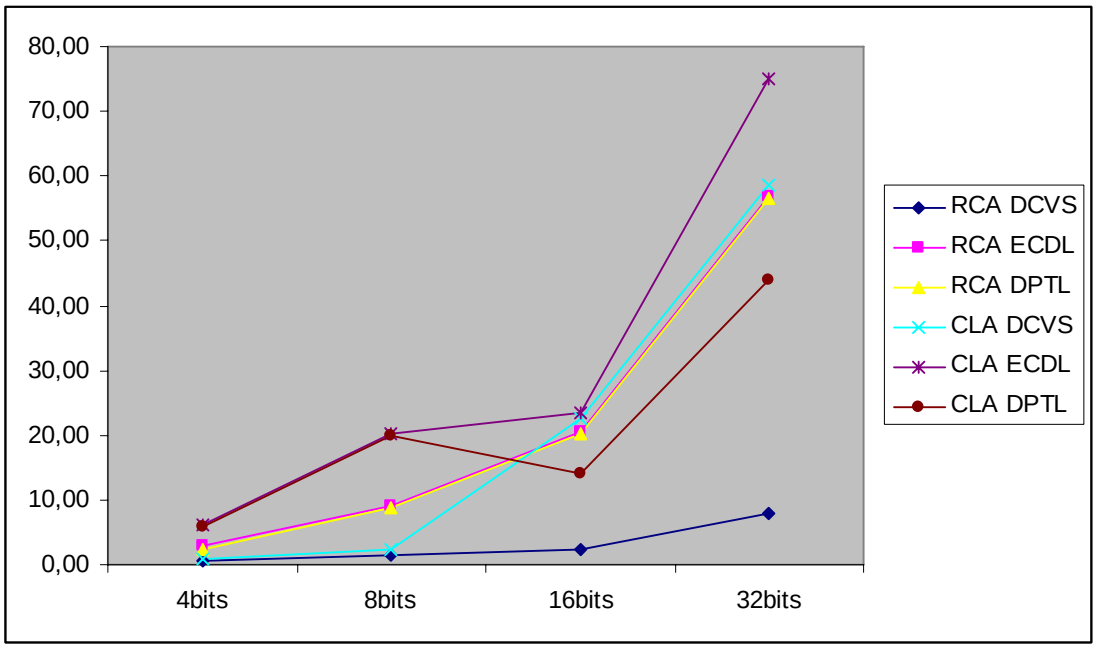


b)

Gráficos consumo X largura de palavra quando se compara as famílias DCVS, ECDL e DPTL em suas implementações RCA e CLA; a) para parâmetros de processo AMI 0.6; b) processo AMS 0.35.



a)



b)

Gráficos AxP X largura de palavra quando se compara as famílias DCVS, ECDL e DPTL em suas implementações RCA e CLA; a) para parâmetros de processo AMI 0.6; b) processo AMS 0.35.